AUTOMATIC ANALYSIS OF HYBRID SYSTEMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Pei-Hsin Ho

August 1995

AUTOMATIC ANALYSIS OF HYBRID SYSTEMS

Pei-Hsin Ho, Ph.D.

Cornell University 1995

Hybrid systems are real-time systems that react to both discrete and continuous activities (such as analog signals, time, temperature, and speed). Typical examples of hybrid systems are embedded systems, timing-based communication protocols, and digital circuits at the transistor level. Due to the rapid development of microprocessor technology, hybrid systems directly control much of what we depend on in our daily lives. Consequently, the formal specification and verification of hybrid systems has become an active area of research.

This dissertation presents the first general framework for the formal specification and verification of hybrid systems, as well as the first hybrid-system analysis tool—HyTech. The framework consists of a graphical finite-state-machine-like language for modeling hybrid systems, a temporal logic for modeling the requirements of the hybrid systems, and a computer procedure that verifies modeled hybrid systems against modeled requirements. The tool HyTech is the implementation of the framework using C++ and Mathematica.

More specifically, our hybrid-system modeling language, *Hybrid Automata*, is an extension of timed automata [AD94] with discrete and general continuous variables whose dynamics are governed by differential equations. Our requirement modeling language, Ictl, is a branching-time temporal logic, and is an exten-

sion of TCTL [ACD93] with stop-watch variables. Our verification procedure is a symbolic model-checking procedure that verifies linear hybrid automata against ICTL formulas, and is an extension of the symbolic model-checking procedure for real-time systems in [HNSY94].

To make HYTECH more efficient and effective, we designed and implemented model-checking strategies and abstract operators that can expedite the verification process. To enable HYTECH to verify nonlinear hybrid automata, we also introduce two translations from nonlinear hybrid automata to linear hybrid automata that can be fed into HYTECH for automatic analysis. We have applied HYTECH to analyze more than 30 hybrid-system benchmarks. In this dissertation, we show the application of HYTECH to three nontrivial hybrid systems taken from the literature.

Biographical Sketch

Pei-Hsin Ho was born January 31, 1965 in a Catholic family in Taipei, Taiwan, Republic of China. He has wonderful parents and a cool younger brother who is now doing a Ph.D. in Physics at U.C. Berkeley.

The first computer that Pei-Hsin had was an Apple clone that his father purchased in 1981. Fascinated by the power of the wonderful machine (6502 CPU, 16K memory and a cassette recorder), he spent most of his time in the Pen-Chao senior high school programming BASIC.

In 1983, Pei-Hsin ventured into mathematics at Chung-Yuan Christian University, where he was awarded the degree of Bachelor of Science four years later. Inspired by the excellent faculty at Chung-Yuan Christian University, Pei-Hsin discovered his love for applied mathematics and theoretical computer science. Most importantly, he got to spend some time with a lovely classmate Chih-Ying Shih who later became his wife in 1991.

Pei-Hsin started to pursue graduate studies in the Department of Applied Mathematics at the National Chiao-Tung Univerity in 1987. He studied discrete mathematics and the design and analysis of graph algorithms from a great professor Gerard J. Chang, who taught Pei-Hsin how to conduct academic research in the area, and inspired Pei-Hsin to pursue a career in computer science.

To fulfill the military service requirement, Pei-Hsin attended Taiwan ROTC in

1989. He became an itinerary instructor officer in the Department of Defense of Taiwan. In about two years, the young officer traveled the whole island with his driver and Jeep to deliver lectures.

In the summer of 1991, Pei-Hsin moved to Ithaca, New York, to pursue his Ph.D. degree at Cornell University. Faced with the Q-exam, he spent the two coldest Christmas holidays in his life studying. Finally he managed to conquer the nightmare of the Q-exam, with the second highest combined score on his second shot. On January 30, 1994, Pei-Hsin and Chih-Ying got their little baby boy, Eric, in Tompkins County Hospital beside beautiful Cayuga Lake. Pei-Hsin started to work with Professor Thomas A. Henzinger on formal methods in the Spring of 1992. Pei-Hsin received the Ph.D. degree in Computer Science from Cornell University in August, 1995.

To my parents, Chao-Sin Ho and Cheng-Luan Chang,
to my wife, Chih-Ying Shih, and to my son, Eric Ho

Acknowledgements

Chapter 2 of this dissertation extends joint work with Rajeev Alur, Costas Courcoubetis and Tom Henzinger [ACHH93]. Chapter 3 is based on joint work with Rajeev and Tom [AHH93]. Chapter 4 is evolved from joint work with Tom [HH95b]. Chapter 5, 6 and 7 are respectively based on the collaborations [HH95c], [HH95a] and [HH95b] with Tom. I am indebted to each of my three coauthors for their inspiration and contribution. Their influence can be felt throughout the entire thesis.

I am most grateful to my advisor Tom Henzinger for his spiritual, technical and financial support that carried me through the doctoral program. His clear vision of the field led me to the most important and promising research problems. Tom is always so generous with his time and ideas to me. For two years we met to discuss the research work almost every day. Tom also made possible my fruitful trips to conferences and summer schools in America and Europe. To me, Tom is the best mentor as well as a great friend. Without his contribution, guidance and encouragement, this dissertation would simply not exist.

I also thank my other committee members, Anil Neorde and Dexter Kozen, who have both assisted me in many different ways throughout my time here. It has been a great experience and pleasure to learn from Anil Nerode, who established the most important work on the control of hybrid systems. The work of Chapter 6

was actually inspired by a conversation with Anil. Many thanks go to Dexter, who taught me how to deliver technical talks and gave me detailed comments on various drafts of my thesis.

Thanks to my colleagues Peter Kopke and Howard Wong-Toi in Tom's group for their friendship and inspiration and the ideas they shared at numerous discussions. I learned a lot from Howard about how to be a good researcher through our collaboration on three papers. Howard also almost single-handedly implemented the new version of HyTech, which greatly improved the usefulness of the tool.

The Computer Science Department at Cornell has been an exciting place to learn and work. I am indebted to many people who have taught and helped me. Special thanks go to David Chang, Ashvin Dsouza, David Karr and Marcel Rosu, who greatly enriched my life here. David Chang deserves my special gratitude for his friendship and valuable counsel, especially in my early years at Cornell.

I had the fortune to spend a summer at the Cadence Berkeley Labs. Thanks to my host Ken McMillan, who introduced me the Petri net model and patiently answered all my questions about Petri net. I am also grateful for the enthusiastic interest, help and advice that I received from colleagues in this field: Rajeev Alur, Edmund Clarke, David Dill, Limor Fix, Insup Lee, Nancy Lynch, John Rushby, Natarajan Shankar, Frits Vaandrager and Sergio Yovine. I also want to thank my master-thesis advisor Gerard J. Chang who brought me into the field of computer science six years ago.

Finally, I wish to express my deepest gratitude to my wife, Chih-Ying Shih, for all the love, help and support she has given me, and especially for all the happiness she has brought to me.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

"Where shall I begin, please your Majesty?" he asked.
"Begin at the beginning" the king said, gravely,
"and go on till you come to the end: then stop."
— Lewis Carroll[1]

1.1 Hybrid Systems

This dissertation begins with the answer of the following question. What is a hybrid system? Let us first look at the definitions of hybrid systems given by some participants of the Workshop on Hybrid Systems held at Lyngby, Denmark in October 1992:

> *Hybrid systems are interacting networks of digital and continuous devices. Typically they occur in digital control systems, in business, industry and the military.*
>
> — Grossman, Nerode, Ravn and Rischel [GNRR93]

[1]The quotes like this in this dissertation were found in [Rok93] or [BH95].

> *Hybrid systems are reactive systems that intermix discrete and continuous components. Typical examples are digital controllers that interact with continuously changing physical environments.*
>
> — Manna and Pnueli [MP93]

> *A hybrid system consists of discrete programs within an analog environment.* — Alur, Courcoubetis, Henzinger and Ho [ACHH93]

All of the three definitions share the common point that hybrid systems contain both digital and analog components. This dissertation aims at the hybrid systems whose digital components are real-time computing systems and analog components are controlled continuous activities in the environment. In other words, we use the notion of hybrid systems to model the combined behavior of embedded real-time computing systems and their physical environments.

Hybrid systems form an interdisciplinary topic that lies at the junction of Computer Science and Control Theory. Traditionally, computer scientists consider systems that consist of complex discrete programs and control theorists consider systems that consist of complex continuous activities. But due to the rapid development of microprocessor technology, more and more things that we depend on in our daily lives are controlled by systems that consist of both complex discrete programs and complex continuous activities. For instance, a modern automobile is usually equipped with a small local-area network of embedded systems including the fuel injection system, automatic cruise control system, anti-lock brake system and air bag system. Thus we think it is the right time to investigate the systems that consist of a nontrivial mixture of discrete programs and continuous activities; that is, hybrid systems.

The correctness of hybrid systems in safety-critical machinery, for example, a car or an airplane, is of obvious vital importance, and calls for software and mathematical tools that support the design and development of correct hybrid systems. Inspired by the previous work on the formal specification and verification of real-time and discrete systems, we present a general framework for the formal specification and verification of hybrid systems, as well as a software tool, HyTech, that can automatically analyze hybrid systems. The framework consists of (1) a graphical language for modeling hybrid systems, which is easy for engineers to understand and use, (2) a temporal logic for modeling the requirements of hybrid systems, which is more precise than, yet close to, the traditional English requirement specifications, and (3) a computer procedure for verifying modeled hybrid systems against modeled requirements, which is easier to use than the interactive machine-proving techniques. HyTech[2] is the implementation of this framework using C++ and Mathematica.

Based on the result of this dissertation, we plan to build an integrated environment that supports the rapid prototyping of hybrid systems by facilitating the specification, simulation, verification and analysis of the prototype. We expect that the tool can help designers develop more reliable hybrid systems in shorter turnaround time.

1.2 Overview

This section provides an overview of the dissertation.

[2]We thank HyTech users James Corbett at University of Hawaii, Sanjai Narain at Bellcore, Gino Labinaz at Queen's University in Canada, Rudolf E. Klug at CWI in the Netherlands, Simin Nadjm-Tehrani at Linkoping University in Sweden, Wang Yi and Paul Pettersson at Uppsala University in Sweden, Kim Larsen at Aalborg University in Denmark, and Satoshi Yamane at Shimane University in Japan for valuable feedbacks.

1.2.1 Hybrid Automata

We model the components of a hybrid system as a *hybrid automaton*. A hybrid automaton [ACHH93,ACH+95] is a generalized finite-state machine that is equipped with continuous variables. The discrete actions of a program are modeled by a change of the control locations. The continuous activities of the environment are modeled by real-valued variables whose values change continuously over time according to differential equations. This model for hybrid systems is inspired by the phase transition systems of [MMP92] and [NSY93], and can be viewed as a generalization of timed automata [AD94] with discrete and general continuous variables; a similar model has been proposed and studied independently in [NOSY93].

For verification purposes, we first consider *linear hybrid automata*. In each location of a linear hybrid automaton, the behavior of all variables are governed by linear constraints on the first derivatives. Common examples of linear constraints are constant differential equations, rectangular differential inclusions, and rate comparisons. We present decidability and undecidability results for the emptiness problem of linear hybrid automata.

1.2.2 Integrator Logic

Real-time requirements of systems can be specified in TCTL [ACD93], a branching-time temporal logic that extends CTL [CES86] with clock variables. We introduce *Integrator Computation Tree Logic*, ICTL, which strengthens TCTL in the style of [BES93] by admitting integrator variables. An *integrator* is a stop watch that can be stopped and restarted. While clocks suffice for the specification of time-bounded requirements—such as "A response is obtained if a bell has been pressed continuously for at least d seconds"—integrators are necessary to accumulate delays, and are useful for specifying duration requirements—such as "A response is

obtained if a bell has been pressed, possibly intermittently, for at least d seconds." We use ICTL to specify safety, liveness, time-bounded, and duration requirements of hybrid automata.

1.2.3 Symbolic Model Checking

Model checking is a powerful technique for the automatic verification of systems. A model-checking algorithm determines whether a mathematical model of a system meets a specification that is given as a temporal-logic formula. For discrete finite-state systems, model checking has a long history spanning more than a decade, and has been successful in validating communication protocols and hardware circuits [CES81,QS81,LP85,McM93]. In recent years, model-checking algorithms have also been developed for real-time systems that are described by discrete programs with real-valued clocks [AFH91,ACD93].

As the variables of a hybrid system range over the real numbers, the state space is infinite, and state sets—so-called *regions*—must be represented *symbolically* rather than enumeratively. A symbolic model-checking algorithm for verifying TCTL-requirements of real-time systems is presented in [HNSY94]. We extend this result and present a symbolic model-checking procedure for verifying ICTL-requirements of linear hybrid automata.

Given an ICTL-formula and a hybrid automaton, we compute the target region of states that satisfy the formula by successive approximation, as the limit of an infinite sequence of regions. The approximation sequence is generated by iterating boolean operators and weakest-precondition operators on regions. For linear hybrid automata, all regions of the approximation sequence are *linear* in the sense that they can be defined in the theory $(\mathbb{R}, \leq, +)$ of the reals with addition, whose formulas are boolean combinations of linear inequalities.

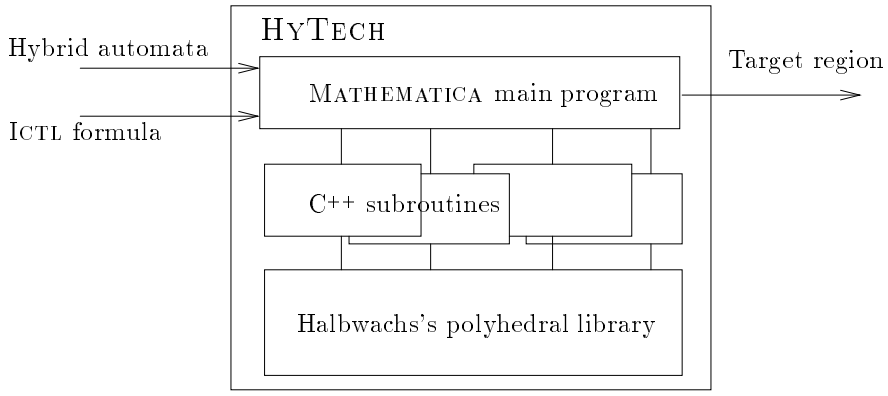

Figure 1.1: The architecture of HyTech

1.2.4 Automatic Analysis Tool : HyTech

The model-checking procedure has been implemented as part of the *Cornell HYbrid TECHnology Tool*, HyTech.[3] The first version of HyTech uses the symbolic computation system MATHEMATICA [Wol88] for manipulating and simplifying $(\mathbb{R}, \leq, +)$-formulas. In particular, the computation of weakest preconditions of linear regions requires quantifier elimination in the theory $(\mathbb{R}, \leq, +)$.

We have improved the performance of HyTech by representing and manipulating linear regions geometrically: each data region is represented as a union of convex polyhedra. The current implementation of HyTech consists of a MATHEMATICA main program and a collection of C++ subroutines that make use of a polyhedron-manipulation library by Halbwachs [Hal93,HRP94]. The architecture of HyTech is shown in Figure 1.1.

[3] HyTech is available by anonymous ftp from ftp.cs.cornell.edu, cd ~pub/tah/HyTech. See also http://www.cs.cornell.edu/Info/People/tah/hytech.html.

1.2.5 Model-checking Strategies

To make HyTech more efficient and effective, we designed and implemented several model-checking strategies for the *reachability problem* for hybrid automata. The reachability problem for a hybrid automaton, an initial region, and a final region asks if the final region can be reached from the initial region by the trajectories of the hybrid automaton.

Forward versus Backward Analysis

HyTech can attack a reachability problem by forward analysis, or by backward analysis or both. For a given reachability problem, one direction may perform better than the other direction. In fact, one direction may terminate and the other not. HyTech can also iteratively refine the approximation in both directions until sufficient precision is obtained.

Abstract Interpretation

To expedite the reachability analysis and to force the termination of the analysis, HyTech provides several abstract-interpretation operators [CC77,HH95c], including the convex-hull operator and the extrapolation operator. Our extrapolation operator is similar to the widening operator of [CH78,Hal93].

An abstract-interpretation operator approximates a set of convex regions with a single convex region. The *convex-hull operator* overapproximates a union of convex regions by its convex hull. The *extrapolation operator* overapproximates a directed chain $R \subset f(R) \subset f^2(R) \subset \cdots$ of convex regions by a "guess" of the limit of the computation. Either operator, or the combination of both operators, may cause the termination of a forward or backward reachability analysis that does not terminate otherwise. However, since the use of either operator results in an overapproximation of the target region, the abstract analysis is sound but

not complete: if HyTech says the final region is not reachable from the initial region, then the final region is indeed not reachable. But if HyTech says the final region is reachable from the initial region, the final region may or may not reachable. In the latter case, we have to refine our approximation, by applying fewer abstract-interpretation operators, or by using two-way iterative approximation[CC92,DW95].

1.2.6 Verification of Nonlinear Hybrid Systems

HyTech can be used to automate the verification of nonlinear hybrid systems with the help of two algorithmic translations, the *clock translation* and the *rate translation* [HH95a], from nonlinear hybrid automata to linear hybrid automata. Both translations are sound and, when it applies, the clock translation is also complete (up to numeric errors). A by-product of the clock translation is a new decidable class of hybrid systems.

1.2.7 Case Studies

We have applied HyTech to analyze more than 30 hybrid-system benchmarks. Besides some small examples, we show the application of HyTech to three non-trivial benchmark problems taken from the literature, rather than devised by us. The first case study is a distributed control system introduced by Corbett [Cor94]. The system consists of a controller and two sensors, and is required to issue control commands to a robot within certain time limits. The two sensor processes are executed on a single processor, as scheduled by a priority scheduler. This scenario is modeled by linear hybrid automata with clocks and integrators. HyTech automatically computes the maximum time difference between two consecutive control commands generated by the controller.

The second case study is a two-robot manufacturing system introduced by Puri

and Varaiya [PV95b]. The system consists of a conveyor belt with two boxes, a service station, and two robots. The boxes will not fall to the floor iff initially the boxes are not positioned closely together on the conveyor belt. HyTech automatically computes the minimum allowable initial distance between the two boxes.

The third case study is the Philips audio control protocol presented by Bosscher, Polak, and Vaandrager [BPV94]. The protocol consists of a sender that converts a bit string into an analog signal using the so-called Manchester encoding, and a receiver that converts the analog signal back into a bit string. The sender and the receiver use clocks that may be drifting apart. In [BPV94], it was shown, by a human proof, that the receiver decodes the signal correctly if and only if the clock drift is bounded by a certain constant. HyTech automatically computes that constant for input strings up to 8 bits. In [HW95], Howard Wong-Toi and the author applied HyTech to verify the protocol with input strings of arbitrary length. This result is not included in this dissertation.

1.3 Related Work

The research of hybrid systems has a short history and yet fruitful results in both computer science and control theory. The related research of hybrid systems can be classified into two categories: Verification and Control. The verification work has benefit from the control-related research [HWT95]. On the other hand, as Anil Nerode predicted in [Hub95], the process of extracting the control automata will also be benefit from the verification techniques. A clear evidence of the trend is shown in [MPS95], which shows that the verification techniques developed in this thesis can be applied to extract controllers.

1.3.1 Verification

Both algorithmic approaches and deductive approaches have been used for the verification of hybrid systems. The algorithmic approaches verify the modeled systems automatically. The deductive approaches are often aided by machine provers but still require human intelligence to verify modeled systems. On the other hand, the deductive methods can usually verify a more general class of hybrid systems. We divide the verification work on hybrid systems into eight classes. The first three classes are algorithmic approaches and the last five classes are deductive approaches.

Hybrid Automata

Most algorithmic approaches use the hybrid automaton model that we introduce in this dissertation.

Hybrid automaton model. The first related work dated back in 1990, Alur and Dill [AD90] investigated *timed automata*, an extensions of ω-automata with real-valued clocks, for expressing real-time systems. In parallel, Henzinger, Manna, and Pnueli [HMP92,HMP94] also introduced *timed transition systems* to model real time systems. Subsequently, Maler, Manna, and Pnueli [MMP92] and also Nicollin, Sifakis and Yovin [NSY93] generalized the timed transition systems to hybrid phase transition systems. On the other hand, we extended timed automata to hybrid automata in [ACHH93]. In the same workshop proceedings, Nicollin, Olivero, Sifakis and Yovine [NOSY93] independently proposed and studied a similar model.

Model checking. In [ACD93], Alur, Courcoubetis and Dill presented an enumerative model checking algorithm for verifying TCTL-requirements of timed automata. After a while, Henzinger, Nicollin, Sifakis, and Yovine [HNSY94] presented a symbolic model-checking algorithm for the same purpose. Later we generalized this symbolic model-checking algorithm to become a symbolic model-

checking procedure for hybrid systems. In [HH95c], we substituted the quantifier-elimination operations in our symbolic model-checking procedure with more efficient polyhedron-manipulation operations provided in Halbwachs's polyhedron manipulation library [Hal93]. In [HRP94], Halbwachs, Raymond and Proy also applied Halbwachs's polyhedron manipulation library to verify some hybrid system benchmarks presented in [AHH93,HH95c].

Abstract interpretation. The abstract interpretation techniques [CC77] were first applied to real-time systems and then hybrid systems. Halbwachs [Hal93] used two abstract interpretation operators, the convex hull operator and the widening operator for verifying real-time systems. Shortly after, Dill and Wong-Toi [DW93, DW95] also applied the two-way iterative approximation method [CC92] to the verification of timed automata. It was shown that all the above abstract interpretation methods can be applied to hybrid systems in [HH95c] and also in [HRP94] in parallel. In addition, an extrapolation operator similar to the widening operator was investigated in [HH95c].

In CAV'94 (Conference on Computer Aided Verification), Olivero, Sifakis and Yovine [OSY94] introduced translations (abstract interpretation operators) that translate some classes of linear hybrid automata into timed automata. The translated timed automata can be verified by their symbolic model checker KRONOS [DY95] based on [HNSY94]. Since the reachability problem and even the TCTL model checking is decidable for timed automata, the by-product of this work is some decidability results. Similar transformations were also studied by Howard Wong-Toi in his dissertation [WT94].

Theory. There has been lots of theoretical work on hybrid automata. Kesten, Pnueli, Sifakis and Yovine [KPSY93] introduced a class of decidable linear hybrid automata called *integration graphs*. In CAV'94, Puri and Varaiya [PV94] proved that the reachability problem for a class of linear hybrid automata is de-

cidable. Shortly after, this work was generalized by Henzinger, Kopke, Puri and Varaiya [HKPV95]. They introduced a translation from a even more general class of linear hybrid automata to timed automata. The work thus implies a more general decidability result. The same paper also finds a sharp boundary between decidable and undecidable linear hybrid automata.

Henzinger and Kopke [HK95] investigated the hybrid automata that have finite mutual simulations. The finite mutual simulation is sufficient for reachability analysis and thus this result leads to verification procedures as well as decidability results. Other decidability and undecidability results can be found in [ACHH93, KPSY93,AD94,BER94,MV94,BR95,HH95a,HHK95].

Case studies. Corbett [Cor94] developed a hybrid-automaton-like model and a formal verification tool to specify and verify timing properties of Ada programs. In the paper, Corbett demonstrated his tool by modeling and verifying distributed control systems in Ada. In Chapter 7, we apply HyTech to verify a system of the same kind. To compare the two tools, Corbett also wrote a translator that translates his model to linear hybrid automata.

Bosscher, Polak and Vaandrager applied the notion of hybrid systems to specify and verify a Philips audio control protocol [BPV94]. This timing-based protocol was modeled by an extension of the timed I/O automata model [LV93,LV92], and verified mathematically without computer support. Then we verified an instance of this protocol in [HH95b] and the general protocol in [HW95]. Shortly after, Daws and Yovine [DY95] applied Kronos, and Bengtsson, Larsen, Larsson, Pettersson, and Yi [BLL+95,LPY95] applied UPPAAL to verify the same protocol. Based on the model checking methodology for timed automata in [HNSY94], Kronos and UPPAAL are symbolic model checkers for timed automata. Both tools are applicable to the audio control protocol because the linear hybrid automata that model this protocol can be translated into timed automata. The translation was

done by hand in [DY95] and automatically by a program called `hs2ta` in [BLL+95].

Other case studies for verifying hybrid systems using hybrid automata can be also found in [HHWT95a,HHWT95b].

Nonlinear hybrid automata. The first result on the algorithmic formal verification of nonlinear hybrid automata was the clock translation and the rate translation in [HH95a] (Chapter 6). Subsequently, Henzinger and Wong-Toi [HWT95] introduced the *phase portrait approximation* method to translate nonlinear hybrid automata to linear hybrid automata. The phase portrait approximation can be made more accurate than the rate translation. However, a good phase portrait approximation requires a seemly insight of the differential equations that govern the continuous activities.

Symbolic Reduction

Another algorithmic approach is the symbolic reduction method introduced by Kurshan and McMillan [KM91]. Kurshan and McMillan applied symbolic reduction to analyze a circuit modeled at the transistor level. The circuit at the transistor level is essentially a finite-state machine with differential equations that govern the analog behavior of the circuit. Kurshan and McMillan applied symbolic reduction to obtain a conservative finite-state model of the circuit such that any safety property verified in the finite-state model is true for the circuit. We plan to adapt the symbolic reduction method to become a sound translation from nonlinear hybrid automata to finite-state machines.

Martin and Seger [MS94] introduced *trace automata* to model linear or nonlinear hybrid systems. They modified the symbolic reduction method to translate trace automata to finite-state machines. The translated finite-state machines can be verified automatically using VOSS [Seg93], a BDD-based model checker.

Dynamical Systems

The third algorithmic approach is the dynamical system approach. Asarin, Maler and Pnueli [AMP95] considered the reachability problem for a class of linear hybrid systems, namely the dynamic systems with piecewise-constant derivatives. They presented a decision procedure for two-dimensional dynamic systems and an undecidability result for three or more dimensions. Guckenheimer and others designed and implemented the tool DSTOOL [Ner92,BGM93] for simulating two or more dimensional hybrid dynamical systems. Zhang and Mackworth [ZM95] also developed a semantic model, *constraint nets*, to model hybrid dynamical systems.

Duration Calculus

The first deductive approach that we survey is the *duration calculus* [CHR91, RRH93,CHS93,HHF$^+$94,HPC95], which is an interval-based temporal logic developed by Chaochen and others. Duration calculus can be used to specify the accumulated time that a state predicate holds, namely, the *duration* of the state predicate. In the framework of the duration calculus, both the system and the requirement are specified by duration calculus formulas. To verify if a system satisfies a requirement, a deductive system for duration calculus is used to check the validity of the duration calculus formulas.

Most decidability results for the duration calculus are negative. The most recent good news is in [BLR95]; Bouajjani and others show that the verification of a subclass of duration calculus properties can be reduced to reachability problems for a subclass of integration graphs that can be analyzed using the algorithm of [ACH93]. The reachability problems for integration automata is decidable [KPSY93]. Consequently, this subclass of duration calculus properties is decidable and can be automatically verified by HyTech.

Lakhaneche and Hooman [LH95] extended *metric temporal logic* with durations

to specify and verify duration properties of hybrid systems using their proof system.

Phase Transition Systems

Manna and Pnueli [MP93] designed the phase transition system model for modeling hybrid systems and also a proof rule for verifying phase transition systems. Kapur, Henzinge, Manna and Pnueli [HMP93,KHMP94] introduced a proof system for verifying phase transition systems against hybrid temporal logic (HTL) specifications.

TLA+

TLA+ [AL92,Lam93,Lam94] is a general-purpose formal specification language based on Temporal Logic in Actions, with no built-in primitives for specifying real-time properties. Lamport [Lam93] used TLA+ to define operators for specifying real-time and duration properties. These operators are used to specify and prove a gas burner example (Section 4.4.2) in the paper. Both the system, F, and the requirement, G, are specified by TLA+ formulas. The system F satisfies the requirement G if all behaviors satisfy the formula $F \Rightarrow G$, which can be proved or disproved by a proof system for TLA+. In [Lam93], the proof for the gas burner example was done manually, but the proof could be also assisted by TLP [EGL92], a system for mechanically checking TLA proofs.

Timed I/O automata

In [BPV94], Bosscher, Polak and Vaandrager extended the timed I/O automata model [LV93,LV92] to specify linear hybrid systems. In the same paper, they applied the timed I/O automata model to specify and verify a Philips audio control protocol. Both the system and the requirement are described by timed I/O automata. The correctness proof is done by the simulation mapping technique that

proves the inclusion of timed traces. The proof was done manually. According to [BPV94], a system for mechanically verifying systems and requirements specified in the timed I/O automata model is being developed.

Hoare Logic

Hooman [Hoo93] extended the classical Hoare triples with timing primitives to specify hybrid systems. A compositional proof system is also introduced to verify the specifications in Hoare triples.

1.3.2 Control

Both control theorists and computer scientists are advancing the techniques of extracting hybrid-system controllers. We divide the work in the area into four classes.

Nerode and Kohn's Model

We first discuss the work of Nerode, Kohn, and others [GKNY92,Ner93,NK93b, NK93a,KN93,KNRY95,LGKN95,KJN$^+$95,GNKJ95].

A simple example of Nerode and Kohn's hybrid system model [KN93] is illustrated in Figure 1.2. The hybrid system consists of a continuous plant interacting with a digital controller at times $n \cdot \Delta$. The digital controller is a finite automaton that issues control command c_n to the plant at times $n \cdot \Delta$, based on the state of the control automaton and a measurement of the state of the plant. The plant runs based on the control c_n in the time interval $[n\Delta, (n+1)\Delta]$. The digital command issued by the controller is converted to analog actuation by the D/A converter. The sensor supplies feedback to the digital control automaton through the A/D converter.

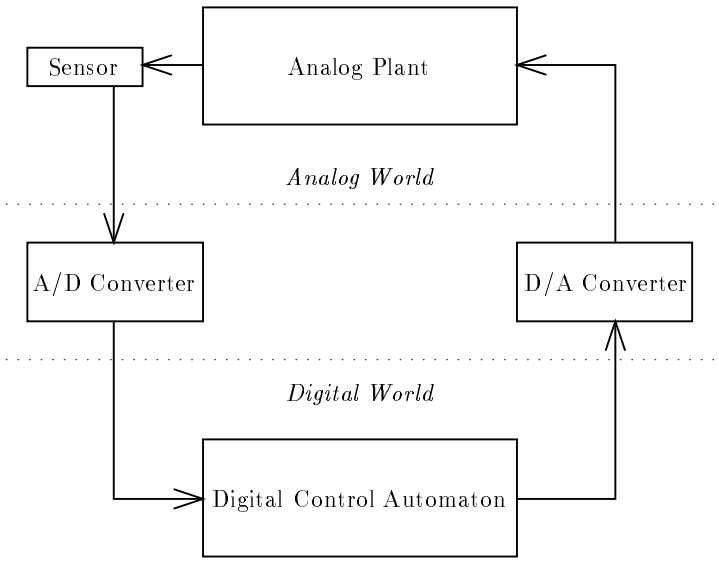

Figure 1.2: Nerode and Kohn's model

The continuos plant is described by an ordinary vector differential equation

$$\dot{x} = f(x, c, d),$$

where x is the state of the plant, c is the control function, and d is the disturbance function. For any initial plant state $x(0)$, and for any control function $c(t)$ and disturbance function $d(t)$, the function $x(t)$ that satisfies the differential equation is the *trajectory* of the plant. The challenge of the controller is to make the trajectory of the plant stays within acceptable bounds.

The *performance specification* for the hybrid-system controller is expressed by the value of a single non-negative cost function, *Lagrangian*, on the trajectory. The integral of the Lagrangian is computed on the plant trajectories, the smaller its value the better the performance. A trajectory is *acceptable* if the integral of the Lagrangian along the trajectory is within ϵ of the minimum. The goal is to extract digital control automata that will control the plant to generate acceptable trajectories only.

Kohn's *declarative control* can be used for solving this problem. Basically, the Lagrangian constraint on the trajectory can be translated into symbolic formulas for desirable control automaton using the method. Kohn and Nerode [KNRG94, KNR95] suggested that the state space of a hybrid system should be considered as a differentiable manifold, the *carrier manifold*. A point (state) of the carrier manifold represents the state of all the digital and analog variables. The digital variables are "continualized" to become analog variables; that is, view the digital variables as finite real-valued piecewise constant functions of continuous time and then smooth them. Every constraint on the system, including the Lagrangian constraint, is incorporated into the definition of what points are on the carrier manifold. Then we consider the *feedback function* $y = F(x)$ of each point x on the carrier manifold. The control problem can be reduced to the following problem: for each point x on the carrier manifold, find a direction $\dot{x}$ to reach a state x_1 at a specific direction $\dot{x}_1 = y_1$. The choice of the vector $y = F(x)$ specifies that in what direction $\dot{x} = y$ should go when the plant is in state x. So the control is determined by the feedback function. Kohn and Nerode have shown how to formulate the feedback function construction problem as a single relaxed variational problem, and how to solve the latter problem for the required controls.

To implement appropriate adaptive controller handling unmodeled dynamics and other uncertainties, sometimes it is necessary to compute and install a new finite control automaton to replace the old one on the fly when performance specifications for the system are violated. Kohn and Nerode also introduced the concept of an *autonomous agent* to accomplish this task. An autonomous agent monitors a hybrid system like the example in Figure 1.2. The agent looks for non-compliance with specifications and uses this and other historical information to compute, and occasionally install, a new finite control automaton.

This concept can be generalized to a set of distributed cooperating agents. The

idea is as follows. In addition to the global Lagrangian for the whole hybrid system, each agent is assigned a local Lagrangian based on the evolution of the continuous process it monitors. The distributed agents cooperate through message passing. The messages received by an agent are symbolic terms that are used to modify that agent's current Lagrangian. Each agent autonomously extracts a control automaton for its monitored process to force the monitored process to produce a trajectory approximately minimizing that agent's local Lagrangian. The definition of a successful cooperation is that the local and global Lagrangians are designed so that choosing a control automaton that lowers an individual agent's Lagrangian also lowers the global Lagrangian.

This multi-agent architecture can be used to control the hybrid systems for which a single agent can not possibly meet the performance requirement. Applications include the national distributed multi-media systems, interactive video-audio-text on demand, manufacturing processes, virtual enterprises, distributed interactive simulation, and traffic control.

Fixpoint Computation

In [MPS95], Maler, Pnueli and Sifakis show that the synthesis of hybrid-system controllers can be done by the computation of fixpoints of the precondition operators that we developed for the symbolic model-checking algorithm (Chapter 3). From this result, our symbolic model-checking efforts for hybrid automata can be applied to the synthesis of hybrid-system controllers.

PATH

The California PATH project [Var93] uses the notion of hybrid systems to develop more efficient highway systems. Besides the theoretical work of Puri and Varaiya like [PV94,PV95b] that we mentioned earlier, the PATH hybrid control papers

include [DV95b,DV95a,PV95a].

Other Results

Other well-known results of extracting hybrid-system controllers in the framework of control theory include [GL93,LSA93,ASL93,GL95,Bra95,LA95].

Chapter 2

Modeling Languages

By relieving the mind of all unnecessary work,
a good notation sets it free to concentrate on
more advanced problems, and in effect increases
the mental power of the race.
— Alfred North Whitehead

We now introduce the system modeling language, hybrid automata, and the requirement modeling language, ICTL.

2.1 System Modeling Language: Hybrid Automata

Informally, a hybrid automaton is a labeled multigraph (V, E) with a finite set $\vec{x}$ of real-valued variables. The real-valued variables are used to model the continuous environment activities. The edges E represent discrete actions and are labeled with nondeterministic guarded assignments to $\vec{x}$. The vertices V represent different control modes of the system and are labeled with constraints on the derivatives

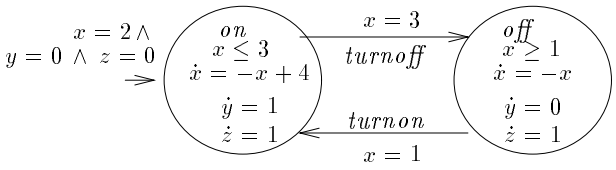

Figure 2.1: Thermostat automaton

of $\vec{x}$. The state of the automaton changes either through instantaneous system actions or, while time elapses, through continuous activities.

For example, the hybrid automaton of Figure 2.1 models a thermostat that controls the temperature of a manufacturing plant. The variable x models the plant temperature. In control location *on*, a heater is turned on; in control location *off*, the heater is turned off. The variable x follow the differential equations $\dot{x} = -x+4$ in location *on* and $\dot{x} = -x$ in location *off*. Thus the plant temperature follows exponential functions with negative exponents. Initially, the temperature is 2 and the heater is turned on. If a thermometer detects that the plant temperature reaches 3, the heater is turned off. If the thermometer detects that the temperature falls to 1, the heater is turned on. The two variables y and z are auxiliary variables: y, a stop watch, records the accumulated time that the heater is turned on, and z, a clock, records the total elapsed time.

2.1.1 Syntax

Now we introduce the formal definition of hybrid automata. Let $\vec{y}$ be a vector of real-valued variables. A *linear term* over $\vec{y}$ is a linear combination of variables from $\vec{y}$. A *linear inequality* over $\vec{y}$ is an inequality between linear terms over $\vec{y}$. A (*closed*) *convex linear formula* over $\vec{y}$ is a finite conjunction of (nonstrict) linear inequalities over $\vec{y}$. A *linear formula* over $\vec{y}$ is a finite boolean combination of linear inequalities over $\vec{y}$. Every linear formula can be transformed into disjunctive

normal form, that is, into a finite disjunction of convex linear formulas.

A *hybrid automaton* A consists of the following components:

Data variables A finite vector $\vec{x} = (x_1, \ldots, x_n)$ of real-valued *data variables*. The size n of $\vec{x}$ is called the *dimension* of A. The thermostat automaton in Figure 2.1 has the vector (x, y, z) of data variables. The dimension of the thermostat automaton is thus 3.

A *data state* is a point $\vec{s} = (s_1, \ldots, s_n)$ in n-dimensional real space $\mathbb{R}^n$ or, equivalently, a function that assigns to each data variable x_i a real value $s_i \in \mathbb{R}$. A *convex data region* is a convex polyhedron in $\mathbb{R}^n$. A *data region* is a finite union of convex data regions. A (*convex*) *data predicate* is a (convex) linear formula over $\vec{x}$. The (convex) data predicate p defines the (convex) data region $[\![p]\!] \subseteq \mathbb{R}^n$, where $\vec{s} \in [\![p]\!]$ iff $p[\vec{x} := \vec{s}]$ is true. The data region S is (*convex*) *linear* if there is a (convex) data predicate that defines S.

For each data variable x_i, we use the dotted variable $\dot{x}_i$ to denote the first derivative of x_i. A *rate predicate* is a conjunction of inequalities between linear terms over $\vec{x} \uplus \dot{\vec{x}}$ and differentiable functions $f(\vec{x})$ over $\vec{x}$. A rate predicate is *linear* if it is a convex linear formula over $\vec{x} \uplus \dot{\vec{x}}$. A rate predicate is *uniform* if it is a convex linear formula over the set $\dot{\vec{x}}$ of dotted variables only.

A *differential inclusion* is a connected region in $\mathbb{R}^n$. A rate predicate r maps each data state $\vec{s} \in \mathbb{R}^n$ to the differential inclusion $[\![r]\!](\vec{s}) \subseteq \mathbb{R}^n$, where $\dot{\vec{s}} \in [\![r]\!](\vec{s})$ iff $r[\vec{x} := \vec{s}, \dot{\vec{x}} := \dot{\vec{s}}]$ is true. A linear rate predicate maps each data state to a differential inclusion that is a convex polyhedron in $\mathbb{R}^n$. A uniform rate predicate maps every data state to the same differential inclusion that is a convex polyhedron in $\mathbb{R}^n$. A differential inclusion is *bounded* if it is contained in an open ball with finite radius.

For each data variable x_i, we use the primed variable x_i' to denote the new value of x_i after a transition. An *action predicate* is a convex linear formula over the set $\vec{x} \uplus \vec{x}'$ of data variables $\vec{x}$ and primed variables $\vec{x}' = (x_1', \ldots, x_n')$. The action predicate q maps each data state $\vec{s} \in \mathbb{R}^n$ to the convex data region $[\![q]\!](\vec{s}) \subseteq \mathbb{R}^n$, where $\vec{s}' \in [\![q]\!](\vec{s})$ iff $q[\vec{x} := \vec{s}, \vec{x}' := \vec{s}']$ is true.

The data or action predicate p is (*real-valued*) *integral* if all constants that occur in p are (real-valued) integral respectively. Notice that any data predicate containing a rational coefficient is equivalent to an integer data predicate. The hybrid automaton A is (*real-valued*) *integral* if all data and action predicates that specify the invariants and actions of A are (real-valued) integral.

Control locations A finite set V of vertices called *control locations*. The thermostat automaton has the control locations *on* and *off*.

A *state* $(v, \vec{s})$ of the automaton A consists of a control location $v \in V$ and a data state $\vec{s} \in \mathbb{R}^n$. A *region* $R = \bigcup_{v \in V}(v, S_v)$ is a collection of data regions $S_v \subseteq \mathbb{R}^n$, one for each control location $v \in V$. A *state predicate* $\phi = \bigcup_{v \in V}(v, p_v)$ is a collection of data predicates p_v, one for each control location $v \in V$. The state predicate ϕ defines the region $[\![\phi]\!] = \bigcup_{v \in V}(v, [\![p_v]\!])$. The region R is *linear* if there is a state predicate that defines R.

We write (v, S) for the region $(v, S) \cup \bigcup_{v' \neq v}(v', \emptyset)$, and (v, p) for the state predicate $(v, p) \cup \bigcup_{v' \neq v}(v', \mathit{false})$. When writing state predicates, we use the *location counter* ℓ, which ranges over the set V of control locations. The location constraint $\ell = v$ denotes the state predicate (v, true). The data predicate p, when used as a state predicate, denotes the collection $\bigcup_{v \in V}(v, p)$. For two state predicates $\phi = \bigcup_{v \in V}(v, p_v)$ and $\phi' = \bigcup_{v \in V}(v, p_v')$, we define $\neg\phi = \bigcup_{v \in V}(v, \neg p_v)$, $\phi \vee \phi' = \bigcup_{v \in V}(v, p_v \vee p_v')$, and $\phi \wedge \phi' = \bigcup_{v \in V}(v, p_v \wedge p_v')$.

A state predicate is *integral*, *rational*, or *real-valued* if it contains only integral, rational or real-valued data predicates, respectively.

Location invariants A labeling function inv that assigns to each control location $v \in V$ a convex data predicate $inv(v)$, the *invariant* of v. The invariants are used to enforce the progress of a system from one control location to another, because the control of the automaton A may reside in the location v only as long as the invariant $inv(v)$ is true. In the thermostat automaton, the invariants of the locations *on* and *off* are $x \leq 3$ and $x \geq 1$, respectively.

The state $(v, \vec{s})$ is *admissible* if $\vec{s} \in [\![inv(v)]\!]$. We write Σ_A for the set of admissible states of A, and ϕ_A for the state predicate $\bigcup_{v \in V}(v, inv(v))$ that defines the set of admissible states.

Continuous activities A labeling function dif that assigns to each control location $v \in V$ a rate predicate $dif(v)$ that maps every data state in $inv(v)$ to a bounded differential inclusion. The rate predicate constrains the rates at which the values of data variables change: when the state of the automaton is $(v, \vec{s})$, the first derivatives of all data variables stay within the bounded differential inclusion $[\![dif(v)]\!](\vec{s})$.

The data variable x is *linear* if for all locations $v \in V$, the rate predicate $dif(v)$ is uniform; otherwise, x is *nonlinear*. The integral hybrid automaton A is *linear* if all data variables in $\vec{x}$ are linear; otherwise, A is *nonlinear*. In the thermostat automaton, $dif(on) = \dot{x} = -x + 4 \wedge \dot{y} = 1 \wedge \dot{z} = 1$ and $dif(off) = \dot{x} = -x \wedge \dot{y} = 0 \wedge \dot{z} = 1$. The data variables y and z are linear and x is nonlinear. Thus the thermostat automaton is a nonlinear integral hybrid automaton.

Transitions A finite multiset E of edges called *transitions*. Each transition (v, v')

identifies a source location $v \in V$ and a target location $v' \in V$. For each location $v \in V$, there is a *stutter transition* $e_v = (v, v)$.

Discrete actions A labeling function *act* that assigns to each transition $e \in E$ an action predicate $act(e)$, the *action* of e. If the automaton control proceeds from the location v to the location v' via the transition $e = (v, v')$, then the values of all data variables change nondeterministically from $\vec{s}$ to a point in the data region $[\![act(e)]\!](\vec{s})$. For example, a transition with the action label

$$x_1 \leq 3 \ \wedge \ 3 \leq x_1' \leq 5 \ \wedge \ x_2' = x_2 \ \wedge \ x_3' = x_1 + 1$$

can be traversed only when the value of x_1 is at most 3. The transition updates the value of x_1 to a real number in the interval $[3, 5]$, the value of x_2 remains unchanged, and the new value of x_3 is 1 greater than the old value of x_1. All stutter transitions are labeled with the action predicate $\vec{x}' = \vec{x}$. In the graphical representation of actions, if the primed variable x' occurs only in the conjunct $x' = x$, then we usually omit that conjunct. In the thermostat automaton, $act(on, off)$ is $x = 3 \wedge x' = x \wedge y' = y \wedge z' = z$.

Synchronization labels A finite set L of *synchronization labels* and a labeling function *syn* that assigns to each transition $e \in E$ a label from L. The set L is called the *alphabet* of A. The synchronization labels are used to define the parallel composition of two automata: if both automata share a synchronization label a, then each a-transition of one automaton must be accompanied by an a-transition of the other automaton. The stutter transition e_v of location v is labeled with v. In the thermostat automaton, $label(on, off)$ is *turnoff* and $label(on, off)$ is *turnon*.

The convexity restriction to invariants, activities, and actions does not limit the expressiveness of hybrid automata, because nonconvex invariants and activities

can be modeled by splitting locations (see Chapter 3), and nonconvex actions can be modeled by splitting transitions.

