INTERVAL AND POINT-BASED APPROACHES TO HYBRID SYSTEM VERIFICATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Arjun Kapur
August, 1997

Abstract

Hybrid systems are real-time systems consisting of both continuous and discrete components. This thesis presents deductive and diagrammatic methodologies for proving point-based and interval-based properties of hybrid systems, where the hybrid system is modeled in either a sampling semantics or a continuous semantics. Under a *sampling semantics* the behavior of the system consists of a discrete number of system snapshots, where each snapshot records the state of the system at a particular moment in time. Under a continuous semantics, the system behavior is given by a function mapping each point in time to a system state. Two continuous semantics are studied: a *continuous interval semantics*, where at any given point in time the system is in a unique state, and a *super-dense semantics*, where no such requirement is needed.

We use *Linear-time Temporal Logic* for expressing properties under either a sampling semantics or a super-dense semantics, and we introduce *Hybrid Temporal Logic* for expressing properties under a continuous interval semantics. Linear-time Temporal Logic is useful for expressing *point-based* properties, whose validity is dependent on individual states, while Hybrid Temporal Logic is useful for expressing both *interval-based* properties, whose validity is dependent on intervals of time, and point-based properties.

Finally, two different verification methodologies are presented: a *diagrammatic approach* for verifying properties specified in Linear-time Temporal Logic, and a *deductive approach* for verifying properties specified in Hybrid Temporal Logic.

Acknowledgements

Forget the bad,
Remember the good,
But learn from both.

There are few times in this world where we tell those around us how much they mean to us and how profoundly they have affected us. I want to take this opportunity to give thanks to those who have had faith in me even when I did not have faith in myself.

The Stanford Connection

I would like to start by thanking my advisor Zohar Manna, who for some unknown reason believed I could graduate even when I thought otherwise. He had the presence to give me space when I needed it, criticism when I deserved it, and advice when I asked for it. I would like to thank John Mitchell for acting as my guardian angel through the PhD program. Unaffected by the seemingly tumultuous world, he has the ability to offer simple solutions in a complex web of entanglements. Considerable thanks also go to Amir Pnueli. With Amir, I have learned that virtually every remark he makes must be carefully weighed and processed. For even in his casual remarks, there is often great wisdom. Amir provided much needed confidence and a safe shelter far from the madding crowd. For that, I will always treasure my stay in Israel.

Apart from my advisors and mentors, there are several people without whom this thesis would not have been written. First and foremost is Luca de Alfaro with whom much of this research has been developed. His enthusiasm in the pure beauty of mathematics has been a welcome solace these past years. I have had the good fortune to be his officemate and watch him mature into a great researcher. I would also like to thank my fellow colleagues in the STeP group for their honest and often rambunctious discussions. In particular, I would like to thank Henny Sipma for our discussions on hybrid systems, research in general, etc., and Michael Colón for our discussions on Java, practical computer science, etc.

My officemates have always been more than colleagues; they have been treasured confidantes. Before Luca, I was fortunate to share an office with Anuchit Anuchitanukul during his last year at

Stanford. It is amazing the insight a departing graduate has about matters of life and philosophy, but I guess that is why they call the degree a doctor of philosophy, for it is the philosophy underlying all research that doctoral candidates learn. Theses themselves are mere details.

And of course, before Anuchit there was Kathleen Fisher. I was at my best with Kathleen: the sparkle of research, the sparkle of life, the magic of knowing another even better than knowing oneself, the best poetry I have written, the best I have ever given, and the best times I have ever shared. I was there when she needed me, and she has been there whenever I needed her. Of all the people I have encountered while at Stanford, I doubt anyone has had a more profound and fundamental influence on my life. She gave me the experiences of the Mayor, Jude, and Far From the Madding Crowd, and in the process revealed how the Mayor, Jude, and Gabriel were all just different projections of the same soul.

The Cornell Connection

Our hearts require friendship and love as surely as our bodies require food. Cherish those connections, especially to those closest to you today.
Frank H. T. Rhodes, president emeritus, Cornell University

I was very fortunate at Cornell. While Kathleen has most affected me at Stanford, Zachary Carter has most influenced me at Cornell. Zack laid the foundations to so much of what I am today. (That's right, blame him!) There is a great story of friends in the Mahabharata. It is the story of Sudaama and Krsna. While I may not be a Krsna, he will always be my Sudaama. He has kept me laughing, even during my worst days, with his wit, candor, and personal experiences. He has seen me through hell and back more times than he probably cares to remember, and each time provided the perfect advice to help me recover. His is a rare and special friendship, and I hope to God I never lose it.

Cornell also blessed me with some of the greatest teachers I have ever had. Teachers, who you follow from class to class just because they are teaching them. Sam Toueg taught me how to teach through his own teaching style and through his philosophy on what teaching is really all about. I have always tried to live up to his standard of teaching, and one day I hope I will. Harold Hodes was my first logic professor. Even before I had applied to Cornell for college, I attended a six-week summer course at Cornell that he taught. I was hooked! I would end up taking four more logic courses with him and have preserved every page of his lecture notes. Dale Boyer has been my photography mentor for the last year. Even though my education with Dale has been in California, I still consider it part of my Cornell experience; for Dale, like me, is a Cornell graduate. Dale has opened my eyes, and taught me how to see when before I could only look. To Sam, Harold, and Dale, I say thanks for recognizing the importance that teaching has in the development of our future.

Is There Life After Cornell?

Yes, but only if you consciously build it.
Frank H. T. Rhodes, president emeritus, Cornell University

I must admit that it took me a while to build my post-Cornell life. Even then, it was probably more a result of my friends building my life than of me actually doing anything. Three people in particular have built me the life that I now enjoy: Cyndy Dy, Tony Porterfield, and Mrinal Chopra. Cyndy created the hiker, a madman with an unrelenting ambition to scale mountains. She patiently showed me the joy and beauty that America's wildernesses have to offer. Our weekend hikes in the mountains gave me the mental equanimity to face the week's challenges. Without her, I probably would have gone insane a long time ago. Tony added skiing and camping to my life. While Luca reintroduced me to skiing, it was Tony who gave me the disease. Our weekend trips have furnished me with indelible memories and returned me to my former youth. As Dylan once said:

Romantic facts of musketeers,
Foundationed, deep somehow.
Ah, but I was so much older then.
I'm younger than that now.

Mrinal added the most crucial element to the mix, photography. Photography has been our great excuse for traveling and exploring the outdoors, and Mrinal has been a faithful companion on my many mad adventures. From carrying an ice-cold tripod in the blistering Buffalo winters to photograph Niagara Falls, to jeeping the Utah slick-rock in rebellious defiance of risk-averse park rangers. My mother once said that Mrinal and I were on parallel paths that somehow crossed and became one. My mother was right. Together, Cyndy, Tony, and Mrinal have accompanied me on almost thirty trips in the past two years, and we have only just begun. I thank each of them for building my life after Cornell.

Finally, I thank my sister and my parents for their unconditional support through these difficult years. Everyone needs some unconditional support and love, and I have always had theirs. I want to thank my parents for instilling in me the quest for knowledge and pursuit of higher education. They were my first teachers, and they continue to shape my thoughts and beliefs. I also want to thank my sister, Gayatri Kapur, and my brother-in-law, Rasesh Shah, for calming the rest of my family when I went off and did crazy things. To all of them, all I can say is I intend to do more. Finally, I want to thank Ernest Stoessel, my first real mathematics teacher, who bestowed upon me a desire to explore science by showing me the reality of the abstract.

Preparation of thesis

Much of my research has been supported by a National Science Foundation Graduate Research Fellowship. In an age of dwindling governmental funds for research, their support has been much appreciated. Incarnations of various chapters have been read by my colleagues. I would like to thank Luca de Alfaro, Nikolaj Björner, Yassine Lakhneche, Hugh McGuire, Henny Sipma, and Tomás Uribe for their feedback and comments. Extra special thanks go to Henny Sipma for reading and editing innumerable drafts of this thesis and providing valuable advice on content, form, and grammar. Any remaining errors are due to my not listening to her.

Arjun Kapur
Stanford, California
August 1997

Chronology

The work on stuttering automata, presented in Chapter 4, is joint work with Zohar Manna and Amir Pnueli. The work on diagrammatic verification, presented in Chapter 6 and Chapter 7, is joint work with Luca de Alfaro and Zohar Manna. The completeness proofs, in particular, have been primarily developed by de Alfaro. The work on diagrams under the sampling semantics was originally presented in [47]. The work on deductive verification under the continuous interval semantics, presented in Chapter 8, is joint work with Tom Henzinger, Zohar Manna, and Amir Pnueli. The work was originally presented in [91].

Contents

Chapter 1

Overview

This thesis presents several methodologies for proving properties about *hybrid systems*, systems with an intermixing of continuous and discrete components. All analog physical phenomena, controlled by computer systems, and interacting via sensors and actuators, are hybrid systems. The increasing use of such systems in safety-critical applications, demands that such systems not exhibit catastrophic failure. Moreover, due to the ever-growing complexity of such systems, formal methods must be developed to allow both designers and users of such systems assurance that the systems function according to their specifications. This problem, of *verifying* that hybrid systems do indeed satisfy their specification, is the primary question addressed by this thesis.

1.1 Properties

There are two important classes of temporal properties for hybrid systems: *point-based* properties and *interval-based* properties. We will formally define these classes in Chapter 3.4. Informally, point-based properties are properties whose validity is dependent on individual states. For example, the temporal property that the variable x is always less than 5, specifies that each individual state of the system satisfy the formula $x \leq 5$. Interval-based properties are properties whose validity is dependent on intervals of time. As an example, suppose we have a gas burner which can be in two states: leak and non-leak. Consider the property which states that in every interval longer than 60 seconds, the cumulative amount of time that the system is in the leak state is only 1/20 of the length of the interval. To determine the validity of this property we must examine arbitrarily long (but finite) sequences of states, where the time from the beginning of the sequence to the end of the sequence is at least 60 seconds.

Point-based properties are often expressed in *Linear-Time Temporal Logic* (LTL) [116], a logic whose models are represented by sequences of states. In Chapter 3.3.1, we define the syntax and semantics of LTL, and present examples of properties written in LTL.

Interval-based properties, in general, are not expressible in LTL. Thus, a more expressive logic is needed. We use *Hybrid Temporal Logic* (HTL) [70, 91], a logic whose models are represented by piecewise-smooth functions mapping time points to particular system states. In Chapter 3.3.2, we define the syntax and semantics of HTL, and present examples of properties written in HTL. In Chapter 3.3.2, we compare LTL and HTL.

1.2 Models

Orthogonal to the issue of which kind of property we wish to prove is the question of which semantics best models our system. When analyzing hybrid systems, with their interplay of both discrete and continuous behavior, two natural semantics emerge: the *sampling semantics* [48, 133, 118, 99] and the *continuous semantics* [70, 114, 10, 125, 91]. We defer the formal definitions of these two semantics and their relative advantages and disadvantages to Chapter 3. For now, we view the sampling semantics as one where we take *snapshots* of the state of the system at discrete points in time: a finite number of snapshots in a finite amount of time, and an unbounded number of snapshots as time progresses. These infinite traces of system snapshots represent the possible behaviors of the system. With the continuous semantics, the behaviors of the system are also represented as infinite traces of the system, but this time, for every time point, a complete description of the system's state is given.[1]

1.3 Verification Methods

Once we have a particular property to prove and have chosen a particular semantics for our hybrid system, we still have to decide how to do the actual proof. That is, we must decide which methodology to use for the verification task. Several methodologies exist in the computer science literature: deductive approaches [99, 43, 54, 70, 111, 114], algorithmic approaches [9, 39, 95], and more recently, combinations of deductive and algorithmic approaches [47, 145].

The most widespread family of purely deductive approaches to verification rely on rules for reducing the task of verifying a temporal property over a system to checking the validity of many first-order verification conditions. Deductive approaches are user intensive, requiring the user to generate intermediate assertions and lemmas for completing the proof. However, since they use the wisdom of the user, they are usually applicable to a larger class of problems, at least in theory. In practice, due to the user-intensive nature of the deductive approach, fairly small examples are verified.

[1]For the reader familiar with the super-dense semantics (e.g., Maler, Manna, and Pnueli super-dense semantics [114]), we consider the super-dense semantics a subclass of the continuous semantics. We will come back to the distinctions between various continuous semantics in Chapter 3.

Algorithmic approaches, on the other hand, come in three varieties: symbolic model checking, where some symbolic enumeration of the system's states is done and checked against the specification; explicit model checking, where all the states are explicitly constructed and then checked against the specification; and on-the-fly enumerative algorithms, which construct only those states needed to check whether the property does indeed hold. Algorithmic approaches are computer intensive, suffering from the state-explosion problem. However, since they are automatic in nature, they are easier to apply, at least in theory. In practice, due to the state explosion problem, considerable massaging of the original problem is needed before it can be passed on to a model checker.

In combined deductive-algorithmic approaches, rules are given to deductively transform the original verification problem to a new verification problem, where the new verification problem can be algorithmically verified. This approach inherits both the advantages and disadvantages of the purely deductive approach and the purely algorithmic approach: it requires user interaction, but in a limited way; and it can be computer-intensive if the rules are not carefully applied.

1.4 Road Map

This thesis presents deductive and diagrammatic methodologies to prove both point-based and interval-based properties of hybrid systems, where the system is modeled in either the continuous or the sampling semantics. In Figure 1.1, we present an overview of the research papers underlying this thesis, and the chapters presenting the different methodologies, organized according to the dichotomy in the type of property we wish to prove and the dichotomy in the type of semantics we assign to the system. HTL is an abbreviation for *Hybrid Temporal Logic*, a logic for specifying both point-based and interval-based properties of hybrid systems, which we introduce in Chapter 3.3.2, and HTL verification is a rule-based approach for verifying properties specified in HTL.

The rest of this thesis is organized as follows. Chapter 2 introduces hybrid systems and several examples. Chapter 3 formally defines the behaviors of hybrid systems and introduces several logics for specifying properties about them. Chapter 4 develops a theory of stuttering and shows how stuttering applies to the verification task. In Chapter 5 we define several transition-based models for hybrid systems that will be used in the rest of the thesis for presenting verification strategies. Chapter 6 develops a theory of diagrammatic verification of point-based properties of hybrid systems when viewed under the discrete semantics, while Chapter 7 extends the diagrammatic approach to the super-dense (i.e., continuous) semantics. Chapter 8 develops a theory of deductive verification of point-based and interval-based properties of hybrid systems when viewed under the continuous semantics. In Chapter 9, we relate our approach to those of the rest of the verification community, and finally, in Chapter 10, we conclude with some observations about hybrid systems. Chapters 3–8 each contain a section called Contributions, which briefly highlight new results presented in this thesis.

	Point-based	**Interval-based**
Sampling	Hybrid Diagrams [dAKM97] Chapter 6	Stuttering Automata [unpublished] Chapter 4
Continuous	HTL Verification [KHMP94] Chapter 8 Hybrid Diagrams II [unpublished] Chapter 7	HTL Verification [KHMP94] Chapter 8

Figure 1.1: Road Map.

Chapter 2

Introduction to Hybrid Systems

Hybrid systems are systems with an intermixing of continuous and discrete components. For example, analog physical phenomena, controlled by computer systems, and interacting via sensors and actuators, are hybrid systems. Typically, we view hybrid systems as real-time systems that allow continuous state changes over time periods of positive duration and discrete state changes in zero time. It is this interaction of continuous and discrete change that make hybrid systems interesting and nontrivial targets for formal analysis. While mathematical methods for continuous equations and for discrete transitions have been studied independently for quite some time, the development of methods for formal reasoning about hybrid systems is relatively recent; its origin in computer science can be traced to [140, 114].

2.1 Basic Concepts

All hybrid systems include two important components: a *continuous component* and a *discrete component*. The continuous component includes all variables governed by physical laws. Such variables typically range over the real numbers. For example, analog devices and the variables they measure, such as an odometer for measuring speed, an altimeter for measuring altitude, and a barometer for measuring pressure, are all continuous components of hybrid systems.

The discrete component includes all discrete variables, which typically range over the natural numbers. For example, computer science data structures such as hash tables and integer arrays, as well as engineering switches for turning components on or off, are all representative discrete components of hybrid systems.

Specific components of real-world hybrid systems often include *sensors* for measuring continuous variables, *actuators* for changing the system's modus operandi, *channels* for communicating between systems' parts, and a *supervisory control* for managing the system. In this thesis, we will abstract away most of these components, and only record their effects indirectly. This simplification does not alter any of the results of the thesis, however, it does make the presentation a bit simpler.

2.2 Examples

In this section, we present several examples of hybrid systems that appear in the computer science literature. Of these examples, we will use two in particular—the gas burner example of Chaochen, Hoare, and Ravn [41] and the room-heater example of Nicollin, Sifakis, and Yovine [126]—as our running examples and present our various verification methodologies on them.

2.2.1 Real World

Real-world hybrid systems are ubiquitous. They range from the relatively simple automobile controllers that have overtaken our lives to the more complicated controllers for rockets, missiles, and nuclear power plants. In particular, consider an airplane hybrid system. The airplane has many continuous components including the airplane flaps, whose position can vary continuously within some predetermined range, and the engines, whose rotation speed can vary. These components govern other continuous variables such as velocity, acceleration, altitude, and location, all of which are measured by various sensors and whose values are displayed in the cockpit. The discrete components of the airplane include the myriad of cockpit switches including those that turn an engine on or off, raise or lower the landing gear, etc. Notice that it is the interaction between the discrete and the continuous component that governs the behavior of the system. The two components are inherently intertwined and can not be decoupled without making gross oversimplifications to the system's behavior.

2.2.2 Gas Burner

We now introduce our first running example, a variant of the gas burner example introduced by Chaochen, Hoare, and Ravn in [41], which we call system GAS. We will come back to this example several times to illustrate various concepts.

Suppose an engineer wishes to design a controller for a gas burner which is used to heat a home (see Figure 2.1). The gas burner has two switch settings, ($\mathit{switch} \in \{\text{Off, On}\}$), representing Off and On, respectively. The user of the gas burner expresses his desire to change the switch's setting through a request variable, R, that also has two possible values, ($R \in \{\text{Off, On}\}$). Unfortunately, when the switch is on, there is a possibility, due to the mechanics of the system, that some of the gas leaks. In this hazardous situation, gas leaks at a rate not greater than 1 unit/sec. Moreover, the controller has no way of determining the rate that gas is actually leaking when the switch is on. The only guarantee that the controller has is that no gas is leaking when the switch is in the off position.

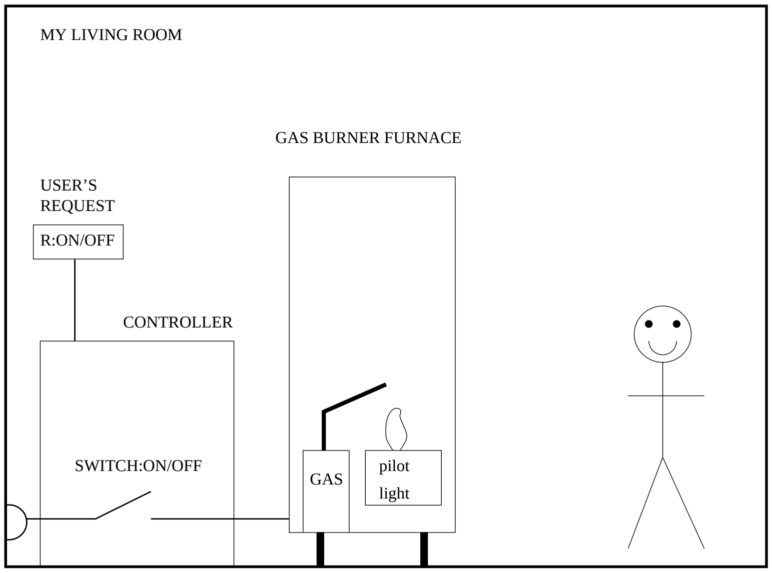

Figure 2.1: System GAS: A typical gas burning furnace found in overpriced California apartments.

The continuous components of this system are the rate at which gas leaks, and the rate at which time progresses. Time always progresses at a rate of 1. The discrete components are the internal system-controlled switch (*switch*) and the external user-controlled request (R).

2.2.3 Room Heater

As our second running example, we consider a variant of the temperature control system introduced by Nicollin, Sifakis, and Yovine in [126]. The system, which we call RH, consists of a room with a window and a heater (see Figure 2.2). The window, controlled by some independent agent, may be opened or closed at will. The heater turns on when the temperature is below the threshold temperature of 68°F and turns off when the temperature is above the threshold temperature of 72°F. To prevent mechanical stress, the heater has an embedded clock that prevents it from changing state within 60 seconds of the last change. Initially, the room temperature is below 60°F and the environment temperature (i.e. the temperature outside the room) is 60°F. For simplicity, we assume that the temperature of the environment remains constant at 60°F.

We let H denote the state of the heater, which ranges over the domain $\{\mathit{On}, \mathit{Off}\}$, W denote the state of the window, which ranges over the domain $\{\mathit{Open}, \mathit{Closed}\}$, T denote the global clock, y measure the time elapsed since the last switching *On*/*Off* of the heater, and x denote the temperature of the room. The general form of the evolution function for x is $F^x = 60 + h/c_1 + e^{-\Delta/c_1}(x - c_1 - h/c_1)$ when the heater is on, and $F^x = 60 + e^{-\Delta/c_1}(x - c_1)$ when the heater is off, where Δ measures the elapsed time since some initial reading of the temperature, h is a constant depending on the amount

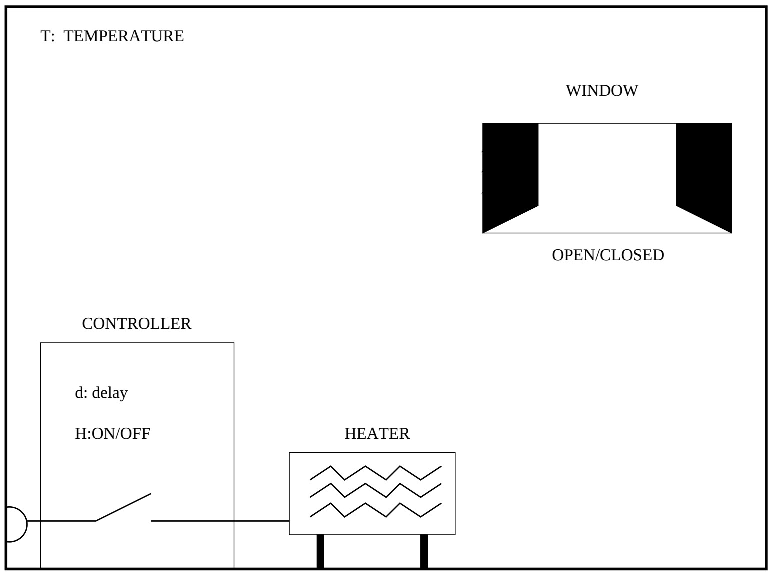

Figure 2.2: An electric heater found in overpriced California homes where the gas furnace fails to heat the house.

of heat given off by the heater, and c_1 is a constant based on the heat lost due to the window and depends on whether the window is open or closed. Note that the temperature of the room depends on an initial temperature reading and the time elapsed since that initial reading. In this thesis, we will use $h = 1/7$, $c_1 = 1/70$ when the heater is off, and $c_1 = 1/105$ when the heater is on. These assumptions give us the following evolution function for the temperature:

Discrete state of variables	Temperature
$H = \mathit{Off} \wedge W = \mathit{Closed}$	$60 + e^{-\Delta/105}(x - 60)$
$H = \mathit{Off} \wedge W = \mathit{Open}$	$60 + e^{-\Delta/70}(x - 60)$
$H = \mathit{On} \wedge W = \mathit{Closed}$	$75 + e^{-\Delta/105}(x - 75)$
$H = \mathit{On} \wedge W = \mathit{Open}$	$70 + e^{-\Delta/70}(x - 70)$

2.2.4 Summary

Many examples of hybrid systems have been analyzed in the computer science literature. Related work on a gas burner example can be found in [10, 39, 66, 96, 99, 117, 118, 145], while related work on a water level example can be found in [10, 53, 54, 73, 70, 85, 92, 121, 143, 151].

Heitmeyer, Jeffords, and Labaw [67] introduce a railroad crossing example, in which a group of trains pass a railroad crossing that is governed by a gate. When the gate is down, trains may pass safely through without crashing into cars trying to cross at the crossing; when the gate is up, cars may pass safely as no trains are allowed to pass. The railroad crossing example has been well studied in various incarnations [22, 27, 97, 142, 12, 18, 19, 77, 52, 108, 134], with Heitmeyer and Lynch [68] providing a formal description of the generalized railroad crossing example and clarifying Heitmeyer, Jeffords, and Labaw's [67] original example.

Jaffe, Leveson, Heimdahl, and Melhart [89] describe a controller for a nuclear reactor, where the controller's job is to insure that the nuclear reactor temperature stays within some predetermined range so as to avoid a core meltdown or an emergency shutdown. Subsequent work with this example is described in [92, 75, 89, 125].

Another well-studied example is the Cat and Mouse example of Maler, Manna, and Pnueli [114], in which a mouse is trying to avoid getting pounced on by a cat. Subsequent work with this example is described in [20, 43, 42, 59, 58, 94, 96, 118, 120, 125, 157].

Chapter 3

Specification of Hybrid Systems

In this chapter, we distinguish between several possible semantics for analyzing the behavior of hybrid systems. We then introduce two temporal logics—linear-time temporal logic and hybrid temporal logic—for the specification of properties about hybrid systems. For linear-time temporal logic we introduce two different semantics: a sampling semantics based on Manna and Pnueli [116] and a continuous semantics based on Maler, Manna, and Pnueli [114]. For hybrid temporal logic we present a single continuous semantics based on [70, 91]. Finally, we formally introduce the distinction between point-based properties and interval-based properties.

3.1 Contributions

The contributions of this chapter are

- a new presentation of a super-dense semantics,
- a conservative extension of linear-time temporal logic to our version of the super-dense semantics,
- a continuous interval semantics, and
- a new logic, hyrid temporal logic, interpreted under our continuous interval semantics.

3.2 Behavior of Hybrid Systems

As hybrid systems consist of both discrete and continuous behavior, a natural question is how should we model their behavior. Several possibilities have emerged over the years, and we discuss each one in turn. Common among all models is the partition of the set of variables $\mathcal{V}$ into *discrete variables*, $\mathcal{V}_d$, and continuous variables, $\mathcal{V}_h$. We denote the domain of an arbitrary variable $y \in \mathcal{V}$ as $\mathcal{D}_y$.

3.2.1 Sampling Semantics

Under the *sampling semantics* [15, 32, 47, 54, 68, 69, 95, 117, 100, 118, 126, 141], the behavior of a hybrid system is viewed as a set of infinite sequences of *snapshots*. Each snapshot records the value of all the system's variables (i.e., the *state* of the system), and also the value of some master clock which records the passage of time (known as the *global time clock*). Each particular infinite sequence of snapshots, which we call a *sampled run* of the system, represents one possible execution of the system. Fragments of sampled runs are called *sampled run fragments*. We denote sampled runs and sampled run fragments with the letter σ, we denote snapshots with the letter s, we denote the global time clock with the letter T, and we refer to the value of the global time clock at snapshot s as the *time-stamp* of s or $time(s)$, where *time* is a function that extracts the time-stamp of s. Snapshots in sampled runs have an inherent ordering, denoted by their position in the sampled run. We require that global time not decrease over the course of a sampled run. In particular, two snapshots may have the same value of global time (i.e., they have the same time-stamp, but assign different values to the other variables). We also require that time diverge. That is, for any value t, there is a snapshot s in the sampled run that has $time(s) > t$.

For example, let us consider the Room Heater example introduced in Chapter 2.2.3. In Figure 3.1, we present several possible sampled runs of the system. The reason for the multitude of sampled runs is:

1. the system may have some inherent nondeterminism in it. That is, at a given point in time, the system may proceed in several possible ways, and each possible way may lead to a different set of sampled runs. For example, in system RH, the window may be opened or closed at any point in time, which alters the evolution of the variable measuring the temperature, namely x;

2. there are many possible points of time at which we may take our snapshot. In fact, as there are an uncountable number of time points in the real world, we have an uncountable number of sampled runs of the system.

Often associated with a sampled run is the notion of a *sampling rate*. The sampling rate is the rate at which we take system snapshots. For example, in Figure 3.1, σ_1 has a sampling rate of 1/60 per second, representing the fact that one snapshot is taken every 60 seconds. In this thesis, we will not require our sampled runs to have a fixed sampling rate. Our only requirement is that within each finite period of time, there are only a finite number of snapshots recorded. As such, it is possible that the sampled snapshots miss many crucial events that the hybrid system undergoes. Some researchers [117, 118] work around this problem by introducing the concept of important events, which are events that the system is required to sample[1]. Unfortunately, this approach has severe limitations as Pnueli points out [133]. In particular, even with important events, our sampling

[1]More accurately, authors require that any sampled run not miss sampling the important event.

σ_1: $\langle$*Off*, *Closed*, 0, 50, 0$\rangle$, $\langle$*Off*, *Closed*, 60, 54.35, 60$\rangle$, $\langle$*On*, *Closed*, 60, 63.34, 120$\rangle$,
$\langle$*On*, *Closed*, 120, 68.41, 180$\rangle$, $\langle$*On*, *Closed*, 180, 71.28, 240$\rangle$,
$\langle$*Off*, *Closed*, 37.4, 68.4, 300$\rangle$

σ_2: $\langle$*Off*, *Closed*, 0, 50, 0$\rangle$, $\langle$*Off*, *Closed*, 50, 53.79, 50$\rangle$, $\langle$*Off*, *Closed*, 60, 54.35, 60$\rangle$,
$\langle$*On*, *Closed*, 0, 54.35, 60$\rangle$, $\langle$*On*, *Closed*, 60, 63.34, 120$\rangle$, $\langle$*On*, *Closed*, 120, 68.41, 180$\rangle$,
$\langle$*On*, *Closed*, 180, 71.28, 240$\rangle$, $\langle$*On*, *Closed*, 202.6, 72, 262.6$\rangle$,
$\langle$*Off*, *Closed*, 0, 72, 262.6$\rangle$, $\langle$*Off*, *Closed*, 37.4, 68.4, 300$\rangle$,
$\langle$*Off*, *Closed*, 42.55, 68, 305.15$\rangle$, $\langle$*On*, *Closed*, 0, 68, 305.15$\rangle$

σ_3: $\langle$*Off*, *Closed*, 0, 50, 0$\rangle$, $\langle$*Off*, *Open*, 0, 50, 0$\rangle$, $\langle$*Off*, *Open*, 50, 55.1, 50$\rangle$,
$\langle$*Off*, *Closed*, 50, 55.1, 50$\rangle$, $\langle$*Off*, *Closed*, 60, 55.55, 60$\rangle$, $\langle$*On*, *Closed*, 0, 55.55, 60$\rangle$,
$\langle$*On*, *Closed*, 20, 58.55, 80$\rangle$, $\langle$*On*, *Closed*, 50, 62.64, 110$\rangle$, $\langle$*On*, *Closed*, 90, 66.55, 150$\rangle$

Figure 3.1: Several sampled run fragments of system RH. Each tuple represents one snapshot: $\langle H, W, y, x, T\rangle$. (Values have been rounded to 2 decimal places.)

semantics may miss interesting events. For example, each of the runs in Figure 3.1 does not record the system when the global clock T is 3.1415. Are we therefore to conclude that because some sampled runs do not contain a snapshot at 3.1415, the system does not satisfy the property that eventually the global time of the system is 3.1415? Unfortunately, with the sampling semantics, the answer is yes, global time is not necessarily ever equal to 3.1415!

3.2.2 Dense and Super-Dense Semantics

Because of the limitations of the sampling semantics, some researchers [114, 8, 32, 40, 78, 72] have suggested using a *dense* or *super-dense semantics*. The difference between the dense and super-dense semantics is that in the super-dense semantics, as in the sampling semantics, two snapshots may have the same time-stamp, whereas in the dense semantics we require that all snapshots have unique time-stamps. In this thesis, we will use the super-dense semantics. We first give a presentation of a super-dense semantics based on Maler, Manna, and Pnueli [114]. In the next section, we will refine this semantics slightly when defining a super-dense version of temporal logic. This refinement will help us formulate verification rules, and resolve some technical problems with [114]'s semantics.

The underlying time domain of the super-dense semantics is the nonnegative real numbers. As with the sampling semantics, the behavior of a hybrid system is viewed as a set of infinite sequences of snapshots, but now there is an additional component f, where f is a tuple of piecewise continuous functions, one for each variable $y \in \mathcal{V}$ ($f_y : \mathcal{R}^+ \mapsto \mathcal{D}_y$). Each particular infinite sequence of snapshots is called a *super-dense run* of the system. The snapshots represent *discrete moments* in time, while the interludes between these snapshots represent *continuous regions* made up of an uncountable number of *continuous moments*.[2] For each variable $y \in \mathcal{V}$, the piecewise continuous function f_y denotes the value of y during the continuous regions and we require that the discontinuities only occur

[2]The concept of moments and the corresponding terminology comes from Maler, Manna, and Pnueli [114].

at the discrete moments. That is, during any continuous region, say between adjacent snapshots s_i and s_{i+1}, with time-stamps t_i and t_j, respectively, we require that the following two properties are satisfied:

1. for any discrete variable $y \in \mathcal{V}_d$, f_y is constant over the range $[t_i, t_j)$ and equal to the value at discrete moment s_i.

2. for any continuous variable $y \in \mathcal{V}_c$, f_y is a continuous function over the range $[t_i, t_j)$. For the end point t_i, it is only required that f_y be continuous from the right.

We also require the start of the system (i.e., $T = 0$) to be one of the sampled points. As before, we denote super-dense runs and super-dense run fragments with the letter $\sigma = \langle \bar{s}, f \rangle$, where $\bar{s}$ denotes an infinite sequence of discrete moments and f is a tuple of piecewise continuous functions. Once again, $\mathit{time}(s)$ extracts the time-stamp of discrete moment s. Note that the continuous region between s_i and s_{i+1} is an open region that includes all the continuous moments from $\mathit{time}(s_i)$ to $\mathit{time}(s_{i+1})$, but not including $\mathit{time}(s_i)$ and $\mathit{time}(s_{i+1})$. The *time-structure* induced by σ is defined as $\mathcal{T}_\sigma = \{\langle i, t\rangle \mid i \in \mathbb{N}, t = \mathit{time}(s_i) \lor \mathit{time}(s_i) < t < \mathit{time}(s_{i+1})\}$. The set $\mathcal{T}_\sigma$ is ordered by the lexicographic ordering

$$\langle i, t\rangle \succ \langle i', t'\rangle \ \mathit{iff} \quad i < i' \ \mathit{or} \quad (i = i' \ \mathit{and} \quad t < t') \ .$$

This ordering induces an ordering on triples of the form $\langle i, \mathit{time}(s), s\rangle$ and $\langle i, t, f(t)\rangle$, where s is a discrete moment and $f(t)$ is a continuous moment. The ordering is the $\succ$ ordering where we ignore the particular (discrete or continuous) moment. We will use these two orderings interchangeably.

For example, let us consider the super-dense runs of the Room Heater example. In Figure 3.2, we present two possible super-dense runs of the system, which are based on the sampled runs presented in Figure 3.1. Note that σ_1 can not be extended to a super-dense run because it violates the requirement that discrete variables do not change during intervening continuous regions. Thus, we present only two super-dense runs, σ_4 and σ_5 based on σ_2 and σ_3, respectively. The only difference for these two runs is that now we also present the value of the variables at continuous regions through the function f. Since for discrete variables, f is completely determined by its values at the discrete moments, and f_T is exactly the value of time, we present only f_x and f_y.

The super-dense semantics resembles closely the continuous interval semantics, which we discuss next.

3.2.3 Continuous Interval Semantics

As with the super-dense semantics, runs in the *continuous interval semantics* [10, 125, 70, 91] allow one to recover the precise description of the system at any point in time. With the super-dense semantics, this is done in two stages: giving the value of variables at discrete moments explicitly and

σ_2: $\langle$*Off*, *Closed*, 0, 50, 0$\rangle$, $\langle$*Off*, *Closed*, 50, 53.79, 50$\rangle$, $\langle$*Off*, *Closed*, 60, 54.35, 60$\rangle$,
$\langle$*On*, *Closed*, 0, 54.35, 60$\rangle$, $\langle$*On*, *Closed*, 60, 63.34, 120$\rangle$, $\langle$*On*, *Closed*, 120, 68.41, 180$\rangle$,
$\langle$*On*, *Closed*, 180, 71.28, 240$\rangle$, $\langle$*On*, *Closed*, 202.6, 72, 262.6$\rangle$, $\langle$*Off*, *Closed*, 0, 72, 262.6$\rangle$,
$\langle$*Off*, *Closed*, 37.4, 68.4, 300$\rangle$, $\langle$*Off*, *Closed*, 42.55, 68, 305.15$\rangle$, $\langle$*On*, *Closed*, 0, 68, 305.15$\rangle$

$$f_x(t) = \left\{ \begin{array}{ll} 60 + e^{-t/105}(50-60) & \forall t \in [0,60) \\ 75 + e^{-(t-60)/105}(54.35-75) & \forall t \in [60,262.6) \\ 60 + e^{-(t-262.6)/105}(72-60) & \forall t \in [262.6,305.15) \end{array} \right\}$$

$$f_y(t) = \left\{ \begin{array}{ll} t & \forall t \in [0,60) \\ t-60 & \forall t \in [60,262.6) \\ t-262.6 & \forall t \in [262.6,305.15) \end{array} \right\}$$

σ_3: $\langle$*Off*, *Closed*, 0, 50, 0$\rangle$, $\langle$*Off*, *Open*, 0, 50, 0$\rangle$, $\langle$*Off*, *Open*, 50, 55.1, 50$\rangle$,
$\langle$*Off*, *Closed*, 50, 55.1, 50$\rangle$, $\langle$*Off*, *Closed*, 60, 55.55, 60$\rangle$, $\langle$*On*, *Closed*, 0, 55.55, 60$\rangle$,
$\langle$*On*, *Closed*, 20, 58.55, 80$\rangle$, $\langle$*On*, *Closed*, 50, 62.64, 110$\rangle$, $\langle$*On*, *Closed*, 90, 66.55, 150$\rangle$

$$f_x(t) = \left\{ \begin{array}{ll} t & \forall t \in [0,60) \\ t-60 & \forall t \in [60,150) \end{array} \right\}$$

$$f_y(t) = \left\{ \begin{array}{ll} t & \forall t \in [0,60) \\ t-60 & \forall t \in [60,150) \end{array} \right\}$$

Figure 3.2: Two super-dense run fragments of system RH. Each tuple represents one discrete moment: $\langle H, W, y, x, T\rangle$, respectively. (Values have been rounded to 2 decimal places.)

giving a piecewise continuous function to recover the value of variables at the intervening continuous moments. In the continuous semantics we dispense with the discrete moments and give the value of all variables through a tuple of piecewise smooth functions. The approach we follow is from Kapur, Henzinger, Manna, and Pnueli [91].

Time is modeled by the nonnegative real line $\mathcal{R}^+$. A (left-closed right-open) *interval* $[a,b)$, where $a \in \mathcal{R}^+$, $b \in \mathcal{R}^+ \cup \{\infty\}$, and $a < b$, is the set of points $t \in \mathcal{R}^+$ such that $a \le t < b$. Let $I = [a,b)$ be an interval. A function $f\colon I \to \mathcal{R}$ is *piecewise smooth* in I if

- at a, the limit from the right of f exists, and the derivative from the right of f exists;
- at all internal points $t \in (a,b)$, the limit from the right, the limit from the left, and all left and right derivatives of f exist;
- at all points $t \in [a,b)$, f is continuous from the right;[3]
- if $b < \infty$, then the limit from the left of f exists at b, and the left derivative of f exists at b.

A *phase* $P = \langle I, f\rangle$ over $\mathcal{V}$ is a pair consisting of

[3]This condition allows one to chop an arbitrary piecewise smooth function into intervals of the form $[a,b)$ that are continuous from both the left and the right.

- a nonempty left-closed right-open interval $I = [a, b)$, and
- a type-consistent family $f = \{f_x \mid x \in \mathcal{V}\}$ of functions $f_x : I \to \mathcal{D}_x$ that are piecewise smooth in I and assign to each point $t \in I$ a value for the variable $x \in \mathcal{V}$.

It follows that the phase P assigns to every real-valued time $t \in I$ a complete description of the hybrid system. Furthermore, the limit from the right of f at a and the limit from the left of f at b, if $b < \infty$, are defined.

The behavior of hybrid systems under the continuous interval semantics is described as a set of phases, where the interval of time is $\mathcal{R}^+$. Each phase that is a possible execution for the hybrid system is called a *continuous run* of the hybrid system. For example, in Figure 3.3 we present two possible continuous run fragments of hybrid system RH. Once again these runs are based on the sampled runs presented in Figure 3.1 and resemble the super-dense runs presented in Figure 3.2. Note that in the super-dense semantics, a particular point in time can satisfy two possible snapshots; however, in the continuous interval semantics this is not possible. Thus, in the continuous semantics, discrete actions either occur at precisely the same instance and their effects are recorded together, or they occur at distinct points in time.

3.3 Logics

Before presenting our two logics, we introduce some notation common to both. Let $\mathcal{V}$ be a finite set of typed variables, where the allowed types are *boolean*, *integer*, and *real*. We view the booleans and the integers as subsets of the reals, where *false* and *true* correspond to 0 and 1, respectively. A *state* $s\colon \mathcal{V} \to \mathcal{R}$ is a type-consistent interpretation of the variables in $\mathcal{V}$ (i.e., boolean variables may only be interpreted as 0 or 1, and integer variables may only be interpreted over the integers). We write $\Sigma_{\mathcal{V}}$ for the set of states.

3.3.1 Linear-Time Temporal Logic

In this section we introduce linear-time temporal logic and interpret the logic using two different semantics: the traditional sampling semantics described above (as presented in Manna and Pnueli [116]) and a more radical super-dense semantics, which is a variant of the super-dense semantics presented earlier. Both semantics are defined on the same syntax of the logic.

Syntax

The formulas φ of *Linear-Time Temporal Logic* (LTL) are defined inductively as follows:

$$\varphi \;:=\; \psi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \Box\,\varphi \mid \varphi_1\,\mathcal{U}\,\varphi_2 \mid \forall x \,.\, \varphi$$

$P_2 = [f, [0, 305.15)]$ where:

$$f_H(t) = \left\{ \begin{array}{ll} \mathit{Off} & \forall t \in [0,60) \\ \mathit{On} & \forall t \in [60,262.6) \\ \mathit{Off} & \forall t \in [262.6,305.15) \end{array} \right\}$$

$$f_W(t) = \left\{ \begin{array}{ll} \mathit{Closed} & \forall t \in [0,305.15) \end{array} \right\}$$

$$f_T(t) = \left\{ \begin{array}{ll} t & \forall t \in [0,305.15) \end{array} \right\}$$

$$f_x(t) = \left\{ \begin{array}{ll} 60 + e^{-t/105}(50-60) & \forall t \in [0,60) \\ 75 + e^{-(t-60)/105}(54.35-75) & \forall t \in [60,262.6) \\ 60 + e^{-(t-262.6)/105}(72-60) & \forall t \in [262.6,305.15) \end{array} \right\}$$

$$f_y(t) = \left\{ \begin{array}{ll} t & \forall t \in [0,60) \\ t-60 & \forall t \in [60,262.6) \\ t-262.6 & \forall t \in [262.6,305.15) \end{array} \right\}$$

$P_3 = [f, [0, 150)]$ where:

$$f_H(t) = \left\{ \begin{array}{ll} \mathit{Off} & \forall t \in [0,60) \\ \mathit{On} & \forall t \in [60,150) \end{array} \right\}$$

$$f_W(t) = \left\{ \begin{array}{ll} \mathit{Open} & \forall t \in [0,50) \\ \mathit{Closed} & \forall t \in [50,150) \end{array} \right\}$$

$$f_T(t) = \left\{ \begin{array}{ll} t & \forall t \in [0,305.15) \end{array} \right\}$$

$$f_x(t) = \left\{ \begin{array}{ll} t & \forall t \in [0,60) \\ t-60 & \forall t \in [60,150) \end{array} \right\}$$

$$f_y(t) = \left\{ \begin{array}{ll} t & \forall t \in [0,60) \\ t-60 & \forall t \in [60,150) \end{array} \right\}$$

Figure 3.3: Two continuous run fragments of system RH. (Values have been rounded to 2 decimal places.)

where $x \in V$ and ψ is any first-order formula. The other boolean connectives, such as $\wedge$ (conjunction) and $\rightarrow$ (implication), can be defined using $\vee$ and $\neg$ in the usual way. We use the standard temporal logic abbreviations:

$\diamondsuit\, \varphi$	stands for	$\neg \square \neg\varphi$
$\varphi \Rightarrow \psi$	stands for	$\square(\varphi \rightarrow \psi)$.

Sampling Semantics

The formulas of linear-time temporal logic under the sampling semantics are interpreted over infinite sequences of states, denoted by σ. Position i of sequence σ is written as (σ, i). The first position of the sequence σ is $(\sigma, 0)$. A sequence σ *satisfies* the linear-time temporal formula φ, denoted $\sigma \models \varphi$, iff $(\sigma, 0) \models \varphi$. For any position i, $(\sigma, i) \models \varphi$ according to the following inductive definition:

For a first-order formula ψ, $(\sigma, i) \models \psi$ iff the first-order formula ψ evaluates to *true* at state (σ, i).

$(\sigma, i) \models \neg\varphi$ iff $(\sigma, i) \not\models \varphi$.

$(\sigma, i) \models \varphi_1 \vee \varphi_2$ iff $(\sigma, i) \models \varphi_1$ or $(\sigma, i) \models \varphi_2$.

$(\sigma, i) \models \Box\,\varphi$ iff for all $j \geq i$, $(\sigma, j) \models \varphi$.

$(\sigma, i) \models \varphi_1 \mathcal{U} \varphi_2$ iff there exists $k \geq i$ such that $(\sigma, k) \models \varphi_2$ and for all $i \leq j < k$, $(\sigma, j) \models \varphi_1$.

$(\sigma, i) \models \forall x \,.\, \varphi$ iff $(\sigma', i) \models \varphi$ for all sequences σ' that differ from σ at most in the interpretation of x at position i.

Continuous Semantics

Our presentation of a continuous semantics (or more precisely, super-dense semantics) for linear-time temporal logic differs from the approach presented in Chapter 3.2.2. Our reasons for departure from existing approaches are based on two requirements that we wish our semantics to possess:

1. we want our super-dense semantics to be an extension of the discrete semantics, in the sense that
   - any run of the discrete semantics can be obtained from some run of the super-dense semantics by forgetting information or refining the super-dense sampling points;
   - any run of the super-dense semantics generates a run of the discrete semantics by forgetting information or refining the super-dense sampling points.
2. we want any logic for the super-dense semantics to preserve the intended meaning of the standard temporal operators.

Unfortunately, approaches such as Maler, Manna, and Pnueli [114] fail to satisfy the second of these requirements. In particular, the definition of the $\mathcal{U}$ operator is problematic. Consider the hybrid system with one variable x, whose sole behavior is represented as follows:

$$\forall t, f_x(t) = t\ .$$

If we consider any discrete run σ_d, obtained by sampling the system at some set of points that includes the start of the system, then σ_d will satisfy the linear-time temporal formula $(x \leq 10)\mathcal{U}(x > 10)$. Moreover, intuitively any super-dense run should also satisfy the formula. However, consider the super-dense run fragment where the discrete moments are at $T = 5$, $T = 10$, and $T = 15$, presented in Figure 3.4. Does this run fragment satisfy the formula $(x \leq 10)\mathcal{U}(x > 10)$? Intuitively, we would like the answer to be yes, since x is continuously less than 10 until finally it is greater than 10;

$$\sigma_{sd}\colon \langle 5,\ 5\rangle,\ \langle 10,\ 10\rangle,\ \langle 15,\ 15\rangle$$
$$f_x(t) = t, \forall t \in [0, 15)$$

Figure 3.4: Super-dense run fragment. Each tuple represents one discrete moment, and denotes the value of the variables x and T, respectively.

moreover, the instant after it fails to be less than or equal to 10, it is greater than 10. Unfortunately, under the semantics presented in Maler, Manna, and Pnueli [114] this formula is not true. The reason this formula is false is that there is no t such that both the following hold: for all $t' < t$, $x \leq 10$ is true at t' and for all $t'' \geq t$, $x > 10$ is true at t''.[4] The problem is that the semantics of $\varphi \mathcal{U} \psi$ given by [114] is sensitive to whether ψ holds at the limit from the right or not.[5]

Our solution requires some new definitions and notation, which we now introduce. An *atomic state formula* is a formula without any boolean connectives (i.e., without $\neg$, $\wedge$, $\vee$, etc.) or temporal operators. For example, $x = 10$ is an atomic state formula. The *subformulas* of a temporal formula φ, denoted $\mathit{Sub}(\varphi)$, are defined inductively as follows:

If φ is an atomic state formula then $\mathit{Sub}(\varphi) = \{\varphi\}$.

If φ is of the form $\neg\varphi_1$ then $\mathit{Sub}(\varphi) = \{\varphi\} \cup \mathit{Sub}(\varphi_1)$.

If φ is of the form $\varphi_1 \vee \varphi_2$ then $\mathit{Sub}(\varphi) = \{\varphi\} \cup \mathit{Sub}(\varphi_1) \cup \mathit{Sub}(\varphi_2)$.

If φ is of the form $\square\, \varphi_1$ then $\mathit{Sub}(\varphi) = \{\varphi\} \cup \mathit{Sub}(\varphi_1)$.

If φ is of the form $\varphi_1 \mathcal{U} \varphi_2$ then $\mathit{Sub}(\varphi) = \{\varphi\} \cup \mathit{Sub}(\varphi_1) \cup \mathit{Sub}(\varphi_2)$.

If φ is of the form $\forall x\, .\, \varphi_1$ then $\mathit{Sub}(\varphi) = \{\varphi\} \cup \mathit{Sub}(\varphi_1)$.