The data variable x of the hybrid automaton A is a *clock* if $dif(v)$ implies $\dot{x} = 1$ for all locations v of A; that is, each clock always increases with the rate at which time advances. The data variable x of the hybrid automaton A is a *skewed clock* if $dif(v)$ implies $\dot{x} = c$ for all locations v of A, where c is a nonnegative integer. The data variable x of the hybrid automaton A is an *integrator* if $dif(v)$ implies $\dot{x} = 1$ or $\dot{x} = 0$ for all locations v of A. An hybrid automaton A is a *n-rate automaton* if all data variables of A are skewed clocks that proceed in n different rates. An hybrid automaton A is an *integrator automaton* if all data variables of A are integrators.

Another special case of a linear hybrid automaton is a timed automaton [AD94]. An atomic data predicate is *simple* if it has the form $x \leq c$ or $x \geq c$, for some $c \in \mathbb{R}$; an atomic action predicate is *simple* if it is a simple atomic data predicate or has the form $x' = c$ or $x' = x$. The data variable x of A is *simple* if in all initial conditions, invariants, and actions of A, x and x' occur only in atomic data and action predicates that are simple. The hybrid automaton A is *simple* if all data variables of A are simple. A simple rational hybrid automaton A is a *timed automaton* if all data variables of A are clocks.

2.1.2 Semantics

At any time instant, the state of a hybrid automaton specifies a control location and values for all data variables. The state can change in two ways: (1) by an instantaneous transition that changes both the control location and the values of data variables, or (2) by a time delay that changes only the values of data variables in a continuous manner according to the rate predicate of the current control location.

A *data trajectory* (δ, ρ) of the hybrid automaton A consists of a nonnegative *duration* $\delta \in \mathbb{R}_{\geq 0}$ and a differentiable function $\rho\colon [0, \delta] \to \mathbb{R}^n$. The data trajectory (δ, ρ) maps every real $t \in [0, \delta]$ to the data state $\rho(t)$. The data trajectory (δ, ρ) is *admissible* if there is some location v of hybrid automaton A such that

Invariants for all reals $t \in [0, \delta]$, $\rho(t) \in [\![inv(v)]\!]$; and

Activities for all reals $t \in (0, \delta)$, $\frac{d\rho(t)}{dt} \in [\![dif(v)]\!](\rho(t))$.

Consider two reals t_1 and t_2 with $0 \leq t_1 \leq t_2 \leq \delta$. We write $\rho[t_1, t_2]$ for the data trajectory (δ', ρ') with $\delta' = t_2 - t_1$, and $\rho'(t) = \rho(t + t_1)$ for all $t \in [0, \delta']$. The data trajectory (δ, ρ) is *linear* if there is a constant rate vector $\vec{s} \in \mathbb{R}^n$ such that $\frac{d\rho(t)}{dt} = \vec{s}$ for all $t \in (0, \delta)$. The data trajectory (δ, ρ) is *piecewise linear* if there are *finitely* many reals $t_1, \ldots, t_k \in [0, \delta]$ such that the data trajectories $\rho[0, t_1], \rho[t_1, t_2], \ldots, \rho[t_k, \delta]$ are linear.

A *trajectory* τ of A is an infinite sequence

$$(v_0, \delta_0, \rho_0) \;\to\; (v_1, \delta_1, \rho_1) \;\to\; (v_2, \delta_2, \rho_2) \;\to\; (v_3, \delta_3, \rho_3) \;\to\; \cdots$$

of control locations $v_i \in V$ and admissible data trajectories (δ_i, ρ_i) in location v_i such that for all $i \geq 0$, there is a transition $e_i = (v_i, v_{i+1}) \in E$ with $\rho_{i+1}(0) \in [\![act(e_i)]\!](\rho_i(\delta_i))$.

A *position* of the trajectory τ is a pair (i, ϵ) that consists of a nonnegative integer i and a nonnegative real $\epsilon \leq \delta_i$. The positions of τ are ordered lexicographically: the position (i, δ) precedes the position (j, ϵ), denoted $(i, \delta) < (j, \epsilon)$, iff either $i < j$, or $i = j$ and $\delta < \epsilon$. The *state at position* (i, ϵ) of τ is $\tau(i, \epsilon) = (v_i, \rho_i(\epsilon))$ (notice that all states of τ are admissible). The *time at position* (i, ϵ) of τ is the finite sum $t_\tau(i, \epsilon) = (\Sigma_{0 \leq j < i}\, \delta_j) + \epsilon$. The *duration* of the trajectory τ is the infinite sum $\delta_\tau = \Sigma_{j \geq 0}\, \delta_j$.

The trajectory τ *diverges* if $\delta_\tau = \infty$. The trajectory τ is *linear* if all data trajectories (δ_i, ρ_i) of τ are linear. By $[\![A]\!]$ we denote the set of trajectories of the automaton A. If $\mathcal{T}$ is a set of trajectories, $\mathcal{T}^{div}$ is the set of divergent trajectories in $\mathcal{T}$, and $\mathcal{T}_{lin}$ is the set of linear trajectories in $\mathcal{T}$.

Consider two positions $\pi_1 = (v_i, \epsilon_1)$ and $\pi_2 = (v_j, \epsilon_2)$ of the trajectory τ. We write $\tau[\pi_1, \pi_2]$ for the trajectory fragment

$$(v_i, \rho_i[\epsilon_1, \delta_i]) \ \rightarrow \ (v_{i+1}, \delta_{i+1}, \rho_{i+1}) \ \rightarrow \ \cdots \ \rightarrow \ (v_j, \rho_j[0, \epsilon_2]),$$

and $\tau[\pi_1, \infty]$ for the trajectory

$$(v_i, \rho_i[\epsilon_1, \delta_i]) \ \rightarrow \ (v_{i+1}, \delta_{i+1}, \rho_{i+1}) \ \rightarrow \ (v_{i+2}, \delta_{i+2}, \rho_{i+2}) \ \rightarrow \ \cdots$$

The trajectory set $[\![A]\!]$ is closed under suffixes (if $\tau \in [\![A]\!]$ and π is a position of τ, then $\tau[\pi, \infty] \in [\![A]\!]$); stuttering (if $\tau \in [\![A]\!]$ and π is a position of τ, then $(\tau[(0,0), \pi]\tau[\pi, \infty]) \in [\![A]\!]$); fusion (if $\tau, \tau' \in [\![A]\!]$, π is a position of τ, π' is a position of τ', and $\tau(\pi) = \tau'(\pi')$, then $(\tau[(0,0), \pi]\tau'[\pi', \infty]) \in [\![A]\!]$); and limits (if for all positions π of τ there is a trajectory $\tau' \in [\![A]\!]$ and a position π' of τ' such that $\tau[(0,0), \pi] = \tau'[(0,0), \pi']$, then $\tau \in [\![A]\!]$). Notice that fusion closure asserts that the future evolution of a hybrid automaton is completely determined by the present state of the automaton. Also notice that the suffix, stutter, and fusion closures of a trajectory set $\mathcal{T}$ are inherited by the subsets $\mathcal{T}^{div}$ and $\mathcal{T}_{lin}$. If $\mathcal{T}$ is closed under limits, then so is $\mathcal{T}_{lin}$, and $\mathcal{T}^{div}$ is closed under divergent limits ("divergence-safe") [HNSY94].

The hybrid automaton A is *nonzeno* if for every admissible state σ of A there is a divergent trajectory τ of A such that $\tau(0,0) = \sigma$. In other words, A is nonzeno iff every finite prefix of a trajectory is a prefix of a divergent trajectory. Notice that if A is nonzeno, then the states that occur on divergent trajectories of A are precisely the admissible states Σ_A. We restrict our attention to nonzeno hybrid automata.

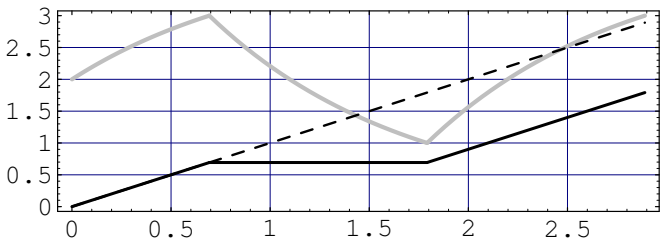

Figure 2.2: A trajectory segment of the thermostat automaton

In [HNSY94] it is shown how a timed automaton may be turned into a nonzeno automaton with the same divergent trajectories; this is done by strengthening the location invariants, and applies to many hybrid automata also.

Figure 2.2 illustrates a segment of a trajectory of the thermostat automaton from Figure 2.1. The thick grey curve represents the temperature x; the dashed straight line represents the clock z; and the piecewise-linear solid line represents the stop watch y.

2.1.3 Composition

A hybrid system typically consists of several components that operate concurrently and communicate with each other. We describe each component as a hybrid automaton. The component automata coordinate through shared data variables that model shared memories, and through synchronization labels that model message-passing coordinations. The hybrid automaton that models the entire system is then constructed from the component automata using a product operation.

Let $A_1 = (\vec{x}_1, V_1, inv_1, dif_1, E_1, act_1, L_1, syn_1)$ and $A_2 = (\vec{x}_2, V_2, inv_2, dif_2, E_2, act_2, L_2, syn_2)$ be two hybrid automata of dimensions n_1 and n_2, respectively. The *product* $A_1 \times A_2$ of A_1 and A_2 is the hybrid automaton $A = (\vec{x}_1 \cup \vec{x}_2, V_1 \times V_2, inv, dif, E, act, L_1 \cup L_2, syn)$:

- Each location (v, v') in $V_1 \times V_2$ has the invariant $inv(v, v') = inv_1(v) \wedge inv_2(v')$ and the activity $dif(v, v') = dif_1(v) \wedge dif_2(v')$; that is, an admissible state of A consists of an admissible state of A_1 and an admissible state of A_2, whose shared parts coincide, and whose rate vector obeys the differential inclusions that are associated with both components locations.

- E contains the transition $e = ((v_1, v_2), (v_1', v_2'))$ iff

  (1) $v_1 = v_1'$ and there is a transition $e_2 = (v_2, v_2') \in E_2$ with $syn_2(e_2) \notin L_1$; or

  (2) there is a transition $e_1 = (v_1, v_1') \in E_1$ with $syn_1(e_1) \notin L_1$, and $v_2 = v_2'$; or

  (3) there is a transition $e_1 = (v_1, v_1') \in E_1$ and a transition $e_2 = (v_2, v_2') \in E_2$ such that $syn_1(e_1) = syn_2(e_2)$.

  In case (1), $act(e) = (\bigwedge_{x \in \vec{x}_1 \setminus \vec{x}_2} x' = x) \wedge act_2(e_2)$ and $syn(e) = syn_2(e_2)$. In case (2), $act(e) = act_1(e_1) \wedge (\bigwedge_{x \in \vec{x}_2 \setminus \vec{x}_1} x' = x)$ and $syn(e) = syn_1(e_1)$. In case (3), $act(e) = act_1(e_1) \wedge act_2(e_2)$ and $syn(e) = syn_1(e_1) = syn_2(e_2)$.

Since the two component automata A_1 and A_2 may share data variables, the dimension of A lies between $max(n_1, n_2)$ and $n_1 + n_2$. According to the definition of E, the transitions of the two component automata are interleaved, provided that there is no label in $L_1 \cap L_2$. Labels in $L_1 \cap L_2$ must be synchronized, and cause the simultaneous traversal of component transitions. Notice that, in cases (1) and (2), the stutter transitions of the component automata result in stutter transitions of the product automaton.

2.1.4 Example: Railroad Gate Controller

We model a control system for a railroad crossing using hybrid automata. The system consists of three processes—a train, a gate, and a gate controller.

The variable x represents the distance of the train from the gate. The dotted variable $\dot{X}$ represents the first derivative of the variable x with respect to time; that is, the velocity of the train. Initially, the train is far from the gate and always moves at a speed that varies between 48 and 52 meters per second. When the train approaches the gate, a sensor that is placed at a distance of 1000 meters from the crossing detects the train and sends the signal *app* to the controller. The train then may slow down to a speed between 40 and 52 meters per second. If the controller is idle upon receipt of the approach signal *app*, it requires up to 5 seconds to send the command *lower* to the gate; the delay of the controller is modeled by the clock z. If the gate is open, it is lowered from 90 radius degrees to 0 degrees at the constant rate of 20 degrees per second; the position of the gate in degrees is represented by the variable y. A second sensor placed at 100 meters past the crossing detects the leaving train and signals *exit* to the controller, which, after another delay of up to 5 seconds, sends the command *raise* to the gate. We assume that the distance between consecutive trains is at least 1500 meters, so when the sensor detects a leaving train, the next (or returning) train is at least 1500 meters from the crossing.

The controller must accept arriving *app* and *exit* signals at any time, and the gate must always accept controller commands. For fault tolerance considerations, we design the controller so that an *exit* signal is ignored if the gate is about to be lowered, while an *app* signal always causes the gate to be lowered.

The three hybrid automata that model the train, the gate, and the controller are shown in Figure 2.3. In the graphical representation of the automata we use

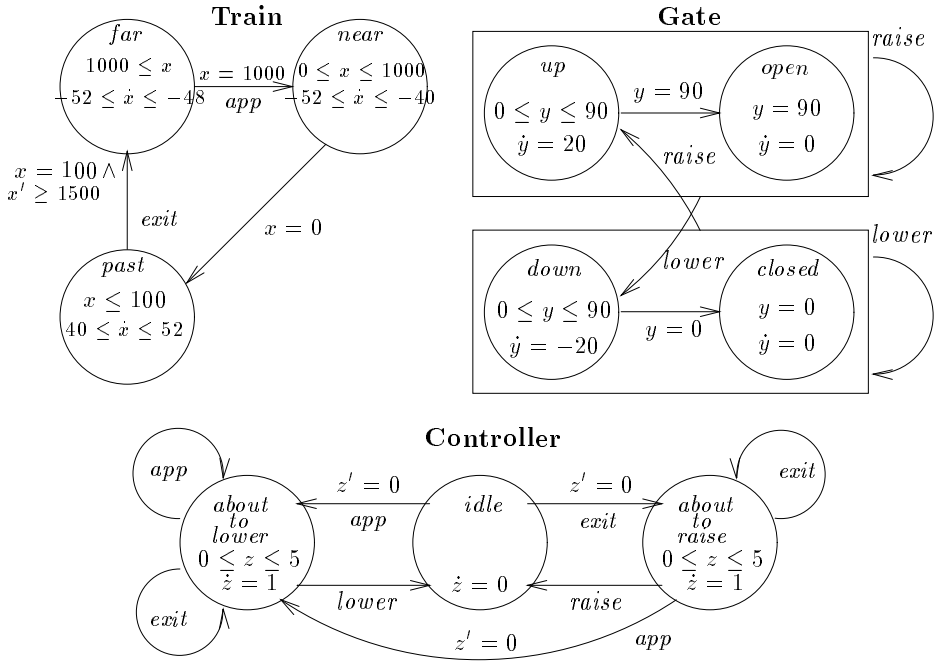

Figure 2.3: Railroad gate controller

"superlocations" to save on edges. In particular, the gate automaton has four locations—*up* ("being raised"), *open*, *down* ("being lowered"), and *closed*. We suppress invariants of the form *true*, conjuncts of the form $\dot{x} = 0$ for activities, conjuncts of the form $x' = x$ for actions, and we suppress stutter transitions. The location *up* of the gate automaton has the invariant $0 \leq y \leq 90$, the activity $\dot{y} = 20$, and two outgoing transitions; the transition to the location *open* has the activity $y = 90 \wedge y' = y$, and no synchronization labels; the transition to *down* has the activity $y' = y$ and the synchronization label *lower*. The synchronization labels model signals (from the train to the controller) and commands (from the controller to the gate). For instance, when the train automaton changes location using an edge labeled with *app*, the controller automaton is required to traverse an edge with the same label.

2.1.5 Decidability and Undecidability Results

Define the *Muller accepting condition* for a hybrid automaton A to be a collection of location sets $\mathit{final}(A) \subseteq 2^V$. A *Muller accepting trajectory* τ of A is a trajectory of A such that $\tau_\infty \in \mathit{final}(A)$, where τ_∞ is the set of locations that are visited infinitely often during τ.

The design of verification algorithms for hybrid automata is impaired by the fact that the Muller emptiness problem ("Does a hybrid automata have an Muller accepting trajectory?") is undecidable already for very restricted classes of hybrid automata. On the positive side, the Muller emptiness problem for timed automata (only clock variables) is PSPACE-complete [AD94]. On the negative side, we have the following undecidability result.

Theorem 1 *The Muller emptiness problem is undecidable for 2-rate automata and for simple integrator automata.*

Proof. The first part of the theorem follows from the undecidability of the halting problem for nondeterministic 2-counter machines (NCMs). Given any two distinct clock rates, a 2-rate automaton system can encode the computations of an NCM. Suppose we have three clocks of rate 1 and two skewed clocks x_1 and x_2 of rate 2. Then we can encode the values of two counters in the i-th machine configuration by the values of x_1 and x_2 at accurate time i: the counter value n is encoded by the clock value $1/2^n$.

The clock y is reset whenever it reaches 1 and thus marks intervals of length 1. It is obvious how a counter can be initialized to 0 and tested for being 0. Hence it remains to be shown how a counter can be incremented and decremented. To increment the counter represented by the skewed clock x from time i to time $i+1$, start an accurate clock z with x in the interval $[i-1, i]$ and reset z when it reaches 1; then nondeterministically reset x in the interval $[i, i+1]$ and test $x = z$ at

time $i+1$. To decrement the counter represented by the skewed clock x from time i to time $i+1$, nondeterministically start an accurate clock z in the interval $[i-1, i]$ and test $x = z$ at time i; when z reaches 1 in the interval $[i, i+1]$, reset x. Given an NCM M, we can so construct a 2-rate timed system that has a Muller accepting trajectory iff M halts. (Indeed, using acceptance conditions, we can construct a 2-rate automaton that has a Muller accepting trajectory iff a counter is 0 infinitely often along some trajectories of M; this shows that the emptiness problem is Σ_1^1-complete for 2-rate automata [HPS83].)

The second part of the theorem follows from an undecidability result for timed systems with memory cells [Cer92]. ■

We point out that the Muller emptiness problem is decidable for simple n-rate automata. This is because any simple n-rate automaton can be transformed into a timed automaton by factoring and scaling the rate of the skewed clocks into the same clock rate. An analogous result holds for real-time temporal logics [WME92]. More recent decidability and undecidability results can be found in [AD94,KPSY93,BER94,MV94,PV94,BR95,HKPV95].

2.2 Property Modeling Language: Integrator Logic

The formulas of the *Integrator Computation Tree Logic* ICTL for a given hybrid automaton A contain two kinds of variables—data and control variables of A, and integrators. An integrator is a stop watch that can be stopped and restarted. We adopt the notation of [BES93] to generalize the clock reset ("freeze") quantifier of TPTL [AH94] to a reset quantifier for integrators. While the clock reset quantifier $z.\varphi$ introduces (binds) the clock z and sets its value at 0, the integrator reset quantifier $(z : U).\varphi$ introduces (binds) the integrator z, declares its type to be U,

and sets its value to 0. The *type* $U \subseteq V$ of z is a set of locations from A. The value of an integrator of type U increases with the rate at which time advances whenever the automaton control is in a location in U, and its value stays unchanged whenever the automaton control is in a location in $V \backslash U$. In particular, a clock variable is an integrator of type V.

2.2.1 Syntax

The formulas of ICTL are built from *integral* state predicates using boolean operators, the two temporal operators $\exists\mathcal{U}$ ("possibly") and $\forall\mathcal{U}$ ("inevitably"), and the reset quantifier for integrators. Intuitively, the formula $\varphi_1 \exists\mathcal{U} \varphi_2$ holds in the automaton state σ if along *some* automaton trajectory that starts from σ, the second argument φ_2 holds in some state of the trajectory, and the first argument φ_1 holds in all intermediate states. The formula $\varphi_1 \forall\mathcal{U} \varphi_2$ asserts that along *every* trajectory that starts from σ, the first argument φ_1 is true until the second argument φ_2 becomes true.

Let A be a hybrid automaton with the data variables $\vec{x}$ and the control locations V, and let $\vec{z}$ be a vector of real-valued variables called *integrators*. A *$\vec{z}$-extended data predicate* of A is a formula over $\vec{x} \uplus \vec{z}$. A *$\vec{z}$-extended state predicate* of A is a collection of $\vec{z}$-extended data predicates, one for each location in V. The *A-formulas* of ICTL are defined inductively by the grammar

$$\varphi \ ::= \ \phi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \exists\mathcal{U} \varphi_2 \mid \varphi_1 \forall\mathcal{U} \varphi_2 \mid (z : U).\, \varphi$$

where ϕ is an integral $\vec{z}$-extended state predicate, $U \subseteq V$ is a set of locations, and z is an integrator from $\vec{z}$. The ICTL-formula φ is *closed* if every occurrence of an integrator in φ is bound by a reset quantifier. We restrict ourselves to closed formulas of ICTL. We also assume that different reset quantifiers in φ bind different integrators, which can be achieved by renaming bound variables.

We also use *location predicates* to denote the type of an integrator. A location predicate is a boolean combination of equalities of the form $\ell = v$, where ℓ is the location counter. For the railroad gate controller example, $\ell = (far, open, idle)$ denotes the location set $\{(far, open, idle)\}$, and $\ell = true$ denotes the whole location set U. We also write $\ell[i]$ for the i-th component of the program counter ℓ, so for the railroad gate controller example, $\ell[1]$ ranges over the locations of the train automaton, etc. Thus $\ell[2] = closed$ denotes the set of all the locations whose second component are *closed*.

If all integrators in φ have the type U, then φ is a formula of TCTL [ACD93]. An ICTL-formula that does not contain any integrator is a CTL-formula [CES86]. When writing ICTL-formulas, we suppress the integrator type U, and we use boolean combinations of location constraints for defining integrator types. Typical abbreviations for ICTL-formulas include the standard temporal operators $\forall\Diamond\varphi$, $\exists\Box\varphi$, and $\varphi_1\exists\mathcal{W}\varphi_2$, for $true\forall\mathcal{U}\varphi$, $\neg\forall\Diamond\neg\varphi$, and $\varphi_1\exists\mathcal{U}\varphi_2 \vee \exists\Box\varphi_1$, respectively. We also use time-bounded temporal operators [ACD93] such as $\forall\Diamond_{\leq 5}\varphi$, which stands for the ICTL-formula $z.\forall\Diamond(z \leq 5 \wedge \varphi)$, that is, $(z : U).(true\forall\mathcal{U}(z \leq 5 \wedge \varphi))$.

2.2.2 Semantics

Every closed A-formula ψ of ICTL defines a region $[\![\psi]\!]_A$ of the automaton A. The region $[\![\psi]\!]_A$ is called the *characteristic A-region* of ψ, and is defined in three steps. First, we extend A to an automaton $A_{\vec{z}}$ with data variables $\vec{x} \uplus \vec{z}$. Second, we interpret ψ over the states of the extended automaton $A_{\vec{z}}$. Third, we relate the states of $A_{\vec{z}}$ to the states of A.

Let $\vec{z} = (z_1, \ldots, z_m)$ be the vector of integrators that occur in the formula ψ, and let $U_1, \ldots, U_n$ be the corresponding types (as specified by ψ). From the n-dimensional hybrid automaton $A = (\vec{x}, V, inv, dif, E, act, L, syn)$ we construct the $(n + m)$-dimensional *$\vec{z}$-extension* of A as the hybrid automaton $A_{\vec{z}} = (\vec{x} \uplus$

$\vec{z}, V, inv, dif', E, act', L, syn$):

- For each integrator z_i in $\vec{z}$, and each location $v \in V$, if $v \in U_i$ then $dif'(v) = dif(v) \wedge \dot{z}_i = 1$; if $v \notin U_i$ then $dif'(v) = dif(v) \wedge \dot{z}_i = 0$.
- For each integrator z_i in $\vec{z}$, and each transition $e \in E$, $act'(e) = act(e) \wedge z_i' = z_i$.

Each data state $\vec{s} \in \mathbb{R}^{n+m}$ of $A_{\vec{z}}$ consists of an $\vec{x}$-*projection* $\vec{s}|_{\vec{x}} \in \mathbb{R}^n$, which is a data state of A, and a $\vec{z}$-*projection* $\vec{s}|_{\vec{z}} \in \mathbb{R}^m$, which assigns to each integrator in $\vec{z}$ a real value. Each state $\sigma = (v, \vec{s})$ of $A_{\vec{z}}$, then, consists of a state $\sigma|_{\vec{x}} = (v, \vec{s}|_{\vec{x}})$ of A, and an integrator valuation $\sigma|_{\vec{z}} = \vec{s}|_{\vec{z}}$. In particular, $\Sigma_{A_{\vec{z}}} = \Sigma_A \times \mathbb{R}^m$.

The projection operation $|_{\vec{x}}$ is extended to regions and trajectories in the natural way: the $\vec{x}$-projection of the $A_{\vec{z}}$-region R is the A-region that contains the $\vec{x}$-projections of all states in R; the $\vec{x}$-projection of the data trajectory (δ, ρ) of $A_{\vec{z}}$ is the data trajectory of A that maps every real $t \in [0, \delta]$ to the data state $\rho(t)|_{\vec{x}}$; the $\vec{x}$-projection of the $A_{\vec{z}}$-trajectory τ is the A-trajectory that results from τ by replacing all data trajectories with their $\vec{x}$-projections. Notice that each trajectory τ of $A_{\vec{z}}$ is completely determined by the trajectory $\tau|_{\vec{x}}$ of A, and the initial integrator valuation $\tau(0,0)|_{\vec{z}} \in \mathbb{R}^m$.

Given a set $\mathcal{T}$ of A-trajectories, the $\vec{z}$-*extension* $\mathcal{T}_{\vec{z}}$ consists of all $A_{\vec{z}}$-trajectories whose $\vec{x}$-projections are in $\mathcal{T}$. For a state σ of $A_{\vec{z}}$, the satisfaction relation $\sigma \models_{\mathcal{T}} \psi$ is defined inductively on the subformulas of ψ:

$\sigma \models_{\mathcal{T}} \phi$ iff $\sigma \in [\![\phi]\!]$;

$\sigma \models_{\mathcal{T}} \neg\varphi$ iff $\sigma \not\models_{\mathcal{T}} \varphi$;

$\sigma \models_{\mathcal{T}} \varphi_1 \vee \varphi_2$ iff $\sigma \models_{\mathcal{T}} \varphi_1$ or $\sigma \models_{\mathcal{T}} \varphi_2$;

- $\sigma \models_{\mathcal{T}} \varphi_1 \exists\mathcal{U} \varphi_2$ iff for some trajectory $\tau \in \mathcal{T}_{\vec{z}}$ with $\tau(0,0) = \sigma$, there is a position π of τ such that $\tau(\pi) \models_{\mathcal{T}} \varphi_2$, and for all positions π' of τ, if $\pi' \leq \pi$ then $\tau(\pi') \models_{\mathcal{T}} \varphi_1 \vee \varphi_2$;

- $\sigma \models_{\mathcal{T}} \varphi_1 \forall\mathcal{U} \varphi_2$ iff for all trajectories $\tau \in \mathcal{T}_{\vec{z}}$ with $\tau(0,0) = \sigma$, there is a position π of τ such that $\tau(\pi) \models_{\mathcal{T}} \varphi_2$, and for all positions π' of τ, if $\pi' \leq \pi$ then $\tau(\pi') \models_{\mathcal{T}} \varphi_1 \vee \varphi_2$;

- $\sigma \models_{\mathcal{T}} (z : p).\, \varphi$ iff $\sigma[z := 0] \models_{\mathcal{T}} \varphi$, where $\sigma[z := 0]$ is the state that differs from σ at most in the value of z, which is 0.

The disjunctions in the definitions of the temporal operators $\exists\mathcal{U}$ and $\forall\mathcal{U}$ account for the possibility that the second argument φ_2 may hold throughout a left-open interval of a trajectory [HNSY94].

We write $[\psi]_{\mathcal{T}}$ for the $A_{\vec{z}}$-region of all states σ such that $\sigma \models_{\mathcal{T}} \psi$. Since ψ is closed, if $\sigma \models_{\mathcal{T}} \psi$ and $\sigma|_{\vec{x}} = \sigma'|_{\vec{x}}$, then $\sigma' \models_{\mathcal{T}} \psi$; that is, $[\psi]_{\mathcal{T}} = ([\psi]_{\mathcal{T}})|_{\vec{x}} \times \mathbb{R}^m$. The *characteristic A-region* $[\![\psi]\!]_A$ is defined to be the $\vec{x}$-projection of the $A_{\vec{z}}$-region $[\psi]_{[\![A]\!]^{div}}$. Recall that $[\![A]\!]^{div}$ is the set of divergent trajectories of A. The state σ of the automaton A *satisfies* the formula ψ if $\sigma \in [\![\psi]\!]_A$.

2.2.3 Example: Railroad Gate Controller

The hybrid automaton A *meets* the requirement specified by the A-formula ψ of ICTL iff all admissible states of A satisfy ψ; that is, $[\![\psi]\!]_A = \Sigma_A$.

To illustrate the use of ICTL as a specification language, recall the railroad gate controller from Section 2. The initial condition of the system is given by the state predicate

$$\phi_0 : \quad \ell = (far, open, idle),$$

which asserts that the train is far from the gate, which is open, and the controller

is idle. We require the following properties of the controller. The *safety property*

$$\phi_0 \;\rightarrow\; \forall\Box(x \leq 10 \;\rightarrow\; \ell[2] = \mathit{closed})$$

asserts that whenever a train is within 10 meters of the gate, the gate must be closed. Since the safety requirement is met by a controller that keeps the gate closed forever, we add the *liveness* (*response*) *property*

$$\phi_0 \;\rightarrow\; \forall\Box\,\forall\Diamond(\ell[2] = \mathit{open})$$

that the gate will always open again. Indeed, this eventual liveness requirement may not be satisfactory (imagine you are in a car waiting to cross at a closed gate!), so we may wish to require instead the stronger *time-bounded response property*

$$\mathit{init} \;\rightarrow\; \forall\Box\,\forall\Diamond_{\leq 33}\,(\ell[2] = \mathit{open})$$

that the gate will always open within 33 seconds.

To demonstrate the use of integrators, we consider the additional requirement that within any time interval longer than an hour, the gate must be open at least 80% of the time. This *duration* (*utility*) *property* can be expressed in ICTL by the formula

$$\phi_0 \;\rightarrow\; \forall\Box(z_1\colon \mathit{true}).(z_2\colon \ell[2] = \mathit{open}).\forall\Box(z_1 \geq 3600 \;\rightarrow\; 10z_2 \leq 8z_1).$$

Here z_1 is a clock that measures the length of a time interval, and z_2 is an integrator that measures the accumulated time that the gate is open during the interval measured by z_1.

Chapter 3

Symbolic Modeling Checking of Linear Hybrid Systems

You should not put too much trust in any unproved conjecture, even if it has been propounded by a great authority, even if it has been propounded by yourself. You should try to prove it or disprove it ...

— George Polya

We present the symbolic model-checking procedure that verifies if a linear hybrid automaton satisfies an ICTL formula. The correctness of the symbolic model-checking procedure is proved.

3.1 Time and Transition Steps

The state of a hybrid automaton A can change to another state by either a *time step* that reflects a time delay that changes the value of data variables according to the rate predicate of the current control location, or a *transition step* that reflects a transition of A that changes both the control location and the value

of data variables according to the action predicate of the automaton transition. The hybrid automaton A thus also determines an infinite-state transition system S where the state set of S is Σ_A and the transition set of S consists of all the time steps and transition steps of A.

In this section, we formally defines the time steps and the transition steps and then we show how to compute the weakest preconditions of reaching a region of A with respect to a time step or a transition step. The computation of the weakest preconditions forms the basis of the symbolic model-checking procedure.

3.1.1 Step Relations

We now define the time-step and transition-step relations. Let $Q = \bigcup_v (v, Q_v)$ be a region of A.

Time step For all states $\sigma_1 = (v_1, \vec{s}_1)$ and $\sigma_2 = (v_2, \vec{s}_2)$ of A, $\sigma_1 \xrightarrow{Q} \sigma_2$ if $v = v_1 = v_2$, and there exists an admissible data trajectory (δ, ρ) in v such that

(1) $\rho(0) = \vec{s}_1$ and $\rho(\delta) = \vec{s}_2$;
(2) for all reals $t \in [0, \delta]$, $\rho_i(t) \in [\![inv(v)]\!] \cap Q_v$.

In other words, $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$ iff in location v, starting from the data state $\vec{s}_1$, it is possible to reach the data state $\vec{s}_2$ by letting time pass, without leaving the region Q. In this case, we call (δ, ρ) the *witness* for the time step $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$. We define $\sigma_1 \xrightarrow{Q}_{lin} \sigma_2$ if there is a piecewise-linear witness for $\sigma_1 \xrightarrow{Q} \sigma_2$; and $\sigma_1 \xrightarrow{Q}_1 \sigma_2$, if there is a linear witness for $\sigma_1 \xrightarrow{Q} \sigma_2$. Clearly, $\xrightarrow{Q}_1 \subseteq \xrightarrow{Q}_{lin} \subseteq \xrightarrow{Q}$. For simplicity, we write $\rightarrow$ for $\xrightarrow{\Sigma_A}$.

Transition step For all states σ_1 and σ_2 of A, $\sigma_1 \overset{Q}{\mapsto} \sigma_2$ if $\sigma_1, \sigma_2 \in \Sigma_A \cap Q$, and there exists a transition $e \in E$ such that $\sigma_2 \in [\![act(e)]\!](\sigma_1)$.

The binary relation $\stackrel{Q}{\Rightarrow}$ on the states of A is *Q-reflexive* if (1) $\sigma_1 \stackrel{Q}{\Rightarrow} \sigma_2$ implies $\sigma_1, \sigma_2 \in \Sigma_A \cap Q$, and (2) for all $\sigma \in \Sigma_A \cap Q$, we have $\sigma \stackrel{Q}{\Rightarrow} \sigma$. The time-step relation $\stackrel{Q}{\rightarrow}$ is Q-reflexive because of witness trajectories with duration 0, and the transition-step relation $\stackrel{Q}{\mapsto}$ is Q-reflexive because of stutter transitions.

We now show that for linear regions Q the time-step relation $\stackrel{Q}{\rightarrow}$ and the piecewise-linear time-step relation $\stackrel{Q}{\rightarrow}_{lin}$ coincide. For this purpose, we need a lemma and a few definitions.

Lemma 1 *Let A be a linear hybrid automaton, let v be a location of A, and let S be a convex data region contained in $[\![inv(v)]\!]$. If there is an admissible data trajectory (ρ, δ) of A such that $\rho(0) = \vec{s}_1 \in S$ and $\rho(\delta) = \vec{s}_2 \in S$, then there is a admissible linear data trajectory (ρ', δ) of A such that $\rho'(0) = \vec{s}_1$, $\rho'(\delta) = \vec{s}_2$, and $\rho'(t) \in S$ for all $t \in [0, \delta]$.*

Proof. Suppose that $\rho(0) = \vec{s}_1$ and $\rho(\delta) = \vec{s}_2$. We define a continuous function $\rho' : [0, \delta] \rightarrow \mathbb{R}^n$ such that $\rho'(t) = \vec{s}_1 + t \cdot (\frac{\vec{s}_2 - \vec{s}_1}{\delta})$. Then $\rho'(0) = \vec{s}_1$, $\rho'(\delta) = \vec{s}_2$, and $\frac{d\rho'(\vec{x})(t)}{dt} = \frac{\vec{s}_2 - \vec{s}_1}{\delta}$ for all $t \in (0, \delta)$.

Suppose that the rate predicate $dif(v)$ has the form $\bigwedge_{i=1,\ldots,k} r_i$. Let $r_i = (c_0 \sim \vec{c} \cdot \vec{x})$ be a conjunct of $dif(v)$, where $\sim \in \{<, \leq\}$ and $\vec{c} \cdot \vec{x}$ is the inner product of a constant vector $\vec{c}$ and vector $\vec{x}$. Since (δ, ρ) is an admissible data trajectory, for all time instants $t \in [t_i, t_{i+1}]$,

$$c_0 \sim \vec{c} \cdot \frac{d\rho(\vec{x})(t)}{dt}.$$

Integrating both sides of the above inequality from 0 to δ, we get $c_0(\delta) \sim \vec{c} \cdot (\vec{s}_2 - \vec{s}_1)$; that is,

$$c_0 \sim \vec{c} \cdot \frac{\vec{s}_2 - \vec{s}_1}{\delta}.$$

Since $\frac{d\rho'(\vec{x})(t)}{dt} = \frac{\vec{s}_2 - \vec{s}_1}{\delta}$ for all $t \in (0, \delta)$, we know that $\frac{d\rho'(\vec{x})(t)}{dt} \in [\![r_i]\!]$, for all $t \in (0, \delta)$. Moreover, since r_i is an arbitrary conjunct of $dif(v)$, we can conclude

that $\frac{d\rho'(\vec{x})(t)}{dt} \in [\![dif(v)]\!]$ for all $t \in (0, \delta)$.

In addition, ρ' is linear, and both of its endpoints, $\vec{s}_1$ and $\vec{s}_2$, are in some convex data region S, so $\rho'(t) \in S$ for all $t \in [0, \delta]$. This proves our claim that (δ, ρ') is an admissible linear data trajectory such that $\rho'(0) = \rho(0)$, $\rho'(\delta) = \rho(\delta)$ and $\rho'(t) \in S$ for all $t \in [0, \delta]$. ■

We now extend the result to non-convex data regions. Let p be a data predicate. A *closed convex covering* of p is a set $\{p_1, \ldots, p_k\}$ of closed convex data predicates p_i such that $[\![p]\!] \subseteq \bigcup_{1 \leq i \leq k} [\![p_i]\!]$. A closed convex covering of p can be easily constructed from the disjunctive normal form of p. Assume that $p = p_1 \vee \ldots \vee p_k$ is in disjunctive normal form, with each disjunct p_i of the form $\bigwedge_j (e_j \sim c_j)$, where e_j is a linear term and c_j is an integer constant. We define the data predicate transformer *close* such that the data predicate $close(p)$ results from the data predicate p by replacing each strict inequality $e_j > c_j$ or $e_j < c_j$ by the corresponding nonstrict inequality $e_j \geq c_j$ or $e_j \leq c_j$, respectively. Then $\{close(p_1), \ldots, close(p_k)\}$ is a closed convex covering of p.

If $\{p_1, \ldots, p_k\}$ is a closed convex covering of the data predicate p, then the set $C = \{p_1 \wedge p, \ldots, p_k \wedge p\}$ of convex data predicates is an *exact convex covering* of p; that is, $[\![p]\!] = \bigcup_{1 \leq i \leq k} [\![p_i \wedge p]\!]$. We call each convex data region $[\![p_i \wedge p]\!]$ a *patch* of C.

Theorem 2 *Let A be a linear hybrid automaton, let $Q = \bigcup_v (v, Q_v)$ be a region of A, let v be a location of A, and let $\vec{s}_1$ and $\vec{s}_2$ be two data states of A. If Q_v is linear, then $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$ iff $(v, \vec{s}_1) \xrightarrow{Q}_{lin} (v, \vec{s}_2)$. If Q_v is convex linear, then $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$ iff $(v, \vec{s}_1) \xrightarrow{Q}_1 (v, \vec{s}_2)$.*

Proof. It is clear that $(v, \vec{s}_1) \xrightarrow{Q}_{lin} (v, \vec{s}_2)$ implies $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$. We show that $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$ implies $(v, \vec{s}_1) \xrightarrow{Q}_{lin} (v, \vec{s}_2)$. We say a data trajectory is *trivial* if its duration is 0. If $\vec{s}_1 = \vec{s}_2$, then a witness of $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$ is trivial, and thus $(v, \vec{s}_1) \xrightarrow{Q}_{lin} (v, \vec{s}_2)$. So assume that $\vec{s}_1 \neq \vec{s}_2$, and therefore, every witness for

$(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$ is not trivial. Since Q_v is linear, there is a data predicate p_v such that $[\![p_v]\!] = Q_v$. Let $C = \{p_1, \ldots, p_k\}$ be an exact convex covering of $p_v \wedge inv(v)$. We define a *finite crossing sequence* of a nontrivial data trajectory (δ, ρ) on C to be a time sequence $(0 = t_0, t_1, t_2, \ldots, t_m = \delta)$ with $t_0 < t_1 < t_2 < \cdots < t_m$ such that (1) for each $1 \leq i \leq m-1$, $\rho(t_i)$ is in two distinct patches of C, and (2) for each $0 \leq i \leq m-1$, both $\rho(t_i)$ and $\rho(t_{i+1})$ are in the same patch of C. Notice that although the two endpoints $\rho(t_i)$ and $\rho(t_{i+1})$ of the data trajectory $\rho[t_i, t_{i+1}]$ are in the same patch, say $[\![p_j]\!]$, of C, the interior points of $\rho[t_i, t_{i+1}]$ may not all be in $[\![p_j]\!]$. Since C is finite, every nontrivial data trajectory (δ, ρ) must have a finite crossing sequence.

Consider a finite crossing sequence $(t_0 = 0, t_1, \ldots, t_m = \delta)$ of a witness (δ, ρ) for $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$ on C. For each $i \in \{0, \ldots, m-1\}$, $\rho[t_i, t_{i+1}]$ is a data trajectory such that both data state $\rho(t_i)$ and $\rho(t_{i+1})$ are in the convex data region $[\![p_j \wedge inv(v)]\!]$, where $[\![p_j]\!]$ is a patch of C. By Lemma 1, there is a linear data trajectory $(t_{i+1} - t_i, \rho_i)$ such that $\rho_i(0) = \rho(t_i)$, $\rho_i(t_{i+1} - t_i) = \rho(t_{i+1})$ and $\rho_i(t) \in [\![p_j \wedge inv(v)]\!]$ for all $t \in [0, t_{i+1} - t_i]$. So the concatenation of these linear data trajectories $(t_1, \rho_0), (t_2 - t_1, \rho_1), \ldots, (\delta - t_{m-1}, \rho_{m-1})$ is a piecewise-linear witness for $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$, and thus $(v, \vec{s}_1) \xrightarrow{Q}_{lin} (v, \vec{s}_2)$.

On the other hand, if Q_v is convex, then $C = \{p_v \wedge inv(v)\}$ is an exact convex covering of $p_v \wedge inv(v)$. Then it follows immediately from Lemma 1 that $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$ if and only if $(v, \vec{s}_1) \xrightarrow{Q}_1 (v, \vec{s}_2)$. ■

3.1.2 Precondition Operators

Let Q and R be two regions of the hybrid automaton A, and let $\overset{Q}{\Rightarrow}$ be a Q-reflexive binary relation on the states of A. The $\overset{Q}{\Rightarrow}$*-precondition* $pre^Q_{\Rightarrow}(R)$ of R is the region of A from which a state in R can be reached in a single $\overset{Q}{\Rightarrow}$-step; that is, $\sigma \in pre^Q_{\Rightarrow}(R)$ if there is a state $\sigma' \in R$ such that $\sigma \overset{Q}{\Rightarrow} \sigma'$. Since $\overset{Q}{\Rightarrow}$ is Q-reflexive,

the precondition operator $pre_{\Rightarrow}$ is monotonic on the subregions of $\Sigma_A \cap Q$; that is, for all regions $R \subseteq \Sigma_A \cap Q$, we have $R \subseteq pre^Q_{\Rightarrow}(R) \subseteq \Sigma_A \cap Q$.

We will define the *time-precondition* operator $pre_{\rightarrow}$ and also the *transition-precondition* operator $pre_{\mapsto}$, and show that if both regions Q and R are linear, then so are the preconditions $pre^Q_{\rightarrow}(R)$ and $pre^Q_{\mapsto}(R)$. This is done by constructing from the state predicates ϕ and χ that define Q and R, respectively, two state predicates $pre^{\phi}_{\rightarrow}(\chi)$ and $pre^{\phi}_{\mapsto}(\chi)$ that define $pre^Q_{\rightarrow}(R)$ and $pre^Q_{\mapsto}(R)$, respectively. We then define the *A-precondition* $pre^Q_A(R)$ to be the union $pre^Q_{\rightarrow}(R) \cup pre^Q_{\mapsto}(R)$. If ϕ defines Q, and χ defines R, then $pre^Q_A(R)$ is defined by the state predicate

$$pre^{\phi}_A(\chi) \;=\; pre^{\phi}_{\rightarrow}(\chi) \vee pre^{\phi}_{\mapsto}(\chi).$$

In the following, suppose that $Q = \bigcup_v (v, Q_v)$ and $R = \bigcup_v (v, R_v)$. Let $\phi = \bigcup_v (v, q_v)$ be a state predicate such that for each location v of A, $[\![q_v]\!] = Q_v$, and let $\chi = \bigcup_v (v, r_v)$ be a state predicate such that for each location v of A, $[\![r_v]\!] = R_v$.

Time Precondition

We write $pre^{Q_v}_{\rightarrow}(v; R_v)$ for the data region such that from any state in the region $(v, pre^{Q_v}_{\rightarrow}(v; R_v))$ a state in the region (v, R_v) can be reached in a single $\xrightarrow{Q}$-step. We show that the data region $pre^{Q_v}_{\rightarrow}(v; R_v)$ is linear by constructing from q_v and r_v a data predicate $pre^{q_v}_{\rightarrow}(v; r_v)$ that defines the data region $pre^{Q_v}_{\rightarrow}(v; R_v)$. Then

$$pre^{\phi}_{\rightarrow}(\chi) \;=\; \bigcup_{v \in V} (v, pre^{q_v}_{\rightarrow}(v; r_v)).$$

The construction of $pre^{q_v}_{\rightarrow}(v; r_v)$ proceeds in two steps. First we construct a data predicate $pre^{true}_{\rightarrow}(v; r_v)$ such that $[\![pre^{true}_{\rightarrow}(v; r_v)]\!] = pre^{[\![inv(v)]\!]}_{\rightarrow}(v; R_v)$. Then we apply the precondition operator $pre^{true}_{\rightarrow}$ repeatedly to construct the data predicate $pre^{q_v}_{\rightarrow}(v; r_v)$.