A *ground super-dense run* over a quantifier-free formula φ is a super-dense run where for every atomic state subformula of φ the truth value of φ is constant throughout every continuous region. For formulas with quantifiers we require that under every instantiation of the quantifiers, the atomic state subformulas of φ have constant truth value throughout each continuous region in order to be a ground super-dense run. Note that in general, a super-dense run consists of a sequence of discrete moments followed by a continuous region. We denote by (σ, i) the $i + 1^{st}$ discrete moment or continuous region, where $i \geq 0$. For example, consider σ_2 of Figure 3.2. Then $(\sigma, 0)$ is the discrete moment $\langle \mathit{Off},\ \mathit{Closed},\ 0,\ 50,\ 0\rangle$, and $(\sigma, 1)$ is the continuous region over the time interval $(0, 60)$ where $f_x(t) = 60 + e^{-t/105}(50 - 60)$, $f_y(t) = t$, and $f_T(t) = t$, $\forall t \in (0, 60)$.

[4]Altering the semantics of $\mathcal{U}$ to require that for all $t' \leq t$ φ be true at t' and for all $t'' > t$ $x > 10$ be true at t'' makes the formula $(x \leq 10)\mathcal{U}(x > 10)$ valid, however, introduces a similar problem to the formula $(x < 10)\mathcal{U}(x \geq 10)$.

[5]Lakhnech [98] incorrectly associates the problem to whether or not intervals are open or closed. However, it is really a problem of whether or not ψ holds at the limit from the right, and not merely a problem of whether intervals are open or closed.

A super-dense run $\sigma_1 = \langle \bar{s}^1, f \rangle$ is a *sampling refinement* of $\sigma_2 = \langle \bar{s}^2, f \rangle$ if

- for each discrete moment s^2_j in $\bar{s}^2$ there exists a discrete moment s^1_i in $\bar{s}^1$ such that $s^1_i = s^2_j$.
- for each discrete moment s^1_i in $\bar{s}^1$ with $\mathit{time}(s^1_i) = t_i$, either there exists a discrete moment s^2_j in $\bar{s}^2$ such that $s^1_i = s^2_j$ or, for each variable $x \in \mathcal{V}$, $s^1_i(x) = f_x(t_i)$.
- the ordering of moments induced by $\succ_{\sigma_1}$ and $\succ_{\sigma_2}$ is the same. That is, for (discrete or continuous) moments m_1 and m_2 in σ_1 with $m_1 \succ_{\sigma_1} m_2$, the corresponding moments m'_1 and m'_2 in σ_2 have $m'_1 \succ_{\sigma_2} m'_2$.

Intuitively, σ_1 and σ_2 describe the same behavior, except that σ_1 is sampled more often (i.e., some of the continuous moments of σ_2 have become discrete moments).

We define the satisfaction relation $\models_c$ for a formula φ and its ground super-dense runs σ as follows:

For a first-order formula ψ, $(\sigma, i) \models_c \psi$ iff the first-order formula ψ evaluates to *true* at (σ, i).

$(\sigma, i) \models_c \neg\varphi$ iff $(\sigma, i) \not\models_c \varphi$.

$(\sigma, i) \models_c \varphi_1 \vee \varphi_2$ iff $(\sigma, i) \models_c \varphi_1$ or $(\sigma, i) \models_c \varphi_2$.

$(\sigma, i) \models_c \Box\, \varphi$ iff for all $j \geq i$, $(\sigma, j) \models_c \varphi$.

$(\sigma, i) \models_c \varphi_1 \mathcal{U} \varphi_2$ iff there exists $k \geq i$ such that $(\sigma, k) \models_c \varphi_2$ and for all $i \leq j < k$, $(\sigma, j) \models_c \varphi_1$.

$(\sigma, i) \models_c \forall x.\varphi$ iff $(\sigma', i) \models_c \varphi$ for all sequences σ' that differ from σ at most in the interpretation of x at position i.

For a ground super-dense run, $\sigma \models_c \varphi$ iff $(\sigma, 0) \models_c \varphi$. For a formula φ and an arbitrary super-dense run σ, if φ has a ground super-dense run σ' that is a sampling refinement of σ, then $\sigma \models_c \varphi$ iff $\sigma' \models_c \varphi$; if no such ground super-dense sampling refinement exists, then the $\models_c$ relation is undefined. The following definition shows that the $\models_c$ relation does not depend on which ground super-dense sampling refinement we consider.

Proposition 1 *For any temporal formula φ and super-dense run σ, if σ' is a ground super-dense sampling refinement of σ, then for all σ'' that are ground super-dense sampling refinements of σ, $\sigma \models_c \varphi$ iff $\sigma' \models_c \varphi$ iff $\sigma'' \models_c \varphi$.*

3.3.2 Hybrid Temporal Logic

Under the continuous interval semantics, the behavior of a hybrid system is modeled by a function that assigns to each time-point a system state. We require that, at each point, the behavior function has a limit from the left and a limit from the right. Discontinuities are points where the two limits differ. To specify properties of hybrid systems under the continuous interval semantics, we present a continuous-time interval temporal logic with a chop operator [65], denoted as ";", whose semantics is a continuous-time extension to Moszkowski's discrete-time chop operator [122].

Syntax

Because we wish to reason about physical phenomena in a natural and formal way, we introduce a logic that allows derivatives and limits as atomic expressions. Our logic, *Hybrid Temporal Logic* (HTL), is a variant of the hybrid temporal logic of Henzinger, Manna, and Pnueli [70].[6] For a variable $x \in V$, we write $\overleftarrow{x}$ for the limit from the right (the *right limit*), and $\overrightarrow{x}$ for the limit from the left (the *left limit*) of x. We write $\overleftarrow{\dot{x}}$ for the right derivative of x (with respect to time), and $\overrightarrow{\dot{x}}$ for the left derivative of x (with respect to time). Note that the terms, right-hand limit and left-hand limit, are consistent with standard calculus terminology, and that right-hand limits are applied at the left end of an interval, while left-hand limits are applied at the right end. To avoid confusion, we will mostly use the terms "limit from the right" and "limit from the left".

A *local formula* is a formula over the variables in V, their left and right limits, their left and right derivatives, and function and predicate symbols from a language $\mathcal{L}$. The formulas φ of HTL are defined inductively as follows:

$$\varphi \;:=\; \psi \mid \mathit{fin} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 ; \varphi_2 \mid \forall x \,.\, \varphi$$

where $x \in V$, ψ is an atomic local formula, and *fin* is a symbol which we will define below.

A *state formula* is a first-order logic formula over the variables in V (i.e., in which no limits, derivatives, or chops appear). If ψ is a state formula, we write $\overleftarrow{\psi}$ (and $\overrightarrow{\psi}$) for the local formula that results from ψ by replacing each variable occurrence x in ψ with its limit from the right $\overleftarrow{x}$ (and limit from the left $\overrightarrow{x}$, respectively).

Semantics

As in our presentation of the continuous interval semantics, time is modeled by the nonnegative real line $\mathcal{R}^+$, intervals are left-closed right-open subsets of the nonnegative real numbers, and phases, denoted $P = \langle I, f \rangle$ are pairs consisting of an interval and a type-consistent family of functions. We write

$$\overleftarrow{P} \;=\; \lim_{t \to a} \{ f(t) \mid a < t < b \}$$

for the *left-end limit state* $\overleftarrow{P} \in \Sigma_V$ of the phase P, and

$$\overrightarrow{P} \;=\; \lim_{t \to b} \{ f(t) \mid a < t < b \}$$

[6]We restrict ourselves to piecewise smooth functions that are always right continuous.

for the *right-end limit state* $\overrightarrow{P} \in \Sigma_V$ of P, if $b < \infty$.

Let $I_1 = [a, b)$ and $I_2 = [c, d)$ be two intervals, and let $P_1 = \langle I_1, f\rangle$ and $P_2 = \langle I_2, g\rangle$ be two phases. The phase P_2 is a *subphase* of P_1 if $I_2 \subseteq I_1$ and, for all $t \in I_2$, $g(t) = f(t)$. The phases P_1 and P_2 are *adjacent* if $b = c$. For two adjacent phases $P_1 = \langle [a, b), f\rangle$ and $P_2 = \langle [b, c), g\rangle$, we denote by $P_1{}^\frown P_2$ the phase $\langle [a, c), h\rangle$ such that h coincides with f on $t \in [a, b)$ and h coincides with g on $t \in [b, c)$. The phase P is said to be *partitioned* by the phases P_1 and P_2 if $P = P_1{}^\frown P_2$.

The formulas of hybrid temporal logic are interpreted over phases. A phase $P = \langle [a, b), f\rangle$ *satisfies* the hybrid temporal formula φ, denoted $P \models \varphi$, according to the following inductive definition:

For a local formula ψ, we distinguish between two cases.

1. If ψ does not contain left limits or left derivatives, then
   $P \models \psi$ iff the local formula ψ evaluates to *true*, where
   - x is interpreted as the value of f_x at a,
   $$x = f_x(a)$$
   - $\overleftarrow{x}$ is interpreted as the limit from the right of f_x at a,[7]
   $$\overleftarrow{x} = \lim_{t \to a} \{f_x(t) \mid a < t < b\}$$
   - $\overleftarrow{\dot{x}}$ is interpreted as the right derivative of f_x at a,
   $$\overleftarrow{\dot{x}} = \lim_{t \to a} \{(f_x(t) - \overleftarrow{x})/(t - a) \mid a < t < b\}.$$
2. If ψ contains left limits or derivatives, then $P \models \psi$ iff $b < \infty$ and the local formula ψ evaluates to *true*, where we evaluate variables, right limits, and right derivatives as above, and
   - $\overrightarrow{x}$ is interpreted as the limit from the left of f_x at b,
   $$\overrightarrow{x} = \lim_{t \to b} \{f_x(t) \mid a < t < b\}$$
   - $\overrightarrow{\dot{x}}$ is interpreted as the left derivative of f_x at b,
   $$\overrightarrow{\dot{x}} = \lim_{t \to b} \{(f_x(t) - \overrightarrow{x})/(t - b) \mid a < t < b\}.$$

$P \models \mathit{fin}$ iff $b < \infty$.

[7]The requirement that f_x is continuous from the right, guarantees that $\overleftarrow{x} = f_x(a)$.

$P \models \neg\varphi$ iff $P \not\models \varphi$.

$P \models \varphi_1 \vee \varphi_2$ iff $P \models \varphi_1$ or $P \models \varphi_2$.

$P \models \varphi_1 ; \varphi_2$ iff there are two phases, P_1 and P_2, that partition P such that $P_1 \models \varphi_1$ and $P_2 \models \varphi_2$.

$P \models \forall x \,.\, \varphi$ iff $P' \models \varphi$ for all phases $P' = \langle [a,b), f' \rangle$ that differ from P at most in the interpretation f'_x of x.

We will freely use the first-order connectives, "$\wedge$", "$\rightarrow$", and "$\exists$" as they can be defined in terms of the other connectives in the usual way.

Note that because of the dependence of the satisfaction relation on the syntactic occurrence of left limits and derivatives in local formulas, one should be careful in substitutions of formulas referring to left limits and derivatives. For example, the formula $\overrightarrow{x} = \overrightarrow{x}$ is not equivalent to *true* because $\overrightarrow{x} = \overrightarrow{x}$ is false on all infinite intervals. Also, the formula $\exists y.\, \Box(y = \overset{\leftarrow\cdot}{x})$ is not always valid. In particular, any phase in which $\overset{\leftarrow\cdot}{x}$ is not continuous from the right will fail to satisfy the formula, since variables are required to be right continuous, while derivatives are not.

Abbreviations

As in Henzinger, Manna, and Pnueli [70], we define abbreviations for common temporal formulas. The following abbreviations express that a leftmost subphase, a rightmost subphase, or any subphase of a phase satisfies the formula φ:

$\lhd\varphi$	stands for	$\varphi \vee (\varphi; \mathit{true})$
$\rhd\varphi$	stands for	$\varphi \vee (\mathit{true}; \varphi)$
$\Diamond\varphi$	stands for	$(\lhd\varphi) \vee (\rhd\varphi) \vee (\mathit{true}; \varphi; \mathit{true})$.

Thus, we can express that all subphases of a phase satisfy φ as $\Box\varphi$, where:

$$\Box\varphi \quad \text{stands for} \quad \neg\Diamond\neg\varphi \;.$$

We also introduce the abbreviations:

inf	stands for	$\neg \mathit{fin}$
$\Diamond_f\varphi$	stands for	$\Diamond(\mathit{fin} \wedge \varphi)$
$\Box_f\varphi$	stands for	$\Box(\mathit{fin} \rightarrow \varphi)$
$\varphi \Rrightarrow_f \psi$	stands for	$\Box_f(\varphi \rightarrow \psi)$
$\Diamond_i\varphi$	stands for	$\Diamond(\mathit{inf} \wedge \varphi)$
$\Box_i\varphi$	stands for	$\Box(\mathit{inf} \rightarrow \varphi)$
$\varphi \Rrightarrow_i \psi$	stands for	$\Box_i(\varphi \rightarrow \psi)$.

The formulas $\Diamond_f \varphi$ and $\Box_f \varphi$ can be viewed as finitary versions of $\Diamond\, \varphi$ and $\Box\, \varphi$ which restrict our attention to finite intervals only.

A phase $\langle I, f\rangle$ is called *continuous* if for all $v \in V$, f_v is continuous at all internal points of I. The continuity of all variables can be specified by the formula

$$\mathit{continuous}: \quad \neg\exists U. \Big(\overleftarrow{U} = \overrightarrow{U} \;\; \wedge \;\; (\overleftarrow{U} = \overrightarrow{V}); (\overleftarrow{V} \neq \overrightarrow{U}) \Big) \,,$$

where U and V are tuples of variables of the same length. This formula states that it is impossible to break the phase into two adjacent subphases such that the left limit of the state variables at the left subphase differs from the right limit of the state variables at the right subphase. We distinguish between two types of variables: *rigid* and *flexible*. Rigid variables have the same value throughout a phase, whereas flexible variables can have different values at different points in time.

$$x \in C^0 \quad \text{stands for} \quad \Box(\overleftarrow{x} = x) \wedge \forall u, v \in \mathit{Rigid} \,.\, \Big[(\overrightarrow{x} = u); (\overleftarrow{x} = v) \;\rightarrow\; u = v \Big] \,.$$

$$x \in C^1 \quad \text{stands for} \quad x \in C^0 \wedge \forall u, v \in \mathit{Rigid} \,.\, \Big[(\overrightarrow{\dot{x}} = u); (\overleftarrow{\dot{x}} = v) \;\rightarrow\; u = v \Big] \,.$$

The formula $x \in C^0$ requires that for any partition of a phase P into two subphases, the left and right limits of x at the point of partitioning coincide. The formula $x \in C^1$ adds the analogous requirement for the first derivatives of x. Because our intervals are left-closed, we care more about derivatives from the right. Hence, when we write $\dot{x}$ we mean $\overleftarrow{\dot{x}}$. Thus, we will use $\dot{x}$ as an abbreviation for $\overleftarrow{\dot{x}}$.

As shown in [70], hybrid temporal logic subsumes many of the real-time temporal logics presented in Alur and Henzinger [14] and the duration calculus of Chaochen, Hoare, and Ravn [41]. For example, the following formulas show how to write typical real-time and duration calculus formulas in HTL. In particular, the formula

$$\Box\, \forall x \in C^0 \,.\, \Big[\Big(p \wedge x = 0 \wedge \Box(\dot{x} = 1) \wedge \overrightarrow{x} > 5 \Big) \;\rightarrow\; \Diamond(q \wedge x \leq 5) \Big]$$

asserts that every p-state (i.e., a state where p is true) of a phase is followed within 5 time units either by a q-state or by the end of the phase. The variable x is a "clock" that measures the length of all subphases starting with a p-state.

The formula

$$\forall x \in C^0 \,.\, \Big[\Big(\overleftarrow{x} = 0 \wedge \Box(p \rightarrow \dot{x} = 1) \wedge \Box(\neg p \rightarrow \dot{x} = 0) \Big) \;\rightarrow\; \overrightarrow{x} \leq 10 \Big]$$

asserts that the cumulative time that p is true in a phase is at most 10. Here the variable x is an "integrator" that measures the accumulated duration of p-states. The formula

$$\forall x, y \in C^0 \,.\, \Big[\Big(x = 0 \wedge \Box(p \rightarrow \dot{x} = 1) \wedge \Box(\neg p \rightarrow \dot{x} = 0) \wedge y = 0 \wedge \Box(\dot{y} = 1) \Big) \rightarrow \overrightarrow{x} = \overrightarrow{y} \Big]$$

asserts that almost all points of a phase are p-states.

Connection with Linear Temporal Logic

Our desire to reason about point-based properties in HTL leads to the obvious question; namely, when does a temporal formula φ have the "same semantics" as in HTL?

The following proposition states that HTL subsumes linear-time temporal logic (under both the sampling semantics and the continuous semantics) without nested temporal operators in a natural way.

Proposition 2 *For any state formula φ and phase $P = \langle I, f\rangle$:*

1. *$P \models \Diamond\, \varphi$ iff $\exists\, t \in I$ such that φ holds at t.*

2. *$P \models \Box\, \varphi$ iff $\forall\, t \in I$ φ holds at t.*

The proof of this proposition follows in a straightforward manner from the definitions of the derived HTL operators $\Box$ and $\Diamond$.

The following proposition, stated without proof, allows us to use first order tautologies as valid formulas of hybrid temporal logic:

Proposition 3 *For any state formula φ, if φ is a tautology of first order logic then $\Box\, \varphi$ is valid.*

3.3.3 Other Logics

Numerous other logics have been proposed for the specification of hybrid systems. Duration Calculus, introduced in Chaochen, Hoare, and Ravn [41] and extended to hybrid systems in Chaochen, Ravn, and Hansen [43], introduces a duration operator, denoted $\int$, that measures the duration of time a proposition p is true over an interval. Like HTL, the duration calculus has a chop operator. The version of the duration calculus that is extended to hybrid systems [43] allows one to specify values at the left and right endpoints of a phase, a feature that is not present in the original duration calculus ([41]). For example in the extended duration calculus, the safety requirement for system GAS would be $\mathbf{e}.x - \mathbf{b}.x \geq 60 \;\; \rightarrow \;\; 6(\mathbf{e}.L - \mathbf{b}.L) \leq \mathbf{e}.x - \mathbf{b}.x$, where $\mathbf{e}.x - \mathbf{b}.x \geq 60$ states that the length of the interval is less than 60 seconds, and $6(\mathbf{e}.L - \mathbf{b}.L) \leq \mathbf{e}.x - \mathbf{b}.x$ states that the accumulated leak time in the given interval is less than $1/6$ of the length of the interval. In the extended duration calculus, $\int$ is a derived operator. Its encoding in HTL is similar to its encoding in the extended duration calculus, the latter of which can be found in [43].

Real-time logics [2, 12, 15, 14, 24, 69, 72, 90, 112, 137, 141, 142] have also been used with varying degrees of success. In particular, work in real-time logics has provided a basis for decidability results for hybrid systems, since real-time clocks are a special type of continuous variable. The interested reader is referred to [11, 13, 15, 14, 31, 152].

Lamport is a strong advocate of using “old-fashioned” formalisms for specifying hybrid systems [3]. He advocates using TLA [100], the temporal logic of actions, for specifying hybrid systems. TLA is a temporal logic with a restricted next-time operator, and importantly, no built-in primitives for specifying real-time or hybrid properties. Instead any operators needed for specifying real-time properties are defined using TLA and ordinary mathematics. For example, TLA+, uses TLA and the standard integral operator to define durations [99].

The interval temporal logic (ITL) of Moszkowski [122] uses a discrete semantics involving finite intervals consisting of a finite number of states. This assumption is justified because ITL is a logic for hardware verification, where discretization is both natural and possible. The interval temporal logic we propose here (i.e., HTL) is intended to be used for verification of controllers governing hybrid systems, which by definition have continuous components. Hence, the need for a continuous interval semantics.

3.4 Properties: Point-Based vs. Interval-Based

We now have enough machinery to formally define point-based and interval-based properties. A *point-based* property is a property that can be expressed by an HTL formula which has no occurrences of limits or derivatives. For example, all state formulas express point-based properties. An *interval-based property* is a property that can only be expressed by an HTL formula that contains limits or derivatives. For example, $(\overleftarrow{x} = 1);(\overleftarrow{x} = 2)$ is a point-based property because it can also be expressed by the equivalent HTL formula $(x = 1);(x = 2)$. On the other hand, $(\overrightarrow{x} = 1);(\overleftarrow{x} = 2)$ specifies an interval-based property. For a variable x and a phase P, the semantics of HTL assigns the same value to $\overleftarrow{x}$ and x, and so all occurrences of right limits may be replaced by corresponding variable occurrences. Thus, the presence of right limits in a formula does not preclude it from being a point-based property.

3.5 Examples

3.5.1 Real World

Let us return to the airplane hybrid system introduced in Chapter 2.2.1. If we were to design a controller for this plane that could serve as an automatic pilot for when our human pilot wanted to nap, we would want our controller to satisfy several basic properties:

1. our plane does not crash into another plane or hit the ground;

2. our plane moves closer to its destination;

3. within any minute of time, our velocity does not change by more than 100 mph.

The first two properties are point-based properties. In particular, the first property is a *safety property* [6, 116], which states that nothing bad ever happens, while the second property is a liveness property [132, 116]. Finally, the third property is an interval-based property.

3.5.2 Gas Burner

Let us reconsider system GAS introduced in Chapter 2.2.2. In the competitive world of gas burner design, the engineer must meet the following safety requirement:

> In any subinterval, if the duration of the subinterval is at least 60 seconds, then the cumulative leak amount within the subinterval is less than one-sixth of the subinterval duration. The purpose of this requirement is to prevent an excessive amount of gas from leaking into my living room and possibly killing me.

We write this property in HTL as follows. Letting $\dot{L}$ represent the rate at which gas leaks from the system, and x represent the system's global clock, we can express the property as follows[8]:

$$\overrightarrow{x} - \overleftarrow{x} \geq 60 \quad \Rightarrow_f \quad 6(\overrightarrow{L} - \overleftarrow{L}) \leq \overrightarrow{x} - \overleftarrow{x}$$

3.5.3 Room Heater

Finally, we reconsider system RH. We are interested in proving that the room temperature eventually falls within the range of 65^{o}F to 75^{o}F, and that once the temperature is in this range, it will remain in this range forever. This can be written in LTL as follows:

1. $\Diamond(65 \leq x \leq 75)$;

2. $(65 \leq x \leq 75) \Rightarrow \Box(65 \leq x \leq 75)$.

3.6 Discussion

Several papers have compared some of the semantics we have discussed in terms of expressiveness [114, 133]. Logics based on the various semantics have been developed, and, not surprisingly, have varying decidability results. The interested reader is referred to Bouajjani, Echahed, and Sifakis [32], who show that the duration calculus with a sampling semantics is decidable, whereas the duration calculus with a dense semantics is undecidable. Henzinger, Kopke, and Wong-Toi [79] present decidability results that depend on the number of clocks and whether such clocks have an underlying dense or discrete time domain.

[8]We have dropped the units in the equation, but if $\dot{L}$ were measured in lbs/sec, then the constant 6 in the equation would really be 6sec/lbs, otherwise the units for $6(\overrightarrow{L} - \overleftarrow{L})$ and $\overrightarrow{x} - \overleftarrow{x}$ would be different.

Chapter 4

Stuttering Automata

A feature of many interval temporal logics, including HTL is that their propositional fragments are insensitive to stuttering. A language is insensitive to stuttering if whenever it accepts a word w it accepts any word obtained from w by repeating literals or deleting repeated literals. In this chapter, we study the underlying theory of insensitivity to stuttering, also known as stuttering invariance. We develop a theory of regular expressions and automata that are insensitive to stuttering, and give decision procedures for determining language emptiness and for determining when a particular word is in a language.

As an application of our theory of stuttering invariance, we study the propositional fragment of HTL. In particular, we give a decision procedure for the finite propositional fragment of HTL—a fragment which is useful when analyzing the control locations of hybrid automata (which will be introduced in Chapter 5) and when generating intermediary invariants for proving safety properties of hybrid systems. For example, two formulas in this fragment are (1) $\Box(q; p) \Rightarrow\!\!\succ_f \Box\, p$, a valid formula which states that every finite interval in which all subintervals have a q-state eventually followed by a p-state, have p continuously true in the interval; and (2) $\Box\, p \Rightarrow\!\!\succ_f \Box(q; p)$, an invalid formula which states that every finite interval which continuously satisfies p has all its subintervals satisfying q eventually followed by p.

4.1 Contributions

The contributions of this chapter are

- a theory of stuttering regular expressions that captures the notion of stuttering invariance,
- a corresponding theory of stuttering automata that is equivalent in expressive power to stuttering regular expressions, and
- a decision procedure for the propositional fragment of hybrid temporal logic.

4.2 Basic Concepts

4.2.1 Stuttering Regular Expressions

Let V be a fixed finite vocabulary, i.e., $V = \{x_1, \ldots, x_n\}$. A Σ-*state* over V is a set $U \subseteq V \cup \overline{V}$ (where $\overline{V} = \{\overline{x_1}, \ldots, \overline{x_n}\}$ and $\overline{x_i} = \neg x_i$, for all $x_i \in V$) such that for all $x_i \in V$, either $x_i \in U$ or $\overline{x_i} \in U$, but not both. We let Σ be the set of all Σ-states over V and write particular Σ-states as σ. A Σ-*word* is a finite string $w \in \Sigma^*$[1]. We use Λ to denote the empty string (i.e., $w\Lambda = \Lambda w = w$). We are really concerned with the theory of non-empty strings, but we use Λ to make the presentation a bit cleaner. A Σ-*proposition* is a boolean formula over V using the the logical constant *true* and the logical connectives $\vee$, $\wedge$, and $\neg$.

We say a Σ-state σ satisfies a Σ-proposition ψ (denoted $\sigma \models \psi$) according to the following inductive definition:

$$\begin{array}{lll l}
\sigma \models \textit{true} & & & \\
\sigma \models x_i & \text{iff} & x_i \in \sigma & \text{for any } x_i \in V \\
\sigma \models \psi_1 \vee \psi_2 & \text{iff} & \sigma \models \psi_1 \text{ or } \sigma \models \psi_2 & \\
\sigma \models \psi_1 \wedge \psi_2 & \text{iff} & \sigma \models \psi_1 \text{ and } \sigma \models \psi_2 & \\
\sigma \models \neg\psi_1 & \text{iff} & \sigma \not\models \psi_1 &
\end{array}$$

We define the extended regular expressions (EREs) as follows: ψ is an ERE for any Σ-proposition ψ. If φ_1 and φ_2 are EREs, then

$$\varphi_1 + \varphi_2 \qquad (\varphi_1)(\varphi_2) \qquad \varphi_1^+ \qquad \varphi_1^* \qquad \neg\varphi_1$$

are all EREs.

We define the depth of an ERE as follows:

$$\begin{aligned}
\text{depth}(\psi) &= 1 \\
\text{depth}(E_1 + E_2) &= \max(\text{depth}(E_1), \text{depth}(E_2)) \\
\text{depth}(E_1 E_2) &= \text{depth}(E_1) + \text{depth}(E_2) \\
\text{depth}(E_1^+) &= 2 \cdot \text{depth}(E_1) \\
\text{depth}(E_1^*) &= 2 \cdot \text{depth}(E_1) \\
\text{depth}(\neg E_1) &= \text{depth}(E_1)
\end{aligned}$$

[1] As in the theory of regular expressions, Σ^* denotes the Kleene closure. We freely borrow the notation of regular expressions, and point out any differences between our notation and that of regular expressions. In particular, we use the standard symbols Λ, $\neg$, and $+$, and let $|w|$ be the length of a word w.

Let $\Sigma^{\geq m} = \{w \mid w \in \Sigma^+ \text{ and } |w| \geq m\}$ for $m \geq 1$. Thus, for an ERE E,

$$\Sigma^{\geq \text{depth}(E)} = \{w \mid w \in \Sigma^+ \text{ and } |w| \geq \text{depth}(E)\}.$$

Note that $\Sigma^{\geq 1} = \Sigma^+$. We define $Sat(E)$ inductively as follows:

$$\begin{aligned}
Sat(\psi) &= \{\sigma \mid \sigma \models \psi\} \qquad \text{for any } \Sigma\text{-proposition } \psi \\
Sat(\varphi_1 + \varphi_2) &= Sat(\varphi_1) \cup Sat(\varphi_2) \\
Sat(\varphi_1\varphi_2) &= \{w_1w_2 \mid w_1 \in Sat(\varphi_1),\ w_2 \in Sat(\varphi_2)\} \\
Sat(\varphi_1^+) &= \textstyle\bigcup_{i\geq 1}\{w \mid w \in Sat(\varphi_1^i)\}^2 \\
Sat(\varphi_1^*) &= \textstyle\bigcup_{i\geq 0}\{w \mid w \in Sat(\varphi_1^i)\} \\
Sat(\neg\varphi_1) &= \Sigma^* - Sat(\varphi_1) \qquad \text{for any } \varphi_1 \text{ not a } \Sigma\text{-proposition}
\end{aligned}$$

A Σ-word w *conventionally satisfies* an ERE E, $w \models_C E$, if $w \in Sat(E)$. Note that $\Lambda \not\models \psi$ for any Σ-proposition ψ. Thus, $\Lambda \not\models \psi$ and $\Lambda \not\models \neg\psi$.

We are interested in only a small subclass of EREs, which represent Σ-words that are insensitive to stuttering. Thus, if a word stutteringly satisfies a regular expression, then all stuttering variants of the word stutteringly satisfy the regular expression.

The class of stuttering-invariant EREs (denoted SRE) is formed from the following inductive definition: ψ^+ is an SRE for any Σ-proposition ψ. If φ_1 and φ_2 are SREs, then

$$\varphi_1 + \varphi_2 \qquad (\varphi_1)(\varphi_2) \qquad \varphi_1^* \qquad \neg\varphi_1$$

are all SREs. Clearly a stuttering-invariant ERE is also an ERE. We can define the intersection of two expressions, $E_1 \cap E_2$, as $\neg(\neg E_1 + \neg E_2)$. We will freely use $\cap$ in the rest of this chapter.

For any two Σ-words w and w', we say $w' \succeq w$ if w' is a *stretching* of w (i.e., if $w = \sigma_1 \ldots \sigma_n$, then w' is a stretching iff $w' = \sigma_1^{i_1} \ldots \sigma_n^{i_n}$ where $i_1, \ldots, i_n \geq 1$). Thus, a stretching of w can duplicate states, but not add any states not already in w nor remove states from w. Note that $\Lambda \not\preceq w$ and $\Lambda \not\succeq w$.

For any Σ-word $w = \sigma_1 \ldots \sigma_n$, for any $m \geq 1$, let $(w)^{St(m)} = \sigma_1^m \ldots \sigma_n^m$. That is, $(w)^{St(m)}$ is just like w except every state is stretched out m times. A Σ-word w stutteringly satisfies a SRE E (denoted $w \models_S E$) according to the following inductive definition:

$$\begin{array}{lll}
w \models_S \psi^+ & \text{iff} & w \models_C \psi^+ \\
w \models_S E_1 + E_2 & \text{iff} & w \models_S E_1 \text{ or } w \models_S E_2 \\
w \models_S E_1E_2 & \text{iff} & \exists v_1, v_2 \text{ such that } (w)^{St(2)} = v_1v_2,\ v_1 \models_S E_1,\ v_2 \models_S E_2 \\
w \models_S E_1^* & \text{iff} & \text{either } w = \Lambda \text{ or } w \models_S E_1^i, \text{ for some } i \geq 1 \\
w \models_S \neg E_1 & \text{iff} & w \not\models_S E_1
\end{array}$$

For any SRE E, we denote $SSat(E) = \{w \mid w \models_S E\}$.

Proposition 4 *For any Σ-proposition ψ, for any Σ-state σ, either $\sigma \models_C \psi$ or $\sigma \models_C \neg\psi$.*

The proof follows from the definition of satisfaction, and can be found in Chapter 4.6.

The next lemma shows that our definition of stuttering satisfaction constructs only those languages that are stuttering invariant. Thus, if $w \in SSat(E)$, then any word obtained from w by repeating literals or deleting repeated literals is also in $SSat(E)$. The lemma is proved by induction on the structure of E, and appears in Chapter 4.6.

Lemma 5 (Stuttering Invariance) *For any* SRE *E, for any Σ-word w,*

1. *if $w \models_S E$ then for all $w' \succeq w$, $w' \models_S E$.*
2. *if $\exists\ w' \succeq w$ such that $w' \models_S E$, then $w \models_S E$.*

The following lemma gives insight into the structure of expressions allowed in the class SRE. It relies only on the definition of conventional satisfaction: since our definition of stuttering satisfaction relies on conventional satisfaction, this lemma is useful in understanding when stuttering a state within a word does not change satisfaction.

Lemma 6 (Depth Expansion) *For any* SRE *E, any Σ-word w, and any Σ-state σ, if $w = v_1\sigma v_2$ for some $v_1, v_2 \in \Sigma^*$ and $w' = v_1\sigma^m v_2 \models_C E$ for some $m \geq \text{depth}(E)$, then for all $n \geq \text{depth}(E)$, $w'' = v_1\sigma^n v_2 \models_C E$.*

Proof of 6:

We prove the lemma by induction on the structure of E.

Case: Base

E is of the form ψ^+ for some Σ-proposition ψ. Suppose $w = v_1\sigma v_2$ for some $v_1, v_2 \in \Sigma^*$ and some Σ-state σ. Suppose $\exists\ m \geq \text{depth}(E) = 1$ such that $w = v_1\sigma^m v_2 \models_C E$. Fix such an m. As $v_1\sigma^m v_2 \models_C \psi^+$, we have $\sigma \models_C \psi$. So, for any $n \geq \text{depth}(E)$, $w'' = v_1\sigma^n v_2 \models_C \psi^+$.

Case: Inductive

Case: $E = E_1 + E_2$

Consider an arbitrary $w = v_1\sigma v_2$ for some $v_1, v_2 \in \Sigma^*$ and some Σ-state σ. Suppose $\exists\ m \geq \text{depth}(E)$ such that $w = v_1\sigma^m v_2 \models_C E_1 + E_2$. Fix such an m.

Then either $w' \models_C E_1$ or $w' \models_C E_2$. That is, $w' \models_C E_i$ for either $i = 1$ or $i = 2$. Fix such an i.

By induction hypothesis, as $m \geq \text{depth}(E) \geq \text{depth}(E_i)$, we get for all $n \geq \text{depth}(E_i)$, $v_1\sigma^n v_2 \models_C E_i$. So clearly, for all $n \geq \text{depth}(E_i)$, $v_1\sigma^n v_2 \models_C E_1 + E_2$. So for all $n \geq \text{depth}(E)$, $v_1\sigma^n v_2 \models_C E_1 + E_2$.

Case: $E = E_1E_2$

Consider an arbitrary $w = v_1\sigma v_2$ for some $v_1, v_2 \in \Sigma^*$ and some Σ-state σ. Suppose $\exists\ m \geq \text{depth}(E)$ such that $w = v_1\sigma^m v_2 \models_C E_1E_2$. Fix such an m.
Then as $v_1\sigma^m v_2 \models_C E_1E_2$, there exist Σ-words w_1 and w_2 where $w_1w_2 = v_1\sigma^m v_2$ and $w_1 \models_C E_1$ and $w_2 \models_C E_2$. Fix such w_1 and w_2.

Case: $v_1\sigma^m$ is a substring of w_1

Thus, $w_1 = v_1\sigma^m v_1'$ and $w_2 = v_1''$, where $v_2 = v_1'v_1''$. Fix such v_1' and v_1''.
As $v_1\sigma^m v_1' \models_C E_1$ and $m \geq \text{depth}(E) > \text{depth}(E_1)$, we get for all $n \geq \text{depth}(E_1)$, $v_1\sigma^n v_1' \models_C E_1$, by the induction hypothesis. So, for all $n \geq \text{depth}(E)$, $v_1\sigma^n v_1' \models_C E_1$. So, for all $n \geq \text{depth}(E)$, $w'' = v_1\sigma^n v_1'v_2' \models_C E_1E_2$.

Case: $\sigma^m v_2$ is a substring of w_2

Similar to the previous case.

Case: $w_1 = v_1\sigma^i$ and $w_2 = \sigma^j v_2$, where $i + j = m$

As $i + j = m \geq \text{depth}(E) = \text{depth}(E_1) + \text{depth}(E_2)$, either $i \geq \text{depth}(E_1)$ or $j \geq \text{depth}(E_2)$.

Case: $i \geq \text{depth}(E_1)$

Then $v_1\sigma^i \models_C E_1$ and $i \geq \text{depth}(E_1)$, so by induction hypothesis, for all $n \geq \text{depth}(E_1)$, $v_1\sigma^n \models_C E_1$. So consider any $k \geq \text{depth}(E) = \text{depth}(E_1) + \text{depth}(E_2)$.
If $j < \text{depth}(E_2)$ then $k - j \geq \text{depth}(E_1)$. So, $v_1\sigma^{k-j} \models_C E_1$ and $\sigma^j v_2 \models_C E_2$. So $v_1\sigma^k v_2 = v_1\sigma^{k-j}\sigma^j v_2 \models_C E_1E_2$.
If $j \geq \text{depth}(E_2)$ then by induction hypothesis, for all $\ell \geq \text{depth}(E_2)$, $\sigma^\ell v_2 \models_C E_2$. Let $\ell = \text{depth}(E_2)$. Then $k - \ell \geq \text{depth}(E_1)$. So $v_1\sigma^{k-\ell} \models_C E_1$ and $\sigma^\ell v_2 \models_C E_2$. So $v_1\sigma^k v_2 \models_C E_1E_2$.
Thus, in either case $v_1\sigma^k v_2 \models_C E_1E_2$. As k was arbitrarily chosen $\geq \text{depth}(E)$, we get for all $n \geq \text{depth}(E)$, $w'' = v_1\sigma^n v_2 \models_C E_1E_2$ as desired.

Case: $i < \text{depth}(E_1)$

Then $j \geq \text{depth}(E_2)$. Similar to the previous case.

Case: $E = \neg E_1$

Consider an arbitrary $w = v_1\sigma v_2$ for some $v_1, v_2 \in \Sigma^*$ and some Σ-state σ. Suppose $\exists\ m \geq \text{depth}(E)$ such that $w = v_1\sigma^m v_2 \models_C \neg E_1$. Fix such an m.
Assume that $\exists\ n \geq \text{depth}(E)$ such that $w'' = v_1\sigma^n v_2 \not\models_C E$. Fix such an n. We will reach a contradiction.
As $w'' = v_1\sigma^n v_2 \models_C E_1$, by induction hypothesis, for all $\ell \geq \text{depth}(E_1)$, $v_1\sigma^\ell v_2 \models_C E_1$. As $m \geq \text{depth}(E) = \text{depth}(E_1)$, $v_1\sigma^m v_2 \models_C E_1$. But as $v_1\sigma^m v_2 \models_C E$, we have $v_1\sigma^m v_2 \not\models_C E_1$. $\Rightarrow\Leftarrow$
So our assumption is false. For all $n \geq \text{depth}(E)$, $w'' = v_1\sigma^n v_2 \models_C E$.

Case: $E = E_1^*$

Consider an arbitrary $w = v_1 \sigma v_2$ for some $v_1, v_2 \in \Sigma^*$ and some Σ-state σ. Suppose $\exists\, m \geq \text{depth}(E_1^*) = 2 \cdot \text{depth}(E_1)$ such that $w = v_1 \sigma^m v_2 \models_C E_1^*$. Fix such an m.

As $w \neq \Lambda$, $w = v_1 \sigma^m v_2 \models_C E_1^i$, for some $i \geq 1$. Fix such an i.

Note that $\text{depth}(E_1^i) = i \cdot \text{depth}(E_1)$ so the induction hypothesis is not applicable. But we do know that $\exists w_1, \ldots w_i$ such that $\forall \ell,\ 1 \leq \ell \leq i,\ w_\ell \models_C E_1$. Moreover for some $1 \leq k < i$, σ^m is a substring of $w_k w_{k+1}$. Fix such a k. Thus, $w_k = w'_k \sigma^p$ and $w_{k+1} = \sigma^q w'_{k+1}$, where $p + q = m \geq 2 \cdot \text{depth}(E_1)$.

Case: $p = 0$

Then $w_{k+1} = \sigma^m w'_{k+1} \models_C E_1$. As $m \geq \text{depth}(E_1)$ by induction hypothesis we get $\forall n \geq \text{depth}(E_1)$, $\sigma^n w'_{k+1} \models_C E_1$. So, $\forall n \geq 2 \cdot \text{depth}(E_1)$, $\sigma^n w'_{k+1} \models_C E_1$. So, $\forall n \geq \text{depth}(E_1^*)$, $w_1 \ldots w_k \sigma^n w'_{k+1} \ldots w_i \models_C E_1^i$. So, $\forall n \geq \text{depth}(E_1^*)$, $v_1 \sigma^n v_2 \models_C E_1^*$.

Case: $q = 0$

Similarly done.

Case: $p \neq 0$, $q \neq 0$, $p \geq \text{depth}(E_1)$

This is similarly to the corresponding case for concatenation.

Case: $p \neq 0$, $q \neq 0$, $p < \text{depth}(E_1)$

Then as $p + q = 2 \cdot \text{depth}(E_1)$, $q \geq \text{depth}(E_1)$. This is similarly to the corresponding case for concatenation.

So by induction the lemma holds. ∎

4.2.2 Decision Procedure for $w \models_S E$

We now present a simple decision procedure for determining if $w \models_S E$ for any Σ-word w and any SRE E. The algorithm is based on a dynamic programming technique as described in [5, 86]. The essential idea is to construct a table which for each subexpression E' of E and for each substring x_{ij} of w gives the answer to the question:

$$\text{Is } x_{ij} \text{ in } SSat(E')?$$

where x_{ij} represents the substring of w starting from position i and of length j. The correctness of the algorithm is based on the Stuttering Invariance Lemma (Lemma 5).

Algorithm SSat: Given any Σ-word w and any SRE E, to determine if $w \models_S E$:

1. Enumerate all the subexpressions of E, starting from the simplest. That is, if E' is a proper subexpression of E'' then E' must appear before E''. Let m denote the number of subexpressions of E and let E_n denote the nth entry in the list. Note that $E_m = E$ since E is a subexpression of E and clearly a superexpression of all other subexpressions of E.

2. Form a three-dimensional table A, where A_{ijk} stores the answer to the question:

   Is x_{ij} in $SSat(E_k)$?

   The table is constructed as follows:

```
for $j = 1$ to $|w|$
  for $i = 1$ to $(|w| - j + 1)$
    for $k = 1$ to $m$
      Fill in the entry of $A_{ijk}$ with “yes” or “no” according to the rules below
      (a) if $E_k$ is of the form $\psi^+$ for some $\Sigma$-proposition $\psi$ then,
          if $\forall \ell,\ i \leq \ell \leq j,\ x_{\ell 1} \models_C \psi$, then “yes” else “no”.
      (b) if $E_k$ is of the form $E_{k_1} \cap E_{k_2}$ then,
          if $A_{ijk_1}$=“yes” and $A_{ijk_2}$=“yes”, then “yes” else “no”.
      (c) if $E_k$ is of the form $E_{k_1} + E_{k_2}$ then,
          if $A_{ijk_1}$=“yes” or $A_{ijk_2}$=“yes”, then “yes” else “no”.
      (d) if $E_k$ is of the form $\neg E_{k_1}$ then,
          if $A_{ijk_1}$=“yes”, then “no” else “yes”.
      (e) if $E_k$ is of the form $E^*_{k_1}$ then[3],
          for each $x_{i\ell}$ and $x_{(i+\ell)(j-\ell)}$, where $1 \leq \ell < j$
            if $A_{i\ell k_1}$=“yes” and $A_{(i+\ell)(j-\ell)k}$=“yes” then “yes”
          if $A_{ijk_1}$=“yes” then “yes”
          if $A_{ijk}$ is not filled with a “yes” via any of the above, then enter “no”.
      (f) if $E_k$ is of the form $E_{k_1} E_{k_2}$ then,
          for each $x_{i\ell}$ and $x_{(i+\ell)(j-\ell)}$, where $1 \leq \ell < j$
            if $A_{i\ell k_1}$=“yes” and $A_{(i+\ell)(j-\ell)k_2}$=“yes” then “yes”
          for each $x_{i\ell}$ and $x_{(i+\ell-1)(j-\ell+1)}$, where $1 \leq \ell \leq j$
            if $A_{i\ell k_1}$=“yes” and $A_{(i+\ell-1)(j-\ell+1)k_2}$=“yes” then “yes”
          if $\Lambda \in SSat(E_{k_1})$ and $A_{ijk_2}$=“yes” then “yes”
          if $\Lambda \in SSat(E_{k_2})$ and $A_{ijk_1}$=“yes” then “yes”
          if $A_{ijk}$ is not filled with a “yes” via any of the above, then enter “no”.
```

3. Return the entry of $A_{1|w|m}$

Since we construct the entry of a A_{ijk} based on its subexpressions and substrings, and we start from the simplest expressions and substrings, we are guaranteed that the necessary entries are

[3]Note that a non-empty word is in $E^*_{k_1}$ iff it is in $E^+_{k_1}$.

already computed. The correctness of the algorithm is straightforward; the only interesting case is concatenation, whose correctness relies on the Stuttering Invariance Lemma. The proof can be found in Chapter 4.6. The algorithm relies on an auxiliary algorithm for determining whether $\Lambda \in SSat(E_k)$. We present this algorithm below. Once again we use dynamic programming to form an array B, where B_k stores the answer to the question "Is Λ in $SSat(E_k)$?"

```
for $k = 1$ to $m$
  Fill in the entry of $B_k$ with "yes" or "no" according to the rules below
    1. if $E_k$ is of the form $\psi^+$ for some $\Sigma$-proposition $\psi$ then "no".

    2. if $E_k$ is of the form $E_{k_1} \cap E_{k_2}$ then,
       if $B_{k_1}$="yes" and $B_{k_2}$="yes", then "yes" else "no".

    3. if $E_k$ is of the form $E_{k_1} + E_{k_2}$ then,
       if $B_{k_1}$="yes" or $B_{k_2}$="yes", then "yes" else "no".

    4. if $E_k$ is of the form $\neg E_{k_1}$ then,
       if $B_{k_1}$="yes", then "no" else "yes".

    5. if $E_k$ is of the form $E_{k_1}^*$ then "yes".

    6. if $E_k$ is of the form $E_{k_1} E_{k_2}$ then,
       if $B_{k_1}$="yes" and $B_{k_2}$="yes", then "yes" else "no".
```

We now analyze the cost of the above algorithm. We first define some measures on the size of SREs and EREs. The *length* of an ERE E is defined as follows:

$$\begin{aligned}
\text{length}(\psi) &= |\psi| \\
\text{length}(E_1 + E_2) &= \text{length}(E_1) + \text{length}(E_2) + 1 \\
\text{length}(E_1 \cap E_2) &= \text{length}(E_1) + \text{length}(E_2) + 1 \\
\text{length}(E_1 E_2) &= \text{length}(E_1) + \text{length}(E_2) \\
\text{length}(E_1^+) &= \text{length}(E_1) + 1 \\
\text{length}(E_1^*) &= \text{length}(E_1) + 1 \\
\text{length}(\neg E_1) &= \text{length}(E_1) + 1 \qquad \text{for } E_1 \text{ not a } \Sigma\text{-proposition}
\end{aligned}$$

The *llength* of an ERE E is defined similarly except $\text{llength}(\psi) = 1$. As the class SRE is a subclass of ERE, this defines the measures length and llength on SREs as well.

The number of entries in the table constructed for Algorithm SSat is $O(\text{llength}(E)|w|^2)$. Each entry where E_k is of the form ψ^+ for some Σ-proposition ψ takes $O(|\psi||w|)$ time, whereas all other entries take $O(|w| + \text{llength}(E))$ time. The latter is caused by the fact that concatenation requires time $O(|w| + \text{llength}(E))$. Thus, each entry is bounded by $O(|\psi||w| + |w| + \text{llength}(E))$, which in turn is bounded by $O(|w||\psi| + \text{llength}(E))$. Finally, the overall algorithm is bounded by

$O(\text{llength}(E)|w|^2(|w||\psi| + \text{llength}(E)))$, which is bounded by $O((\text{length}(E) + |w|)^4)$.

4.2.3 Stuttering Automata

In this section, we develop a theory of stuttering-invariant automata, which corresponds to the theory of regular expressions developed in Section 4.2.1.

A *stuttering-invariant table* T on Σ, where Σ is the input alphabet consisting of Σ-states as defined in Section 2, is a triple $T = \langle S, M, S_0 \rangle$ where S is a finite set of states, $S_0 \subseteq S$ is the set of initial states, and M is a function $M \colon S \times \Sigma \to 2^S$, called the *transition function*, such that

$$\text{for all } s_i, s_j \in S, \text{ if } s_j \in M(s_i, \sigma) \text{ then } s_j \in M(s_j, \sigma) \qquad \dagger$$

($\dagger$) is called the *stuttering condition*.

A *stuttering finite automaton* is a quadruple $\mathcal{A} = \langle S, M, S_0, F \rangle$ where $\langle S, M, S_0 \rangle$ is a stuttering insensitive table and $F \subseteq S$ is the set of final states. The automaton is deterministic if $|S_0| = 1$ and M is deterministic, i.e., $|M(s_j, \sigma)| = 1$ for all $s_j \in S$. The size of an automaton, $|\mathcal{A}|$, is the number of states in S.

For an arbitrary n-tuple $\langle a_1, \ldots, a_n \rangle$, the j^{th} projection $p_j(\langle a_1, \ldots, a_n \rangle) = a_j$, where $1 \leq j \leq n$. A tape X of length $n \leq \omega$ on Σ is a sequence of length n over the alphabet Σ. $X(i)$ represents the i^{th} element of X, which we will also write as x_i. $X(i, j)$ represents the portion of the tape from position i to position $j - 1$. Let ζ be a sequence of length $n \leq \omega$ over some arbitrary set A. We view ζ as a function from $\{i < n\}$ to A, and also denote it by $(\zeta(i))$. For such a sequence ζ, we let $\zeta(*)$ be the last element of ζ if $n < \omega$ and the set of all elements of A which appear infinitely often in ζ if $n = \omega$.

Given a table $T = \langle S, M, S_0 \rangle$, a tape X over Σ, and a state $s \in S$, we define an *s-run* of T on X as any sequence $\zeta = (s_i)$ of length $1 + |X|$ of states such that $s_0 = s$ and $s_{i+1} \in M(s_i, x_i)$ for every $i < |X|$. The set of all s-runs of T on X is written as $R_s(T, X)$, and the set of runs of T on X is written as $R(T, X) = \bigcup_{s \in S_0} R_s(T, X)$.