Lemma 2 *Let A be a linear hybrid automaton, let v be a location of A, and let r_v be a data predicate of A. Define*

$$pre^{true}_{\rightarrow}(v; r_v) \;=\; inv(v) \wedge (\exists \delta \geq 0.\, \exists \vec{d}.\, dif(v)[\vec{x} := \vec{d}] \wedge (r_v \wedge inv(v))[\vec{x} := \vec{x} + \delta \cdot \vec{d}]).$$

Then $[\![pre^{true}_{\rightarrow}(v; r_v)]\!] = pre^{[\![inv(v)]\!]}_{\rightarrow}(v; [\![r_v]\!])$.

Proof. The quantified formula in $pre^{true}_{\rightarrow}(v; r_v)$ specifies that a data state $\vec{x}$ satisfies $pre^{true}_{\rightarrow}(v; r_v)$ iff the following four conditions hold:

1. $\vec{x}$ is in $[\![inv(v)]\!]$ (the leading conjunct $inv(v)$);
2. there exist a duration δ and a slope vector $\vec{d}$ that constitute a linear witness trajectory $(\delta, \rho(t) = \vec{x} + t \cdot \vec{d})$ for a single $\overset{[\![(v, inv(v))]\!]}{\rightarrow}_1$ step (the quantified variables δ and $\vec{d}$);
3. the slope vector $\vec{d}$ satisfies the rate predicate $dif(v)$ of location v (the conjunct $dif(v)[\vec{x} := \vec{d}]$); and
4. $\rho(\delta)$ is $[\![r]\!] \cap [\![inv(v)]\!]$ (the conjuncts $(r \wedge inv(v))[\vec{x} := \vec{x} + \delta \cdot \vec{d}]$).

Since $[\![inv(v)]\!]$ is convex, by Theorem 2, we know that the time-step relation $\overset{[\![(v, inv(v))]\!]}{\rightarrow}$ is equivalent to $\overset{[\![(v, inv(v))]\!]}{\rightarrow}_1$. Thus Condition 2 only consider linear witnesses. In addition, since the witness is linear, and since both $\rho(0)$ and $\rho(\delta)$ are in $[\![inv(v)]\!]$, Condition 1 together with Condition 4 implies that $\rho(t) \in [\![inv(v)]\!]$ for all $t \in [0, \delta]$. Now according to the definition of the time-step relation $\overset{[\![(v, inv(v))]\!]}{\rightarrow}_1$, it is clear that $[\![pre^{true}_{\rightarrow}(v; r_v)]\!] = pre^{[\![(v, inv(v))]\!]}_{\rightarrow}(v; R_v)$. ■

The formula $pre^{true}_{\rightarrow}(v; r_v)$ of Lemma 2 contains the vector $\delta \cdot \vec{d}$ of variable products, which gives rise to nonlinear terms. We therefore replace the vector $\delta \cdot \vec{d}$ of variable products by a vector $\vec{c}$ of new quantified variables. Let $\delta \cdot dif(v)$ be the rate predicate that results from multiplying each constant of the rate predicate $dif(v)$

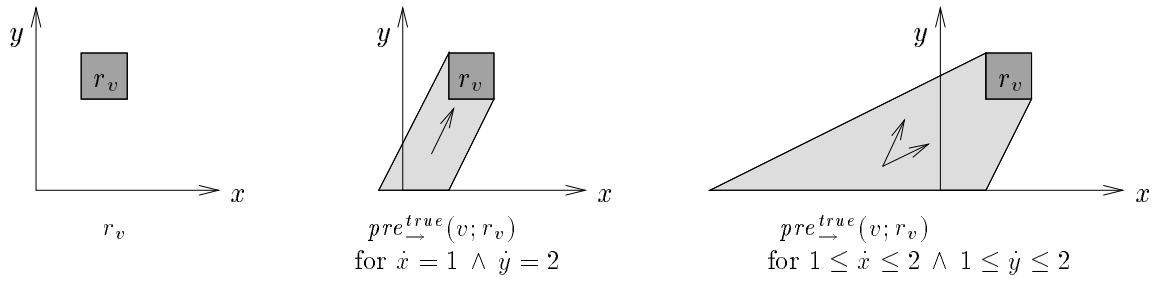

Figure 3.1: The time-precondition operator $pre^{true}_{\rightarrow}$

by the variable δ. For example, if r is the rate predicate $\dot{x}_1 \leq 3\dot{x}_2 + 6 \wedge \dot{x}_3 = 1$, then $\delta \cdot r$ is the rate predicate $\dot{x}_1 \leq 3\dot{x}_2 + 6\delta \wedge \dot{x}_3 = \delta$. Then the formula

$$inv(v) \wedge (\exists \delta \geq 0. \exists \vec{d}.\, dif(v)[\vec{\dot{x}} := \vec{d}] \wedge (r_v \wedge inv(v))[\vec{x} := \vec{x} + \delta \cdot \vec{d}])$$

is equivalent to the formula

$$inv(v) \wedge (\exists \delta \geq 0. \exists \vec{c}.\, (\delta \cdot dif(v))[\vec{\dot{x}} := \vec{c}] \wedge (r_v \wedge inv(v))[\vec{x} := \vec{x} + \vec{c}]).$$

The next proposition follows.

Proposition 1 *Let A be a linear hybrid automaton, let v be a location of A, and let r_v be a data predicate of A. Define*

$$pre^{true}_{\rightarrow}(v; r_v) \;=\; inv(v) \wedge (\exists \delta \geq 0. \exists \vec{c}.\, (\delta \cdot dif(v))[\vec{\dot{x}} := \vec{c}] \wedge (r_v \wedge inv(v))[\vec{x} := \vec{x} + \vec{c}]).$$

Then $[\![pre^{true}_{\rightarrow}(v; r_v)]\!] = pre^{[\![inv(v)]\!]}_{\rightarrow}(v; [\![r_v]\!])$.

The formula $pre^{true}_{\rightarrow}(v; r_v)$ is a formula of the first-order theory $(\mathbb{R}, \leq, +)$ of the reals with addition. Since this theory admits quantifier elimination, the formula $pre^{true}_{\rightarrow}(v; r_v)$ is equivalent to a data predicate. The quantifier-elimination procedure used by an earlier version of HyTech is discussed later in this chapter.

Example. Let us consider two simple examples of computing the data predicate $pre^{true}_{\rightarrow}(v; r_v)$ using Proposition 1. First, suppose that the linear hybrid automaton

A has two data variables, x and y, the invariant $inv(v) = (y \geq 0)$, and the activity $dif(v) = (\dot{x} = 1 \wedge \dot{y} = 2)$ for the location v. Consider the data predicate $r_v = (1 \leq x \leq 2 \wedge 2 \leq y \leq 3)$ (see the left of Figure 3.1). Then, according to Proposition 1,

$$pre_{\rightarrow}^{true}(v; r_v) \; =$$

$$(y \geq 0 \wedge (\exists \delta \geq 0. \exists c_1, c_2. c_1 = \delta \wedge c_2 = 2\delta \wedge 1 \leq x + c_1 \leq 2 \wedge 2 \leq y + c_2 \leq 3)).$$

Eliminating the two existential quantifiers inside out, we obtain

$$pre_{\rightarrow}^{true}(v; r_v) \; = \; (y \geq 0 \wedge (\exists \delta \geq 0. 1 \leq x + \delta \leq 2 \wedge 2 \leq y + 2\delta \leq 3))$$

and, finally, the data predicate

$$pre_{\rightarrow}^{true}(v; r_v) \; = \; (x \leq 2 \wedge 0 \leq y \leq 3 \wedge -2 \leq y - 2x \leq 1)$$

(see the center of Figure 3.1). Second, suppose that the activity $dif(v)$ of the location v is $1 \leq \dot{x} \leq 2 \wedge 1 \leq \dot{y} \leq 2$. Then, according to Proposition 1,

$$pre_{\rightarrow}^{true}(v; r_v) \; = \; (y \geq 0 \wedge$$

$$(\exists \delta \geq 0. \exists c_1, c_2. \delta \leq c_1 \leq 2\delta \wedge \delta \leq c_2 \leq 2\delta \wedge 1 \leq x + c_1 \leq 2 \wedge 2 \leq y + c_2 \leq 3)).$$

Eliminating the two existential quantifiers, we obtain

$$pre_{\rightarrow}^{true}(v; r_v) \; = \; (y \geq 0 \wedge (\exists \delta \geq 0. 1 - x \leq 2\delta \wedge 2 - y \leq 2\delta \wedge \delta \leq 2 - x \wedge \delta \leq 3 - y))$$

and the equivalent data predicate

$$pre_{\rightarrow}^{true}(v; r_v) \; = \; (x \leq 2 \wedge 0 \leq y \leq 3 \wedge 2x - y \leq 2 \wedge 2y - x \leq 5)$$

(see the right of Figure 3.1). ■

We show the reduction from the construction of the data predicate $pre^{q_v}_{\rightarrow}(v, r_v)$ to a sequence of applications of the precondition operator $pre^{true}_{\rightarrow}$ defined in Proposition 1. We proceed in three steps. First, let v' be a new, fictitious location with the invariant $inv(v') = (inv(v) \wedge q_v)$ and the activity $dif(v)$. Then

$$pre^{Q_v}_{\rightarrow}(v; R_v) \;=\; pre^{[\![inv(v')]\!]}_{\rightarrow}(v'; R_v).$$

The invariant $inv(v')$, however, may not be convex, in which case we cannot compute a data predicate $pre^{true}_{\rightarrow}(v'; r_v)$ using Proposition 1.

So, second, we split the location v' into several locations with convex invariants. Let $C = \{p_1, \ldots, p_k\}$ be an exact convex covering of the data predicate $inv(v')$. We split v' into the set $V' = \{v'_1, \ldots, v'_k\}$ of new, fictitious locations v'_i such that for all $1 \leq i \leq k$, the invariant $inv(v'_i)$ is p_i, and the activity $dif(v'_i)$ is $dif(v)$. Let

$$pre_{\rightarrow}(V'; R_v) \;=\; \bigcup_{v'_i \in V'} pre^{[\![inv(v'_i)]\!]}_{\rightarrow}(v'_i; R_v).$$

Then the data predicate $pre_{\rightarrow}(V'; r_v)$ that defines the data region $pre_{\rightarrow}(V'; R_v)$ can be constructed using Proposition 1 and disjunction.

A single $\stackrel{Q}{\rightarrow}$-step, however, may proceed from the data region $pre^{Q_v}_{\rightarrow}(v; R_v)$ to the data region R_v through more than one of the patches of C. Since there are k different patches of C, it will suffice to iterate the precondition $pre_{\rightarrow}(V'; R_v)$ k times. So, third, we define a sequence of k data regions, S_0 to S_k, such that $S_0 = S_v$ and for all $1 \leq j \leq k$, $S_j = pre_{\rightarrow}(V'; S_{j-1})$. Let $s_0, \ldots, s_k$ be the sequence of corresponding data predicates; that is, for all $0 \leq j \leq k$, $[\![s_j]\!] = S_j$. The next proposition shows that $S_k = pre^{[\![inv(v')]\!]}_{\rightarrow}(v'; R_v)$. Thus the data predicate $pre^{q_v}_{\rightarrow}(v; r_v) = s_k$ defines the data region $pre^{Q_v}_{\rightarrow}(v; R_v)$.

Proposition 2 *Let A be a hybrid automaton, let v be a location of A, and let q_v and r_v be two data predicates of A. Let $\{p_1, \ldots, p_k\}$ be an exact convex covering*

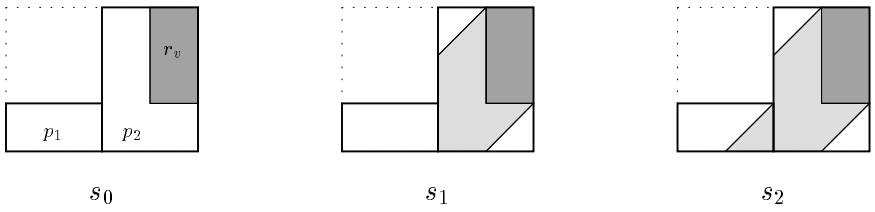

Figure 3.2: The time-precondition operator $pre^{q_v}_{\rightarrow}$

of the data predicate $inv(v) \wedge q_v$. *Let* $V' = \{v'_1, \ldots, v'_k\}$ *be a set of locations such that for all* $1 \leq i \leq k$, $inv(v'_i) = p_i$ *and* $dif(v'_i) = dif(v')$. *If* $S_0 = [\![r_v]\!]$ *and for all* $1 \leq j \leq k$, $S_j = pre_{\rightarrow}(V'; S_{j-1})$, *then* $S_k = pre^{[\![q_v]\!]}_{\rightarrow}(v; [\![r_v]\!])$.

Proof. Since $dif(v'_i) = dif(v')$ and $\bigvee_{v'_i} inv(v'_i) = inv(v')$, by definition, S_k defines the set of data states that can reach a data state in R_v by k $\rightarrow_1$ steps. According to the definition of $pre_{\rightarrow}(v'; R_v)$, $S_k \subseteq pre_{\rightarrow}(v'; R_v)$. On the other hand, since $inv(v')$ has an exact convex covering $\{p_1, \ldots, p_k\}$, the proof of Theorem 2 implies that the time-step relation $(v', \vec{s}) \rightarrow (v', \vec{s'})$ must have a piecewise-linear witness that consists of k linear data trajectories. In other words, $(v', \vec{s}) \rightarrow (v', \vec{s'})$ implies that for some $0 \leq j \leq k$, we have $(v', \vec{s}) \rightarrow_1{}^j (v', \vec{s'})$. Consequently, $pre_{\rightarrow}(v'; R_v) \subseteq S_k$. Thus $pre_{\rightarrow}(v'; R_v) = S_k$. ■

Example. Let us consider an example of computing the data predicate $pre^{q_v}_{\rightarrow}(v; r_v)$ using Proposition 2. Suppose that the linear hybrid automaton A has two data variables, x and y, the invariant $inv(v) = (0 \leq x \leq 4 \wedge 0 \leq y \leq 3)$ and the activity $dif(v) = (\dot{x} = 1 \wedge \dot{y} = 1)$ for the location v. Consider the data predicates

$$q_v \;=\; ((0 \leq x \leq 2 \wedge 0 \leq y \leq 1) \vee (2 \leq x \leq 4 \wedge 0 \leq y \leq 3))$$

and $r_v = (3 \leq x \leq 4 \wedge 2 \leq y \leq 3)$. The new location v' has the invariant

$$inv(v') \;=\; (0 \leq x \leq 2 \wedge 0 \leq y \leq 1) \vee (2 \leq x \leq 4 \wedge 0 \leq y \leq 3).$$

Since $inv(v')$ is not convex, and

$$\{p_1, p_2\} \;=\; \{0 \le x \le 2 \;\wedge\; 0 \le y \le 1,\; 2 \le x \le 4 \;\wedge\; 0 \le y \le 3\}$$

is an exact convex covering of $inv(v')$, we split the location v' into two locations $V' = \{v_1, v_2\}$ such that $inv(v_1) = p_1$ and $inv(v_2) = p_2$. Then

$$\begin{array}{rcll} s_0 & = & r_v & = \; (3 \le x \le 4 \;\wedge\; 2 \le y \le 3), \\ s_1 & = & pre_{\rightarrow}(V'; s_0) & = \; (2 \le x \le 4 \;\wedge\; 0 \le y \le 3 \;\wedge\; 1 \le x - y \le 3), \\ s_2 & = & pre_{\rightarrow}(V'; s_1) & = \; ((2 \le x \le 4 \;\wedge\; 0 \le y \le 3 \;\wedge\; 1 \le x - y \le 3) \;\vee \\ & & & \quad (1 \le x \le 2 \;\wedge\; 0 \le y \le 1 \;\wedge\; 1 \le x - y)) \end{array}$$

(see Figure 3.2). By Proposition 2, $pre^{q_v}_{\rightarrow}(v; r_v) = s_2$. ■

Transition Precondition

We write $pre^{Q}_{\mapsto}(v; R_v)$ for the region of states from which a state in (v, R_v) can be reached in a single $\overset{Q}{\mapsto}$-step. We show that the region $pre^{Q}_{\mapsto}(v; R_v)$ is linear by constructing from ϕ and r_v a state predicate $pre^{\phi}_{\mapsto}(v; r_v)$ that defines the region $pre^{Q}_{\mapsto}(v; R_v)$. Then

$$pre^{\phi}_{\mapsto}(\chi) \;=\; \bigcup_{v \in V} pre^{\phi}_{\mapsto}(v; r_v).$$

The next proposition constructs $pre^{\phi}_{\mapsto}(v; r_v)$ as a formula of the theory $(\mathbb{R}, \le, +)$, from which a data predicate can be obtained by quantifier elimination.

Proposition 3 *Let A be a linear hybrid automaton with the transition set E, let v be a location of A, let $\phi = \bigcup_v (v, q_v)$ be a state predicate of A, and let r_v be a data predicate of A. Define*

$$pre^{\phi}_{\mapsto}(v; r_v) = \bigcup_{(v', v) \in E} (v', q_{v'} \wedge inv(v') \wedge (\exists \vec{x}'.\, act(v', v) \wedge (r_v \wedge q_v \wedge inv(v))[\vec{x} := \vec{x}'])).$$

Then $[\![pre^{\phi}_{\mapsto}(v; r_v)]\!] = pre^{[\![\phi]\!]}_{\mapsto}(v; [\![r_v]\!])$.

Proof. For each location v', the quantified formula in $pre^{\phi}_{\mapsto}(v; r_v)$ specifies that a state $(v', \vec{x})$ satisfies $pre^{\phi}_{\mapsto}(v; r_v)$ iff the following three conditions hold:

1. the data state $\vec{x}$ is in $[\![q_{v'} \wedge inv(v')]\!]$ (the conjuncts $q_{v'} \wedge inv(v')$);
2. there is a state $(v, \vec{x'})$ that is reachable from state $(v', \vec{x})$ by taking a single $\mapsto$ step through transition (v', v) (the conjunct $act(v', v)$); and
3. $\vec{x'}$ is in $[\![r_v \wedge q_v \wedge inv(v)]\!]$ (the conjuncts $(r_v \wedge q_v \wedge inv(v))[\vec{x} := \vec{x'}]$).

According to the definition of the transition-step relation $\overset{Q}{\mapsto}$, the proposition holds. ■

Example. We show an example of computing the state predicate $pre^{\phi}_{\mapsto}(v; r_v)$ using Proposition 3. Suppose that the linear hybrid automaton A has two data variables, x and y, and only one non-stutter transition, $e = (v', v)$, with the target location v. Suppose that $act(e) = (x \leq 3 \wedge x' \geq 5 \wedge y' = x)$, and that $q_v = q_{v'} = inv(v) = inv(v') = true$. Consider the data predicate $r_v = (x \geq 6 \wedge y \leq 2)$. Then, according to Proposition 3,

$$\begin{aligned} pre^{\phi}_{\mapsto}(v; r_v) \;=\; & (v',\ (\exists x'.\, \exists y'.\, x \leq 3 \wedge x' \geq 5 \wedge y' = x \wedge x' \geq 6 \wedge y' \leq 2)) \wedge \\ & (v,\ (\exists x'.\, \exists y'.\, x' = x \wedge y' = y \wedge x' \geq 6 \wedge y' \leq 2)) \end{aligned}$$

(the first conjunct corresponds to the transition e, and the second conjunct corresponds to the stutter transition of v). Eliminating the two existential quantifiers inside out, we obtain

$$pre^{\phi}_{\mapsto}(v; r_v) \;=\; (v',\ (\exists x'.\, x \leq 3 \wedge x' \geq 5 \wedge x' \geq 6 \wedge x \leq 2)) \wedge$$

$$(v,\ (\exists x'.\, x' = x \wedge x' \geq 6 \wedge y \leq 2))$$

and, finally, the state predicate

$$pre^{\phi}_{\mapsto}(v; r_v) \;=\; (v',\ x \leq 2) \wedge (v,\ x \geq 6 \wedge y \leq 2). \blacksquare$$

The concluding theorem follows from Propositions 1, 2, and 3.

Theorem 3 *Let A be a linear hybrid automaton, and let ϕ and χ be two state predicates of A. Then $[\![pre_A^{\phi}(\chi)]\!] = pre_A^{[\![\phi]\!]}([\![\chi]\!])$.*

3.2 Symbolic Model Checking

Given a nonzeno linear hybrid automaton A, and a closed A-formula ψ of ICTL, the *model-checking problem* (A, ψ) asks to compute the characteristic A-region $[\![\psi]\!]_A$, by providing a state predicate ϕ that defines the answer $[\![\psi]\!]_A$; that is, $[\![\phi]\!] = [\![\psi]\!]_A$. The state predicate ϕ is called a *characteristic predicate* of (A, ψ). In general, a characteristic predicate may not exist, and it is undecidable if a given state predicate is a characteristic predicate of (A, ψ) [ACHH93,KPSY93]. In [HNSY94], a symbolic model-checking algorithm, SMC, is presented for computing a characteristic predicate of (A, ψ) in the case of a timed automaton A and a TCTL-formula ψ. We extend the SMC-algorithm and obtain a semi-decision procedure that, provided it terminates, returns a characteristic predicate of (A, ψ) in the general case.

3.2.1 The SMC-procedure

The SMC-procedure approximates a characteristic predicate of (A, ψ) by a sequence of state predicates. If the successive-approximation sequence converges in a finite number of steps, then the SMC-procedure terminates and returns a characteristic predicate of (A, ψ). The successive-approximation sequence, however, may diverge, in which case the SMC-procedure does not terminate.

Let $\vec{z}$ be the vector of integrators that occur in the formula ψ, together with a new clock $z_{\varphi_1 \forall \mathcal{U} \varphi_2}$ for each subformula $\varphi_1 \forall \mathcal{U} \varphi_2$ of ψ (i.e., z is different from the data variables of A and the integrators of ψ). The SMC-procedure uses the hybrid automaton $A_{\vec{z}}$, which extends A with the integrators from $\vec{z}$ (see Section 2.2.2),

and the precondition operator $pre_{A_{\vec{z}}}$ on state predicates, which was introduced and computed in Section 3.1.2.

Procedure SMC**:**

Input: a nonzeno linear hybrid automaton A;
a closed A-formula ψ of ICTL.

Output: a characteristic predicate $|\psi|$ of (A, ψ).

Recall that $\phi_A = \bigcup_{v \in V}(v, inv(v))$. The characteristic predicate $|\psi|$ is computed inductively on the subformulas of ψ:

$|\phi| \;:=\; \phi \wedge \phi_A$;

$|\neg\varphi| \;:=\; \neg|\varphi|$;

$|\varphi_1 \vee \varphi_2| \;:=\; |\varphi_1| \vee |\varphi_2|$;

$|\varphi_1 \exists\mathcal{U} \varphi_2| \;:=\; \bigvee_{i \geq 0} \chi_i$, where
$\chi_0 \;:=\; |\varphi_2|$ and
$\chi_{i+1} \;:=\; \chi_i \vee pre_{A_{\vec{z}}}^{|\varphi_1 \vee \varphi_2|}(\chi_i)$
(i.e., compute the sequence $\chi_0, \chi_1, \chi_2, \ldots$ of state predicates until the state predicate $\chi_i \approx \chi_{i+1}$ is valid[1]);

$|\varphi_1 \forall\mathcal{U} \varphi_2| \;:=\; \bigvee_{i \geq 0} \chi_i$, where
$\chi_0 \;:=\; |\varphi_2|$ and
$\chi_{i+1} \;:=\; |\chi_i \vee \neg z_{\varphi_1 \forall\mathcal{U} \varphi_2}.((\neg\chi_i)\exists\mathcal{U}(\neg(\varphi_1 \vee \chi_i) \vee z_{\varphi_1 \forall\mathcal{U} \varphi_2} > 1^2))|$;

$|(z\colon U).\,\varphi| \;:=\; |\varphi|[z := 0]$ (i.e., replace all occurrences of z in $|\varphi|$ by 0). ■

The SMC-procedure operates on the $\vec{z}$-extended automaton $A_{\vec{z}}$, and all intermediate results are $\vec{z}$-extended state predicates. First, observe that if the given au-

[1] As the state predicates are quantifier-free formulas of the theory $(\mathbb{R}, \leq, +)$, validity can be decided.

[2] Or any positive integer. This choice may effect the number of iterations.

tomaton A is nonzeno, then so is the $\vec{z}$-extension $A_{\vec{z}}$, and that $\phi_{A_{\vec{z}}} = \phi_A$. Second, observe that a characteristic predicate of $(A_{\vec{z}}, \psi)$ can always be simplified to a characteristic predicate of (A, ψ), because $|\psi|$ does not constrain the integrators from $\vec{z}$.

3.2.2 Possibility

Let ϕ_1 and ϕ_2 be two $\vec{z}$-extended state predicates, and consider the region $R = \Sigma_{A_{\vec{z}}} \cap [\![\phi_1 \vee \phi_2]\!]$ of $A_{\vec{z}}$. The SMC-procedure computes the characteristic region $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_{A_{\vec{z}}}$ as the least solution X of the equation $f(X) = X$, where

$$f(X) \;=\; [\![\phi_2]\!] \cup pre_{A_{\vec{z}}}^{[\![\phi_1 \vee \phi_2]\!]}(X)$$

is a monotonic function on the subregions of R. This is justified by the following proposition.

Proposition 4 *Let B be a linear hybrid automaton, and let ϕ_1 and ϕ_2 be two state predicates of B. Then $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$ is the least solution of the equation $X = [\![\phi_2]\!] \cup pre_B^{[\![\phi_1 \vee \phi_2]\!]}(X)$.*

Proof. We define the function $f : \Sigma_B \to \Sigma_B$ such that $f(X) = [\![\phi_2]\!] \cup pre_B^{[\![\phi_1 \vee \phi_2]\!]}(X)$. We first prove that $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$ is a fixpoint of the function f; that is,

$$[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B = f([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B)$$

. Since the precondition operator $pre_B^{[\![\phi_1 \exists\mathcal{U} \phi_2]\!]}$ is monotonic on the subregions of Σ_B, $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B \subseteq pre_B^{[\![\phi_1 \vee \phi_2]\!]}([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B)$. Thus $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B \subseteq f([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B)$.

On the other hand, let σ be a state in $pre_B^{[\![\phi_1 \vee \phi_2]\!]}([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B)$. By definition, there is a state σ' in $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$ such that either (1) $\sigma \overset{[\![\phi_1 \vee \phi_2]\!]}{\longrightarrow} \sigma'$ or (2) $\sigma \overset{[\![\phi_1 \vee \phi_2]\!]}{\mapsto} \sigma'$. In either case, by the definition of trajectories of a linear hybrid automaton and the definition of $\phi_1 \exists\mathcal{U} \phi_2$, state σ is in $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$. Thus $pre_B^{[\![\phi_1 \vee \phi_2]\!]}([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B) \subseteq$

$[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$. Moreover, by the definition of $\phi_1 \exists\mathcal{U} \phi_2$, $[\![\phi_2]\!] \subseteq [\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$. Therefore, $f([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B) \subseteq [\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$.

Now we show that $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$ is the least fixpoint of f. Let Q be a fixpoint of f; we claim that $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B \subseteq Q$. Let σ be a state in $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$. Then there is a trajectory $\tau \in [\![B]\!]^{div}$ and a position (i_1, δ) of τ such that $\tau(0,0) = \sigma$, $\tau(i_1, \delta) \in [\![\phi_2]\!]$, and for all positions $(j, \epsilon) < (i_1, \delta)$, $\tau(j, \epsilon) \in [\![\phi_1 \vee \phi_2]\!]$. Since $[\![\phi_2]\!] \subseteq Q$, $\tau(i_1, \delta) \in Q$. By the definition of the precondition operator $pre_B^{[\![\phi_1 \vee \phi_2]\!]}$, we know that $\tau(i_1 - 1, 0) \in f(Q)$. Since Q is a fixpoint of f and thus $f(Q) = Q$, we conclude that $\tau(i_1 - 1, 0) \in Q$. Similarly, $\tau(i_1 - k, 0) \in Q$ for all $1 \leq k \leq i_1$. In particular, $\sigma = \tau(0,0) \in Q$, which implies that $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B \subseteq Q$. ■

The SMC-procedure computes the least solution of the equation $f(X) = X$ as the limit of the successive-approximation sequence $\emptyset$, $f(\emptyset)$, $f^2(\emptyset)$, $f^3(\emptyset)$, ... of regions, which, unlike in the case of timed automata, may not converge in any finite number of steps. The SMC-procedure, therefore, may not terminate.

3.2.3 Inevitability

The computation of the characteristic region for $\forall\mathcal{U}$-formulas is more involved, because the time-step and transition-step relations are reflexive [HNSY94]. We proceed in two steps. First we reduce inevitability $\forall\mathcal{U}$ (for nonzeno automata) to an iteration of time-bounded inevitability. Second, we reduce time-bounded inevitability (over divergent trajectories) to possibility $\exists\mathcal{U}$, which we know how to compute.

Reduction to Time-bounded Inevitability

Let B be a linear hybrid automaton, and let c be a nonnegative integer. We define the *time-bounded inevitability operator* $\forall\mathcal{U}_{\leq c}$ on regions such that for all state predicates ϕ_1 and ϕ_2, $[\![\phi_1]\!] \forall\mathcal{U}_{\leq c} [\![\phi_2]\!] = [\![\phi_1 \forall\mathcal{U}_{\leq c} \phi_2]\!]_B$. Given two regions Q

and R of B, the region $Q\forall\mathcal{U}_{\leq c}\,R$ contains all states σ such that on all divergent trajectories of B that start in σ, the region Q is not left until, within c time units, a state in R is reached; that is, $\sigma \in Q\forall\mathcal{U}_{\leq c}\,R$ if for all divergent trajectories τ of B with $\tau(0,0) = \sigma$, there is a position π of τ such that $\tau(\pi) \in R$ and $t_\tau(\pi) \leq c$, and for all positions π' of τ, if $\pi' \leq \pi$ then $\tau(\pi') \in Q \cup R$. Notice that for nonzeno B, the $\forall\mathcal{U}_{\leq c}$-operator is monotonic in its second argument on the subregions of Σ_A; that is, for all regions $Q, R \subseteq \Sigma_A$, we have $R \subseteq Q\forall\mathcal{U}_{\leq c}\,R \subseteq \Sigma_A$.

Proposition 5 *Let B be a nonzeno linear hybrid automaton, let ϕ_1 and ϕ_2 be two state predicates of B, and and let $c \in \mathbb{N}_{>0}$ be a positive integer constant. Then $[\![\phi_1\forall\mathcal{U}\phi_2]\!]_B$ is the least solution of the equation $X = [\![\phi_2]\!] \cup ([\![\phi_1 \vee \phi_2]\!]\forall\mathcal{U}_{\leq c}\,X)$.*

Proof. We define the function $f : \Sigma_B \to \Sigma_B$ such that $f(X) = [\![\phi_2]\!] \cup ([\![\phi_1 \vee \phi_2]\!]\forall\mathcal{U}_{\leq c}\,X)$. We first show that $[\![\phi_1\forall\mathcal{U}\phi_2]\!]_B$ is a fixpoint of f; that is, $[\![\phi_1\forall\mathcal{U}\phi_2]\!]_B = f([\![\phi_1\forall\mathcal{U}\phi_2]\!]_B)$. Since the precondition operator $pre_B^{[\![\phi_1\forall\mathcal{U}\phi_2]\!]}$ is monotonic on the subregions of Σ_B, $[\![\phi_1\forall\mathcal{U}\phi_2]\!]_B \subseteq [\![\phi_1 \vee \phi_2]\!]\forall\mathcal{U}_{\leq c}\,[\![\phi_1\forall\mathcal{U}\phi_2]\!]_B$ Thus $[\![\phi_1\forall\mathcal{U}\phi_2]\!]_B \subseteq f([\![\phi_1\forall\mathcal{U}\phi_2]\!]_B)$.

On the other hand, let σ be a state in $[\![\phi_1 \vee \phi_2]\!]\forall\mathcal{U}_{\leq c}\,[\![\phi_1\forall\mathcal{U}\phi_2]\!]_B$. By the definitions of f and the $\forall\mathcal{U}_{\leq c}$-operator, for all divergent trajectories τ of B with $\tau(0,0) = \sigma$, there is a position π of τ such that $\tau(\pi) \in [\![\phi_1\forall\mathcal{U}\phi_2]\!]_B$ and $t_\tau(\pi) \leq c$, and for all positions π' of τ, if $\pi' \leq \pi$ then $\tau(\pi') \in [\![\phi_1 \vee \phi_2]\!] \cup [\![\phi_1\forall\mathcal{U}\phi_2]\!]_B$. By the definition of $\phi_1\forall\mathcal{U}\phi_2$, we know that $\sigma \in [\![\phi_1\forall\mathcal{U}\phi_2]\!]_B$, which implies that $[\![\phi_1 \vee \phi_2]\!]\forall\mathcal{U}_{\leq c}\,[\![\phi_1\forall\mathcal{U}\phi_2]\!]_B \subseteq [\![\phi_1\forall\mathcal{U}\phi_2]\!]_B$. Furthermore, by the definition of $\phi_1\forall\mathcal{U}\phi_2$, $[\![\phi_2]\!] \subseteq [\![\phi_1\forall\mathcal{U}\phi_2]\!]_B$. Therefore, $f([\![\phi_1\forall\mathcal{U}\phi_2]\!]_B) \subseteq [\![\phi_1\forall\mathcal{U}\phi_2]\!]_B$.

Now we show that $[\![\phi_1\forall\mathcal{U}\phi_2]\!]_B$ is the least fixpoint of f. Let Q be a fixpoint of f; we claim that $[\![\phi_1\forall\mathcal{U}\phi_2]\!]_B \subseteq Q$. Assume that there is a state $\sigma_0 \in [\![\phi_1\forall\mathcal{U}\phi_2]\!]_B \backslash Q$; we will show a contradiction.

We construct inductively an infinite sequence of states $\sigma_i \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B \backslash Q$, for $i \in \mathbb{N}$. Suppose that we have had σ_i and we show how to construct σ_{i+1}. Since $\sigma_i \notin Q$ and $Q = f(Q)$, we know that $\sigma_i \notin f(Q) = [\![\phi_1 \vee \phi_2]\!] \forall\mathcal{U}_{\leq c}\, Q$. Since B is nonzeno, there is a trajectory $\tau_i \in [\![B]\!]^{div}$ with $\tau_i(0,0) = \sigma_i$, and a position π_i of τ_i with $t_{\tau_i}(\pi_i) = c$, such that either

1. for some position $\pi \leq \pi_i$ of τ_i, $\tau_i(\pi) \notin [\![\phi_1 \vee \phi_2]\!]$ and for all positions $\pi' \leq \pi$ of τ_i, $\tau_i(\pi') \notin Q$; or

2. for all positions $\pi \leq \pi_i$ of τ_i, $\tau_i(\pi) \in [\![\phi_1 \vee \phi_2]\!] \backslash Q$.

But since $\sigma_i \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$ and $[\![\phi_2]\!] \subseteq Q$, Condition 1 cannot happen. Condition 2 together with the assumption $\sigma_i \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B \backslash Q$ implies that

3. for all positions $\pi \leq \pi_i$ of τ_i, $\tau_i(\pi) \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!] \backslash Q$.

So $\tau_i(\pi_i) \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!] \backslash Q$ and thus we pick σ_{i+1} to be $\tau_i(\pi_i)$.

Now consider the trajectory τ that results from concatenating the trajectories $\tau_i[(0,0), \pi_i]$ for all $i \geq 0$:

$$\tau \;=\; \tau_0[(0,0), \pi_0] \rightarrow \tau_1[(0,0), \pi_1] \rightarrow \tau_2[(0,0), \pi_2] \rightarrow \cdots.$$

The trajectory τ diverges, because $c > 0$. Since $[\![B]\!]^{div}$ is fusion-closed and divergence-safe, $\tau \in [\![B]\!]^{div}$. Furthermore, $\tau(0,0) = \sigma_0$ and, by Condition 3, $\tau(\pi) \notin Q$ for all positions π of τ. Because $[\![\phi_2]\!] \subseteq Q$, we conclude that $\sigma_0 \notin [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$, which is a contradiction. ■

Reduction to Possibility

For linear arguments, the formula $\phi_1 \forall\mathcal{U}_{\leq c}\, \phi_2$ is definable by the $\exists\mathcal{U}$-operator. Roughly speaking, the condition ϕ_2 must inevitably become true within c time units iff it is not possible that ϕ_2 remains false for c time units.

Proposition 6 *Let B be a linear hybrid automaton, let ϕ_1 and ϕ_2 be two state predicates of B, let $c \in \mathbb{N}$ be a nonnegative integer constant, and let z be a new clock (i.e., z is different from the data variables of B). Then*

$$[\![\phi_1]\!] \forall \mathcal{U}_{\leq c} [\![\phi_2]\!] \;=\; [\![\neg z.((\neg\phi_2)\exists \mathcal{U}(\neg(\phi_1 \vee \phi_2) \;\vee\; z > c))]\!]_B.$$

Proof. By the definition of the $\forall\mathcal{U}_{\leq c}$-operator and the semantics of the temporal operator $\forall\mathcal{U}$, and the reset quantifier z, it is clear that

$$[\![\phi_1]\!] \forall \mathcal{U}_{\leq c} [\![\phi_2]\!] = [\![z.((\phi_1 \wedge z \leq c) \forall \mathcal{U}(\phi_2 \wedge z \leq c))]\!]_B,$$

which over all trajectories of $[\![B]\!]^{div}$ is equivalent to

$$[\![z.((\phi_1 \wedge z \leq c) \forall \mathcal{W}(\phi_2 \wedge z \leq c))]\!]_B,$$

where the temporal operator $\forall\mathcal{W}$ is defined as follows: $\sigma \models_{[\![B]\!]^{div}} \varphi_1 \forall\mathcal{W} \varphi_2$ iff for all trajectories τ of $[\![B]\!]^{div}$ with $\tau(0,0) = \sigma$, either

1. there exists a position π of τ such that $\tau(\pi) \models_{[\![B]\!]^{div}} \varphi_2$ and for all positions $\pi' \leq \pi$, $\tau(\pi') \models_{[\![B]\!]^{div}} \varphi_1 \vee \varphi_2$; or
2. for all positions π of τ, $\tau(\pi) \models_{[\![B]\!]^{div}} \varphi_1$.

Second, we claim that for every hybrid automaton B, and all state predicates ϕ_1 and ϕ_2,

$$[\![\phi_1 \forall\mathcal{W} \phi_2]\!]_B \;=\; [\![\neg((\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2))]\!]_B.$$

The above equality implies that

$$[\![z.((\phi_1 \wedge z \leq c) \forall \mathcal{W}(\phi_2 \wedge z \leq c))]\!]_B = [\![\neg z.((\neg\phi_2 \vee z > c) \exists \mathcal{U}(\neg(\phi_1 \vee \phi_2) \vee z > c))]\!]_B,$$

which would prove the proposition, since the right-hand side of the equality is equal to

$$[\![\neg z.((\neg\phi_2)\exists\mathcal{U}(\neg(\phi_1 \;\vee\; \phi_2) \;\vee\; z > c))]\!]_B.$$

Consider a state $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$. Then there exists a trajectory $\tau \in [\![B]\!]^{div}$ with $\tau(0,0) = \sigma$ and there is a position π_0 of τ such that $\tau(\pi_0) \notin ([\![\phi_1 \vee \phi_2]\!])$ and for all positions $\pi < \pi_0$, $\tau(\pi) \notin [\![\phi_2]\!]$. By the definition of the operator $\forall\mathcal{W}$, it is clear that $\sigma \notin [\![\phi_1\forall\mathcal{W}\phi_2]\!]_B$. It remains to be shown that if $\sigma \notin [\![\phi_1\forall\mathcal{W}\phi_2]\!]_B$, then $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$. Assume that $\sigma \notin [\![\phi_1\forall\mathcal{W}\phi_2]\!]_B$. Then we know that $\sigma \notin [\![\phi_2]\!]$, and there is a trajectory $\tau \in [\![B]\!]^{div}$ with $\tau(0,0) = \sigma$ such that either

1. for all positions π of τ, $\tau(\pi) \notin [\![\phi_2]\!]$, and there is a position π of τ such that $\tau(\pi) \notin [\![\phi_1]\!]$; or

2. there are some positions π of τ with $\tau(\pi) \in [\![\phi_2]\!]$, and for each such position π, there is a position $\pi' < \pi$ such that $\tau(\pi') \notin [\![\phi_1 \vee \phi_2]\!]$.

Condition 1 implies that $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$. Now we assume that Condition 2 holds. Let π_0 be the infimum of all positions π with $\tau(\pi) \in [\![\phi_2]\!]$. Then we know that for all positions $\pi < \pi_0$ of τ, $\tau(\pi) \notin [\![\phi_2]\!]$, and either one of the following two cases must hold: (a) $\tau(\pi_0) \in [\![\phi_2]\!]$ or, (b) $\tau(\pi_0) \notin [\![\phi_2]\!]$ and there is a position π' that is arbitrarily close to position π_0 such that $\pi' > \pi_0$ and $\tau(\pi') \in [\![\phi_2]\!]$.

In case (a), Condition 2 implies that there is a position $\pi' < \pi_0$ such that $\tau(\pi') \notin [\![\phi_1 \vee \phi_2]\!]$. Moreover, we know that for all positions $\pi < \pi_0$ of τ, $\tau(\pi) \notin [\![\phi_2]\!]$, so $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$.

In case (b), assume that position $\pi_0 = (i, \delta_0)$ is the last position of trajectory τ in location v. Then no matter whether state $\tau(i+1,0)$ is in $[\![\phi_2]\!]$ or not, position $(i+1,0)$ should have been the infimum of all positions π with $\tau(\pi) \in [\![\phi_2]\!]$, which is a contradiction. So there must be some position $\pi_2 = (i, \delta_2)$ in the same location v such that $\delta_2 > \delta_0$ and $\tau(\pi_2) \in [\![\phi_2]\!]$. If there is such a position π_2 such that for all position π_1 with $\pi_0 < \pi_1 < \pi_2$, $\pi_1 \in [\![\phi_1 \vee \phi_2]\!]$, then according to Condition 2,

there must be a position $\pi' < \pi_0$ such that $\tau(\pi') \notin [\![\phi_1 \vee \phi_2]\!]$. Thus we can conclude $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$.

Now it remains to consider the cases that satisfy the following condition (Condition 3): for each position π_2 such that $\pi_2 > \pi_0$ and $\tau(\pi_2) \in [\![\phi_2]\!]$, there exists a position π_1 such that $\pi_2 > \pi_1 > \pi_0$, and $\tau(\pi_1) \in [\![\neg\phi_1 \wedge \neg\phi_2]\!]$. Suppose that $\phi_2 = \bigcup_v(v, p_v)$, and $\neg\phi_1 \wedge \neg\phi_2 = \bigcup_v(v, q_v)$. Let (v, δ, ρ) be the trajectory fragment of the trajectory τ containing the position π_0 in location v.

Since $\pi_0 = (v, \delta_0)$ is the infimum of all positions π with $\tau(\pi) \in [\![\phi_2]\!]$, for each $\epsilon > 0$, there is a positive real number ϵ' such that $0 < \epsilon' < \epsilon$ and $\rho(\delta_0 + \epsilon') \in [\![p_v]\!]$. Moreover, by Condition 3, we know that there is a positive real number ϵ'', such that $0 < \epsilon'' < \epsilon'$ and $\rho(\delta_0 + \epsilon'') \in [\![q_v]\!]$. In other words, for any $\epsilon > 0$, the open ball $B(\rho(\delta_0), \epsilon)$ intersects $[\![q_v]\!]$. Therefore $\rho(\delta_0)$ is an *accumulation point* of $[\![q_v]\!]$. By point-set topology, $\rho(\delta_0)$ is in $[\![close(q_v)]\!]$.

In addition, since $\rho(\delta_0)$ is an accumulation point of $[\![q_v]\!]$, again, by point-set topology, for any $\epsilon > 0$, the open ball $B(\rho(\delta_0), \epsilon)$ contains infinitely many points (data states) in $[\![q_v]\!]$. Note that $[\![close(q_v)]\!]$ is defined by the data predicate $close(q_v)$, which cannot define infinitely many isolated data states (this would require infinitely many disjuncts). Therefore, there must be a convex data region $S \subseteq [\![close(q_v)]\!]$ such that for some $\hat{\epsilon} > 0$, $\rho(\delta_0)$ and $\rho(\delta_0 + \hat{\epsilon}) \in [\![q_v]\!]$ are both in S. By the proof of Lemma 1, we know that there is also a linear data trajectory $(\rho', \hat{\epsilon})$ such that $\rho'(0) = \rho(\delta_0)$, $\rho'(\hat{\epsilon}) = \rho(\delta_0 + \hat{\epsilon})$, and $\rho'(t) \in [\![close(q_v)]\!]$ for all $t \in [0, \hat{\epsilon}]$.

Since B is nonzeno and $[\![B]\!]$ is fusion-closed, the trajectory fragment

$$\tau[(0,0), \pi_0] \rightarrow (v, \hat{\epsilon}, \rho')$$

can be extended to a divergent B-trajectory that is a witness trajectory for $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$. ■

The correctness of the SMC-procedure follows from Propositions 4, 5, and 6 (let B be $A_{\vec{z}}$).

Theorem 4 *If the procedure* SMC *halts on the input* (A, ψ)*, then* $|\psi|$ *is a characteristic predicate of* (A, ψ)*; that is,* $[\![|\psi|]\!] = [\![\psi]\!]_A$.

3.3 Implementation of the Symbolic Model Checking Procedure

The SMC-procedure was first implemented as the first version of HYTECH on Sun Sparc stations under MATHEMATICA. Throughout the verification process, state predicates are represented as quantifier-free formulas of the theory $(\mathbb{R}, \leq, +)$ of the reals with addition, over data variables, integrators, and the program counter ℓ.

3.3.1 Direct Implementation

The implementation performs the following operations on state predicates:

Boolean operations Trivial.

Quantifier elimination The quantifier elimination This is necessary to compute the time-precondition operation $pre_{\rightarrow}$ and the transition-precondition operation $pre_{\mapsto}$ on state predicates. The details of the quantifier elimination will be shown later this section.

Validity checking The validity checking is necessary (1) throughout the SMC-procedure, to see if a successive-approximation sequence of state predicates has converged, and (2) after termination of the SMC-procedure, to see if the input automaton A meets the requirement specified by the input formula ψ. Let $|\psi|$ be a characteristic predicate of (A, ψ). Then A meets ψ iff the state predicate $|\psi| \approx \phi_A$ is valid.