The conventional language of a stuttering finite automaton $\mathcal{A} = \langle S, M, S_0, F \rangle$, denoted $\mathcal{L}_C(\mathcal{A})$, is defined as $\{X \in \Sigma^* \mid \exists \zeta \in R(T, X) \text{ such that } \zeta(*) \in F\}$.

When we define an automaton $\mathcal{A}$, we are interested not only in those runs that are conventionally accepted by $\mathcal{A}$, but also those runs whose stuttering variants are conventionally accepted by $\mathcal{A}$. That is, if w is an accepting run and $w' \succeq w$ or $w' \preceq w$, then w' should be an accepting run. We define the *downward closure* of an arbitrary set A, denoted $\downarrow A$, as the smallest set such that (1) $A \subseteq \downarrow A$ and (2) if $w \in \downarrow A$ then $\forall w' \preceq w$, $w' \in \downarrow A$. We define the *stuttering language* of an automaton $\mathcal{A}$, denoted $\mathcal{L}_S(\mathcal{A})$, as $\downarrow \mathcal{L}_C(\mathcal{A})$.

Lemma 7

$S = \{s_0, s_1, s_2\}$ $\quad S_0 = \{s_0\}$ $\quad F = \{s_1\}$

M	ψ	$\neg\psi$
s_0	s_1	s_2
s_1	s_1	s_2
s_2	s_2	s_2

Figure 4.1: The deterministic stuttering-invariant finite automaton for accepting ψ^+

1. *If $\mathcal{A}$ is deterministic, then $\mathcal{L}_S(\mathcal{A}) = \mathcal{L}_C(\mathcal{A})$.*
2. *For any stuttering-invariant automaton, $w \in \mathcal{L}_S(\mathcal{A})$ iff $(w)^{St(|\mathcal{A}|)} \in \mathcal{L}_C(\mathcal{A})$.*
3. *For any* SRE *E_1 and E_2, $SSat(E_1E_2) = \downarrow(SSat(E_1) \cdot SSat(E_2))$, where $\cdot$ is concatenation of sets.*

From now on we will use this lemma as an alternate definition for the stuttering language of automata. Let $\mathbf{DSA}(\Sigma)$ be the class of stuttering languages defined by deterministic stuttering-invariant finite automata. That is,

$$\mathbf{DSA}(\Sigma) = \{\mathcal{L}_S(\mathcal{A}) \mid \mathcal{A} \text{ is a stuttering-insensitive automaton over } \Sigma\}.$$

For example, the deterministic table in Figure 4.1 defines the language ψ^+ for some Σ-proposition ψ, and the nondeterministic table in Figure 4.2 defines the language $\psi^+ true^+$. To be precise, in Figure 4.1, we would take all $\sigma \in \Sigma$ such that $\sigma \models_C \psi$ and define $M(s_i, \sigma)$ as the value given for $M(s_i, \psi)$. That is, we take $M(s_i, \psi) = U$, for some set U, to be an abbreviation for

$$\forall \sigma \in \Sigma, M(s_i, \sigma) = U.$$

4.2.4 Closure of Automata

Our two main goals in this section are (1) to show that the theory of automata that we have previously defined is closed under the operations of union, product, intersection, and complementation; and, (2) to show that our stuttering-invariant theory of automata is equivalent to our stuttering-invariant theory of extended regular expressions. We begin with (1).

Theorem 8 (Closure) $\mathbf{DSA}(\Sigma)$ *is closed under complementation, union, intersection, product, and Kleene closure.*

$S = \{s_0, s_1, s_2, s_3\}$ $\qquad S_0 = \{s_0\}$ $\qquad F = \{s_3\}$

M	ψ	$\neg\psi$
s_0	s_1	s_2
s_1	$\{s_1, s_3\}$	$\{s_2, s_3\}$
s_2	s_2	s_2
s_3	s_3	s_3

Figure 4.2: The nondeterministic stuttering finite automaton for accepting $\psi^+ \mathit{true}^+$

To prove this theorem we will need a way to take the product of two deterministic stuttering-invariant automata such that the resulting automaton satisfies the stuttering condition and is deterministic. Note that the traditional construction for product (e.g., Hopcroft and Ullman [86]) results in a nondeterministic automaton, which can then be determinized using the subset construction. However, such a construction results in an automaton that violates the stuttering condition[4]. Instead we adapt the approach taken by Choueka in [44], where given a deterministic automaton $\mathcal{A}$ and an automaton $\mathcal{B} = \langle T, F' \rangle$ we construct a new table T' such that $\langle T', F' \rangle$ accepts the language $\mathcal{L}(\mathcal{A}) \times \mathcal{L}(\mathcal{B})$.

Flag Construction

The flag construction takes as input a deterministic stuttering-insensitive automaton $\mathcal{A}$ and a (nondeterministic) stuttering insensitive table T' and constructs a new table, called the *flag-table of* $\mathcal{A}$ *relative to* T', $\mathrm{fl}(\mathcal{A}, T')$, which also satisfies the stuttering condition.

The intuition behind the flag construction is taken verbatim from Choeuka [44].

> Take $n+2$ copies of T' [where $n = |T'|$], number them 1 through $n+2$, connect them in parallel with $\mathcal{A}$, and add a control unit to the resulting structure. Each copy of T' can be either *dormant*, which means that it ignores the input until it is switched "on" by a "control unit" C. Once it is switched "on", it remains *active* and acts according to table T' until it is switched "off" by C.
>
> The configuration works as follows. In the initial state of the flag table, $\mathcal{A}$ is in its initial state, and all T' copies are dormant. At each time t, C checks for all T' copies which are in the same state, and switches them off, except for the one with the least index. It then checks $\mathcal{A}$'s state to see whether it is a final state; if so, it switches on one of the dormant copies of T'.

[4]Furthermore, it does not generalize to infinite automata.

Suppose that at some time t, n of T' copies are active and in different states, and that, further, C switches on then one of the dormant copies. At $t+1$, there will be only $n+1$ active copies, so that one copy is still available for switching on by C if necessary.

We now formally construct the flag table. Given a deterministic stuttering-invariant automaton $\mathcal{A} = \langle S, M, \{s_0\}, F\rangle$ and a stuttering-invariant table $T' = \langle S', M', S'_0\rangle$ over Σ where $|S'| = n$. We define the *flag-table of $\mathcal{A}$ relative to T'* as the table $T'' = \text{fl}(\mathcal{A}, T') = \langle Q, P, r_0\rangle$ where

- $S'' = S' \cup \{0\}$ where $0 \notin S'$ (0 represents the dormant state)
- M'' is just like M' extended on the new state 0 as follows: $M''(0,\sigma) = \{0\}$, for all $\sigma \in \Sigma$.
- $G = (S'')^k - (S')^k$ where $k = n+2$ (thus G is the set of $n+2$ tuples with at least one occurrence of 0).
- $Q = S \times \Sigma \times \{0, 1, \ldots, k\} \times G$. The S position represents the state $\mathcal{A}$ is in, the Σ position represents the last input character, the $\{0, 1, \ldots, k\}$ position represents the copy of T' that was turned on by the last input or 0 if no copy was turned on, and G represents the states that the $n+2$ copies of T' are in. These latter two positions will help us satisfy the stuttering condition.
- $r_0 = (s_0, \sigma, 0, \underbrace{0, \ldots, 0}_{n+2 \text{ times}})$ where σ is the first symbol appearing in the set Σ.
  We could pick any σ, but for concreteness we have chosen the first σ appearing in Σ.
- In the following, let g and g'' be a $k = n+2$ tuple of integers, g_a, g_b, and g_j represent the a, b, and j^{th} position of g, respectively, g''_m and g''_j represent the m and j^{th} position of g'', respectively, and $\gamma, \sigma \in \Sigma$. We define M'' as follows:
  1. $P(\langle s, \sigma, 0, g\rangle, \gamma) = \{\langle M(s,\gamma), \gamma, a, g'\rangle\}$ where $\forall j \in [1..k]$, $g''_j \in M''(g_j, \gamma)$.

     (OFF-condition) $g'_j = g''_j$ unless $\exists m < j$ such that $g''_j = g''_m$ whence $g'_j = 0$.
     This statement says that the OFF condition does nothing unless there is a position of g'' smaller than j that is in the same state as g''_j, in which case we shut j down.

     (ON-condition)
     ```
     if $M(s,\gamma) \in F$ then
         let $a$ be the least index for which $g_a = 0$
         $g'_a \in \bigcup_{t \in S'_0} M''(t,\gamma)$
     else $a = 0$
     ```

     Notice that when we turn on a copy of T', we set the third position of our tuple to the number of the copy turned on and assume that our next input will also be γ. This ensures that if our next input is indeed γ, the stuttering condition is satisfied. If our next input is not γ, then we will still be able to correctly put the a^{th} copy of T' into its correct state since we have recorded the value of a (see also condition (3)).

2. $P(\langle s, \sigma, a \neq 0, g\rangle, \sigma) = \{\langle M(s,\sigma), \sigma, b, g'\rangle\}$ where $\forall j \in [1..k]$, $g''_j \in M''(g_j, \sigma)$. If $g''_a = g_a$ (i.e., our a^{th} copy of T' has stuttered into the same state) then $b = a$, else $b = 0$.

   (OFF-condition) $g'_j = g''_j$ unless $\exists m < j$ such that $g''_j = g''_m$ whence $g'_j = 0$; moreover if $j = a$ then $b = 0$.

   (ON-condition) None

   This is where we must ensure that the stuttering condition is satisfied: σ was the last input as well as the current input. Since $\mathcal{A}$ is deterministic and s is the state resulting from input σ, by the stuttering condition $M(s,\sigma) = s$. As the requirements for turning on a copy are subsumed by the stuttering condition, we have no ON condition.

3. $P(\langle s, \gamma, a \neq 0, g\rangle, \gamma \neq \sigma) = \{\langle M(s,\gamma), \gamma, b, g'\rangle\}$ where $\forall j \in [1..k]$, $j \neq a$, $g''_j \in M''(g_j, \sigma)$; $g''_a \in \bigcup_{t \in S'_0} M(t, \gamma)$.

   (OFF-condition) $g'_j = g''_j$ unless $\exists m < j$ such that $g''_j = g''_m$ whence $g'_j = 0$.

   (ON-condition)
   ```
   if $M(s,\gamma) \in F$ then
       let $b$ be the least index for which $g_b = 0$
       $g'_b \in \bigcup_{t \in S'*} M''(t,\gamma)$
   else $b = 0$
   ```

   In this case since our current input is different from the last input, our stuttering condition is trivially satisfied. However, if we turned on any copy of T' (which is represented by $a \neq 0$), we must correct the state of this a^{th} copy of T' (see the note regarding the first case).

The following terminology is borrowed from [44]. Let X be a tape over Σ and $\zeta'' \in R(T'', X)$, where $T'' = \text{fl}(\mathcal{A}, T')$. Given some $i < |X|$, let $\zeta''(i-1) = (s, \sigma, a, g)$ and $\zeta''(i) = (s', \gamma, b, g')$ We say that copy j is *switched on* at i if $g_j = 0$, $g'_j \neq 0$; *switched off* if $g_j \neq 0$, $g'_j = 0$; *dormant* if $g_j = g'_j = 0$; *active* if $g_j \neq 0$, $g'_j \neq 0$. If copy j is switched off at i and j' is the index of the unique copy which is active at i and is in the same state as copy j, then we say that j has been switched off *because* of j'.

Suppose that copy j_0 has been switched on at i_0. Its *representative* at $i > i_0$, relative to i_0, $\text{rep}(j_0, i_0, i)$, is defined as follows: $\text{rep}(j_0, i_0, i_0 + 1) = j_0$, and $\text{rep}(j_0, i_0, i + 1) = \text{rep}(j_0, i_0, i + 1) = \text{rep}(j_0, i_0, i)$ unless this right-hand copy has been switched off at i because of j' in which case $\text{rep}(j_0, i_0, i + 1) = j'$. Since $\text{rep}(j_0, i_0, i)$ is a nonincreasing sequence of numbers bounded by j_0, there is some $j^* \leq j_0$ such that ultimately $\text{rep}(j_0, i_0, i)$ is constant and equal to j^*. We call j^* the *ultimate representative of* j_0 *relative to* i_0 and denote it by $\text{urep}(j_0, i_0)$. We define the *virtual run of* j_0 *relative to* i_0 as the sequence $v(j_0, i_0, \zeta'') = \zeta'$ defined by $\zeta'(0) = s'$ where s' is any state for which $p_{j_0}(\zeta''(i_0 + 1)) \in M(s', x_{i_0})$, and for $i > 0$, $\zeta'(i) = p_{j'}(\zeta''(i + i_0))$, where $j' = \text{rep}(j_0, i_0, i + i_0)$.

Lemma 9 (Flag) *Let $\mathcal{A}$ be a deterministic stuttering finite automaton, T' a stuttering-invariant table, $T'' = \text{fl}(\mathcal{A}, T')$ and X a Σ-word. If $\zeta'' \in R(T'', X)$, copy j_0 is switched on at i_0, $Y = X(0, i_0)$*

and $Z = X(i_0, |X|)$, *then* Y *is accepted by* $\mathcal{A}$, $\zeta' = v(j_0, i_0, \zeta'') \in R(T', Z)$, *and* $\zeta'(*) = p_{j^*}(\zeta''(*))$. *On the other hand if* $X = YZ$ *where* $Y = X(0, i_0)$ *is accepted by* $\mathcal{A}$, *then for every* $\zeta' \in R(T', Z)$ *there is some* $\zeta'' \in R(T'', X)$ *and some* $1 \le j_0 \le k$ *such that* $\zeta' = v(j_0, i_0, \zeta'')$ *and* $\zeta'(*) = p_{j^*}(\zeta''(*))$. *In both cases,* $j^* = \mathrm{urep}(j_0, i_0)$.

By Lemma 9, we can construct the product of two deterministic stuttering-insensitive automata $\mathcal{A}$ and $\mathcal{B}$ as follows: $\mathcal{C} = \langle \mathrm{fl}(\mathcal{A}, T'), F'' \rangle$ where $F'' = \{r | \text{for some } 1 \le j \le n+2, p_j(r) \in F'\}$; if $\Lambda \in \mathcal{L}_S(\mathcal{B})$ then $\mathcal{C}$ is the desired product automaton, else $\mathcal{C} \cup \mathcal{A}$ is the desired product automaton. Given two deterministic stuttering-insensitive automata $\mathcal{A}$ and $\mathcal{B}$, their union is $\langle T \times T', \{f | p_1(f) \in F \text{ or } p_2(f) \in F\} \rangle$, and their intersection is $\langle T \times T', \{f | p_1(f) \in F \text{ and } p_2(f) \in F\} \rangle$. The complement of an automaton $\mathcal{A} = \langle S, M, s_0, F \rangle$ is $\langle S, M, s_0, S - F \rangle$. With this lemma in hand we can prove Theorem 8. The proof appears in Chapter 4.6.

We now show that our automata theory is equivalent to the expression theory developed in Chapter 4.2.1.

Theorem 10 (Equivalence) $\mathbf{DSA}(\Sigma)$=$\mathbf{SRE}(\Sigma)$.

Theorem 10 is proved in two stages: (1) showing $\mathbf{SRE}(\Sigma) \subseteq \mathbf{DSA}(\Sigma)$; and (2) showing $\mathbf{DSA}(\Sigma) \subseteq \mathbf{SRE}(\Sigma)$.

To prove $\mathbf{SRE}(\Sigma) \subseteq \mathbf{DSA}(\Sigma)$, note that the basic SRE ψ^+ can be defined using stuttering-insensitive automata as in Figure 4.1. As $\mathbf{DSA}(\Sigma)$ is closed under ordinary negation, ordinary union, and ordinary Kleene closure, all we need to show is that $\mathbf{DSA}(\Sigma)$ is closed under SRE's definition of concatenation. This fact can be shown using Lemma 7 as follows: let $\mathcal{A}$ and $\mathcal{B}$ be two deterministic stuttering-insensitive automata and let E_1 and E_2 be two SREs such that $\mathcal{L}_S(\mathcal{A}) = SSat(E_1)$ and $\mathcal{L}_S(\mathcal{B}) = SSat(E_2)$, then $\mathcal{L}_S(\mathcal{A} \times \mathcal{B}) = \downarrow(\mathcal{L}_S(\mathcal{A}) \cdot \mathcal{L}_S(\mathcal{B})) = \downarrow(SSat(E_1) \cdot SSat(E_2)) = SSat(E_1 E_2)$.

To prove $\mathbf{DSA}(\Sigma) \subseteq \mathbf{SRE}(\Sigma)$, we show how to construct an SRE E given an arbitrary deterministic stuttering-insensitive automaton $\mathcal{A} = \langle S, M, s_0, F \rangle$, where $S = \{s_0, \ldots, s_n\}$. For $1 \le i, j \le n$ and $0 \le k \le n$ define the following sets:

$$\begin{aligned} V^0_{i,j} &= \{X^+ | M(s_i, X) = s_j, |X| = 1\} && \text{if } i \ne j \\ V^0_{i,j} &= \{X^+ | M(s_i, X) = s_j, |X| = 1\}^* && \text{if } i = j \\ V^{k+1}_{i,j} &= V^k_{i,j} + V^k_{i,k+1}(V^k_{k+1,k+1})^* V^k_{k+1,j} \end{aligned}$$

We can show $SSat(V^k_{i,j}) = \mathcal{L}_S(\mathcal{A}^k_{i,j})$, where $\mathcal{A}^k_{i,j} = \langle S_k, M \text{ restricted to } S_k, s_i, s_j \rangle$ and $S_k = \{s_1, \ldots, s_k\} \cup \{s_i, s_j\}$. Setting $V = \bigcup_{s_i \in F} V^n_{i,i}$ gives us $SSat(V) = \mathcal{L}_S(\mathcal{A})$ as desired.

Using the classical techniques of automata [86], we can decide the language emptiness and language inclusion problem for $\mathbf{DSA}(\Sigma)$.

Theorem 11 *Given two deterministic stuttering-insensitive automata* $\mathcal{A}$ *and* $\mathcal{B}$, *we can decide if* $\mathcal{L}_S(\mathcal{A}) = \emptyset$ *and* $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$.

4.3 From Prop-HTL to Stuttering Automata

In this section, we use our theory of stuttering invariance to answer the following questions about finite models of propositional-HTL.

- Given a finite phase P over V and an HTL formula φ, does $P \models \varphi$?
- Given an HTL formula φ over V, is there a finite phase P such that $P \models \varphi$? That is, is φ a satisfiable formula of propositional HTL restricted to finite models?

To answer these questions, we define a translation from HTL to SRE.

The ability to refer to time is present in HTL because of its ability to refer to derivatives, which can express, among other things, clocks and stop-watches. It is important to note that the propositional fragment of HTL is essentially an untimed logic. Thus, the following two finite models of propositional HTL are equivalent in the sense that each set satisfies precisely the same HTL formulas.

- $P_1 = \langle I^1, f^1 \rangle$ where $I^1 = [0, 10)$ and (for each $x_i \in V$) $f^1_{x_i}(t) = \mathit{true}$ for all $t \in [0, 5)$, false otherwise.
- $P_2 = \langle I^2, f^2 \rangle$ where $I^2 = [0, 30)$ and (for each $x_i \in V$) $f^2_{x_i}(t) = \mathit{true}$ for all $t \in [0, 12)$, false otherwise.

We formalize this property below.

We call a phase $P = \langle I, f \rangle$ (over $V = \{x_1, \ldots, x_n\}$) a *constant-slope* phase if for all $x_i \in V$, either for all $t \in I$, $f_{x_i}(t) = \mathit{true}$ or for all $t \in I$, $f_{x_i}(t) = \mathit{false}$. That is, for each x_i, f_{x_i} is constant. For a constant-slope phase we use $f_{x_i}(I)$ to denote the value of variable x_i throughout the phase.

Two phases P_1 and P_2 are *propositionally stuttering equivalent* (denoted $P_1 =_s P_2$) if there exist constant slope phases $P^{1,1} = \langle I^{1,1}, f^{1,1} \rangle, \ldots, P^{1,a} = \langle I^{1,a}, f^{1,a} \rangle$ and constant slope phases $P^{2,1} = \langle I^{2,1}, f^{2,1} \rangle, \ldots, P^{2,a} = \langle I^{2,a}, f^{2,a} \rangle$ such that

- $P_1 = P^{1,1} \frown \cdots \frown P^{1,a}$ and
- $P_2 = P^{2,1} \frown \cdots \frown P^{2,a}$ and
- for all $i \in [1..a]$, for all $x_j \in V$, $f^{1,i}_{x_j}(I^{1,i}) = f^{2,i}_{x_j}(I^{2,i})$

Note that if two finite phase sequences are equivalent (as defined by Henzinger et al [70]) then when each is viewed as a single phase restricted to the propositional variables, the resulting two phases are propositionally stuttering equivalent. However, in general, the converse is not true. In addition, note that the precise times of the intervals are ignored when determining propositional stuttering equivalence.

The following lemma, whose proof appears in Chapter 4.6, formalizes the fact that propositionally stuttering-equivalent phases satisfy the same formulas and thus are satisfaction indistinguishable.

Lemma 12 (Satisfaction Indistinguishability) *For any propositional* HTL *formula φ, for any two finite phases P_1 and P_2, if $P_1 =_s P_2$ then*

$$P_1 \models \varphi \qquad \textit{iff} \qquad P_2 \models \varphi.$$

In the following, fix a finite vocabulary $V = \{x_1, \ldots, x_n\}$ of propositional variables. We define two translation functions: T_f, which is a one-to-one function that translates an arbitrary HTL formula (over V) into a SRE (over V), and T_m, which is a surjective function that translates an arbitrary finite phase of propositional HTL (over V) into a Σ-word w, where Σ is the set of all Σ-states over V. Let T_f^- be the inverse of the one-to-one function T_f, defined for all star-free SRE. In addition, we define a reverse translation T_m^r that translates an arbitrary Σ-word into a phase such that

- For any Σ-word w, $T_m(T_m^r(w)) = w$
- For any phase P, $P' = T_m^r(T_m(P))$ where $P' =_s P$.

Our two basic lemmas are:

Lemma 13 (Soundness) *For any* HTL *formula φ and any phase P,*

$$P \models \varphi \qquad \textit{iff} \qquad T_m(P) \models_S T_f(\varphi).$$

Lemma 14 (Completeness) *For any star-free* SRE *E and any Σ-word w,*

$$w \models_S E \qquad \textit{iff} \qquad T_m^r(w) \models T_f^-(E).$$

We now define the actual translation functions.

Definition: T_f: HTL $\rightarrow$ SRE

$T_f(\psi) = \psi^+ \mathit{true}^+$ for any boolean proposition ψ

$T_f(\varphi_1 \vee \varphi_2) = T_f(\varphi_1) + T_f(\varphi_2)$

$T_f(\varphi_1 \wedge \varphi_2) = T_f(\varphi_1) \cap T_f(\varphi_2)$

$T_f(\varphi_1 ; \varphi_2) = T_f(\varphi_1) T_f(\varphi_2)$

$T_f(\neg\varphi_1) = \neg T_f(\varphi_1)$

Definition: T_m: Phases $\rightarrow$ Σ-words

Any finite phase P of propositional HTL can be partitioned into constant-slope subphases $P_1, \ldots, P_\ell$. In addition, there is a unique maximal such partition, such that for no $i \in [1..\ell-1]$ do we have $\forall x_j \in V$, $f^i_{x_j}(I^i) = f^{i+1}_{x_j}(I^{i+1})$. We call this partition the *maximal constant-slope partition.*

We define $T_m(P) = \sigma_1 \ldots \sigma_\ell$, where (for each $i \in [1..\ell]$)

$$\sigma_i = (\{x_j \mid f^i_{x_j}(I^i) = \mathit{true}\} \cup \{\overline{x_j} \mid f^i_{x_j}(I^i) = \mathit{false}\}).$$

Definition: T_m^r: Σ-words $\rightarrow$ Phases
For $w = \sigma_1 \ldots \sigma_\ell$, we define $P_i = \langle I^i, f^i \rangle$ as follows:
$I^i = [i-1, i)$
$f^i_{x_j} = true$ iff $\sigma_i \models x_j$
$f^i_{x_j} = false$ iff $\sigma_i \models \overline{x_j}$
$T_m^r(w) = P_1 {}^\frown \cdots {}^\frown P_\ell$.

The soundness and completeness proofs can be found in Chapter 4.6. Thus, questions of satisfiability and validity of HTL formulas can be answered by translating the formulas into the theory of SRE and using the decision procedures established for the theory of SRE. In particular, given an SRE E to determine if E is satisfiable, check if the deterministic stuttering automaton for $T_f(E)$ is empty: if the automaton is empty, then E is not satisfiable, otherwise it is satisfiable. To determine if E is valid check if the deterministic stuttering automaton for $T_f(\neg E)$ is empty: if the automaton is empty, then E is valid, otherwise it is not valid.

4.4 Application to Verification

The primary application of stuttering regular expressions and stuterring automata to verification is in determining validity and satisfiability questions of HTL formulas. We will see in Chapter 8 that determining the validity of HTL formulas will be necessary for determining when an HTL property holds for a particular hybrid system. In this chapter, we have only dealt with the propositional fragment of HTL. In Chapter 8.6, we present an additional method for proving HTL properties.

4.5 Summary

Much work has been done in the area of regular expressions and automata. For an overview of automata and regular expressions, we refer the reader to Hopcroft and Ullman [86]. Stuttering closure has been studied for LTL by Peled, Wilke, and Wolper [130]. Choueka [44] was the first to give a flag construction for computing the product of two automata. Our construction is modified so that the resulting flag table also obeys the stuttering condition. Our decision procedures for **DSA**(Σ) and **SRE**(Σ) are based on the classical decision procedures for automata and regular expressions, which are presented in [86] and Stockmeyer's thesis [147].

The downward closure of a set is used by Chaochen, Hansen, and Sestoft in [40] to prove that the propositional fragment of the duration calculus (CoD) is decidable. The logics CoD and HTL are similar in that both are used for the specification and verification of hybrid systems, and that both have propositional fragments that are insensitive to stuttering. However, the two logics are different in other respects. For example, HTL is a local logic[5], whereas CoD is a non-local logic. Among the

[5]A logic is local if each propositional variable p is true of an interval $s_0 \ldots s_n$ iff p is true of the first state s_0.

interval temporal logics, Ramakrishna et al's Future Interval Temporal Logic [136] is insensitive to stuttering. However, it differs from our work here in that it does not have a chop operator.

Note that although the logics COD and HTL are defined on continuous, dense time domains, their propositional fragments are discretizable. Interval logics that do not suffer from non-elementariness do so by eliminating the chop operator (e.g., [136]). In fact, it is the interplay of negation and chop that result in the non-elementariness of these logics [147, 60, 136]. In practice however, the nesting of negations and chop operators can be controlled (e.g., typical safety properties are of the form $\Box\,\varphi_1; \Box\,\varphi_2; \Box\,\varphi_3$ where each φ_i is chop-free). Thus, this is not as severe a limitation as the theoretical results suggest.

4.6 Proofs

In this section we give the proofs of several Chapter 4 theorems. We restate the theorems for convenience.

Proposition 4 *For any Σ-proposition ψ, for any Σ-state σ, either $\sigma \models_C \psi$ or $\sigma \models_C \neg\psi$.*

Proof of 4:

Consider an arbitrary Σ-proposition ψ and an arbitrary Σ-state σ.

Case:
If $\sigma \models_C \psi$ then we are done.

Case:
If $\sigma \not\models_C \psi$ then $\sigma \not\models \psi$, by definition of $\models_C$.
So $\sigma \models \neg\psi$, by definition of $\models$. So $\sigma \models_C \neg\psi$.

Thus, either $\sigma \models_C \psi$ or $\sigma \models_C \neg\psi$. ∎

Lemma 5 (Stuttering Invariance) *For any SRE E, for any Σ-word w,*

1. *if $w \models_S E$ then for all $w' \succeq w$, $w' \models_S E$.*
2. *if $\exists\ w' \succeq w$ such that $w' \models_S E$, then $w \models_S E$.*

Proof of 5:

We proceed by induction on the structure of E.

Case: Base
Consider an arbitrary SRE E of the form ψ^+ and an arbitrary Σ-word $w = \sigma_1 \ldots \sigma_n$.

1. Suppose $w \models_S E$. Then by definition of $\models_S$, $w \models_C \psi^+$. Consider any arbitrary Σ-word $v \succeq w$. As $w \models_C \psi^+$, $\forall i \in [1..n]$, $\sigma_i \models_C \psi$. As $v = \sigma_1^{i_1} \ldots \sigma_n^{i_n}$ (for some $i_1, \ldots, i_n$), we have $v \models_C \psi^+$. So $v \models_S \psi^+$.
2. Suppose $\exists\ w' \succeq w$ such that $w' \models_S E$. Fix such a w'. Thus, $w' = \sigma_1^{i_1} \ldots \sigma_n^{i_n}$ for some $i_1, \ldots, i_n \geq 1$. Fix such $i_1, \ldots, i_n$. Then, by definition of $\models_S$, $w' \models_C \psi^+$. So $\forall i \in [1..n]$, $\sigma_i \models \psi$. So $w = \sigma_1 \ldots \sigma_n \models_C \psi^+$, whence $w \models_S \psi^+$.

Case: Inductive

Case: $E = E_1 + E_2$

Consider an arbitrary Σ-word w.

1. Suppose $w \models_S E_1 + E_2$. By definition of $\models_S$ for $+$, either $w \models_S E_1$ or $w \models_S E_2$.

   Case: $w \models_S E_1$

   Consider any $w'' \succeq w$. As $w \models_S E_1$, by part one of the induction hypothesis, $\forall w' \succeq w$, $w' \models_S E_1$. In particular $w'' \models_S E_1$. So $w'' \models_S E_1 + E_2$. As w'' was arbitrarily chosen, we get $\forall w'' \succeq w$, $w'' \models_S E_1 + E_2$ as desired.

   Case: $w \models_S E_2$

   Similarly done.
2. Suppose $\exists w' \succeq w$ such that $w' \models_S E_1 + E_2$. Fix such w'. Either $w' \models_S E_1$ or $w' \models_S E_2$.

   Case: $w' \models_S E_1$

   By induction hypothesis, $w \models_S E_1$, whence $w \models_S E_1 + E_2$.

   Case: $w' \models_S E_2$

   By induction hypothesis, $w \models_S E_2$, whence $w \models_S E_1 + E_2$.

Case: $E = \neg E_1$

Consider an arbitrary Σ-word w.

1. Suppose $w \models_S \neg E_1$. Assume $\exists w' \succeq w$ such that $w' \not\models_S \neg E_1$. Fix such a w'. Thus, $w' \models_S E_1$. By part two of the induction hypothesis applied to E_1, we get $w \models_S E_1$. $\Rightarrow\!\Leftarrow$

   So our assumption is false, and $\forall w' \succeq w$, $w' \models_S \neg E_1$ as desired.
2. Suppose $\exists w' \succeq w$ such that $w' \models_S \neg E_1$. Fix such w'. Assume $w \not\models_S \neg E_1$. Thus, $w \models_S E_1$ and by part one of the induction hypothesis, $w' \models_S E_1$. $\Rightarrow\!\Leftarrow$

   So our assumption is false, and $w \models_S \neg E_1$ as desired.

Case: $E = E_1 E_2$

Consider an arbitrary Σ-word w.

1. Thus, $w = \sigma_1 \ldots \sigma_n$ for some Σ-states $\sigma_1, \ldots, \sigma_n$. Fix such σ's. Suppose $w \models_S E_1 E_2$. So by definition of $\models_S$ for concatenation, there exist Σ-words v_1 and v_2 such that $(w)^{St(2)} = v_1 v_2$, $v_1 \models_S E_1$, and $v_2 \models_S E_2$. Fix such v_1 and v_2.

Thus $v_1 = \sigma_1^2 \ldots \sigma_{k-1}^2 \sigma_k^\ell$ and $v_2 = \sigma_k^{2-\ell}\sigma_{k+1}^2 \ldots \sigma_n^{i_n}$ for some ℓ and k such that $1 \leq k \leq n$, $0 \leq \ell \leq 2$, $|v_1| \geq 1$, and $|v_2| \geq 2$. Fix such k and ℓ.

Consider any $w' = \sigma_1^{j_1} \ldots \sigma_n^{j_n} \succeq w$. To show: $w' \models_S E_1E_2$.

Let $v_1' = \begin{cases} \sigma_1^{2j_1} \ldots \sigma_{k-1}^{2j_{k-1}} & \text{if } \ell = 0 \\ \sigma_1^{2j_1} \ldots \sigma_{k-1}^{2j_{k-1}} \sigma_k^{j_k} & \text{if } \ell = 1 \\ \sigma_1^{2j_1} \ldots \sigma_k^{2j_k} & \text{if } \ell = 2 \end{cases}$

Let $v_2' = \begin{cases} \sigma_k^{2j_k} \ldots \sigma_n^{2j_n} & \text{if } \ell = 0 \\ \sigma_k^{j_k} \sigma_{k+1}^{2j_{k+1}} \ldots \sigma_n^{2j_n} & \text{if } \ell = 1 \\ \sigma_{k+1}^{2j_{k+1}} \ldots \sigma_n^{2j_n} & \text{if } \ell = 2 \end{cases}$

Claim: $v_1' \models_S E_1$ and $v_2' \models_S E_2$

Proof:

As $v_1' \succeq v_1$ and $v_1 \models_S E_1$, by part one of the induction hypothesis, $v_1' \models_S E_1$. Similarly $v_2' \models_S E_2$. ∎

So $(w')^{St(2)} = v_1'v_2'$ is such that $(w')^{St(2)} = v_1'v_2'$, $v_1' \models_S E_1$, and $v_2' \models_S E_2$. So by definition of $\models_S$ for concatenation, $w' \models_S E_1E_2$ as desired.

2. Suppose $\exists w' \succeq w$ such that $w' \models_S E_1E_2$. Fix such w'. Thus $w' = \sigma_1^{i_1} \ldots \sigma_n^{i_n}$, for some $i_1, \ldots, i_n \geq 1$. Fix such $i_1, \ldots, i_n$. As $w' \models_S E_1E_2$, by definition of $\models_S$ for concatenation, there exists v_1' and v_2' such that $(w')^{St(2)} = v_1'v_2'$, $v_1' \models_S E_1$, and $v_2' \models_S E_2$. Fix such v_1' and v_2'. So $v_1' = \sigma_1^{2i_1} \ldots \sigma_{k-1}^{2i_{k-1}} \sigma_k^\ell$ and $v_2' = \sigma_k^{2i_k - \ell} \ldots \sigma_{k+1}^{2i_k+1} \sigma_n^{i_n}$, for some ℓ and k such that $1 \leq k \leq n$, $0 \leq \ell \leq 2i_k$, $|v_1'| \geq 1$, and $|v_2'| \geq 2$. Fix such k and ℓ.

Let $v_1 = \begin{cases} \sigma_1^2 \ldots \sigma_{k-1}^2 & \text{if } \ell = 0 \\ \sigma_1^2 \ldots \sigma_{k-1}^2 \sigma_k & \text{if } 1 \leq \ell < 2i_k \\ \sigma_1^2 \ldots \sigma_k^2 & \text{if } \ell = 2i_k \end{cases}$

Let $v_2' = \begin{cases} \sigma_k^2 \ldots \sigma_n^2 & \text{if } \ell = 0 \\ \sigma_k \sigma_{k+1}^2 \ldots \sigma_n^2 & \text{if } 1 \leq \ell < 2i_k \\ \sigma_{k+1}^2 \ldots \sigma_n^2 & \text{if } \ell = 2i_k \end{cases}$

Then $v_1' \succeq v_1$ and $v_2' \succeq v_2$, so by part two of the induction hypothesis, $v_1 \models_S E_1$ and $v_2 \models_S E_2$. As $(w)^{St(2)} = v_1v_2$, we get $w \models_S E_1E_2$ as desired.

Case: $E = E_1^*$

Consider an arbitrary Σ-word w.

1. Suppose $w \models_S E_1^*$. By definition of $\models_S$ for *, either $w = \Lambda$ or $w \models_S E_1^i$ for some $i \geq 1$.

Case: $w = \Lambda$

Then trivially, $\forall w'' \succeq w$, $w'' \models_S E_1^*$ as desired.

Case: $w \models_S E_1^i$, for some $i \geq 1$

Fix such an i. Consider any $w'' \succeq w$. Then by induction hypothesis, $\forall w' \succeq w$, $w' \models_S E_1^i$. In particular $w'' \models_S E_1^i$. So $w'' \models_S E_1^*$. As w'' was arbitrarily chosen, we get $\forall w'' \succeq w$, $w'' \models_S E_1^*$ as desired.

2. Suppose $\exists w' \succeq w$ such that $w' \models_S E_1^*$. Fix such w'. As $w' \neq \Lambda$, $w' \models_S E_1^i$, for some $i \geq 1$. Fix such an i. So by induction hypothesis, $w \models_S E_1^i$. So $w \models_S E_1^*$ as desired.

So by induction the lemma holds. ∎

Proof of Correctness of Algorithm SSat:

We show

$$x_{ij} \in SSat(E_k) \qquad \text{iff} \qquad \text{entry } A_{ijk} = \text{“yes”}.$$

Consider an arbitrary word $w = \sigma_1 \ldots \sigma_n$ and SRE E. We proceed by induction on the structure of E.

Case: Base

E_k is of the form ψ^+.

$x_{ij} \in SSat(E_k)$ iff $\forall \ell,\ i \leq \ell \leq j,\ \sigma_\ell \models_C \psi$
iff $A_{ijk} =$“yes”

Case: Inductive

Case: $E_k = E_{k_1} + E_{k_2}$, for $k_1, k_2 < k$

$x_{ij} \in SSat(E_k)$ iff $x_{ij} \in SSat(E_{k_1})$ or $x_{ij} \in SSat(E_{k_2})$
iff $A_{ijk_1} =$“yes” or $A_{ijk_2} =$“yes”
iff $A_{ijk} =$“yes”

Case: $E_k = E_{k_1} \cap E_{k_2}$, for $k_1, k_2 < k$

$x_{ij} \in SSat(E_k)$ iff $x_{ij} \in SSat(E_{k_1})$ and $x_{ij} \in SSat(E_{k_2})$
iff $A_{ijk_1} =$“yes” and $A_{ijk_2} =$“yes”
iff $A_{ijk} =$“yes”

Case: $E_k = \neg E_{k_1}$, for $k_1 < k$

$x_{ij} \in SSat(E_k)$ iff $x_{ij} \notin SSat(E_{k_1})$
iff $A_{ijk_1} =$“no”
iff $A_{ijk} =$“yes”

Case: $E_k = E_{k_1} E_{k_2}$, for $k_1, k_2 < k$

Case: $\Rightarrow$

Suppose $x_{ij} \in SSat(E_k)$. Then $\exists v_1, v_2$ such that $(x_{ij})^{St(2)} = v_1 v_2$, $v_1 \in SSat(E_{k_1})$, and $v_2 \in SSat(E_{k_2})$. Fix such v_1 and v_2.

Case:

For some ℓ, $v_1 = \sigma_i^2 \dots \sigma_{i+\ell-1}^2$ and $v_2 = \sigma_{i+\ell}^2 \dots \sigma_{i+j-1}^2$. Then $v_1 \succeq x_{i\ell}$ and $v_2 \succeq x_{(i+\ell)(j-\ell)}$. So by part two of Lemma 5, as $v_1 \in SSat(E_{k_1})$ and $v_2 \in SSat(E_{k_2})$, we get $x_{i\ell} \in SSat(E_{k_1})$ and $x_{(i+\ell)(j-\ell)} \in SSat(E_{k_2})$. Thus, by induction hypothesis, $A_{i\ell k_1} =$"yes" and $A_{(i+\ell)(j-\ell)k_2} =$"yes", whence $A_{ijk} =$"yes".

Case:

$$v_1 = \begin{cases} \sigma_i & \text{if } \ell = 1 \\ \sigma_i^2 \dots \sigma_{i+\ell-2}^2 \sigma_{i+\ell-1} & \text{for some } \ell,\ 1 < \ell \leq j \end{cases}$$
$$v_2 = \begin{cases} \sigma_{i+\ell-1}\sigma_{i+\ell}^2 \dots \sigma_{i+j-1}^2 & \text{for some } \ell,\ 1 \leq \ell < j \\ \sigma_{i+j-1} & \text{if } \ell = j \end{cases}$$

Then $v_1 \succeq x_{i\ell}$ and $v_2 \succeq x_{(i+\ell)(j-\ell)}$. So by part two of Lemma 5, as $v_1 \in SSat(E_{k_1})$ and $v_2 \in SSat(E_{k_2})$, we get $x_{i\ell} \in SSat(E_{k_1})$ and $x_{(i+\ell)(j-\ell)} \in SSat(E_{k_2})$. Thus, by induction hypothesis, $A_{i\ell k_1} =$"yes" and $A_{(i+\ell)(j-\ell)k_2} =$"yes", whence $A_{ijk} =$"yes".

So in either case, $A_{ijk} =$"yes".

Case: $\Leftarrow$

Suppose $A_{ijk} =$"yes". So either

- $\exists \ell$, $1 \leq \ell < j$ such that $A_{i\ell k_1}=$"yes" and $A_{(i+\ell)(j-\ell)k_2}=$"yes";
- or $\exists \ell$, $1 \leq \ell \leq j$, such that $A_{i\ell k_1}=$"yes" and $A_{(i+\ell-1)(j-\ell+1)k_2}=$"yes";
- or $\Lambda \in E_{k_1}$ and $A_{ijk_2}=$"yes";
- or $\Lambda \in E_{k_2}$ and $A_{ijk_1}=$"yes".

Case:

Suppose there exists $1 \leq \ell < j$ such that $A_{i\ell k_1}=$"yes" and $A_{(i+\ell)(j-\ell)k_2}=$ "yes". Fix such an ℓ. By the induction hypothesis, $x_{i\ell} \in SSat(E_{k_1})$ and $x_{(i+\ell)(j-\ell)} \in SSat(E_{k_2})$. By Lemma 5 part one, $(x_{i\ell})^{St(2)} \in SSat(E_{k_1})$ and $(x_{(i+\ell)(j-\ell)})^{St(2)} \in SSat(E_{k_2})$. As $(x_{ij})^{St(2)} = (x_{i\ell})^{St(2)}(x_{(i+\ell)(j-\ell)})^{St(2)}$, we get $x_{ij} \in SSat(E_k)$ as desired.

Case:

Suppose there exists $1 \leq \ell \leq j$ such that $A_{i\ell k_1}=$"yes" and $A_{(i+\ell-1)(j-\ell+1)k_2}=$ "yes". Fix such an ℓ. By the induction hypothesis, $x_{i\ell} \in SSat(E_{k_1})$ and $x_{(i+\ell-1)(j-\ell+1)} \in SSat(E_{k_2})$. By Lemma 5 part one, $(x_{i(\ell-1)})^{St(2)}x_{(1+\ell-1)1} \in SSat(E_{k_1})$ and $x_{(i+\ell-1)}(x_{(i+\ell)(j-\ell)})^{St(2)} \in SSat(E_{k_2})$. So, $x_{ij} \in SSat(E_k)$ as desired.

Case:

Suppose $\Lambda \in E_{k_1}$ and $A_{ijk_2}=$"yes". By the induction hypothesis, $x_{ij} \in SSat(E_{k_2})$. By Lemma 5 part one, $(x_{ij})^{St(2)} \in SSat(E_{k_2})$ and $(\Lambda)^{St(2)} \in SSat(E_{k_1})$. So, as $(x_{ij})^{St(2)} = (\Lambda)^{St(2)}(x_{ij})^{St(2)}$, we get $x_{ij} \in SSat(E_k)$ as desired.

Case:

Suppose $\Lambda \in E_{k_2}$ and A_{ijk_1}="yes". By the induction hypothesis, $x_{ij} \in SSat(E_{k_1})$. By Lemma 5 part one, $(x_{ij})^{St(2)} \in SSat(E_{k_1})$ and $(\Lambda)^{St(2)} \in SSat(E_{k_2})$. So, as $(x_{ij})^{St(2)} = (x_{ij})^{St(2)}(\Lambda)^{St(2)}$, we get $x_{ij} \in SSat(E_k)$ as desired.

Thus, by induction Algorithm SSat is correct. ∎

Theorem 8 (Closure) DSA(Σ) *is closed under complementation, union, intersection, product, and Kleene closure.*

Proof of 8:

We prove the lemma by induction on the operations of automata.

Case: Complementation

Suppose $V = \mathcal{L}(\mathcal{A})$ where $\mathcal{A} = \langle T, F\rangle$. Then $\overline{V} = T(\mathcal{A}')$ where $\mathcal{A}' = \langle T, \overline{F}\rangle$ and $\overline{F} = S - F$. Claim: $w \in \mathcal{L}(\mathcal{A})$ iff $w \notin \mathcal{L}(\mathcal{A}')$.

Proof:

If $w \in \mathcal{L}(\mathcal{A})$ then $(w)^{St(|\mathcal{A})} \in \mathcal{L}_C(\mathcal{A})$. Thus, there is a run from s_0 to some s_j where $s_j \in F$. As $\mathcal{A}$ is deterministic, $s_j \notin \overline{F}$. So $(w)^{St(|\mathcal{A})} \notin \mathcal{L}_C(\mathcal{A}')$. Hence $w \notin \mathcal{L}(\mathcal{A})$. Similarly, we can show if $w \notin \mathcal{L}(\mathcal{A}')$ then $w \in \mathcal{L}(\mathcal{A})$. ∎

Case: Union

Suppose $V_j = T(\mathcal{A}_j)$ for $1 \leq j \leq n$, where $\mathcal{A}_j = \langle T_j, F_j\rangle$. Then $\bigcup_j V_j = \mathcal{A} = \langle T, F\rangle$ where $T = \prod_j T_j$ and $f \in F$ iff $p_j(f) \in F_j$ for some $1 \leq j \leq k$. The product table can be easily shown to satisfy the stuttering condition since it is defined componentwise and each component satisfies the stuttering condition.

Case: Intersection

As complementation can be expressed in terms of union and intersection, by the two previous cases we have closure under intersection.

Case: Product

Suppose that $U = \mathcal{L}(\mathcal{A})$ and $V = \mathcal{L}(\mathcal{B})$, where $\mathcal{A} = \langle T, F\rangle$ and $\mathcal{B} = \langle T', F'\rangle$ has n states.

Assume that the empty word, Λ, is not in $\mathcal{L}(\mathcal{B})$. Thus the initial state of T' is not in F'. Let $\mathcal{C} = \langle T'', G'\rangle$ where $T'' = \text{fl}(\mathcal{A}, T')$ and $g' \in G'$ iff $p_j(g') \in F'$ for some $1 \leq j \leq n+2$.

By Lemma 9, we have the following:

$$\begin{array}{lll} w \in \mathcal{L}(\mathcal{C}) & \text{iff} & (w)^{St(|\mathcal{C}|)} \in \mathcal{L}_C(\mathcal{C}) \\ & \text{iff} & \exists w_1, w_2 \text{ such that} \\ & & w_1 \in \mathcal{L}_C(\mathcal{A}),\ w_2 \in \mathcal{L}_C(\mathcal{A}),\ \text{and } (w)^{St(|\mathcal{C}|)} = w_1 w_2\ . \end{array}$$

Case: Kleene closure

Suppose that $V = \mathcal{L}(\mathcal{A})$, where $\mathcal{A} = \langle T, F \rangle$ has n states. Take $k = n + 2$ copies of T and let the control unit switch ON a dormant copy every time any one of the T copies is in a final state of $\mathcal{A}$. In the initial state, the first copy is in the initial state of T while all other copies are dormant. Call the resulting table for such a systm T'' and define $\mathcal{C} = \langle T'', G' \rangle$, where $g' \in G'$ iff $p_j(g') \in F$ for some $1 \leq j \leq k$. By Lemma 9, for any $X \neq \Lambda$, $X \in \mathcal{L}(\mathcal{C})$ iff $X = X_1 \cdots X_m$ where $X_i \in V$ and $X_i \neq \Lambda$. (This can be seen as follows: suppose $X \in \mathcal{L}(\mathcal{C})$ and let j_0 be the copy which is in a final state at the end of X. If j_0 was last switched ON at i_0 then $X(i_0, |X|) \in V$. So some copy is in the final state at i_0. Call this copy j_1. If j_1 was switched ON at $i_1 < i_0$ then $X(i_1, i_0) \in V$. After a finite number of steps, we arrive at a splitting of X as a finite product of nonempty tapes from V. Conversely, if $X = X_1 \cdots X_m$ where $X_i \in V$, then by induction on r some copy is in a final state at the end of $X_1 \cdots X_r$ for each $1 \leq r \leq m$; whence $X \in \mathcal{L}(\mathcal{C})$.) Taking $\mathcal{B}$ to be the automaton which defines $\mathcal{L}(\mathcal{C}) \cup \{\Lambda\}$, we get $\mathcal{L}(\mathcal{B}) = V^*$ as desired. ∎

Lemma 12 (Satisfaction Indistinguishability) *For any propositional* HTL *formula φ, for any two finite phases P_1 and P_2, if $P_1 =_s P_2$ then*

$$P_1 \models \varphi \qquad \textit{iff} \qquad P_2 \models \varphi.$$

Proof of 12:

We prove the lemma by induction on the structure of φ.

Case: Base

Suppose φ is a boolean formula. Let P_1 and P_2 be finite phases such that $P_1 =_s P_2$. Thus there exist constant slope $P^{1,1} = \langle I^{1,1}, f^{1,1} \rangle, \ldots, P^{1,a} = \langle I^{1,a}, f^{1,a} \rangle$ and constant slope $P^{2,1} = \langle I^{2,1}, f^{2,1} \rangle, \ldots, P^{2,a} = \langle I^{2,a}, f^{2,a} \rangle$ such that

- $P_1 = P^{1,1} \frown \cdots \frown P^{1,a}$ and
- $P_2 = P^{2,1} \frown \cdots \frown P^{2,a}$ and
- $\forall i \in [1..a]$, $\forall x_j \in V$, $f^{1,i}_{x_j}(I^{1,i}) = f^{2,i}_{x_j}(I^{2,i})$.

Fix such P's. Then $I^{1,1} = [t_1, t_2)$ for some $t_1, t_2 \in \mathcal{R}$ and $I^{2,1} = [t_3, t_4)$ for some $t_3, t_4 \in \mathcal{R}$. Then,

$P_1 \models \varphi$	iff	φ holds at t_1	
	iff	φ holds throughout $I^{1,1}$	since the interpretation of all variables is constant throughout $I^{1,1}$
	iff	φ holds throughout $I^{2,1}$	since for all $x_j \in V$, $f^{1,1}_{x_j} = f^{2,1}_{x_j}$
	iff	φ holds at t_3	since the interpretation of all variables is constant throughout $I^{2,1}$
	iff	$P_2 \models \varphi$	

Case: Inductive

Case: $\varphi = \varphi_1 \vee \varphi_2$

Suppose $P_1 =_s P_2$. Then,

$P_1 \models \varphi$	iff	$P_1 \models \varphi_1$ or $P_1 \models \varphi_2$	
	iff	$P_2 \models \varphi_1$ or $P_2 \models \varphi_2$	by induction hypothesis
	iff	$P_2 \models \varphi$	

Case: $\varphi = \neg\varphi_1$

Suppose $P_1 =_s P_2$. Then,

$P_1 \models \varphi$	iff	$P_1 \models \neg\varphi_1$	
	iff	$P_1 \not\models \varphi_1$	
	iff	$P_2 \not\models \varphi_1$	by induction hypothesis
	iff	$P_2 \models \neg\varphi_1$	
	iff	$P_2 \models \varphi$	

Case: $\varphi = \varphi_1 ; \varphi_2$

Suppose $P_1 =_s P_2$. Suppose $P_1 \models \varphi_1;\varphi_2$. Then $\exists\ P_1^A, P_1^B$ such that $P_1 = P_1^A \frown P_1^B$ and $P_1^A \models \varphi_1$ and $P_1^B \models \varphi_2$. Fix such P_1^A and P_1^B.

As $P_1 =_s P_2$, there exist constant slope $P^{1,1} = \langle I^{1,1}, f^{1,1}\rangle, \ldots, P^{1,a} = \langle I^{1,a}, f^{1,a}\rangle$ and constant slope $P^{2,1} = \langle I^{2,1}, f^{2,1}\rangle, \ldots, P^{2,a} = \langle I^{2,a}, f^{2,a}\rangle$ such that

- $P_1 = P^{1,1} \frown \cdots \frown P^{1,a}$ and
- $P_2 = P^{2,1} \frown \cdots \frown P^{2,a}$ and
- $\forall i \in [1..a], \forall x_j \in V,\ f^{1,i}_{x_j}(I^{1,i}) = f^{2,i}_{x_j}(I^{2,i})$.

Fix such P's.

Either for some k, $P_1^A = P^{1,1} \frown \cdots \frown P^{1,k}$ or there is a least k such that P_1^A is a left subphase of $P^{1,1} \frown \cdots \frown P^{1,k}$.

In the latter case, we may partition $P^{1,k}$ into P^{1,k_1} and P^{1,k_2} such that $P_1^A = P^{1,1} \frown \cdots \frown P^{1,k_1}$ and $P_1^B = P^{1,k_2} \frown \cdots \frown P^{1,a}$. In addition, we may then partition $P^{2,k}$ into two equal segments P^{2,k_1} and P^{2,k_2}.

So, without loss of generality, consider the case where $P_1^A = P^{1,1} \frown \cdots \frown P^{1,k}$ and $P_1^B = P^{1,k+1} \frown \cdots \frown P^{1,a}$. Let $P_2^A = P^{2,1} \frown \cdots \frown P^{2,k}$ and $P_2^B = P^{2,k+1} \frown \cdots \frown P^{2,a}$. As $\forall i \in [1..a], \forall x_j \in V,\ f^{1,i}_{x_j}(I^{1,i}) = f^{2,i}_{x_j}(I^{2,i})$, $P_1^A =_s P_2^A$, and $P_1^B =_s P_2^B$. So, by induction hypothesis $P_2^A \models \varphi_1$ and $P_2^B \models \varphi_2$, whence $P_2 \models \varphi_1;\varphi_2$.

Similarly, we can show if $P_2 \models \varphi_1;\varphi_2$ then $P_1 \models \varphi_1;\varphi_2$.

So, by induction the lemma holds. ∎

Lemma 13 (Soundness) *For any* HTL *formula* φ *and any phase* P*,*

$$P \models \varphi \qquad \textit{iff} \qquad T_m(P) \models_S T_f(\varphi).$$

Proof of 13:

We prove the lemma by induction on the structure of φ.

Case: Base

Suppose φ is a boolean formula. Consider any phase P and let $P^1, \ldots, P^a$ be the corresponding unique maximal constant-slope partition, where $P^1 = \langle [t_1, t_2), f^1 \rangle$. Then $T_m(P) = \sigma_1 \ldots \sigma_a$, where (for each $i \in [1..a]$)

$$\sigma_i = (\{x_j \mid f^i_{x_j}(I^i) = \mathit{true}\} \cup \{\overline{x_j} \mid f^i_{x_j}(I^i) = \mathit{false}\})$$

and $T_f(\varphi) = \varphi^+ \mathit{true}^+$.

Case: $P \models \varphi$

Then φ holds at t_1, so φ holds throughout I^1. So $\sigma_1 \models \varphi$. Thus $\sigma_1^2 \models_C \varphi^+$. As $\sigma_2^2 \ldots \sigma_a^2 \models_C \mathit{true}^+$, we get $\sigma_1^2 \ldots \sigma_a^2 \models_C \varphi^+ \mathit{true}^+$. Hence $\sigma_1 \ldots \sigma_a \models_S \varphi^+ \mathit{true}^+$. Thus, $T_m(P) \models_S T_f(\varphi)$.

Case: $T_m(P) \models_S T_f(\varphi)$

So $\sigma_1 \models \varphi$, whence φ holds throughout I^1. Thus, φ holds at t and $P \models \varphi$ as desired.

Case: Inductive

Case: $\varphi = \varphi_1 \vee \varphi_2$

Consider an arbitrary phase P.

$P \models \varphi$	iff	$P \models \varphi_1$ or $P \models \varphi_2$	
	iff	$T_m(P) \models_S T_f(\varphi_1)$ or $T_m(P) \models_S T_f(\varphi_2)$	by induction hypothesis
	iff	$T_m(P) \models_S T_f(\varphi_1) + T_f(\varphi_2)$	by definition of $\models_S$ for $+$
	iff	$T_m(P) \models_S T_f(\varphi_1 \vee \varphi_2)$	by definition of T_f

Case: $\varphi = \neg\varphi_1$

Consider an arbitrary phase P.

$P \models \varphi$	iff	$P \models \neg\varphi_1$	
	iff	$P \not\models \varphi_1$	
	iff	$T_m(P) \not\models_S T_f(\varphi_1)$	by induction hypothesis
	iff	$T_m(P) \models_S \neg T_f(\varphi_1)$	by definition of $\models_S$ for $\neg$
	iff	$T_m(P) \models_S T_f(\neg\varphi_1)$	by definition of T_f

Case: $\varphi = \varphi_1; \varphi_2$

Consider an arbitrary phase P.

Case: $P \models \varphi_1; \varphi_2$

Thus, there exist P_1 and P_2 partitioning P such that $P_1 \models \varphi_1$ and $P_2 \models \varphi_2$. Fix such P_1 and P_2.

By induction hypothesis, $T_m(P_1) \models_S T_f(\varphi_1)$ and $T_m(P_2) \models_S T_f(\varphi_2)$. There exist σ's such that $T_m(P_1) = \sigma_1 \ldots \sigma_\ell$, where $\sigma_\ell \neq \sigma_{\ell-1}$, and $T_m(P_2) = \sigma_{\ell+1} \ldots \sigma_a$, where $\sigma_{\ell+1} \neq \sigma_{\ell+2}$.

Case: $\sigma_\ell = \sigma_{\ell+1}$

Then $\sigma_1^2 \ldots \sigma_{\ell-1}^2 \sigma_\ell \models_S T_f(\varphi_1)$ and $\sigma_\ell^2 \sigma_{\ell+1}^2 \ldots \sigma_a^2 \models_S T_f(\varphi_2)$. So by definition of $\models_S$ for concatenation, $T_m(P) = \sigma_1 \ldots \sigma_\ell \sigma_{\ell+2} \ldots \sigma_a \models_S T_f(\varphi_1) T_f(\varphi_2)$.

Case: $\sigma_\ell \neq \sigma_{\ell+1}$

Then $\sigma_1^2 \ldots \sigma_\ell^2 \models_S T_f(\varphi_1)$ and $\sigma_{\ell+1}^2 \ldots \sigma_a^2 \models_S T_f(\varphi_2)$. So by definition of $\models_S$ for concatenation, $T_f(P) = \sigma_1 \ldots \sigma_a \models_S T_f(\varphi_1) T_f(\varphi_2)$.

In either case, we have shown $T_m(P) \models_S T_f(\varphi_1; \varphi_2)$.

Case: $T_m(P) \models_S T_f(\varphi_1; \varphi_2)$

There exist σ's such that $T_m(P) = \sigma_1 \ldots \sigma_a$. Thus, $\sigma_1 \ldots \sigma_a \models_S T_f(\varphi_1; \varphi_2)$. So by definition of $\models_S$ for concatenation, there exists an ℓ such that either

$\sigma_1^2 \ldots \sigma_{\ell-1}^2 \sigma_\ell \models_S T_f(\varphi_1)$ and $\sigma_\ell \sigma_{\ell+1}^2 \ldots \sigma_a^2 \models_S T_f(\varphi_2)$

or $\sigma_1^2 \ldots \sigma_\ell^2 \models_S T_f(\varphi_1)$ and $\sigma_{\ell+1}^2 \ldots \sigma_a^2 \models_S T_f(\varphi_2)$

Case: $\sigma_1^2 \ldots \sigma_{\ell-1}^2 \sigma_\ell \models_S T_f(\varphi_1)$ and $\sigma_\ell \sigma_{\ell+1}^2 \ldots \sigma_a^2 \models_S T_f(\varphi_2)$

Thus, $\sigma_1 \ldots \sigma_\ell \models_S T_f(\varphi_1)$ and $\sigma_\ell \ldots \sigma_a \models_S T_f(\varphi_2)$. So there exist partitions P_1 and P_2 such that $T_f(P_1) = \sigma_1 \ldots \sigma_\ell$ and $T_f(P_2) = \sigma_\ell \ldots \sigma_a$ and $P = P_1; P_2$. This is possible since the last constant-slope subphase of P_1 equals the first constant-slope subphase of P_2.

Case: $\sigma_1^2 \ldots \sigma_\ell^2 \models_S T_f(\varphi_1)$ and $\sigma_{\ell+1}^2 \ldots \sigma_a^2 \models_S T_f(\varphi_2)$

Thus, $\sigma_1 \ldots \sigma_\ell \models_S T_f(\varphi_1)$ and $\sigma_{\ell+1} \ldots \sigma_a \models_S T_f(\varphi_2)$. So there exist partitions P_1 and P_2 such that $T_f(P_1) = \sigma_1 \ldots \sigma_\ell$ and $T_f(P_2) = \sigma_\ell \ldots \sigma_a$ and $P = P_1; P_2$.

So by induction the lemma holds. ∎

Lemma 14 (Completeness) *For any star-free* SRE E *and any* Σ*-word* w,

$$w \models_S E \qquad \textit{iff} \qquad T^r_m(w) \models T^-_f(E).$$

Proof of 14:

We prove the lemma by induction on the structure of E. Recall, that we only consider the star-free SRE.

Case: Base

Suppose E is of the form $\psi^+ \mathit{true}^+$ for some propositional formula ψ. Consider an arbitrary word $w = \sigma_1 \ldots \sigma_a$. Then $T^r_m(w) = P_1 {}^\frown \ldots {}^\frown P_a$.

Case: $w \models_S \psi^+ \mathit{true}^+$

As $w \models_S E$, $\sigma_1 \models_C \psi$, whence ψ holds throughout P_1. So by definition of $\models$ in HTL $T^r_m(w) \models \psi = T^-_f(\psi^+ \mathit{true}^+)$.

Case: $T^r_m(w) \models T^-_f(E)$

Thus, $P_1 \models \psi$. As P_1 is a constant-slope phase, we get ψ holds throughout P_1, whence $\sigma_1 \models_C \psi$ (as $P_1 = T^r_m(\sigma_1)$). So $w \models_S \psi^+ \mathit{true}^+$ as desired.

Case: Inductive

Case: $E = E_1 \vee E_2$

Consider an arbitrary Σ-word w.

$$\begin{array}{lll l}
w \models_S E & \text{iff} & w \models_S E_1 \text{ or } w \models_S E_2 & \\
& \text{iff} & \quad T^r_m(w) \models T^-_f(E_1) & \text{by induction hypothesis} \\
& & \quad \text{or } T^r_m(w) \models T^-_f(E_2) & \\
& \text{iff} & T^r_m(w) \models T^-_f(E_1) + T^-_f(E_2) & \text{by definition of} \models \text{for} + \\
& \text{iff} & T^r_m(w) \models T^-_f(E_1 \vee E_2) & \text{by definition of } T^-_f
\end{array}$$

Case: $E = \neg E$

Consider an arbitrary Σ-word w.

$$\begin{array}{llll}
w \models_S E & \text{iff} & w \models_S \neg E_1 & \\
& \text{iff} & w \not\models_S E_1 & \\
& \text{iff} & T^r_m(w) \not\models T^-_f(E_1) & \text{by induction hypothesis} \\
& \text{iff} & T^r_m(w) \models \neg T^-_f(E_1) & \text{by definition of} \models \text{for} \neg \\
& \text{iff} & T^r_m(w) \models T^-_f(\neg E) & \text{by definition of } T^-_f
\end{array}$$

Case: $E = E_1 E_2$

Consider an arbitrary Σ-word $w = \sigma_1 \ldots \sigma_a$. Then $T^r_m(w) = P_1 {}^\frown \ldots {}^\frown P_a$ for appropriate $P_1, \ldots, P_a$.

Case: $w \models_S E$

Then there exists an ℓ such that $(w)^{St(2)} = v_1 v_2$ and either

$v_1 = \sigma_1^2 \ldots \sigma_\ell^2$ and $v_2 = \sigma_{\ell+1}^2 \ldots \sigma_a^2$.
or $v_1 = \sigma_1^2 \ldots \sigma_{\ell-1}^2 \sigma_\ell$ and $v_2 = \sigma_\ell \sigma_{\ell+1}^2 \ldots \sigma_a^2$.

Case: $v_1 = \sigma_1^2 \ldots \sigma_\ell^2$ and $v_2 = \sigma_{\ell+1}^2 \ldots \sigma_a^2$

So, $\sigma_1 \ldots \sigma_\ell \models_S E_1$ and $\sigma_{\ell+1} \ldots \sigma_a \models_S E_2$. Then $T_m^r(\sigma_1 \ldots \sigma_\ell) \models T_f^-(E_1)$ and $T_m^r(\sigma_{\ell+1} \ldots \sigma_a) \models T_f^-(E_2)$. $T_m^r(\sigma_1 \ldots \sigma_\ell) T_m^r(\sigma_{\ell+1} \ldots \sigma_a) \models T_f^-(E_1); T_f^-(E_2)$, whence $T_m^r(w) = T_m^r(\sigma_1 \ldots \sigma_a) \models T_f^-(E_1); T_f^-(E_2) = T_f^-(E_1 E_2)$.

Case: $v_1 = \sigma_1^2 \ldots \sigma_{\ell-1}^2 \sigma_\ell$ and $v_2 = \sigma_\ell \sigma_{\ell+1}^2 \ldots \sigma_a^2$

So, $\sigma_1 \ldots \sigma_\ell \models_S E_1$ and $\sigma_{\ell+1} \ldots \sigma_a \models_S E_2$. Then $T_m^r(\sigma_1 \ldots \sigma_\ell) \models T_f^-(E_1)$ and $T_m^r(\sigma_\ell \ldots \sigma_a) \models T_f^-(E_2)$. So, $T_m^r(\sigma_1 \ldots \sigma_\ell) T_m^r(\sigma_\ell \ldots \sigma_a) \models T_f^-(E_1); T_f^-(E_2)$, in which case $P_1 {}^\frown \ldots {}^\frown P_\ell {}^\frown P_\ell {}^\frown \ldots {}^\frown P_a \models T_f^-(E_1); T_f^-(E_2)$. As $P_1 {}^\frown \ldots {}^\frown P_\ell {}^\frown P_\ell {}^\frown \ldots {}^\frown P_a =_s P_1 {}^\frown \ldots {}^\frown P_a = T_m^r(w)$, $T_m^r(w) \models T_f^-(E_1); T_f^-(E_2)$. Thus, $T_m^r(w) \models T_f^-(E_1 E_2)$ as desired.

Thus, in either case $T_m^r(w) \models T_f^-(E_1 E_2)$.

Case: $T_m^r(w) \models T_f^-(E_1 E_2)$

Then $P_1 {}^\frown \ldots {}^\frown P_a \models T_f^-(E_1); T_f^-(E_2)$. So there exists ℓ such that either

$P_1 {}^\frown \ldots {}^\frown P_\ell \models T_f^-(E_1)$ and $P_{\ell+1} {}^\frown \ldots {}^\frown P_a \models T_f^-(E_2)$
or $P_1 {}^\frown \ldots P_\ell \models T_f^-(E_1)$ and $P_\ell {}^\frown \ldots P_a \models T_f^-(E_2)$

Case: $P_1 {}^\frown \ldots {}^\frown P_\ell \models T_f^-(E_1)$ and $P_{\ell+1} {}^\frown \ldots {}^\frown P_a \models T_f^-(E_2)$

So by induction hypothesis, $T_m^r(\sigma_1 \ldots \sigma_\ell) \models T_f^-(E_1)$ and $T_m^r(\sigma_{\ell+1} \ldots \sigma_a) \models T_f^-(E_2)$. So $\sigma_1 \ldots \sigma_\ell \models_S E_1$ and $\sigma_{\ell+1} \ldots \sigma_a \models_S E_2$. So $(\sigma_1 \ldots \sigma_\ell)^{St(2)} \models_S E_1$ and $(\sigma_{\ell+1} \ldots \sigma_a)^{St(2)} \models_S E_2$, whence $w = \sigma_1 \ldots \sigma_a \models_S E_1 E_2$.

Case: $P_1 {}^\frown \ldots {}^\frown P_\ell \models T_f^-(E_1)$ and $P_\ell {}^\frown \ldots {}^\frown P_a \models T_f^-(E_2)$

So by induction hypothesis, $T_m^r(\sigma_1 \ldots \sigma_\ell) \models T_f^-(E_1)$ and $T_m^r(\sigma_\ell \ldots \sigma_a) \models T_f^-(E_2)$. So $\sigma_1 \ldots \sigma_\ell \models_S E_1$ and $\sigma_\ell \ldots \sigma_a \models_S E_2$. So $(\sigma_1 \ldots \sigma_{\ell-2})^{St(2)} \sigma_\ell \models_S E_1$ and $\sigma_\ell (\sigma_{\ell+1} \ldots \sigma_a)^{St(2)} \models_S E_2$, whence $w = \sigma_1 \ldots \sigma_a \models_S E_1 E_2$.

Thus, in either case $w \models_S E_1 E_2$.

So by induction the lemma holds. ∎

Chapter 5

Transition Systems

In this chapter, we formally introduce models for hybrid systems. These models will serve as a basis for forming verification rules in the rest of the thesis. Each model is a formal representation of hybrid systems under one of the three semantics introduced in Chapter 3. We introduce two transition system approaches to modeling hybrid systems: *phase transition systems* and *concrete phase transition systems*. As was the case for linear-time temporal logic, phase transition systems have two associated semantics: a sampling semantics based on [114, 93, 47] and a super-dense semantics. Concrete phase transition systems have only a single semantics, namely a continuous semantics based on Kapur, Henzinger, Manna, and Pnueli [91].

5.1 Contributions

The contributions of this chapter are

- an extension of the phase transition system model of [114, 93] to our super-dense semantics introduced in Chapter chap-spec, and
- the concrete phase transition system model, which formally models the behavior of hybrid systems under the continuous interval semantics.

5.2 Phase Transition Systems

We first introduce *phase transition systems* (PTS) [114, 93, 47]. A PTS is a transition system that allows continuous state changes over time periods of positive duration as well as discrete state changes in zero time. The model considers the solutions of the differential equations that govern the continuous evolution of the system as given, separating the concerns of solving the differential equations from those of studying the temporal properties of the system. A PTS $\mathcal{S} = (\mathcal{V}, \theta, \mathcal{T}, \Pi, \mathcal{A})$ consists of the following components:

1. A set $\mathcal{V}$ of typed state variables, partitioned into the set $\mathcal{V}_d$ of *discrete variables*, the set $\mathcal{V}_c$ of *clock variables*, and the set $\mathcal{V}_h$ of *hybrid variables.* Clock variables have type $\mathcal{R}^+$ (i.e., the set of non-negative real numbers) and hybrid variables have type $\mathcal{R}$. We distinguish a special clock variable $T \in \mathcal{V}_c$, representing a *master clock* that measures the amount of time elapsed during the system behavior. The state space S consists of all type-consistent interpretations of the variables in $\mathcal{V}$; we denote by $s[\![x]\!]$ the value at state $s \in S$ of variable $x \in \mathcal{V}$. We define $\mathcal{V}'$ as the set $\mathcal{V}' = \{x' \mid x \in \mathcal{V}\}$.

2. An assertion θ over $\mathcal{V}$, which defines the set $\{s \in S \mid s \models \theta\}$ of initial states.

3. A finite set $\mathcal{T}$ of transition assertions over $\mathcal{V}$, $\mathcal{V}'$ representing the discrete state changes. Each assertion $\pi \in \mathcal{T}$ represents the transition relation $\{(s_1, s_2) \mid (s_1, s_2) \models \pi\}$, where (s_1, s_2) interprets $x \in \mathcal{V}$ as $s_1[\![x]\!]$ and $x' \in \mathcal{V}'$ as $s_2[\![x]\!]$. For all $\pi \in \mathcal{T}$, we require that the implication $\pi \rightarrow T = T'$ holds. This implies that discrete transitions occur in zero time.

4. A *time-progress* assertion Π over $\mathcal{V}$, used to specify a restriction on the progress of time. The time-progress assertion, introduced by Nicollin, Olivero, Sifakis, and Yovine in [125][1], is often used to express different scheduling policies for executing transitions. For example, to express a synchronous scheduling policy, where a transition τ must be taken once it is enabled, our time-progress assertion becomes $\neg \mathit{Enabled}(\tau)$, where $\mathit{Enabled}(\tau)$ is a predicate denoting when τ is enabled. Alternative scheduling policies can also be expressed using the time-progress assertion, and we refer the reader to [125] where such alternatives are discussed and compared. The time progress assertion was originally introduced [125] to show the generality of the transition system model. By abstracting away scheduling policies into Π, general verification rules can be developed that work for a wide-array of hybrid systems.

5. A finite set $\mathcal{A}$ of *activities* representing the continuous state changes. Each activity $a \in \mathcal{A}$ consists of an *enabling assertion* C_a over $\mathcal{V}_d$ and of an evolution function $F_a : S \times \mathcal{R} \mapsto S$. At every $s \in S$ there must be exactly one $a \in \mathcal{A}$ such that $s \models C_a$. If at time t the system is at a state $s \models C_a$, at time $t + \Delta$ the system will be at state $F_a(s, \Delta)$. For every $a \in \mathcal{A}$, the function F_a must satisfy the equations

$$\forall x \in \mathcal{V}_d \,.\, F_a(s,t)[\![x]\!] = s[\![x]\!] \qquad F_a(s,0) = s \tag{5.1}$$

$$\forall x \in \mathcal{V}_c \,.\, F_a(s,t)[\![x]\!] = s[\![x]\!] + t \qquad F_a(s,t) = F_a(F_a(s,t'), t-t') \tag{5.2}$$

[1]In [125], the time-progress assertion is called the "time can progress" predicate.

for every $s \models C_a$, $t \geq 0$ and $0 \leq t' \leq t$. The function F_a is represented by the set of terms $\{F_a^x\}_{x \in \mathcal{V}}$ over $\mathcal{V} \cup \{\Delta\}$, where the term F_a^x gives the temporal evolution of the value of x as a function of the elapsed time Δ.

To define the set of computations of a PTS, we introduce the assertions $\{tick_a[\Delta]\}_{a \in \mathcal{A}}$, where each $tick_a[\Delta]$ is an assertion over $\mathcal{V} \cup \mathcal{V}'$ and over the parameter Δ, whose domain is the set $\mathcal{R}^+$ of non-negative real numbers. Assertion $tick_a[\Delta]$ describes a state change of the system caused by activity a when an amount of time $\Delta \geq 0$ elapses and is given by:

$$C_a \wedge \Big(\bigwedge_{x \in \mathcal{V}} (x' = F_a^x[\Delta]) \Big) \wedge \forall t \,.\, \Big(0 \leq t < \Delta \rightarrow \Pi \Big[F_a^x[t]/x \Big]_{x \in \mathcal{V}} \Big) \;.$$

In the above formula, $\Pi[F_a^x[t]/x]_{x \in \mathcal{V}}$ denotes the result of simultaneously replacing each occurrence of x in Π with $F_a^x[t]$, for all $x \in \mathcal{V}$. The form of the assertion $tick_a[\Delta]$ insures that the progress constraint Π holds at every moment of a time-step, except possibly for the final one. As discussed in Kesten, Manna, and Pnueli [93], if Π is used only to encode upper bounds on the transition waiting times, assertion $tick_a[\Delta]$ can be rewritten without quantifiers.

Sampling Semantics

In this subsection, we define the behavior of a PTS in the sampling semantics, in which a computation of the system consists of an enumerable sequence of system states.

A *sampling computation* of a PTS $\mathcal{S} = (\mathcal{V}, \theta, \mathcal{T}, \Pi, \mathcal{A})$ is an infinite sequence $\sigma : s_0, s_1, s_2, \ldots$ of states of $\mathcal{S}$ that satisfies the following conditions:

Initiality: $s_0 \models \theta$.

Consecution: for each $i \geq 0$, one of the following holds:

1. there is a transition $\pi \in \mathcal{T}$ such that $(s_i, s_{i+1}) \models \pi$;
2. there is an activity $a \in \mathcal{A}$ such that $(s_i, s_{i+1}) \models \exists \Delta \geq 0 \,.\, tick_a[\Delta]$.

Time progress: for each $t \in \mathcal{R}$ there is $i \in \mathbb{N}$ such that $s_i(T) \geq t$.

We denote by $\mathcal{L}(\mathcal{S})$ the set of sampling computations of a PTS $\mathcal{S}$.

Continuous Semantics

In this subsection, we define the behavior of a PTS in the continuous (i.e., super-dense) semantics, in which a computation of the system consists of a super-dense run of the system. The relationship between the sampling semantics and the continuous semantics for hybrid systems has been studied by de Alfaro and Manna in [48].

A *continuous computation* of a PTS $\mathcal{S} = (\mathcal{V}, \theta, \mathcal{T}, \Pi, \mathcal{A})$ is super-dense run $\sigma = \langle \bar{s}, f \rangle$, where $\bar{s}$ is an infinite sequence of states of $\mathcal{S}$, $s_0, s_1, s_2, \ldots$, and where σ satisfies the following conditions:

Initiality: $s_0 \models \theta$.

Consecution: for each $i \geq 0$, one of the following holds:

1. there is a transition $\pi \in \mathcal{T}$ such that $(s_i, s_{i+1}) \models \pi$;
2. there is an activity $a \in \mathcal{A}$ such that $(s_i, s_{i+1}) \models \exists \Delta \geq 0 \,.\, \mathit{tick}_a[\Delta]$, and for each $x \in \mathcal{V}$, for each $t \in [\mathit{time}(s_i), \mathit{time}(s_{i+1}))$, $f_x(t) = F_a^x(t)$.

Time progress: for each $t \in \mathcal{R}$ there is $i \in \mathbb{N}$ such that $s_i(T) \geq t$.

We denote by $\mathcal{L}_c(\mathcal{S})$ the set of continuous computations of a PTS $\mathcal{S}$.

Room-Heater Example

Returning to system *RH* introduced in Chapter 2.2.3, our PTS $\mathcal{S} = (\mathcal{V}, \theta, \mathcal{T}, \Pi, \mathcal{A})$ is defined as follows.

1. $\mathcal{V}_d = \{H, W\}$, where H denotes the state of the heater and ranges over domain $\{\mathit{On}, \mathit{Off}\}$, and W denotes the state of the window and ranges over domain $\{\mathit{Open}, \mathit{Closed}\}$. $\mathcal{V}_c = \{T, y\}$, where T is the global clock, and y measures the time elapsed since the last switching *On*/*Off* of the heater. $\mathcal{V}_h = \{x\}$, where x is the temperature of the room.

2. $\theta : H = \mathit{Off} \;\wedge\; W = \mathit{Closed} \;\wedge\; x < 60 \;\wedge\; y = 0 \;\wedge\; T = 0$.

3. $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_i : E_i \wedge R_i$ for $i \in \{1, 2, 3\}$, and

$$\begin{array}{ll}
E_1 : H = \mathit{Off} \;\wedge\; x \leq 68 \;\wedge\; y \geq 60 & R_1 : H' = \mathit{On} \;\wedge\; y' = 0 \\
E_2 : H = \mathit{On} \;\wedge\; x \geq 72 \;\wedge\; y \geq 60 & R_2 : H' = \mathit{Off} \;\wedge\; y' = 0 \\
E_3 : \mathit{true} & R_3 : W' = \neg W
\end{array}$$

   where $\neg \mathit{Open} = \mathit{Closed}$ and $\neg \mathit{Closed} = \mathit{Open}$. Variables not mentioned in R_1, R_2, and R_3, respectively, are left unchanged by the transitions.

4. $\Pi = \neg E_1 \wedge \neg E_2$. This condition insures that τ_1 and τ_2 are taken as soon as they become enabled.

5. $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$, where $F^T_{a_i} = T + \Delta$, $F^y_{a_i} = y + \Delta$, for every $i \in \{1, 2, 3, 4\}$, and C_{a_i} and $F^x_{a_i}$ are defined as follows:

$$\begin{array}{ll} C_{a_1} : H = \mathit{Off} \ \wedge \ W = \mathit{Closed} & F^x_{a_1} = 60 + e^{-\Delta/105}(x - 60) \\ C_{a_2} : H = \mathit{Off} \ \wedge \ W = \mathit{Open} & F^x_{a_2} = 60 + e^{-\Delta/70}(x - 60) \\ C_{a_3} : H = \mathit{On} \ \wedge \ W = \mathit{Closed} & F^x_{a_3} = 75 + e^{-\Delta/105}(x - 75) \\ C_{a_4} : H = \mathit{On} \ \wedge \ W = \mathit{Open} & F^x_{a_4} = 70 + e^{-\Delta/70}(x - 70) \ . \end{array}$$

5.3 Concrete Phase Transition Systems

Following [70, 114, 126], we model hybrid systems as transition systems. Just as discrete transitions can be represented as binary relations on states, hybrid transitions can be represented as binary relations on phases. Phases are characterized by phase invariants, which are presented as assertions (first-order formulas) $\rho_\phi(V, \dot{V})$ in the two variable tuples V and $\dot{V}$, intended to hold at all intermediate points $t \in [a, b)$ of the phase, where the set $\dot{V}$ is defined as $\dot{V} = \{\dot{x} \mid x \in V\}$.

For a given phase invariant ϕ, a phase $P = \langle I, f\rangle$ over V is said to be a *ϕ-phase* if $P \models \mathit{continuous} \wedge \Box(\rho_\phi(V, \dot{V}))$.

For example the phase invariant ϕ presented as:

$$\rho_\phi(V, \dot{V}):\ 3 \leq x < 6 \ \wedge \ \dot{x} = 1$$

characterizes all phases in which x steadily increases at a rate of 1 and always remains within the interval $[3, 6)$.

A *Concrete Phase Transition System* (CPTS) $\mathcal{S} = (V, \Phi, \Theta, \mathcal{T})$ consists of four components:

1. A finite set V of *state variables.*
2. A finite set Φ of *phase invariants* over V. Each phase invariant $\phi \in \Phi$ is presented by an assertion of the form $\rho_\phi(V, \dot{V})$, referring to the state variables and their derivatives.
3. An initial condition, Θ, which is a state formula over V that specifies the initial value of the variables at the left end of the first phase in computations.
4. A set $\mathcal{T}$ of *transitions.* Each transition $\tau \in \mathcal{T}$ is associated with an assertion $\rho_\tau(V, V')$, relating values at the right-end limit state of a phase to the values at the left-end of a successor phase.

A *phase sequence* is a finite or infinite sequence of adjacent phases. For a phase sequence $\overline{P} = P_0, P_1, \ldots$, we denote by $\overline{P}^*$ the single phase obtained by the concatenation $P_0 {}^\frown P_1 {}^\frown \cdots$. An HTL-formula can be interpreted over a phase sequence $\overline{P}$ by interpreting it over the single phase $\overline{P}^*$.

Two phase sequences $\overline{P}_1$ and $\overline{P}_2$ are *equivalent* if $\overline{P}_1^* = \overline{P}_2^*$. It follows that all equivalence classes of state sequences are *closed under stuttering*: if a phase P_i of the phase sequence $\overline{P}$ is split into two phases P' and P'' that partition P_i, the resulting phase sequence

$$P_0, \ldots P_{i-1}, P', P'', P_{i+1}, \ldots P_n$$

is equivalent to $\overline{P}$. Closure under stuttering allows for undersampling and oversampling. That is, the truth value of a formula over a phase does not change by refinement or fusion of some of its subphases.

Let $\overline{P} = P_0, P_1, P_2, \ldots$ be an infinite phase sequence with $P_i = \langle [a_i, a_{i+1}), f_i \rangle$ for all $i \geq 0$. The infinite phase sequence $\overline{P}$ *diverges* if a_i grows beyond any bound as i increases. A finite phase sequence $\overline{P} = P_0, \ldots, P_n$, with $P_i = \langle [a_i, a_{i+1}), f_i \rangle$ for all $0 \leq i \leq n$, *diverges* if $a_{n+1} = \infty$.

A phase sequence is a *computation* of the CPTS $\mathcal{S}$ if it is equivalent to a phase sequence $\overline{P} = P_0, P_1, \ldots, P_n, \ldots$ that satisfies the following conditions:[2]

Initiality: If $P_0 = [a, b)$ then Θ holds at a.

Continuous activities: For all $0 \leq i < |\overline{P}|$, there is a phase invariant $\rho_\phi \in \Phi$ such that P_i is a ϕ-phase.

Discrete transitions: For all $0 \leq i < |\overline{P}| - 1$, there is a transition $\tau \in \mathcal{T}$ such that $\rho_\tau(\overrightarrow{P_i}[V], \overleftarrow{P_{i+1}}[V])$ holds.

Divergence: $\overline{P}$ is divergent.

A finite sequence of finite phases $\overline{P} = P_0, P_1, \ldots, P_n$ is called a *run fragment* of $\mathcal{S}$ if it satisfies the first three requirements of a computation but is not required to be divergent. In fact, such a sequence cannot be divergent. The system $\mathcal{S}$ is called a *non-Zeno* CPTS if every run fragment of $\mathcal{S}$ can be extended to a computation of $\mathcal{S}$. From now on we restrict our attention to non-Zeno CPTS's.

The CPTS $\mathcal{S}$ *satisfies* a hybrid temporal formula φ, written $\mathcal{S} \models \varphi$, if all computations of $\mathcal{S}$ satisfy φ.

5.4 Hybrid Automata

Many of the standard automata and diagram-based methods for presenting hybrid systems have a natural representation as CPTSs. We use hybrid automata [10, 9, 83] to specify CPTSs. We present a brief description of our representation of hybrid automata. Details can be found in [10, 9].

A *hybrid automaton* is a directed labeled graph $D = (V_D, L, E, E_\theta, \Omega, \mu, \kappa)$ consisting of the following:

- A finite set V_D of *data variables.*

[2] $\overline{P}$ may be finite or infinite. If it is infinite, then $|\overline{P}| = \infty$.

- A finite set L of locations where each location $\ell \in L$ is labeled by
  - a finite set $\Omega(\ell)$ of differential equations over the variables V_D, and
  - a *stay condition* $\mu(\ell)$, which specifies the conditions under which the system can stay in location ℓ.
- A finite set E of *edges* between the locations in L. Each edge e is labeled by a guarded command $\kappa(e) : \gamma \rightarrow \alpha$, where γ is a state formula over the variables in V_D (the *guard* of e) and α is a conjunction of the form $u_1 := e_1 \wedge \cdots \wedge u_m := e_m$, where $\{u_1, \ldots, u_m\}$ is a subset of V_D and $e_1, \ldots, e_m$ are expressions over V_D.
- An *entry edge*, E_θ, that has no originating location, but an entry location $\ell_i \in L$. E_θ is labeled by a formula $\kappa(E_\theta)$ of the form $v_1 = c_1 \wedge \cdots \wedge v_n = c_n$, which specifies initial values for all the data variables $\{v_1, \ldots, v_n\} = V_D$.

System GAS

An implementation of the gas burner introduced in Section 2.2 is given in Figure 5.4. The system GAS has two environment variables: $\dot{L}$, which represents the rate at which gas leaks from the system and which varies depending on the switch's setting; and R, which represents the environment's wish to change the switch's setting. We also have the control variables *switch*, x, y, and T, where:

- *switch* represents the setting of the gas burner switch,
- x represents the system's global clock and advances at the rate of 1 at all times,
- y represents a node's local clock, and
- T represents the cumulative time spent in the leaking node ℓ_2 since the beginning of the computation or the most recent period in which *switch* has been continuously off for at least 100 time units.

In the figure, ¬Off = On and ¬On = Off. The transition from ℓ_1 to itself represents the environment's changing of the request variable. Similarly, the transition from ℓ_0 to ℓ_2 represents the environment's changing of the request variable immediately followed by the system's response which, in our formalism, is represented as a single transition. As stated earlier, we wish to prove the following safety property about system GAS:

$$\overrightarrow{x} - \overleftarrow{x} \geq 60 \quad \Rightarrow_f \quad 6(\overrightarrow{L} - \overleftarrow{L}) \leq \overrightarrow{x} - \overleftarrow{x}.$$

The concrete phase transition system corresponding to the above system is given in Figure 5.2.

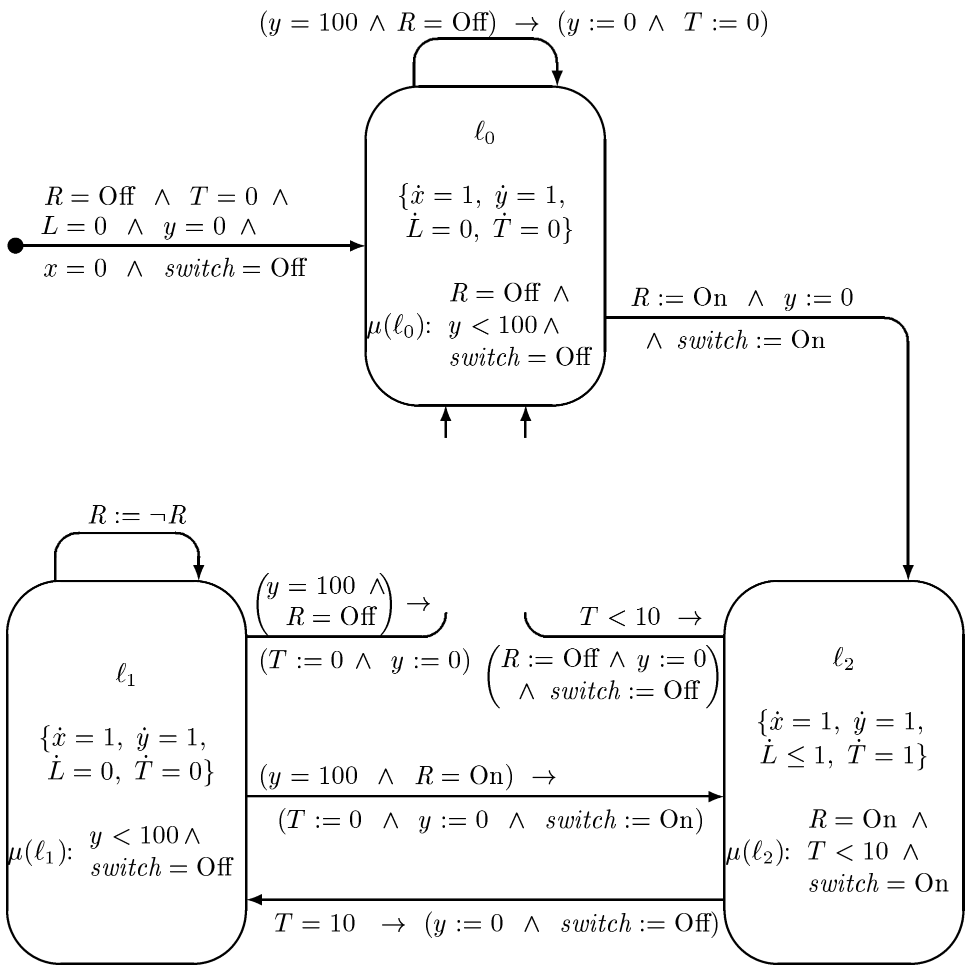
$(y = 100 \wedge R = \text{Off}) \rightarrow (y := 0 \wedge T := 0)$
ℓ_0
$\{\dot{x} = 1,\ \dot{y} = 1,\ \dot{L} = 0,\ \dot{T} = 0\}$
$\mu(\ell_0)$: $R = \text{Off} \wedge y < 100 \wedge \mathit{switch} = \text{Off}$
$R = \text{Off} \wedge T = 0 \wedge L = 0 \wedge y = 0 \wedge x = 0 \wedge \mathit{switch} = \text{Off}$
$R := \text{On} \wedge y := 0 \wedge \mathit{switch} := \text{On}$
$R := \neg R$
ℓ_1
$\{\dot{x} = 1,\ \dot{y} = 1,\ \dot{L} = 0,\ \dot{T} = 0\}$
$\mu(\ell_1)$: $y < 100 \wedge \mathit{switch} = \text{Off}$
$(y = 100 \wedge R = \text{Off}) \rightarrow (T := 0 \wedge y := 0)$
$T < 10 \rightarrow (R := \text{Off} \wedge y := 0 \wedge \mathit{switch} := \text{Off})$
$(y = 100 \wedge R = \text{On}) \rightarrow (T := 0 \wedge y := 0 \wedge \mathit{switch} := \text{On})$
$T = 10 \rightarrow (y := 0 \wedge \mathit{switch} := \text{Off})$
ℓ_2
$\{\dot{x} = 1,\ \dot{y} = 1,\ \dot{L} \le 1,\ \dot{T} = 1\}$
$\mu(\ell_2)$: $R = \text{On} \wedge T < 10 \wedge \mathit{switch} = \text{On}$

Figure 5.1: System GAS—Three state gas burner

$$\mathcal{S}_{\mathrm{GAS}} \;=\; \big(V,\ \Phi,\ \Theta,\ \mathcal{T}\big), \text{ where:}$$

$$\begin{array}{rl}
V = & \{R,\ L,\ T,\ \pi,\ x,\ y,\ \mathit{switch}\} \\
\Phi = & \{\phi_{\ell_0}, \phi_{\ell_1}, \phi_{\ell_2}\} \\
\Theta: & R = \mathrm{Off} \ \wedge\ T = 0 \ \wedge\ L = 0 \ \wedge\ y = 0 \ \wedge\ \mathit{switch} = \mathrm{Off} \ \wedge\ \pi = \ell_0 \ \wedge\ x = 0 \\
\mathcal{T} = & \{\tau_{\langle \ell_0,\ell_0\rangle}, \tau_{\langle \ell_0,\ell_2\rangle}, \tau_{\langle \ell_1,\ell_0\rangle}, \tau_{\langle \ell_1,\ell_1\rangle}, \tau_{\langle \ell_1,\ell_2\rangle}, \tau_{\langle \ell_2,\ell_0\rangle}, \tau_{\langle \ell_2,\ell_1\rangle}\} \\
\\
\rho_{\ell_0}: & \dot{x} = 1 \ \wedge\ \dot{y} = 1 \ \wedge\ \dot{L} = 0 \ \wedge\ \dot{T} = 0 \\
& \wedge\ y < 100 \ \wedge\ R = \mathrm{Off} \ \wedge\ \pi = \ell_0 \ \wedge\ \mathit{switch} = \mathrm{Off} \\
\rho_{\ell_1}: & \dot{x} = 1 \ \wedge\ \dot{y} = 1 \ \wedge\ \dot{L} = 0 \ \wedge\ \dot{T} = 0 \\
& \wedge\ y < 100 \ \wedge\ \pi = \ell_1 \ \wedge\ \mathit{switch} = \mathrm{On} \\
\rho_{\ell_2}: & \dot{x} = 1 \ \wedge\ \dot{y} = 1 \ \wedge\ \dot{L} \le 1 \ \wedge\ \dot{T} = 1 \\
& \wedge\ R = \mathrm{On} \ \wedge\ T < 10 \ \wedge\ \pi = \ell_2 \ \wedge\ \mathit{switch} = \mathrm{Off} \\
\rho_{\langle \ell_0,\ell_0\rangle}: & y = 100 \ \wedge\ R = \mathrm{Off} \ \wedge\ \pi = \ell_0 \ \wedge\ R' = R \ \wedge\ L' = L \\
& \wedge\ T' = 0 \ \wedge\ x' = x \ \wedge\ y' = 0 \ \wedge\ \mathit{switch}' = \mathrm{Off} \wedge\ \pi' = \ell_0 \\
\rho_{\langle \ell_0,\ell_2\rangle}: & \pi = \ell_0 \ \wedge\ R' = \mathrm{On} \ \wedge\ L' = L \\
& \wedge\ T' = T \ \wedge\ x' = x \ \wedge\ y' = 0 \ \wedge\ \mathit{switch}' = \mathrm{On} \ \wedge\ \pi' = \ell_2 \\
\rho_{\langle \ell_1,\ell_0\rangle}: & R = \mathrm{Off} \ \wedge\ y = 100 \ \wedge\ \pi = \ell_1 \ \wedge\ R' = R \ \wedge\ L' = L \\
& \wedge\ T' = 0 \ \wedge\ x' = x \ \wedge\ y' = 0 \ \wedge\ \mathit{switch}' = \mathrm{Off} \ \wedge\ \pi' = \ell_0 \\
\rho_{\langle \ell_1,\ell_1\rangle}: & \pi = \ell_1 \ \wedge\ R' = \neg R \ \wedge\ L' = L \\
& \wedge\ T' = T \ \wedge\ x' = x \ \wedge\ y' = 0 \ \wedge\ \mathit{switch}' = \mathit{switch} \ \wedge\ \pi' = \ell_1 \\
\rho_{\langle \ell_1,\ell_2\rangle}: & R = \mathrm{On} \ \wedge\ y = 100 \ \wedge\ \pi = \ell_1 \ \wedge\ R' = R \ \wedge\ L' = L \\
& \wedge\ T' = 0 \ \wedge\ x' = x \ \wedge\ y' = 0 \ \wedge\ \mathit{switch}' = \mathrm{On} \ \wedge\ \pi' = \ell_2 \\
\rho_{\langle \ell_2,\ell_0\rangle}: & T < 10 \ \wedge\ \pi = \ell_2 \ \wedge\ R' = \mathrm{Off} \ \wedge\ L' = L \ \wedge\ T' = T \\
& \wedge\ x' = x \ \wedge\ y' = 0 \ \wedge\ \mathit{switch}' = \mathrm{Off} \ \wedge\ \pi' = \ell_0 \\
\rho_{\langle \ell_2,\ell_1\rangle}: & T = 10 \ \wedge\ \pi = \ell_2 \ \wedge\ R' = R \ \wedge\ L' = L \ \wedge\ T' = T \\
& \wedge\ x' = x \ \wedge\ y' = 0 \ \wedge\ \mathit{switch}' = \mathrm{Off} \ \wedge\ \pi' = \ell_1
\end{array}$$

Figure 5.2: The concrete phase transition system associated with system GAS

$V: \ \{\pi\} \cup V_D$
$\Theta: \ \kappa(E_\theta) \ \wedge \ \pi = \ell_i$ where ℓ_i is the entry location for E_θ
$\Phi: \ \{\phi_{\ell_i} \mid \ell_i \in L\}$ where, for each $\ell_i \in L$
$\quad \rho_{\ell_i}: \ \ \mu(\ell_i) \ \wedge \ \pi = \ell_i \ \wedge \ \left(\bigwedge_{\psi \in \Omega(\ell_i)} \psi\right)$
$\mathcal{T}: \ \{\tau_{\langle \ell_i, \ell_j \rangle} \mid \langle \ell_i, \ell_j \rangle \in E\}$ where, for each $e = \langle \ell_i, \ell_j \rangle \in E$ such that $\kappa(e) : \gamma \ \rightarrow \ \alpha$
$\quad$ where α is of the form $\bigwedge_{i=1}^{m} u_i := e_i$,
$\quad \rho_{\langle \ell_i, \ell_j \rangle}: \ \ \gamma \ \wedge \ \left(\bigwedge_{i=1}^{m} u_i' = e_i\right) \ \wedge \ \pi = \ell_i \ \wedge \ \pi' = \ell_j \ \wedge \ \left(\bigwedge_{v \in (V - \mathtt{var}(\alpha))} v' = v\right)$

Figure 5.3: The concrete phase transition system $\mathcal{S} = (V, \Phi, \Theta, \mathcal{T})$ corresponding to the hybrid automaton, $D = (V_D, L, E, E_\theta, \Omega, \mu, \kappa)$

It is not difficult to construct a CPTS S, corresponding to a given hybrid automaton, and in Figure 5.3 we present this construction. In the figure, $\mathtt{var}(\alpha)$ is the set of variables that get assigned in α (i.e., $\{u_i | 1 \leq i \leq m\}$).

From now on, we restrict our attention to *non-Zeno hybrid automata*, i.e., hybrid automata whose corresponding CPTS's are non-Zeno. As explained in Abadi and Lamport [3], a safety property π is non-Zeno iff every finite behavior satisfying π can be exended to an infinite behavior satisfying π in which time can progress without any bound.

Chapter 6

Diagrammatic Verification: Sampling Semantics

In this chapter, we present a methodology for the verification of temporal properties of hybrid systems. The methodology is based on the deductive transformation of *hybrid diagrams*, which represent the system and its properties. The original hybrid system is represented as a one-vertex diagram and is transformed using a set of rules where the final diagram can be algorithmically checked against the specification. Two classes of rules are presented in this chapter: safety rules for studying safety properties; and justice, compassion, and pruning rules for studying progress properties. The resulting methodology is complete for quantifier-free linear-time temporal logic, and the proof of completeness is presented. The algorithmic check of a hybrid diagram against its specification either gives a positive answer to the verification problem or provides guidance for the further transformation of the diagrams. The transformation rules and the application of guidance is illustrated on a simple room-heater example.