The state predicate ϕ is valid iff $\neg\phi$ is unsatisfiable. The implementation determines the satisfiability of state predicates using linear programming. First each linear formula is converted into disjunctive normal form. Then for each disjunct, which is a conjunction, the linear-programming algorithm of MATHEMATICA decides if that conjunction of linear inequalities has a solution. This is an exponential decision procedure for deciding the satisfiability of linear formulas, whose satisfiability problem is NP-complete [HNSY94].

Simplification For quantifier elimination and validity checking, the implementation converts state predicates into disjunctive normal form, which may cause an exponential blow-up of the size of the formulas. To alleviate this problem, we simplify all formulas after each step of the verification process by applying rewrite rules. We use a polynomial-time set of rewrite rules, whose iterated application to a linear formula in disjunctive normal form brings each disjunct into normal form, but may not eliminate redundant or overlapping disjuncts. It is worth noting that the repeated simplification of state predicates is, overall, by far the most time-consuming activity of the implementation.

We now describe the quantifier elimination procedure that we implemented in the first version of HYTECH. In fact, we first implemented the theoretically optimal decision procedure of Ferrante and Rackoff [FR75]. We found, however, that the following "naive" quantifier-elimination procedure performs better for our purposes, perhaps because we need to deal only with quantified formulas in a particular form.

We eliminate quantifiers one by one, starting with innermost quantifiers. Consider the formula $\exists\delta.\, p$, for a linear (quantifier-free) formula p over the set $\vec{x}$ of data variables and the quantified variable δ. We first convert $\exists\delta.\, p$ into an equivalent

formula of the form $\bigvee \exists \delta.\, p_i$, where each p_i is a convex data predicate over $\vec{x} \uplus \{\delta\}$. This is always possible, because the existential quantifier distributes over disjunction. We then convert each linear inequality in p_i into the form $0 \sim e$, where e is a linear term, and $\sim \in \{<, \leq\}$. So suppose that p_i is of the form

$$0 \sim_1 e_1 + c_1 \cdot \delta \wedge \ldots \wedge 0 \sim_k e_k + c_k \cdot \delta \wedge 0 \sim_{k+1} e_{k+1} + c_{k+1} \cdot \delta \wedge \ldots \wedge 0 \sim_m e_m + c_m \cdot \delta,$$

where $c_1, \ldots, c_k > 0$ and $c_{k+1}, \ldots, c_m < 0$. We now "solve" each conjunct for δ; that is, we convert each conjunct into the form $\frac{e_j}{c_j} \sim_j \delta$, for $1 \leq j \leq k$; and $\delta \sim_{j'} \frac{e_{j'}}{c_{j'}}$, for $k+1 \leq j' \leq m$. Then we eliminate δ by combining conjuncts. For example, the conjuncts $x + 3 < \delta$ and $\delta \leq 4y$ are combined to the new conjunct $x + 3 < 4y$. We so obtain the following formula p_i':

$$\bigwedge_{\substack{1 \leq j \leq k \\ k+1 \leq j' \leq m}} (\frac{e_j}{c_j} \sim_{jj'} \frac{e_{j'}}{c_{j'}}),$$

where $\sim_{jj'}$ is $<$ if $\sim_j$ or $\sim_{j'}$ is $<$; and $\sim_{jj'}$ is $\leq$, otherwise.

It is not difficult to check that the resulting quantifier-free formula $\bigvee p_i'$ equivalent to the original formula $\exists \delta.\, p$.

Proposition 7 *Let p_i' be the formula obtained from the formula $\exists \delta.\, p_i$ by the quantifier elimination procedure. Then $p_i' = \exists \delta.\, p_i$, for each i.*

Proof. According to the quantifier elimination procedure, it is clear that $\exists \delta.\, p_i$ implies p_i'. Conversely, suppose that p_i' is *true*. It's sufficient to show that there exist a witness δ_0 such that and $p_i[\delta := \delta_0] = \mathit{true}$.

Suppose that $a = max\{\frac{e_j}{c_j} \mid 1 \leq j \leq k\}$ and $b = min\{\frac{e_{j'}}{c_{j'}} \mid k+1 \leq j' \leq m\}$. Then $a \sim b$ is in p_i'. There exist one or more pairs of indexes j and j' such that the following conditions hold: $j \in [1, k]$, $j' \in [k+1, m]$, $a = \frac{e_j}{c_j}$, $b = \frac{e_{j'}}{c_{j'}}$, and the

conjunct $\frac{e_j}{c_j} \sim_{jj'} \frac{e_{j'}}{c_{j'}}$ is in p'_i. We want to choose a particular pair of j and j' such that $\sim_j$ and $\sim_{j'}$ are as restricted as possible; that is, we prefer $<$ to $\leq$. Precisely, if there is any j such that $a = \frac{e_j}{c_j}$ and $\sim_j = <$, we choose this j as j_0; if there is no such j, we choose any j with $a = \frac{e_j}{c_j}$ as j_0. We choose j'_0 in the same way. From the quantifier elimination procedure, we can always find a number δ_0 such that $a \sim_{j_0} \delta_0 \sim_{j'_0} b$. It is easy to check that δ_0 satisfies all the required conditions and $p_i[\delta := \delta_0]$ is true.■

3.3.2 Better Implementation

After reading Halbwachs's paper [Hal93] about his polyhedral-manipulation library, we realized that most operations in the symbolic model-checking procedure can be substituted with more efficient polyhedral-manipulation operations provided in Halbwachs's polyhedral-manipulation library. Accordingly, we reimplemented HyTech. The details of the new implementation will be described in Chapter 4.

3.3.3 Symbolic Model Checking of the Railroad Gate Controller

We now illustrate the application of the Smc-procedure to the running example, the railroad gate controller. We use HyTech to verify the safety and time-bounded response properties of the railroad gate controller.

Recall the railroad gate controller from Section 2.1.4, and the initial condition *init* from Section 2.2.3. The safety requirement of Section 2.2.3 can be rewritten as

$$init \ \rightarrow \ \neg\exists\Diamond(x \leq 10 \ \wedge \ \ell[2] \neq closed).$$

To avoid negation, we ask HyTech to first compute the characteristic predicate

$$\phi: \quad |\exists\Diamond(x \leq 10 \ \wedge \ \ell[2] \neq closed)|$$

and then check that the state predicate $init \wedge \phi$ is unsatisfiable. It follows that the railroad gate controller meets the safety requirement. HyTech takes 74.3 seconds of CPU time to verify this task.[3] The characteristic predicate ϕ computed by HyTech is the state predicate

$$(\ell[1] = 2 \wedge \ell[2] = 1 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge -1220 \leq -5x - 13y \wedge -90 \leq -y \wedge 0 \leq x \wedge -5 \leq -z \wedge -270 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 1 \wedge \ell[3] = 2 \wedge 90 = y \wedge 0 \leq z \wedge 0 \leq x \wedge -5 \leq -z \wedge -270 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 1 \wedge \ell[3] = 3 \wedge 0 \leq z \wedge 0 \leq x \wedge 0 \leq y \wedge -5 \leq -z \wedge -50 \leq -5x + 13y \wedge -270 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 2 \wedge \ell[3] = 1 \wedge 0 \leq x \wedge -90 \leq -y \wedge -1220 \leq -5x - 13y) \vee (\ell[1] = 2 \wedge \ell[2] = 2 \wedge \ell[3] = 2 \wedge 90 = y \wedge 0 \leq x) \vee (\ell[1] = 2 \wedge \ell[2] = 2 \wedge \ell[3] = 3 \wedge 0 \leq y \wedge 0 \leq x \wedge -50 \leq -5x + 13y) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge -1220 \leq -5x - 13y \wedge -90 \leq -y \wedge 0 \leq x \wedge -5 \leq -z \wedge -270 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 2 \wedge 90 = y \wedge 0 \leq z \wedge 0 \leq x \wedge -5 \leq -z \wedge -270 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 3 \wedge 0 \leq z \wedge 0 \leq x \wedge 0 \leq y \wedge -5 \leq -z \wedge -50 \leq -5x + 13y \wedge -270 \leq -x - 52z) \vee (\ell[1] = 3 \wedge \ell[2] = 1 \wedge \ell[3] = 1 \wedge -10 \leq -x \wedge -90 \leq -y \wedge -5 \leq -z \wedge 0 \leq z) \vee (\ell[1] = 3 \wedge \ell[2] = 1 \wedge \ell[3] = 2 \wedge 90 = y \wedge -10 \leq -x \wedge -5 \leq -z \wedge 0 \leq z) \vee (\ell[1] = 3 \wedge \ell[2] = 1 \wedge \ell[3] = 3 \wedge -10 \leq -x \wedge 0 \leq y \wedge -5 \leq -z \wedge 0 \leq z) \vee (\ell[1] = 3 \wedge \ell[2] = 2 \wedge \ell[3] = 1 \wedge -90 \leq -y \wedge -10 \leq -x) \vee (\ell[1] = 3 \wedge \ell[2] = 2 \wedge \ell[3] = 2 \wedge 90 = y \wedge -10 \leq -x) \vee (\ell[1] = 3 \wedge \ell[2] = 2 \wedge \ell[3] = 3 \wedge 0 \leq y \wedge -10 \leq -x) \vee (\ell[1] = 3 \wedge \ell[2] = 3 \wedge \ell[3] = 1 \wedge -10 \leq -x \wedge -90 \leq -y \wedge -5 \leq -z \wedge 0 \leq z) \vee (\ell[1] = 3 \wedge \ell[2] = 3 \wedge \ell[3] = 2 \wedge 90 = y \wedge -10 \leq -x \wedge -5 \leq -z \wedge 0 \leq z) \vee (\ell[1] = 3 \wedge \ell[2] = 3 \wedge \ell[3] = 3 \wedge -10 \leq -x \wedge 0 \leq y \wedge -5 \leq -z \wedge 0 \leq z) \vee (\ell[1] = 2 \wedge \ell[2] = 1 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge -90 \leq -y \wedge -270 \leq -x - 52z \wedge -10 \leq y - 20z \wedge 180 \leq x + 2y) \vee (\ell[1] = 2 \wedge \ell[2] = 1 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge 0 \leq x \wedge 0 \leq x + 2y \wedge -2650 \leq -5x + 13y - 520z \wedge -90 \leq -y \wedge -100 \leq$$

[3] All performance data are measured on a Sun SPARC-670MP server running Mathematica.

$y - 20z \wedge -5 \leq -z \wedge -2390 \leq -5x - 13y) \vee (\ell[1] = 2 \wedge \ell[2] = 1 \wedge \ell[3] = 2 \wedge 90 =$
$y \wedge 0 \leq z \wedge 0 \leq x \wedge -5 \leq -z \wedge -504 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 2 \wedge \ell[3] =$
$1 \wedge 180 \leq x + 2y \wedge -90 \leq -y) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge -90 \leq$
$-y \wedge -270 \leq -x - 52z \wedge -10 \leq y - 20z \wedge 180 \leq x + 2y) \vee (\ell[1] = 2 \wedge \ell[2] =$
$3 \wedge \ell[3] = 3 \wedge 0 \leq z \wedge 0 \leq y \wedge 0 \leq x \wedge -180 \leq x - 2y \wedge -190 \leq -y - 20z \wedge -1220 \leq$
$-5x + 13y \wedge -3820 \leq -5x - 13y - 520z \wedge -5 \leq -z) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] =$
$4 \wedge 0 = y \wedge 0 \leq z \wedge 0 \leq x \wedge -5 \leq -z \wedge -504 \leq -x - 52z) \vee (\ell[1] = 3 \wedge \ell[2] =$
$3 \wedge \ell[3] = 4 \wedge 0 = y \wedge -10 \leq -x \wedge -5 \leq -z \wedge 0 \leq z) \vee (\ell[1] = 2 \wedge \ell[2] =$
$1 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge -90 \leq -y \wedge -504 \leq -x - 52z \wedge -10 \leq y - 20z \wedge 180 \leq$
$x + 2y) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 3 \wedge 0 \leq z \wedge 0 \leq y \wedge 180 \leq x + 2y \wedge -5 \leq$
$-z \wedge -190 \leq -y - 20z \wedge -180 \leq x - 2y) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 3 \wedge 0 \leq$
$z \wedge 0 \leq x - 2y \wedge 0 \leq y \wedge -100 \leq -y - 20z \wedge -504 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] =$
$3 \wedge \ell[3] = 4 \wedge 0 = y \wedge 180 \leq x \wedge -5 \leq -z \wedge 0 \leq z).$

The time-bounded response requirement of Section 2.2.3 is verified by HyTech using 215.1 seconds of CPU time.

Chapter 4

HyTech : the Implementation of the Symbolic Model-checking Procedure

The advantage of implicit definition over construction
are roughly those of theft over honest toil
— Bertrand Russell

We describe HyTech, the HYbrid TECHnology Tool, an implementation of the symbolic model-checking procedure described in Chapter 3. This chapter also serves as a reference manual for potential users of HyTech.

HyTech currently has three generations. The first generation is the complete implementation of the model-checking procedure introduced in the last chapter. The first generation was implemented in Mathematica to utilize the language's powerful symbolic manipulation and to allow rapid development and experimentation with algorithms and heuristics. Regions are represented as symbolic formulas. The evaluation of the successors of time-step or transition-step uses existential

quantifications, which are easily coded in this language. However, because the quantifier elimination process takes too much time in MATHEMATICA, we started to look for other implementation methods.

The second generation of the verifier is the currently stable version and is ftp available. This version of HYTECH uses a MATHEMATICA main program that called efficient C++ routines from Halbwachs' library. Data regions are represented as sets of convex polyhedral. The quantifier elimination task is replaced by computing the "projection" of a polyhedron with respect to the axis corresponding to the variable to be eliminated. However, we only implemented the reachability analysis portion of the symbolic model checking procedure in this version of HYTECH. We can apply this version of HYTECH to verify the safety properties, time-bounded properties and some duration properties of hybrid systems. This thesis concentrates on this version of HYTECH.

The third generation of HYTECH [HHWT95a,HHWT95b] is also running. The new generation HYTECH that we introduce here avoids MATHEMATICA altogether and is built entirely in C++. It is an even more powerful tool for analyzing complex systems. The third generation of HYTECH is under development, and it is mainly implemented by Howard Wong-Toi. Thus this dissertation refer to the second generation of HYTECH only. There is no significant difference between the algorithms in the second and the third generations.

4.1 Bounded-drift Linear Hybrid Automata

HYTECH handles the *bounded-drift linear hybrid automata*, which are linear hybrid automata A such that

- for each location v of A, $dif(v)$ can be written in the form $\bigwedge_{x_i \in \vec{x}} l_x \leq \dot{x_i} \leq u_x$; and

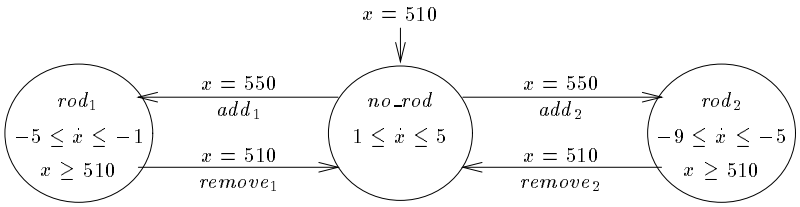

Figure 4.1: The reactor core automaton

- for each transition $e \in E$ of A, $act(e)$ can be written in the form $p \wedge \alpha$, where p is a data predicate over the data variables $\vec{x}$, and α is a data predicate in the form $\bigwedge_{x_i \in \vec{x}} x_i' = t_i$, where t_i is a linear term over the data variables $\vec{x}$.

It is worth noting that, in most literature, the definitions of linear hybrid automata are as general as, or more restricted than, the definition of bounded-drift linear hybrid automata here.

If $dif(v) = \bigwedge_{x_i \in \vec{x}} l_x \leq \dot{x_i} \leq u_x$, the labeling function dif basically assigns to each control location $v \in V$ and each data variable $x_i \in \vec{x}$ a *rate interval* $dif(v, x_i) = [l_x, u_x]$, where l_x and u_x are integer constants. The rate interval $dif(v, x_i) = [l_x, u_x]$ specifies that the first derivative of the data variable x_i may vary within the interval $[l_x, u_x] \subset \mathbb{R}$ while the automaton control resides in location v.

We call p and α the *guard* and the *assignments* of the discrete action $act(e) = p \wedge \alpha$ of a bounded-drift linear hybrid automaton.

4.1.1 The Input Language

The input language of HyTech is straight-forward and can be clearly illustrated by a simple example.

Example: Reactor Temperature Control

We use a variant of the reactor temperature control system from [NOSY93] as a running example for this chapter. The system consists of a reactor core and two control rods that control the temperature of the reactor core. The reactor core is modeled by the linear hybrid automaton in Figure 4.1. The temperature of the reactor core is represented by the variable x. Initially the core temperature is 510 degrees and both control rods are not in the reactor core. In this case, the core temperature rises at a rate that varies between 1 and 5 degrees per second. Notice that $\dot{x}$ is the first derivative of the variable x. The reactor is shut down once the core temperature increases beyond 550 degrees. In order to prevent a shutdown, one of two control rods can be put into the reactor core to dampen the reaction. If control rod 1 is put in, the core temperature falls at a rate that may vary between -5 and -1 degrees per second. Control rod 2 has a stronger effect; if it is put in, the core temperature falls at a rate that varies between -9 and -5 degrees per second. Either control rod is removed once the core temperature falls back to 510 degrees.

Input a Bounded-drift Linear Hybrid Automaton

We now describe the specification of each component of a single bounded-drift linear hybrid automaton using the HyTech input language. The grammar of the HyTech input language is in Appendix A.

Data variables The reactor core automaton from Figure 4.1 has the data variable x. We declare the data variable x as follows.

```
AnaVariables = {x}
```

If we also had a discrete variable y in this example, then we would write the following.

```
AnaVariables = {x}
DisVariables = {y}
```

Location invariants In the reactor core automaton, we have $inv(no\_rod) = (x \leq 550)$, $inv(rod_1) = (x \geq 510)$, and $inv(rod_2) = (510 \leq x)$. In HyTech, we specify these invariants as follows.

```
inv[l[core]==norod] = x<=550
inv[l[core]==rodone] = 510<=x
inv[l[core]==rodtwo] = 510<=x
```

Continuous activities In the reactor core automaton, $dif(no\_rod, x) = [1, 5]$, $dif(rod_1, x) = [-5, -1]$, and $dif(rod_2, x) = [-9, -5]$. In HyTech, we specify these rate intervals as follows:

```
dif[core,norod,x] = {1,5}
dif[core,rodone,x] = {-5,-1}
dif[core,rodtwo,x] = {-9,-5}
```

Discrete actions In HyTech, we specify the transitions, synchronization letters, and guarded commands of the reactor core automaton as follows:

```
act[core,1]={l[core]==norod && 550==x, add1,
{l[core]->rodone}}
act[core,2]={l[core]==norod && 550==x, add2,
{l[core]->rodtwo}}
act[core,3]={l[core]==rodone && 510==x, remove1,
{l[core]->norod}}
act[core,4]={l[core]==rodtwo && 510==x, remove2,
{l[core]->norod}}
```

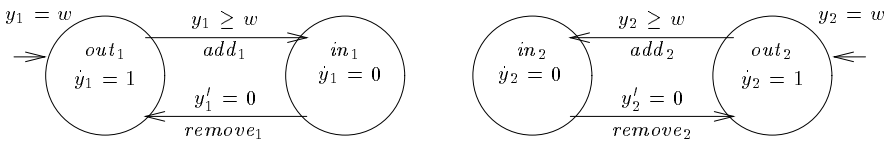

Figure 4.2: The control rod automata

Notice that we encode the source and target locations of a transition within a guarded command.

Input a Set of Bounded-drift Linear Hybrid Automata

When we specify a hybrid system that consists of the parallel composition of several component hybrid automata, we specify each individual hybrid automaton in any order and then specify the *scope* of each synchronization label. For the reactor example, we use the two linear hybrid automata of Figure 4.2 to model the two control rods. Due to the mechanics of moving control rods, after a control rod is removed from the reactor core, it cannot be put back into the core for w seconds, where w is an unknown parameter. This requirement is enforced by the integrator y_1 that measures the time that has elapsed since control rod 1 was removed from the reactor core, and the integrator y_2 that measures the time that has elapsed since control rod 2 was removed. The rod automata synchronize with the core automaton through synchronization letters such as $remove_1$, which indicates the removal of control rod 1. The entire reactor system, then, is obtained by constructing the product of the core automaton of Figures 4.1 and the two rod automata of Figure 4.2.

We now show how the complete reactor temperature control system is specified in HyTech. First we declare the data variables:

```
AnaVariables = {x, y1, y2}
```

```
DisVariables = {w}
```

The data variables x, $y1$, and $y2$ are analog variables, and the data variable w is a discrete variable. We have already defined the reactor core automaton. Now we define the two control rod automata:

```
inv[l[rod1]==out] = 0<=y1
inv[l[rod1]==in] = 0<=y1
inv[l[rod2]==out] = 0<=y2
inv[l[rod2]==in] = 0<=y2

dif[rod1,in,y1]={0,0}
dif[rod1,out,y1]={1,1}
dif[rod2,in,y2]={0,0}
dif[rod2,out,y2]={1,1}

act[rod1,1]={l[rod1]==out && w<=y1, add1, {l[rod1] -> in}}
act[rod1,2]={l[rod1]==in, remove1, {l[rod1] -> out, y1 -> 0}}
act[rod2,1]={l[rod2]==out && w<=y2, add2, {l[rod2] -> in}}
act[rod2,2]={l[rod2]==in, remove2, {l[rod2] -> out, y2 -> 0}}
```

The synchronization alphabet of each automaton is defined by declaring a scope for each synchronization letter. The *scope* of the letter σ is the set of automata that contain σ in their synchronization alphabet. For the reactor temperature control system, we specify

```
syn[remove1] = {rod1,core}
syn[remove2] = {rod2,core}
syn[add1] = {rod1,core}
syn[add2] = {rod2,core}
```

For example, the letter $remove_1$ is used by the reactor core automaton and by the first control rod automaton. This means that the core automaton and the rod 1 automaton must synchronize on transitions labeled with $remove_1$.

While we have given symbolic names like *core* and *no_rod* to automata and locations, the analysis procedures of HyTech require that all automaton names and location names are integers starting from 1. To replace the symbolic names with integers, HyTech calls a macro language preprocessor when it reads an input file. Therefore, we need to define the integer values of the symbolic names at the beginning of the input file. The symbolic names that we use for the reactor temperature control system may be defined as follows:

```
define(rod1,1)
define(rod2,2)
define(core,3)
define(rodone,1)
define(rodtwo,2)
define(norod,3)
define(out,1)
define(in,2)
```

We also must declare the number of input automata, and the number of locations and transitions of each automaton:

```
AutomataNo = 3
locationo = {2,2,3}
transitiono = {2,2,4}
```

The expression `locationo = {2,2,3}` means that the first (control rod 1), second (control rod 2), and third (reactor core) automaton has 2, 2, and 3 locations, respec-

tively. The expression `transitiono = {2,2,4}` specifies the number of transitions in each input automaton.

4.1.2 Global Invariants for Modeling Urgent Transitions

Although the product automaton is constructed automatically by HyTech, it is sometimes useful to specify global conjuncts of all invariants of the product automaton. Such global invariants permit, in particular, the modeling of *urgent transitions*, which are transitions that must be taken as soon as possible. In the graphical representation of hybrid automata, we use boldface synchronization letters to mark urgent transitions. HyTech allows the user to specify location invariants for locations of the product automaton using the command `GlobalInvar`. We will show how urgent transitions can be modeled with global invariants as we analyze the examples of Chapter 7. The reactor temperature control system does not have any urgent transitions, so we write:

```
GlobalInvar = {}
```

This completes the specification of the reactor temperature control system. Except for initial `define` statements, all HyTech input commands can be written in any order.

4.2 Parametric Reachability Analysis

4.2.1 Reachability Analysis

For a region R, recall that we define $pre_A^{\Sigma_A}(R)$ to be the set of all states σ such that σ can reach some state $\sigma' \in R$ by a single time-step or a single transition-step. Similarly, we now define $post_A^{\Sigma_A}(R)$ to be the set of all states σ such that some state $\sigma' \in R$ can reach σ by a single time-step or a single transition-step. To simplify the notations, we write $pre(R)$ for $pre_A^{\Sigma_A}(R)$ and $post(R)$ for $post_A^{\Sigma_A}(R)$ if

no ambiguity would occur. We show in Section 3.1.2 that $pre(R)$ are again regions. By the similar argument, we can also show that $post(R)$ are again regions. We write $pre^*(R)$ for the infinite union $\cup_{i\geq 0}\ pre^i(R)$, and $post^*(R)$ for the infinite union $\cup_{i\geq 0}\ post^i(R)$. In other words, $pre^*(R)$ is the set of all states that can reach a state in R by a trajectory of A; and $post^*(R)$ is the set of all states that can be reached from a state in R by a trajectory of A.

The *reachability problem* $(A, \varphi_I, \varphi_F)$ for a nonzeno linear hybrid automaton A, an initial state predicate φ_I, and a final state predicate φ_F, asks if the region $post^*([\![\varphi_I]\!]) \cap [\![\varphi_F]\!]$ is empty or, equivalently, if the region $[\![\varphi_I]\!] \cap pre^*([\![\varphi_F]\!])$ is empty. In other words, the reachability problem $(A, \varphi_I, \varphi_F)$ asks if there is a trajectory of A from some state in $[\![\varphi_I]\!]$ to some state in $[\![\varphi_F]\!]$. If $[\![\varphi_I]\!]$ represents the set of "initial" states of the automaton A, and $[\![\varphi_F]\!]$ represents the set of "unsafe" states specified by a safety requirement, then the safety requirement can be verified by reachability analysis: the automaton satisfies the safety requirement iff the reachability problem has the answer *yes* (i.e., $post^*([\![\varphi_I]\!]) \cap [\![\varphi_F]\!] = \emptyset$).

Unfortunately, the computation of $post^*([\![\varphi_I]\!])$ or $pre^*([\![\varphi_F]\!])$ may not terminate within a finite number of *post* or *pre* operations, because the reachability problem for linear hybrid automata is undecidable. HyTech, in other words, offers a semidecision procedure for the reachability analysis. It is our experience, however, that for practical examples, including the examples in this dissertation, the computation does terminate and HyTech solves the corresponding reachability problems. Indeed, as for the practitioner there is little difference between a nonterminating computation and one that runs out of time or space resources, we submit that decidability questions are mostly of theoretical interest.

For the reactor temperature control system, we wish to check the safety requirement that *the reactor never needs to be shut down*; more precisely, whenever the core temperature reaches 550 degrees, then either y_1 or y_2 shows more than

w seconds, thus allowing the corresponding control rod to be put into the reactor core. Let A denote the product of the reactor core automaton and the two control rod automata. We define the reachability problem $(A, \varphi_I, \varphi_F)$ as follows. The initial states are characterized by the state predicate φ_I =

$$(\ell[rod_1] = out \wedge \ell[rod_2] = out \wedge \ell[core] = no\_rod \wedge x = 510 \wedge y_1 = w \wedge y_2 = w);$$

that is, initially no rod is in the reactor core, the initial temperature is 510 degrees, and $y_1 = y_2 = w$. The unsafe states are characterized by the state predicate φ_F =

$$(\ell[rod_1] = out \wedge \ell[rod_2] = out \wedge \ell[core] = no\_rod \wedge x = 550 \wedge y_1 \leq w \wedge y_2 \leq w);$$

that is, the unsafe situation is that the core temperature reaches 550 degrees and neither y_1 nor y_2 shows more than w seconds (and, thus, none of the control rods is available). The answer to the reachability problem $(A, \varphi_I, \varphi_F)$ is *yes* iff the reactor temperature control system satisfies the safety requirement.

In HyTech, the reachability problem is specified as follows:

```
InitialState = l[rod1]==out && l[rod2]==out &&
l[core]==norod && 510==x && w==y1 && w==y2
Bad = l[rod1]==out && l[rod2]==out && l[core]==norod &&
550==x && y1<=w && y2<=w
```

Forward versus backward analysis

HyTech can attack a reachability problem by forward analysis or by backward analysis. Given the reachability problem $(A, \varphi_I, \varphi_F)$, the *forward analysis* computes the state predicate that defines the region $post^*([\![\varphi_I]\!])$, and then takes the conjunction with the final state predicate φ_F; the *backward analysis* computes the state predicate that defines the region $pre^*([\![\varphi_F]\!])$, and then takes the conjunction with the initial state predicate φ_I. Chapter 5 would cover more details about this

issue. Only the backward analysis terminates for the reactor temperature control system.

We ask HyTech to perform a forward or backward analysis, respectively, by writing

```
Go := PrintTime[ Iterative[Forward] ]
```

or

```
Go := PrintTime[ Iterative[Backward] ]
```

These commands also print the CPU time consumed by the reachability analysis.

4.2.2 Parametric Analysis

The automatic derivation of delay parameters was introduced for real-time systems in [AHV93] and now we apply the technique to hybrid systems. We can use HyTech to synthesize necessary and sufficient conditions on system parameters such that a hybrid automaton satisfies a requirement.

Recall that the reactor temperature control system contains the parameter w, which specifies the necessary rest time for a control rod. Clearly, the safety requirement will not be satisfied for large values of w. Indeed, the *target region* $[\![\varphi_I]\!] \cap pre^*([\![\varphi_F]\!])$ gives a sufficient and necessary condition on w such that the safety requirement is not satisfied. Typically the state predicate that defines the target region is too complex to see the conditions on the parameters clearly, but these can be isolated in HyTech using *elimination operators*. By writing

```
EliminateLocList = {rod1,rod2,core}
EliminateVarList = {x,y1,y2}
```

we eliminate all location information from the state predicate that defines the target region, and we project out all information about the data variables x, y_1, and

y_2. Then the resulting projection of the target region, as computed by HyTech using backward analysis, is

```
9w>=184
```

In other words, the target region is empty if and only if $9w < 184$. It follows that $9w < 184$ is a necessary and sufficient condition on the parameter w that prevents the reactor from shutdown. The verification requires 17.27 seconds of CPU time.[1]

4.2.3 Abstract Interpretation

To expedite the reachability analysis and to force the termination of the analysis, HyTech provides several abstract-interpretation operators [CC77,HH95c], including the convex-hull operator and the extrapolation operator. Chapter 5 would provide more details about the abstract-interpretation operators.

In HyTech, we write

```
TakeConvex = True
```

or

```
TakeConvex = False
```

to turn the convex-hull operator on or off, respectively. The extrapolation operator can be turned on or off selectively for individual control locations. For example, if we want to apply the extrapolation operator only to data regions that correspond to the two locations $\ell[rod_1] = out \wedge \ell[rod_2] = in \wedge \ell[core] = no\_rod$ and $\ell[rod_1] = in \wedge \ell[rod_2] = out \wedge \ell[core] = no\_rod$ of the reactor temperature control system, then we write:

[1] The performance figure is given for a SPARC 670MP station.

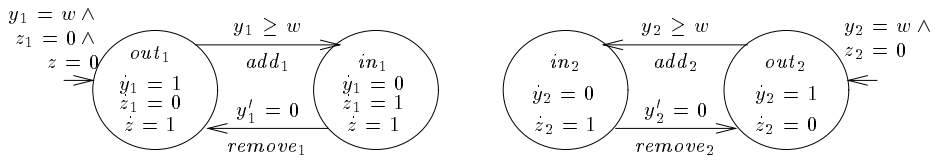

Figure 4.3: The augmented control rod automata

```
WideSet[lc_] =
((lc === l[rod1]==out && l[rod2]==in && l[core]==norod) ||
(lc === l[rod1]==in && l[rod2]==out && l[core]==norod))
```

The commands

```
WideSet[lc_] = True
```

and

```
WideSet[lc_] = False
```

ask HyTech to apply the extrapolation operator to all or none of the control locations, respectively. (In our analysis of the reactor temperature control system, it was not necessary to use any abstract-interpretation operators.)

4.2.4 Checking More Properties by Reachability Analysis

While the first HyTech prototype accepts Ictl input, we have not yet completed the implementation of Ictl model checking for the current version of HyTech. Fortunately, besides safety requirements, we can check most time-bounded response requirements and duration requirements using reachability analysis by moving clocks and integrators from the requirement specification to the system model. Consider, for example, the duration requirement of the reactor temperature control system that *independent of the control strategy that is used for deciding which*

control rod to put into the reactor core, each control rod is used at most one third of the time:

$$\varphi_I \;\rightarrow\; (z\colon true)(z_1\colon l[rod_1] = in)(z_2\colon l[rod_2] = in)\; \forall\Box(3z_1 \leq z \wedge 3z_2 \leq z).$$

To verify this ICTL requirement, we move the clock z and the integrators z_1 and z_2 to the control rod automata as shown in Figure 4.3. Then we use HYTECH to check if any state in the unsafe region $[\![3z_1 \geq z \vee 3z_2 \geq z]\!]$ is reachable from an initial state.

4.3 Geometric Implementation

HYTECH uses a MATHEMATICA main program that calls C++ routines from Halbwachs' polyhedron-manipulation library. The verifier requires conversions between MATHEMATICA expressions and C++ data structures.

4.3.1 Polyhedron-manipulation Library

We describe in this chapter what operators the polyhedron-manipulation library provides and how we implement the fixpoint computation procedure using the provided operators. The readers should refer to [Hal93,HRP94] for more details about the library.

The most common representation of a convex polyhedron P is a system of closed linear inequalities $P = \{\vec{x} \mid M\vec{x} \geq \vec{c}\}$. We shall call this representation the *inequality representation*. To store a convex polyhedron P in the inequality representation, we store the matrix $[M|\vec{c}]$. On the other hand, by Minkowski's theorem [NW88], every nonempty convex polyhedron P in $\mathbb{R}^n$ has a unique (within scalar multiplication) representation by its extreme points $X = \{\vec{x}_i \mid i \in I\}$ and

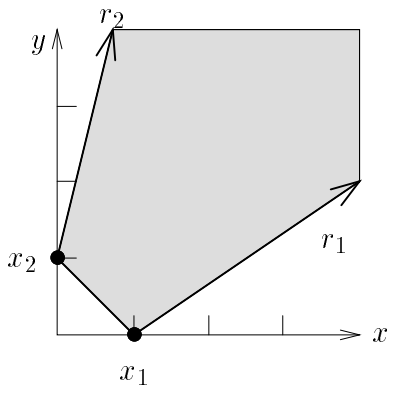

Figure 4.4: Two representations of a convex polyhedron

extreme rays $R = \{\vec{r}_j \mid j \in J\}$ such that

$$P = \{\vec{x} \in \mathbb{R}^n \mid \vec{x} = \sum_{i \in I} \lambda_i \vec{x}_i + \sum_{j \in J} \mu_j \vec{r}_j, \text{ where } \lambda_i, \mu_j \in \mathbb{R}_{\geq 0} \text{ and } \sum_{i \in I} \lambda_i = 1\}.$$

We shall call this representation the *frame representation*. Notice that if the point $\vec{y}$ and the ray $\vec{r}$ are in P, then so are all points of the form $\vec{y} + \lambda\vec{r}$, for $\lambda \in \mathbb{R}_{\geq 0}$. To store a convex polyhedron P in the frame representation, we store the pair of matrices (X, R) where X is the matrix consisting of the points as columns and R is the matrix consisting of the rays as columns.

For example, the polyhedron P in Figure 4.4 can be represented in the inequality representation as below

$$P = \begin{cases} 4x - y \geq -1 \\ -2x + 3y \geq 2 \\ x + y \geq 1 \end{cases} \quad \text{or} \quad P = \begin{bmatrix} 4 & -1 \\ -2 & 3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \geq \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix}$$

or can be represented in the frame representation as the points

$$x_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \; x_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

and the rays

$$r_1 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \; r_2 = \begin{bmatrix} 1 \\ 4 \end{bmatrix}.$$

We need both representations because some polyhedral operations are performed on the frame representation and some polyhedral operations are performed on the inequality representation. There exist two efficient translations that translate each representation to the other and minimize the representations on-the-fly [Che68, LeV92].

Halbwachs's library provides the implementation of the two translations and also the following operators on convex polyhedra.

- Projection operator: given a convex polyhedron $P = (X, R)$ and a ray $\vec{r}$, the projection operator adds ray $\vec{r}$ to P; that is, the operator returns $(X, [R\ \vec{r}])$. The projection operator is used to implement the quantifier elimination and the time-precondition operator in HyTech.

- Assignment operator: given a convex polyhedron $P = (X = [\vec{x}_1 \cdots \vec{x}_n], R)$ and a set of assignments $\vec{x'} := M\vec{x} + \vec{c}$ represented by the matrix M, the assignment operator computes the convex polyhedron P' that consists of the states that can be obtained by applying the assignments to all the states in P. In fact,

  $$P' = ([M\vec{x}_1 + \vec{c} \cdots M\vec{x}_n + \vec{c}], MR).$$

  The assignment operator is used to implement transition-precondition operator in HyTech.

- Intersection operator: given a convex polyhedron $P = [M|\vec{c}]$ and a convex polyhedron $Q = [N|\vec{d}]$, the intersection operator computes the convex poly-

hedron $P \cap Q$ by computing

$$P \cap Q = \left[\begin{array}{c|c} M & \vec{c} \\ N & \vec{d} \end{array}\right].$$

HyTech uses the intersection operator to compute the conjunction of two data predicates.

- Emptyness test operator: given a convex polyhedron $P = (X, R)$, the emptyness test operator checks if P is empty using the fact that P is empty if and only if X is empty. HyTech uses the emptyness test operator for validity tests.

- Inclusion test operator: given a convex polyhedron $P = (X, R)$ and a convex polyhedron $Q = [M|\vec{c}]$, the inclusion test operator tests if $P \subseteq Q$ using the fact that $P \subseteq Q$ if and only if (1) for all $\vec{x} \in X$, $M\vec{x} \geq \vec{c}$ and (2) for all $\vec{r} \in R$, $M\vec{r} \geq 0$. HyTech uses the inclusion test operator to check if the fixpoint computation converges.

- Convex-hull operator: given a convex polyhedron $P = (X, R)$ and a convex polyhedron $Q = (Y, S)$, the convex-hull operator computes the convex hull of P and Q, denoted by $P \sqcup Q$. The convex hull $P \sqcup Q$ in its frame representation is simply $(X \cup Y, R \cup S)$. The convex-hull operator is used by HyTech as an abstract operator, which will be discussed in Chapter 5.

4.3.2 Implementation using the Polyhedron-manipulation Library

We now outline how our tool utilizes the efficient convex polyhedron operators supplied by Halbawachs's polyhedron-manipulation library. Symbolic analysis of a linear hybrid automaton requires boolean operations on the data predicates and

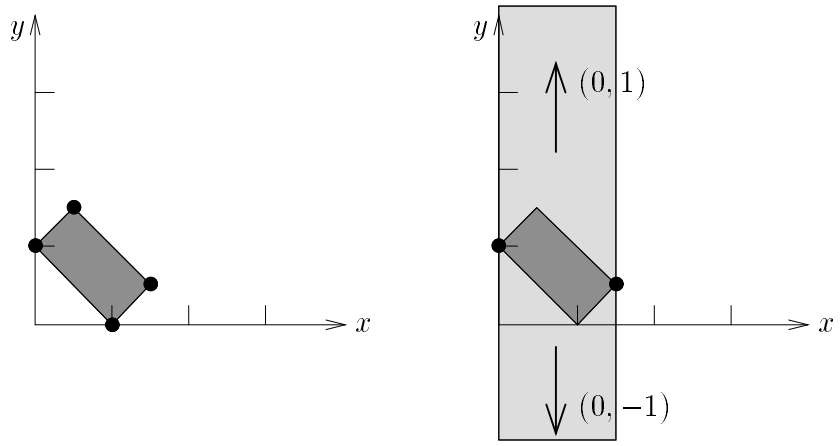

Figure 4.5: Quantifier elimination: eliminating variable y

the iterated computation of the *pre* and *post* operators. To compute the *pre* and *post* operators, we need to know how to compute the time-precondition operator, time-postcondition operator, transition-precondition operator and transition-postcondition operator. We describe the computation for these operators below.

Quantifier elimination The elimination of an existential quantified data variable x_i from a formula $\exists x_i.\, p$ is achieved by adding the ray whose i-th component is 1 and whose other components are all 0, and also the ray whose i-th component is -1 and whose other components are all 0, to the frame representation of the data region $R = [\![p]\!]$. We use an example to illustrate this algorithm. Suppose that we want to eliminate the variable y from the quantified formula be $\exists y.\, -1 \leq x - y \leq 1 \wedge 1 \leq x + y \leq 2$. It is easy to verify that the result is the data predicate $0 \leq x \leq \frac{3}{2}$ using the quantifier-elimination procedure that we introduced in Chapter 3. The data region $R = [\![-1 \leq x - y \leq 1 \wedge 1 \leq x + y \leq 2]\!]$ is depicted on the left of Figure 4.5. After adding the rays $(0, 1)$ and $(0, -1)$ to the frame representation of the data region R, we get the (shaded) data region $[\![0 \leq x \leq \frac{3}{2}]\!]$ on the right of

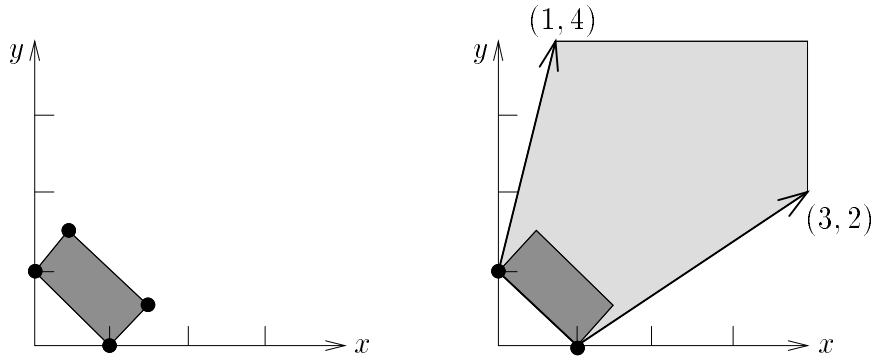

Figure 4.6: Time step

Figure 4.5, which exactly represents the quantifier-eliminated data predicate.

Time step The time-postcondition operator $post^{[\![inv(v)]\!]}_{\rightarrow}(v; R)$ returns the data region that consists of all the states that can be reached by a state in the data region R with respect to a time step. Notice that the time-postcondition operator is defined analogously to the definition of the time-precondition operator in Section 3.1.2. From now on we write $post_{\rightarrow}(v; R)$ for $post^{[\![inv(v)]\!]}_{\rightarrow}(v; R)$ for simplicity. The time-postcondition operator is used for computing the operator $post$. The time-postcondition operator $post_{\rightarrow}(v; R)$ is computed by adding rays that delineate the "shadow" created by the continuous evolution of the data variables to the frame representation of the data region R. In general, if a bounded-drift linear hybrid automaton A has n data variables $(x_1, \ldots, x_n)$, we require the addition of $2^n - 2$ rays, where the i-th coordinate of each ray is either the upper or lower bound of the rate interval for the variable x_i. The ray whose all coordinates are lower bounds and the ray whose all coordinates are upper bounds, are captured by linear combinations of the other rays and thus need not to be added. The polyhedron obtained by adding the $2^n - 2$ rays is then intersected with the polyhedron that defines

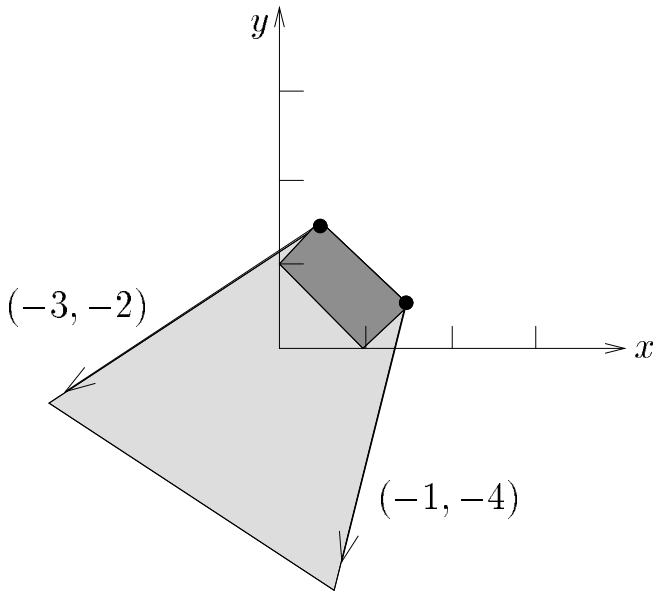

Figure 4.7: Reverse time step

the location invariant.

For example, suppose hybrid automaton A has two variables x_1 and x_2, where x_1 has the rate interval $[1, 3]$ and x_2 has the rate interval $[2, 4]$ in a location v with location invariant *true*. Let the data region be $R = -1 \leq x - y \leq 1 \wedge 1 \leq x + y \leq 2$ (shown on the left in Figure 4.6). The time-postcondition of R, $post_{\rightarrow}(v; R)$, is shown on the right in Figure 4.6. It is computed by adding two rays $(1, 4)$ and $(3, 2)$ to the frame representation of the data region R.

The time precondition operator $pre_{\rightarrow}(v; R)$ can be computed by quantifier elimination shown in Section 3.1.2. But we can compute it more efficiently using the same method that we compute $post_{\rightarrow}(v; R)$ except that we add rays of opposite directions to the data region R. For example, the time precondition $pre_{\rightarrow}(v; R)$ of the data region R from the previous example is

shown in Figure 4.7, which is obtained by adding the rays $(-1,-4)$ and $(-3,-2)$ to the data region R.

Transition step The transition postcondition operator is defined analogously to the definition of the transition precondition operator in Section 3.1.2. So the transition postcondition of a convex region R of an automaton A, $post^{\Sigma_A}_{\mapsto}(v;R)$, is the region that consists of all the states that can be reached by a state in the region R with respect to a transition step. We shall write $post_{\mapsto}(v;R)$ for $post^{\Sigma_A}_{\mapsto}(v;R)$. The transition postcondition operator can be computed directly using the intersection operator and the assignment operator provided by the polyhedron-manipulation library as follows. For a specific automaton transition e with the action predicate $p \wedge \alpha$, where p is the guard and α is the set of assignments of the transition, we first compute the data region $P = R \cap [\![p]\!]$ using the intersection operator. Then we use the assignment operator in the library to compute the region that consists of all the states that can be reached by a state in the region (v,P) via the assignments α.