6.1 Why Diagrams?

Hybrid diagrams are related to the *fairness diagrams* of de Alfaro and Manna [49] and to the *hybrid automata* of [10, 9]. They consist of a graph whose vertices are labeled by assertions and whose edges are labeled by transition relations. Associated with each diagram are *fairness constraints* that encode acceptance conditions similar to those of ω-automata. The diagrams represent the system behavior and the safety and progress properties that have been proved about it: the vertex and edge labels represent the safety properties; the fairness constraints represent the progress properties. Hybrid diagrams are sufficiently expressive to encode the phase transition systems of Chapter 5.2, which will be the hybrid system model adopted in this chapter.

The construction of the proof of a temporal specification begins by representing the system as a one-vertex diagram, whose single edge encodes the possible state transitions of the system. This initial diagram can be transformed using a set of rules that preserve the inclusion of system behaviors, producing a chain of diagram transformations. The aim of this process is to obtain a diagram that can be shown to satisfy the specification by purely algorithmic means.

After any number of transformations, an algorithmic procedure can be applied to the last diagram: it either establishes that the final diagram (and, by behavior inclusion, the original PTS) satisfies the specification, or it returns a set of *candidate counterexample paths* (CCP) in the diagram. The CCPs provide guidance for the extension of the chain of transformations, following the insights of Sipma, Uribe, and Manna [144]. Additionally, the CCPs can be used to guide the search for counterexamples, by directing the simulation of the original system along the CCPs.

There are four rules to transform diagrams. The *simulation rule* modifies the graph structure of the diagram, enabling the study of safety properties [49]. The *justice* and *compassion rules* prove progress properties of the diagrams and represent them as additional fairness constraints. The *pruning rule* eliminates portions of the diagram that are never traversed by any computation along which time diverges. These rules generate first-order verification conditions that must be proved to justify the transformation. The justice and compassion rules are one of the main contributions of this chapter and are at the basis of the completeness results of the methodology. By relying on *ranking* and *delay* functions to measure progress towards given goals, the rules enable the proof of justice and compassion properties of the systems; these properties are then represented as fairness constraints which are added to the diagrams.

While the transformation rules have been presented in their full generality, it is possible to construct libraries that list special cases of the rules that occur frequently in practice and which can be applied with little user intervention. We present two such special cases and illustrate them on an example.

6.1.1 Contributions

The advantages of the proposed methodology over the rule-based approach of [117, 93] include

- a visual representation of the proof process,
- the possibility for proof guidance,
- the incremental construction of a proof,
- the ability to prove specifications expressed by temporal formulas not in canonical form [115], and
- completeness of the methodology.

Diagrams can later serve as documentation for a given system and can be reused when similar proofs are carried out for similar systems [50]. In addition, incomplete or failed proofs can alert the engineer to potential counterexamples.

6.2 Hybrid Diagrams

To study the temporal behavior of a PTS, we introduce *hybrid diagrams*, derived from the *fairness diagrams* of [49]. A *hybrid diagram* (diagram, for short) $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ consists of the following components.

1. A set $\mathcal{V}$ of typed state variables that includes the master clock T. A *state* is a type-consistent interpretation of all the variables in $\mathcal{V}$; the *state space* S of the diagram is the set of all such variable interpretations.

2. A set V of *vertices*.

3. A labeling ρ that assigns to each vertex $v \in V$ an assertion $\rho(v)$ over $\mathcal{V}$. A *location* of a diagram is a pair (v, s) such that $v \in V$ and $s \models \rho(v)$. It represents an instantaneous configuration of the diagram.

4. A labeling θ that assigns to each vertex $v \in V$ an initial assertion $\theta(v)$ over $\mathcal{V}$. This labeling defines the set of initial locations $\{(v, s) \mid v \in V \text{ and } s \models \theta(v)\}$. For all $v \in V$, we require that $\theta(v) \rightarrow T = 0$.

5. A labeling τ that assigns to each edge $(u, v) \in V \times V$ a transition assertion $\tau(u, v)$ over $\mathcal{V} \cup \mathcal{V}'$ and Δ. For $u, v \in V$, assertion $\tau(u, v)$ represents the possible state changes of the system when going from vertex u to vertex v by a time-step of duration $\Delta \in \mathcal{R}^+$. We require that the assertion $\tau(u, v) \rightarrow T' = T + \Delta$ holds for all $u, v \in V$.

6. A set $\mathcal{J}$ of *justice constraints* and a set $\mathcal{C}$ of *compassion constraints*. The elements of $\mathcal{J}$ and $\mathcal{C}$ are pairs (R, G) where $R \subseteq V$ and $G \subseteq V \times V$.

The justice and compassion constraints, collectively called *fairness constraints*, represent fairness properties that have been proved about the system. For a constraint (R, G), the set $R \subseteq V$ specifies a *request* region. The request is *gratified* when a transition from a vertex u to a vertex v is taken, with $(u, v) \in G$. A justice constraint indicates that a request that is performed without interruptions will eventually lead to gratification; a compassion constraint indicates that a request performed infinitely often will be gratified infinitely often [117, 49].

Given an assertion φ over $\mathcal{V}$, we denote by φ' the formula obtained by replacing each free $x \in \mathcal{V}$ by $x' \in \mathcal{V}'$. A *run* of a diagram is an infinite sequence of locations (v_0, s_0), (v_1, s_1), (v_2, s_2), ..., satisfying the following conditions:

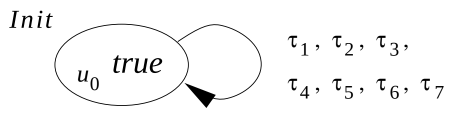

Figure 6.1: Hybrid Diagram A_0.

Initiality: $s_0 \models \theta(v_0)$.

Vertex Labels: for all $i \geq 0$, $s_i \models \rho(v_i)$ (this condition is implied by the fact that (s_i, v_i) is a location).

Edge Labels: for all $i \geq 0$, $(s_i, s_{i+1}) \models \exists \Delta \,.\, \tau(v_i, v_{i+1})$.

Time Progress: for each $t \in \mathcal{R}$ there is $i \in \mathbb{N}$ such that $s_i(T) \geq t$.

Justice: for each constraint $(R, G) \in \mathcal{J}$, if there is $k \in \mathbb{N}$ such that $v_i \in R$ for all $i \geq k$, then there is $j \geq k$ such that $(v_j, v_{j+1}) \in G$.

Compassion: for each constraint $(R, G) \in \mathcal{C}$, if $v_i \in R$ for infinitely many $i \in \mathbb{N}$, then there are infinitely many $j \in \mathbb{N}$ such that $(v_j, v_{j+1}) \in G$.

If $\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$ is a run of A, the sequence of states $s_0, s_1, s_2, \ldots$ is a *computation* of A. We denote by *Runs*(A) and $\mathcal{L}(A)$ the sets of runs and computations of A, respectively.

We note that the above definition of diagram run differs slightly from the one presented in de Alfaro, Kapur, and Manna [47]. Here, we have directly the condition $s_i \models \rho(v_i)$ for $i \geq 0$, instead of requiring that the diagram satisfies $[\rho(u) \wedge \tau(u, v)] \rightarrow \rho'(v)$ for all $u, v \in V$ (the so-called *consecution requirement*). Thus, when a state transition (s_i, s_{i+1}) occurs along a computation, it must satisfy the formula $\rho(v_i) \wedge \tau(v_i, v_{i+1}) \wedge \rho'_{i+1}$. The current choice often enables us to draw diagrams with edges labeled by simpler assertions, yielding a more concise graphical representation. We define the abbreviation

$$\widehat{\tau}(u, v) \overset{\text{def}}{=} \rho(u) \wedge \tau(u, v) \wedge \rho'(v)$$

denoting the possible state changes corresponding to the traversal of edge (u, v).

Every PTS can be represented by a one-vertex diagram, as the following construction shows.

Construction 1 Given a PTS $\mathcal{S} = (\mathcal{V}, \theta, \mathcal{T}, \Pi, \mathcal{A})$, we define the diagram $hd(\mathcal{S}) = (\mathcal{V}, V, \rho, \tilde{\theta}, \tau, \mathcal{J}, \mathcal{C})$ by $V = \{v_0\}$, $\rho(v_0) = \mathit{true}$, $\tilde{\theta}(v_0) = \theta$, $\mathcal{J} = \emptyset$, $\mathcal{C} = \emptyset$, and

$$\tau(v_0, v_0) = \left(\bigvee_{\pi \in \mathcal{T}} (\pi \wedge \Delta = 0) \right) \vee \left(\bigvee_{a \in \mathcal{A}} \mathit{tick}_a[\Delta] \right) . \qquad \blacksquare$$

Theorem 15 *For a* PTS $\mathcal{S}$, $\mathcal{L}(\mathcal{S}) = \mathcal{L}(hd(\mathcal{S}))$.

The proof of Theorem 15 is trivial. In Figure 6.1, we present the initial diagram $A_0 = \mathit{hd}(RH)$ corresponding to system RH. The transitions τ_1, τ_2, and τ_3 are as in RH (with the added conjunct $\Delta = 0$), and transitions τ_4, τ_5, τ_6, and τ_7 are $\mathit{tick}_{a_1}[\Delta]$, $\mathit{tick}_{a_2}[\Delta]$, $\mathit{tick}_{a_3}[\Delta]$, and $\mathit{tick}_{a_4}[\Delta]$, respectively. The single node u_0 is marked *Init* as a reminder that its initial label $\theta(u_0)$ is equal to the initial condition of the PTS.

Hybrid diagrams vs. hybrid automata. Hybrid diagrams are related to *hybrid automata*, a formalism widely adopted for the modeling of hybrid systems and for the study of their temporal properties [10, 35, 9]. While sharing a similar labeled-graph structure, the two formalisms differ in some respects.

In a hybrid automaton, the dynamic behavior of the system and the discrete state-transitions are described by different components: the first by differential equations labeling the vertices, the second by transition relations labeling the edges. In a hybrid diagram, both types of system evolution are described by the traversal of diagram edges. Vertex labels are used to express invariants that hold along diagram computations.

These differences are motivated by the purposes that hybrid automata and hybrid diagrams serve. Hybrid automata were proposed as a formal model of hybrid systems, to which various formal verification methods could be applied. Hybrid diagrams, on the other hand, are meant to provide a deductive representation of a hybrid system and of the safety and progress properties that have been proved about it. They are suited to the application of the diagram transformation rules that will be presented next.

6.2.1 Diagram Transformation Rules

The temporal properties of a PTS are studied by means of *transformation rules* [49]. There are four rules: the *simulation rule*, used to study safety properties; the *justice* and *compassion rules*, used to study progress properties; and the *pruning rule*, used to prune portions of a diagram that are never traversed by runs along which time diverges. If a diagram A can be transformed into a diagram B by one of these rules, we write $A \Rightarrow B$. We indicate by $\overset{*}{\Rightarrow}$ the reflexive transitive closure of $\Rightarrow$. The rules preserve language containment: $A \Rightarrow B$ implies $\mathcal{L}(A) \subseteq \mathcal{L}(B)$. Given a PTS $\mathcal{S}$, the rules are used to construct a chain of transformations $\mathit{hd}(\mathcal{S}) \equiv A_0 \Rightarrow A_1 \Rightarrow \cdots \Rightarrow A_n$. At any time, it is possible to check algorithmically whether the last diagram of the chain complies with the specification. This test, discussed in the next section, provides a sufficient condition for the diagram to satisfy the specification and returns either a positive answer to the verification problem, or guidance for the extension of the chain of transformations.

Simulation Rule

The *simulation rule*, derived from [49], enables the transformation of a diagram into a new one, such that the second diagram is capable of simulating the first one. A simulation relation between two

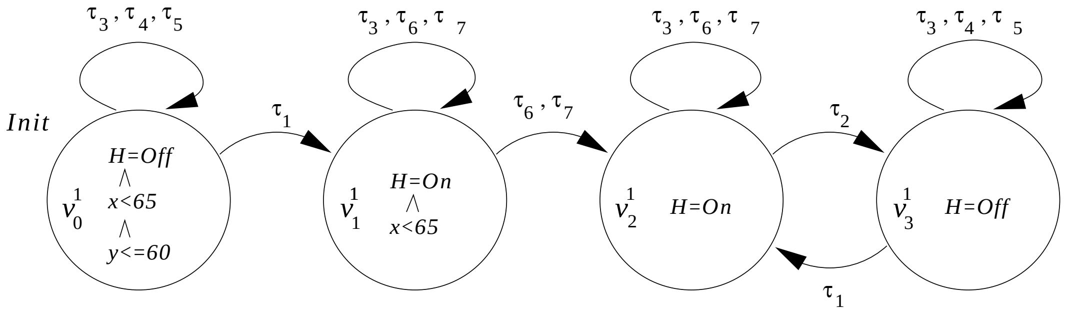

Figure 6.2: Hybrid Diagram A_1. Edges labeled with *false* are not shown.

diagrams A_1 and A_2 is induced by a function $\mu\colon V_1 \mapsto 2^{V_2}$ from the vertices of A_1 to those of A_2.

Rule 1 (Simulation) Let $A_1 = (\mathcal{V}, V_1, \rho_1, \theta_1, \tau_1, \mathcal{J}_1, \mathcal{C}_1)$ and $A_2 = (\mathcal{V}, V_2,\ \rho_2,\ \theta_2,\ \tau_2,\ \mathcal{J}_2,\ \mathcal{C}_2)$ be two diagrams sharing the same variables. If there is a function $\mu : V_1 \mapsto 2^{V_2}$ that satisfies the conditions below, then $A_1 \Rightarrow A_2$.

1. For all $u \in V_1$, $\theta_1(u) \wedge \rho_1(u) \rightarrow \bigvee_{v \in \mu(u)} (\theta_2(v) \wedge \rho_2(v))$.
2. For all $u, u' \in V_1$ and $v \in \mu(u)$, $\left(\widehat{\tau}_1(u, u') \wedge \rho_2(v)\right) \rightarrow \bigvee_{v' \in \mu(u')} \widehat{\tau}_2(v, v')$.
3. For each $(R_2, G_2) \in \mathcal{J}_2$ (resp. $\in \mathcal{C}_2$) there is $(R_1, G_1) \in \mathcal{J}_1$ (resp. $\in \mathcal{C}_1$) such that:
   (a) for all $u \in V_1$, if $\mu(u) \cap R_2 \neq \emptyset$ then $u \in R_1$;
   (b) for all $(u, u') \in G_1$ and $v \in \mu(u)$,
   $$\widehat{\tau}_1(u, u') \wedge \rho_2(v) \quad \rightarrow \quad \bigvee_{v' \in H(u', v)} \widehat{\tau}_2(v, v')\ ,$$
   where $H(u', v) = \{v' \mid v' \in \mu(u') \wedge (v, v') \in G_2\}$. ∎

Theorem 16 (soundness of Rule 1) *If $A_1 \Rightarrow A_2$ by Rule 1, then $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$.*

The proof of Theorem 16 is straightforward.

By applying the simulation rule to the diagram A_0 of Figure 6.1, we obtain the diagram A_1 presented in Figure 6.2. The application of the rule is based on the function μ defined by $\mu(u_0) = \{v_0^1, v_1^1, v_2^1, v_3^1\}$. In Figure 6.2, v_0^1 is the only vertex satisfying the initial condition specified by θ.

Simulation Rule: Special Cases

Some instances of simulation transformations are used with particular frequency in proofs and deserve special mention. Two special cases of the simulation rule arise in practice: *vertex-split*, which splits a node of the diagram into several nodes enabling us to further analyze the system, and vertex-strengthen, which strengthens the labeling of a vertex of the diagram, thereby allowing us to prove a safety property of the system. We first present the vertex-split rule.

Rule 2 (Vertex-Split) Let $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ be a diagram, $v \in V$ a vertex, $U = \{u_1, \ldots, u_n\}$ a set of new vertices (in which we wish to split v), and $\psi_1, \ldots, \psi_n$ a set of formulas over $\mathcal{V}$ such that $\bigvee_{i \in [1..n]} \psi_i \equiv \mathit{true}$. We can transform A into a diagram A' obtained as follows:

1. Replace vertex v with the set of vertices U, where for each $1 \leq i \leq n$, $u_i \in U$ is labeled by $\rho(u_i) = \rho(v) \wedge \psi_i$ and $\theta(u_i) = \theta(v)$.

2. For all $i, j \in [1..n]$ and $z \in V - \{v\}$, let

$$\tau(z, u_i) = \tau(z, v) \qquad \tau(u_i, z) = \tau(v, z) \qquad \tau(u_i, u_j) = \tau(v, v) \ .$$

   For every new edge (w, w') thus labeled, if $\widehat{\tau}(w, w') \equiv \mathit{false}$, then set $\tau(w, w') = \mathit{false}$, thereby eliminating the edge from the diagram.

3. For each constraint (R, G), if $v \in R$ then replace v with the set U, and for every edge going to or from v in G, replace the edge with the corresponding new edges going to or from U. ∎

By applying the vertex-split rule twice to the diagram A_1 of Figure 6.2, we obtain the diagram A_2 presented in Figure 6.3. The application of the rule is based on splitting v_1^2 by $\{v_2^2, v_4^2\}$ and splitting v_3^1 by $\{v_3^2, v_5^2\}$ in succession.

The other special case of the simulation rule we consider is the vertex-strengthen rule.

Rule 3 (Vertex-Strengthen) Let $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ be a diagram, $v_1, \ldots, v_n \subseteq V$ a list of vertices whose vertex labels we wish to strengthen, and $\psi_1, \ldots, \psi_n$ a set of formulas over $\mathcal{V}$. If for all $i, j \in [1..n]$

$$\theta(v_i) \rightarrow \psi_i \qquad (\psi_i \wedge \widehat{\tau}(v_i, v_j)) \rightarrow \psi_j' \ ,$$

then we can transform A into diagram $A' = (\mathcal{V}, V, \rho', \theta, \tau, \mathcal{J}, \mathcal{C})$ obtained by defining $\rho'(v_i)$ to be $\rho(v_i) \wedge \psi_i$, for all $i \in [1..n]$ and $\rho'(u) = \rho(u)$ for all $u \in V - \{v_1, \ldots, v_n\}$. ∎

By applying the vertex-strengthen rule to the diagram A_2 of Figure 6.3, we obtain the diagram A_3 presented in Figure 6.4. The application of the rule is based on $\psi_2 = \psi_4 = x \leq 75 - 7 \cdot e^{-y/105} \wedge 65 \leq x \wedge 75$ for vertices v_2^3 and v_4^3, and $\psi_3 = \psi_5 = x \geq 60 + 12 \cdot e^{-y/70} \wedge 65 \leq x \wedge 75$ for vertices v_3^3 and v_5^3. ∎

Progress Rules

The *justice* and *compassion rules* add new constraints to the justice or compassion sets of a diagram, respectively. Since the rules must preserve language containment, it is possible to add a constraint

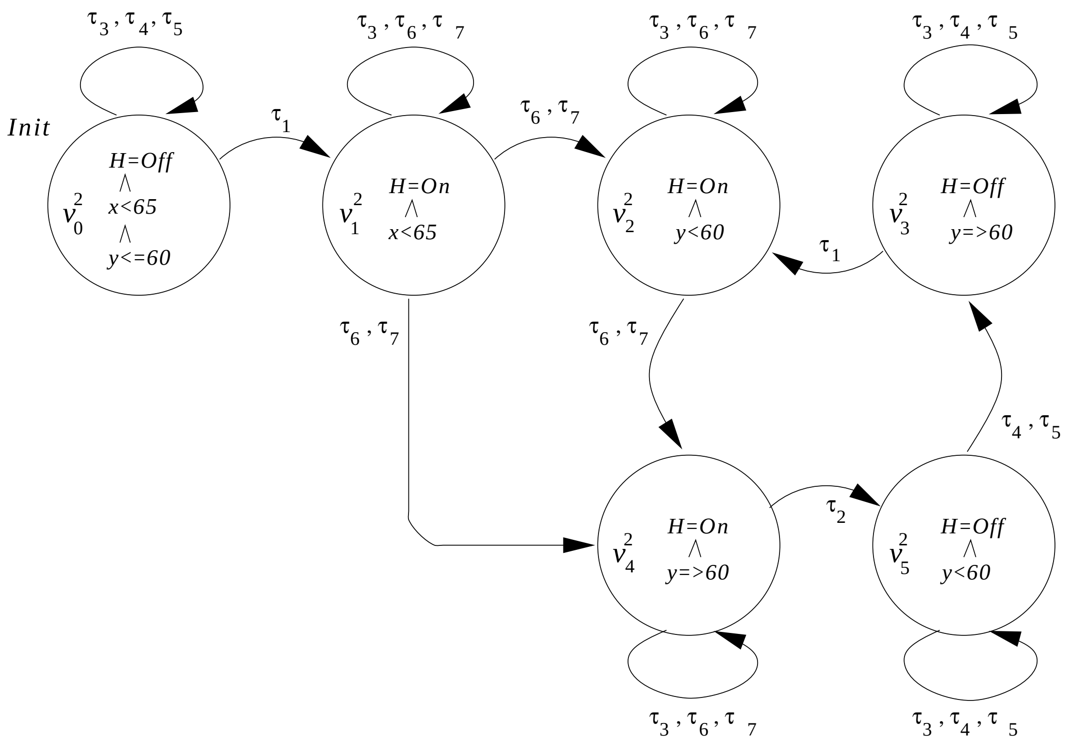

Figure 6.3: Hybrid Diagram A_2. Edges labeled with *false* are not shown.

only if all runs of the diagram already obey it, implying that the constraint represents a progress property of the runs of the diagram. To prove that all runs obey the constraint, the rules rely on *ranking* and *delay* functions to measure progress towards its gratification. The delay functions are similar to the mappings of Lynch and Attiya [112]; our results indicate that to achieve completeness they need to be used in conjunction with ranking functions.

Recall that a *well-founded domain* is a set D together with a relation $>$ such that there is no infinite descending chain $d_0 > d_1 > d_2 > \cdots$ of elements in D.

Given a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$, let $\mathit{loc}(A) = \{(v,s) \in V \times S \mid s \models \rho(v)\}$ denote the set of locations of A. A *ranking function* $\delta\colon \mathit{loc}(A) \mapsto D$ for a diagram A is a function mapping locations of A into elements of a well-founded domain D. A *delay function* $\gamma\colon \mathit{loc}(A) \mapsto \mathcal{R}^+$ is a function mapping locations of A into non-negative real numbers. The ranking and delay functions δ, γ are represented by the families $\{\delta(u)\}_{u \in V}$, $\{\gamma(u)\}_{u \in V}$ of terms on $\mathcal{V}$.

To add a constraint (R, G), the justice rule relies on ranking and delay functions δ, γ. While in R, δ cannot increase unless an edge in G is taken, and γ gives an upper bound to the amount of time before either an edge in G is taken or R is left.

Rule 4 (Justice) Consider a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ and a constraint (R, G) such that $R \subseteq V$ and $G \subseteq V \times V$. Assume that there are ranking and delay functions δ, γ such that, for all

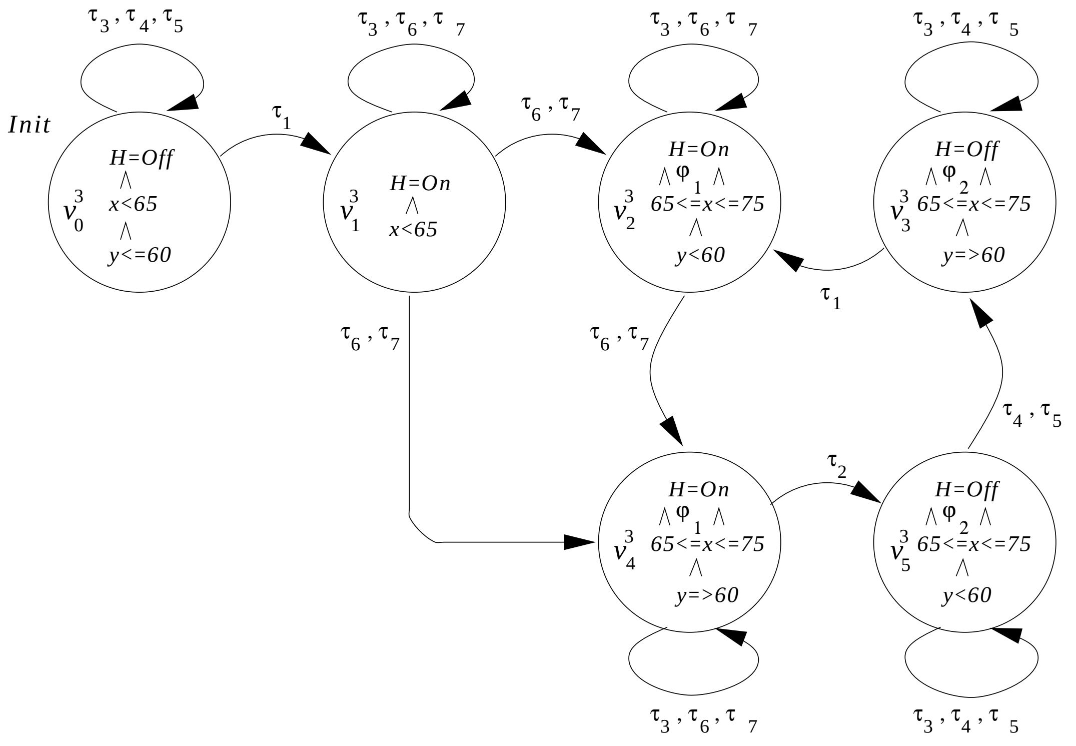

Figure 6.4: Hybrid Diagram A_3, where $\varphi_1 : x \leq 75 - 7 \cdot e^{-y/105}$ and $\varphi_2 : x \geq 60 + 12 \cdot e^{-y/70}$. Edges labeled with *false* are not shown.

$u, v \in R$ with $(u, v) \notin G$, the assertion

$$\widehat{\tau}(u,v) \quad \rightarrow \quad \delta(u) > \delta'(v) \;\vee\; \Big(\delta(u) = \delta'(v) \wedge \gamma(u) \geq \gamma'(v) + \Delta\Big) \tag{6.1}$$

holds. Then, $A \Rightarrow A'$, where $A' = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J} \cup \{(R, G)\}, \mathcal{C})$. ∎

A special case that occurs frequently is when, for every location (v, s) with $v \in R$, there is an upper bound $t_M(v, s)$ for the time before a G-edge is followed. In this case we can take $\gamma(v, s) = t_M(v, s)$ and $\delta(v, s) = 0$, and assertion (6.1) reduces to

$$\widehat{\tau}(u,v) \quad \rightarrow \quad \gamma(u) \geq \gamma'(v) + \Delta\ . \tag{6.2}$$

The function δ is used to cover the case in which there is no upper bound for the time before a G-edge is followed. An example where the ranking function δ is required is presented below. Consider the hybrid diagram in Figure 6.5. We wish to add the justice requirement $(\{v_0, v_1\}, (v_1, v_2))$. While x measures the length of time that the system can stay in vertex v_1 before taking a gratify transition, it does not measure the amount of time that the system can stay in vertex v_0 before taking a gratify transition since x gets reset to an arbitrary value when going from v_0 to v_1. Moreover, there is no *global* upper bound on how long we can stay in vertex v_0 before taking a gratify transition, even

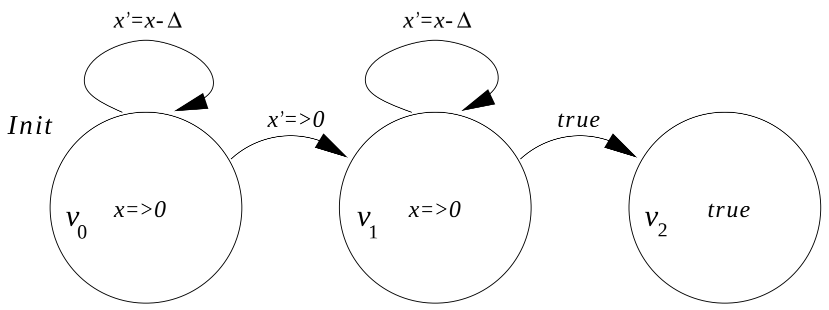

Figure 6.5: Hybrid Diagram illustrating ranking functions. All edges also have the conjunct $T' = T + \Delta$. Edges labeled with *false* are not shown.

though there is a *local* upper bound on how long we can stay in v_0 before taking some transition. Thus, we need to use both a ranking and a delay function to prove the justice requirement. To apply the rule, we use a ranking function defined by $\delta(v_0) = 1$ and $\delta(v_1) = \delta(v_2) = 0$ and a delay function defined by $\gamma(v_0) = \gamma(v_1) = x$ and $\gamma(v_2) = 0$.

Returning to our Room-Heater example, to show that the temperature eventually reaches the desired range, we apply Rule 4 to the diagram A_3 of Figure 6.4, adding the justice constraint $(\{v_0^3, v_1^3\}, \{(v_1^3, v_2^3), (v_1^3, v_4^3)\})$. We denote the resulting diagram by A_4. This constraint shows that a run of A_3 cannot stay forever in v_0^3 or v_1^3, and must eventually proceed to either v_2^3 or v_4^3. The rule uses a ranking function defined by $\delta(v_0^3) = 1$ and $\forall i \in [1..5]$, $\delta(v_i^3) = 0$. The delay function is given by

$$\begin{aligned}
\gamma(v_0^3) = {} & 60 - y \ , \\
\gamma(v_1^3) = {} & \textit{if } \ x \leq 60 \ \textit{ then } \ 175 + 105 \ln((75 - x)/49) \\
& \quad \textit{else } \ 150 + 70 \ln((70 - x)/10) \ , \\
\forall i \in [2..5], \gamma(v_i^3) = {} & 0 \ .
\end{aligned}$$

We point out that while we use both a rank and a delay function, it is possible to use the rule with only a delay function. We chose to use both functions as it makes the presentation simpler.

Theorem 17 (Soundness of Rule 4) *If a constraint (R, G) is added by Rule 4 to the justice set of a diagram A, producing diagram A', then Runs(A) = Runs(A') and $\mathcal{L}(A) = \mathcal{L}(A')$.*

Proof of 17:

Since A' has more constraints than A, it is immediate that *Runs*$(A') \subseteq$ *Runs*(A). To show the reverse containment, assume towards the contradiction that the conditions of Rule 4 are satisfied and that there is a run $\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots \in$ *Runs*(A) that does not satisfy (R, G). By definition, there is $k \in \mathbb{N}$ such that $v_i \in R$, $(v_i, v_{i+1}) \notin G$ for all $i \geq k$. By Condition (6.1) the value of $\delta(v_i, s_i)$ does not increase for $i \geq k$. Since the domain of δ is well-founded, this value cannot decrease infinitely often, and there must be $k' \geq k$ such that $\delta(v_i, s_i)$ is constant for $i \geq k'$. For $m \geq k'$, let $\xi_m = s_m[\![T]\!] - s_{k'}[\![T]\!]$. From Condition (6.1) it is easy to prove by

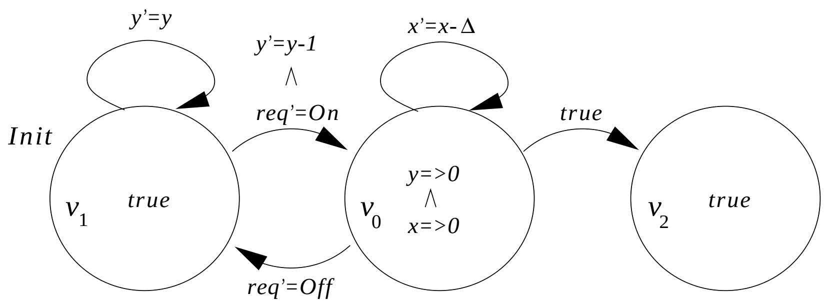

Figure 6.6: Hybrid Diagram illustrating compassion rule. All edges also have the conjunct $T' = T + \Delta$. Edges labeled with *false* are not shown.

induction on $m \geq k'$ that $\xi_m \leq \gamma(v_{k'}, s_{k'}) - \gamma(v_m, s_m)$. Since $\lim_{m\to\infty} \xi_m = \infty$ because of the divergence of time, we have $\lim_{m\to\infty} \gamma(v_m, s_m) = -\infty$, contradicting the non-negativity of γ. $\Rightarrow\!\Leftarrow$

So our assumption is false, and $Runs(A) \subseteq Runs(A')$ as desired. ∎

Rule 5 (Compassion) Given a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ and a constraint (R, G) such that $R \subseteq V$ and $G \subseteq V \times V$. Assume that there is a ranking function δ and a delay function γ such that, for every $u, v \in V$ with $(u, v) \notin G$, the following conditions hold:

$$\widehat{\tau}(u,v) \rightarrow \delta(u) \geq \delta'(v) \tag{6.3}$$

$$\text{If } u \notin R, v \in R: \quad \widehat{\tau}(u,v) \rightarrow \delta(u) > \delta'(v) \tag{6.4}$$

$$\text{If } u \in R, v \in R: \quad \widehat{\tau}(u,v) \rightarrow \delta(u) > \delta'(v) \ \vee\ \gamma(u) \geq \gamma'(v) + \Delta \tag{6.5}$$

Then $A \Rightarrow A'$, where $A' = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C} \cup \{(R, G)\})$. ∎

Consider the hybrid diagram in Figure 6.6. The diagram represents a system where a request can be made to do some action, modeled by variable *req*. This request can be undone at any time, modeled by entry into vertex v_1, and then remade at a later time, modeled by entry into vertex v_0. However, only a finite number of requests can be made, modeled by variable y.

We would like to prove that if we are infinitely often in vertex v_0, then we will eventually reach vertex v_2, where the edge (v_0, v_2) represents the gratification of the request. That is, we wish to add the compassion requirement $(\{v_0\}, (v_0, v_2))$. To apply the rule, we use a ranking function defined by $\delta(v_0) = \delta(v_1) = y$ and $\delta(v_2) = 0$, and a delay function defined by $\gamma(v_0) = x$ and $\gamma(v_1) = \gamma(v_2) = 0$.

Theorem 18 (Soundness of Rule 5) *If a constraint (R, G) is added by Rule 5 to the compassion set of a diagram A, producing diagram A', then $Runs(A) = Runs(A')$, and therefore $\mathcal{L}(A) = \mathcal{L}(A')$.*

Proof of 18:

Again, it is immediate that $Runs(A') \subseteq Runs(A)$. To show the reverse containment, assume towards the contradiction that the conditions of Rule 5 are satisfied and that there is a run

$\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots \in \mathit{Runs}(A)$ that visits infinitely often R taking only finitely many edges in G. Thus, there is $k \in \mathbb{N}$ such that $(v_i, v_{i+1}) \notin G$ for all $i \geq k$. By (6.3), the value of δ does not increase along σ after k.

If σ visits $V - R$ infinitely often, it must infinitely often return to R, and by (6.5) the value of δ beyond k decreases infinitely often, violating the hypothesis that the domain of δ is well-founded. Thus, there is $m \geq k$ such that σ stays in R forever beyond position m. Once σ is confined to R, the proof follows that of Theorem 17, because of the similarity between assertions (6.5) and (6.1). ∎

Pruning Rule

The *pruning rule* prunes from a diagram a subset of vertices that, because of the presence of a justice constraint, cannot appear in any run of the system.

Rule 6 (Pruning) Let $A_1 = (\mathcal{V}, V_1, \rho_1, \theta_1, \tau_1, \mathcal{J}_1, \mathcal{C}_1)$ be a diagram and let $U_1 \subseteq V_1$ be a subset of its vertices such that the following two conditions hold:

1. there is $(R_1, G_1) \in \mathcal{J}_1$ such that $U_1 \subseteq R_1$, $(U_1 \times V_1) \cap G_1 = \emptyset$;
2. for all $u \in U_1$ and $v \in V_1 - U_1$, $\widehat{\tau}_1(u, v) \equiv \mathit{false}$.

Then $A_1 \Rightarrow A_2$, where $A_2 = (\mathcal{V}, V_2, \rho_2, \theta_2, \tau_2, \mathcal{J}_2, \mathcal{P}_2)$ is obtained as follows:

1. $V_2 = V_1 - U_1$;
2. ρ_2, θ_2, τ_2 are obtained by restricting the domain of ρ_1, θ_1, and τ_1 to V_2, V_2, and $V_2 \times V_2$, respectively;
3. for each constraint $(R, G) \in \mathcal{J}_1$ (resp. $\in \mathcal{C}_1$), we insert the constraint $(R \cap V_2, G \cap (V_2 \times V_2))$ into $\mathcal{J}_2$ (resp. into $\mathcal{C}_2$). ∎

This rule can be used in conjunction with Rule 4 to prune from the diagram vertices reached only by invalid runs along which time does not diverge. The soundness of the rule follows from the observation that, if the conditions of the rule are satisfied, no run of the diagram can contain vertices in U. In fact, if a run entered U, it would not be able to leave it, and by staying forever in U it would violate at least one justice constraint of the diagram.

6.2.2 Proving Temporal Properties

In this section we present an algorithm to check whether a diagram satisfies a specification written in the linear-time temporal logic TL_s. The formulas of TL_s are obtained by combining first-order logic formulas by means of the future temporal operators $\Box$ (always), $\Diamond$ (eventually), $\mathcal{U}$ (until), and the corresponding past ones $\boxminus$, $\diamondminus$ and $\mathcal{S}$ [117].[1] Given a diagram A and a formula $\varphi \in TL_s$, the algorithm provides either a positive answer to $A \models \varphi$ or information about the region of the diagram that can contain a counterexample to φ. This information can be used as guidance for the extension of the chain of transformations. The first step of the algorithm consists in constructing a Streett automaton $N_{\neg\varphi}$ that accepts all the computations that do not satisfy φ. The automaton is a first-order version of a classical Streett automaton [139].

A (first-order) *Streett automaton* N consists of the components $(\mathcal{V}, (V, E), \rho, Q, \mathcal{B})$, where $\mathcal{V}$, ρ are as in hybrid diagrams; (V, E) is a directed graph with set of vertices V and set of edges $E \subseteq V \times V$; $Q \subseteq V$ is the set of *initial vertices*, and $\mathcal{B}$, called the *acceptance list*, is a set of pairs (P, R) such that $P \subseteq V$ and $R \subseteq V$. A *run* σ of N is an infinite sequence of locations $(v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$ such that $v_0 \in Q$, and:

1. for all $i \geq 0$, $s_i \models \rho(v_i)$ and $(v_i, v_{i+1}) \in E$;
2. for each pair $(P, R) \in \mathcal{B}$, either $v_i \in R$ for infinitely many $i \in \mathbb{N}$, or there is $k \in \mathbb{N}$ such that $v_i \in P$ for all $i \geq k$.

If $\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$ is a run of N, the sequence of states $s_0, s_1, s_2, \ldots$ is a *computation* of N. The set of runs (resp. computations) of a Streett automaton N is denoted by $\mathit{Runs}(N)$ (resp. $\mathcal{L}(N)$).

To show that no behavior of A satisfies $\neg\varphi$, the algorithm constructs the *graph product* $A \otimes N_{\neg\varphi}$ and checks that no infinite path in it corresponds to a computation of both A and $N_{\neg\varphi}$. The construction of the graph product relies on a terminating proof procedure $\vdash$ for the first-order language used in the specification and in the labels of the diagram. The procedure $\vdash$ should be able to prove a subset of the valid sentences that includes all substitution instances of propositional tautologies. Given a first-order formula ψ, we write $\vdash \psi$ or $\not\vdash \psi$ depending on whether $\vdash$ terminates with or without a proof of ψ, respectively.

Construction 2 (Graph Product) Given diagram $A = (\mathcal{V}, U, \rho_A, \theta, \tau, \mathcal{J}, \mathcal{C})$ and Streett automaton $N_{\neg\varphi} = (\mathcal{V}, (V, E), \rho_N, Q, \mathcal{B})$, the *graph product* $A \otimes N_{\neg\varphi} = (W, Z, H)$ consists of a graph (W, H) and of a set of initial vertices $Z \subseteq W$. It is defined by:

1. $W = \{(u, v) \in U \times V \mid \not\vdash \neg(\rho_A(u) \wedge \rho_N(v))\}$;
2. $Z = \{(u, v) \in W \mid v \in Q \ \mathit{and} \ \ \not\vdash \neg(\theta(u) \wedge \rho_N(v))\}$;
3. $$H = \Big\{ \Big((u_1, v_1), (u_2, v_2)\Big) \in W \times W \ \Big| \ (v_1, v_2) \in E \ \mathit{and} \ \not\vdash \neg\Big(\widehat{\tau}(u_1, u_2) \wedge \rho_N(v_1) \wedge \rho'_N(v_2)\Big) \Big\}. \qquad \blacksquare$$

[1]This logic differs slightly from Manna and Pnueli [117] and our own approach of [47], where the $\bigcirc$ (next) and $\ominus$ (previous) operators are also used. We omit these two operators since we would like our results in this chapter to extend to the continuous semantics presented in Chapter 7, and both $\bigcirc$ and $\ominus$ have problematic interpretations under a continuous semantics.

To show that there is no infinite path in the product that corresponds to a computation of both A and $N_{\neg\varphi}$, we check that every infinite path in (W, H), starting from Z, violates either a constraint of A or a pair in the acceptance list of $N_{\neg\varphi}$. To this end, consider a *strongly connected subgraph* (SCS) $X \subseteq W$ of the graph (W, H). We say that X is *admissible* if the following conditions hold:

1. for all $(R, G) \in \mathcal{J}$, if $X \subseteq R \times V$ then there are $(u_1, v_1), (u_2, v_2) \in X$ such that $(u_1, u_2) \in G$ and $((u_1, v_1), (u_2, v_2)) \in H$;
2. for all $(R, G) \in \mathcal{C}$, if $X \cap (R \times V) \neq \emptyset$ then there are $(u_1, v_1), (u_2, v_2) \in X$ such that $(u_1, u_2) \in G$ and $((u_1, v_1), (u_2, v_2)) \in H$;
3. for all $(P, R) \in \mathcal{B}$, if $X \not\subseteq (U \times P)$ then $X \cap (U \times R) \neq \emptyset$.

The following theorem states that if there are no reachable admissible SCSs in the products, then $A \models \varphi$. This check can be done in time polynomial in $|W|$ using efficient graph algorithms.

Theorem 19 (Diagram Checking) *Given a diagram A and a specification $\varphi \in TL_s$, let $A \otimes N_{\neg\varphi} = (W, Z, H)$. If all SCSs of (W, H) that are reachable in (W, H) from Z are not admissible, then $A \models \varphi$.*

The proof is based on the classical arguments presented by Manna and Pnueli in [115]. The following theorem states that the verification methodology presented in this chapter is complete.

Theorem 20 (Completeness for TL_s) *Given a* PTS *$\mathcal{S}$ and a specification $\varphi \in TL_s$, if $\mathcal{S} \models \varphi$ then there is a chain of transformations $hd(\mathcal{S}) \overset{*}{\Rightarrow} A$ such that $A \models \varphi$ can be proved using Theorem 19.*

Proof of 20:

From φ, it is possible to obtain a deterministic Streett automaton M_φ such that $\mathcal{L}(M_\varphi) = \mathcal{L}(\varphi)$. By the methods of de Alfaro and Manna [49], M_φ can easily be translated into a deterministic diagram $hd(M_\varphi)$. Since $\mathcal{S} \models \varphi$, by Theorem 26 it is possible to construct a chain of transformations $hd(\mathcal{S}) \overset{*}{\Rightarrow} hd(M_\varphi)$. It is easy to see that the graph product $hd(M_\varphi) \otimes N_{\neg\varphi}$ does not contain any connected subgraph that is both reachable and admissible. ∎

Note that while this completeness result has been achieved by relying on a terminating and incomplete proof procedure $\vdash$ for the construction of the graph product, the proof of the assertions arising from the transformations requires general first-order reasoning.

Obtaining Guidance

The presence of admissible and reachable SCSs in the product graph can be used to guide the further analysis of the system, following the insights of Sipman, Uribe, and Manna [144]. Given an admissible and reachable SCS X of $(W, Z, H) = A \otimes N_{\neg\varphi}$, let $X_r \subseteq W$ be the set of vertices that can appear along a path from Z to X in (W, H). Consider the projections $Y = \{u \mid (u, v) \in X\}$, $Y_r = \{u \mid (u, v) \in X_r\}$ of X and X_r, respectively, onto the diagram A. We say that Y_r and Y constitute a *candidate counterexample path* (CCP) in A. The CCPs correspond to regions of the diagram that can contain counterexamples: if a run $\sigma \in \mathit{Runs}(A)$ violates φ, there must be a CCP Y_r, Y such that σ first follows Y_r until it reaches Y and then remains in Y forever while visiting all vertices of Y infinitely often.

The information provided by the CCPs can be used either to guide the search for a counterexample or to extend the chain of transformations to show that no counterexample is contained in the CCPs.

Search for counterexample. Given a CCP Y_r, Y, it may be possible to prove that there is a behavior shared by the diagram A and the original PTS $\mathcal{S}$ that follows Y_r and then remains in Y forever, visiting all vertices of Y infinitely often. The existence of such a behavior would establish $\mathcal{S} \not\models \varphi$.

Alternatively, the CCPs can be used to guide the simulation of the behavior of $\mathcal{S}$ by simulating $\mathcal{S}$ along the CCPs.

Search for proof. The CCPs provide guidance for the extension of the chain of transformations. The aim of the additional transformations is to show that for every CCP Y_r, Y:

- either there is no path in Y_r from Z to Y;
- or, after following Y_r, a computation cannot remain in Y forever and visit all the vertices of Y infinitely often.

To show that there is no path in Y_r from Z to Y, it is possible to use the simulation rule to strengthen the assertions of the edges and vertices along Y_r until the path is interrupted by labeling some edge or vertex with *false*. To show that a computation cannot stay in Y forever and visit all vertices of Y infinitely often, the simulation rule can be used to strengthen the labels of vertices and split vertices into new vertices, thus analyzing in more detail the structure of the SCS Y and possibly splitting it into several SCSs. The justice and compassion rules can be used to show that the system cannot stay forever in Y or infinitely often in some subsets of Y.

Using the algorithm presented in this section, it is possible to check that diagram A_3 of Figure 6.4 satisfies the specification $(65 \leq x \leq 75) \Rightarrow \Box(65 \leq x \leq 75)$.

On the other hand, if we check A_3 against the specification $\Diamond(65 \leq x \leq 75)$, we obtain two CCPs, corresponding to the SCSs $\{v_0^3\}$, $\{v_1^3\}$. To prove the specification, we must thus show that either v_0^3 and v_1^3 are not reachable, which evidently is not possible, or that a run cannot be forever confined to v_0^3 or v_1^3. These conditions can be shown by adding the justice constraint $(\{v_0^3, v_1^3\}, \{(v_1^3, v_2^3)\})$. The diagram-checking algorithm shows that the resulting diagram A_4 satisfies $\Diamond(65 \leq x \leq 75)$.

6.3 Completeness

In this section, we present a completeness result that establishes the existence of chains of transformations from diagrams of PTSs to deterministic diagrams, provided that language containment holds between the two diagrams. This result will be used to establish the completeness of the methodology for proving linear-temporal logic specifications in which no quantifier appears in the scope of a temporal operator.

The completeness results we present for the transformation rules are relative to some assumptions about the expressive power of the assertion language and to the existence of an oracle that is able to prove all valid assertions. In particular, following Manna and Pnueli [115] we assume that the language contains predicate calculus and is powerful enough to represent records of values and lists of values and records. Moreover, the language includes all interpreted function and relation symbols that occur in the labeling of diagrams, in addition to the usual mathematical function and relation symbols (e.g. $+, \cdot, \exp, \ldots$) interpreted over the integers. Finally, the language is augmented with the least and greatest fixpoint operators, necessary to enable the encoding of some auxiliary assertions [115]. To state the completeness results for Rules 1 and 4, we introduce the following definitions.

Consider a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$.

1. A *run prefix* of A is a finite sequence $(v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots, (v_n, s_n)$ of locations of A satisfying the Initiality, Vertex label, and Edge label conditions of diagram computations.

2. We say that A is *non-livelocking* if every run prefix of A can be extended to a run of A.

3. We say that A is *globally reachable* if for every location $(v, s) \in loc(A)$ there is a run prefix $(v_0, s_0), \ldots, (v_n, s_n)$ ending with $(v_n, s_n) = (v, s)$.

4. We say that A is *deterministic* if $\theta(u) \wedge \theta(v) \leftrightarrow \mathit{false}$ and $\widehat{\tau}(u, v) \wedge \widehat{\tau}(u, w) \leftrightarrow \mathit{false}$ for all $u, v, w \in V$.

5. We say that a constraint (R, G) is *J-compatible* (resp. *C-compatible*) with a diagram A if its addition to the justice (resp. compassion) set of A does not change the set of runs of A. Collectively, J-compatible constraints and C-compatible constraints are called *compatible constraints*. The intuition behind this definition is that if a constraint is J- or C-compatible with a diagram, it expresses a progress property obeyed by the diagram.

6.3.1 Justice

Our first theorem states that if a diagram is totally reachable and has no constraints, then every J-compatible constraint can be added with a single application of Rule 4.

Theorem 21 (Completeness, Justice) *If a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ is globally reachable, and (R, G) is J-compatible with $A' = (\mathcal{V}, V, \rho, \theta, \tau, \emptyset, \emptyset)$, then (R, G) can be added to the justice set of A with one application of Rule 4.*

Proof of 21:

The proof is based on the completeness proofs for verification rules presented in [115]. For $n \geq 0$, we call a finite sequence of locations $(v_0, s_0), (v_1, s_1), \ldots, (v_n, s_n)$ a *j-path* if all its vertices are contained in R and for $0 \leq i \leq n$, $(v_i, v_{i+1}) \notin G$ and $(s_i, s_{i+1}) \models \exists \Delta \,.\, \tau(v_i, v_{i+1})$. Let $\widetilde{R} = \{(v, s) \in \mathit{loc}(A) \mid v \in R\}$, and define the relation $\sqsubset \subseteq \widetilde{R} \times \widetilde{R}$ by

$$(v, s) \sqsubset (v', s') \quad \textit{iff} \quad s'[\![T]\!] - s[\![T]\!] \geq 1 \textit{ and there is a j-path from } (v, s) \textit{ to } (v', s').$$

The relation $\sqsubset$ is well-founded on $\widetilde{R}$. To prove $\sqsubset$ is well founded, assume towards the contradiction that there is an infinite descending chain $\ell_0 \sqsubset \ell_1 \sqsubset \ell_2 \sqsubset \cdots$ of locations of $\widetilde{R}$. Since the diagram is globally reachable, there is a j-path η_0 from an initial location to ℓ_0, and for all $i \in \mathbb{N}$, there is a j-path η_{i+1} from ℓ_i to ℓ_{i+1} of temporal duration at least 1, since $\ell_i \sqsubset \ell_{i+1}$. Thus, the infinite path $\eta_0, \eta_1, \eta_2, \ldots$ is a run of diagram A_1 that stays forever in $\widetilde{R}$ while never following an edge in G, contradicting the J-compatibility of (R, G). $\Rightarrow\!\Leftarrow$

Using the results of [115, 104], we can define from $\sqsubset$ a ranking function $\delta \colon \widetilde{R} \mapsto \mathit{Ord}$, where Ord is the set of ordinals, having the following properties for $\ell, \ell', \ell'' \in \widetilde{R}$:

1. $\ell \sqsubset \ell' \to \delta(\ell) > \delta(\ell')$;
2. if $\ell' \sqsubset \ell'' \to \ell \sqsubset \ell''$ for all ℓ'', then $\delta(\ell) \geq \delta(\ell')$.

If there is a j-path from ℓ to ℓ' in $\widetilde{R}$, then $\delta(\ell) \geq \delta(\ell')$. In fact, let η be the j-path, and consider any location $\ell'' \in \widetilde{R}$. If $\ell' \sqsubset \ell''$, there must be a j-path η' from ℓ' to ℓ'' of duration at least 1, and because of the existence of j-path $\eta\eta'$, we have $\ell \sqsubset \ell''$. Property 2 then leads to the desired conclusion.

Given a j-path $\eta = (v_0, s_0), (v_1, s_1), \ldots, (v_n, s_n)$, we define $\mathrm{len}(\eta) = s_n[\![T]\!] - s_0[\![T]\!]$, and we say that η is *level* if $\delta(v_0, s_0) = \delta(v_1, s_1) = \cdots = \delta(v_n, s_n)$. Given $\ell \in \widetilde{R}$, we denote by $E(\ell)$ the set of level j-paths from ℓ, and we define

$$\gamma(\ell) = \sup_{\eta \in E(\ell)} \mathrm{len}(\eta) \ .$$

To see that $\gamma : \widetilde{R} \mapsto \mathcal{R}^+$ is well-defined, note that if $\eta : \ell_0, \ell_1, \ldots, \ell_n$ is a level j-path, then $\mathrm{len}(\eta) < 1$, for otherwise $\ell_0 \sqsubset \ell_n$, leading to $\delta(\ell_0) > \delta(\ell_n)$ by Property 1 above. The functions δ, γ can then be extended to $\mathit{loc}(A)$ by assigning an arbitrary value to locations not in $\widetilde{R}$.

We claim that the functions δ and γ satisfy assertion (6.1) for all $u, v \in R$ such that $(u,v) \notin G$. To see this fact, consider two locations $\ell = (u,s) \in \widetilde{R}$ and $\ell' = (v,t) \in \widetilde{R}$ and assume that $(s,t) \models \exists\Delta \,.\, \tau(u,v)$. Let $d = t[\![T]\!] - s[\![T]\!]$ be the value of parameter Δ that satisfies the quantifier. We have $\delta(\ell) \geq \delta(\ell')$, since there is a j-path from ℓ to ℓ'. If $\delta(\ell) > \delta(\ell')$, assertion (6.1) holds. Else, $\delta(\ell) = \delta(\ell')$, and for any $\eta \in E(\ell')$, the j-path ℓ, η is also level, and $\mathrm{len}(\ell,\eta) = \mathrm{len}(\eta) + d$. Thus, $\gamma(\ell) \geq \gamma(\ell') + d$ and again assertion (6.1) holds.

The functions δ and γ may not be expressible directly in the assertion language available. However, by using the methods of Manna and Pnueli [115], it is possible to define alternative functions $\widehat{\delta}$ and $\widehat{\gamma}$, related to δ and γ, respectively, that are expressible in the assertion language. ∎

6.3.2 Compassion

Unlike Rule 4, which is complete for adding J-compatible constraints, Rule 5 is not complete by itself for adding C-compatible constraints. To prove that the methodology is nonetheless complete for adding progress constraints, we proceed in two steps. First, we introduce a more complex rule that is complete by itself for adding C-compatible constraints. Then we show that each application of this rule can be mimicked by the application of Rule 2 to split some vertices, by the application of Rule 5 to add the compassion constraint, and by the application of Rule 1 to merge back the vertices.

The advanced rule requires the use of a family of auxiliary assertions $\{\varphi(v)\}_{v\in V}$, used to represent a set of locations $\{(v,s) \in loc(A) \mid s \models \varphi(v)\}$ that plays the same role as R in leading to the goal.

Rule 7 (Compassion, with Auxiliary Assertions) Given a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ and a constraint (R, G) such that $R \subseteq V$ and $G \subseteq V \times V$. Assume that there are

1. a family of assertions $\{\varphi(v)\}_{v\in V}$ over $\mathcal{V}$, such that $\varphi(v) = \mathit{true}$ for all $v \in R$ and
2. a ranking function δ and a delay function γ

such that for every $u, v \in V$ with $(u,v) \notin G$, the assertions

$$\widehat{\tau}(u,v) \quad \rightarrow \quad \delta(u) \geq \delta'(v) \tag{6.6}$$

$$\varphi(u) \wedge \widehat{\tau}(u,v) \quad \rightarrow \quad \delta(u) > \delta'(v) \;\vee\; \neg\varphi'(v) \;\vee\; \gamma(u) \geq \gamma'(v) + \Delta \tag{6.7}$$

$$\neg\varphi(u) \wedge \widehat{\tau}(u,v) \quad \rightarrow \quad \delta(u) > \delta'(v) \;\vee\; \neg\varphi'(v) \tag{6.8}$$

hold. Then $A \Rightarrow A'$, where $A' = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C} \cup \{(R,G)\})$. ∎

The soundness and completeness of this advanced rule are expressed by the following theorems.

Theorem 22 (Soundness of Rule 7) *If a constraint (R, G) is added by Rule 7 to the compassion set of a diagram A, producing diagram A', then $Runs(A) = Runs(A')$, and therefore $\mathcal{L}(A) = \mathcal{L}(A')$.*

Proof of 22:

Again, it is immediate that $Runs(A') \subseteq Runs(A)$. To show the reverse containment, assume towards the contradiction that the conditions of Rule 7 are satisfied and that there is a run $\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots \in Runs(A)$ that visits infinitely often R, taking only finitely many edges in G. Thus, there is $k \in \mathbb{N}$ such that $(v_i, v_{i+1}) \notin G$ for all $i \geq k$. By (6.6), the value of δ does not increase along σ after k.

Let $B_1 = \{(v, s) \in loc(A) \mid s \models \varphi(v)\}$, $B_0 = loc(A) - B_1$, and note that no vertices of R are part of locations of B_0. If σ visits B_0 infinitely often, it must infinitely often return to B_1 to visit R, and by (6.8) the value of δ beyond k decreases infinitely often, violating the hypothesis that the domain of δ is well-founded. Thus, there is $m \geq k$ such that σ stays in B_1 forever beyond position m. Once σ is confined to B_1, the proof follows that of Theorem 17, because of the similarity between the combination of assertions (6.6) and (6.7) and assertion (6.1). ∎

Theorem 23 (Completeness, Compassion) *If a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{C}, \mathcal{J})$ is globally reachable and (R, G) is C-compatible with $A' = (\mathcal{V}, V, \rho, \theta, \tau, \emptyset, \emptyset)$, then (R, G) can be added to the compassion set of A with one application of Rule 7.*

Proof of 23:

The proof of the theorem is related to the proof of Theorem 21, from which we borrow some notation. For $n \geq 0$, we call a finite sequence of locations $(v_0, s_0), (v_1, s_1), \ldots, (v_n, s_n)$ a *g-free-path* if $\forall i \in [1..n-1]$, no pair (v_i, v_{i+1}) of consecutive vertices is in G and $(s_i, s_{i+1}) \models \exists \Delta \,.\, \tau(v_i, v_{i+1})$. A g-free-path is a *c-path* if it contains at least one vertex in R. Define the relation $\sqsubset \subseteq loc(A) \times loc(A)$ by

$$(v, s) \sqsubset (v', s') \quad \textit{iff} \quad s'[\![T]\!] - s[\![T]\!] \geq 1 \textit{ and there is a c-path from } (v, s) \textit{ to } (v', s').$$

Since (R, G) is C-compatible with A, relation $\sqsubset$ is well-founded on $loc(A)$. In fact, reasoning as in the previous proof, it can be shown that any infinite descending chain provides a counterexample to the C-compatibility of (R, G). Again, on the basis of $\sqsubset$ we can define a ranking function $\delta \colon loc(A) \mapsto Ord$. For all locations $\ell \in loc(A)$ let

$$A(\ell) = \{\eta \mid \eta \textit{ is a level c-path from } \ell\} \ ,$$

and define

$$B_1 = \{\ell \in loc(A) \mid A(\ell) \neq \emptyset\} \qquad B_0 = loc(A) - B_1 \ .$$

For a location $(v, s) \in loc(A)$, if $v \in R$ then $(v, s) \in B_1$, since $A(v, s)$ contains at least the single-location c-path (v, s). Thus, $(v, s) \in B_1$ if and only if either $v \in R$ or there is a level c-path from (v, s).

Consider a family of assertions $\{\varphi(v)\}_{v \in V}$ such that

$$v \in R \;\rightarrow\; \varphi(v) = \mathit{true} \qquad\qquad (v, s) \in B_1 \;\;\leftrightarrow\;\; s \models \varphi(v) \wedge \rho(v)$$

for all $v \in V$. These assertions characterize B_1 and, by complementation, also B_0. Define the function γ: $loc(A) \mapsto \mathcal{R}^+$ by

$$\begin{array}{ll} \gamma(\ell) = \sup_{\eta \in A(\ell)} \operatorname{len}(\eta) & \text{for } \ell \in B_1 \\ \gamma(\ell) = 0 & \text{for } \ell \in B_0 \,. \end{array}$$

We claim that the family of assertions $\{\varphi(v)\}_{v \in V}$, together with the functions δ and γ, satisfy the conditions of Rule 7. In fact, let (u, s) and (v, t) be two locations such that $(u, v) \notin G$ and $(s, t) \models \exists \Delta \,.\, \tau(u, v)$, and consider assertions (6.6), (6.7), and (6.8).

1. Assertion (6.6) is proved by showing that for any $\ell \in loc(A)$, $(v, t) \sqsubset \ell$ implies $(u, s) \sqsubset \ell$. The conclusion follows from the properties of δ.
2. Consider assertion (6.7), and assume $(u, s) \in B_1$. If $\delta(u, s) > \delta(v, t)$, the conclusion follows. Otherwise, $(u, s), (v, t)$ is a level g-free-path. If $A(v, t) = \emptyset$, then $(v, t) \in B_0$, so $t \models \neg\varphi(v)$ and the conclusion follows again. Else, let $d = t[\![T]\!] - s[\![T]\!]$ be the time elapsed from s to t, and consider any $\eta \in A(v, t)$. Path $(u, s), \eta$ is a level c-path, so $(u, s), \eta \in A(u, s)$; moreover $\operatorname{len}((u, s), \eta) = \operatorname{len}(\eta) + d$. By definition of γ, $\gamma(u, s) \geq \gamma(v, t) + d$, and the conclusion follows once more.
3. Consider assertion (6.8), and assume $(u, s) \in B_0$. If $\delta(u, s) > \delta(v, t)$, the conclusion follows. Otherwise, $(u, s), (v, t)$ is a level g-free-path. Since $A(u, s) = \emptyset$, we have $A(v, t) = \emptyset$. In fact, the existence of $\eta \in A(v, t)$ would imply that $(u, s)\eta \in A(u, s)$, contradicting $(u, s) \in B_0$. Since $A(v, t) = \emptyset$, we have $(v, t) \in B_0$, and thus $t \models \neg\varphi(v)$, from which the conclusion follows.

As in the proof of Theorem 21, even though the functions δ and γ may not be representable directly in the assertion language, it is possible to construct alternative functions $\widehat{\delta}$, $\widehat{\gamma}$ that are expressible, using the methods of [115]. Similarly, it can be shown that the family of assertions $\{\varphi(v)\}_{v \in V}$ can be constructed on the basis of $\widehat{\delta}$. ∎

The following theorem states that we can mimic an application of Rule 7 with one application of Rule 5 and two applications of Rule 1.

Theorem 24 (Simulating Rule 7) *Every application of Rule 7 can be mimicked by one application of Rule 5 and two applications of Rule 1.*

Proof of 24:

Let $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ be the original diagram, where $V = \{v_1, \ldots, v_n\}$. Let (R, G) be the constraint added by Rule 7, and let δ, $\varphi_1, \ldots, \varphi_n$ be the ranking function and the auxiliary assertions used by the rule. First, we use Rule 2 to split each node $v_i \in V$ into two new nodes u_i, w_i, where

$$\rho(u_i) = \rho(v_i) \wedge \varphi_i \qquad \rho(w_i) = \rho(v_i) \wedge \neg\varphi_i$$

for $1 \leq i \leq n$, producing diagram A_1.

Second, we apply Rule 5 to A_1 to add the constraint (R', G') defined by

$$R' = \{u_i \mid 1 \leq i \leq n\}$$
$$G' = \{x_i, y_j \mid x, y \in \{u, w\} \wedge (v_i, v_j) \in G \wedge 1 \leq i, j \leq n\} ,$$

producing diagram A_2. The application of the rule relies on the ranking function defined by $\delta(u_i) = \delta(w_i) = \delta(v_i)$ for $1 \leq i \leq n$. It can be seen that Conditions (6.6)–(6.8) of Rule 7 imply that the corresponding Conditions (6.3)–(6.5) of Rule 5 hold.

Third, we apply Rule 1 to merge nodes u_i and w_i back into a single node z_i, labeled by

$$\rho(z_i) = \rho(u_i) \vee \rho(w_i) = \rho(v_i) ,$$

for $1 \leq i \leq n$. The function μ corresponding to this merge is specified by $\mu(u_i) = \mu(w_i) = \mu(z_i)$, for $1 \leq i \leq n$. The structure of the resulting diagram A_3 will be identical to the structure of A, aside for the names of the vertices, which are different to avoid confusion, and for the presence of an additional constraint (R'', G''), defined by

$$R'' = \{z_i \mid v_i \in R\} \qquad G'' = \{(z_i, z_j) \mid (v_i, v_j) \in G\} .$$

This constraint is the same, except for the vertex names, as the constraint (R, G) added by Rule 7. When applying Rule 1 to merge the vertices, Condition 3 requires that the presence of constraint (R'', G'') in A_3 be justified in terms of a constraint of A_2. By inspection, we see that the constraint (R', G') indeed justifies (R'', G''), which concludes the proof. ∎

6.3.3 Eliminating Livelocking Locations

The following lemma will be used to transform the diagram obtained from a PTS into a non-livelocking, globally reachable diagram.

Lemma 25 *Given a diagram A with empty justice and compassion sets, it is possible to transform A into a globally reachable, non-livelocking diagram B having the same set of computations, the same number of vertices, and empty justice and compassion sets.*

Proof of 25:

Let $C_0 = A = (\mathcal{V}, V, \rho, \theta, \tau, \emptyset, \emptyset)$ be the original diagram. Let $F \subseteq loc(C_0)$ be the set of reachable locations of C_0. By the methods of [115], it is possible to construct a family of assertions $\{\psi(v)\}_{v \in V}$ such that $s \models \psi(v)$ iff $(v, s) \in F$. Using the simulation rule, it is possible to transform C_0 into $C_1 = (\mathcal{V}, V, \bar{\rho}, \theta, \tau, \emptyset, \emptyset)$, where $\bar{\rho}(u) = \rho(u) \wedge \psi(u)$ for all $u, v \in V$.

To obtain a non-livelocking diagram, define the set $E \subseteq loc(C_1)$ as the set of locations of A_1 that appear in some run of A_1. Note that if $\ell \in loc(C_1) - E$ and $\ell' \in E$, there can be no transition from ℓ to ℓ'. Consider a family of assertions $\{\varphi(v)\}_{v \in V}$ such that $s \models \varphi(v)$ iff $(v, s) \in E$, for all $v \in V$. To see that these families of assertions exist, we define the relation $\sqsubset \subseteq loc(A) \times loc(A)$ by

$$(v, s) \sqsubset (v', s') \quad \textit{iff} \quad s'[\![T]\!] - s[\![T]\!] \geq 1 \textit{ and there is a path from } (v, s) \textit{ to } (v', s').$$

For a location $\ell \in loc(C_1)$, we define the predicates $T(\ell)$ and $H(\ell)$ by the formulas

$$T(\ell) \equiv \forall \ell' \, . \, \neg(\ell \sqsubset \ell') \tag{6.9}$$

$$H(\ell) \equiv \mu Q(\ell) \, . \, \Big[T(\ell) \vee \forall \ell' \, . \, (\ell \sqsubset \ell' \to Q(\ell')) \Big] \, , \tag{6.10}$$

where μ denotes the least fixpoint operator. It is immediate that $H(\ell)$ implies $\ell \notin E$, since all locations of C_1 are reachable. Conversely, assume that $\neg H(\ell)$. From (6.10), we know that there is at least one ℓ_1 such that $\neg H(\ell_1)$ and $\ell \equiv \ell_0 \sqsubset \ell_1$. Continuing in this way, we can construct an infinite run $\ell \equiv \ell_0, \ell_1, \ell_2, \ldots$, implying $\ell \in E$. The existence of the family of assertions $\{\varphi(v)\}_{v \in V}$ is then a consequence of the fact that the relation $\sqsubset$ and the other constructs of (6.9) and (6.10) can be expressed in our logic [115].

By a second application of the simulation rule, it is possible to transform C_1 into $C_2 = (\mathcal{V}, V^{(1)} \cup V^{(2)}, \widetilde{\rho}, \theta, \widetilde{\tau}, \emptyset, \emptyset)$, where

1. $V^{(1)} = \{v^{(1)} \mid v \in V\}$ and $V^{(2)} = \{v^{(2)} \mid v \in V\}$;
2. for $v^{(1)} \in V^{(1)}$ and $v^{(2)} \in V^{(2)}$,

$$\widetilde{\rho}(v^{(1)}) = \bar{\rho}(v) \wedge \varphi(v) \qquad \widetilde{\rho}(v^{(2)}) = \bar{\rho}(v) \wedge \neg\varphi(v)$$

3. for $u^{(1)}, v^{(1)} \in V^{(1)}$ and $u^{(2)}, v^{(2)} \in V^{(2)}$,

$$\widetilde{\tau}(u^{(i)}, v^{(j)}) = \begin{cases} \textit{false} & \text{if } i = 2, \, j = 1; \\ \tau(u, v) & \text{otherwise.} \end{cases}$$

The function $\mu\colon V \mapsto V^{(1)} \cup V^{(2)}$ is defined by $\mu(u) = \{u^{(1)}, u^{(2)}\}$ for all $u \in V$.

Note that the constraint $(V^{(2)}, \emptyset)$ is J-compatible with C_2, since no run of A_2 contains vertices in $V^{(2)}$. By Theorem 21, using Rule 4 we can transform C_2 into $C_3 = (\mathcal{V}, V^{(1)} \cup V^{(2)}, \widetilde{\rho}, \theta, \widetilde{\tau}, \{(V^{(2)}, \emptyset)\})$. Finally, using Rule 6 we can prune from C_3 all the vertices in $V^{(2)}$, producing B. By construction, B is globally reachable, non-livelocking, and has empty justice and compassion sets; moreover, it has the same number of vertices as A. ∎

6.3.4 General Completeness

Combining the results of the previous completeness theorems and the lemma, we obtain the following result that will be used to show the completeness of the verification methodology based on hybrid diagrams with respect to linear temporal logic specifications.

Theorem 26 (Completeness for PTS) *If $\mathcal{S}$ is a PTS and A is a deterministic diagram over $\mathcal{V}$ such that $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(A)$, Rules 1, 4, 5, and 6 enable the construction of a chain of transformations $hd(\mathcal{S}) \overset{*}{\Rightarrow} A$.*

Proof of 26:

Let $A_0 = hd(\mathcal{S}) = (\mathcal{V}, \{u_0\}, \rho(u_0) = true, \theta_0, \tau_0, \emptyset, \emptyset)$ and $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$, where $\mathcal{J} = \{(R_1, G_1), \ldots, (R_m, G_m)\}$ and $\mathcal{C} = \{(R_{m+1}, G_{m+1}), \ldots, (R_n, G_n)\}$.

By Lemma 25, A_0 can be transformed into a single-vertex, globally reachable, non-livelocking diagram $A_1 = (\mathcal{V}, \{u_1\}, \rho_1, \theta_1, \tau_1, \emptyset, \emptyset)$ such that $\mathcal{L}(A_0) = \mathcal{L}(A_1)$.

Since diagram A is deterministic, for every sequence of states $\xi : s_0, s_1, s_2, \ldots$, there corresponds at most one sequence of locations $L(\xi) = (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$ that satisfies the Initiality requirement of diagram computations and which also satisfies the implicit consecution requirement. Define $Runs_{\mathcal{S}}(A) = \{L(\xi) \mid \xi \in \mathcal{L}(\mathcal{S})\}$, and let $E \subseteq loc(A)$ be the set of locations that appear in some run in $Runs_{\mathcal{S}}(A)$. Again, it is possible to define a family of assertions $\{\varphi(v)\}_{v \in V}$ such that $s \models \varphi(v)$ iff $(v, s) \in E$, for all $(v, s) \in loc(A)$. Consider the diagram $A_2 = (\mathcal{V}, V, \rho_2, \theta_2, \tau_2, \emptyset, \emptyset)$, where for all $v, v' \in V$:

1. $\rho_2(v) = \rho(v) \wedge \varphi(v)$;
2. $\theta_2(v) = \theta(v) \wedge \theta_1(v)$;
3. $\tau_2(v, v') = \tau(v, v') \wedge \tau_1(v, v)$.

To show that $\mathcal{L}(A_1) = \mathcal{L}(A_2)$, we show both $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ and $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$.

1. $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$. Consider a computation $\sigma \in \mathcal{L}(A_1)$. By hypothesis, $\sigma \in \mathcal{L}(A)$, and there is a run $\widehat{\sigma} \in Runs_{\mathcal{S}}(A)$ that corresponds to σ. By construction, all locations of $\widehat{\sigma}$ are present in A_2, and all transitions of $\widehat{\sigma}$ are possible in A_2, by definition of τ_2. Thus, $\widehat{\sigma} \in Runs(A_2)$, and $\sigma \in \mathcal{L}(A_2)$, as was to be shown.

2. $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$. Consider a computation $\widehat{\sigma} \in \mathcal{L}(A_2)$ arising from a run $\sigma \in \mathcal{L}(A_2)$. By construction of θ_2 and τ_2, we see that every state-transition possible along $\widehat{\sigma}$ is also possible in A_1. Since the progress set of both diagrams is empty, we conclude that $\widehat{\sigma} \in \mathcal{L}(A_1)$, as was to be shown.

There is a simulation transformation that transforms A_1 into A_2, based on the mapping $\mu : \{u_1\} \mapsto V$. To see this, consider the conditions of Rule 1.

1. Since $\mathcal{L}(A_1) \subseteq \mathcal{L}(A)$ and A_1 is non-livelocking, we must have $\theta_1(u_1) \to \bigvee_{u \in V} \theta(u)$. The result then follows from the definition of θ_2.
2. Let (u_1, s) be a location of A_1 from which there is a transition to (u_1, s'), and let (v, s) be a location of A_2 related to (u_1, s). By construction of A_2, there is a run $\sigma' \in \mathcal{L}(A_1)$ containing (u_1, s) that induces a run $\sigma \in \mathcal{L}(A_2)$ containing (v, s). Let $\eta, (v, s)$ be the run prefix of A_2 leading to (v, s), and let $\eta', (u_1, s)$ be the corresponding prefix of σ' leading to (u_1, s). Since there is a transition from (u_1, s) to (u_1, s') and since A_1 is non-livelocking, the run prefix η' can be extended to a run $\eta', (u_1, s), (u_1, s'), \xi' \in \mathit{Runs}(A_1)$. Since A_2 is deterministic, run $\eta, (v, s)$ is the only run prefix of A_2 that corresponds to $\eta', (u_1, s)$ and since $\mathcal{L}(A_1) = \mathcal{L}(A_2)$, run $\eta, (v, s)$ can also be extended to a run $\eta, (v, s), (v', s'), \xi \in \mathit{Runs}(A_2)$, for some $(v', s') \in \mathit{loc}(A_2)$ and ξ. This fact shows that indeed A_2 can take a transition from (v, s) to a location (v', s') related to (u_1, s'), as Condition 2 of Rule 1 requires.
3. Immediate, since A_2 has empty progress set.

For $1 \le i \le m$, constraint (R_i, G_i) is J-compatible with A_2, and for $m + 1 \le i \le n$, constraint (R_i, G_i) is C-compatible with A_2. To see this fact, assume towards the contradiction that there is a run $\sigma \in \mathit{Runs}(A_2)$ that does not satisfy (R_i, G_i) for $1 \le i \le n$, and let $\widehat{\sigma}$ be the computation of A arising from σ. Since $\mathcal{L}(A_2) = \mathcal{L}(A_1) = \mathcal{L}(\mathcal{S})$, $\widehat{\sigma} \in \mathcal{L}(\mathcal{S})$, and since A_2 and A are deterministic, σ is the only run of A_2 that corresponds to $\widehat{\sigma} \in \mathcal{L}(\mathcal{S})$. Thus, if σ does not satisfy (R_i, G_i), $\sigma \notin \mathit{Runs}(A)$, and $\widehat{\sigma} \notin \mathcal{L}(A)$, contradicting $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(A)$. $\Rightarrow\Leftarrow$
Thus, by Theorems 21 and 23 we can construct a series of transformations $A_2 \Rightarrow A_3 \Rightarrow \cdots \Rightarrow A_{n+2}$, where:

1. for $1 \le i \le m$, A_{i+2} is obtained by adding constraint (R_i, G_i) to the justice set of A_{i+1} using Rule 4;
2. for $m + 1 \le i \le n$, A_{i+2} is obtained by adding constraint (R_i, G_i) to the compassion set of A_{i+1} using Rules 5 and 1, as described by Theorem 24.

The final step consists in proving that there is a simulation transformation based on the identity mapping between A_{m+2} and A. This step is a simple consequence of the fact that A_{m+2} and A share the same progress set and the labelings of A_{m+2} are stronger (i.e. they imply) the corresponding labelings of A. ∎

6.4 Summary

In this chapter, we have presented a diagram-based methodology for the verification of hybrid system properties expressed in linear-time temporal logic. The proof of a system property consists of a chain of stepwise diagram transformations. The visual representation of the system behavior, coupled with the guidance provided by the algorithm of the previous section, directs the gradual construction of the proof. Unlike previous approaches, the methodology followed in this chapter is complete (relative to first-order reasoning) with respect to both safety and progress properties, and its application is not restricted to non-Zeno systems [3].

While we have chosen phase transition systems as our basic system model, the methodology can be adapted to other models as well, including hybrid automata. In particular, we remark that the definition of a hybrid diagram does not require that the hybrid activities are *deterministic*, as is the case for the definition of PTS. Thus, hybrid diagrams can be used to study systems in which the dynamic evolution of some hybrid variables is specified only by bounds on their derivatives, rather than by exact differential equations.

We conclude by observing that it is possible to formulate *verification rules*, in the style of [117, 93], that correspond to Rules 4, 5, and 6. These rules would also lead to a verification methodology for hybrid systems complete for TL_s. We chose to present the rules in the context of diagram transformations, rather than premise-conclusion reasoning, because of the perceived advantages of the diagram-based approach.

Chapter 7

Diagrammatic Verification: Continuous Semantics

In this chapter, we extend the diagrammatic approach introduced in Chapter 6 to the super-dense semantics introduced in Chapter 3. Our main motivation for using the super-dense semantics is to overcome the following problem of the sampling semantics. Suppose we wish to prove the formula $\Diamond(T = 10)$ for system RH. Intuitively, this property should be true since the master clock proceeds at a rate of 1, starts at 0, and never gets reset. However, under the sampling semantics, there are sampling runs which never sample the system at $T = 10$. For these runs, the formula $\Diamond(T = 10)$ is false. Thus, the formula $\Diamond(T = 10)$ is not a property of the system under the sampling semantics. As discussed in Chapter 3, our solution is to move to a super-dense semantics for expressing system behavior and to interpret LTL under this super-dense semantics.

7.1 Contributions

The contribution of this chapter is the extension of the diagrammatic approach introduced in Chapter 6 to prove properties of linear-time temporal logic under our super-dense semantics.

7.2 Preliminaries

We start by noting that the syntax of hybrid diagrams, TL_s, and Streett automata does not change. However, instead of interpreting their behavior under the sampling semantics, we interpret their behavior under the continuous semantics. For TL_s, this semantics is similar to the semantics for LTL that we presented in Chapter 3. For hybrid diagrams and Streett automata, we present the semantics below.

A *super-dense run* of a hybrid diagram is a pair $(\bar{\ell}, f)$, where $\bar{\ell}$ is an infinite sequence of locations (v_0, s_0), (v_1, s_1), (v_2, s_2), $\ldots$ and f is a tuple of functions, one for each variable $x \in \mathcal{V}$ that satisfies the following conditions:

Initiality: $s_0 \models \theta(v_0)$.

Vertex Labels: for all $i \geq 0$, $s_i \models \rho(v_i)$ (this condition is implied by the fact that (s_i, v_i) is a location).

Edge Labels: for all $i \geq 0$, $(s_i, s_{i+1}) \models \exists\Delta\,.\,\tau(v_i, v_{i+1})$. Fix this Δ and call it Δ_i. We require $\forall t \in [0, \Delta_i)$, either $(s_i, f(t)) \models (\tau(v_i, v_{i+1}) \wedge \Delta = t)$ or $(s_i, f(t)) \models (\tau(v_i, v_i) \wedge \Delta = t)$.

Time Progress: for each $t \in \mathcal{R}$ there is $i \in \mathbb{N}$ such that $s_i(T) \geq t$.

Justice: for each constraint $(R, G) \in \mathcal{J}$, if there is $k \in \mathbb{N}$ such that for all $i \geq k$, $v_i \in R$, then there is $j \geq k$ such that $(v_j, v_{j+1}) \in G$.

Compassion: for each constraint $(R, G) \in \mathcal{C}$, if $v_i \in R$ for infinitely many $i \in \mathbb{N}$, then there are infinitely many $j \in \mathbb{N}$ such that $(v_j, v_{j+1}) \in G$.

If $(\bar{\ell}, f)$ is a super-dense run of A, where $\bar{\ell} = (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$, then $\sigma = \langle \bar{s}, f \rangle$ is a *super-dense computation* of A where $\bar{s} = s_0, s_1, s_2, \ldots$. We denote by $\mathit{Runs}_c(A)$ and $\mathcal{L}_c(A)$ the sets of super-dense runs and super-dense computations of A, respectively.

A *super-dense run* σ of a Streett automaton $N = (\mathcal{V}, (V, E), \rho, Q, \mathcal{B})$, is a pair $(\bar{\ell}, f)$, where $\bar{\ell}$ is an infinite sequence of locations (v_0, s_0), (v_1, s_1), (v_2, s_2), $\ldots$ such that $v_0 \in Q$ and f is a tuple of functions, one for each variable $x \in \mathcal{V}$ that satisfies the following conditions:

1. for all $i \geq 0$, $s_i \models \rho(v_i)$ and $(v_i, v_{i+1}) \in E$;
2. there exists a $t_i \in [\mathit{time}(s_i), \mathit{time}(s_{i+1}))$ such that for all $t \in [\mathit{time}(s_i), t_i)$, $f(t) \models \rho(v_i)$ and for all $t \in [t_i, \mathit{time}(s_{i+1}))$, $f(t) \models \rho(v_{i+1})$.
3. for each pair $(P, R) \in \mathcal{B}$, either $v_i \in R$ for infinitely many $i \in \mathbb{N}$, or there is $k \in \mathbb{N}$ such that $v_i \in P$ for all $i \geq k$.

If $(\bar{\ell}, f)$ is a super-dense run of N, where $\bar{\ell} = (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$, then $\sigma = \langle \bar{s}, f \rangle$ is a *super-dense computation* of N where $\bar{s} = s_0, s_1, s_2, \ldots$. The set of runs (resp. computations) of a Streett automaton N is denoted by $\mathit{Runs}_c(N)$ (resp. $\mathcal{L}_c(N)$).

Theorem 27 *For a* PTS $\mathcal{S}$, $\mathcal{L}_c(\mathcal{S}) = \mathcal{L}_c(hd(\mathcal{S}))$, *where* $hd(\mathcal{S})$ *is the hybrid diagram of Contstruction 1.*

The proof of Theorem 27 is trivial.

The simulation rule (Rule 1), the two progress rules (Rule 4 and Rule 5), and the pruning rule (Rule 6) are all transformation rules for hybrid diagrams under the continuous semantics.

Theorem 28 (Soundness of Chapter 6 rules)

1. *If $A_1 \Rightarrow A_2$ by Rule 1, then $\mathcal{L}_c(A_1) \subseteq \mathcal{L}_c(A_2)$.*

2. *If a constraint (R, G) is added by Rule 4 to the justice set of a diagram A, producing diagram A', then $Runs_c(A) = Runs_c(A')$ and $\mathcal{L}_c(A) = \mathcal{L}_c(A')$.*

3. *If a constraint (R, G) is added by Rule 5 to the compassion set of a diagram A, producing diagram A', then $Runs_c(A) = Runs_c(A')$ and $\mathcal{L}_c(A) = \mathcal{L}_c(A')$.*

The proofs of soundness for the rules are similar to the proofs presented in Chapter 6.

7.3 New Diagram Transformation Rules

In this section, we present two new diagram transformation rules that apply only to the continuous semantics. These rules will help prove propertis that are valid under the continuous semantics but not valid under the sampling semantics. They are primarily used to prove progress (e.g., $\Diamond \varphi$) and until (e.g., $\psi \mathcal{U} \varphi$) properties (capturing the precise instant of time when φ is true).

Motivating Example

Consider the hybrid diagram C_1 presented in Figure 7.1. We would like to prove the LTL formula $\Diamond(T = 10)$. Under the sampling semantics, this formula is not valid for system C_1. For example, the following is a sampling run fragment of C_1 that does not satisfy the formula.

$$\sigma\colon\ (v_0^1, T = 0),\ (v_1^1, T = 3),\ (v_1^1, T = 9),\ (v_1^1, T = 11),\ \ldots$$

Note that all extensions of this run fragment will not satisfy the formula as our last snapshot in the fragment has $T > 10$ and time can not decrease. This run fragment exhibits two problems:

1. the run σ does not sample the system at $T = 10$, and

2. the system jumps from vertex v_1^1 and $T = 9$ to vertex v_1^1 and $T = 11$, bypassing any transition to v_0^1 and $T \in [10, 11)$.

Our new rules overcome these problems by allowing us to put upper bounds on Δ, insuring that the τ transitions do not miss desired locations.

We first introduce some notation. Given a hybrid diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ and a real-valued expression f over $\mathcal{V}$, $\tau[f](u, v)$ is the transition assertion over $\mathcal{V} \cup \mathcal{V}'$ where each occurrence of Δ is replaced by the real-valued expression f.

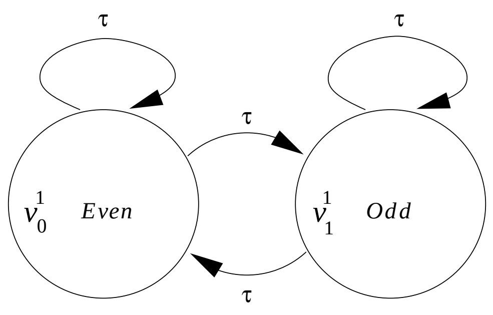

Figure 7.1: Hybrid Diagram C_1, where τ is the transition $T = T + \Delta$, *Even* is the predicate $\exists n \in \mathbb{N} \,.\, 2n \leq T < 2n+1$, and *Odd* is the predicate $\exists n \in \mathbb{N} \,.\, 2n+1 \leq T < 2n+2$.

Rule 8 (Point-Jump) Let $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ be a hybrid diagram. Let u and v be two vertices whose corresponding edge we wish to strengthen, and let g be a term on $\mathcal{V}$ such that $\rho(u) \rightarrow g \geq 0$. If

$$\rho(u) \ \wedge \ \tau[g](u,v) \ \wedge \ (\rho'(u) \ \vee \ \rho'(v)) \ \leftrightarrow \ \mathit{false} \ ,$$

then let A' be a hybrid diagram just like A except with the edge from u to v labeled with $\tau(u,v) \wedge \Delta < g$.

Theorem 29 (Soundness of Rule 8) *If $A \Rightarrow A'$ by Rule 8, then $\mathcal{L}_c(A) \subseteq \mathcal{L}_c(A')$.*

The proof of Theorem 29 is straightforward. The intuition of the rule is as follows. Suppose we are at a location (u, s_u) and from vertex u there is an edge going to v. If there is a $t_0 \in \mathcal{R}^+$ such that going from u and *any* allowable state $s_i \models \rho(u)$, we do not reach a state satisfying $\rho(v)$ or $\rho(u)$, then the transition $\tau(u,v)$ should be restricted to only allow transitions where $\Delta < t_0$. Any transition taking $\Delta > t_0$ either results in a state that can not be reached in a continuous manner or results in a state that can be reached in a continuous manner but along a different path (i.e., along a different sequence of transitions). Our point-jump rule insures that our edge does not jump over any time-points that do not correspond to legal locations (i.e., locations that can actually appear in computations).

Rule 9 (Interval-Jump) Let $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ be a hybrid diagram. Let u and v be two vertices whose corresponding edge we wish to strengthen, and let g be a term on $\mathcal{V}$ such that $\rho(u) \rightarrow g \geq c > 0$ for some constant c. If

$$\rho(u) \ \wedge \ \tau[g](u,v) \ \wedge \ \rho'(v) \ \leftrightarrow \ \mathit{false} \ ,$$

then let A' be a hybrid diagram just like A except with the edge from u to v labeled with $\tau(u,v) \wedge \Delta < g$.

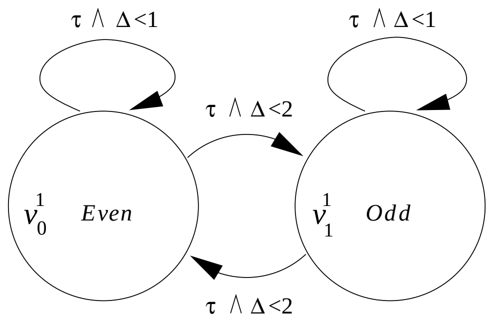

Figure 7.2: Hybrid Diagram C_2, where τ is the transition $T = T + \Delta$, *Even* is the predicate $\exists n \in \mathbb{N} \,.\, 2n \leq T < 2n+1$, and *Odd* is the predicate $\exists n \in \mathbb{N} \,.\, 2n+1 \leq T < 2n+2$.

Theorem 30 (Soundness of Rule 9) *If $A \Rightarrow A'$ by Rule 8, then $\mathcal{L}_c(A) \subseteq \mathcal{L}_c(A')$.*

The proof of Theorem 30 is straightforward. The intuition of the rule is as follows. Suppose we are at a location (u, s_u) and from vertex u there is an edge going to v. If there is an interval $[t_0, t_1) \subset \mathcal{R}^+$ such that going from u and *any* allowable state $s_i \models \rho(u)$, we do not reach a state satisfying $\rho(v)$, then the transition $\tau(u, v)$ should be restricted to only allow transitions where $\Delta < t_0$. Any transition taking $\Delta > t_0$ either results in a state that can not be reached in a continuous manner or results in a state that can be reached in a continuous manner but along a different path (i.e., along a different sequence of transitions). Our interval-jump rule insures that our edge does not jump over any interval of time that does not correspond to legal locations.

With these two rules, we can prove additional properties of hybrid diagrams. The construction of the graph product of a hybrid diagram and a Streett automaton remains the same.

Theorem 31 (Diagram Checking) *Given a diagram A and a specification $\varphi \in TL_s$, let $A \otimes N_{\sim\varphi} = (W, Z, H)$. If all SCSs of (W, H) that are reachable in (W, H) from Z are not admissible, then $A \models_c \varphi$.*

The proof is based on the classical arguments presented by Manna and Pnueli in [115].

Examples

Let us return to the example introduced above. Using Rule 8 on the edge from vertex v_0^1 to itself with $g = 1$ allows us to add the requirement $\Delta < 1$ to the edge from v_0^1 to itself. Using the same g, we can add the same requirement to the edge from v_1^1 to itself. Finally, using $g = 2$, we can add the requirement $\Delta < 2$ to the edge from v_0^1 to v_1^1 and to the edge from v_1^1 to v_0^1. The resulting diagram is shown in Figure 7.2.

Returning to our Room-Heater example, suppose we wish to prove the property $\diamond(x = 65)$. Clearly this formula is valid for system RH under the continuous semantics since the system starts

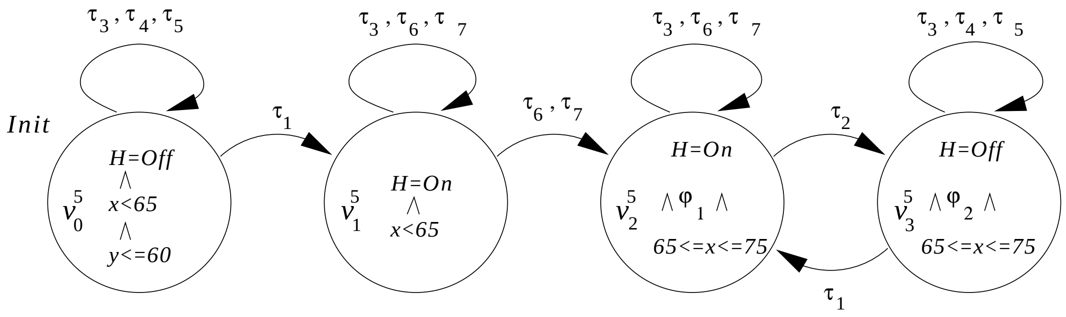

Figure 7.3: Hybrid Diagram A_5, where $\varphi_1 : x \leq 75 - 7 \cdot e^{-y/105}$ and $\varphi_2 : x \geq 60 + 12 \cdot e^{-y/70}$. Edges labeled with *false* are not shown.

at a temperature less than 65 and continuously increases until it reaches a temperature greater than 65. Under the sampling semantics, this formula need not be true since a particular run need not sample the system at precisely the point where $x = 65$. Thus, our only hope of proving this property is by using the point-jump and interval-jump rules.

We first use the simulation rule to simplify the diagram. The rule is based on the function μ defined as follows:

$$\mu(v_0^4) = v_0^5 \quad \mu(v_1^4) = v_1^5 \quad \mu(v_2^4) = v_2^5 \quad \mu(v_3^4) = v_3^5 \quad \mu(v_4^4) = v_2^5 \quad \mu(v_5^4) = v_3^5 \ ,$$

and the resulting diagram is presented in Figure 7.3. We further refine diagram A_5 using the simulation rule and the function μ defined as follows:

$$\mu(v_0^5) = v_0^6 \quad \mu(v_1^5) = v_1^6 \quad \mu(v_2^5) = \{v_2^6, v_4^6, v_5^6\} \quad \mu(v_3^5) = v_3^6$$

to generate diagram A_6 presented in Figure 7.4. One last application of the simulation rule to split vertex v_1^5 gives us the diagram A_7 presented in Figure 7.5. We now use two applications of Rule 8. The first application of the point-jump rule is on the edge from vertex v_2^7 to v_3^7 with $g = -70 \ln(5/(70 - x))$. The result is to add the conjunct $\Delta < -70 \ln(5/(70 - x))$ to the edge. As the new $\widehat{\tau}(v_2^7, v_3^7)$ is *false*, we can eliminate the edge. The second application of the point-jump rule is on the edge from vertex v_6^7 to v_3^7 with $g = -105 \ln(10/(75 - x))$. The result is to add the conjunct $\Delta < -105 \ln(10/(75 - x))$ to the edge. Once again, as the new $\widehat{\tau}(v_6^7, v_3^7)$ is *false*, we can eliminate the edge. The resulting diagram is presented in Figure 7.6. This diagram with its justice requirements inherited from diagram A_4 can be shown to satisfy the property $\diamond(x = 65)$.

7.4 Discussion

The differences between the sampling semantics and the continuous semantics become exposed when trying to prove liveness properties and $\mathcal{U}$ properties. Suppose φ and ψ are state formulas. Under

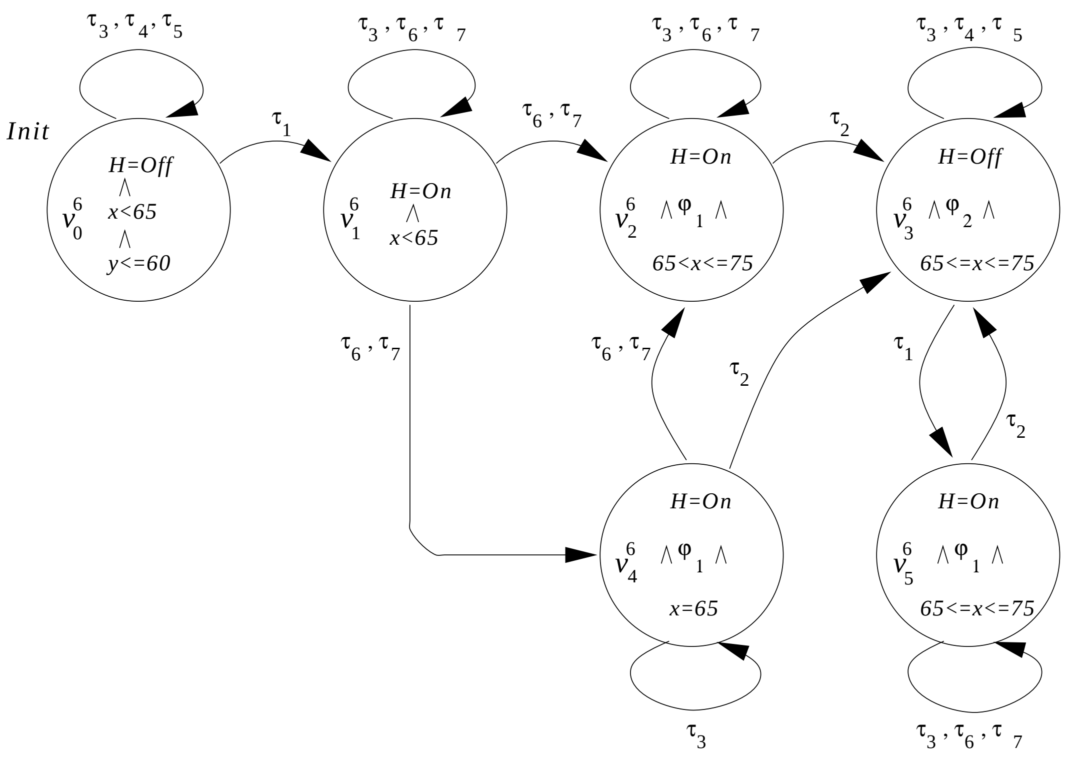

Figure 7.4: Hybrid Diagram A_6, where $\varphi_1 : x \leq 75 - 7 \cdot e^{-y/105}$ and $\varphi_2 : x \geq 60 + 12 \cdot e^{-y/70}$. Edges labeled with *false* are not shown.

the sampling semantics, the formula $\diamondsuit(\varphi)$ is valid only if the run exhibits a snapshot where φ holds. Similarly, under the sampling semantics, the formula $\psi \mathcal{U} \varphi$ is valid only if the run exhibits a snapshot where φ holds. However, for both these formulas, there is no guarantee that a run will sample such a point where φ holds. For this reason, we chose to move to a continuous semantics that records the system's values at every point in time.

Of course, not all $\diamondsuit$ properties require moving to the continuous semantics. For example, the original Room-Heater requirement, $\diamondsuit(65 \leq x \leq 75)$ is valid under the sampling semantics, and we were able to prove it using the rules of Chapter 6. However, for the property $\diamondsuit(x = 65)$ we had to move to the continuous semantics. The reason for the difference is as follows. System RH satisfies the formulas

$$\diamondsuit(65 \leq x \leq 75)$$
$$(65 \leq x \leq 75) \Rightarrow \square(65 \leq x \leq 75)\ ,$$

which are equivalent to

$$\diamondsuit(65 \leq x \leq 75) \tag{7.1}$$

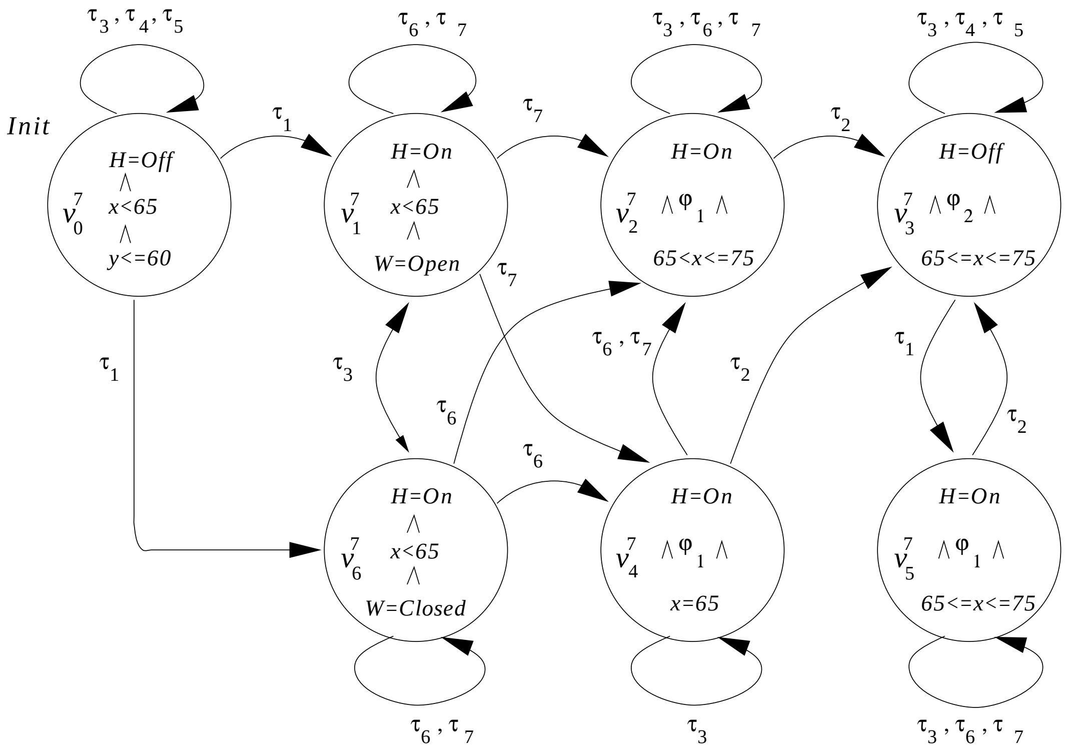

Figure 7.5: Hybrid Diagram A_7, where $\varphi_1 : x \leq 75 - 7 \cdot e^{-y/105}$ and $\varphi_2 : x \geq 60 + 12 \cdot e^{-y/70}$. Edges labeled with *false* are not shown.

$$\Diamond(65 \leq x \leq 75) \rightarrow \Box(65 \leq x \leq 75)\ . \tag{7.2}$$

Under the sampling semantics, even if we fail to sample the first occurrence of $65 \leq x \leq 75$, we know that by 7.2, all subsequent snapshots satisfy $65 \leq x \leq 75$. For the formula, $\Diamond(x = 65)$, we have no such guarantee. Thus, if we fail to sample the first (and possibly only) occurrence of $x = 65$, our run may not satisfy $\Diamond(x = 65)$. Consequently, the formula $\Diamond(x = 65)$ need not be valid for system RH under the sampling semantics.