We follow the method introduced in Section 3.1.2 to compute the transition precondition $pre_{\mapsto}(v;R)$ using quantifier elimination and the intersection operator.

4.4 More Examples Verified by HyTech

We introduce more verification examples using HyTech in this section to show the applicability of HyTech and to provide the readers more opportunities to practice the usage of the model and the tool.

4.4.1 Timing-based Mutual Exclusion

Consider the mutual-exclusion problem for an asynchronous distributed system with local clocks. The system consists of two processes, P_1 and P_2, with atomic read and write operations on a shared memory. Each process has a critical section, and at every time instant at most one of the two processes is allowed to be in its critical section. Mutual exclusion can be ensured by a version of Fischer's protocol [Lam87], which we first describe in pseudocode. Each process P_i, for $i = 1, 2$, executes the protocol in Table 4.1.

Table 4.1: The timing-based mutual exclusion protocol

repeat
  repeat
    await $k = 0$
    $k := i$
    delay b
  until $k = i$
  Critical section
  $k := 0$
forever

The two processes P_1 and P_2 share the variable k, and process P_i is allowed into its critical section iff $k = i$. Each process has a private clock. The instruction **delay** b delays a process for at least b time units as measured by its local clock. Furthermore, each process takes at most a time units, as measured by its local clock, for a single write access to the shared memory (i.e., for the assignment $k := i$).

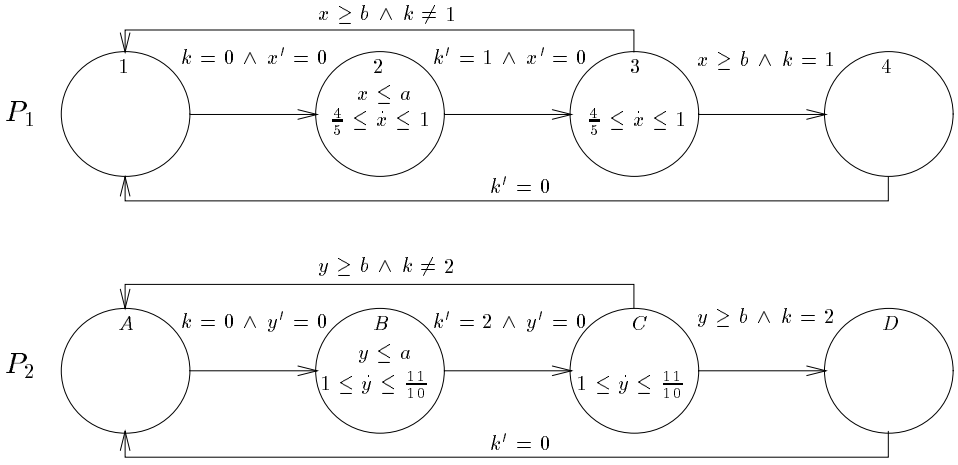

Figure 4.8: Timing-based mutual-exclusion protocol

To make matters more interesting, we assume that the two local clocks of the processes P_1 and P_2 are inaccurate and proceed at different rates. Indeed, the clock rates may vary within certain bounds: the local clock of P_1 is slow—its rate varies between 4/5 and 1—and the local clock of P_2 is fast—its rate varies between 1 and 11/10. The resulting system can be modeled by the product of the two hybrid automata shown in Figure 4.8. Each of the two automata models one process, with the two critical sections being represented by the locations 4 and D, respectively. The parameters a and b are shared data variables with unknown, constant values (i.e., their rate of change is 0 in all locations, and they are not altered by any transitions).

The initial condition φ_I of the system is given by the state predicate

$$\ell = (1, A) \wedge k = 0.$$

The final condition is the state predicate

$$\ell = (4, D).$$

The target region $[\![\varphi_I]\!] \cap pre^*([\![\varphi_F]\!])$ computed by HyTech in 50.1 seconds of CPU time, is the represented by the state predicate

$$\ell = (1, A) \wedge k = 0 \wedge (a \geq b \vee 11a \geq 8b).$$

Therefore, the protocol guarantees mutual exclusion iff the delay parameters a and b are chosen such that $8b > 11a$.

A further advantage of the symbolic approach to system analysis is its insensitivity to the magnitude of constants in the system description. To demonstrate this, we performed the following experiment. We verified the mutual-exclusion property for hybrid automata that are identical to those shown in Figure 4.8, except that the parameters a and b are instantiated, first with $a = 2$ and $b = 4$, then with $a = 2 \cdot 10^3$ and $b = 4 \cdot 10^3$, and finally with $a = 2 \cdot 10^6$ and $b = 4 \cdot 10^6$. We found that, for this example, the verification time taken by HyTech is independent of the magnitude of the parameter values. The results are shown in Table 4.2.

Table 4.2: Coefficient size versus performance

Magnitude of the coefficients	CPU time (in seconds)
10^0	6.11
10^3	6.55
10^6	6.36

4.4.2 Leaking Gas Burner

In [CHR91], the duration calculus is used to prove that a gas burner does not leak excessively. It is known that (1) any leakage can be detected and stopped within 1 second and (2) the gas burner will not leak for 30 seconds after a leakage has

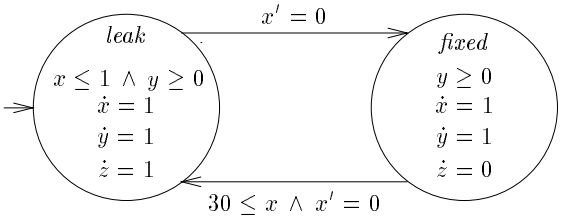

Figure 4.9: The Gas Burner Automaton

been stopped. We wish to prove that the accumulated time of leakage is less than one twentieth of the time in any interval of at least 60 seconds.

The system is modeled by the hybrid automaton in Figure 4.9. The automaton has two locations: in location *leak*, the gas burner leaks; *fixed* is the nonleaking location. The integrator z records the cumulative leakage time; that is, the accumulated amount of time that the system has spent in location *leak*. The clock x records the time the system has spent in the current location; it is used to model the facts (1) and (2). The clock y records the total elapsed time. Initially the gas burner is leaking and all the variables are zero. In 6.5 seconds, HyTech shows that the final region $y \geq 60 \wedge 20z \geq y$ would not be reached from the initial state.

4.4.3 Two Different Schedulers

Data variables that range over a finite domain of values can be encoded within the control of a hybrid automaton. Our experience has been that HyTech performs better if all finite-state variables are encoded in the control. We illustrate this observation with two examples taken from scheduling—a priority scheduler and a round-robin scheduler. Schedulers are examples of hybrid systems with integrators, which measure the accumulated amount of execution time for each task.

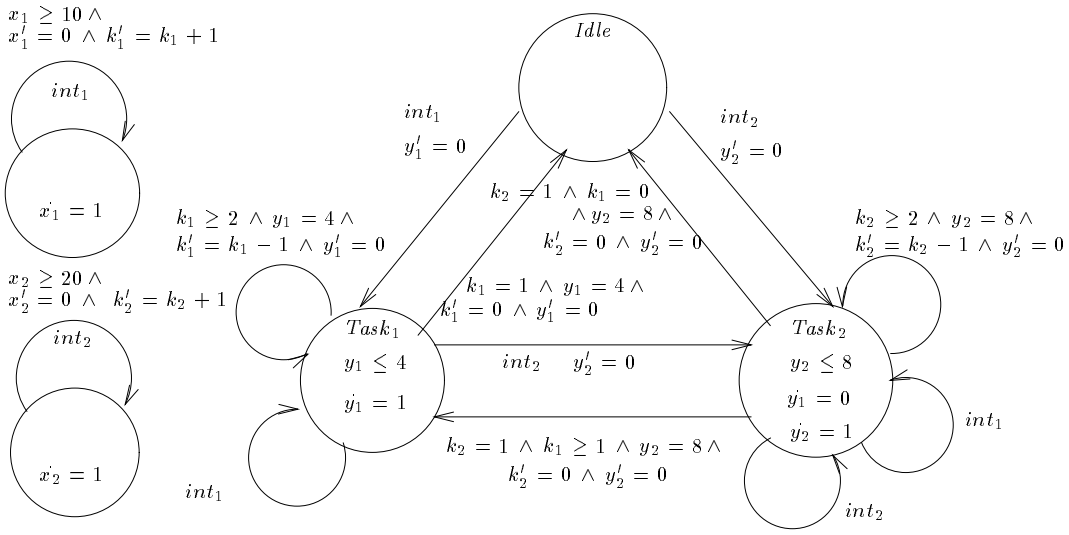

Figure 4.10: The priority scheduler, data version

Priority Scheduler

The two small hybrid automata on the left of Figure 4.10 model an environment that generates two types of interrupts: an interrupt of type int_1 arrives at most once every 10 sec; an interrupt of type int_2, at most once every 20 sec. For every int_i interrupt, a task of type i needs to be executed: each type-1 task requires 4 sec; each type-2 task, 8 sec. Moreover, only one processor is available for the execution of all tasks, and type-2 tasks have priority over type-1 tasks and interrupt their execution. The resulting priority scheduler is modeled by the hybrid automaton on the right of Figure 4.10. In location $Task_1$, a type-1 task is being executed; in location $Task_2$, a type-2 task. The variable k_i represents the number of incomplete (i.e., running and pending) tasks of type i; the variable y_i represents the execution time of the current task of type i. The three automata synchronize on the synchronization labels int_1 and int_2; that is, whenever an int_i interrupt arrives, the scheduler takes a transition that is labeled with int_i. The entire scheduling

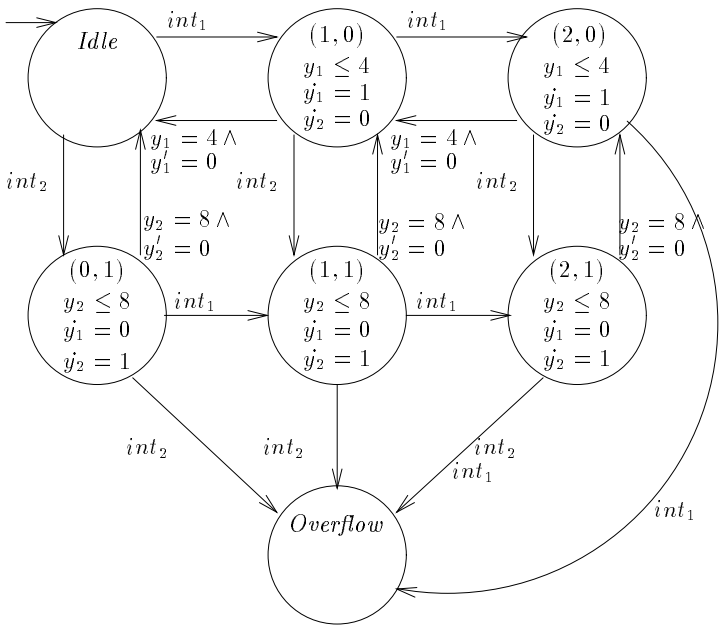

Figure 4.11: The priority scheduler, control version

system, then, is the product of the three component automata. Using HyTech, we verified that in any trajectory starting from the region $(Idle, k_1 = k_2 = 0)$ the number of incomplete type-1 tasks never exceeds 2 and the number of incomplete type-2 tasks never exceeds 1; that is, no state that satisfies $k_1 \geq 3 \vee k_2 \geq 2$ is reachable. The HyTech input file for this example is in Appendix B.

Since the variables k_1 and k_2 range over finite domains, they can be encoded in the control of the scheduler. The control version of the priority scheduler (Figure 4.11), then, contains a location (k_1, k_2) for all possible values of the variables k_1 (namely, 0, 1, and 2) and k_2 (0 and 1). The location *Overflow* represents any state with $k_1 \geq 3$ or $k_2 \geq 2$. We checked that the region *Overflow* cannot be reached from the region *Idle*. We also increased the number of interrupt and task types and the results of these experiments, which were performed on a Sparc 670MP station, are shown in Table 4.3. Clearly, our verifier handles finite-state control

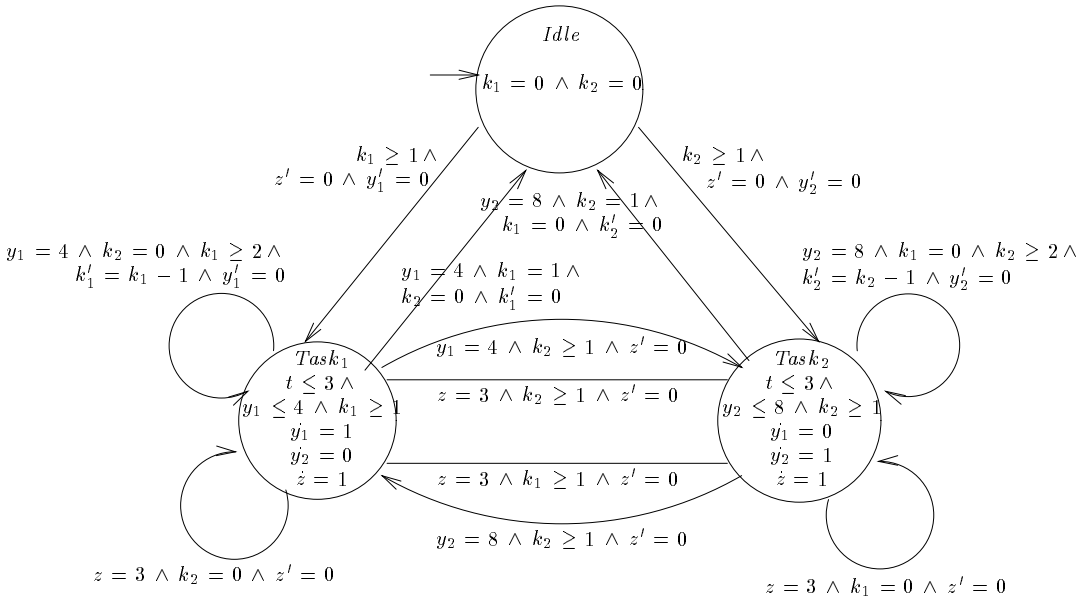

Figure 4.12: The round-robin scheduler

more efficiently than real-valued data.

Table 4.3: Performance data (CPU time)

Priority scheduler, 2 tasks, data version	47 sec
Priority scheduler, 2 tasks, control version	20 sec
Priority scheduler, 3 tasks, data version	89 sec
Priority scheduler, 3 tasks, control version	32 sec
Round-robin scheduler, 2 tasks, data version	171 sec
Round-robin scheduler, 2 tasks, control version	38 sec

Round-robin Scheduler

Figure 4.12 shows a round-robin scheduler. The two task types are assigned alternating time slices of 2 sec, as measured by the clock z. If all tasks of one type are completed, the scheduler starts a new time slice for the tasks of the other type. Again we check that the number of incomplete type-1 tasks is bounded by 2 and the number of incomplete type-2 tasks is bounded by 1, which allows us to encode the variables k_1 and k_2 in the control of the scheduler. The verification results are also shown in Table 4.3.

Chapter 5

Efficient Symbolic Model Checking

"A Physicist believes", said the mathematician, "that 60 is divisible by all numbers. He observes that 60 is divisible by 1, 2, 3, 4, 5, and 6. He examines a few more cases, as 10, 20, and 30, taken at random as he says. Since 60 is divisible also by these, he considers the experimental evidence sufficient." "Yes, but look at the engineer", said the physicist. "An engineer suspected that all odd numbers are prime numbers. At any rate 1 can be considered as a prime number, he argued. Then there comes 3, 5 and 7, all indubitably primes. Then there comes 9; and awkward case, it does not seem to be a prime number, yet 11 and 13 are certainly primes. "Coming back to 9", he said, "I conclude that 9 must be an experimental error"." — George Polya

In this chapter we discuss several model-checking strategies and approximation methods that are designed to improve the performance of HyTech. We (1) simultaneously compute the target region from different directions, (2) conservatively approximate the target region by dropping constraints, and (3) iteratively refine

the approximation until sufficient precision is obtained. We consider the standard abstract convex-hull operator and a novel abstract extrapolation operator.

The main theoretical limitation of the SMC-procedure for the hybrid systems is that termination (i.e., finite approximability) is not guaranteed (problem A); the main practical limitation of the implementation is its cost, which is largely due to two factors. First, continuous activities correspond to quantifier-elimination steps on real-valued linear expressions (problem B); second, if expressions are transformed into disjunctive normal form for further manipulation, then the number of disjunctions grows rapidly (problem C).

Having identified the three problems, we improved the performance of HYTECH by adding various abstract interpretation strategies, both exact and conservative, to guide the fixpoint computation of the target region. To address problem A, we approach the target region simultaneously from several directions, so that success is guaranteed even if only one direction terminates (Section 5.2). To address problem B, we represent regions as unions of convex polyhedra, so that quantifier elimination is replaced by operations on polyhedra. To address problem C, we conservatively approximate the union of several convex polyhedra by a single convex polyhedron. We discuss two abstract operators—convex-hull approximation and extrapolation (Section 5.3). Each approximation can be iteratively refined to achieve the desired precision (Section 5.4).

We wish to point out that the abstract operators and the iterative approximation process are well-known *abstract interpretation* techniques [CC77,Cou81] applied to a new domain—that of linear hybrid automata. The convex-hull operator [CH78] and the widening operator [CC77], which is similar to our extrapolation operator, are dynamic abstract operators used for convergence acceleration [CC92]. Also the idea of two-way iteratively refining approximations is due to Patrick and Radhia Cousot [CC92], and it has been used for the analysis of real-

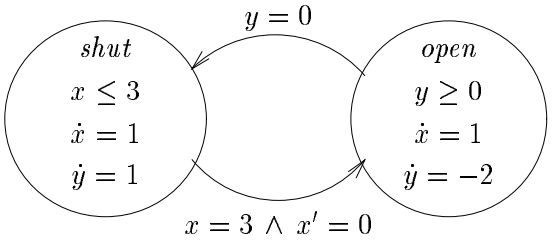

Figure 5.1: The water-tank automaton

time systems [DW93]. We acknowledge the work of Nicolas Halbwachs, who has applied the convex-hull and widening operations to the analysis of synchronous programs [Hal93] and linear hybrid automata [HRP94]; also, the current implementation of HyTech makes use of Halbwachs' polyhedron manipulation library. We suggest a new extrapolation operator instead of Halbwachs' widening operator, which is sometimes too coarse for our purposes (on the other hand, widening, unlike extrapolation, guarantees the convergence of iterative application). We feel that the convex-hull and extrapolation operators are particularly suited for the analysis of the *continuous* and *linear* state spaces, because a polyhedron more naturally represents all of its interior points rather than just the grid of interior integer points, and because the linear regions are closed under both approximation operations.

5.1 Example: Water Tank

The hybrid automaton of Figure 5.1 models a water-level controller that opens and shuts the outflow of a water tank. The automaton has two data variables, x and y, and two locations, *shut* and *open*. The variable x represents a clock of the water-level controller and the variable y represents the water level in the tank. Since the clock x measures time, the first derivative of x is always 1 (i.e., $\dot{x} = 1$). In

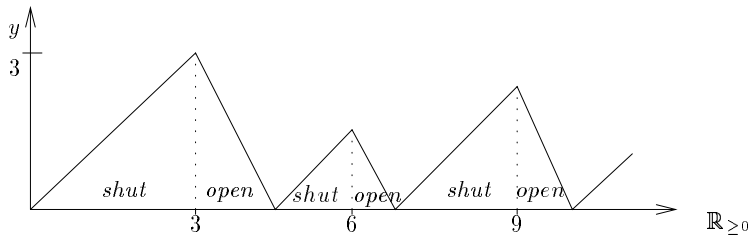

Figure 5.2: The change of the water level y of the water-tank automaton

location *shut*, the outflow of the water tank is shut and the water level increases 1 inch per second ($\dot{y} = 1$); in location *open*, the outflow of the water tank is open and the water level decreases 2 inches per second ($\dot{y} = -2$). The transition from *open* to *shut* (shut the outflow) is taken as soon as the water tank becomes empty: the guard $y = 0$ on the transition ensures that the transition may be taken only when the water level is 0; the constraint $y \geq 0$ on *open* ensures that the transition to *shut* must be taken before the water level becomes negative. The transition from *shut* to *open* (open the outflow) is taken every 3 seconds: the transition is taken whenever $x = 3$, and the transition restarts the clock x at 0. If the automaton is started from the state $(shut, x = y = 0)$, then Figure 5.2 shows how the water level y changes as a piecewise-linear function of time.

5.2 Forward versus Backward Reachability Analysis

Recall that there are two possible approaches for solving the reachability problem by symbolic model checking: we may compute the region (represented by a data predicate in HyTech) $post^*(S)$ of states that can be reached from the initial region S and check if $post^*(S) \cap T = \emptyset$ (forward reachability analysis), or we may compute the region $pre^*(T)$ of states from which the final region T can be reached

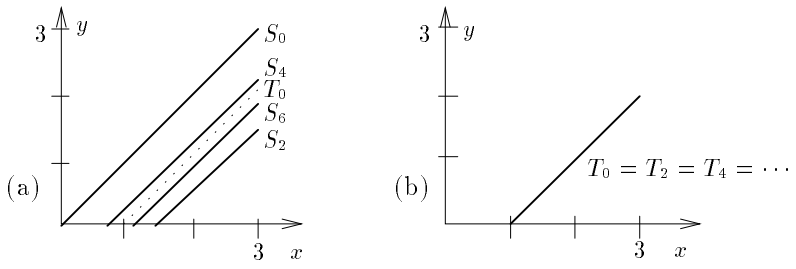

Figure 5.3: Exact forward and backward analysis

and check if $pre^*(T) \cap S = \emptyset$ (backward reachability analysis). For any given reachability problem, one approach may perform better than the other approach; indeed, it may be that one approach terminates and the other does not.

Recall, for example, the water-tank automaton of Figure 5.1. We wish to check if the final region $T = (shut, 1 \leq x \leq 3 \wedge x = y + 1)$ can be reached from the initial region $S = (shut, x = y \wedge x \leq 3)$. The procedure computes the two regions $post^*(S)$ and $pre^*(T)$ iteratively as the limits of two infinite sequences of regions. Let $S_{i+1} = post(S_i)$ be the region of states that can be reached from S by $i + 1$ single steps, and let $T_{i+1} = pre(T_i)$ be the region of states that can reach T by $i + 1$ single steps. Then $post^*(S)$ is the limit $\cup_{i \geq 0} S_i$ of the infinite region sequence $\cup_{0 \leq i \leq n} S_n$, for $n \geq 0$, and $pre^*(T)$ is the limit $\cup_{i \geq 0} T_i$ of the infinite region sequence $\cup_{0 \leq i \leq n} T_n$, $n \geq 0$. It is clear that if $post^{n+1}(S) \subseteq \cup_{0 \leq i \leq n} post^i(S)$, then $post^*(S) = \cup_{0 \leq i \leq n} post^i(S)$, and an analogous termination condition holds for the computation of $pre^*(T)$. HyTech can check if T is reachable from S on the fly; that is, if $post^n(S) \cap T \neq \emptyset$ or $pre^n(T) \cap S \neq \emptyset$, then T is reachable from S and the limit computation is aborted. Since for each $i \geq 0$,

$$S_{2i} = (shut, x - y = 1 + \frac{(-1)^{i+1}}{2^i} \wedge x \leq 3)$$

(Figure 5.3(a)) and

$$S_{2i+1} = (\mathit{open}, 2x + y = 2 + \frac{(-1)^i}{2^i} \wedge y \geq 0),$$

the forward computation of $\mathit{post}^*(S)$ does not terminate within any finite number of iterations. The backward computation, however, converges in a single iteration with the result

$$T_0 = T_2 = (\mathit{shut}, 1 \leq x \leq 3 \wedge x = y + 1)$$

(Figure 5.3(b)) and

$$T_1 = T_3 = (\mathit{open}, x + 2y = 2).$$

It follows that

$$\mathit{pre}^*(T) = (\mathit{shut}, 1 \leq x \leq 3 \wedge x = y + 1) \cup (\mathit{open}, x + 2y = 2).$$

Since $S \cap \mathit{pre}^*(T) = \emptyset$, we conclude that the final region T is not reachable from the initial region S. For optimal performance, we can also implement a strategy that dovetails both approaches by computing the alternating sequence $S_0, T_0, S_1, T_1, S_2, \ldots$ of regions.

5.3 Convergence Acceleration through Abstract Operators

Sometimes the exact computation of a region is prohibitively expensive or does not terminate, independent of the verification strategy (forward v.s. backward). In such cases, we approximate the target region and iteratively refine the approximation until sufficient precision is obtained.

We discuss two convergence acceleration operators, convex hull and extrapolation, which can be turned on or off by the user of HyTech. Both operators overapproximate a union of convex polyhedra by a single convex polyhedron and

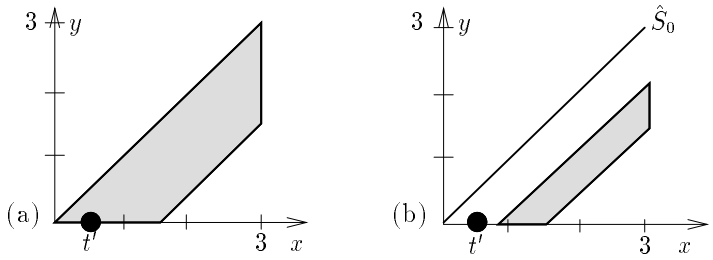

Figure 5.4: Application of the naive and the refined convex-hull operators

thus reduce the time and space requirements of the verifier; indeed, either operator, or the combination of both operators, may cause the termination of an otherwise infinite computation. If we overapproximate, by $\hat{S}$, the target region $post^*(S)$ of states that are reachable from the initial region S (i.e., $post^*(S) \subseteq \hat{S}$), and $\hat{S}$ contains no final state from T, then we can conclude that T is not reachable from S; if, on the other hand, $\hat{S}$ does contain a final state, then we cannot reach a valid conclusion and must refine the approximation.

An *abstract operator* $+$ is a binary operator on data regions such that for two data regions S and S', the result $S+S'$ is a convex data region with $S \cup S' \subseteq S+S'$. For the abstract operator $+$, the *approximated postcondition* of the region S is the region $post_+(S) = \cup_v(v, S_v + post(S)_v)$. Then $\cup_{0 \leq i \leq n} post^i(S) = post^n_\cup(S) \subseteq post^n_+(S)$, and we can overapproximate the target region $post^*(S)$ by the limit $\hat{S} = \cup_{i \geq 0} post^i_+(S)$. While we concentrate on the approximate forward analysis in this chapter, analogous observations hold for the approximate backward analysis using *pre*.

5.3.1 Convex Hull

An obviously abstract operator is the *convex-hull operator* $\sqcup$ [Hal93,HRP94], which maps two data regions S and S' to the convex hull of the union $S \cup S'$. Recall

once again the water-tank automaton of Figure 5.1. Now we wish to check if the final state $T' = (\mathit{shut}, x = \frac{1}{2} \wedge y = 0)$ can be reached from the initial region $S = (\mathit{shut}, x = y \wedge x \leq 3)$. Again, the forward computation of the region $\mathit{post}^*(S)$ does not terminate. This time, however, we can make the forward approach work with the help of the convex-hull operator. Let $\tilde{S}_i = \mathit{post}^i_{\sqcup}(S)$. Then $\tilde{S}_0 = S$,

$$\begin{aligned}
\tilde{S}_1 &= (\mathit{shut}, x - y = 0 \wedge x \leq 3) \cup (\mathit{open}, y + 2x = 3 \wedge y \geq 0), \\
\tilde{S}_2 &= (\mathit{shut}, (x - y = 0 \wedge x \leq 3) \sqcup (x - y = \tfrac{3}{2} \wedge x \leq 3)) \\
&\quad \cup (\mathit{open}, y + 2x = 3 \wedge y \geq 0) \\
&= (\mathit{shut}, 0 \leq x - y \leq \tfrac{3}{2} \wedge x \leq 3) \quad (\mathit{Figure}\ 5.4(a)) \\
&\quad \cup (\mathit{open}, y + 2x = 3 \wedge y \geq 0), \\
\tilde{S}_3 &= (\mathit{shut}, 0 \leq x - y \leq \tfrac{3}{2} \wedge x \leq 3) \cup (\mathit{open}, \tfrac{3}{2} \leq y + 2x \leq 3 \wedge y \geq 0), \\
\tilde{S}_4 &= \tilde{S}_3.
\end{aligned}$$

The forward computation terminates with the overapproximation $\tilde{S}_3$ of the target region. Since $\tilde{S}_3$ contains T', however, we cannot conclude that T' is or is not reachable from S. Thus we refine our approximation strategy, and apply the convex-hull operator only if the resulting region does not contain any final states. This refinement is sensible for all abstract operators. Formally, we overapproximate the target region $\mathit{post}^*(S)$ by the limit $\hat{S} = \cup_{i \geq 0} \hat{S}_i$, where

$$\hat{S}_{i+1} = \begin{cases} \mathit{post}_+(\hat{S}_i) & \text{if } \mathit{post}_+(\hat{S}_i) \cap T = \emptyset, \\ \mathit{post}(\hat{S}_i) & \text{otherwise.} \end{cases}$$

For the water-tank example, we obtain $\hat{S}_0 = S$,

$$\begin{aligned}
\hat{S}_1 &= (\mathit{shut}, x - y = 0 \wedge x \leq 3) \cup (\mathit{open}, 2x + y = 3 \wedge y \geq 0), \\
\hat{S}_2 &= (\mathit{shut}, x - y = \tfrac{3}{2} \wedge x \leq 3) \cup (\mathit{open}, 2x + y = 3 \wedge y \geq 0),
\end{aligned}$$

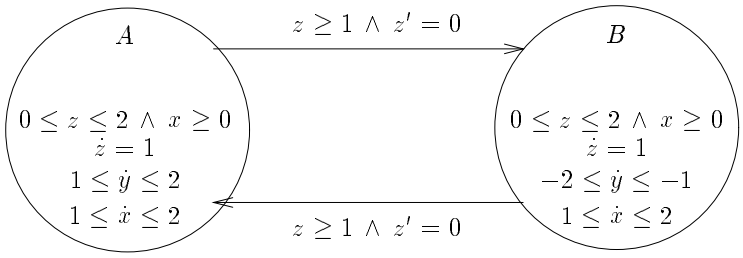

Figure 5.5: The robot automaton

$$
\begin{aligned}
\hat{S}_3 &= (\mathit{shut}, x - y = \tfrac{3}{2} \wedge x \leq 3) \\
&\quad \cup (\mathit{open}, (2x + y = 3 \wedge y \geq 0) \sqcup (2x + y = \tfrac{3}{2} \wedge y \geq 0)) \\
&= (\mathit{shut}, x - y = \tfrac{3}{2} \wedge x \leq 3) \cup (\mathit{open}, \tfrac{3}{2} \leq 2x + y \leq 3 \wedge y \geq 0), \\
\hat{S}_4 &= (\mathit{shut}, (x - y = \tfrac{3}{2} \wedge x \leq 3) \sqcup (\tfrac{3}{4} \leq x - y \leq \tfrac{3}{2} \wedge x \leq 3)) \\
&\quad \cup (\mathit{open}, \tfrac{3}{2} \leq 2x + y \leq 3 \wedge y \geq 0) \\
&= (\mathit{shut}, \tfrac{3}{4} \leq x - y \leq \tfrac{3}{2} \wedge x \leq 3) \quad (\mathit{Figure}\ 5.4(b)) \\
&\quad \cup (\mathit{open}, \tfrac{3}{2} \leq 2x + y \leq 3 \wedge y \geq 0), \\
\hat{S}_5 &= \hat{S}_4.
\end{aligned}
$$

The forward computation terminates with the overapproximation $\cup_{0 \leq i \leq 4} \hat{S}_i$ of the target region. Since $\cup_{0 \leq i \leq 4} \hat{S}_i$ does not contain T', we conclude that the final state T' is not reachable from the initial region S.

5.3.2 Extrapolation

Convex-hull approximation is typically useful for systems with variables whose values are bounded, such as the water-tank automaton. By contrast, if the regions in the sequence $\cup_{0 \leq i \leq n} S_i$, $n \geq 0$, grow without bound, we may want an extrapolation operator that "guesses" an overapproximation of the limit $\cup_{i \geq 0} S_i$. Consider, for example, the hybrid automaton of Figure 5.5, which models the movement of a

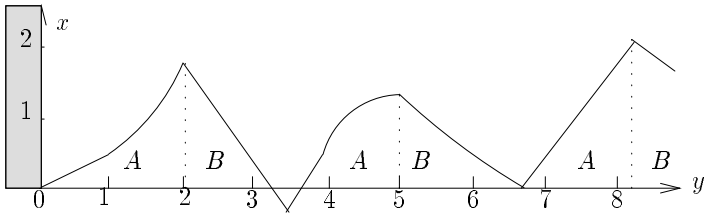

Figure 5.6: A trajectory of the robot automaton

robot in the (x, y)-plane. The robot can be in one of two modes—heading roughly northeast (location A) or heading roughly southeast (location B): in mode A, the derivatives of the coordinates x and y vary within the closed interval $[1, 2]$; in mode B, the derivative of x remains between 1 and 2, while the derivative of y changes its sign and varies between -1 and -2. The robot changes its mode every 1 to 2 minutes, according to its clock z. Moreover, there is a wall at the $x = 0$ line, causing the system invariant $x \geq 0$. Figure 5.6 shows how the position of the robot in the (x, y)-plane may change with time, assuming the robot is started in the initial state $S = (\mathit{shut}, x = y = z = 0)$.

We wish to check if the robot can reach, from S, the final position $T = (x = 9 \wedge y = 12)$. At this point, we invite the ambitious reader to prove that T cannot be reached from S. Using HyTech, the backward computation of the region $\mathit{pre}^*(T)$ terminates within 74 seconds of CPU time[1] after seven iterations of pre, and $S \cap \mathit{pre}^*(T) = \emptyset$. The forward computation of the region $\mathit{post}^*(S)$, by contrast, does not terminate, because the robot keeps zig-zagging to the east and the limit $\cup_{i \geq 0} \mathit{post}^i(S)$ cannot be computed exactly.[2] As convex-hull approximation does not help with this example, we define an extrapolation operator that approximates

[1] All performance figures are given for a SPARC 670MP station.

[2] True, if the target region T is reachable, then it can be reached within a finite number of discrete transitions, and if not, then the invariant $x > 9$ becomes true after a finite number of transitions; extrapolation, however, avoids the ad-hoc "guessing" of suitable invariants.

the unbounded target region $post^*(S)$; using HyTech, the forward computation with extrapolation, then, terminates within 8 seconds of CPU time.

We first give an intuitive motivation for the extrapolation operator. Suppose that the iterative computation of the target region leads, for a given control state, to the data region (polyhedron) S and, in the subsequent iteration, to the polyhedron S'. Suppose, furthermore, that there is a function f such that f maps S to S', mapping extreme points to extreme points. A reasonable guess for the target region, then, would be $f^\infty(S)$.

The nonempty convex polyhedron S has *dimension* k, denoted $dim(S) = k$, if there are $k+1$ points (states) $x_1, x_2, \ldots, x_{k+1}$ S such that the k differences $x_2 - x_1, x_3 - x_1, \ldots, x_{k+1} - x_1$ are linearly independent. Notice that if $dim(S) = 0$, then S is a point, and if $dim(S) = 1$, then S is a straight line, a ray, or a line segment. The set $F = \{x \in S \mid ax = b\}$, for two vectors a and b, is a *face* of S if every point x in S satisfies the inequality $ax \leq b$. Notice that every face of S is also a convex polyhedron. The face F is a *facet* of S if $dim(F) = dim(S) - 1$. The set F is a 1-dimensional face of S iff F is the intersection of $dim(S) - 1$ facets of S (Criterion (†)).

Now consider two polyhedra S and S'. We first compute the convex hull $S \sqcup S'$ of $S \cup S'$. All points and rays of the frame representation of the convex hull $S \sqcup S'$ are extreme points and extreme rays of either S or S'. Let $\{x_k \mid k \in I\}$ be the extreme points of both $S \sqcup S'$ and S, let $\{x_l \mid l \in I'\}$ be the remaining extreme points of $S \sqcup S'$, and let $\{r_j \mid j \in J\}$ be the extreme rays of $S \sqcup S'$. Suppose that $k \in I$ and $l \in I'$, and x_k and x_l are the endpoints of a 1-dimensional face of $R \sqcup S'$, which can be checked by Criterion (†). Then it is possible that $f(x_k) = x_l$ for the imaginary function f from S to S'. Moreover, if f is applied infinitely many times, then all points of the form $x_k + \lambda(x_l - x_k)$, for $\lambda \in \mathbb{R}_{\geq 0}$, are included in $f^\infty(R)$. We therefore add the ray $x_l - x_k$ into the generators of $S \propto S'$: the *extrapolation*

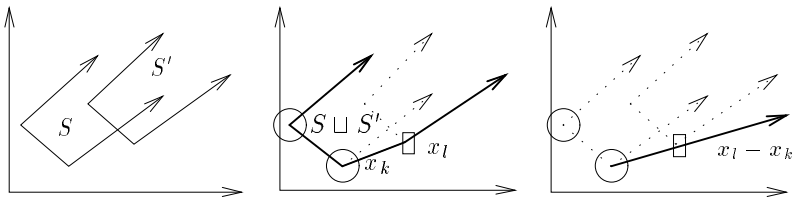

Figure 5.7: Extrapolation operator

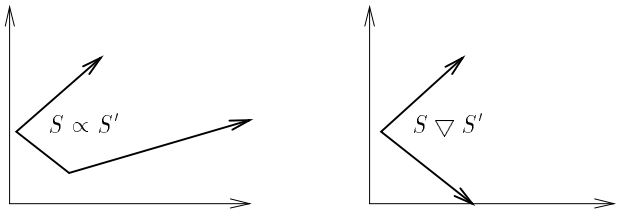

Figure 5.8: Results of the extrapolation operator and the widening operator

operator $\propto$ maps the two polyhedra S and S' to $S \propto S' =$

$$\{x \in \mathbb{R}_n \mid x = \sum_{i \in I \cup I'} \lambda_i x_i + \sum_{j \in J} \mu_j r_j + \sum_{(k,l) \in K} \mu_{k,l}(x_l - x_k),$$
$$\text{where } \lambda_i, \mu_j, \mu_{k,l} \in \mathbb{R}_{\geq 0} \text{ and } \sum_{i \in I \cup I'} \lambda_i = 1\},$$

where $(k,l) \in K$ iff $k \in I$ and $l \in I'$ and both x_k and x_l are in the intersection of $dim(R \sqcup S') - 1$ facets of $S \sqcup S'$. Notice that $\propto$ is an abstract operator, that $\propto$ is not symmetric (i.e., $S \propto S'$ and $S' \propto S$ may be different), and that $S \sqcup S' \subseteq S \propto S'$.

Figure 5.7 shows the application of the extrapolation operator to two convex 2-dimensional polyhedra S and S'. The result of the extrapolation $S \propto S'$ is shown as the left figure in Figure 5.8. As a comparison, the right figure in Figure 5.8 shows the result of applying the widening operator $\triangledown$ [Hal93,HRP94] to the same regions.

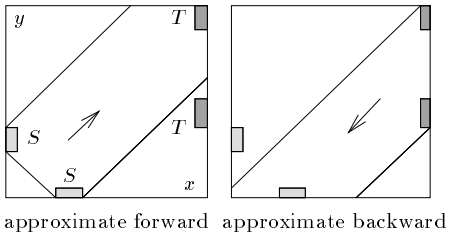

Figure 5.9: One-way approximative analysis

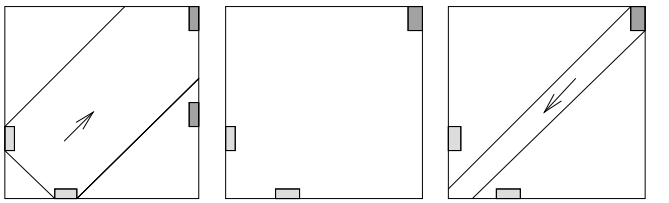

Figure 5.10: Two-way iterative approximative analysis

5.4 Two-way Iterative Approximation

Suppose that an approximate forward reachability analysis is inconclusive; that is, the overapproximation $\hat{S}$ of the target region $post^*(S)$ contains some final states from T. If the intersection $\hat{S} \cap T$ is a proper subset of T, then we have nonetheless obtained new information—namely, that the states in $T - \hat{S}$ are not reachable from S—and we may proceed computing backward the new target region $pre^*(T \cap \hat{S})$, or an overapproximation $\hat{T}$ of $pre^*(T \cap \hat{S})$. If $\hat{T} \cap S = \emptyset$, then T is not reachable from S; if, on the other hand, $\hat{T}$ does contain some initial states from S, then we may continue the two-way iterative approximation process, this time computing forward from $\hat{T} \cap S$.

The two-way iterative approximation strategy is illustrated in Figure 5.9 and Figure 5.10, assuming a 2-dimensional state space and convex-hull approximation.

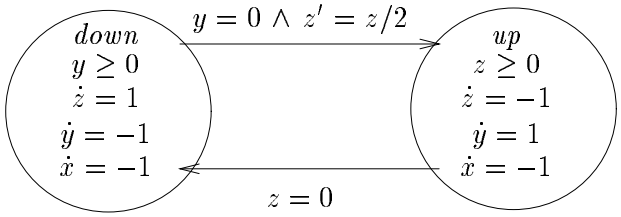

Figure 5.11: The bouncing-ball automaton

The two shaded boxes in the lower left corner of the state space represent the initial region S; the two shaded boxes in the upper right corner represent the final region T. The derivatives of both variables x and y are 1. While an exact forward or backward analysis would show that T cannot be reached from S, both approximate forward analysis and approximate backward analysis are inconclusive (Figure 5.9). An approximate forward analysis followed by an approximate backward analysis is successful (Figure 5.10).

We now apply the two-way iterative approximation strategy to analyze the bouncing-ball automaton of Figure 5.11. The variable x represents the horizontal distance of the ball from a reference point, the variable y represents the distance of the ball from the floor, and the variable z represents the "energy" of the ball. Suppose that the ball is dropped at the position $x = 14 \wedge y = 4$ in the direction of the reference point; that is, $S = (\mathit{down}, x = 14 \wedge y = 4 \wedge z = 0)$. While the ball is falling (location *down*), its energy is increasing ($\dot{z} = 1$); when the ball hits the floor ($y = 0$), it loses half of its energy and bounces back up (location *up*); on the way up, the energy decreases ($\dot{z} = -1$) until it becomes 0 and the ball starts to fall again. The trajectory of the ball is shown in Figure 5.12.

We wish to prove that the ball never reaches the region $T = (x \leq 2 \wedge y = 0)$ of the floor. First notice that the exact forward computation of the target region $\mathit{post}^*(S)$ does not terminate, and convex-hull approximation is of no help.

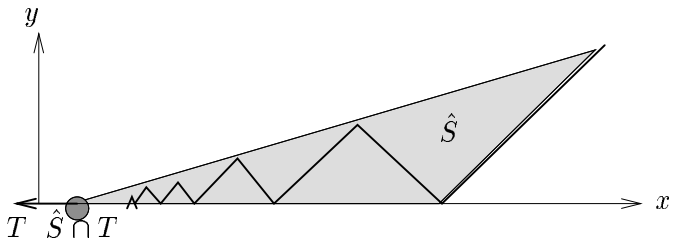

Figure 5.12: A trajectory of the bouncing-ball automaton

If we use extrapolation, then we obtain the overapproximation $\hat{S}$ of the target region $post^*(S)$, and $\hat{S}$ contains the final states $\hat{S} \cap T = (x = 2 \wedge y = 0 \wedge z = 0)$. Since $pre(\hat{S} \cap T) = \hat{S} \cap T$, a second, backward, pass terminates immediately without reaching the initial state S. HyTech requires 12 seconds of CPU time for both passes. (Notice that if we were to begin with a backward pass, attempting to compute the target region $pre^*(T)$, neither the exact computation, nor the approximate computation with convex-hull approximation, nor the approximate computation with extrapolation, would terminate.)

Chapter 6

Analysis of Nonlinear Hybrid Automata

But two permissible and correct models of the same external objects may yet differ in respect of appropriateness

— Heinrich Hertz

This chapter extends the model-checking approach to the analysis of certain nonlinear systems.[1]

We present, analyze, and apply two algorithms for translating nonlinear hybrid systems into linear hybrid systems to analyze nonlinear hybrid automata, which alone provide an accurate model for most real-time environments (for example, the temperature of a furnace decreases along an exponential curve with negative

[1] Control theory, of course, has a long tradition of analyzing what we call nonlinear hybrid systems (in control theory, the term *linearity* usually refers to differential equations, not trajectories). There, however, the number of discrete modes of a controller is typically quite small. Model checking, on the other hand, allows the analysis of controllers that are defined by arbitrary finite-state programs. While the control theorists start with complex environments—differential equations—and steadily increase the complexity of the controllers that can be analyzed, we computer scientists start with complex controllers—programs!—and steadily increase the complexity of the environments.

exponent).[2]

As we know how to compute time-precondition and time-postcondition accurately only for linear hybrid automata, we propose a two-step methodology for verifying a nonlinear hybrid automaton A. In Step 1, we translate A into a linear hybrid automaton B. In Step 2, we apply the HyTech tool to the translated automaton B. The translation is *sound*, for a class $\mathcal{P}$ of properties, if all $\mathcal{P}$-properties of B are inherited by A; *complete*, if all $\mathcal{P}$-properties of A are inherited by B. If the translation is $\mathcal{P}$-sound and B satisfies the property $\phi \in \mathcal{P}$, then so does A. Incomplete translations may lead to false negatives: B may not satisfy the $\mathcal{P}$-property ϕ although A does. We therefore accompany incomplete translations with error analyses: given a metric d on hybrid automata, what is the automaton A' d-closest to A such that if A' satisfies ϕ, then so does B? If the d-difference between A and A' can be made arbitrarily small, then the translation is called *asymptotically complete* under the metric d.

We present two translations, which transform two incomparable classes of nonlinear hybrid automata into linear hybrid automata. Both translations can be automated. The *clock translation* replaces nonlinear variables by clocks (i.e., linear variables with slope 1). The clock translation is applicable to the nonlinear variable x if the value of x is always uniquely determined by the latest assignment to x and the time that has expired since that assignment. If the clock translation of the automaton A yields the automaton B, then the underlying transition systems are timed bisimilar. It follows that the clock translation is both sound and complete for all branching-time properties. If all variables of a nonlinear hybrid automaton can be replaced by clocks, then the resulting linear hybrid automaton

[2] We insist on representing nonlinear behavior accurately in our underlying *model*, because we feel that linearization or digitization ought to occur *after* the modeling phase, so that the errors that are introduced by these processes can be analyzed and bounded. Such an analysis, indeed, is performed in the present paper for both linearizations we propose.

is a timed automaton [AD94]. As a corollary, we obtain a new decidable class of hybrid automata. We verify a nonlinear version of the railroad gate controller in Chapter 2 using the clock translation and the HyTech model checker.

The *rate translation* approximates nonlinear variables by piecewise-linear variables. The rate translation is applicable to the nonlinear variable x if the value of x is bounded. If the rate translation of the automaton A yields the automaton B, then the transition system of B simulates the transition system of A (but not vice versa). It follows that the rate translation, while sound for all linear-time properties, is not complete for safety properties. We show that the rate translation is asymptotically complete for safety properties. We verify the thermostat automaton in Chapter 2 using the rate translation and HyTech. Technically, both the clock translation and the rate translation can be viewed as abstract interpretations of nonlinear hybrid automata [CC77].

6.1 Verification of Nonlinear Hybrid Automata

We associate with a hybrid automaton A a labeling function *init* that assigns to each control location $v \in V$ a convex data predicate $\mathit{init}(v)$, the *initial condition* of v; and also a labeling function *final* that assigns to each location $v \in V$ of A a data predicate $\mathit{final}(v)$, the *accepting condition* of v. The control of A may start in the location v only when the initial condition $\mathit{init}(v)$ is true. We write I_A for the *initial region* $\bigcup_{v \in V} \{(v, [\![\mathit{init}(v)]\!])\}$ of A; and we write F_A for the *accepting region* $\bigcup_{v \in V} \{(v, [\![\mathit{final}(v)]\!])\}$ of A. We use the initial and accepting conditions to analyze reachability problems of hybrid automata. In the graphical representation of hybrid automata, we suppress initial conditions of the form *false* and suppress accepting conditions of the form *true*.

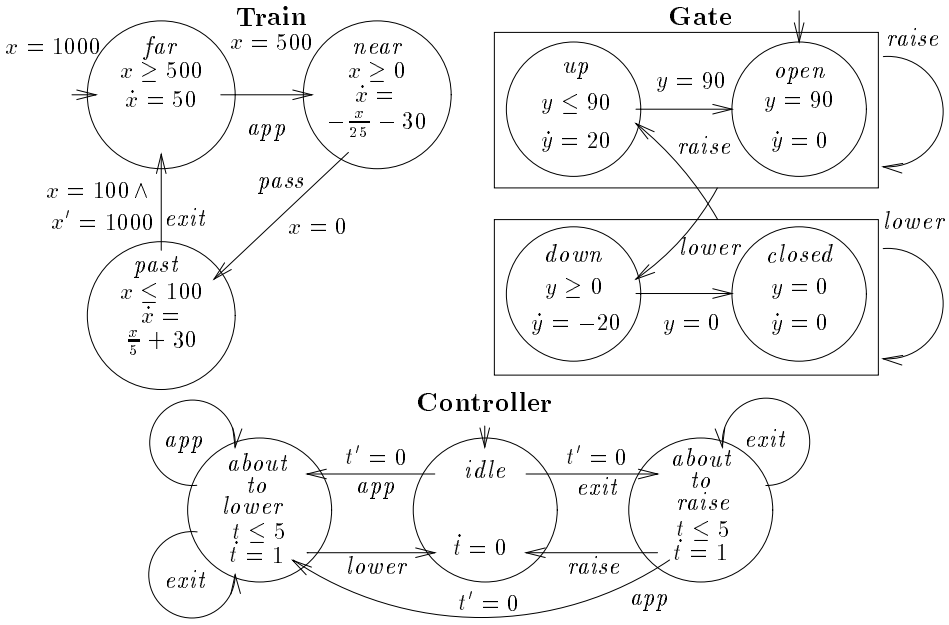

Figure 6.1: A railroad gate controller

6.1.1 Examples of Nonlinear Hybrid Automata

The first example of a nonlinear hybrid automaton is actually the thermostat automaton of Figure 2.1 in Chapter 2.

The second example is a nonlinear variation of the train-gate-controller example in Chapter 2 and 3. The three hybrid automata of Figure 6.1 model three processes—a train, a gate, and a controller. The nonlinear variable x represents the distance of the train from the gate.

Initially, the train is far from the gate and moves at the speed 50 meters per second. When the train approaches the gate, a sensor placed at a distance of 500 meters from the crossing detects the train and sends the signal *app* to the controller. The train starts to slow down following the differential equation $\dot{x} = -\frac{x}{25} - 30$. The controller waits 5 seconds before sending the command *lower* to the gate; the

delay of the controller is modeled by the clock t. Consequently, the gate is lowered from 90 radius degrees to 0 degrees at the constant rate of 20 degrees per second; the position of the gate in degrees is represented by variable y. After passing the gate, the train begins to accelerate following the differential equation $\dot{x} = \frac{x}{5} + 30$. A second sensor placed at 100 meters past the crossing detects the leaving train and signals *exit* to the controller, which, after another delay of 5 seconds, sends the command *raise* to the gate. The distance between consecutive trains is (at least) 1,000 meters.

6.1.2 Another Semantics for Hybrid Automata: Labeled Transition Systems

We have to define another semantics for hybrid systems in order to use the notion of bisimulation to discuss the equivalence of two hybrid automata. A hybrid automaton A defines a labeled transition system $\mathcal{S}_A$ with the state space Σ_A and the transition relation that consists of both the time steps $\overset{\Sigma_A}{\rightarrow}$ and the transition steps $\overset{\Sigma_A}{\mapsto}$. For simplicity and expressiveness, We write $\overset{\delta}{\mapsto}$ for the time step $\overset{\Sigma_A}{\rightarrow}$ corresponding to a witness data trajectory (δ, ρ) with the duration δ; and we write $\overset{\ell}{\mapsto}$ for the transition step $\overset{\Sigma_A}{\mapsto}$ corresponding to an automaton transition e with the synchronization label ℓ.

Formally, the hybrid automaton A defines the labeled transition system $\mathcal{S}_A = \langle \Sigma_A, I_A, \mathcal{L}, \rightarrow_A, F_A \rangle$ that consists of (1) the infinite state space Σ_A, (2) the set I_A of initial states, (3) the alphabet $\mathcal{L} = L \cup \mathbb{R}_{\geq 0}$, (4) the transition relation $\rightarrow_A = \bigcup\{\overset{\delta}{\mapsto} \mid \delta \geq 0\} \cup \bigcup\{\overset{\ell}{\mapsto} \mid \ell \in L\}$, and (5) the set F_A (accepting region) of accepting states. A *finite trajectory* τ of A is a finite path $\sigma_0 \overset{m_0}{\mapsto} \sigma_1 \overset{m_1}{\mapsto} \cdots \overset{m_{k-1}}{\mapsto} \sigma_k$ in $\mathcal{S}_A$ such that $\sigma_0 \in I_A$ and for all $i \in \{0, \ldots, k-1\}$, $(\sigma_i \overset{m_i}{\mapsto} \sigma_{i+1}) \in \rightarrow_A$. The finite trajectory τ is *accepting* if the final state of τ is accepting; that is, $\sigma_k \in F_A$. The *reachable region* $R(A) \subseteq \Sigma_A$ of the hybrid automaton A is the set of all final states on the

finite trajectories of A.

6.1.3 The Emptiness Problem

The *emptiness problem* for hybrid automata asks, given a hybrid automaton A, if A has an accepting finite trajectory. It is obvious that a reachability problem $(A, \varphi_I, \varphi_F)$ is equivalent to an emptiness problem for hybrid automata A with the initial region $I_A = [\![\varphi_I]\!]$ and the accepting region $F_A = [\![\varphi_F]\!]$.

For the thermostat automaton of Figure 2.1, we will verify the safety property whose "unsafe" region is characterized by the state predicate $\bigcup_{v \in \{on, off\}} \{(v, 6 \leq z \leq 2y - 1)\}$; that is, after 6 time units the heater has always been on at most half of the time plus 1 time unit. So we associate the thermostat automaton with the initial region $(on, x = 2 \wedge y = 0 \wedge z = 0)$ and the accepting region $\bigcup_{v \in \{on, off\}} \{(v, 6 \leq z \leq 2y - 1)\}$.

For the railroad gate controller of Figure 6.1, we will verify the safety property whose unsafe region is characterized by the state predicate $\bigcup_{v \not\models closed} \{(v, x \leq 100)\}$; that is, whenever the train is within 100 meters from the gate, the gate is closed (we write $v \models closed$ if the gate component of location v is *closed*). So we associate the railroad gate controller with the initial region $(far, open, idle)$ and the accepting region $\bigcup_{v \not\models closed} \{(v, x \leq 100)\}$. Both of the safety properties can be verified as emptiness problems.

6.2 Clock Translation

The clock translation of a hybrid automaton replaces each nonlinear data variable x by a clock t_x that is restarted whenever the value of x is changed by a discrete action. The clock translation is applicable if at every point of a finite trajectory, the value of x is uniquely determined by the value of t_x.

6.2.1 Solvable Automata

Let A be a hybrid automaton. The simple nonlinear data variable x of A is (*rationally*) *determined* in the location v of A if (1) variable x and the dotted variable $\dot{x}$ occurs in exactly one conjunct of the form $\dot{x} = f(x)$, denoted by $dif(v, x)$, (2) $dif(v, x) = f(x)$ and for all (rational) initial values $x_0 \in \mathbb{R}$, the initial-value problem "$\dot{x}(t) = f(x);\ x(0) = x_0$" has an algebraic solution $x_{v,x_0}(t)$ such that for each finite constant c that appears in an atomic data predicate $x \sim c$, for $\sim \in \{\leq, \geq\}$, or an atomic action predicate $x' = c$ of A, the function $x_{v,x_0}(t) - c$ has a finite number of (rational) roots. For example, suppose that the function $x_{v,x_0}(t) - c_0$ has the two roots r_0 and r_1, and all finite trajectories that enter the location v have the initial value x_0 for x. Then the value of the variable x is $x_{v,x_0}(t_x)$ for all reachable states in v. Thus an exit edge of location v guarded with $x = c_0$ can be replaced by two exit edges guarded with $t_x = r_0$ and $t_x = r_1$, respectively.

The location v of A is *definite* for the data variable x if the initial condition $init(v)$ implies $x = c$, for some *initial value* $c \in \mathbb{R}$. The edge e of A is *definite* for x if the action $act(e)$ implies $x' = c$, for some *arrival value* $c \in \mathbb{R}$. The *simple* nonlinear data variable x of A is *solvable* if the following three conditions hold:

1. For all locations v of A, x is determined in v.
2. All locations of A are definite for x.
3. For all edges $e = (v, v')$ of A, if $dif(v, x) \neq dif(v', x)$, then e is definite for x;

For example, the nonlinear variable x of the thermostat automaton of Figure 2.1 is solvable, and so is the the nonlinear variable x of the train automaton of Figure 6.1. The hybrid automaton A is *solvable* if all nonlinear data variables of A are simple and solvable. The automaton A is *rationally solvable* if A is (1) rational, (2) solvable, and (3) all data variables of A are rationally determined in all locations of A.

All (rational) timed automata are (rationally) solvable and the class of (rationally) solvable hybrid automata is closed under parallel composition. For each solvable data variable x, we collect the initial values of x for all locations and the arrival values of x for all definite edges in the finite set $\mathit{CritVal}(x) \subseteq \mathbb{R}$ of *critical values* for x.

6.2.2 The Clock Translation Algorithm

Given a solvable nonlinear hybrid automaton A, we construct a linear hybrid automaton A^c—the *clock translation* of A—by replacing each nonlinear data variable x with a new clock t_x. For each nonlinear data variable x, the construction proceeds in two steps:

1. Let $\mathit{CritVal}(x) = \{c_1, \ldots, c_n\}$ with $c_1 < \cdots < c_n$. Each location v of A is split into a collection $v_{c_1}, \ldots, v_{c_n}$ of locations, one for each critical value c_i of x. We then add the clock t_x such that the value of x in location v_{c_i} is $x(t_x)$, where $x(t)$ is the solution of the initial-value problem "$\dot{x}(t) = \mathit{dif}(v, x)$; $x(0) = c_i$".
2. All initial and accepting conditions, invariants, and actions are translated from conditions on x to conditions on t_x.

We now provide more details.

Step 1. Splitting Locations and Edges

After the application of Step 1, each new location v_{c_i} has the same activities as v and, in addition, the new activity $\mathit{dif}(v_{c_i}, t_x) = 1$ for the clock t_x. The new location v_{c_i} has the initial condition $\mathit{init}(v) \wedge t_x = 0$, the accepting condition $\mathit{final}(v)$, and the invariant $\mathit{inv}(v)$. The location v_{c_i} has the initial condition $\mathit{init}(v_{c_i}) = \mathit{init}(v) \wedge t_x = 0$ if $\mathit{inv}(v)$ implies $x = c_i$, and $\mathit{init}(v_{c_i}) = \mathit{false}$ otherwise. For each indefinite edge $e = (v, v')$, we introduce all edges of the form (v_{c_i}, v'_{c_i}) with the

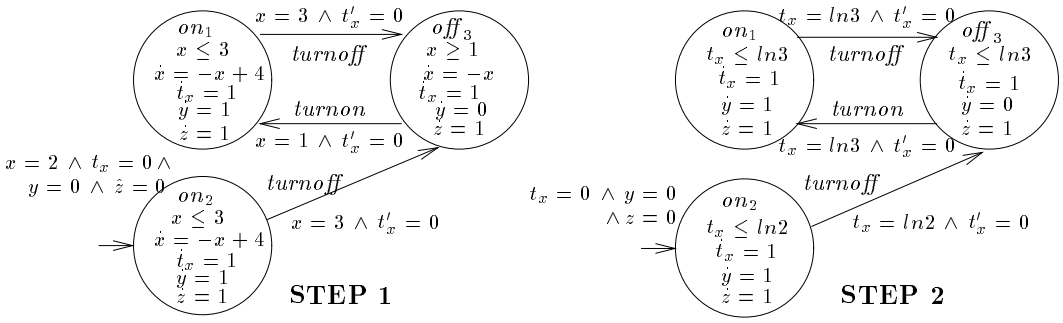

Figure 6.2: Clock translation of the thermostat automaton

action $act(e) \wedge t'_x = t_x$ and the label $label(e)$; for each definite edge $e = (v, v')$ with the arrival value c_j, we introduce all edges of the form (v_{c_i}, v'_{c_j}) with the action $act(e) \wedge t'_x = 0$ and the label $label(e)$.

For example, the thermostat automaton of Figure 2.1 has only definite edges. The critical values of x are 1, 2, and 3, so we split both locations *on* and *off* into three locations each. Since the locations on_3, off_1, and off_2 are not reachable by a sequence of automaton edges from the initial location on_2, we remove these three locations from the clock-translated automaton. The result of Step 1 is shown on the left in Figure 6.2.

Step 2. Updating Initial Conditions, accepting conditions, invariants, and actions

Let the function $x(t)$ be the solution of the initial-value problem "$\dot{x}(t) = dif(v, x)$; $x(0) = c_i$". We now eliminate the nonlinear variable x from the initial and accepting conditions, invariant, and exit-edge actions of each new location v_{c_i}.

First, we simply remove all the atomic data predicates that involve the variable x from the initial condition $init(v_{c_i})$. Second, we translate the accepting condition of location v_{c_i}. Suppose that $x \le c$ is an atomic data predicate of the accepting condition $final(v_{c_i})$ (other atomic data predicates are handled similarly). We find

all finite roots $r_0, \ldots, r_k$ of $x(t) - c$ (count roots with zero derivatives twice). If no such root exists and $c_i \leq c$, then $x \leq c$ is always satisfied and we replace $x \leq c$ by *true*; if no root exists and $c_i > c$, then $x \leq c$ is not satisfiable and we replace $x \leq c$ by *false*. Otherwise, if $c_i \leq c$, then $x \leq c$ is satisfied when the value of t_x is in any of the intervals $[0, r_0], [r_1, r_2], \ldots$; if $c_i > c$, then $x \leq c$ is satisfied when the value of t_x is in any of the intervals $[r_0, r_1], [r_2, r_3], \ldots$ We therefore replace $x \leq c$ by the disjunction $\bigvee_{[r_i, r_{i+1}] \in I} r_i \leq t_x \leq r_{i+1}$, where I is the set of *root intervals* during which $x \leq c$ is satisfied. The result can be transformed into disjunctive normal form.

Third, we translate the invariant of location v_{c_i}. Suppose that $x \leq c$ is a conjunct of the invariant $inv(v_{c_i})$ (other conjuncts are handled similarly). If $c_i > c$, then the arrival value of x does not satisfy the invariant, and thus we remove the location v_{c_i}. Otherwise, we find the smallest nonnegative finite root r of $x(t) - c$. If such a root r exists, then the automaton control can reside in the location v_{c_i} up to r time units, and thus we replace the conjunct $x \leq c$ of the invariant by the conjunct $t_x \leq r$. If no such root r exists, then the automaton control can reside in the location v_{c_i} forever, and we replace the conjunct $x \leq c$ by *true*.

Fourth, we translate the actions of all edges that leave the location v_{c_i}. Suppose that $x \leq c$ is a conjunct of the action $act(e)$, where $e = (v, v')$. We find all finite roots $r_0, \ldots, r_k$ of $x(t) - c$ (count roots with zero derivatives twice). If no such root exists and $c_i \leq c$, then the edge e is always enabled and we replace the conjunct $x \leq c$ by *true*; if no root exists and $c_i > c$, we remove the edge e. Otherwise, if $c_i \leq c$, then the edge e is enabled when the value of t_x is in any of the intervals $[0, r_0], [r_1, r_2], \ldots$; if $c_i > c$, then e is enabled when the value of t_x is in any of the intervals $[r_0, r_1], [r_2, r_3], \ldots$ For each root interval $[r_i, r_{i+1}]$ during which e is enabled, we introduce an edge with the action $act(e)$ and the label $label(e)$ except that (1) the conjunct $x \leq c$ is replaced by the conjunct $r_i \leq t_x \leq r_{i+1}$ and (2) any

atomic subformula involving x' is removed. In the thermostat example, we have $x_{on_1}(t) = -3e^{-t}+4$, $x_{on_2}(t) = -2e^{-t}+4$, and $x_{off_3}(t) = 3e^{-t}$. Consider the action $x = 3$ of the edge from on_2 to off_3. Since $ln2$ is the unique root of $-2e^{-t}+4-3$, it follows that $x = 3$ iff $t_x = ln2$. Hence we replace the action $x = 3$ with the action $t_x = ln2$. The final result of Step 2 is shown on the right in Figure 6.2.

6.2.3 Soundness, Completeness, and Decidability

We show that the clock translation is both sound and complete for checking the emptiness of solvable automata. Let A be a solvable hybrid automaton, and let A^c be the automaton clock translated from A by translating a nonlinear variable x into a clock t_x. We show that A and A^c are bisimilar.

We first recall the definition of (timed) bisimulation. Let $T_1 = \langle \Sigma_1, I_1, \mathcal{L}, \rightarrow_1 \rangle$ and $T_2 = \langle \Sigma_2, I_2, \mathcal{L}, \rightarrow_2 \rangle$ be two labeled transition systems. The binary relation $\approx \subseteq \Sigma_1 \times \Sigma_2$ is a *bisimulation* between T_1 and T_2 if for all states $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$, $\sigma_1 \approx \sigma_2$ implies for every letter $m \in \mathcal{L}$ that (1) if $\sigma_1 \stackrel{m}{\mapsto} \sigma_1'$, then there exists a state σ_2' such that $\sigma_2 \stackrel{m}{\mapsto} \sigma_2'$ and $\sigma_1' \approx \sigma_2'$; and (2) if $\sigma_2 \stackrel{m}{\mapsto} \sigma_2'$, then there exists a state σ_1' such that $\sigma_1 \stackrel{m}{\mapsto} \sigma_1'$ and $\sigma_1' \approx \sigma_2'$. The two states $\sigma \in \Sigma_1$ and $\sigma' \in \Sigma_2$ are *bisimilar* if there exists a bisimulation $\approx$ between T_1 and T_2 such that $\sigma \approx \sigma'$. The labeled transition systems T_1 and T_2 are *bisimilar*, denoted $T_1 \approx T_2$, if for each initial state $\sigma \in I_1$, there is a initial state $\sigma' \in I_2$ such that $\sigma \approx \sigma'$, and vice versa. The two hybrid automata A and B are *bisimilar* if $\mathcal{S}_A \approx \mathcal{S}_B$.

We define the function $\alpha_x \colon \Sigma_{A^c} \rightarrow \Sigma_A$ such that $\alpha_x(v_c, \vec{s}_1) = (v, \vec{s}_2)$, where the location v_c is split from the location v for the critical value c, the states $\vec{s}_1$ and $\vec{s}_2$ agree on all data variables except x and t_x, and $\vec{s}_2(x) = f_c(\vec{s}_1(t_x))$ if the function $f_c(t)$ is the solution of the initial-value problem "$\dot{x}(t) = dif(v, x)$; $x(0) = c$". To show that A and A^c are bisimilar, we will find the following two lemmas useful.

Lemma 3 *Let A be a solvable hybrid automaton, and let A^c be the clock translation of A that results from replacing the nonlinear variable x by the clock t_x. Then for all states $\sigma_1, \sigma_1' \in \Sigma_{A^c}$, if $\sigma_1 \stackrel{m}{\mapsto} \sigma_1'$ in $\mathcal{S}_{A^c}$ then $\alpha_x(\sigma_1) \stackrel{m}{\mapsto} \alpha_x(\sigma_1')$ in $\mathcal{S}_A$.*

Proof. We consider the time step first. Assume that the function $f_c(t)$ is the solution of the initial-value problem "$\dot{x}(t) = dif(v, x)$; $x(0) = c$". Let $\sigma_1 = (v_c, \vec{s}_1)$ and $\sigma_1' = (v_c, \vec{s}_1')$ be two states in Σ_{A^c}, and let $\alpha_x(\sigma_1) = \sigma_2 = (v, \vec{s}_2)$ and $\alpha_x(\sigma_1') = \sigma_2' = (v_c, \vec{s}_2')$ be two states in Σ_A, where the location v_c is split from the location v for the critical value c.

Suppose that $\vec{s}_1(t_x) = t_a$ and $\sigma_1 \stackrel{\delta}{\mapsto} \sigma_1'$ for some witness data trajectory (δ, ρ) with duration $\delta \geq 0$. We shall show that $\sigma_2 \stackrel{\delta}{\mapsto} \sigma_2'$. First notice that $\vec{s}_2(x) = f_c(t_a)$ and $\vec{s}_2'(x) = f_c(\delta + t_a)$. Let (δ, ρ') be the data trajectory such that for all $t \in [0, \delta]$, $(\rho'(t))(x) = f_c(t + t_a)$ and $(\rho'(t))(y) = (\rho(t))(y)$ for all data variables $y \neq x$. We claim that the data trajectory (δ, ρ') is a witness of $\sigma_2 \stackrel{\delta}{\mapsto} \sigma_2'$. It is clear that $\rho'(0) = \sigma_2$ and $\rho'(\delta) = \sigma_2'$, so it remains to be shown that (δ, ρ') is an admissible data trajectory in the location v. According to the clock translation of the invariant $inv(v_c)$, for all $t \in [0, \delta]$, the state $\rho'(t) \in [\![inv(v)]\!]$. Since the function $f_c(t)$ is the solution of the initial-value problem "$\dot{x}(t) = dif(v, x)$; $x(0) = c$", for all $t \in [0, \delta]$, $f_c(t + t_a) \in [\![dif(v, x)]\!]$. Moreover, since $\rho'(t)(y) = \rho(t)(y)$, for all data variables $y \neq x$ and for all $t \in [0, \delta]$, we have $\rho'(t) \in [\![dif(v)]\!]$. Thus (δ, ρ') is admissible.

Now let us consider the transition step. Let $\sigma_1 = (v_c, \vec{s}_1)$ and $\sigma_1' = (u_d, \vec{s}_1')$ be two states in Σ_{A^c}, and let $\alpha_x(\sigma_1) = \sigma_2 = (v, \vec{s}_2)$ and $\alpha_x(\sigma_1') = \sigma_2' = (u, \vec{s}_2')$, where the location v_c is split from the location v for the critical value c and the location u_d is split from the location u for the critical value d. Suppose that $\sigma_1 \stackrel{\ell}{\mapsto} \sigma_1'$, where $\ell = syn(v_c, u_d)$.

According to the clock translation of the transitions of A^c, there must be a transition (v, u) of A such that $syn(v, u) = syn(v_c, u_d) = \ell$. Also assume that

the function $f_c(t)$ is the solution of the initial-value problem "$\dot{x}(t) = dif(v, x)$; $x(0) = c$" and the function $g_d(t)$ is the solution of the initial-value problem "$\dot{x}(t) = dif(u, x)$; $x(0) = d$". We now show that $\sigma_2 \stackrel{\ell}{\mapsto} \sigma_2'$ for the following two cases:

1. $act(v_c, u_d)$ implies $t_x' = 0$. In this case, according to the clock translation, $act(v, u)$ implies $x' = d$. After the reset of the variable x, we have $\vec{s}_1'(t_x) = 0$. Since $\vec{s}_2'(x) = g_d(\vec{s}_1'(t_x)) = g_d(0) = d$, it is clear that $\sigma_2 \stackrel{\ell}{\mapsto} \sigma_2'$.

2. $act(v_c, u_d)$ implies $t_x' = t_x$. In this case, according to the clock translation, $act(v, u)$ implies $x' = x$ and $dif(v, x) = dif(u, x)$. Then $\vec{s}_2'(x) = \vec{s}_2(x) = f_c(\vec{s}_1(t_x)) = f_c(\vec{s}_1'(t_x))$. Thus $\sigma_2 \stackrel{\ell}{\mapsto} \sigma_2'$.

The proof is complete. ■

Lemma 4 *Let A be a solvable hybrid automaton, and let A^c be the clock translation of A that results from replacing the nonlinear variable x by the clock t_x. Then for all states $\sigma_2, \sigma_2' \in \alpha_x(\Sigma_{A^c})$, if $\sigma_2 \stackrel{m}{\mapsto} \sigma_2'$ in $\mathcal{S}_A$ and $\sigma_2 = \alpha_x(\sigma_1)$, then there exists a state σ_1' in Σ_A^c such that $\sigma_1 \stackrel{m}{\mapsto} \sigma_1'$ and $\sigma_2' = \alpha_x(\sigma_1')$.*

Proof. We consider the time step first. Suppose that σ_2 and σ_2' are both in the image $\alpha_x(\Sigma_A)$ of the function α_x and $\sigma_2 \stackrel{\delta}{\mapsto} \sigma_2'$ for some witness data trajectory (δ, ρ) with duration $\delta \geq 0$. For a constant d, define the function $f_d(t)$ to be the solution of the initial-value problem "$\dot{x}(t) = dif(v, x)$; $x(0) = d$".

Let σ_1 be any state of A^c such that $\alpha_x(\sigma_1) = \sigma_2$. Define (δ, ρ') be the data trajectory such that for all $t \in [0, \delta]$, $\rho'(t) = \alpha_x(\rho(t))$. According to the clock translation of the invariant $inv(v_c)$, the data trajectory (δ, ρ') is admissible in location v_c. Suppose that $\sigma_1(t_x) = t_a$ and thus $\sigma_1'(t_x) = t_a + \delta$. It remains to be shown that $\sigma_2' = \alpha_x(\sigma_1')$. Since the duration of the finite trajectories (δ, ρ) and (δ, ρ') are the same, for all $t \in [0, \delta]$, and for all variables $y \neq x$, we have $\sigma_1'(t)(y) = \sigma_2'(t)(y)$. We now show that $\sigma_2'(t)(y) = f_c(t_a + \delta)$. Since $\sigma_2 \stackrel{\delta}{\mapsto} \sigma_2'$ and

$f_c(t_a) = \sigma_2(x)$, we have

$$\begin{aligned}
\sigma_2'(x) &= \sigma_2(x) + \int_0^{\delta} f'_{\sigma_2(x)}(t)\ dt \\
&= f_c(t_a) + \int_0^{\delta} f'_{f_c(t_a)}(t)\ dt \\
&= f_c(t_a) + \int_{t_a}^{t_a+\delta} f_c'(t)\ dt \\
&= f_c(t_a) + (f_c(t_a+\delta) - f_c(t_a)) \\
&= f_c(t_a+\delta).
\end{aligned}$$

Notice that $f_{f_c(t_a)}$ is the solution of the initial-value problem "$\dot{x}(t) = dif(v,x)$; $x(0) = f_c(t_a)$" and $f'_{f_c(t_a)}$ is the first derivative of the function $f_{f_c(t_a)}$.

Now let us consider the transition step. We suppose $\sigma_2 \stackrel{\ell}{\mapsto} \sigma_2'$, where $\ell = syn(v,u)$. Again, we have to consider two cases:

1. $act(v,u)$ implies $x' = d$. In this case, according to the clock translation, there must be a location u_d split from the location u and transitions of the form (v_c, u_d) with the action predicate $act(v_c,u_d) = act(v,u) \wedge t_x' = 0$ and the label $syn(v_c,u_d) = \ell$. Let $\sigma_1 = (v_c, \vec{s}_1)$ be any state such that $\alpha_x(\sigma_1) = \sigma_2$. We pick the state $\sigma_1' = (u_d, \vec{s}_1')$ such that $\sigma_1'(t_x) = 0$ and $\sigma_1'(y) = \sigma_2'(y)$ for all data variables $y \neq x$. Let g_d be the solution of the initial-value problem "$\dot{x}(t) = dif(\hat{u}_d, x)$; $x(0) = d$". Since $\sigma_2'(x) = g_d(0) = d$, and $\sigma_2'(y) = \sigma_1'(y)$ for all variables $y \neq x$, the state σ_1' is admissible and $\sigma_1 \stackrel{\ell}{\mapsto} \sigma_1'$.

2. $act(v,u)$ implies $x' = x$. In this case, according to the clock translation, there must be a location u_c split from the location u and transitions of the form (v_c, u_c) with the action predicate $act(v_c,u_c) = act(v,u) \wedge t_x' = t_x$ and the label $syn(v_c,u_c) = \ell$. Moreover, $dif(v_c) = dif(u_c)$. Let $\sigma_1 = (v_c, \vec{s}_1)$ be any state such that $\alpha_x(\sigma_1) = \sigma_2$. We pick the state $\sigma_1' = (u_d, \vec{s}_1')$ such that $\sigma_1'(t_x) = \sigma_1(t_x)$ and $\sigma_1'(y) = \sigma_2'(y)$ for all data variables $y \neq x$. Then it is obivious that $\alpha_x(\sigma_1') = \sigma_2'$ and $\sigma_1 \stackrel{\ell}{\mapsto} \sigma_1'$.

The proof is complete. ■

Now we are ready to show that the relation $\{(\sigma, \alpha_x(\sigma)) \mid \sigma \in \Sigma_{A^c}\}$ is a bisimulation between $\mathcal{S}_A$ and $\mathcal{S}_{A^c}$, and thus $\mathcal{S}_A \approx \mathcal{S}_{A^c}$.

Proposition 8 *Let A be a solvable hybrid automaton, and let A^c be the clock translation of A that results from replacing the nonlinear variable x by the clock t_x. then A and A^c are bisimilar.*

Proof. Suppose that I_A and I_{A^c} are the initial regions of A and A^c respectively. Again, we define the function $\alpha_x : \Sigma_{A^c} \to \Sigma_A$ such that $\alpha_x(v_c, \vec{s}) = (v, \vec{s'})$, where the location v_c is split from the location v for the critical value c, the states $\vec{s}$ and $\vec{s'}$ agree on all data variables except x and t_x, and $\vec{s'}(x) = f_c(\vec{s}(t_x))$ if the function $f_c(t)$ is the solution of the initial-value problem "$\dot{x}(t) = dif(v, x)$; $x(0) = c$".

According to the clock translation, it is clear that $\alpha_x(I_{A^c}) = I_A$; that is, α_x is onto. Suppose the value of the variable x is c in an initial state $(v, \vec{s})$. Then among the locations split from the location v, only $init(v_c) = (t_x = 0)$ is not *false*. Thus α_x is also bijection (one-one and onto) between I_A and I_{A^c}.

Now it remains to show that the binary relation $\approx = \{(\alpha_x(\sigma), \sigma) \mid \sigma \in \Sigma_{A^c}\} \subseteq \Sigma_A \times \Sigma_{A^c}$ is a bisimulation between $\mathcal{S}_A$ and $\mathcal{S}_{A^c}$. By Lemma 3 and Lemma 4, it is clear that for all states $\sigma_1 \in \Sigma_{A^c}$ and $\sigma_2 \in \Sigma_A$, $\sigma_1 \approx \sigma_2$ implies for every letter $m \in \mathcal{L}$ that (1) if $\sigma_1 \stackrel{m}{\mapsto} \sigma_1'$, then there exists a state σ_2' such that $\sigma_2 \stackrel{m}{\mapsto} \sigma_2'$ and $\sigma_1' \approx \sigma_2'$; and (2) if $\sigma_2 \stackrel{m}{\mapsto} \sigma_2'$, then there exists a state σ_1' such that $\sigma_1 \stackrel{m}{\mapsto} \sigma_1'$ and $\sigma_1' \approx \sigma_2'$. In other wards, the binary relation $\approx = \{(\alpha_x(\sigma), \sigma) \mid \sigma \in \Sigma_{A^c}\}$ is a bisimulation between $\mathcal{S}_A$ and $\mathcal{S}_{A^c}$. ■

Since bisimilarity is transitive, by Proposition 8, if A^c results from A by replacing several nonlinear variables with clocks, A and A^c are still bisimilar.

Theorem 5 *If A is a solvable hybrid automaton and A^c is a clock translation of A, then A and A^c are bisimilar.*

It follows from Theorem 5 that the clock translation is sound and complete for all branching-time properties. In particular, for safety properties, we have the following corollary.

Corollary 1 *Let A be a solvable hybrid automaton, and let A^c be a clock translation of A. Then A has an accepting finite trajectory iff A^c has an accepting finite trajectory.*

We conclude that for solving the emptiness problem for the nonlinear automaton A, it suffices to solve the emptiness problem for the linear automaton A^c. The emptiness problem for A^c, however, can be solved exactly only if the clock translation A^c is rational. This gives us the following decidability result, which covers a class of nonlinear hybrid automata, while all previously published decidability results refer to linear hybrid automata [HKPV95].

Corollary 2 *The emptiness problem is decidable for rationally solvable hybrid automata.*

6.2.4 δ-approximate Clock Translation

If the clock translation A^c is not rational, we approximate A^c by a rational automaton, and show soundness for emptiness checking. To preserve soundness when approximating irrational roots numerically, we *over*-approximate all root intervals. For example, the action $t_x = ln2$ can be overapproximated by the rational data predicate $693 \leq 1000t_x \leq 694$ with an error bounded by $\delta = 1/1000$. Formally, a *δ-approximation* of the data predicate $t_x \leq c$ ($t_x \geq c$), for $\delta \in \mathbb{R}$, is of the form $t_x \leq c'$ ($t_x \geq c'$) such that $c \leq c' \leq c + \delta$ ($c - \delta \leq c' \leq c$) and c' is rational. A *δ-approximate clock translation* $[A^c]_\delta$ of A is a rational linear hybrid automaton that is obtained from the clock translation A^c by replacing all atomic data predicates in initial and accepting conditions, invariants, and actions by δ-approximations.

Note that an action predicate that involves the primed variable t'_x for a new clock t_x are of the form $t'_x = 0$ or $t'_x = t_x$. So the predicates that contain the primed variable t'_x are not relaxed.

We show that $[A^c]_\delta$ simulates A. To see this, recall the definition of (timed) simulation. Let $T_1 = \langle \Sigma_1, I_1, \mathcal{L}, \rightarrow_1 \rangle$ and $T_2 = \langle \Sigma_2, I_2, \mathcal{L}, \rightarrow_2 \rangle$ be two labeled transition systems. The binary relation $\succeq \subseteq \Sigma_1 \times \Sigma_2$ is a *simulation* of T_2 by T_1 if for all states $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$, $\sigma_1 \succeq \sigma_2$ implies for every letter $m \in \mathcal{L}$ that if $\sigma_2 \stackrel{m}{\mapsto} \sigma'_2$, then there exists a state σ'_1 such that $\sigma_1 \stackrel{m}{\mapsto} \sigma'_1$ and $\sigma'_1 \succeq \sigma'_2$. The state $\sigma \in \Sigma_1$ *simulates* the state $\sigma' \in \Sigma_2$ if there exists a simulation $\succeq$ of T_2 by T_1 such that $\sigma \succeq \sigma'$. The labeled transition system T_1 *simulates* the labeled transition system T_2, denoted $T_1 \succeq T_2$, if each initial state of T_2 is simulated by an initial state of T_1. The hybrid automaton A *simulates* the hybrid automaton B, written $A \succeq B$, if $\mathcal{S}_A \succeq \mathcal{S}_B$. It is clear that $[A^c]_\delta$ simulates A^c. Moreover, since A and A^c are bisimilar, we know that $[A^c]_\delta$ simulates A.

Proposition 9 *Let A be a solvable hybrid automaton. For all $\delta \in \mathbb{R}_{\geq 0}$, if $[A^c]_\delta$ is a δ-approximate clock translation of A, then $[A^c]_\delta$ simulates A.*

Proof. Define the binary relation $\succeq = \{(\sigma, \sigma) \mid \sigma \in \Sigma_A\} \subseteq \Sigma_{[A^c]_\delta} \times \Sigma_A$. Since the δ-approximate clock translation of A, $[A^c]_\delta$, is obtained by relaxing the initial and accepting conditions, invariants, and actions by δ-approximations, it is apparently that if $\sigma \stackrel{m}{\mapsto} \sigma'$ in A then $\sigma \stackrel{m}{\mapsto} \sigma'$ in $[A^c]_\delta$. Therefore, the binary relation $\succeq$ is a simulation for all the initial states of A. In other words, each initial state σ of A is simulated by the initial state σ of $[A^c]_\delta$. ■

It follows that approximate clock translation is sound for all linear-time properties. Again, for safety properties, we have the following corollary.

Corollary 3 *Let A be a solvable hybrid automaton and let $[A^c]_\delta$ be a δ-approximate clock translation of A. If A has an accepting finite trajectory, then so does $[A^c]_\delta$.*

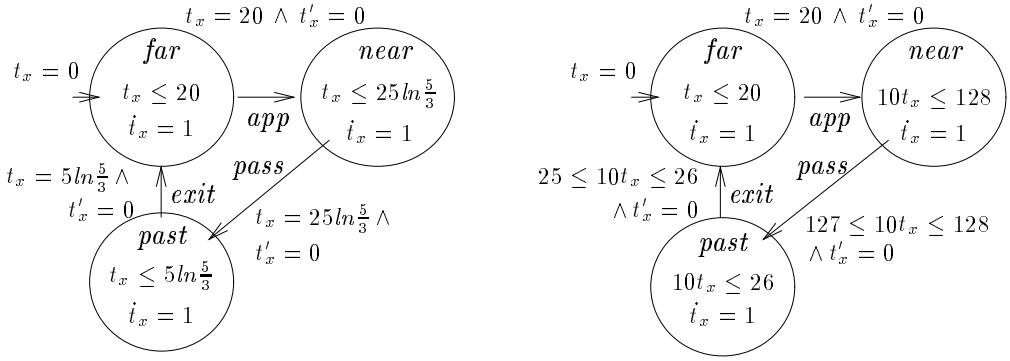

Figure 6.3: Clock translation and 0.1-approximate clock translation of the train automaton

6.2.5 Example: Railroad Gate Controller

The clock translation B_1^c of the train automaton B_1 from Figure 6.1 is shown on the left in Figure 6.3, next to a 0.1-approximate clock translation $[B_1]_{0.1}$ on the right.

By taking the product of $[B_1]_{0.1}$ with the gate and controller automata from Figure 6.1, the HyTech verifier automatically checks (in 25 seconds of CPU time) that whenever the train is within 100 meters from the gate, then the gate is closed. On the other hand, a 1.0-approximate clock translation of the train automaton is not sufficient for proving this safety property. (After clock translation, the safety property of the thermostat automaton from Figure 2.1 is checked by HyTech in 7 seconds of CPU time.)

6.2.6 Error Analysis

The approximate clock translation $[A^c]_\delta$ may have an accepting finite trajectory even if A does not. We now show that there is a hybrid automaton that is very close to A and yet also has an accepting finite trajectory. The following error analysis

relaxes all atomic data and action predicates of A to provide an upper bound on the error of $[A^c]_\delta$. If the hybrid automaton A models a hybrid system with sensors and actuators, then the *ε-relaxed automaton* A^ε, for $\varepsilon \in \mathbb{R}_{\geq 0}$, models the same system with sensors and actuators that suffer from errors bounded by ε: (1) all atomic data predicates of the form $x \leq c$ and $x \geq c$ (in initial and accepting conditions, invariants, and actions) are replaced by $x \leq c + \varepsilon$ and $x \geq c - \varepsilon$, respectively; and (2) all atomic action predicates of the form $x' = c$ are replaced by $c - \varepsilon \leq x' \leq c + \varepsilon$. Notice that if $\varepsilon \to 0$, then $A^\varepsilon \to A$. We define the metric $d_\succeq$ on hybrid automata such that $d_\succeq(A, B)$ is the infimum of all nonnegative reals ε such that $A^\varepsilon \succeq B \succeq A$ or $B^\varepsilon \succeq A \succeq B$, if such an ε exists; otherwise, $d_\succeq(A, B) = \infty$. The *error* of the δ-approximate clock translation $[A^c]_\delta$ is $d_\succeq(A, [A^c]_\delta)$. The following lemma bounds this error from above.

Lemma 5 *Let A be a solvable hybrid automaton, and let $[A^c]_\delta$ be a δ-approximate clock translation that results from translating the data variable x of A. Suppose $\lambda \in \mathbb{R}_{\geq 0}$ bounds the absolute values of the derivatives of all data variables of A in all locations of A. Let $\sigma_1 = (v, \vec{s}_1)$ be a state of $[A^c]_\delta$ and $\sigma_2 = \alpha_x(\sigma_1) = (v_c, \vec{s}_2)$ be an admissible state of A. If $d - \delta \leq \sigma_1(t_x) \leq d + \delta$, then $f_c(d) - \delta \cdot \lambda \leq \sigma_2(x) \leq f_c(d) + \delta \cdot \lambda$, where the function $f_c(t)$ is the solution of the initial-value problem "$\dot{x}(t) = dif(v, x)$; $x(0) = c$".*

Proof. We assume that $t_x = d + \epsilon$ for some non-negative constant $0 \leq \epsilon \leq \delta$. Then we have the following:

$$\begin{aligned} \sigma_2(x) &= f_c(t_x) \\ &= f_c(d + \epsilon) \\ &= f_c(d) + \int_d^{d+\epsilon} f_c'(t)\, dt. \end{aligned}$$

For all $t \in [d, d+\delta]$, we know that $|f_c'(t)| \leq \lambda$. Thus $-\delta \cdot \lambda \leq \int_d^{d+\epsilon} f_c'(t)\ dt \leq \delta \cdot \lambda$. Therefore, $f_c(d) - \epsilon \cdot \lambda \leq \sigma_2(x) \leq f_c(d) + \epsilon \cdot \lambda$.

If $t_x = d - \epsilon$ for some non-negative constant $0 \leq \epsilon \leq \delta$, by the similar arguments, we can show that $f_c(d) - \epsilon \cdot \lambda \leq \sigma_2(x) \leq f_c(d) + \epsilon \cdot \lambda$.

Since $d - \delta \leq \sigma_1(t_x) \leq d + \delta$, we have shown that $f_c(d) - \delta \cdot \lambda \leq \sigma_2(x) \leq f_c(d) + \delta \cdot \lambda$. ■

Lemma 6 *Let A be a solvable hybrid automaton, and let $[A^c]_\delta$ be a δ-approximate clock translation that results from translating the data variable x of A. If $\lambda \in \mathbb{R}_{\geq 0}$ bounds the absolute values of the derivatives of all data variables of A in all locations of A, then $A^{\delta \cdot \lambda}$ simulates $[A^c]_\delta$; that is, the error of the δ-approximate clock translation $[A^c]_\delta$ is bounded by $\delta \cdot \lambda$.*

Proof. We define the function $\alpha_x : \Sigma_{[A^c]_\delta} \to \Sigma_A$ such that $\alpha_x(v_c, \vec{s}_1) = (v, \vec{s}_2)$, where the location v_c is split from the location v for the critical value c, the states $\vec{s}_1$ and $\vec{s}_2$ agree on all data variables except x and t_x, and $\vec{s}_2(x) = f_c(\vec{s}_1(t_x))$ if the function $f_c(t)$ is the solution of the initial-value problem "$\dot{x}(t) = dif(v, x)$; $x(0) = c$". We claim that the relation $\{(\alpha_x(\sigma), \sigma) \mid \sigma \in \Sigma_{[A^c]_\delta}\}$ is a simulation of $\mathcal{S}_{[A^c]_\delta}$ by $\mathcal{S}_{A^{\delta \cdot \lambda}}$ for every initial state of $[A^c]_\delta$.

We first argue that $\alpha_x(\sigma)$ is an admissible state of $A^{\delta \cdot \lambda}$ for any admissible state σ of $[A^c]_\delta$. The conjunct of the form $t_x \leq d + \epsilon$ in $inv(v_c)$ is clock-translated from a conjunct $x \sim f_c(d)$ in $inv(v)$ and then δ-relaxed by $\epsilon \leq \delta$. Since $\sigma(t_x) \leq d + \epsilon$, by Lemma 5, $f_c(d) - \delta \cdot \lambda \leq \alpha_x(\sigma)(x) \leq f_c(d) + \delta \cdot \lambda$. Thus $\alpha_x(\sigma)$ is admissible if σ is admissible.