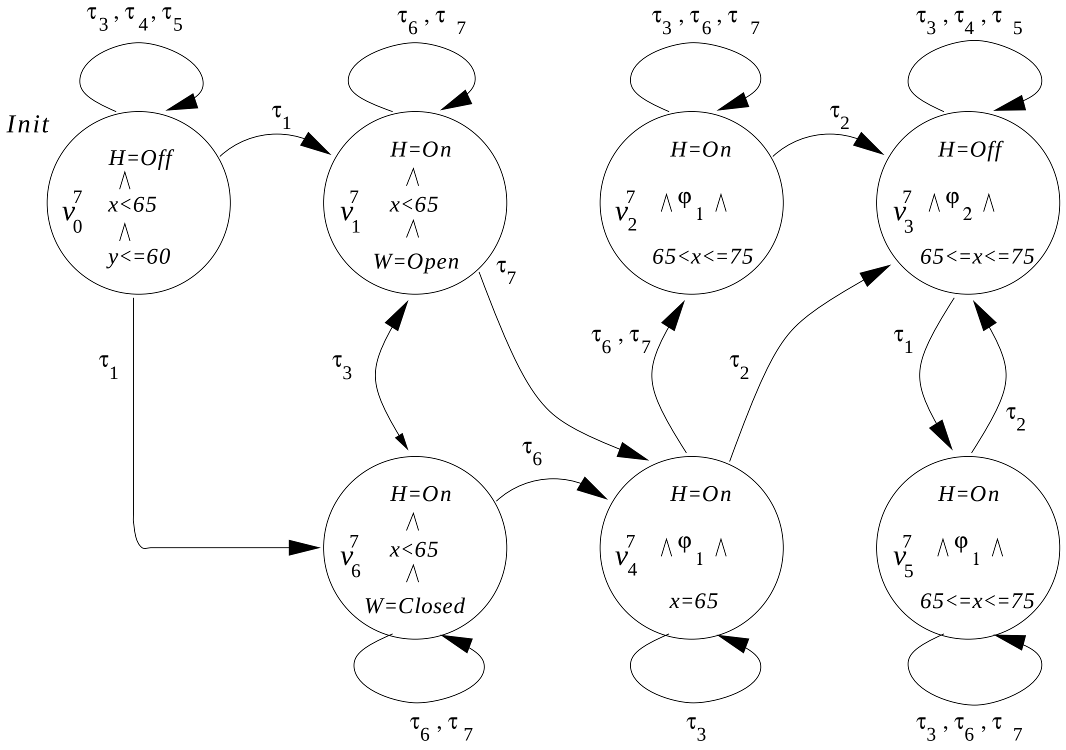

Figure 7.6: Hybrid Diagram A_8, where $\varphi_1 : x \leq 75 - 7 \cdot e^{-y/105}$ and $\varphi_2 : x \geq 60 + 12 \cdot e^{-y/70}$. Edges labeled with *false* are not shown.

Chapter 8

Deductive Verification: Continuous Semantics

In this chapter, we present proof rules for proving properties expressed in hybrid temporal logic (Chapter 3.3.2) over hybrid systems modeled by concrete phase transition systems (Chapter 5.3). We first present the proof rules for point-based properties and then present a proof rule for proving interval-based properties. We illustrate the rules on system GAS introduced in Chapter 2.2.2. We also introduce an induction axiom for proving HTL-validities and system-validities.

8.1 Contributions

The main contribution of this chapter is a verification methodology to prove hybrid temporal logic properties of concrete phase transition systems.

8.2 Proving Point-Based Properties

To prove point-based invariance formulas of the form $\Box\,\psi$ where ψ is a state formula, we use the rule P-INV given in Figure 8.1. We use the notation $\psi(V)$ to emphasize that ψ is a formula over the variables V, and $\psi(V')$ to indicate the result of replacing all variables in $\psi(V)$ by their primed versions. The rule uses two auxiliary assertions φ and χ. Assertion φ is intended to be a stronger version of ψ that is inductive, while assertion χ is a weaker version of φ which holds not only at states within phases but also at the left limits of such states.

Premise PI1 states that φ, where φ is a state formula, is initially true. Premise PI3 states that if χ holds at some state, which could be a left limit of states in the computation, and a discrete transition τ is taken, then φ holds in the new state (since, for transitions, V' represents the values of

$$\boxed{\begin{array}{lll r}
\text{P-INV} & \text{PI1.} & \Theta \rightarrow \varphi(V) & \\
 & \text{PI2.} & \varphi(V) \rightarrow \psi(V) & \\
 & \text{PI3.} & \rho_\tau(V, V') \land \chi(V) \rightarrow \varphi(V') & \forall\tau \in \mathcal{T} \\
 & \text{PI4.} & \rho_\phi(V, \dot{V}) \land \chi(V) \rightarrow \varphi(V) & \forall\phi \in \Phi \\
 & \text{PI5.} & \mathit{continuous} \land \square\, \rho_\phi(V, \dot{V}) \land \varphi(V) \Rightarrow_f \chi(\overleftarrow{V}) & \forall\phi \in \Phi \\
\hline
 & & \square\, \psi(V) &
\end{array}}$$

Figure 8.1: Rule P-INV—Invariance of point-based state formulas

$$\boxed{\begin{array}{lll r}
\text{L-INV} & \text{LI1.} & \Theta \rightarrow \varphi(V) & \\
 & \text{LI2.} & \chi(V) \rightarrow \psi(V) & \\
 & \text{LI3.} & \rho_\tau(V, V') \land \chi(V) \rightarrow \varphi(V') & \forall\tau \in \mathcal{T} \\
 & \text{LI4.} & \rho_\phi(V, \dot{V}) \land \chi(V) \rightarrow \varphi(V) & \forall\phi \in \Phi \\
 & \text{LI5.} & \mathit{continuous} \land \square\, \rho_\phi(V, \dot{V}) \land \varphi(V) \Rightarrow_f \chi(\overleftarrow{V}) & \forall\phi \in \Phi \\
\hline
 & & \square\, \psi(\overleftarrow{V}) &
\end{array}}$$

Figure 8.2: Rule L-INV—Invariance of left-limit state formulas

the variables in the new state). Premise PI4 states that at internal points of a phase, $\chi(V)$ implies $\varphi(V)$.

Premise PI5 is the only temporal premise among the five. It requires that if φ holds at the left end of a ϕ-phase, then χ holds at the state which is the limit from the left of the phase[1].

Premises PI1, PI3, PI4, and PI5 insure that for all time points t, φ holds. By premise PI2, ψ also holds at all time points, which can be written as $\square\, \psi$.

For example, using the above rule we can prove the following point-based invariances for system GAS.

- $at\_\ell_0 \;\rightarrow\; (0 \leq y < 100 \;\land\; 0 \leq T < 10 \;\land\; R = \text{Off} \;\land\; \mathit{switch} = \text{Off})$
- $at\_\ell_1 \;\rightarrow\; (0 \leq y < 100 \;\land\; T = 10 \;\land\; \mathit{switch} = \text{Off})$
- $at\_\ell_2 \;\rightarrow\; (0 \leq y < 10 \;\land\; 0 \leq T < 10 \;\land\; R = \text{On} \;\land\; \mathit{switch} = \text{On})$

We prove the first of these properties in Chapter 8.5; the others are proved in a similar fashion.

A similar rule, L-INV, presented in Figure 8.2, can be used to prove properties of the form $\square\, \psi(\overleftarrow{V})$, where $\psi(\overleftarrow{V})$ is an assertion in $\overleftarrow{V}$.

[1]To prove temporal entailments such as PI5, we use some known facts based on elementary calculus such as $\mathit{continuous} \land \square(\dot{x} = 0) \Rightarrow_f \overleftarrow{x} = \overrightarrow{x}$.

$$
\boxed{
\begin{array}{lllr}
\textsc{i-inv} & \text{II1.} & \varphi(V, V') \;\rightarrow\; \psi(V, V') & \\
 & \text{II2.} & \mathit{continuous} \wedge \Box\, \rho_\phi(V, \dot{V}) \Rightarrow_f \varphi(\overleftarrow{V}, \overrightarrow{V}) & \forall\phi \in \Phi \\
 & \text{II3.} & \varphi(V_1, V_2) \wedge \rho_\tau(V_2, V) \wedge \mathit{continuous} \wedge \Box\, \rho_\phi(V, \dot{V}) \Rightarrow_f \varphi(V_1, \overrightarrow{V}) & \begin{array}{r}\forall\tau \in \mathcal{T} \\ \forall\phi \in \Phi\end{array} \\
\hline
 & & \Box_f \psi(\overleftarrow{V}, \overrightarrow{V}) &
\end{array}
}
$$

Figure 8.3: Rule I-INV—Invariance of interval formulas

8.3 Proving Interval-Based Properties

To prove interval-based invariance formulas of the form $\Box\, \varphi(\overleftarrow{V}, \overrightarrow{V})$ where φ is a formula whose variables appear as left or right limits, we use rule I-INV given in Figure 8.3.

Premise II1 expresses the monotonicity requirements of the rule. The temporal premise II2 states that any ϕ-phase satisfies φ. Premise II3 states that if φ is true over a phase P_1 and we take a discrete transition τ to another phase P_2 on which φ holds, then φ will be true over the phase $P_1 \frown P_2$. Premises II3 and II2 imply that any subphase satisfies φ, and with monotonicity, these conditions guarantee $\Box_f \psi(\overleftarrow{V}, \overrightarrow{V})$.

In addition we may add any previously derived point invariants $p(V)$ to the left of any premise and any previously derived invariants $q(\overrightarrow{V})$ or $r(\overleftarrow{V}, \overrightarrow{V})$ to the left of any temporal premise.

Before presenting example interval invariants, we introduce the following notation. For a variable $x \in V$,

Δx	stands for	$\overrightarrow{x} - \overleftarrow{x}$
$\Delta_1^2 x$	stands for	$x_2 - x_1$
$\Delta_1 x$	stands for	$\overrightarrow{x} - x_1$

For example, using the above rule we can prove the following interval-based invariances for system GAS.

- $\overrightarrow{at}\_\ell_{0,1} \Rightarrow_f \big((\Delta x \le \overrightarrow{y} \wedge \Delta L = 0) \;\vee\; (\Delta x > \overrightarrow{y} \wedge \Delta L < \Delta x - \overrightarrow{y})\big)$
- $\Box_f \big[(\psi_1 \;\vee \psi_2 \;\vee \psi_3) \;\wedge\; \psi_4\big]$ where
  ψ_1: $\Delta L \le \Delta T$
  ψ_2: $\Delta L \le \overrightarrow{T} \wedge \Delta x \le \overrightarrow{T} + 100$
  ψ_3: $\Delta L \le \Delta x - 100 \wedge \Delta x > \overrightarrow{T} + 100 \wedge 6(\Delta L) \le \Delta x$
  ψ_4: $\big(\overrightarrow{at}\_\ell_{0,1} \wedge \Delta x \ge 110\big) \;\rightarrow\; \Delta x \ge 50 + \overrightarrow{T} + 6(\Delta L - \overrightarrow{T})$
- $\Delta x \ge 60 \Rightarrow_f 6(\Delta L) \le \Delta x$

The second property is used to prove the third property using rule I-MON presented in Figure 8.4.

$$\boxed{\begin{array}{lll}
\text{I-MON} & \text{IM1.} & \Box_f \varphi_1(\overleftarrow{V}, \overrightarrow{V}) \\
 & \text{IM2.} & \Box_f \varphi_2(\overleftarrow{V}, \overrightarrow{V}) \\
 & \text{IM3.} & \varphi_1(\overleftarrow{V}, \overrightarrow{V}) \wedge \varphi_2(\overleftarrow{V}, \overrightarrow{V}) \Rightarrow_f \psi(\overleftarrow{V}, \overrightarrow{V}) \\
\hline
 & & \Box_f \psi(\overrightarrow{V}, \overleftarrow{V})
\end{array}}$$

Figure 8.4: Rule I-MON—Monotonicity of interval invariance formulas

8.4 Soundness of Proof Rules

We now prove the soundness of the rules P-INV and I-INV. The other rules are proved similarly.

Theorem 32 *Rule* P-INV *is sound.*

Proof of Soundness of P-INV:

Let $\mathcal{S} = (V, \Phi, \Theta, \mathcal{T})$ be an arbitrary CPTS.
Suppose φ, χ, ψ are state formulas such that the premises of rule P-INV hold.
We will show that for any computation $\overline{P}$ of $\mathcal{S}$, that $\overline{P}^* \models \Box(\psi)$.
Let $\overline{P}_1$ be an arbitrary computation of $\mathcal{S}$.
As $\overline{P}_1$ is a computation of $\mathcal{S}$, it is equivalent to a phase sequence of the form $\overline{P}_2 = P_0, P_1, \ldots$ where:

(1) For each $0 \le i < |\overline{P}|$, $P_i = \langle [a_i, a_{i+1}), f_i \rangle$

(2) Θ holds at a_0.

(3) For all $0 \le i < |\overline{P}|$, there is a phase invariant $\rho_\phi \in \Phi$ such that P_i is a ϕ-phase.

(4) For all $0 \le i < |\overline{P}| - 1$, there is a transition $\tau \in \mathcal{T}$ such that $\rho_\tau(\overrightarrow{P_i}[V], \overleftarrow{P_{i+1}}[V])$ holds.

(5) $\overline{P}$ is divergent.

We proceed to prove that φ and ψ hold at all $t \in [a_0, \infty)$. The proof is by induction on j, $0 \le j < |\overline{P}|$, showing that φ and ψ hold at all $t \in [a_j, a_{j+1})$.

Assume that $\forall k \in [0..j-1]$, we have already shown that φ and ψ hold at all $t \in [a_k, a_{k+1})$. We will show that φ and ψ hold at all $t \in [a_j, a_{j+1})$.

Case: $t = a_j$ and $j = 0$

By requirement (2) above, Θ holds at a_0. As premise PI1 holds, φ holds at a_0. As premise PI2 holds, ψ holds at a_0.

Case: $t = a_j$ and $j \neq 0$

By requirement (4) above, there is a transition $\tau \in \mathcal{T}$ such that $\rho_\tau(\overrightarrow{P_{j-1}}[V], \overleftarrow{P_j}[V])$ holds. Fix such a τ. By requirement (3) above, there is a phase invariant $\rho_\phi \in \Phi$ such that P_{j-1} is a ϕ-phase. Fix such a phase invariant. Thus $P_{j-1} \models$ *continuous* $\wedge \; \Box \, \rho_\phi(V, \dot{V})$. By the induction hypothesis, φ and ψ hold for all $t \in [a_{j-1}, a_j)$. Thus $P_{j-1} \models \varphi(V)$. So by premise PI5, $P_{j-1} \models \chi(\overrightarrow{V})$. As $\rho_\tau(\overrightarrow{P_{j-1}}[V], \overleftarrow{P_j}[V])$ holds, by premise PI3, $P_j \models \varphi(\overleftarrow{V})$. That is, φ holds at $a_j = t$. By premise PI2, ψ holds at t.

Case: $t \in (a_j, a_{j+1})$

By requirement (3) above, there is a phase invariant $\rho_\phi \in \Phi$ such that P_j is a ϕ-phase. Fix such a phase invariant. Consider the subphase $\widehat{P}_j = \langle [a_j, t), \widehat{f} \rangle$, where $\widehat{f}$ is the restriction of f to $[a_j, t)$. Obviously, $\widehat{P}_j$ is also a ϕ-phase. In particular, $\widehat{P}_j \models$ *continuous* $\wedge \; \Box \, \rho_\phi(V, \dot{V})$. By the previous two cases, φ holds at a_j. As φ is a state formula, we have $\widehat{P}_j \models \varphi(V)$. So by premise PI5, $\widehat{P}_j \models \chi(\overrightarrow{V})$. That is, $\chi(\overrightarrow{V})$ holds at t. As P_j is continuous and t is an internal point in $[a_j, a_{j+1})$, we conclude that $\chi(V)$ holds at t. Since t is internal to $[a_j, a_{j+1})$, ρ_ϕ holds at t. By premise PI4, φ holds at t. So by premise PI2, ψ holds at t.

So by induction, φ and ψ hold for all $t \in [a_0, \infty)$. Thus $\Box \, \psi(V)$ holds by Theorem 2. ∎

Theorem 33 *Rule* I-INV *is sound.*

Proof of Soundness of I-INV**:**

Let $\mathcal{S} = (V, \Phi, \Theta, \mathcal{T})$ be an arbitrary CPTS.
Suppose $\varphi(V, V')$ and $\psi(V, V')$ are state formulas such that the premises of rule I-INV hold.
We will show that for any computation $\overline{P}$ of $\mathcal{S}$, $\overline{P}^* \models \Box \, \psi(\overleftarrow{V}, \overrightarrow{V})$.
Let $\overline{P}_1$ be an arbitrary computation of $\mathcal{S}$ and P be an arbitrary finite subphase of $\overline{P}_1^*$. As P is a finite subphase of $\overline{P}_1^*$, it must be equivalent to a sequence of adjacent phases $P_1, \ldots, P_n$ $(n \geq 1)$ such that

(1) For each $i \in [1..n]$, there is a phase invariant $\rho_\phi \in \Phi$ such that P_i is a ϕ-phase.

(2) For each $i \in [1..n-1]$, there is a transition $\tau \in \mathcal{T}$ such that $\rho_\tau(\overrightarrow{P_i}[V], \overleftarrow{P_{i+1}}[V])$ holds.

We proceed by induction on $t \in [1..n]$ to show that $\varphi(\overleftarrow{V}, \overrightarrow{V})$ holds over the phase $P_{1..t} = P_1 {}^\frown P_2 {}^\frown \cdots {}^\frown P_t$.

Case: Base $(t = 1)$

By requirement (1) above, there is a phase invariant $\rho_\phi \in \Phi$ such that P_1 is a ϕ-phase. That is, $P_1 \models$ *continuous* $\wedge \; \Box \, \rho_\phi(V, \dot{V})$. By premise II2, $P_1 \models \varphi(\overleftarrow{V}, \overrightarrow{V})$, and since $P_{1..1} = P_1$, the induction claim holds for $t = 1$.

Case: Inductive (from t to $t + 1 \leq n$)

Let phases $P_{1..t}$ and P_{t+1} be given by $\langle [a,b), g_t \rangle$ and $\langle [b,c), g_{t+1} \rangle$, respectively. Let U_1, U_2, and U_3 denote the values of $\overleftarrow{P_{1..t}}[V]$, $\overrightarrow{P_{1..t}}[V] = \overrightarrow{P_t}[V]$, and $\overleftarrow{P_{t+1}}[V]$, respectively.

By requirement (2) above, there is a transition $\tau \in \mathcal{T}$ such that $\rho_\tau(\overrightarrow{P_t}[V], \overleftarrow{P_{t+1}}[V])$ holds. By requirement (1) above, there is a phase invariant $\rho_\phi \in \Phi$ such that P_{t+1} is a ϕ-phase. Thus, $P_{t+1} \models \mathit{continuous} \wedge \Box\, \rho_\phi(V, \dot{V})$. By the induction hypothesis, $P_{1..t} \models \varphi(\overleftarrow{V}, \overrightarrow{V})$, which implies that $\varphi(U_1, U_2) = \mathit{true}$ (that is, $\varphi(V_1, V_2)$ evaluates to *true* when we interpret V_1 as U_1 and V_2 as U_2). In a similar way, P_{t+1} being a τ-successor of P_t implies that $\rho_\tau(U_2, U_3) = \mathit{true}$. Consider now the augmented phase $\widehat{P}_{t+1}$: $\langle [b,c), \widehat{g}_{t+1} \rangle$ where $\widehat{g}_{t+1}$ agrees with g_{t+1} on the values of V. That is, $\widehat{g}_{t+1}[V](r) = g_{t+1}[V](r)$ for each $r \in [b,c)$ and, in addition, $\widehat{g}_{t+1}$ interprets the additional variables V_1 and V_2 as the *constant* values U_1 and U_2, respectively. It is not difficult to see that the conjunction $\varphi(V_1, V_2) \wedge \rho_\tau(V_2, V_3) \wedge \mathit{continuous} \wedge \Box\, \rho_\phi(V_3, \dot{V}_3)$ holds over the phase $\widehat{P}_{t+1}$.

So by premise II3, $\widehat{P}_{t+1} \models \varphi(V_1, \overrightarrow{V})$. Since $\widehat{P}_{t+1}[V_1] = U_1 = P_{1..t}[V] = P_{1..t+1}[V]$ and $\overrightarrow{\widehat{P}_{t+1}}[V] = \overrightarrow{P_{t+1}}[V] = \overrightarrow{P_{1..t+1}}[V]$, it follows that $P_{1..t+1} \models \varphi(\overleftarrow{V}, \overrightarrow{V})$.

By induction, we conclude that $P_{1..n} \models \varphi(\overleftarrow{V}, \overrightarrow{V})$, which by premise II1, leads to $P_{1..n} \models \psi(\overleftarrow{V}, \overrightarrow{V})$. As P is equivalent to $P_{1..n}$, we have $P \models \psi(\overleftarrow{V}, \overrightarrow{V})$.

Since P was an arbitrary finite phase of the computation $\overline{P}_1$, we get that $\Box\, \psi(\overleftarrow{V}, \overrightarrow{V})$ is an invariant of $\mathcal{S}$. ∎

8.5 Example

We return to system GAS and prove several point-based and interval-based properties.

8.5.1 Proofs of Point-Based Properties

We are interested in proving:

- $T \geq 0$
- $\mathit{at\_}\ell_0 \;\rightarrow\; (0 \leq y < 100 \;\wedge\; 0 \leq T < 10 \;\wedge\; R = \mathrm{Off} \;\wedge\; \mathit{switch} = \mathrm{Off})$
- $\mathit{at\_}\ell_1 \;\rightarrow\; (0 \leq y < 100 \;\wedge\; T = 10 \;\wedge\; \mathit{switch} = \mathrm{Off})$
- $\mathit{at\_}\ell_2 \;\rightarrow\; (0 \leq y < 10 \;\wedge\; 0 \leq T < 10 \;\wedge\; R = \mathrm{On} \;\wedge\; \mathit{switch} = \mathrm{On})$

We prove the second of these four formulas; the others are proved similarly.

Proof of $\mathit{at\_}\ell_0 \;\rightarrow\; (0 \leq y < 100 \;\wedge\; 0 \leq T < 10 \;\wedge\; R = \mathrm{Off} \;\wedge\; \mathit{switch} = \mathrm{Off})$:

We take:

$$\begin{array}{rl} \psi,\ \varphi\text{:} & \mathit{at\_}\ell_0 \;\rightarrow\; (0 \leq y < 100 \;\wedge\; 0 \leq T < 10 \;\wedge\; R = \mathrm{Off} \;\wedge\; \mathit{switch} = \mathrm{Off}) \\ \chi\text{:} & \mathit{at\_}\ell_0 \;\rightarrow\; (0 \leq y \leq 100 \;\wedge\; 0 \leq T < 10 \;\wedge\; R = \mathrm{Off} \;\wedge\; \mathit{switch} = \mathrm{Off}) \end{array}$$

PI1: $\Theta \ \rightarrow \ \varphi(V)$
$\left[R = \text{Off} \ \wedge \ T = 0 \ \wedge \ y = 0 \ \wedge \ \mathit{switch} = \text{Off} \ \wedge \ \pi = \ell_0 \ \wedge \ \ldots\right] \ \rightarrow$
$\left[\mathit{at\_}\ell_0 \ \rightarrow \ (0 \leq y < 100 \ \wedge \ 0 \leq T < 10 \ \wedge \ R = \text{Off} \ \wedge \ \mathit{switch} = \text{Off})\right]$
which clearly holds.

PI2: $\varphi(V) \ \rightarrow \ \psi(V)$
As $\varphi(V)$ and $\psi(V)$ are the same formulas, we get $\varphi(V) \ \rightarrow \ \psi(V)$.

PI3: $\rho_\tau(V, V') \wedge \chi(V) \rightarrow \varphi(V')$ for every $\tau \in \mathcal{T}$
We only need to consider transitions of the form $\rho_{\langle \ell_i, \ell_0 \rangle}$ for $i \in \{0, 1, 2\}$ since for all other transitions $\pi' \neq \ell_0$, making the antecedent of $\varphi(V')$ false. Thus we have three transitions, $\rho_{\langle \ell_0, \ell_0 \rangle}$, $\rho_{\langle \ell_1, \ell_0 \rangle}$, and $\rho_{\langle \ell_2, \ell_0 \rangle}$, to consider.

$\rho_{\langle \ell_0, \ell_0 \rangle}$:

$$\begin{bmatrix} \pi = \ell_0 \ \wedge \ R' = R \\ \wedge \ T' = 0 \ \wedge \ y' = 0 \wedge \ \pi' = \ell_0 \\ \wedge \ \mathit{switch}' = \text{Off} \ \wedge \ \ldots \end{bmatrix} \ \wedge \ \left[\mathit{at\_}\ell_0 \ \rightarrow \ (R = \text{Off} \ \wedge \ \ldots)\right]$$
$$\rightarrow \ \left[\mathit{at}'\mathit{\_}\ell_0 \ \rightarrow \ \begin{pmatrix} 0 \leq y' < 100 \ \wedge \ 0 \leq T' < 10 \\ \wedge \ R' = \text{Off} \ \wedge \ \mathit{switch}' = \text{Off} \end{pmatrix}\right]$$

$\rho_{\langle \ell_1, \ell_0 \rangle}$:

$$\begin{bmatrix} R' = \text{Off} \ \wedge \ T' = 0 \ \wedge \ y' = 0 \\ \wedge \ \mathit{switch}' = \text{Off} \ \wedge \ \pi' = \ell_0 \ \wedge \ \ldots \end{bmatrix} \ \wedge \ \chi(V)$$
$$\rightarrow \ \left[\mathit{at}'\mathit{\_}\ell_0 \ \rightarrow \ \begin{pmatrix} 0 \leq y' < 100 \ \wedge \ 0 \leq T' < 10 \\ \wedge \ R' = \text{Off} \ \wedge \ \mathit{switch}' = \text{Off} \end{pmatrix}\right]$$

$\rho_{\langle \ell_2, \ell_0 \rangle}$:

$$\begin{bmatrix} R' = \text{Off} \ \wedge \ T < 10 \ \wedge \ T' = T \ \wedge \ y' = 0 \\ \wedge \ \mathit{switch}' = \text{Off} \ \wedge \ \pi' = \ell_0 \ \wedge \ \ldots \end{bmatrix} \ \wedge \ \chi(V)$$
$$\rightarrow \ \left[\mathit{at}'\mathit{\_}\ell_0 \ \rightarrow \ \begin{pmatrix} 0 \leq y' < 100 \ \wedge \ 0 \leq T' < 10 \\ \wedge \ R' = \text{Off} \ \wedge \ \mathit{switch}' = \text{Off} \end{pmatrix}\right]$$

The first two formulas are valid formulas, while the third formula also requires the previously established invariant $T \geq 0$.

PI4: $\rho_\phi(V, \dot{V}) \wedge \chi(V) \rightarrow \varphi(V)$ for every $\phi \in \Phi$
We only have to consider the phase relation, ρ_{ℓ_0}, since the other phase relations have $\pi \neq \ell_0$, making the antecedent false.

$$\left[y < 100 \ \wedge \ \ldots\right] \ \wedge \ \left[\mathit{at\_}\ell_0 \ \rightarrow \ \begin{pmatrix} 0 \leq y \leq 100 \ \wedge \ 0 \leq T < 10 \\ \wedge \ R = \text{Off} \ \wedge \ \mathit{switch} = \text{Off} \end{pmatrix}\right]$$
$$\rightarrow \ \left[\mathit{at\_}\ell_0 \ \rightarrow \ \begin{pmatrix} 0 \leq y < 100 \ \wedge \ 0 \leq T < 10 \\ \wedge \ R = \text{Off} \ \wedge \ \mathit{switch} = \text{Off} \end{pmatrix}\right]$$

PI5: $\mathit{continuous} \wedge \Box\, \rho_\phi(V, \dot{V}) \wedge \varphi(V) \Rightarrow_f \chi(\overrightarrow{V})$ for every $\phi \in \Phi$

We only have to consider the phase relation, ρ_{ℓ_0}, since the other phase relations have $\overrightarrow{\pi} \neq \ell_0$, making the antecedent false.

$$\mathit{continuous} \quad \wedge \Box \begin{bmatrix} \dot{y} = 1 \ \wedge \ \dot{T} = 0 \ \wedge \\ y < 100 \ \wedge \ R = \text{Off} \\ \wedge \ \pi = \ell_0 \ \wedge \\ \mathit{switch} = \text{Off} \ \wedge \ \ldots \end{bmatrix} \quad \wedge \quad \begin{bmatrix} \mathit{at\_}\ell_0 \ \rightarrow \\ \quad \begin{pmatrix} 0 \le y < 100 \ \wedge \\ 0 \le T < 10 \ \wedge \ \ldots \end{pmatrix} \end{bmatrix}$$

$$\Rightarrow_f \left[\overrightarrow{\mathit{at}}\_\ell_0 \ \rightarrow \ \begin{pmatrix} 0 \le \overrightarrow{y} \le 100 \ \wedge \ 0 \le \overrightarrow{T} < 10 \\ \wedge \ \overrightarrow{R} = \text{Off} \ \wedge \ \overrightarrow{\mathit{switch}} = \text{Off} \end{pmatrix} \right]$$

Consider an arbitrary phase P. Suppose $P \models \mathit{continuous} \wedge \Box\, \rho_{\ell_0} \wedge \varphi(V)$. As $\mathit{continuous} \wedge \Box(\dot{T} = 0) \Rightarrow_f \overleftarrow{T} = \overrightarrow{T}$, $P \models \overleftarrow{T} = \overrightarrow{T}$, and as $0 \le T < 10$, $P \models 0 \le \overrightarrow{T} < 10$. As $0 \le y < 100 \wedge \Box(\dot{y} = 1)$, $P \models 0 \le \overrightarrow{y}$. As $\mathit{continuous} \wedge \Box(y < 100)$, $P \models \overrightarrow{y} \le 100$. As $\mathit{continuous} \wedge \Box(R = \text{Off})$, $P \models \overrightarrow{R} = \text{Off}$. As $\mathit{continuous} \wedge \Box(\mathit{switch} = \text{Off})$, $P \models \overrightarrow{\mathit{switch}} = \text{Off}$. Thus, $P \models \chi(\overrightarrow{V})$. ∎

8.5.2 Proofs of Interval-Based Properties

We are interested in proving:

- $\overrightarrow{\mathit{at}}\_\ell_{0,1} \Rightarrow_f \big((\Delta x \le \overrightarrow{y} \wedge \Delta L = 0) \ \vee \ (\Delta x > \overrightarrow{y} \wedge \Delta L < \Delta x - \overrightarrow{y})\big)$
- $\Box_f \big[(\psi_1 \ \vee \psi_2 \ \vee \psi_3) \ \wedge \ \psi_4\big]$ where
  - ψ_1: $\Delta L \le \Delta T$
  - ψ_2: $\Delta L \le \overrightarrow{T} \wedge \Delta x \le \overrightarrow{T} + 100$
  - ψ_3: $\Delta L \le \Delta x - 100 \wedge \Delta x > \overrightarrow{T} + 100 \wedge 6(\Delta L) \le \Delta x$
  - ψ_4: $\big(\overrightarrow{\mathit{at}}\_\ell_{0,1} \wedge \Delta x \ge 110\big) \ \rightarrow \ \Delta x \ge 50 + \overrightarrow{T} + 6(\Delta L - \overrightarrow{T})$
- $\Delta x \ge 60 \Rightarrow_f 6(\Delta L) \le \Delta x$

We prove the second formula below. The proof of the first formula is done in a similar manner. The third formula which is the safety requirement for the gas burner, follows from rule I-MON and the second formula.

Proof of $\Box_f \big[(\psi_1 \ \vee \psi_2 \ \vee \psi_3) \ \wedge \ \psi_4\big]$**:**

We take:

$$\psi, \ \varphi\text{:} \ \big[(\psi_1 \ \vee \psi_2 \ \vee \psi_3) \ \wedge \ \psi_4\big]$$

II1: $\varphi(\overleftarrow{V}, \overrightarrow{V}) \ \rightarrow \ \psi(\overleftarrow{V}, \overrightarrow{V})$
As $\varphi(\overleftarrow{V}, \overrightarrow{V})$ and $\psi(\overleftarrow{V}, \overrightarrow{V})$ are the same formulas, we get $\varphi(\overleftarrow{V}, \overrightarrow{V}) \ \rightarrow \ \psi(\overleftarrow{V}, \overrightarrow{V})$.

II2: $\mathit{continuous} \wedge \Box\, \rho_\phi(V, \dot{V}) \Rightarrow_f \varphi(\overleftarrow{V}, \overrightarrow{V})$ for every $\phi \in \Phi$
We must consider all three phase relations.

ρ_{ℓ_0}:

$$\mathit{continuous} \;\wedge\; \Box \left[\begin{array}{l} \dot{x}=1 \;\wedge\; \dot{y}=1 \;\wedge\; \dot{L}=0 \;\wedge\; \dot{T}=0 \;\wedge \\ y<100 \;\wedge\; R=\text{Off} \;\wedge\; \pi=\ell_0 \;\wedge\; \mathit{switch}=\text{Off} \end{array}\right] \Rightarrow_f \left[(\psi_1 \;\vee \psi_2 \;\vee \psi_3) \;\wedge\; \psi_4\right]$$

As $\dot{L}=0$, $\dot{T}=0$, and *continuous*, we immediately get $\overrightarrow{L}=\overleftarrow{L}$ and $\overrightarrow{T}=\overleftarrow{T}$. These facts make the first conjunct in the consequent of $\varphi(\overleftarrow{V},\overrightarrow{V})$ true. By a previously established point invariant, we get $0 \le \overrightarrow{y} \le 100$ and $0 \le \overleftarrow{y} < 100$, so $\overrightarrow{y}-\overleftarrow{y} \le 100$. As $\Delta x = \overrightarrow{y}-\overleftarrow{y} \le 100$, the second conjunct is also true.

ρ_{ℓ_1}:

$$\mathit{continuous} \;\wedge\; \Box \left[\begin{array}{l} \dot{x}=1 \;\wedge\; \dot{y}=1 \;\wedge\; \dot{L}=0 \;\wedge\; \dot{T}=0 \;\wedge \\ y<100 \;\wedge\; \pi=\ell_1 \;\wedge\; \mathit{switch}=\text{Off} \end{array}\right] \Rightarrow_f \left[(\psi_1 \;\vee \psi_2 \;\vee \psi_3) \;\wedge\; \psi_4\right]$$

As $\dot{L}=0, \dot{T}=0$, and *continuous*, we immediately get $\overrightarrow{L}=\overleftarrow{L}$ and $\overrightarrow{T}=\overleftarrow{T}$. These facts make the first conjunct in the consequent of $\varphi(\overleftarrow{V},\overrightarrow{V})$ true. As $\Delta x = \overrightarrow{y}-\overleftarrow{y} \le 100$, the second conjunct is also true.

ρ_{ℓ_2}:

$$\mathit{continuous} \;\wedge\; \Box \left[\begin{array}{l} \dot{x}=1 \;\wedge\; \dot{y}=1 \;\wedge\; \dot{L}\le 1 \;\wedge\; \dot{T}=1 \;\wedge \\ R=\text{On} \;\wedge\; T<10 \;\wedge\; \pi=\ell_2 \;\wedge\; \mathit{switch}=\text{On} \end{array}\right] \Rightarrow_f \left[(\psi_1 \;\vee \psi_2 \;\vee \psi_3) \;\wedge\; \psi_4\right]$$

As $\overrightarrow{at}\_\ell_{0,1}$ is false, the second conjunct in the consequent is true. As *continuous* and $\Box(\dot{L} \le \dot{T}=1)$ implies $\Delta L \le \Delta T$, the first conjunct in the consequent is true.

II3: $\varphi(V_1,V_2) \wedge \rho_\tau(V_2,V) \wedge \mathit{continuous} \wedge \Box\,\rho_\phi(V,\overrightarrow{V}) \Rightarrow_f \varphi(V_1,\overrightarrow{V})$ for every $\tau \in \mathcal{T}$ and for every $\phi \in \Phi$.

There are seven cases to consider (one for each transition).

$\rho_{\langle \ell_0,\ell_0\rangle}, \rho_{\ell_0}$:

As $\Box(\pi=\ell_0)$ and *continuous* implies $\overrightarrow{at}\_\ell_0$, we get:

$$(\Delta_1 x \le \overrightarrow{y} \;\wedge\; \Delta_1 L = 0) \;\vee\; (\Delta_1 x > \overrightarrow{y} \;\wedge\; \Delta_1 L < \Delta_1 x - \overrightarrow{y})$$

continuous and $\Box(y<100)$ implies $\overrightarrow{y} \le 100$. *continuous* and $\Box(\dot{L}=\dot{T}=0)$ implies $\overrightarrow{L}=L=L_2$ and $\overrightarrow{T}=T=0$.

Case: $(\Delta_1 x \le \overrightarrow{y} \;\wedge\; \Delta_1 L = 0)$

In this case, $\Delta_1 L \le 0 = \overrightarrow{T}$ and $\Delta_1 x \le \overrightarrow{y} \le 100 \le \overrightarrow{T}+100$. So the first conjunct of the consequent holds. As $\Delta_1 x \le 100$, the second conjunct of the consequent holds.

Case: $(\Delta_1 x > \overrightarrow{y} \;\wedge\; \Delta_1 L < \Delta_1 x - \overrightarrow{y})$

subcase: $\Delta_1 x = \overrightarrow{y}$

In this case, $\Delta_1 x \le \overrightarrow{T} + 100$ and $\Delta_1 L \le 0 = \overrightarrow{T}$, so the first conjunct of the consequent holds. As $\Delta_1 L \le 0$, the second conjunct holds.

subcase: $\Delta_1 x > \overrightarrow{y}$

subcase: $\overrightarrow{y} = 100$

In this case $\Delta_1 x > \overrightarrow{T} + 100$ and $\Delta_1 L \le \Delta_1 x - \overrightarrow{y} \le \Delta_1 x - 100$. Either $\Delta_1^2 L < 10$ or $(\Delta_1^2 L \le \Delta_1^2 x - 100 \wedge 6(\Delta_1^2 L) \le \Delta_1^2 x)$. In the first case, we get $6(\Delta_1^2 L) < \Delta_1 x$, and so the first conjunct holds. In the second case, $6(\Delta_1^2 L) \le \Delta_1^2 x \le \Delta_1 x$, and so the first conjunct holds.

subcase: $\overrightarrow{y} < 100$

If $\Delta_1 x \le 100 + \overrightarrow{T}$ then the first invariant gives

$$(\Delta_1^2 x \le y_2 \ \wedge\ \Delta_1^2 L = 0) \ \vee\ (\Delta_1^2 x > y_2 \ \wedge\ \Delta_1^2 L < \Delta_1^2 x - y_2)$$

As $y_2 = 100$ and $x - x_1 \le 100 = y_2$, we get $\Delta_1^2 x \le y_2$. Hence $\Delta_1^2 L = 0$ and $\Delta_1 L = 0 \le \overrightarrow{T}$. Thus the first conjunct holds.

If $\Delta_1 x > 100 + \overrightarrow{T}$, then $\Delta_1 L \le \Delta_1 x - 100$ and $6(\Delta_1 L) \le \Delta_1 x$ as in the subcase $\Delta_1 x = \overrightarrow{y}$. Thus the first conjunct holds.

We still need to show that the second conjunct holds. We consider two cases.

subcase: $\Delta_1 x = 110$

In this case $\Delta_1 L = 0 = \overrightarrow{T}$, so the second conjunct holds.

subcase: $\Delta_1 x > 110$

If $\Delta_1^2 x < 110$ then $\Delta_1^2 L < 10$, and so the second conjunct holds. If $\Delta_1^2 x \ge 110$ then $\Delta_1^2 x \ge 50 + T_2 + 6(\Delta_1^2 L - T_2)$. As $\overrightarrow{L} = L_2$ and $\overrightarrow{T} = 0$, the second conjunct holds.

Thus in all subcases, both conjuncts of the consequent hold.

$\rho_{\langle \ell_0, \ell_2 \rangle}, \rho_{\ell_2}$:

As $\overrightarrow{at}\_\ell_{0,1}$ is false, the second conjunct holds. We still need to prove that the first conjunct of the consequent holds. We consider three cases corresponding to the three disjuncts of the first conjunct in the antecedent.

Case: $\Delta_1^2 L \le \Delta_1^2 x$

As $L = L_2$ and $T = T_2$, we get $L - L_1 \le T - T_1$. As *continuous* and $\Box(\dot{L} \le \dot{T} = 1)$ implies $\overrightarrow{L} - L \le \overrightarrow{T} - T$, we get $\Delta_1 L = \overrightarrow{L} - L + L - L_1 \le \overrightarrow{T} - T + T - T_1 \le \Delta_1 T$. So the first conjunct of the consequent holds.

Case: $\Delta_1^2 L \le T_2$ and $\Delta_1^2 x \le T_2 + 100$

As $x = x_2$ and $T = T_2$, $x - x_1 \le T + 100$. As *continuous* and $\Box(\dot{x} = \dot{T} = 1)$ implies

$\overrightarrow{x} - x = \overrightarrow{T} - T$, we get $\Delta_1 x = \overrightarrow{x} - x + x - x_1 \leq \overrightarrow{T} - T + T + 100 \leq \overrightarrow{T} + 100$. So the first conjunct of the consequent holds.

Case: $(\Delta_1^2 L \leq \Delta_1^2 x - 100)$ and $(\Delta_1^2 x > T_2 + 100)$ and $(6(\Delta_1^2 L) \leq \Delta_1^2 x)$

By reasoning similar to the previous cases, we get $\Delta_1 x > \overrightarrow{T} + 100$ and $\Delta_1 L \leq \Delta_1 x - 100$. We still need to show $6(\Delta_1 L) \leq \Delta_1 x$.

As $at_2\_\ell_0$ holds, by the previous invariant

$$(\Delta_1^2 x \leq y_2 \ \wedge \ \Delta_1^2 L = 0) \ \vee \ (\Delta_1^2 x > y_2 \ \wedge \ \Delta_1^2 L < \Delta_1^2 x - y_2)$$

subcase: $\Delta_1^2 x \leq y_2 \ \wedge \ \Delta_1^2 L = 0$

As $0 \leq T < 10$, *continuous*, and $\square(\dot{L} \leq \dot{T} = 1)$ implies $\overrightarrow{T} - T \geq \overrightarrow{L} - L$, we get $10 \geq \overrightarrow{L} - L \leq \Delta_1 L$. As $\Delta_1 x > 100$, we get $6(\Delta_1 L) \leq 60 \leq 100 \leq \Delta_1 x > 100$.

subcase: $\Delta_1^2 x > y_2 \ \wedge \ \Delta_1^2 L < \Delta_1^2 x - y_2$

subcase: $\Delta_1^2 x < 110$

In this case $\Delta_1^2 L \leq \Delta_1^2 x - 100$ (using the point invariant to give $T_2 = 10$). So, $6(\Delta_1 L) = 6(\overrightarrow{L} - L + L - L_1) = 6(\overrightarrow{L} - L) + 6(L - L_1) \leq 6(\overrightarrow{x} - x) + 6(\Delta_1^2 x - 100) \leq (\overrightarrow{x} - x) + 5(\Delta_1 x) - 600 \leq \Delta_1 x$. The last inequality follows from the fact that $\Delta_1 x < 120$.

subcase: $\Delta_1^2 x \geq 110$

$$\begin{array}{rcl}
\Delta_1 x & = & \overrightarrow{x} - x + x - x_1 \\
& = & \overrightarrow{T} - T + \Delta_1^2 x \\
& = & \overrightarrow{T} - T_2 + \Delta_1^2 x \\
& \geq & \overrightarrow{T} - T_2 + 50 + T_2 + 6(\Delta_1^2 L - T_2) \\
& \geq & \overrightarrow{T} + 50 + 6(\Delta_1^2 L) - 6T_2 \\
& \geq & 6\overrightarrow{T} + 6(\Delta_1^2 L) - 6T_2 \\
& \geq & 6(\overrightarrow{L} - L) + 6(\Delta_1^2 L) \\
& \geq & 6(\Delta_1 L)
\end{array}$$

Thus, the first conjunct holds.

$\rho_{\langle \ell_1, \ell_0 \rangle}, \rho_{\ell_0}$:

This case is exactly the same as $\rho_{\langle \ell_0, \ell_0 \rangle}, \rho_{\ell_0}$.

$\rho_{\langle \ell_1, \ell_1 \rangle}, \rho_{\ell_1}$:

This case is exactly the same as $\rho_{\langle \ell_0, \ell_0 \rangle}, \rho_{\ell_0}$.

$\rho_{\langle \ell_1, \ell_2 \rangle}, \rho_{\ell_2}$:

As $\overrightarrow{at}\_\ell_{0,1}$ is false, the second conjunct holds. We still need to prove that the first conjunct of the consequent holds.
As $at_2\_\ell_0$ holds, by the previous invariant

$$(\Delta_1^2 x \leq y_2 \ \wedge\ \Delta_1^2 L = 0) \ \ \vee\ \ (\Delta_1^2 x > y_2 \ \wedge\ \Delta_1^2 L < \Delta_1^2 x - y_2)$$

Case: $\Delta_1^2 x \leq y_2 \ \wedge\ \Delta_1^2 L = 0$

As $L = L_2 = L_1$, $T = 0$, *continuous*, and $\square(\dot{L} \leq \dot{T} = 1)$ implies $\overrightarrow{T} - T \geq \overrightarrow{L} - L$, we get $\overrightarrow{T} \geq \Delta_1 L$. As $y_2 = 100$, we have $\Delta_1^2 x \leq 100$. Together, *continuous* and $\square(\dot{x} = \dot{T} = 1)$ imply $\overrightarrow{T} - T = \overrightarrow{x} - x$. Thus, $\Delta_1 x \leq \overrightarrow{T} + 100$. So the first conjunct holds.

Case: $\Delta_1^2 x > y_2 \ \wedge\ \Delta_1^2 L < \Delta_1^2 x - y_2$

This case is identical to the third case of $\rho_{\langle \ell_0, \ell_2 \rangle}, \rho_{\ell_2}$. Thus the first conjunct holds.

$\rho_{\langle \ell_2, \ell_0 \rangle}, \rho_{\ell_0}$:

The proof of the first conjunct is very similar to the case $\rho_{\langle \ell_0, \ell_0 \rangle}, \rho_{\ell_0}$. Instead of $\overrightarrow{T} = T_2$, we have $0 \leq \overrightarrow{T} \leq 10$. The proof of the second conjunct is very similar to the case $\rho_{\langle \ell_2, \ell_1 \rangle}, \rho_{\ell_1}$.

$\rho_{\langle \ell_2, \ell_1 \rangle}, \rho_{\ell_1}$:

We consider three cases corresponding to the three disjuncts of the first conjunct in the antecedent.

Case: $\Delta_1^2 L \leq \Delta_1^2 x$

In this case $\Delta_1 L \leq \Delta_1 T \leq 10$, so both the first and second conjunct of the consequent hold.

Case: $\Delta_1^2 L \leq T_2$

In this case $\Delta_1 L \leq \overrightarrow{T} \leq 10$, so both the first and second conjunct of the consequent hold.

Case: $\Delta_1^2 L \leq \Delta_1^2 x - 100$ and $\Delta_1^2 x > T_2 + 100$

If $\Delta_1^2 x \geq 110$ then as $\overrightarrow{T} = T_2$ and $\overrightarrow{L} = L_2$, we get $\Delta_1 x \geq \Delta_1^2 x \geq 50 + T_2 + 6(\Delta_1^2 L - T_2) \geq 50 + \overrightarrow{T} + 6(\Delta_1 L - \overrightarrow{T})$. So the second conjunct holds. If $\Delta_1^2 x < 110$ then $\Delta_1^2 L \leq 10$, and so the second conjunct holds. In either case, $\Delta_1 x \geq \Delta_1^2 x \geq 6(\Delta_1^2 L) \geq 6(\Delta_1 L)$, so the first conjunct holds. ∎

8.6 Computational Induction for HTL

Our motivation for introducing an induction axiom for HTL is based on the following observation by Pnueli [131]:

The invariant [for system GAS that] we know how to prove implies:

$$(\overrightarrow{x} - \overleftarrow{x}) \leq 110 \;\rightarrow\; (\overrightarrow{L} - \overleftarrow{L}) \leq 10. \tag{8.1}$$

In particular, this holds if $\overrightarrow{x} - \overleftarrow{x} = 110$. Consider an interval I of length M, and let $n = \lfloor \frac{M}{110} \rfloor$, that is, n is the largest integer not exceeding $M/110$. Then, we can obviously partition I into $n+1$ subintervals, each of length not exceeding 110. By the above, (since L is continuous) $\overrightarrow{L} - \overleftarrow{L}$ in the big interval should be the sum of $(\overrightarrow{L} - \overleftarrow{L})$'s over the subintervals, which lead to $(\overrightarrow{L} - \overleftarrow{L}) \leq (n+1)10 \leq (1 + M/110)10$, from which we get

$$11(\overrightarrow{L} - \overleftarrow{L}) \leq (110 + \overrightarrow{x} - \overleftarrow{x}). \tag{8.2}$$

Properties (1) + (2) imply the desired property:

$$(\overrightarrow{x} - \overleftarrow{x}) \geq 60 \;\rightarrow\; 6(\overrightarrow{L} - \overleftarrow{L}) \leq (\overrightarrow{x} - \overleftarrow{x}). \tag{8.3}$$

We would like a convenient way of proving (8.3) directly from (8.1). The induction axiom that we introduce below will allow us to achieve this goal. For an arbitrary finite phase $P = \langle [a,b), f \rangle$, we denote $|P| = b - a$.

Computational Induction Axiom: To prove $P \models \Box\, \psi(x_1, \ldots, x_m)$, where P is an arbitrary finite phase, $\psi(x_1, \ldots, x_m)$ is an HTL formula with no temporal operators, $\{x_1, \ldots, x_m\}$ are all the free variables in ψ, and *continuous*$(\{x_1, \ldots, x_m\})$ holds, it suffices to prove:

if for some constant $L \in \mathbb{N}$,

$$(\forall n \in \mathbb{N}) \left[\begin{array}{ll} \text{If } (\forall k \in \mathbb{N}, k < n) & \left[\begin{array}{l} \text{for any } P', P^1, \ldots, P^k \\ \text{where } |P'| \leq L \text{ and } |P^i| = L \ (\forall i \in [1..k]) \\ P' \frown P^1 \frown \cdots \frown P^k \models \psi(x_1, \ldots, x_m) \end{array} \right] \\ \text{then} & \left[\begin{array}{l} \text{for any } P', P^1, \ldots, P^n \\ \text{where } |P'| \leq L \text{ and } |P^i| = L \ (\forall i \in [..n]) \\ P' \frown P^1 \frown \cdots \frown P^n \models \psi(x_1, \ldots, x_m) \end{array} \right] \end{array} \right]$$

then $P \models \Box\, \psi(x_1, \ldots, x_m)$.

The constant L is known as the *subinterval-duration-size.*

The induction axiom can be used to prove both HTL-validities (i.e., statements true for arbitrary finite phases) and system-validities (i.e., statements true for all finite runs of a system). In proving HTL-validities we may only use other HTL-validities in our induction proof. For example, we may readily use the following in any induction proof:

$$\text{if} \left[\begin{array}{l} P = P'\,{}^\frown P^1\,{}^\frown \cdots {}^\frown P^k \text{ and } \mathit{continuous}(\{x_1,\ldots,x_m\}) \text{ and} \\ (\forall j \in [1..m]) \left[\begin{array}{l} (\forall i, 1 \leq i \leq k)\ (P^i \models \overrightarrow{x_j} = u^i_j \text{ and } P^j \models \overleftarrow{x_j} = v^i_j) \\ \text{and } P' \models \overrightarrow{x_j} = u'_j \text{ and } P' \models \overleftarrow{x_j} = v'_j \end{array}\right] \\ \text{where } u^i_j,\, v^i_j,\, u'_j, \text{ and } v'_j \text{ are rigid variables} \end{array}\right] \tag{8.4}$$

$$\text{then } (\forall j \in [1..m])\ (\forall i \in [1..k-1])\ \left[u^i_j = v^{i+1}_j \text{ and } u'_j = v^1_j\right]$$

If we are proving a system-validity, then we may use any proven invariant of the system in our induction proof.

Before we give an example induction proof, we need to introduce an axiom relating the length of an interval with derivatives of variables.

Derivative-Duration Axiom: For all finite phases P, if $|P| = L$ then $P \models \Box(\dot{x} = c) \rightarrow (\overrightarrow{x} - \overleftarrow{x}) = cL$.

We illustrate the use of the induction axiom on system GAS.

Example: Suppose we wish to prove the safety specification:

$$\Box\,\psi(x, L) = \Box\left[(\overrightarrow{x} - \overleftarrow{x}) \geq 60 \ \rightarrow\ 6(\overrightarrow{L} - \overleftarrow{L}) \leq (\overrightarrow{x} - \overleftarrow{x})\right]$$

where x and L are the free variables in ψ and are known to be continuous.[2] Moreover, suppose we have already established the following invariants:

$$\begin{array}{ll} \varphi_0\text{:} & \Box(\dot{L} \leq 1) \\ \varphi_1\text{:} & \Box(\dot{x} = 1) \\ \varphi_2\text{:} & (\overrightarrow{x} - \overleftarrow{x}) \leq 110 \ \rightarrow\ (\overrightarrow{L} - \overleftarrow{L}) \leq 10 \end{array}$$

Proof of $(\overrightarrow{x} - \overleftarrow{x}) \geq 60 \ \rightarrow\ 6(\overrightarrow{L} - \overleftarrow{L}) \leq (\overrightarrow{x} - \overleftarrow{x})$:

Fix the subinterval-duration-size to be 110.

Case: $n = 0$

We must show that $P' \models (\overrightarrow{x} - \overleftarrow{x}) \geq 60 \ \rightarrow\ 6(\overrightarrow{L} - \overleftarrow{L}) \leq (\overrightarrow{x} - \overleftarrow{x})$ where $|P'| \leq 110$. Suppose $\overrightarrow{x} - \overleftarrow{x} \geq 60$. By φ_1 and the derivative-duration axiom, $\overrightarrow{x} - \overleftarrow{x} \leq 110$. So by φ_2, $\overrightarrow{L} - \overleftarrow{L} \leq 10$, whence $6(\overrightarrow{L} - \overleftarrow{L}) \leq 60 \leq (\overrightarrow{x} - \overleftarrow{x})$ as desired.

Case: $n > 0$

[2]Recall, in system GAS, x and L are indeed continuous, since they are system variables that never get reset.

Suppose that for all $k < n$ and arbitrary adjacent phases $P', P^1, \ldots, P^k$ with $|P'| \leq L$ and $|P^i| = L$ $(\forall i \in [1..k])$ we have $P' \frown P^1 \frown \cdots \frown P^k \models \psi(x, L)$.
Let $P', P^1, \ldots, P^n$ be arbitrary adjacent phases with $|P'| \leq L$ and $|P^i| = L$ $(\forall i \in [1..n])$.
To show: $P' \frown P^1 \frown \cdots \frown P^n \models \psi(x, L)$.
For all $i \in [1..n]$, let u^i_x, v^i_x, u'_x, v'_x u^i_L, v^i_L, u'_L, and v'_L be as in (8.4). Thus our induction hypothesis yields $(u^{n-1}_x - v'_x) \geq 60 \;\rightarrow\; 6(u^{n-1}_L - v'_L) \leq (u^{n-1}_x - v'_x)$. We must show $(u^n_x - v'_x) \geq 60 \;\rightarrow\; 6(u^n_L - v'_L) \leq (u^n_x - v'_x)$. As $n > 0$, we know $(u^n_x - v'_x) \geq 110 > 60$, so we must show $6(u^n_L - v'_L) \leq (u^n_x - v'_x)$.

Case: $n = 1$

As $|P'| \leq 110$, by φ_0, φ_1, φ_2, and the derivative-duration axiom, we get $(u'_L - v'_L) \leq \min\{(u'_x - v'_x), 10\}$. As $|P^1| = 110$, by φ_1, φ_2, and the derivative-duration axiom, we get $(u^1_L - v^1_L) \leq 10$. Moreover by (8.4), $u^1_L = v'_L$. So $6(u^n_L - v'_L) \leq (u^n_x - v'_x)$ as desired.

Case: $n > 1$

As $|P' \frown P^1 \frown \cdots \frown P^{n-1}| \geq 60$, $6(u^{n-1}_L - v'_L) \leq (u^{n-1}_x - v'_x)$. As $|P^n| = 110$, by φ_1, φ_2, and the derivative-duration axiom, we get $(u^n_L - v^n_L) \leq 10$. Thus,

$$\begin{array}{rcll}
6(u^n_L - v'_L) & \leq & 6(u^n_L - v^n_L) + 6(v^n_L - v'_L) & \\
 & = & 6(u^n_L - v^n_L) + 6(u^{n-1}_L - v'_L) & \text{by (8.4)} \\
 & \leq & 60 + (u^{n-1}_x - v'_x) & \text{by hypothesis and } (u^n_L - v^n_L) \leq 10 \\
 & \leq & (u^n_x - v^n_x) + (u^{n-1}_x - v'_x) & \text{as } (u^n_x - v^n_x) = 110 \\
 & = & (u^n_x - u^{n-1}_x) + (u^{n-1}_x - v'_x) & \text{by (8.4)} \\
 & = & (u^n_x - v'_x) &
\end{array}$$

Thus, $6(u^n_L - v'_L) \leq (u^n_x - v'_x)$ as desired. ∎

One might wonder why we added induction to our proof system. Induction is not just a matter of convenience. It is necessary. Our proof rules presented in Chapters 8.2 and 8.3 can only be used to prove system-properties. Induction is a powerful tool that allows us to prove HTL-properties as well as system-properties. Soundness of the rule is proven below.

Theorem 34 *The computational induction axiom is sound.*

Proof of Theorem 34:

Let $\psi(x_1, \ldots, x_m)$ be an HTL-formula with free variables in $\{x_1, \ldots, x_m\}$ all of which are continuous. Let L be the subinterval-duration-size.

Suppose

$$(\forall n \in \mathbb{N}) \left[\begin{array}{ll} \text{If } (\forall k \in \mathbb{N}, k < n) & \left[\begin{array}{l} \text{for any } P', P^1, \ldots, P^k \\ \text{where } |P'| \leq L \text{ and } |P^i| = L \ (\forall i \in [1..k]) \\ P' \frown P^1 \frown \cdots \frown P^k \models \psi(x_1, \ldots, x_m) \end{array} \right] \\ \text{then} & \left[\begin{array}{l} \text{for any } P', P^1, \ldots, P^n \\ \text{where } |P'| \leq L \text{ and } |P^i| = L \ (\forall i \in [1..n]) \\ P' \frown P^1 \frown \cdots \frown P^n \models \psi(x_1, \ldots, x_m) \end{array} \right] \end{array} \right] \quad (8.5)$$

We must show that for any arbitrary finite phase P, $P \models \psi(x_1, \ldots, x_m)$. We will prove that for any finite phase P, where $rL < |P| \leq (r+1)L$, $P \models \psi(x_1, \ldots, x_m)$. The proof will proceed by natural number induction on r.

Case: $r = 0$

Then (8.5) for $n = 0$ reduces to $P' \models \psi(x_1, \ldots, x_m)$, where P' is an arbitrary finite phase such that $|P'| \leq L$. As the duration of a phase is positive, we immediately get $P \models \psi(x_1, \ldots, x_m)$ for any finite phase P with $0 < P \leq L$.

Case: Inductive Case

Suppose $\forall s < r$, if $sL < |P''| \leq (s+1)L$, then $P'' \models \psi(x_1, \ldots, x_m)$. We must show that $P \models \psi(x_1, \ldots, x_m)$, where $rL < |P| \leq (r+1)L$. Then (8.5) for $n = r$ reduces to

$$\left[\begin{array}{ll} \text{If } (\forall k \in \mathbb{N}, k < r) & \left[\begin{array}{l} \text{for any } P', P^1, \ldots, P^k \\ \text{where } |P'| \leq L \text{ and } |P^i| = L \ (\forall i \in [1..k]) \\ P' \frown P^1 \frown \cdots \frown P^k \models \psi(x_1, \ldots, x_m) \end{array} \right] \\ \text{then} & \left[\begin{array}{l} \text{for any } P', P^1, \ldots, P^r \\ \text{where } |P'| \leq L \text{ and } |P^i| = L \ (\forall i \in [1..r]) \\ P' \frown P^1 \frown \cdots \frown P^r \models \psi(x_1, \ldots, x_m) \end{array} \right] \end{array} \right] \quad (8.6)$$

We will show the antecedent of (8.6). By our induction hypothesis, for all $s < r$, for all P'' such that $sL < |P''| \leq (s+1)L$, we have

$$P'' \models \psi(x_1, \ldots, x_m). \quad (8.7)$$

Now consider any $k < r$ and arbitrary adjacent phases $P', P^1, \ldots, P^k$ such that $|P'| \leq L$ and $|P^i| = L$ $(\forall i \in [1..k])$. Then $kL < |P' \frown P^1 \frown \ldots \frown P^k| \leq (k+1)L$. As $k < r$ by (8.7), $P' \frown P^1 \frown \ldots \frown P^k \models \psi(x_1, \ldots, x_m)$. Thus the antecedent of (8.6) holds. So by (8.6),

$$\begin{array}{l} \text{for any } P', P^1, \ldots, P^r, \text{where } |P'| \leq L \text{ and } |P^i| = L \ (\forall i \in [1..r]) \\ P' \frown P^1 \frown \cdots \frown P^r \models \psi(x_1, \ldots, x_m) \end{array} \quad (8.8)$$

Now P can be chopped up into $P = P_0'{}^\frown P_0^1{}^\frown \ldots {}^\frown P_0^r$, for some adjacent phases P_0' and P_0^i, where $|P_0'| \leq L$ and $|P_0^i| = L$ ($\forall i \in [1..r]$). Fix such phases. Then by (8.8),

$$P = P_0'{}^\frown P_0^1{}^\frown \ldots {}^\frown P_0^r \models \psi(x_1, \ldots, x_m)$$

as desired. ∎

8.7 Summary

Our approach differs from that of the duration calculus community ([32, 34, 39, 40, 41, 43, 42, 53, 61, 62, 63, 66, 87, 88, 137, 135, 154, 155, 156]). The duration calculus approach requires that both specification properties and possible implementation strategies be expressed as duration calculus formulas. Verification is the process of proving that the implementation implies the specification and is done using an axiom system for the duration calculus. In our approach, implementation strategies are expressed using hybrid automata and translated into concrete phase transition systems. It is our belief that automata offer a more natural formalism for describing controllers and other hybrid systems.

Chapter 9

Related Work

9.1 Specification

Over the years, many alternative styles of specification have emerged. In this section, we discuss those specification styles that are used in hybrid systems as well as those styles that have influenced research in hybrid systems.

9.1.1 Real-Time Logics

Real-time logics [2, 12, 15, 14, 24, 69, 72, 90, 112, 137, 141, 142] have been used with varying degrees of success when applied to hybrid systems. Research into real-time logics has contributed significantly into understanding the complexity issues of hybrid systems [11, 13, 15, 14, 31, 152]. This contribution arises from the fact that real-time clocks are a special type of continuous variable whose slope is either 0 or 1.

In addition, real-time logics have spawned several new logics for specifying duration-like properties. For example, several researches have used *integrator computation tree logic* (ICTL) [58, 74, 77, 83], which is a branching-time logic that extends timed computation tree logic (TCTL) [8, 45] by adding integrator variables. This logic is used by HyTech [75, 74, 77], a software tool for the specification and verification of hybrid systems using symbolic model-checking.

Other examples of logics for hybrid systems based on real-time logics include the following:

- *duration temporal logic* (DTL) of Bouajjani, Echahed, and Sifakis [32], which extends TCTL by adding duration constraints. This logic is useful for expressing interval-based properties.

- *duration interval logic* (DIL) of [34, 98], which corresponds to the propositional fragment of CoD extended with duration variables. These variables are introduced via a reset quantification that associates with each variable x a state formula whose duration is measured by x. The authors show how fragments of CoD reduce to DIL. These fragments further reduce to a class of linear hybrid automata that is decidable. Like DTL, this logic is useful for expressing interval-based properties.

- *hybrid automata temporal logic* (HATL) of [33, 98], which extends temporal logic with hybrid automata on bounded trajectories. Unlike the previous two logics (i.e., DTL and DIL), this logic is used to express point-based properties. HATL can be translated to a subclass of hybrid automata that has a decidable emptiness problem.

9.1.2 Duration Calculus

Chaochen et al.'s, *Calculus of Durations* (CoD) [41], designed for the specification and verification of real-time systems, is the logic most similar to ours. Like HTL, CoD has a chop operator and is interpreted over real-time dense intervals. Chaochen, Ravn, and Hansen [43] extend the calculus to hybrid systems by allowing one to specify values at the left and right endpoints of a phase, a feature of HTL that is not present in the original duration calculus ([41]). Moreover, the original duration operator that measures the duration of time a proposition p is true over an interval, denoted $\int$, is now a derived operator. Its encoding in the extended duration calculus and HTL are similar. The encoding can be found in [43].

Verifying that a hybrid system satisfies a duration calculus property is done by translating the system into a set of duration calculus formulas and using a deductive system for the calculus. Much research has gone into the duration calculus in the form of verification examples and applications [39, 42, 53, 63, 66, 137, 135, 154, 155, 156] and decidability and theory [34, 40, 61, 88].

9.1.3 Temporal Logic of Actions

Lamport is a strong advocate of using "old-fashioned" formalisms for specifying hybrid systems [3]. He advocates using TLA [100], the *Temporal Logic of Actions*, for specifying hybrid systems. TLA is a temporal logic with a restricted next-time operator and significantly, no built-in primitives for specifying real-time or hybrid properties. Instead any operators needed for specifying real-time properties are defined using TLA and ordinary mathematics. For example, TLA+ uses TLA and the standard integral operator to define durations [99].

Verification in TLA+ is done by writing both the property and the system in TLA+, and then using a deductive proof system for determining whether the system implies the property. Several verification examples have been done using TLA, including a steamboiler hybrid system by Lesske and Merz [106]. Lamport has also extended the work to give a diagrammatic version of TLA, presented in [101].

9.1.4 Interval Logics

Interval temporal logics [60, 138, 1, 55, 136] have been used for the specification of concurrent systems because they provide context-restricting temporal modalities which enable the easy specification of properties required to hold in these restricted contexts. The advantage of an interval-based logic is that it provides natural expressions for developments and changes across an arbitrary interval. To express the same properties in a point-based logic, it is always necessary to introduce additional auxiliary "freeze" variables which record the state at the beginning of the interval of interest. Since continuous development over an interval is of principal interest, it is important to be able to express such properties in the most natural way. In Table 9.1, we compare the salient features of each of the above logics as well as prop-HTL and prop-CoD, giving the complexity of the validity question for each logic.

Logic	**L/NL**	**C/I**	**$\bigcirc$?**	**Stut?**	**Complexity**	**Misc.**
ITL [60]	B[1]	C	Y	N	NL: Undec. L: Non-elem.	Built on top of linear-time temporal logic
PTL(U, X, C) [138]	L	C	Y	N	Non-elem.	Built on top of linear-time temporal logic
ISL [55]	L	I	N	N	decidable	Equivalent to PTL(Until)
FIL [136]	L	I	N	Y	EXPSPACE complete	Equivalent to PTL(Until)
prop-CoD [41]	NL	C	N	Y	Non-elem.	
prop-HTL [91]	L	C	N	Y	Non-elem.	

Abbreviations:

L: Local	NL: Non-local
B: Both local and non-local fragments are defined	
C: Chop operator present	I: Interval operator present
$\bigcirc$?: Next operator present?	Stut?: Stuttering insensitive?
N: No	Y: Yes

Abbreviation	**Logic**
ITL	Interval Temporal Logic
PTL(U, X, C)	Choppy Logic
ISL	Interval Specification Logic
FIL	Future Interval Logic
prop-CoD	propositional fragment of Calculus of Durations
prop-HTL	propositional fragment of Hybrid Temporal Logic

Table 9.1: Comparison of Interval Temporal Logics

Before describing the logics, we introduce some notation. A logic is local if each propositional variable p is true of an interval $s_0 \dots s_n$ iff p is true of the first state s_0. The chop operator of all the logics is fairly similarly. The semantics of the chop operator in [60, 138] is $\sigma_1 \dots \sigma_n \models \varphi_1; \varphi_2$ iff $\exists k \in [1..n]$, such that $\sigma_1 \dots \sigma_k \models \varphi_1$ and $\sigma_k \dots \sigma_n \models \varphi_2$. The semantics of the chop operator in prop-COD is similar to the semantics of the chop operator in this prop-HTL. Logics with the interval operator satisfy an interval formula $I\varphi$, where I is an expression representing a sub-interval and φ is an interval formula (i.e., may have occurrences of the interval operator), iff either the sub-interval I does not exist or I exists and φ holds in the subinterval expressed by I.

One of the earliest interval temporal logics is the *Interval Temporal Logic* (ITL) of [60, 122]. Designed for hardware verification, where discretization is both natural and possible, the logic uses a discrete semantics involving finite intervals, each consisting of a finite number of states. Intervals are composed using the chop operator, denoted ;. Each occurrence of a propositional literal represents an occurrence of a distinct event, and thus the logic is not stuttering insensitive.

Rosner and Pnueli's *Choppy Logic* [138] is a propositional linear-time temporal logic with a chop operator and has a non-elementary decision procedure based on a tableau construction.

Aaby and Narayana's *Propositional Temporal Interval Logic* (PTIL) [1, 124] is a propositional temporal logic with a chop operator. However, this logic is different than most interval logics that have a chop operator. In particular, box and diamond have the same semantics as they have under linear-time temporal logic, a property not shared by most interval logics. As such, we do not include it in Table 9.1. Because of the restricted semantics of their chop operator, their logic is expressively equivalent to PTL(Until). The main application of this logic is the specification and synthesis of hardware.

Goswami, Bell, and Joseph's *Interval Specification Logic* (ISL) [55] is a logic for the specification of interval-based properties of real-time systems. Because the logic does not have a chop operator, it is decidable. Moreover, like PTIL, it is expressively equivalent to PTL(Until). Another logic expressively equivalent to PTL(Until) is Ramakrishna et al.'s *Future Interval Logic* (FIL) [136]. Like ISL, it does not have a chop operator.

9.1.5 Hybrid CC

Hybrid CC of Gupta, Jagadeesan, Saraswat, and Bobrow [59, 58] is a constraint-based specification language for modeling hybrid systems. Behaviors are specified via constraints and either can be translated to a hybrid automaton description, which can then be used to verify properties, or can be interpreted to see the system running (i.e., a simulation of the system). Hybrid CC is expressive enough to encode ICTL properties. Thus, both properties and systems can be written in one framework.

[1]Both local and non-local versions of ITL are defined. However, future work, e.g., [123], uses the local version of the logic.

9.2 Verification

In this section we discuss the two primary approaches to hybrid system verification used by computer scientists: deductive approaches and algorithmic approaches. Orthogonal to computer science approaches to verification is the control theory approach to verification [29, 56, 82, 92, 105, 109], where the central problem is more one of synthesis (e.g., extracting a controller that meets some specification) than of verification.

9.2.1 Deductive Approaches

Deductive approaches to hybrid system verification can be categorized as either rule-based approaches or diagrammatic approaches. Deductive approaches have some advantages and some disadvantages over algorithmic approaches. In particular, deductive approaches work well with both finite and infinite state systems, yet their strong reliance on user-supplied intermediary assertions makes them difficult to use when verifying large systems.

Rule-Based Approaches

Many rule-based approaches have been studied. One such approach is the TLA approach discussed earlier. Another approach uses timed I/O automata to verify linear hybrid automata, where both the system and its properties are written using I/O automata [30, 111, 110, 68]. Other approaches represent the system as a transition system and the properties to verify in some temporal logic [70, 91, 93, 114, 118, 117, 120, 143, 121]. Further approaches are based on proof outlines [54, 141] or higher-order logics, such as PVS [142, 145, 149, 128, 129].

Diagram-Based Approaches

Diagram-based approaches have been applied to both reactive systems [38, 37, 36, 49, 50, 64, 119] and hybrid systems [47, 94, 101]. In addition, diagram-based approaches have been used to combine deductive and algorithmic approaches, as in [144].

9.2.2 Model Checking

Algorithmic approaches come in many flavors, for example, symbolic model checking, explicit model checking, and on-the-fly algorithms. Primary work in this area relevant to hybrid systems is the work by Alur et al. [10, 9], which use a fixed-point computation to prove properties of ICTL formulas over linear hybrid automata. The algorithms have been used in the HyTech system [75, 74, 73, 77, 76, 81, 80, 84, 146]. Numerous examples have been tried.

Other algorithmic approaches include the integration graphs of Kesten, Pnueli, Sifakis, and Yovine [95], a decidable class of constant slope hybrid systems which restrict how continuous variables are compared, and Chaochen's restricted duration calculus approach [39], where linear duration calculus invariants are algorithmically verified using linear programming techniques.

Numerous tools based on algorithmic techniques have been developed, including KRONOS [46, 45, 125, 127, 148], based on the fixed-point computation of Henzinger et al [72] and UPPAAL [24, 23, 103], based on a constraint-solving model-checker.

Chapter 10

Conclusions

We now summarize what we have accomplished and give insights into future research. We started our journey by characterizing the behavior of hybrid systems along three distinct semantics: a sampling semantics, a super-dense semantics, and a continuous interval semantics. We then introduced temporal logics for each of these semantics: two based on linear-time temporal logic and one based on an interval temporal logic. For the latter logic, we provided a decision procedure for determining validity questions; such a procedure was not presented for the sampling semantics since validity and complexity issues have been well-studied. We next proceeded to discuss the types of properties that we would like to prove for these systems, partitioning the properties into point-based properties and interval-based properties. Having thus set the groundwork for understanding the behavior of hybrid systems, we embarked on our quest: to verify both point-based and interval-based properties of hybrid systems.

We started by presenting a diagrammatic verification methodology for proving point-based properties under the sampling semantics. We next extended the approach to prove point-based properties under the super-dense semantics. Finally, we presented a rule-based methodology for proving both interval and point-based safety properties under the continuous semantics.

10.1 Future Directions

Clearly much work in hybrid system verification remains. Here we highlight work that builds on this thesis and which may have an impact on the field.

Automation: In order for real systems to be verified, tools must be developed to aid the engineer in proving properties. We believe that tools such as HyTech [75, 74, 77] and STeP [26, 25] are excellent first steps toward automation, but of course, we must do more. Our own work, in particular the work on diagrams, could be improved by developing algorithms for the automatic construction of intermediary invariants along the lines of Bjorner, Browne, and Manna [28].

Theorem Proving: While the computational induction axiom and stuttering automata are useful for proving some HTL properties, to fully utilize HTL as a specification language, we should develop a proof system for proving new HTL properties from existing HTL properties.

Extensions: The work on verification rules for HTL concentrated on safety properties. It would be interesting to examine the role of liveness in interval-based properties and to develop proof rules for proving liveness properties. In addition, it would be worthwhile to develop diagrammatic methods for HTL, since diagrams tend to be more understandable and also allow for the gradual construction of proofs.

Completeness: Completeness issues have largely been ignored in the continuous case (for both the super-dense semantics and the continuous interval semantics). Investigating completeness issues is another line of research.

It should be obvious that much research remains to make verification practical and to make its use widespread. Disasters such as the European Space Agency's Ariane 5[1] only emphasize the importance of hybrid system verification and stress the need for industry, government, and academia to work together to provide tools and methods for insuring system correctness and system safety.

[1]Ariane 5 [107] was a launcher that 40 seconds after take-off veered off its flight path and exploded. The error was caused in part by a software bug that resulted in faulty data being sent to the controller. The cost was in the billions of dollars.

Bibliography

[1] A.A. Aaby and K.T. Narayana. Propositional temporal interval logic is PSPACE complete. In *Proc. 9th Int. Conf. on Automated Deduction*, volume 310 of *Lect. Notes in Comp. Sci.*, pages 218–237. Springer-Verlag, 1988.

[2] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.

[3] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In de Bakker et al. [51], pages 1–27.

[4] J.R. Abrial, E. Börger, and H. Langmaack, editors. *Formal Methods for Industrial Applications*, volume 1165 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1996.

[5] Aho, Hopcroft, and Ullman. *The Design and Analysis of Algorithms.* Addison Wesley, 1974.

[6] B. Alpern, A.J. Demers, and F.B. Schneider. Safety without stuttering. *Information Processing Letters*, 23(4):177–180, 1986.

[7] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for real-time systems. In *Proc. 5th IEEE Symp. Logic in Comp. Sci.*, pages 414–425. IEEE Computer Society Press, 1990.

[8] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993. A preliminary version appeared in [7].

[9] 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*, 138(1):3–34, 1995.

[10] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Grossman et al. [57], pages 209–229.

[11] R. Alur, C. Courcoubetis, and T.A. Henzinger. The observational power of clocks. In B. Jonsson and J. Parrow, editors, *Proc. 5th Intl. Conf. on Concurrency Theory*, volume 836 of *Lect. Notes in Comp. Sci.*, pages 162–177. Springer-Verlag, 1994.

[12] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[13] R. Alur and T.A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proc. 5th IEEE Symp. Logic in Comp. Sci.*, pages 390–401. IEEE Computer Society Press, 1990.

[14] R. Alur and T.A. Henzinger. Logics and models of real time: A survey. In de Bakker et al. [51], pages 74–106.

[15] R. Alur and T.A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104:35–77, 1993. A preliminary version appeared in [13].

[16] R. Alur and T.A. Henzinger, editors. *Proc. 8^{th} Intl. Conf. on Computer Aided Verification*, volume 1102 of *Lect. Notes in Comp. Sci.* Springer-Verlag, July/August 1996.

[17] R. Alur, T.A. Henzinger, and E.D. Sontag, editors. *Hybrid Systems III: Verification and Control*, volume 1066 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1995.

[18] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *Proc. 25th ACM Symp. Theory of Comp.*, pages 592–601, 1993.

[19] R. Alur and R.P. Kurshan. Timing analysis in COSPAN. In Alur et al. [17], pages 220–231.

[20] T. Anderson, R. de Lemos, J.S. Fitzgerald, and A. Saeed. On formal support for industrial-scale requirements analysis. In Grossman et al. [57], pages 426–451.

[21] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors. *Hybrid Systems II*, volume 999 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1994.

[22] J. Armstrong and L. Barroca. Specification and verification of reactive system behaviour: The railroad crossing example. *Real-Time Systems*, 10:143–178, 1996.

[23] J. Bengtsson, W.O.D. Griffioen, K.J. Kristoffersen, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using UPPAAL. In Alur and Henzinger [16], pages 244–256.

[24] J. Bengtsson, K.G. Larson, F. Larsson, P. Pettersson, and W. Yi. UPPAAL – a Tool Suite for Automatic Verification of Real-Time Systems. In Alur et al. [17], pages 232–243.

[25] N. Bjørner, A. Browne, E. Chang, M. Colón, A. Kapur, Z. Manna, H.B. Sipma, and T.E. Uribe. STeP: Deductive-algorithmic verification of reactive and real-time systems. In Alur and Henzinger [16], pages 415–418.

[26] N. Bjørner, A. Browne, E. Chang, A. Kapur, Z. Manna, H.B. Sipma, and T.E. Uribe. STeP: The Stanford temporal prover (educational release) user's manual. Technical Report STAN-CS-TR-95-1562, Stanford University, 1995.

[27] N. Bjørner, Z. Manna, H.B. Sipma, and T.E. Uribe. Deductive verification of real-time systems using STeP. In *4th Intl. AMAST Workshop on Real-Time Systems*, volume 1231 of *Lect. Notes in Comp. Sci.*, pages 22–43. Springer-Verlag, May 1997.

[28] N.S. Bjørner, A. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science*, 173(1):49–87, February 1997.

[29] M. Blanke, S.B. Nielsen, and R.B. Jørgensen. Fault accommodation in feedback control systems. In Grossman et al. [57], pages 393–426.

[30] D. Bosscher, I. Polak, and F. Vaandrager. Verification of an audio control protocol. In Langmaack et al. [102], pages 170–193.

[31] A. Bouajjani, R. Echahed, and R. Robbana. Verifying invariance properties of timed systems with duration variables. In Langmaack et al. [102], pages 193–210.

[32] A. Bouajjani, R. Echahed, and J. Sifakis. On model checking for real-time properties with durations. In *Proc. 8th IEEE Symp. Logic in Comp. Sci.*, pages 147–159. IEEE Computer Society Press, 1993.

[33] A. Bouajjani and Y. Lakhnech. Logic vs. automata: The hybid case (extended abstract). In Alur et al. [17], pages 531–542.

[34] A. Bouajjani, Y. Lakhnech, and R. Robbana. From duration calculus to linear hybrid automata. In Wolper [153], pages 196–210.

[35] A. Bouajjani and R. Robbana. Verifying ω-regular properties for a subclass of linear hybrid systems. In Wolper [153], pages 437–450.

[36] A. Browne, L. de Alfaro, Z. Manna, H.B. Sipma, and T.E. Uribe. Diagram-based formalisms for the verification of reactive systems. In *CADE-14 Workshop on Visual Reasoning*, 1996.

[37] A. Browne, Z. Manna, and H.B. Sipma. Hierarchical verification using verification diagrams. In 2^{nd} *Asian Computing Science Conf.*, volume 1179 of *Lect. Notes in Comp. Sci.*, pages 276–286. Springer-Verlag, December 1996.

[38] I.A. Browne, Z. Manna, and H.B. Sipma. Generalized verification diagrams. In *Found. of Software Technology and Theoretical Comp. Sci.*, volume 1026 of *Lect. Notes in Comp. Sci.*, pages 484–498. Springer-Verlag, 1995.

[39] Z. Chaochen. Linear duration invariants. In Langmaack et al. [102], pages 86–109.

[40] Z. Chaochen, M.R. Hansen, and P. Sestoft. Decidability and undecidability results for duration calculus. In *Proc. of 10th Annual Symp. on Theor. Asp. of Comp. Sci.*, volume 665 of *Lect. Notes in Comp. Sci.*, pages 58–68. Springer-Verlag, 1993.

[41] Z. Chaochen, C.A.R Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

[42] Z. Chaochen, W. Ji, and A.P. Ravn. A formal description of hybrid systems. In Alur et al. [17], pages 511–530.

[43] Z. Chaochen, A.P. Ravn, and M.R. Hansen. An extended duration calculus for hybrid real-time systems. In Grossman et al. [57], pages 36–59.

[44] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *J. Comp. Sys. Sci.*, 8:117–141, 1974.

[45] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In Alur et al. [17], pages 208–219.

[46] C. Daws and S. Yovine. Verification of multirate timed automata with KRONOS: Two examples. Technical Report Spectre-95-06, VERIMAG, April 1995.

[47] L. de Alfaro, A. Kapur, and Z. Manna. Hybrid diagrams: A deductive-algorithmic approach to hybrid system verification. In *Proc. of 14th Annual Symp. on Theor. Asp. of Comp. Sci.*, volume 1200 of *Lect. Notes in Comp. Sci.*, pages 153–164. Springer-Verlag, February 1997.

[48] L. de Alfaro and Z. Manna. Verification in continuous time by discrete reasoning. In V.S. Alagar, editor, *Algebraic Methodology and Software Technology*, volume 936 of *Lect. Notes in Comp. Sci.*, pages 292–306. Springer-Verlag, July 1995.

[49] L. de Alfaro and Z. Manna. Temporal verification by diagram transformations. In Alur and Henzinger [16], pages 288–299.

[50] L. de Alfaro, Z. Manna, H.B. Sipma, and T.E. Uribe. Visual verification of reactive systems. In *TACAS 97: Third Intl. Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1217 of *Lect. Notes in Comp. Sci.*, pages 334–350. Springer-Verlag, April 1997.

[51] J.W. de Bakker, K. Huizing, W.-P de Roever, and G. Rozenberg, editors. *Proc. of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1991.

[52] R. de Lemos, A. Saeed, and T. Anderson. Analysis of timeliness requirements in safety-critical systems. In Vytopil [150], pages 171–192.

[53] M. Engel, M. Kubica, J. Madey, D.L. Parnas, A.P. Ravn, and A.J. van Schouwen. A formal approach to computer systems requirements documentation. In Grossman et al. [57], pages 452–474.

[54] L. Fix and F.B. Schneider. Hybrid verification by exploiting the environment. In Langmaack et al. [102], pages 1–18.

[55] A. Goswami, M. Bell, and M. Joseph. ISL: An interval logic for the specification of real-time. In Vytopil [150], pages 1–20.

[56] M. Greenstreet. Verifying safety properties of differential equations. In Alur and Henzinger [16], pages 277–287.

[57] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems*, volume 736 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1993.

[58] V. Gupta, R. Jagadeesan, and V. Saraswat. Hybrid cc, hybrid automata and program verification. In Alur et al. [17], pages 52–63.

[59] V. Gupta, R. Jagadeesan, V. Saraswat, and D. Bobrow. Programming in hybrid constraint languages. In Antsaklis et al. [21], pages 226–251.

[60] J. Halpern, Z. Manna, and B. Moszkowski. A hardware semantics based on temporal intervals. In *Proc. 10th Int. Colloq. Aut. Lang. Prog.*, volume 154 of *Lect. Notes in Comp. Sci.*, pages 278–291. Springer-Verlag, 1983.

[61] M.R. Hansen and Z. Chaochen. Semantics and completeness of duration calculus. In de Bakker et al. [51], pages 209–225.

[62] M.R. Hansen, Z. Chaochen, and J. Staunstrup. A real-time duration semantics for circuits, May 1992.

[63] M.R. Hansen, P.K. Pandya, and Z. Chaochen. Finite divergence. Technical Report 15, UNU/IIST, 1993.

[64] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[65] D. Harel, D. Kozen, and R. Parikh. Process logic: Expressiveness, decidability, completeness. *J. Comp. Sys. Sci.*, 25:144–170, 1982.

[66] J. He, C.A.R. Hoare, M. Fränzle, M. Müller-Olm, E.-R. Olderog, M. Schenke, M.R. Hansen, A.P. Ravn, and H. Rischel. Provably correct systems. In Langmaack et al. [102], pages 288–335.

[67] C. Heitmeyer, R. Jeffords, and B. Labaw. A benchmark for comparing different approaches for specifying and verifying real-time systems. In *Proc. Tenth Intl. Workshop on Real-Time Operating Systems and Software*, 1993.

[68] C. Heitmeyer and N. Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. In *Proc. of the 15th Annual Real-time Systems Symposium*, pages 120–131. IEEE Computer Society Press, December 1994.

[69] T. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In de Bakker et al. [51], pages 226–251.

[70] T. Henzinger, Z. Manna, and A. Pnueli. Towards refining temporal specifications into hybrid systems. In Grossman et al. [57], pages 60–76.

[71] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proc. 7th IEEE Symp. Logic in Comp. Sci.*, pages 394–406. IEEE Computer Society Press, June 1992. A full version of this paper (including all proofs) is available as a technical report from Cornell University and from IMAG in Grenoble.

[72] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994. A preliminary version of this paper appeared in [71].

[73] T.A. Henzinger and P.-H. Ho. A note on abstract-interpretation strategies for hybrid automata. In P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors, *Hybrid Systems II*, volume 999 of *Lect. Notes in Comp. Sci.*, pages 252–264. Springer-Verlag, 1994.

[74] T.A. Henzinger and P.-H. Ho. HyTech: The Cornell hybrid technology tool. In Antsaklis et al. [21], pages 265–293.

[75] T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In Wolper [153], pages 225–238.

[76] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: The next generation. In *Proc. of the 16th Annual Real-time Systems Symposium*, pages 56–65. IEEE Computer Society Press, 1995.

[77] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HYTECH. In E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, editors, *TACAS 95: First Intl. Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 of *Lect. Notes in Comp. Sci.*, pages 41–71. Springer-Verlag, 1995.

[78] T.A. Henzinger and P. Kopke. Verification methods for the divergent runs of clock systems. In Langmaack et al. [102], pages 351–372.

[79] T.A. Henzinger, P.W. Kopke, and H. Wong-Toi. The expressive power of clocks. In Z. Fülöp and F. Gécseg, editors, *Proc. 22nd Int. Colloq. Aut. Lang. Prog.*, Lect. Notes in Comp. Sci., pages 417–428. Springer-Verlag, 1995.

[80] T.A. Henzinger and H. Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. In Alur et al. [17], pages 377–388.

[81] T.A. Henzinger and H. Wong-Toi. Using HYTECH to synthesize control parameters for a steam boiler. In Abrial et al. [4].

[82] M. Heymann, F. Lin, and G. Meyer. Control synthesis for a class of hybrid systems subject to configuration-based safety constraints. In Maler [113], pages 376–390.

[83] P.-H. Ho. *Automatic Analysis of Hybrid Systems.* PhD thesis, Cornell University, 1995.

[84] P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In Wolper [153], pages 381–394.

[85] J. Hooman. A compositional approach to the design of hybrid systems. In Grossman et al. [57], pages 121–148.

[86] Hopcroft and Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979.

[87] Y. Huiqun, P.K. Pandya, and S. Yongqiang. A calculus for hybrid sampled data systems. Technical Report 21, UNU/IIST, 1994.

[88] D.V. Hung and P.H. Giang. Sampling semantics of duration calculus. In B. Jonsson and J. Parrow, editors, *FTRTFT'96*, volume 1135 of *Lect. Notes in Comp. Sci.*, pages 188–207. Springer-Verlag, 1996.

[89] M. Jaffe, N. Leveson, M. Heimdahl, and B. Melhart. Software requirements analysis for real-time process-control systems. *IEEE Trans. Software Engin.*, 17(3):241–258, 1991.

[90] F. Jahanian and A.K. Mok. Safety analysis of timing properties in real-time systems. *IEEE Trans. Software Engin.*, 12(9):890–904, 1986.

[91] A. Kapur, T.A. Henzinger, Z. Manna, and A. Pnueli. Proving safety properties of hybrid systems. In Langmaack et al. [102], pages 431–454.

[92] D. Kapur and R.K. Shyamasundar. Synthesizing controller for hybrid systems. In Maler [113], pages 361–375.

[93] Y. Kesten, Z. Manna, and A. Pnueli. Verifying clocked transition systems. In Alur et al. [17], pages 13–40.

[94] Y. Kesten and A. Pnueli. Timed and hybrid statecharts and their textual representation. In Vytopil [150], pages 591–619.

[95] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: A class of decidable hybrid systems. In Grossman et al. [57], pages 179–208.

[96] Kurki-Suonio. Hybrid models with fairness and distributed clocks. In Grossman et al. [57], pages 103–120.

[97] L. Lai and P. Watson. A case study in timed CSP: The railroad crossing problem. In Maler [113], pages 69–74.

[98] Y. Lakhnech. *Specification and Verification of Hybrid and Real-Time Systems*. PhD thesis, Christian-Albrechts University, Kiel, 1995.

[99] L. Lamport. Hybrid systems in TLA+. In Grossman et al. [57], pages 77–102.

[100] L. Lamport. The temporal logic of actions. *ACM Trans. Prog. Lang. Sys.*, 16(3):872–923, May 1994.

[101] L. Lamport. TLA in pictures. Technical Report 127, Digital Equipment Corporation, Systems Research Center, September 1994.

[102] H. Langmaack, W.-P. de Roever, and J. Vytopil, editors. *FTRTFT'94*, volume 863 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1994.

[103] K.G. Larsen, P. Pettersson, and W. Yi. Diagnostic model-checking for real-time systems. In Alur et al. [17], pages 575–586.

[104] D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In *Proc. 8th Int. Colloq. Aut. Lang. Prog.*, volume 115 of *Lect. Notes in Comp. Sci.*, pages 264–277. Springer-Verlag, 1981.

[105] M. Lemmon and P.J. Antsaklis. Hybrid systems and intelligent control. In *Proc. of the 1993 Intl. Symp. on Intelligent Control*, pages 174–179. IEEE Control Systems Society, 1993.

[106] F. Lesske and S. Merz. Steam boiler control specification problem: A TLA solution. In Abrial et al. [4], pages 339–358.

[107] J.L. Lions, L. Lbeck, J.L. Fauquembergue, G. Kahn, W. Kubbat, S. Levedag, L.M.A. Spazio, D.M. Thmoson, and C. O'Halloran. Ariane 5: Flight 501 failure—report by the inquiry board, July 1996.

[108] J. Lygeros, D.N. Godbole, and S. Sastry. A game-theoretic approach to hybrid system design. In Alur et al. [17], pages 1–12.

[109] J. Lygeros, C. Tomlin, and S. Sastry. Multiobjective hybrid controller synthesis. In Maler [113], pages 109–123.

[110] N. Lynch. Simulation techniques for proving properties of real-time systems. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Proc. of the REX Workshop "A Decade of Concurrency"*, volume 803 of *Lect. Notes in Comp. Sci.*, pages 375–424. Springer-Verlag, 1993.

[111] N. Lynch, R. Segala, F. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In Alur et al. [17], pages 496–510.

[112] N.A. Lynch and H. Attiya. Using mappings to prove timing properties. *Distributed Computing*, 6:121–139, 1992.

[113] O. Maler, editor. *Hybrid and Real-Time Systems*, volume 1201 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1997.

[114] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In de Bakker et al. [51], pages 447–484.

[115] Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83(1):97–130, 1991.

[116] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1991.

[117] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30:609–678, 1993.

[118] Z. Manna and A. Pnueli. Verifying hybrid systems. In Grossman et al. [57], pages 4–35.

[119] Z. Manna and A. Pnueli. Temporal verification diagrams. In *TACS 94*, volume 789 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1994.

[120] Z. Manna and A. Pnueli. Clocked transition systems. In *Logic and Software Workshop*, August 1995. Beijing, China. Available as Comp. Sci. Tech. Report, Stanford University.

[121] Z. Manna and H.B. Sipma. A deductive approach towards controller synthesis. In *IEEE Intl. Symposium on Intelligent Control*, pages 35–41. IEEE Computer Society Press, August 1995.

[122] B. Moszkowski. A temporal logic for multi-level reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.

[123] B. Moszkowski. Some very compositional temporal properties. In *IFIP Working Conf. on Prog. Concepts, Methods, and Calculi (PROCOMET 94)*, pages 303–322, 1994.

[124] K.T. Narayana and A.A. Aaby. Specification of real-time systems in real-time temporal interval logic. In *Proc. of the 9th Annual Real-time Systems Symposium*, pages 86–95. IEEE Computer Society Press, 1988.

[125] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In Grossman et al. [57], pages 149–178.

[126] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. In de Bakker et al. [51], pages 549–572.

[127] A. Olivero, J. Sifakis, and S. Yovine. Using abstractions for the verification of linear hybrid systems. In D. Dill, editor, *Proc. 6^{th} Intl. Conf. on Computer Aided Verification*, volume 818 of *Lect. Notes in Comp. Sci.*, pages 81–94. Springer-Verlag, June 1994.

[128] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In Alur and Henzinger [16], pages 411–414.

[129] S. Owre, N. Shankar, and J.M. Rushby. *The PVS Proof Checker: A Reference Manual (Beta Release)*. SRI International, Menlo Park, CA 94025, USA, March 1993.

[130] D. Peled, T. Wilke, and P. Wolper. An algorithmic approach for checking closure properties of ω-regular languages. In *Proc. 7th Intl. Conf. on Concurrency Theory*, volume 1119 of *Lect. Notes in Comp. Sci.*, pages 596–610. Springer-Verlag, 1996.

[131] A. Pnueli. Personal Communication.

[132] A. Pnueli. How vital is liveness? Verifying timing properties of reactive and hybrid systems. In W.R. Cleaveland, editor, *Proc. 3rd Intl. Conf. on Concurrency Theory*, volume 630 of *Lect. Notes in Comp. Sci.*, pages 162–175. Springer-Verlag, 1992.

[133] A. Pnueli. Development of hybrid systems. In Langmaack et al. [102], pages 77–85. Extended abstract.

[134] A. Puri and P. Varaiya. Verification of hybrid systems using abstractions. In Antsaklis et al. [21], pages 359–369.

[135] X. Qiwen and H. Weidong. Hierarchical design of a chemical concentration control system. In Alur et al. [17], pages 270–281.

[136] Y.S. Ramakrishna, P.M. Melliar-Smith, L.E. Moser, L.K. Dillon, and G. Kutty. Interval logics and their decision procedures, part I: An interval temporal logic. *Theoretical Computer Science*, 166(1–2):1–48, October 1996.

[137] A.P. Ravn, H. Rischel, and K.M. Hansen. Specifying and verifying requirements of real-time systems. *IEEE Trans. Software Engin.*, 19(1):41–55, January 1993.

[138] R. Rosner and A. Pnueli. A choppy logic. In *Proc. First IEEE Symp. Logic in Comp. Sci.*, pages 306–313. IEEE Computer Society Press, 1986.

[139] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. Found. of Comp. Sci.*, pages 319–327, 1988.

[140] F.B. Schneider. Real-time, reliable systems project. In *Proc. of the ONR Kickoff Workshop for the Foundations of Real-time Computing Research Initiative*, pages 28–32, 1988.

[141] F.B. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. In de Bakker et al. [51], pages 618–639.

[142] N. Shankar. Verification of real-time systems using PVS. In C. Courcoubetis, editor, *Proc.* 5^{th} *Intl. Conf. on Computer Aided Verification*, volume 697 of *Lect. Notes in Comp. Sci.*, pages 280–291. Springer-Verlag, June 1993.

[143] H.B. Sipma and Z. Manna. Specification and verification of controlled systems. In Langmaack et al. [102], pages 641–659.

[144] H.B. Sipma, T.E. Uribe, and Z. Manna. Deductive model checking. In Alur and Henzinger [16], pages 208–219.

[145] J.U. Skakkebæk and N. Shankar. Towards a duration calculus proof assistant in PVS. In Langmaack et al. [102], pages 660–679.

[146] T. Stauner, O. Müller, and M. Fuchs. Using HyTECH to verify an automotive control system. In Maler [113], pages 139–153.

[147] L.J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic.* PhD thesis, MIT, 1974.

[148] S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulations. In Alur and Henzinger [16], pages 232–243.

[149] J. Vitt and J. Hooman. Assertional specification and verification using PVS of the steam boiler control system. In Abrial et al. [4], pages 453–472.

[150] J. Vytopil, editor. *FTRTFT'92*, volume 571 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1992.

[151] C. Weise. Weak refinement for modal hybrid systems. In Maler [113], pages 316–330.

[152] T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In Langmaack et al. [102], pages 694–715.

[153] P. Wolper, editor. *Proc. 7th Intl. Conf. on Computer Aided Verification*, volume 939 of *Lect. Notes in Comp. Sci.* Springer-Verlag, July 1995.

[154] Y. Xinyao, W. Ji, Z. Chaochen, and P.K. Pandya. Formal design of hybrid systems. Technical Report 19, UNU/IIST, 1994.

[155] X. Yu, J. Wang, C. Zhou, and P.K. Pandya. Formal design of hybrid systems. In Langmaack et al. [102], pages 738–755.

[156] Z. Yuhua and Z. Chaochen. A formal proof of the deadline driven scheduler. Technical Report 16, UNU/IIST, 1994.

[157] Y. Zhang and A.K. Mackworth. Specification and verification of hybrid dynamic systems with timed ∀-automata. In Alur et al. [17], pages 587–603.