Next we consider the time step. Let $\sigma_1 = (v_c, \vec{s}_1)$ and $\sigma_1' = (v_c, \vec{s}_1')$ be two states in $\Sigma_{[A^c]_\delta}$, and let $\alpha_x(\sigma_1) = \sigma_2 = (v, \vec{s}_2)$ and $\alpha_x(\sigma_1') = \sigma_2' = (v_c, \vec{s}_2')$.

Suppose that $\sigma_1 \overset{\epsilon}{\mapsto} \sigma_1'$ for some witness data trajectory (ϵ, ρ) with duration $\epsilon \geq 0$. Let (ϵ, ρ') be the data trajectory such that for all $t \in [0, \epsilon]$, $\rho'(t) = \alpha_x(\rho(t))$.

We claim that the data trajectory (ϵ, ρ') is a witness of $\sigma_2 \stackrel{\epsilon}{\mapsto} \sigma_2'$. Since f_c satisfies the rate predicate $dif(v)$ of A, and since the rate predicate $dif(v)$ of A and the rate predicate $dif(v)$ of $A^{\delta \cdot \lambda}$ are the same, the data trajectory (ϵ, ρ') satisfies the rate predicate $dif(v)$ of $A^{\delta \cdot \lambda}$. In addition, every state $\rho'(t)$ is admissible since the state $\rho(t)$ is admissible. Thus the data trajectory (ϵ, ρ') is a witness of $\sigma_2 \stackrel{\epsilon}{\mapsto} \sigma_2'$.

Now we consider the transition step. Let $\sigma_1 = (v_c, \vec{s}_1)$ and $\sigma_1' = (u_d, \vec{s}_1')$ be two states in $\Sigma_{[A^c]_\delta}$, and let $\alpha_x(\sigma_1) = \sigma_2 = (v, \vec{s}_2)$ and $\alpha_x(\sigma_1') = \sigma_2' = (u, \vec{s}_2')$, where the location v_c is split from the location v for the critical value c and the location u_d is split from the location u for the critical value d. Suppose that $\sigma_1 \stackrel{\ell}{\mapsto} \sigma_1'$, where $\ell = syn(v_c, u_d)$. Since $\vec{s}_1$ and $\vec{s}_1'$ satisfies the action predicate $act(v_c, u_d)$, by Lemma 5, $\vec{s}_2$ and $\vec{s}_2'$ satisfies the action predicate $act(v, u)$. In addition, by Lemma 5, both $\vec{s}_2$ and $\vec{s}_2'$ are admissible. This completes the proof. ■

Lemma 6 shows the error of a δ-approximate clock translation $[A^c]_\delta$ that results from translating a data variable of A is bounded by $\delta \cdot \lambda$. It is clear that the same argument applies to a δ-approximate clock translation $[A^c]_\delta$ that results from translating several data variables of A. It follows that the approximate clock translation is asymptotically complete, under the metric $d_{\succeq}$, for checking the emptiness of hybrid automata.

Theorem 6 *Let A be a solvable hybrid automaton. For all reals $\varepsilon > 0$, there is a real $\delta > 0$ such that for all δ-approximate clock translations $[A^c]_\delta$ of A, $A^\varepsilon \succeq [A^c]_\delta \succeq A$.*

Proof. Take $\delta < \frac{\varepsilon}{\lambda}$ where λ bounds the absolute values of the derivatives of all data variables of A in all locations of A. Notice that by the definition of hybrid automata, λ is finite. From Lemma 6, the result follows. ■

Corollary 4 *Let A be a solvable hybrid automaton. For all reals $\varepsilon > 0$, there is a real $\delta > 0$ such that if a δ-approximate clock translation of A has an accepting*

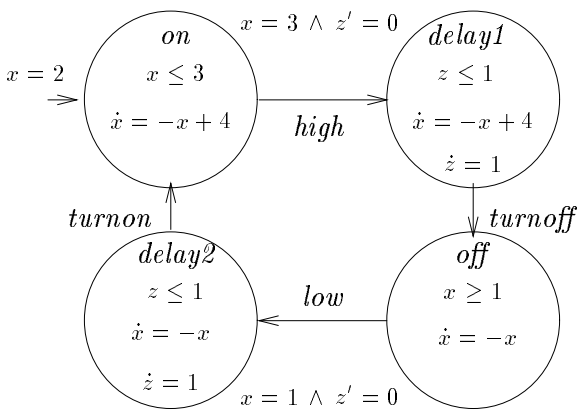

Figure 6.4: A temperature controller with delays

finite trajectory, then so does A^{ε}.

The corollary means that whenever the approximate clock translation $[A^c]_\delta$ gives answer *no*, then there is a problem (B, φ_F) such that $A^{\delta \cdot \lambda} \succeq B$ and (B, φ_F) has answer *no*.

6.3 Rate Translation

The rate translation of a hybrid automaton replaces each nonlinear data variable x by a piecewise-linear variable that approximates x. The rate translation may be applicable also to unsolvable automata.

Consider, for example, the nonlinear hybrid automaton of Figure 6.4, which models a temperature controller with delays: after the thermometer detects that the temperature is low or high, there may be a delay of up to 1 time unit before the heater is turned on or off. We wish to verify that the plant temperature is always between $\frac{1}{5}$ and $\frac{19}{5}$. The automaton is not solvable, because the edge from $delay_1$ to *off* is indefinite for x. Hence we cannot apply the clock translation to eliminate the nonlinear variable x. Instead, we approximate x by a piecewise-linear variable. In

location v with the bounded invariant region $[\![inv(v)]\!]$, we bound the derivative of x by its minimum a and its maximum b, and then replace the activity $dif(v, x)$ by the rate interval $[a, b]$. For a better approximation, we split the location v into several locations and limit the size of the rate intervals. Clearly, smaller rate intervals yield a more accurate overapproximation of the automaton finite trajectories.

6.3.1 Bounded Automata

Let A be a hybrid automaton. The data variable x of A is *nondecreasing* (*nonincreasing*) in location v of A if $inv(v)$ implies that the slope of variable x is nonnegative (nonpositive). The data variable x of A is *bounded* with the *window* $[c, d] \subseteq \mathbb{R}$ if any one of the following three conditions holds:

1. For all states in $R(A)$, the value of x is always within the bounded interval $[c, d]$. In particular, this is the case if for all locations v of A, $inv(v)$ implies $c \leq x \leq d$.
2. A does not contain constants smaller than c or larger than d, and either x is nondecreasing in all locations of A, or x is nonincreasing in all locations.
3. $\neg F_A$ implies $c \leq x \leq d$.

For example, the nonlinear variable x in the temperature controller is bounded with the window $[\frac{1}{5}, \frac{19}{5}]$ because of the third condition. If the data variable x is bounded with the window $[c, d]$, and its value lies outside $[c, d]$, then the exact value of x is irrelevant for checking the emptiness of A. The hybrid automaton A is *bounded* if all nonlinear variables of A are simple and bounded.

6.3.2 The Rate Translation Algorithm

Let A be a bounded hybrid automaton, and let $\delta \in \mathbb{R}_{\geq 0}$ be a nonnegative real. A *δ-approximate rate translation* $[A^r]_\delta$ of A is a linear hybrid automaton that is

obtained by the following construction. Consider a data variable x and a location v of A. First assume that the activity $dif(v,x) = f(x)$ is a function of x only. Let $[c,d]$ be the window of x. We partition the window $[c,d]$ into k subintervals $I_1 = [c_0 = c, c_1], \ldots, I_k = [c_{k-1}, c_k = d]$, each of size at most δ. The location v is split into $k+2$ locations $v_0, \ldots, v_{k+1}$. Each new location v_i has the invariant $inv(v) \wedge c_{i-1} \leq x \leq c_i$, where $c_{-1} = -\infty$ and $c_{k+1} = \infty$. For each v_i, we compute the minimum a and the maximum b of the function $dif(v_i,x)$ for $c_{i-1} \leq x \leq c_i$. We then approximate the derivative of x in the location v_i by the rate interval $dif(v_i,x) = [a,b]$. Finally, we introduce all edges of the form (v_i, v_{i+1}) and (v_{i+1}, v_i) with the action $x = c_i$ and the label v (which is the label of the stutter edge e_v); and for each edge $e = (v,v')$, all edges of the form (v_i, v'_j) with the action $act(e)$ and the label $label(e)$.

Now consider the general case that $dif(v,x_i) = f(x_1, \ldots, x_n)$. We approximate all nonlinear variables $x_1, \ldots, x_n$ simultaneously. Suppose that the window for x_i is I_i. We partition I_i into k_i subintervals $[c_{i,0}, c_{i,1}], \ldots, [c_{i,k_i-1}, c_{i,k_i}]$, each of size at most δ. The location v is split into the set $U_v = \{v(a_1, \ldots, a_n) \mid 0 \leq a_i \leq k_i + 1\}$ of $(k_1+2)\cdots(k_n+2)$ locations, all with the initial condition $init(v)$ and the accepting condition $final(v)$. The invariant of $v(a_1, \ldots, a_n)$ is $inv(v) \wedge \bigwedge_{i=1,\ldots,n} c_{i,a_i-1} \leq x_i \leq c_{i,a_i}$, where $c_{i,-1} = -\infty$ and $c_{i,k_i+1} = \infty$. For each new location, we compute the rate intervals for all x_i. For each pair $v(\vec{a})$ and $v(\vec{b})$ of new locations, we introduce all edges of the form $(v(\vec{a}), v(\vec{b}))$ with the axtion $\vec{x}' = \vec{x}$ and the label v (many of these edges are inconsistent and can be omitted); and for each edge $e = (v,v')$, we introduce all edges of the form $(v(\vec{a}), v'(\vec{b}))$ with the action $act(e)$ and the label $label(e)$.

6.3.3 Soundness

We show that the rate translation is sound for checking the emptiness of bounded automata.

Proposition 10 *Let A be a bounded hybrid automaton. For all $\delta \in \mathbb{R}_{\geq 0}$, if $[A^r]_\delta$ is a δ-approximate rate translation of A, then $[A^r]_\delta$ simulates A.*

Proof. We define the onto function $\beta : \Sigma_{[A^r]_\delta} \to \Sigma_A$ such that $\beta(v', \vec{s}) = (v, \vec{s})$ if $v' \in U_v$. Notice that (1) the automaton $[A^r]_\delta$ and the automaton A have exactly the same transitions, so $\sigma \stackrel{\ell}{\mapsto} \sigma'$ in $\mathcal{S}_A$ iff $\sigma \stackrel{\ell}{\mapsto} \sigma'$ in $\mathcal{S}_{[A^r]_\delta}$; and (2) we relax the rate predicates in the automaton A to obtain the automaton $[A^r]_\delta$. Thus it is clear that for all states σ_1 and σ_2 of $[A^r]_\delta$ and for all transition m of $[A^r]_\delta$, if $\beta(\sigma_1) \stackrel{m}{\mapsto} \beta(\sigma_2)$ in $\mathcal{S}_A$, then $\sigma_1 \stackrel{m}{\mapsto} \sigma_2$ in $\mathcal{S}_{[A^r]_\delta}$. Hence the relation $\{(\sigma, \beta(\sigma)) \mid \sigma \in \Sigma_{[A^r]_\delta}\}$ is a simulation of $\mathcal{S}_A$ by $\mathcal{S}_{[A^r]_\delta}$. ■

It follows that the rate translation is sound for all linear-time properties. In particular, for safety properties, we have the following corollary.

Corollary 5 *Let A be a bounded hybrid automaton, and let $[A^r]_\delta$ be a rate translation of A. If A has an accepting finite trajectory, then so does $[A^r]_\delta$.*

6.3.4 Example: Temperature Controller with Delays

Recall the emptiness problem for the automaton B_2 of Figure 6.4 with the accepting condition $\bigcup_v \{(v, x \leq \frac{1}{5} \vee x \geq \frac{19}{5})\}$.

For the rate translation of the automaton B_2, we partition the window $[\frac{1}{5}, \frac{19}{5}]$ into the eight intervals $[\frac{1}{5}, \frac{3}{5}]$, $[\frac{3}{5}, 1]$, $[1, \frac{8}{5}]$, $[\frac{8}{5}, 2]$, $[2, \frac{13}{5}]$, $[\frac{13}{5}, 3]$, $[3, \frac{17}{5}]$, and $[\frac{17}{5}, \frac{19}{5}]$ of uneven size at most 0.6 (it is a good idea to separate intervals at the point c if $x = c$ is a conjunct of an invariant or action). After removing inconsistent edges and unreachable locations, we obtain the linear hybrid automaton $[B_2^r]_{0.6}$

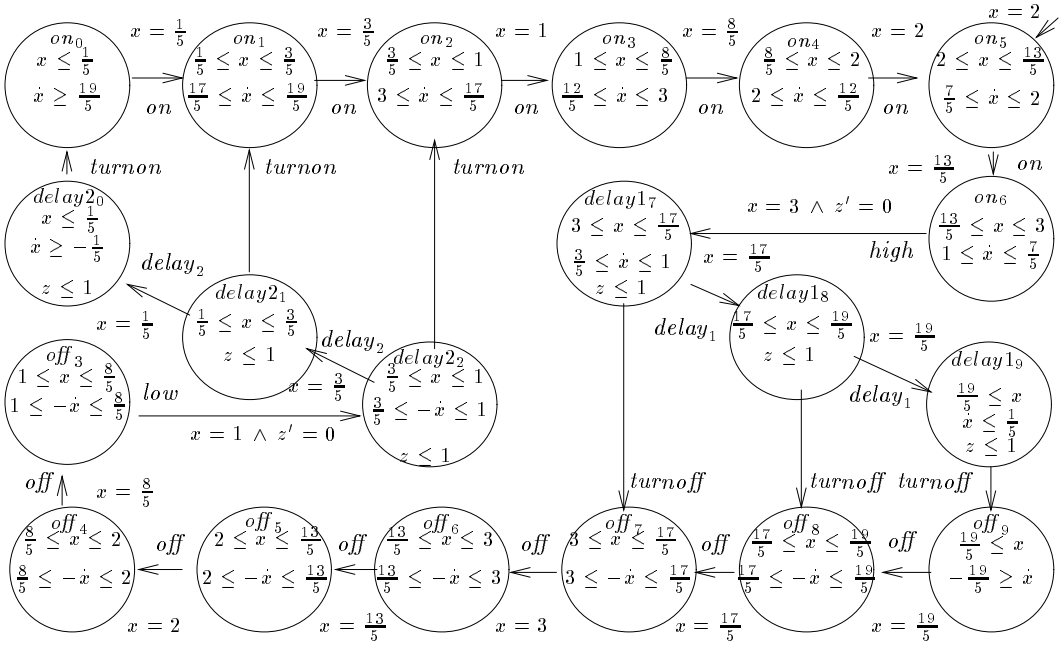

Figure 6.5: Rate translation of the temperature controller with delays

of Figure 6.5, which is a 0.6-approximate rate translation of B_2. The HyTech verifier reports that $[B_2^r]_{0.6}$ satisfies the safety property that the value of x stays within the interval $(\frac{1}{5}, \frac{19}{5})$ In fact, the automatic computation of $R([B_2^r]_{0.6})$ shows that x stays in the smaller range $(\frac{6}{25}, \frac{94}{25})$ (using 45 seconds of CPU time).

6.3.5 Error Analysis

To analyze the error of the δ-approximate rate translation $[A^r]_\delta$, we define the metric d_R such that $d_R(A, B)$ is the infimum of all nonnegative reals ε such that $R(A) \subseteq R(B) \subseteq R(A^\varepsilon)$ or $R(B) \subseteq R(A) \subseteq R(B^\varepsilon)$, if such an ε exists; otherwise, $d_R(A, B) = \infty$. The *error* of the δ-approximate rate translation $[A^r]_\delta$ is $d_R(A, [A^r]_\delta)$, where $R([A^r]_\delta)$ is interpreted as $\beta(R([A^r]_\delta))$ when compared with $R(A)$.

In particular, we analyze the error of rate translation for *monotonic* bounded

hybrid automata, where (1) in each location v each variable x is either nondecreasing or nonincreasing and (2) in the rate predicate $dif(v)$, the dotted variable $\dot{x}$ is compared with the variable x and constants only. We denote the conjuncts in $dif(v)$ involving the dotted variable $\dot{x}$ by $dif(v,x)$. By choosing successively smaller values of δ, the rate translation of a monotonic bounded hybrid automaton can satisfy any desired error bound. (at the cost of increasing the number of locations of the automaton $[A^r]_\delta$, of course). We obtain the following theorem.

Lemma 7 *Let A be a monotonic bounded hybrid automaton and $[A^r]_\delta$ be a δ-approximate rate translation that results from translating the nonlinear variable x with the window $[c,d]$. Let e be any constant such that $c \le e \le d$, and let $\sigma = (v, \vec{s})$ be a state of $[A^r]_\delta$. Define $max_v(\delta) = max\{t \mid \sigma \stackrel{t}{\mapsto} \sigma'; \sigma'(x) = e\}$, $min_v(\delta) = min\{t \mid \sigma_1 \stackrel{t}{\mapsto} \sigma'; \sigma'(x) = e\}$. Then their exist a function $\Delta_v(\delta)$ such that $max_v(\delta) - min_v(\delta) \le \Delta_v(\delta)$; and either $\Delta_v(\delta) \to 0$ as $\delta \to 0$, or $max_v(\delta) = min_v(\delta) = \infty$.*

Proof. Without loss of generality, we assume that x is nondecreasing in v. If $\sigma(x) > e$ then both $max_v(\delta)$ and $min_v(\delta)$ are infinite. Suppose that $\sigma(x) \le e$ and the window $[c,d]$ is partitioned into into k subintervals $I_1 = [c_0 = c, c_1], \ldots, I_k = [c_{k-1}, c_k = d]$, each of size at most δ in the automaton $[A^r]_\delta$. Each location v is split into $k+2$ locations $v_0, \ldots, v_{k+1}$. For each v_i, we compute the minimum a_i and the maximum b_i of the function $dif(v_i, x)$ for $c_{i-1} \le x \le c_i$. Suppose that $c_{j_0} \le \vec{s}(x) \le c_{j_0+1}$ and $c_{j_1} \le e \le c_{j_1+1}$; that is, $\sigma \in v_{j_0}$ and $\sigma \in v_{j_1}$ The longest time a finite trajectory from σ to a state σ' such that $\sigma'(x) = e$.

$$max_v(\delta) = \frac{c_{i_0+1} - \sigma_1(x)}{a_{j_0}} + \frac{\delta}{a_{j_0+1}} + \cdots + \frac{\delta}{a_{j_1-1}} + \frac{c_{j_1+1} - e}{a_{j_1}},$$

and the shortest time is

$$min_v(\delta) = \frac{c_{i_0+1} - \sigma_1(x)}{b_{j_0}} + \frac{\delta}{b_{j_0+1}} + \cdots + \frac{\delta}{b_{j_1-1}} + \frac{c_{j_1+1} - e}{b_{j_1}}.$$

Then $\Delta_v(\delta) = max_v(\delta) - min_v(\delta)$ is bounded by

$$\frac{\delta}{a_0} + \cdots + \frac{\delta}{a_k} - (\frac{\delta}{b_0} + \cdots + \frac{\delta}{b_k}).$$

Since $b_i - a_i \to 0$ as $\delta \to 0$, it is clear that $\Delta_v(\delta) \to 0$ as $\delta \to 0$. ■

Lemma 8 *Let A be a monotonic bounded hybrid automaton and $[A^r]_\delta$ be a δ-approximate rate translation that results from translating the nonlinear variable x of A. Let $\sigma_1 = (v, \vec{s}_1)$ be a state of A, and let $\sigma_2 = (v', \vec{s}_2)$ be a corresponding state of $[A^r]_\delta$ such that $\beta(\sigma_2) = \sigma_1$. If $\sigma_1 \overset{t_0}{\mapsto} \sigma_1'$, then there is an open ball $B(\sigma_1'(x), \mathcal{D}_v(\delta))$ such that (1) for all states σ_2' such that $\sigma_2 \overset{t_0}{\mapsto} \sigma_2'$, $\beta(\sigma_2')(x) \in B(\sigma_1'(x), \mathcal{D}_v(\delta))$, and (2) $\mathcal{D}_v(\delta) \to 0$ as $\delta \to 0$.*

Proof. Without loss of generality, we assume that x is nondecreasing. Suppose the minimum possible value of the data variable x in such a state σ_2' is m. From Lemma 7, we know that if $\sigma_2 \overset{t_0}{\mapsto} \sigma_3$ and $\sigma_3(x) = m$, then $|t - t_0| < \Delta_v(\delta)$. The difference between the maximum and minimum possible value of the data variable x in the state σ_2' is bounded above by the maximum time difference to reach the minimum possible value of the variable x times the maximum slope of x in location v. So if the maximum slope of the data variable x in location v is E_v, then the difference between the maximum and minimum possible values of the data variable x in the state σ_2' is bounded above by $E_v \cdot \Delta_v(\delta)$.

Define $\mathcal{D}_v(\delta) = E_v \cdot \Delta_v(\delta)$. Then for all states σ_2' such that $\sigma_2 \overset{t_0}{\mapsto} \sigma_2'$, $\beta(\sigma_2')(x) \in B(\sigma_1'(x), \mathcal{D}_v(\delta))$. Since $\Delta_v(\delta) \to 0$ as $\delta \to 0$ and E_v is finite, we know that $\mathcal{D}_v(\delta) \to 0$ as $\delta \to 0$. ■

We want to point out a fact that in the context of the previous lemma, for all data variables $y \neq x$, $\beta(\sigma_2')(y) = \sigma_2(y)$, since the differential equation for the data variable y remains unchanged in $[A^r]_\delta$.

Lemma 9 *Let A be a monotonic bounded hybrid automaton and $[A^r]_\delta$ be a δ-approximate rate translation that results from translating the nonlinear variable x of A. Let $\sigma_1 = (v, \vec{s}_1)$ be a state of A, and let $\sigma_2 = (v', \vec{s}_2)$ be a corresponding state of $[A^r]_\delta$, such that $\beta(\sigma_2) = \sigma_1$. If $\sigma'_2 \stackrel{t_0}{\mapsto} \sigma_2$, then there is an open ball $B(\beta(\sigma'_2)(x), \mathcal{D}_v(\delta))$ such that (1) there is a state σ'_1 such that $\sigma'_1 \stackrel{t_0}{\mapsto} \sigma_1$ and $\beta(\sigma'_1)(x) \in B(\beta(\sigma'_2(x)), \mathcal{D}_v(\delta))$, and (2) $\mathcal{D}_v(\delta) \to 0$ as $\delta \to 0$.*

Proof. Without loss of generality, we assume that x is nondecreasing. Let σ_3 be a state such that $\sigma_3(x) = \sigma'_2(x)$ and $\sigma_3 \stackrel{t_0}{\mapsto} \sigma_1$ in A. Then from Lemma 7, we can deduce that $|t - t_0| < \Delta_v(\delta)$.

We first assume that $t - t_0$ is non-negative. So the state σ_3 needs at most $\Delta_v(\delta)$ more time units to reach the state σ_1 in the automaton A than the time needed by the state σ'_2 to reach that the state σ_2 in $[A^r]_\delta$. Let σ'_3 be the state such that $\sigma_3 \stackrel{t-t_0}{\mapsto} \sigma'_3$, and define σ'_1 to be the state such that $\sigma'_1(x) = \sigma'_3(x)$ and $\sigma'_1(y) = \sigma'_2(y)$ for all other data variables y. The we know that $\sigma'_1 \stackrel{t_0}{\mapsto} \sigma_1$. Let us bound the range of $\sigma'_1(x)$ according to $\sigma'_2(x)$. Let x_0 be the minimum common value of the variable x in the witness finite trajectories of $\sigma'_1 \stackrel{t_0}{\mapsto} \sigma_1$ and $\sigma'_2 \stackrel{t_0}{\mapsto} \sigma_2$. Suppose that σ''_1 be the first state of the witness finite trajectories of $\sigma'_1 \stackrel{t_0}{\mapsto} \sigma_1$ such that $\sigma''_1(x) = x_0$. We define σ''_2 analogously. Notice that either $\sigma''_1 = \sigma'_1$ or $\sigma''_2 = \sigma'_2$ From Lemma 7, the maximum time difference needed by the states σ''_1 and σ''_2 to reach the states σ_1 and σ_2 respectively is bounded above by $\Delta_v(\delta)$. Therefore, $|\sigma'_1(x) - \sigma'_2(x)|$ is bounded by $\mathcal{D}_v(\delta) = E_v \cdot \Delta_v(\delta)$. Thus Part (1) is proved for this case.

On the other hand, if $t - t_0$ is negative, by the similar arguments we can also show Part (1). Since $\Delta_v(\delta) \to 0$ as $\delta \to 0$ and E_v is finite, we know that $\mathcal{D}_v(\delta) \to 0$ as $\delta \to 0$. ■

For the automaton A and a δ-approximate rate translation $[A^r]_\delta$ of A, we define

$$\mathcal{D}(\delta) = max\{\mathcal{D}_v(\delta) \mid v \in V\}.$$

We claim that $R([A^r]_\delta) \subseteq R(A^{\mathcal{D}(\delta)})$.

Theorem 7 *Let A be a monotonic bounded hybrid automaton. For all reals $\varepsilon > 0$, there is a real $\delta > 0$ such that for all δ-approximate rate translations $[A^r]_\delta$ that result from translating a nonlinear variable x of A, $R(A) \subseteq R([A^r]_\delta) \subseteq R(A^\varepsilon)$.*

Proof. From Proposition 10, $R(A) \subseteq R([A^r]_\delta)$. We shall show that $R([A^r]_\delta) \subseteq R(A^{\mathcal{D}(\delta)})$. Let P be a finite trajectory from a state $\sigma_1 \in I_{[}A^r]_\delta$ to a state $\sigma_k \in F_{[}A^r]_\delta$. We re-identify the a sub-path of P that corresponds to a single time step in a location of A. Formally, if $\sigma_{j_0} \overset{m_{j_0}}{\mapsto} \sigma_{j_1} \overset{m_{j_1}}{\mapsto} \cdots \overset{m_{j_k-1}}{\mapsto} \sigma_{j_k}$ is a sub-path of P and all the states $\sigma_{j_0} \cdots \sigma_{j_k}$ are in the same location v of A and all the transitions $m_{j_0} \cdots m_{j_k-1}$ are either time steps or transition steps that correspond to the transitions interconnecting the locations split from the location v, then we collapse the sub-path to a single time step $\sigma_{j_0} \overset{t_0}{\mapsto} \sigma_{j_k}$ where t_0 is the sum of all durations of the time steps among $m_{j_0} \cdots m_{j_k-1}$.

We can further remove stutter transitions from the resulting path and add zero duration time steps if necessary to obtain the finite path

$$P' = \sigma_0 \overset{m_0}{\mapsto} \sigma_1 \overset{m_1}{\mapsto} \sigma_2 \overset{m_2}{\mapsto} \cdots \overset{m_{k-2}}{\mapsto} \sigma_{k-1} \overset{m_{k-1}}{\mapsto} \sigma_k$$

such that (1) both m_1 and m_{k-1} are time steps, (2) the time and transition steps occur alternatively in P', and (3) no transition step in P' corresponds to a stutter transition.

We will prove by induction on k that there is an initial state $\sigma'_0 \in I_{A^{\mathcal{D}(\delta)}}$ and a finite trajectory

$$Q = \sigma'_0 \overset{m_0}{\mapsto} \sigma_1 \overset{m_1}{\mapsto} \sigma'_2 \overset{m_2}{\mapsto} \cdots \overset{m_{k-2}}{\mapsto} \sigma'_{k-1} \overset{m_{k-1}}{\mapsto} \sigma_k$$

in $\mathcal{S}_{A^{\mathcal{D}(\delta)}}$.

Consider first the base case that $k = 1$; that is, there is only a time step in P'. Since the conjuncts involving the data variable x in the initial conditions and

location invariants of the automaton $A^{\mathcal{D}(\delta)}$ are relaxed by $\mathcal{D}(\delta)$, by Lemma 9, there must be a state σ_0' in $I_{A^{\mathcal{D}(\delta)}}$ such that $\sigma_0' \stackrel{m_0}{\mapsto} \sigma_1$. So we take $Q = \sigma_0' \stackrel{m_0}{\mapsto} \sigma_1$.

Now consider the induction step that we have constructed a finite trajectory Q from the state σ_0' to the state σ_{2i-1} and $\sigma_{2i-1} \stackrel{e}{\mapsto} \sigma_{2i} \stackrel{t_0}{\mapsto} \sigma_{2i+1}$ is a sub-path of P'. First we notice that the guard involving the variable x is satisfied by σ_{2i-1}. In addition, the nondeterministic assignments to the variable x are relaxed by $\mathcal{D}(\delta)$, so any state σ_{2i}' within the open ball $B(\sigma_{2i}, \mathcal{D}(\delta))$ can be reached. By Lemma 9, there is a state $\sigma_{2i}' \in B(\sigma_{2i}, \mathcal{D}(\delta))$ such that $\sigma_{2i}' \stackrel{t_0}{\mapsto} \sigma_{2i+1}$ in the automaton $A^{\mathcal{D}(\delta)}$. Therefore, we can augment the finite trajectory Q by two steps $\sigma_{2i-1} \stackrel{e}{\mapsto} \sigma_{2i}' \stackrel{t_0}{\mapsto} \sigma_{2i+1}$.

We have shown that if a state σ_k can be a final state of a finite trajectory of $[A^r]_\delta$, then the state σ_k can also be a final state of a finite trajectory of $A^{\mathcal{D}(\delta)}$; that is, $R([A^r]_\delta) \subseteq R(A^{\mathcal{D}(\delta)})$. Therefore, given any $\varepsilon > 0$, we take δ to be a nonnegative real such that $\mathcal{D}(\delta) \leq \varepsilon$. Then $R(A) \subseteq R([A^r]_\delta) \subseteq R(A^\varepsilon)$. ■

Corollary 6 *Let A be a monotonic bounded hybrid automaton. For all reals $\varepsilon > 0$, there is a real $\delta > 0$ such that if a δ-approximate rate translation of A has an accepting finite trajectory, then so does A^ε.*

However, there is not always a positive constant δ such that $[A^r]_\delta$ and A have the same result of the emptyness problem. Let v be a location of a hybrid automaton A such that (1) $dif(v, x) = 4 - x$, (2) $inv(v) = 0 \leq x \leq 4$, and (3) the initial value of the variable x is less than 4 when a finite trajectory reach the location v. Then the region $[\![(v, x = 4)]\!]$ is not reachable by A. But if v_k is the split location with the invariant $4 - \delta \leq x \leq 4$ from a δ-partition of the interval $[0, 4]$, then the maximum slope of x in v_k is $\delta > 0$, so for any δ, the region $[\![(v_k, x = 4)]\!]$ is reachable by $[A^r]_\delta$.

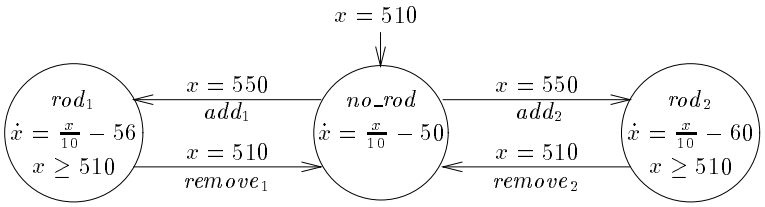

Figure 6.6: The nonlinear reactor core automaton

6.4 Discussion

The two translations presented in this chapter provide algorithmic methods for verifying safety properties of two classes of nonlinear hybrid systems: solvable systems and bounded systems. Whenever both translations are applicable, the clock translation is generally preferable, because the size of the translated automaton does not depend on the precision of the translation. We use a nonlinear version of the reactor temperature control system in Chapter 4 to compare the results of the application of the two translations to the same automaton.

Suppose that the reactor core of the reactor temperature control system is modeled by the hybrid automaton shown in Figure 6.6. The core temperature x increases according to the differential equation $\dot{x} = x/10 - 50$ if no control rod is in the reactor core; x decreases according to the differential equation $\dot{x} = x/10 - 56$ if control rod 1 is in the reactor core; and x decreases according to the differential equation $\dot{x} = x/10 - 60$ if control rod 2 is in the reactor core. Then the clock translation of the reactor core automaton is shown in Figure 6.7. In a second step, we can apply a 0.1-approximate clock translation to the automaton of Figure 6.7 to get the linear hybrid automaton of Figure 6.8.

We now try the rate translation on the same automaton. Consider, for example, the location rod_1 of the reactor core automaton from Figure 6.6, with the

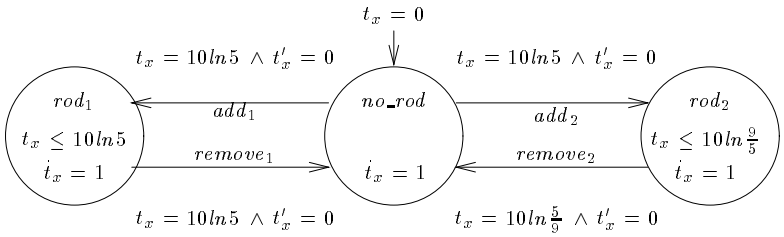

Figure 6.7: The clock-translated reactor core automaton

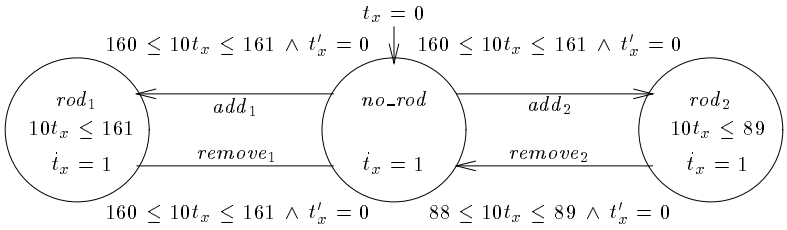

Figure 6.8: The 0.1-approximate clock-translated core automaton

differential equation $\dot{x} = x/10 - 56$. Since we can strengthen the invariant to $510 \leq x \leq 550$, the derivative of x is bounded below by -5 and above by -1. Thus we can replace the differential equation for the variable x with the rate interval $dif(no\_rod, x) = [-5, -1]$. By treating the other locations similarly, we obtain the linear hybrid automaton of Figure 4.1, if we decided not to split any location.

Recall that HyTech guarantees that the reactor core automaton of Figure 4.1 meets its safety requirement iff the parameter w satisfies the condition $9w < 184$ (Section 4.2.2). For the clock-translated reactor core automaton of Figure 6.8, HyTech computes the weaker condition $5w < 189$. This condition is weaker, because the clock translation of Figure 6.8 gives a better approximation of the nonlinear system of Figure 6.6 than does the rate translation of Figure 4.1. Thus better approximations allow the design engineers to use slower mechanisms for moving the control rods. If desired, the approximation by rate translation can be

refined by splitting control locations.

Chapter 7

Case Studies

Why does this magnificent applied science which saves work
and makes life easier bring us so little happiness?
The simple answer runs:
because we have not yet learned to make sensible use of it.
— Albert Einstein

In order to help the practitioners make sensible use of HyTech, we show the application of HyTech to three nontrivial benchmark problems in this chapter. All three examples are taken from the literature, rather than devised by us. The first case study is a distributed control system introduced by Corbett [Cor94]. The system consists of a controller and two sensors, and is required to issue control commands to a robot within certain time limits. The two sensor processes are executed on a single processor, as scheduled by a priority scheduler. This scenario is modeled by linear hybrid automata with clocks and stop-watches. HyTech automatically computes the maximum time difference between two consecutive control commands generated by the controller. It follows, for example, that a scheduler that gives higher priority to one sensor may meet the specification requirement,

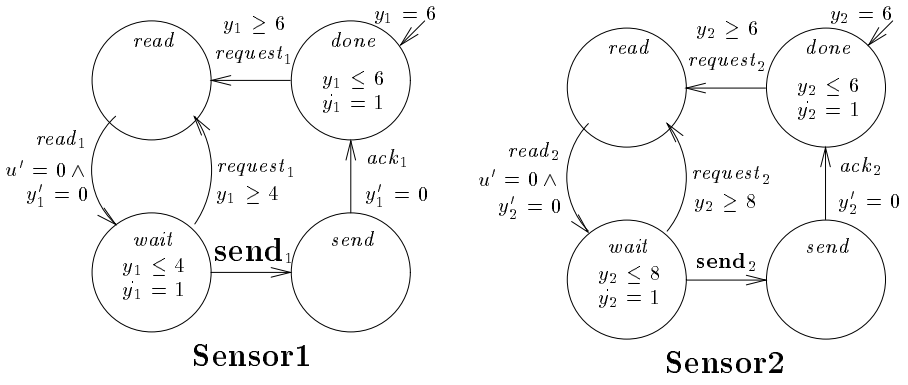

Figure 7.1: The two sensors

while a scheduler that gives priority to the other sensor may fail the requirement.

The second case study is a two-robot manufacturing system introduced by Puri and Varaiya [PV95b]. The system consists of a conveyor belt with two boxes, a service station, and two robots. The boxes will not fall to the floor iff initially the boxes are not positioned closely together on the conveyor belt. HyTech automatically computes the minimum allowable initial distance between the two boxes.

The third case study is the Philips audio control protocol presented by Bosscher, Polak, and Vaandrager [BPV94]. The protocol consists of a sender that converts a bit string into an analog signal using the so-called Manchester encoding, and a receiver that converts the analog signal back into a bit string. The sender and the receiver use clocks that may be drifting apart. In [BPV94], it was shown, by a human proof, that the receiver decodes the signal correctly if and only if the clock drift is bounded by a certain constant. HyTech automatically computes that constant for input strings up to 8 bits.

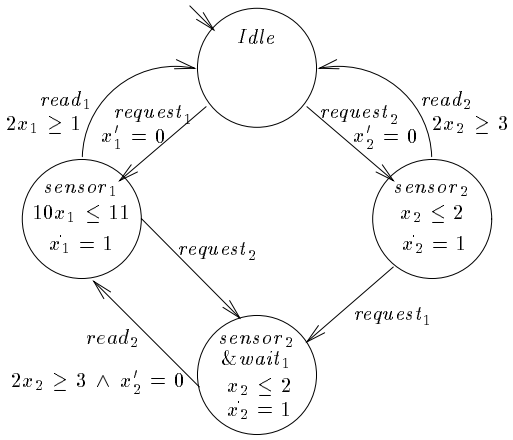

Figure 7.2: The scheduler

7.1 A Distributed Control System with Time-outs

The distributed control system of [Cor94] consists of two sensors and a controller that generates control commands to a robot according to the sensor readings. The programs for the two sensors and the controller are written in ADA. The two sensors share a single processor, and the priority of sensor 2 for using the processor is higher than the priority of sensor 1. In other words, if both sensor 1 and sensor 2 want to use the processor to construct a reading, only sensor 2 obtains the processor, and sensor 1 has to wait. The two sensors are modeled by the two linear hybrid automata in Figure 7.1 and the priorities for using the shared processor are modeled by the scheduler automaton in Figure 7.2.

Each sensor can be constructing a reading (location *read*), waiting for sending the reading (location *wait*), sending the reading (location *send*), or sleeping (location *done*). The processor can be scheduled idle (location *Idle*), serving sensor 1 (location $sensor_1$), serving sensor 2 while sensor 1 is waiting (location

$sensor_2 \& wait_1$), or serving sensor 2 while sensor 1 is not waiting (location $sensor_2$).

Each sensor constructs a reading and sends the reading to the controller. The shared processor for constructing sensor readings is requested via *request* transitions, the completion of a reading is signaled via *read* transitions, and the reading is delivered to the controller via *send* transitions. Sensor 1 takes 0.5 to 1.1 milliseconds and sensor 2 takes 1.5 to 2 milliseconds of CPU time to construct a reading. These times are measured by the stop-watches x_1 and x_2 of the scheduler automaton. Notice that at most one of the two stop-watches x_1 and x_2 runs in a location of the scheduler automaton, which reflects the fact that only one sensor can use the shared processor at a time. If sensor 1 loses the processor because of preemption by sensor 2, it can continue the construction of its reading after the processor is released by sensor 2.

Once constructed, the reading of sensor 1 expires if it is not delivered within 4 milliseconds, and the reading of sensor 2 expires if it is not delivered within 8 milliseconds. These times are measured by the clocks y_1 and y_2 of the sensor automata. If a reading expires, then a new reading must be constructed. After successfully delivering a reading, a sensor sleeps for 6 milliseconds (measured again by the clocks y_1 and y_2), and then constructs the next reading.

The controller is modeled by the automaton in Figure 7.3. The controller is executed on a dedicated processor, so it does not compete with the sensors for CPU time. We use the clock z to measure the delays and time-outs of the controller. The controller accepts and acknowledges a reading from each sensor, in either order, and then computes and sends a command to the robot. The sensor readings are acknowledged via *ack* transitions, and the robot command is delivered via a *signal* transition. It takes 0.9 to 1 milliseconds to receive and acknowledge a sensor reading. The two sensor readings that are used to construct a robot command must be received within 10 milliseconds. If the controller receives a reading from one

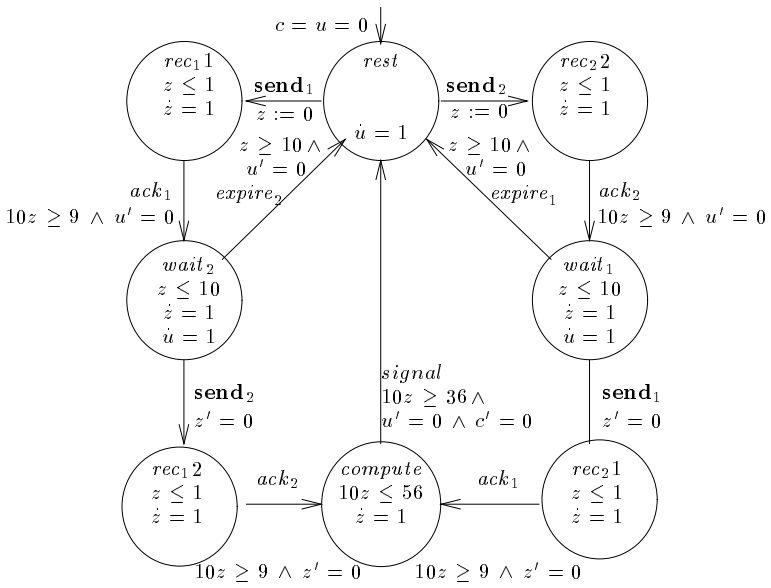

Figure 7.3: The controller

sensor but does not receive the reading from the other sensor within 10 milliseconds, then the first sensor reading expires (via an *expire* transition). Once both reading are received, the controller takes 3.6 to 5.6 milliseconds to synthesize a robot command.

We want to know how often a robot command can be generated by the controller. For this purpose, we add a clock c to the controller automaton such that c measures the elapsed time since the last robot command was sent. The slope of the clock c is 1 in all locations of the controller automaton (this is omitted from Figure 7.3), and c is reset to 0 whenever a robot command is sent. We want to compute the maximum value of the clock c in all states that are reachable in the product of all four automata.

However, the product of the four automata does not model the system exactly according to Corbett's specification. This is because the **send** transitions should be urgent, that is, they should be taken as soon as they are enabled. We model

the urgency of the **send** transitions by adding an additional clock, u, and global invariants. The clock u is reset whenever a sensor is ready to send a reading to the controller, and whenever the controller is ready to receive a sensor reading. Then we use the global invariant that $u = 0$ if both a sensor and the controller are ready for a transmission; that is,

$$
\begin{aligned}
&(l[sensor_1] = wait \wedge l[controller] = rest \rightarrow u = 0) \wedge \\
&(l[sensor_2] = wait \wedge l[controller] = rest \rightarrow u = 0) \wedge \\
&(l[sensor_1] = wait \wedge l[controller] = wait_1 \rightarrow u = 0) \wedge \\
&(l[sensor_2] = wait \wedge l[controller] = wait_2 \rightarrow u = 0).
\end{aligned}
$$

This invariant enforces whenever a transmission of a sensor reading is enabled, the transmission happens immediately.

In HyTech, the global invariant is defined as follows:

```
GlobalInvar={{l[sensor1]==wait && l[controller]==rest, 0==u},
    {l[sensor2]==wait && l[controller]==rest, 0==u},
    {l[sensor1]==wait && l[controller]==wait1, 0==u},
    {l[sensor2]==wait && l[controller]==wait2, 0==u)}}
```

To compute the range of possible values for the clock c in the reachable states, we write:

```
InitialState = l[sensor1]==done && l[sensor2]==done &&
    l[scheduler]==idle && l[controller]==rest &&
    0==c && 6==y1 && 6==y2 && 0==u
Bad = True

EliminateLocList = {sensor1,sensor2,sched,gen}
EliminateVarList = {x1,x2,y1,y2,z,w,u}
```

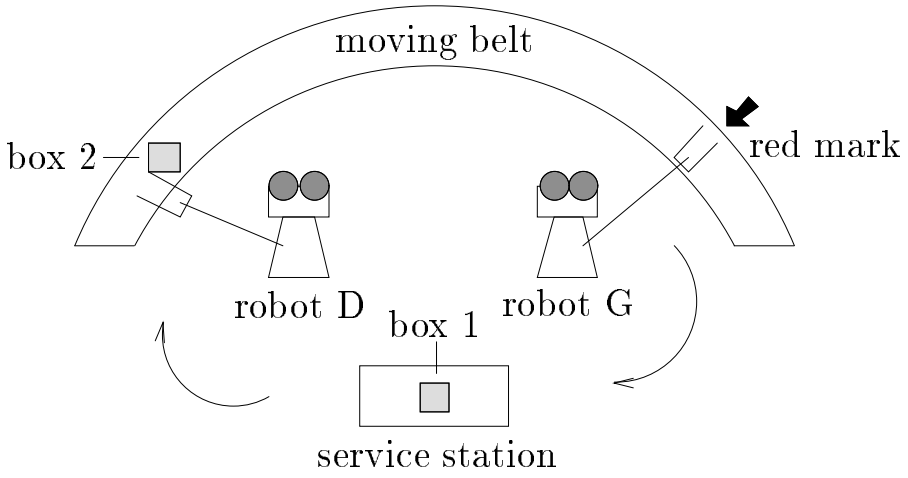

Figure 7.4: The two-robot manufacturing system

Notice that, using the two projection operators, we ask HyTech to print only information about the clock c. Using forward analysis without approximation, HyTech returns, in 89.53 seconds of CPU time, the following answer:

```
0 <= c && -12 <= -5*c || -7 <= -2*c && 9 <= 10*c ||
-3 <= -c && 7 <= 10*c || -9 <= -2*c && 3 <= 2*c ||
12 <= 5*c && -28 <= -5*c || 5 <= 2*c && -18 <= -2*c ||
33 <= 10*c && -105 <= -10*c || 42 <= 5*c && -56 <= -5*c
```

From this result (the last disjunct is $42 \leq 5c \wedge -56 \leq -5c$), it follows that the maximum value of the clock c is 11.2; that is, a robot command is generated by the controller at least once every 11.2 milliseconds. We can also apply HyTech to analyze the same system except that the priority of sensor 1 for using the shared processor is higher than the priority of sensor 2. In that case, a robot command is generated at least once every 11.0 milliseconds.

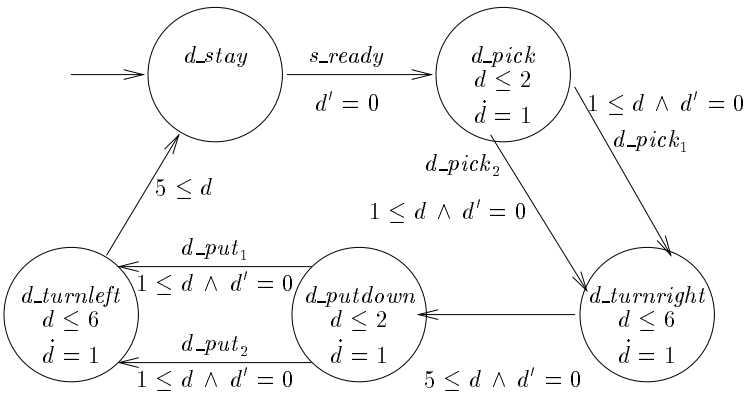

Figure 7.5: Robot D

7.2 A Two-robot Manufacturing System

Puri and Varaiya [PV95b] designed a manufacturing system that consists of a conveyor belt with two boxes, a service station, and two robots. The system is illustrated in Figure 7.4. This system has been also modeled and analyzed in [DY95].

Robot D, one of the two robots, is modeled by the linear hybrid automaton of Figure 7.5. The clock d is used to measure the time needed for the actions performed by robot D. Initially robot D is looking at the service station (location *d_stay*). When it sees an unprocessed box in the service station, it picks up that box from the service station in 1 to 2 seconds (location *d_pick*), makes a right turn in 5 to 6 seconds (location *d_turnright*), puts the box at one end of the conveyor belt in 1 to 2 seconds (location *d_putdown*), makes a left turn back to the service station in 5 to 6 seconds (location *d_turnleft*), and stays at there waiting for the next unprocessed box (location *d_stay*).

The two boxes, box 1 and box 2, are modeled by the indexed linear hybrid automaton in Figure 7.6, where the index i is either 1 (for box 1) or 2 (for box 2).

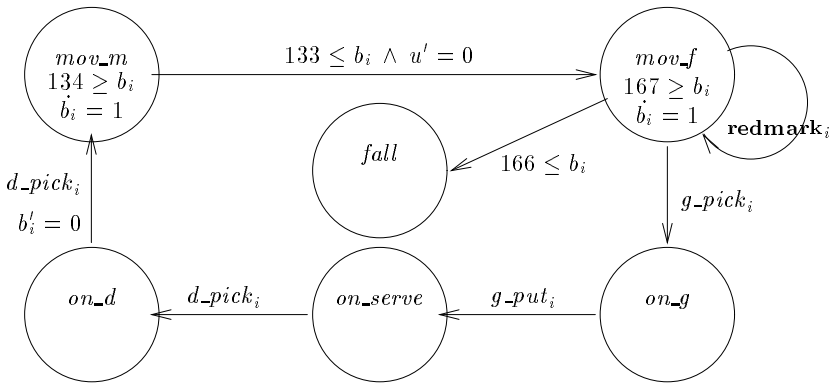

Figure 7.6: Box i

A box may be in the service station (location *on_serve*), held by robot D (location *on_d*), moving on the conveyor belt before a red mark (location *mov_m*), moving on the conveyor belt beyond the red mark (location *mov_f*), held by robot G (location *on_g*), or falling off the end of the conveyor belt (location *fall*). A box on the conveyor belt is processed by the manufacturing system. The conveyor belt is moving at a certain speed from one end to the other. The clock b_i measures the total time that box i spends on the conveyor belt, and thus determines the position of box i on the belt. A box requires 133 to 134 seconds to reach the red mark after it is placed on the belt by robot D. If a box is not picked up by robot G before the end of the belt, then the box falls off the belt 166 to 167 seconds after it is placed on the belt.

Robot G at the end of the conveyor belt is modeled by the automaton in Figure 7.7. The clock g measures the time needed to perform the actions of robot G. Initially robot G is looking at the red mark next to the conveyor belt (location *g_stay*). When it sees a processed box moving beyond the red mark, it picks up that box from the belt in 3 to 8 seconds (location *g_pick*), makes a right turn in 6 to 11 seconds (location *g_turnright*), waits for the service station to be empty

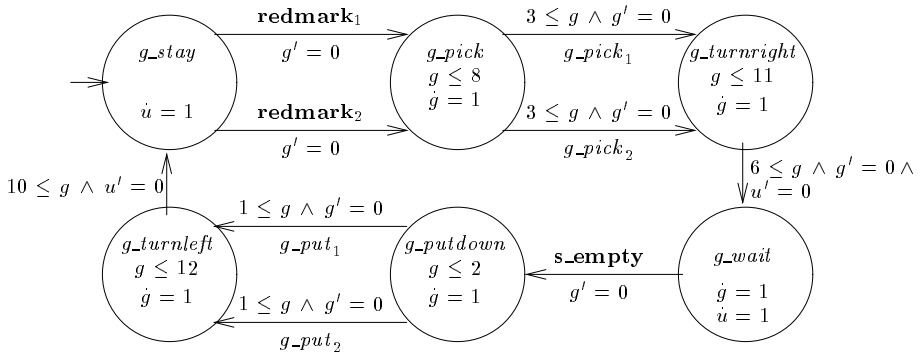

Figure 7.7: Robot G

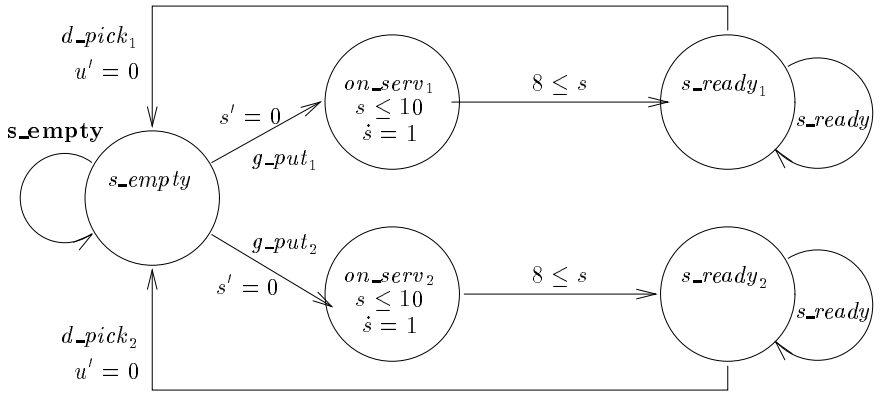

Figure 7.8: The service station

(location *g_wait*), puts the box into the service station in 6 to 11 seconds (location *g_putdown*), makes a left turn back to the conveyor belt in 1 to 2 seconds (location *g_turnleft*), and stays there watching the red mark (location *g_stay*).

The service station is modeled by the automaton in Figure 7.8. Whenever the service station receives a processed box, it pops up an unprocessed box for robot D to pick up. The service station takes 8 to 10 seconds to switch the processed and unprocessed boxes, which is measured by the clock s. Initially both boxes are on the conveyor belt before the red mark. There are at most two boxes on the belt

at any time, because the service station pops up a new box only when it receives a processed box from robot G.

According to Puri and Varaiya's specification, the transitions with the synchronization letters *s_ready*, **redmark**$_1$, **redmark**$_2$, and **s_empty**, are urgent; that is, robot D picks up a box from the service station as soon as it is ready and sees a box in the service station, etc. We treat the *s_ready* transitions as ordinary transitions, because this assumption will not affect our analysis. We use the clock u and the following global invariants to model the urgent transitions:

```
GlobalInvar = {{l[grobot]==stay && l[box1]==movf, 0==u},
    {l[grobot]==stay && l[box2]==movf, 0==u},
    {l[grobot]==wait && l[station]==sempty, 0==u},
    {l[grobot]==wait && l[station]==sempty, 0==u}}
```

We want to check the safety requirement that no box will ever fall off the conveyor belt. This requirement clearly depends on the initial positions of the two boxes on the belt. We use the parameter *dist* such that $dist = b_1 - b_2$ represents the difference of the initial values of the clocks b_1 and b_2.

Then we use HyTech to analyze the reachability problem $(A, \varphi_I, \varphi_F)$, where A is the product of all five automata and

$$\varphi_I = (\ell[box_1] = mov\_m \wedge \ell[box_2] = mov\_m \wedge \ell[robot_G] = g\_stay \wedge \ell[robot_D] = d\_stay \wedge \ell[servicestation] = s\_empty \wedge u = 0),$$
$$\varphi_F = (\ell[box_1] = fall \vee \ell[box_2] = fall).$$

After we simplified the product automaton by eliminating unreachable locations and identifying locations in which a box is fallen, HyTech is able to return, in 163.41 minutes of CPU time, the following target region:

```
-1 <= -b1+b2 && -9 <= b1-b2 || -1 <= b1-b2 && -9 <= -b1+b2
```

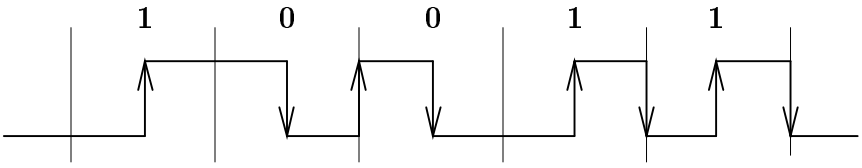

Figure 7.9: The Manchester encoding

using backward computation without approximation. It follows that

$$b2 - b1 > 9 \vee b1 - b2 > 9$$

is a necessary and sufficient condition on the initial condition of the system so that neither box will fall off the conveyor belt; that is, $|b_1 - b_2| > 9$.

7.3 The Philips Audio Control Protocol

In [BPV94], the timing-based Philips audio control protocol is modeled by an extension of the timed I/O automata model [LV92,LV93], and verified mathematically without computer support. We model the same protocol using linear hybrid automata, and verify its correctness for input strings up to length 8 using HyTech.

The protocol consists of a sender and a receiver. The sender uses the Manchester encoding to encode an input string of bits into a continuous signal (see Figure 7.9 for the encoding of 10011). The voltage on the communication bus is either high or low. A 0 bit is sent as a *down* signal from high to low voltage; a 1 bit is sent as a *up* signal from low to high voltage. The time line is divided into time slots of equal length, and the signals are sent in the middle of each time slot. The receiver decodes the continuous signal into an output string of bits. The protocol is correct iff the input and output strings match.

The time slots are measured by local clocks of the sender and receiver. These local clocks, however, may not be accurate, and their derivatives may vary within

the rate interval $[\frac{19}{20}, \frac{21}{20}]$. Besides this potential 5% (or 1/20) timing error, the protocol faces also the following complications:

- The receiver does not know when the first time slot begins. The sender and the receiver can synchronize at the beginning of the transmission by knowing that (1) before the transmission, the voltage is low, and (2) the transmitted string starts with the bit 1.
- The receiver does not know the length of the bit string that is transmitted.
- The receiver sees only up signals and no down signals (because down signals are difficult to detect).

Using HyTech, we verify that whenever the sender encodes and sends a string of up to 8 bits, the receiver correctly decodes all bits in string. HyTech also shows that the protocol is incorrect in the case that the local clocks are subject to timing errors up to 1/15.

We use four linear hybrid automata to model the input, the sender, the receiver, and the output. The input automaton of Figure 7.10 generates all the possible bit strings up to a certain length. The length of the input string is decided by the initial value of the integer variable k. Whenever a bit is nondeterministically generated by the input automaton, the value of k is decremented by 1. When k becomes 0, the input automaton nondeterministically generates a suffix of one or two bits. So the input automaton generates all possible input strings of $k+1$ or $k+2$ bits. The integer variable c stores the message that is sent. If the bit 0 is sent, then c is updated to $2c$; if the bit 1 is sent, then c is updated to $2c+1$. The input automaton synchronizes with the sender through the synchronization letters $head_i$ and $input_i$, which correspond to looking at the next bit of the input string and sending the next bit of the input string, respectively.

The sender is modeled by the automaton of Figure 7.11. The variable x represents the drifting local clock of the sender, and its rate predicate is $\frac{19}{20} \leq \dot{x} \leq \frac{21}{20}$ for all locations of the sender automaton. The sender synchronizes with the receiver through the synchronization letter *up*, which represents up signals.

The receiver is modeled by the automaton of Figure 7.12. The variable y represents the drifting local clock of the receiver, and its rate predicate is $\frac{19}{20} \leq \dot{y} \leq \frac{21}{20}$ for all locations of the receiver automaton. The receiver sees each up signal of the sender and decides if it encodes a 0 or a 1. If no up signal is received for a certain amount of time, the receiver times out and concludes that the transmission is completed. The receiver synchronizes with the output automaton through the synchronization letters $output_i$, which represent the bits of the decoded string.

The output automaton of Figure 7.13 stores the decoded bit string in the integer variable d. Then $c = d$ signals a correct decoding, and $c \neq d$ signals an error in the protocol. We check if it is possible to reach a state that satisfies $c \neq d$ when the transmission is completed. A forward reachability analysis with HyTech successfully verifies the protocol for $k = 6$. The performance of HyTech is summarized in Table 7.3.

Table 7.1: Verification of the audio control protocol

Clock error	Input length	Location number	Transition number	CPU time	
1/20	5 or 6	1300	2795	681.8 sec.	Proved
	7 or 8			4275 sec.	
1/15	5 or 6			2018 sec.	Disproved

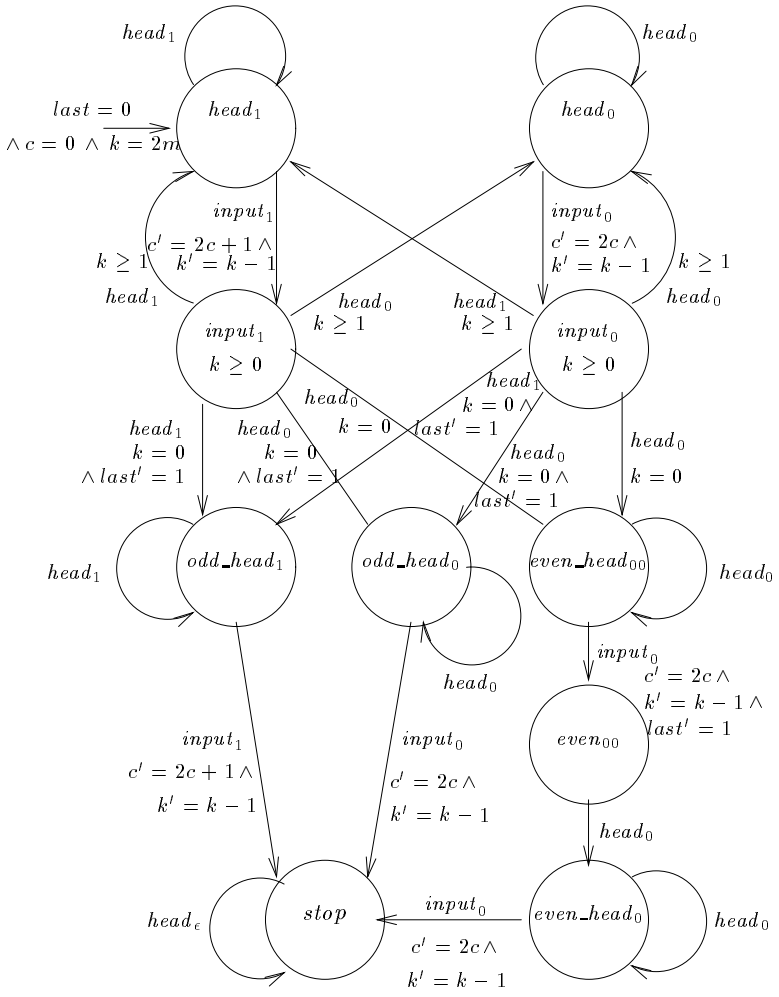

Figure 7.10: The input automaton

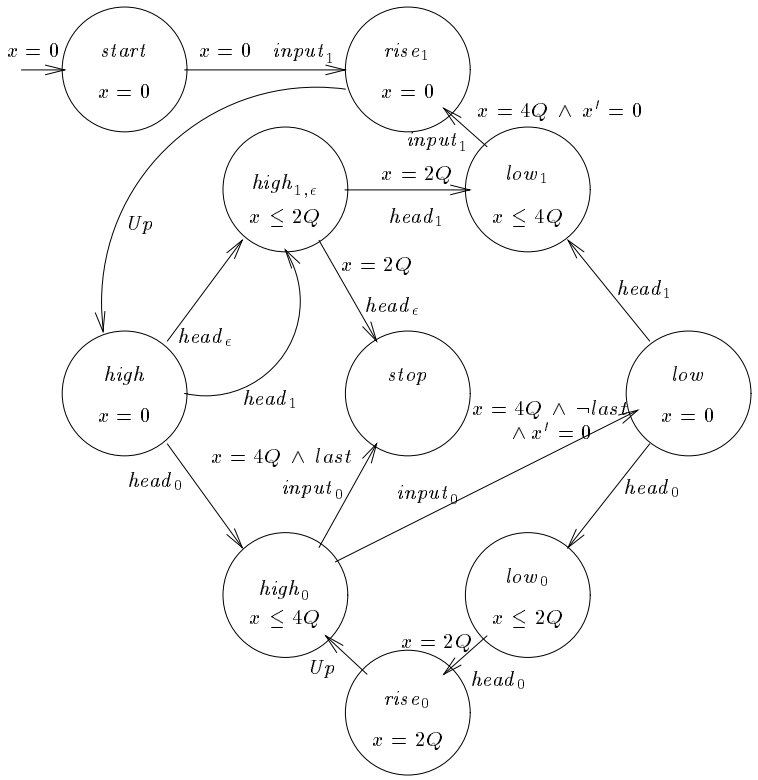

Figure 7.11: The sender automaton

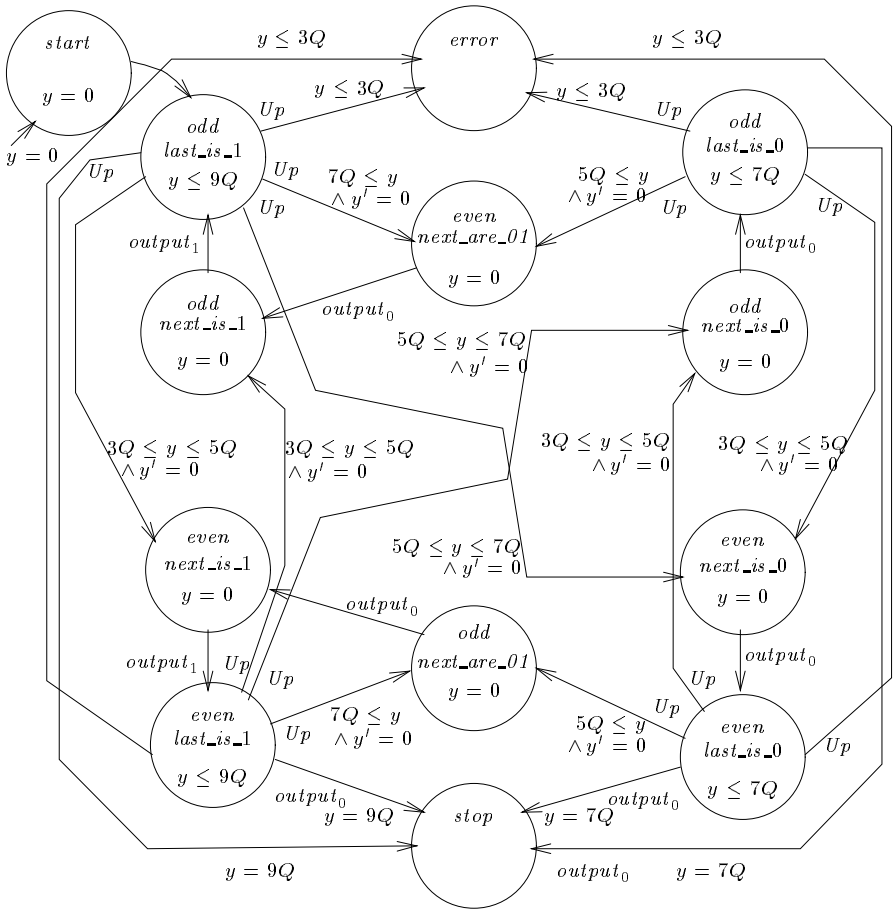

Figure 7.12: The receiver automaton

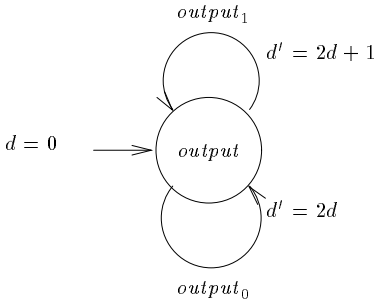

Figure 7.13: The output automaton

Chapter 8

Future Work

This is not the end. It is not even the beginning
of the end. But it is, perhaps, the end of the beginning.
— Winston Churchill

This dissertation presents a general framework for the formal specification and verification of hybrid systems, as well as a software tool—HyTech that can automatically analyze linear hybrid systems.

The long-term goal of this line of research is a software-hardware-mechanism integrated prototyping environment that supports (1) the rapid prototyping of embedded systems in hybrid automata, (2) the simulation of the prototype through the hybrid automaton model, (3) the verification and analysis of the hybrid automata model, and (4) the synthesis of the code and the circuit from the hybrid automata model.

To reach this long-term goal, there are many exciting short-term continuations of this work to be done. We list some of them as follows.

Analysis of More Nonlinear Hybrid Systems

Since most embedded systems operate in nonlinear environments, the study of nonlinear hybrid systems is obvious the most important subject in this line of research.

This dissertation introduces two algorithms for translating nonlinear hybrid systems into linear hybrid systems. The author believes that the *symbolic reduction* method in [KM91] for translating circuits at the transistor level to finite-state machines can be extended to another sound translation from nonlinear hybrid automata to linear hybrid automata. The result would permit the analysis of a more general class of nonlinear hybrid automata.

The advance of this topic would require the knowledge of the existing results from control theory and dynamic systems.

Analysis of More Real-world Applications

The hands-on experiences obtained by analyzing real-world applications can show the practitioners how to model and verify their designs, and also suggest researchers how to enhance and develop their verification tools. Recently, we are interested in applying HyTech to verify mixed-signal circuits and digital circuits at the transistor level.

Hybrid-automaton Based Simulation

Verification is expensive in terms of the required machine time and space. If the system prototype is designed in hybrid automata, a simulation tool based on hybrid automata can provide the designer a faster, yet more accurate, estimation of the behavior of the system design.

Distributed Analysis Tool

We believe that distributed hybrid systems should be analyzed by distributed systems. A parallelized HyTech can certainly verify complex systems that the current HyTech could not verify. Since the fixpoint computation in our symbolic model-checking procedure is location-based, the degree of the locality of the procedure is high. Consequently, the parallelization of HyTech should be a promising direction of future work.

Automatic Synthesis of Hybrid-system Controller

According to the paper [MPS95], the synthesis of a hybrid system controller can be done by computing fixpoints of the precondition operators that we introduced in Chapter 3. Since HyTech can compute the precondition operators, we should be able to augment HyTech to become a hybrid-system controller synthesizer.

Appendix A

The Grammar of the HyTech Input Language

The following is the grammar of the HyTech input language. We want to note that (1) the regular expressions are surrounded by "⟨" and "⟩", (2) the nonterminal symbols are in *italic* characters, (3) the terminal symbols are in `typewriter` characters, (4) each semicolon ";" can be replaced by a line break "`\n`", and (5) the character ε represents the null string.

program $\longrightarrow$ *name_definition variable_definition automaton_definition verification_definition*

name_definition $\longrightarrow$ ε

| `define(`*defined_name* `,` *number* `)` `\n`
name_definition

defined_name $\longrightarrow$ *name*

name $\longrightarrow$ $\langle$`[a-zA-Z][0-9a-zA-Z]*`$\rangle$

number ⟶ ⟨[1 − 9][0 − 9]∗⟩

variable_definition ⟶ *variabledefs* *variable_definition*

| *variabledefs*

variabledefs ⟶ `AnaVariables = {`*variable_sequence* `};`

| `DisVariables = {`*variable_sequence* `};`

variable_sequence ⟶ *variable_sequence* , *variable*

| *variable*

variable ⟶ *name*

automaton_definition ⟶

automaton_numbers *location_definition* *transition_definition*

automaton_numbers ⟶ `AutomataNo =` *number_sequence*

number_sequence ⟶ *number_sequence* , *number*

| *number*

location_definition ⟶ *locnumbers* *invariants* *rates*

locnumbers ⟶ `locationo = {`*number_sequence* `};`

invariants ⟶ *invariantdef* *invariants*

| *invariantdef*

invariantdef ⟶ `inv[`*location_eq* `] == `*locname* `] =` *data_predicate* ;

location_eq ⟶ `l[`*automaton_name* `] ==` *locname*

data_predicate ⟶ *linear_inequalities* `&&` *linear_inequalities*

| *linear_inequalities*

| *boolean*

boolean ⟶ `True`

| `False`

linear_inequalities ⟶ *linear_term* `==` *linear_term*

| *linear_term* `>=` *linear_term*

| *linear_term* `<=` *linear_term*

linear_term ⟶ *number* *variable* `+` *linear_term*

| `-`*number* *variable* `+` *linear_term*

| *number* *variable*

| `-`*number* *variable*

| *number*

| `-`*number*

| *variable*

automaton_name ⟶ *defined_name*

| *number*

locname ⟶ *defined_name*

| *number*

rates ⟶ *ratedef rates*

| *ratedef*

ratedef ⟶ `dif[`*automaton_name* , *locname* , *variable* `] =` *rate* ;

rate ⟶ {*number* , *number* }

| *number*

transition_definition ⟶ *transnumbers transitiondefs labeldefs*

transnumbers ⟶ `transitiono =` {*number_sequence* };

transitiondefs ⟶ *transdef transitiondefs*

| *transdef*

transdef ⟶

`act[`*automaton_name* , *number* `] =` {*guard* , *synlabel*
{*assignments* }}

guard ⟶ *convex_state_predicate*

convex_state_predicate ⟶ *location_predicate* `&&` *data_predicate*

location_predicate ⟶ *location_eq* `&&` *location_predicate*

| *location_eq*

| *boolean*

synlabel ⟶ ε

| *name* ,

assignments $\longrightarrow$ *location_assignment* *assignment_sequence*

| *location_assignment*

location_assignment $\longrightarrow$ `l[`*automaton_name*`]` `->` *location_name*

assignment_sequence $\longrightarrow$ ε

| `,` *assignment* *assignment_sequence*

| `,` *assignment*

assignment $\longrightarrow$ *variable* `->` *linear_term*

labeldefs $\longrightarrow$ ε

| *labeldef* *labeldefs*

labeldef $\longrightarrow$ `syn[`*label*`]` `=` {*automaton_name_sequence* }`;`

automaton_name_sequence $\longrightarrow$ *automaton_name_sequence* `,`
automaton_name

| *automaton_name*

verification_definition $\longrightarrow$

global_invariant *abstract_operators* *init_final* *directions*
eliminatelist *product_file*

global_invariant $\longrightarrow$ `GlobalInvar` `=` {*location_invariant_sequence* }`;`

location_invariant_sequence $\longrightarrow$ ε

| *location_invariant_sequence* `,` *location_invariant_pair*

location_invariant_pair $\longrightarrow$ {*location_predicate* , *data_predicate* }

abstract_operators $\longrightarrow$ `TakeConvex =` *boolean* ;
`WideSet[lc_] =` (*extraplations*)

extraplations $\longrightarrow$ (*extraplocs*)

| *boolean*

extraplocs $\longrightarrow$ *extraplocs* `||` `(lc ===` *location_predicate*)

| `(lc ===` *location_predicate*)

init_final $\longrightarrow$ `InitialState =` *state_predicate* ;
`Bad =` *state_predicate* ;

state_predicate $\longrightarrow$ *convex_state_predicate* `||` *state_predicate*

| *convex_state_predicate*

directions $\longrightarrow$ `Go := PrintTime[ Iterative[Forward] ];`

| `Go := PrintTime[ Iterative[Backward] ];`

eliminatelist $\longrightarrow$ ε

| `EliminateLocList =` {*automaton_name_sequence* };
`EliminateVarList =` {*variable_sequence* };

| `EliminateLocList =` {*automaton_name_sequence* };

| `EliminateVarList =` {*variable_sequence* };

product_file $\longrightarrow$ ε

| `ProductFile = "`*name* `.a"`

Appendix B

HyTech Input File Example

The following is the input file for the priority scheduler in Section 4.4.3.

```
(* Automaton name definitions *)
define(inter1, 1)
define(inter2, 2)
define(sched, 3)

(* Location name definitions *)
define(idle, 1)
define(task1, 2)
define(task2, 3)

(* Variable definitions*)
AnaVariables={x1, x2, y1, y2}
DisVariables={k1, k2}

(* Location definitions*)
```

```
locationo={1,1,3}
dif[sched,idle,y1]=0; dif[sched,idle,y1]=0
dif[sched,task1,y1]=1; dif[sched,task1,y2]=0
dif[sched,task2,y1]=0; dif[sched,task2,y2]=1
dif[inter1,idle,x1]=1; dif[inter2,idle,x2]=1
inv[l[inter1]==idle]=True    (* task1 *)
inv[l[inter2]==idle]=True    (* task2 *)
inv[l[sched]==idle]=True     (* idle  *)
inv[l[sched]==task1]=-4<=-y1 (* task1 *)
inv[l[sched]==task2]=-8<=-y2 (* task2 *)

(* Transition definitions*)
transitiono={1,1,11}
act[inter1,1]={ l[inter1]==idle && 10<=x1, int1,
 {l[inter1]->idle, x1->0, k1->k1+1}}
act[inter2,1]={ l[inter2]==idle && 20<=x2, int2,
 {l[inter2]->idle, x2->0, k2->k2+1}}
act[sched,1]={l[sched]==idle, int1,
 {l[sched]->task1, y1->0}}
act[sched,2]={l[sched]==idle, int2,
 {l[sched]->task2, y2->0}}
act[sched,3]={l[sched]==task1 && 1==k1 && 4==y1,
 {l[sched]->idle, k1->0, y1->0}}
act[sched,4]={l[sched]==task1 && 2<=k1 && 4==y1,
 {l[sched]->task1, k1->k1-1, y1->0}}
act[sched,5]={l[sched]==task1, int1,
 {l[sched]->task1}}
```

```
act[sched,6]={l[sched]==task1, int2,
 {l[sched]->task2, y2->0}}
act[sched,7]={l[sched]==task2 && 1==k2 && 0==k1 && 8==y2,
 {l[sched]->idle, k2->0, y2->0}}
act[sched,8]={l[sched]==task2 && 1==k2 && 1<=k1 && 8==y2,
 {l[sched]->task1, k2->0, y2->0}}
act[sched,9]={l[sched]==task2 && 2<=k2 && 8==y2,
 {l[sched]->task2, k2->k2-1, y2->0}}
act[sched,10]={l[sched]==task2, int1,
 {l[sched]->task2}}
act[sched,11]={l[sched]==task2, int2,
 {l[sched]->task2}}
syn[int1]={inter1, sched}
syn[int2]={inter2, sched}
AutomataNo=3

(* Verification definitions *)
InitialState=( l[inter1]==idle && l[inter2]==idle &&
 l[sched]==idle && 0==x1 && 0==x2 && 0==k1 && 0==k2 &&
 0==y1 && 0==y2 )
Bad = 3 <= k1 || 2 <= k2
GlobalInvar = True
TakeConvey = False
WideSet[lc_] := False
Go := PrintTime[ Iterative[Forward] ]
ProductFile = "psched2.a"
```

Bibliography

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking in dense real time. *Information and Computation*, 104(1):2–34, 1993.

[ACH93] R. Alur, C. Courcoubetis, and T.A. Henzinger. Computing accumulated delays in real-time systems. In C. Courcoubetis, editor, *CAV 93: Computer-aided Verification*, Lecture Notes in Computer Science 697, pages 181–193. Springer-Verlag, 1993.

[ACH+95] R. Alur, C. Coucoubetis, 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*, 138:3–34, 1995.

[ACHH93] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 209–229. Springer-Verlag, 1993.

[AD90] R. Alur and D.L. Dill. Automata for modeling real-time systems. In M.S. Paterson, editor, *ICALP 90: Automata, Languages, and Programming*, Lecture Notes in Computer Science 443, pages 322–335. Springer-Verlag, 1990.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AFH91] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the Tenth Annual Symposium on Principles of Distributed Computing*, pages 139–152. ACM Press, 1991.

[AH94] R. Alur and T.A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[AHH93] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the 14th Annual Real-time Systems Symposium*, pages 2–11. IEEE Computer Society Press, 1993. The full version appeared as Cornell Technical Report CSD-TR-95-1513.

[AHV93] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, pages 592–601. ACM Press, 1993.

[AL92] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 1–27. Springer-Verlag, 1992.

[AMP95] E. Asarin, Z. Manna, and A. Pnueli. Rechability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138(1):201–210, 1995.

[ASL93] P.J. Antsaklis, J.A. Stiver, and M. Lemmon. Hybrid system modeling and autonomous control systems. In *Proceedings of the 1992 Workshop on Hybrid Systems*, Lecture Notes in Computer Science 736, pages 366–392. Springer-Verlag, 1993.

[BER94] A. Bouajjani, R. Echahed, and R. Robbana. Verifying invariance properties of timed systems with duration variables. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *FTRTFT 94: Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science 863, pages 193–210. Springer-Verlag, 1994.

[BES93] A. Bouajjani, R. Echahed, and J. Sifakis. On model checking for real-time properties with durations. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science*, pages 147–159. IEEE Computer Society Press, 1993.

[BGM93] A. Back, J. Guckenheimer, and M. Myers. A dynamical simulation facility for hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 255–267. Springer-Verlag, 1993.

[BH95] J.P. Bowen and M.G. Hinchey. Ten commandments of formal methods. *IEEE Computer*, 28(4):56–63, 1995.

[BLL+95] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppall, a tool suite for symbolic and compositional verification of real-time systems. To appear, 1995.

[BLR95] A. Bouajjani, Y. Lakhnech, and R. Robbana. From duration calculus to linear hybrid automata. In *Proceedings of the Conference on Computer-Aided Verification*, Lecture Notes in Computer Science 939, pages 196–210. Springer-Verlag, 1995.

[BPV94] D. Bosscher, I. Polak, and F. Vaandrager. Verification of an audio-control protocol. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *FTRTFT 94: Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science 863, pages 170–192. Springer-Verlag, 1994.

[BR95] A. Bouajjani and R. Robbana. Verifying ω-regular properties for subclasses of linear hybrid systems. 1995. To appear at CAV.

[Bra95] M.S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67–100, 1995.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for the static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth Annual Symposium on Principles of Programming Languages*. ACM Press, 1977.

[CC92] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In *PLILP*, Lecture Notes in Computer Science 631, pages 269–295. Springer-Verlag, 1992.

[Cer92] K. Cerāns. Decidability of bisimulation equivalence for parallel timer processes. In G. von Bochmann and D.K. Probst, editors, *CAV 92: Computer-aided Verification*, Lecture Notes in Computer Science 663, pages 302–315. Springer-Verlag, 1992.

[CES81] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Design and synthesis of synchronization skeletons using branching-time temporal logic. In

Workshop on Logic of Programs, Lecture Notes in Computer Science 131. Springer-Verlag, 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth Annual Symposium on Principles of Programming Languages*. ACM Press, 1978.

[Che68] N.V. Chernikova. Algorithms for discovering the set of all solutions of a linear programming problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):283–293, 1968.

[CHR91] Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

[CHS93] Z. Chaouchen, M.R. Hanzen, and P. Sestoft. Decidability and undecidability results for duration calculus. In *STACS'93*, Lecture Notes in Computer Science. Springer-Verlag, 1993.

[Cor94] J.C. Corbett. Modeling and analysis of real-time Ada tasking programs. In *Proceedings of the 15th Annual Real-time Systems Symposium*. IEEE Computer Society Press, 1994.

[Cou81] P. Cousot. Semantics fundations of program analysis. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, pages 303–342. Prentice-Hall, 1981.

[DV95a] A. Deshpande and P. Varaiya. Information structures for control and verification of hybrid systems. In *Proceedings of the American Control Conference*, 1995.

[DV95b] A. Deshpande and P. Varaiya. Viable control of hybrid systems. In *Proceedings of the 1994 Workshop on Hybrid Systems and Autonomous Control*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[DW93] D.L. Dill and H. Wong-Toi. Using iterative symbolic approximation for timing verification. In T. Rus, editor, *Proceedings of the First AMAST Workshop on Real-time Systems*, 1993.

[DW95] D.L. Dill and H. Wong-Toi. Verification of real-time systems by successive over- and underapproximation. In P. Wolper, editor, *CAV 95: Computer-aided Verification*, Lecture Notes in Computer Science 939, pages 409–422. Springer-Verlag, 1995.

[DY95] C. Daws and S. Yovine. Verification of multirate timed automata with KRONOS: two examples. Technical Report Spectre-95-06, VERIMAG, apr 1995. To appear at RTSS.

[EGL92] U. Engberg, P. Gronning, and L. Lamport. Machanical verification of concurrent systems with TLA. In *Logic of Programs*, Lecture Notes in Computer Science. Springer-Verlag, 1992.

[FR75] J. Ferrante and C. Rackoff. A decision procedure for the first-order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.

[GKNY92] J. Guckenheimer, W. Kohn, A. Nerode, and A. Yakhnis. Hybrid systems papers for CDC92. Technical Report 92-31, Mathematical Science Institute, Cornell University, 1992.

[GL93] R.L. Grossman and R.G. Larson. Some remarks about flows in hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 317–356. Springer-Verlag, 1993.

[GL95] R.L. Grossman and R.G. Larson. An algebraic approach to hybrid systems. *Theoretical Computer Science*, 138(1):101–112, 1995.

[GNKJ95] X. Ge, A. Nerode, W. Kohn, and J. James. Distributed intelligent control theory of hybrid systems. In *Proceedings of the Fifth Annual Conference on AI, Simulation, and Planning in High Autonomy Systems*, pages 12–15. IEEE, 1995.

[GNRR93] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems*. Lecture Notes in Computer Science 736. Springer-Verlag, 1993.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In C. Courcoubetis, editor, *CAV 93: Computer-aided Verification*, Lecture Notes in Computer Science 697, pages 333–346. Springer-Verlag, 1993.

[HH95a] T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In *Proceedings of the Conference on Computer-Aided Verification*, Lecture Notes in Computer Science 939, pages 225–238. Springer-Verlag, 1995.

[HH95b] T.A. Henzinger and P.-H. Ho. HyTech: The Cornell Hybrid Technology Tool. In *Proceedings of the 1994 Workshop on Hybrid Systems and Autonomous Control*, Lecture Notes in Computer Science. Springer-Verlag, 1995. Also appeared as Cornell Technical Report CSD-TR-95-1521.

[HH95c] T.A. Henzinger and P.-H. Ho. A note on abstract-interpretation strategies for hybrid automata. In *Proceedings of the 1994 Workshop on Hybrid Systems and Autonomous Control*, Lecture Notes in Computer Science. Springer-Verlag, 1995. Priliminary version appeared as Cornell Technical Report CSD-TR-94-1437.

[HHF+94] J. He, C.A.R. Hoare, M. Franzle, M. Muller-Olm, E.R. Olderog, M. Schenke, M.R. Hansen, A.P. Ravn, and H. Rischel. Provably correct systems. In *FTRTFT'94*, Lecture Notes in Computer Science. Springer-Verlag, 1994.

[HHK95] M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. Computing simulations on finite and infinite graphs. To appear at FOCS, 1995.

[HHWT95a] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: The next generation. To appear at Proceedings of the 16th Annual Real-time Systems Symposium (RTSS), 1995.

[HHWT95b] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HYTECH. To appear at Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Aarhus, Denmark, 1995.

[HK95] T.A. Henzinger and P.W. Kopke. Hybrid automata with finite mutual simulations. Technical Report CSD-TR-95-1497, Cornell University, 1995.

[HKPV95] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual Symposium on Theory of Computing*, pages 373–382. ACM Press, 1995.

[HMP92] T.A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 226–251. Springer-Verlag, 1992.

[HMP93] T.A. Henzinger, Z. Manna, and A. Pnueli. Towards refining temporal specifications into hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 60–76. Springer-Verlag, 1993.

[HMP94] T.A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112(2):273–337, 1994.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[Hoo93] J. Hooman. A compositional approach to the design of hybrid systems. In *Proceedings of the 1992 Workshop on Hybrid Systems*, Lecture Notes in Computer Science 736, pages 121–148. Springer-Verlag, 1993.

[HPC95] M.R. Hansen, P.K. Pandya, and Z. Chaochen. Finite divergence. *Theoretical Computer Science*, 138(1):113–140, 1995.

[HPS83] D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.

[HRP94] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximation. In B. LeCharlier, editor, *SAS 94: Static Analysis Symposium*, Lecture Notes in Computer Science 864. Springer-Verlag, 1994.

[Hub95] B. B. Hubbard. Hybrid systems: the control theory of tomorrow? *SIAM NEWS*, 8(6):12–13, July 1995.

[HW95] P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In *Proceedings of the Conference on Computer-Aided Verification*, Lecture Notes in Computer Science 939, pages 381–394. Springer-Verlag, 1995.

[HWT95] T.A. Henzinger and H. Wong-Toi. Phase portrait approximation for hybrid systems. Submitted, 1995.

[KHMP94] A. Kapur, T.A. Henzinger, Z. Manna, and A. Pnueli. Proving safety properties of hybrid systems. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *FTRTFT 94: Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science 863, pages 431–454. Springer-Verlag, 1994.

[KJN+95] W. Kohn, J. James, A. Nerode, K. Harbison, and A. Agrawala. A hybrid system approach to computer-aided control engineering. *IEE Control Systems Magazine*, 15(2):14–25, 1995.

[KM91] R.P. Kurshan and K.L. McMillan. Analysis of digital circuits through symbolic reduction. *IEEE Transactions on Computer Aided Design*, 10(11):1356–1371, 1991.

[KN93] W. Kohn and A. Nerode. A hybrid systems architecture. In J.N. Crossley, J.B. Remmel, R.A. Shore, and M.E. Sweedler, editors, *Logic Methods*, Lecture Notes in Computer Science. Springer-Verlag, 1993.

[KNR95] W. Kohn, A. Nerode, and J.B. Remmel. Hybrid systems as Finsler manifolds: finite state control as approximation to connections. In *To appear at Proceedings of Hybrid System Workshop*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[KNRG94] W. Kohn, A. Nerode, J.B. Remmel, and X. Ge. Mutiple agent hybrid control: carrier manifolds and chattering approximations to optimal control. In *CDC94*, 1994.

[KNRY95] W. Kohn, A. Nerode, J.B. Rammel, and A. Yakhnis. Viability in hybrid systems. *Theoretical Computer Science*, 138(1):141–168, 1995.

[KPSY93] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: a class of decidable hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 179–208. Springer-Verlag, 1993.

[LA95] M. Lemmon and P.J. Antsaklis. Inductively inferring valid logical models of continuous-state dynamical systems. *Theoretical Computer Science*, 138(1):201–210, 1995.

[Lam87] L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.

[Lam93] L. Lamport. Hybrid systems in TLA+. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 77–102. Springer-Verlag, 1993.

[Lam94] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[LeV92] H. LeVerge. A note on chernikova's algorithm. Technical Report Research Report 635, IRISA, 1992.

[LGKN95] J. Liu, X. Ge, W. Kohn, and A. Nerode. A semi-autonomous multiagent decision model for a battlefield environment. To appear at Proceedings of Hybrid System Workshop, 1995.

[LH95] Y. Lakhneche and J. Hooman. Metric temporal logic with durations. *Theoretical Computer Science*, 138(1):201–210, 1995.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. In *Proceedings of the 12th Annual Symposium on Principles of Programming Languages*, pages 97–107. ACM Press, 1985.

[LPY95] K.G. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model-checking of real-time systems. To appear at RTSS, 1995.

[LSA93] M. Lemmon, J.A. Stiver, and P.J. Antsaklis. Event identification and intelligent hybrid control. In *Proceedings of the 1992 Workshop on Hybrid Systems*, Lecture Notes in Computer Science 736, pages 268–296. Springer-Verlag, 1993.

[LV92] N.A. Lynch and F. Vaandrager. Action transducers and timed automata. In R.J. Cleaveland, editor, *CONCUR 92: Theories of Concurrency*, Lecture Notes in Computer Science 630, pages 436–455. Springer-Verlag, 1992.

[LV93] N.A. Lynch and F. Vaandrager. Forward and backward simulations, part ii: timing-based systems. Technical Report CS-R9314, CWI, Amsterdam, 1993.

[McM93] K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic Publishers, 1993.

[MMP92] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 447–484. Springer-Verlag, 1992.

[MP93] Z. Manna and A. Pnueli. Verifying hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 4–35. Springer-Verlag, 1993.

[MPS95] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS 95*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[MS94] A.K. Martin and C.-J.H. Seger. Conservative approximations of hybrid systems. Submitted, 1994.

[MV94] J. McManis and P. Varaiya. Suspension automata: a decidable class of hybrid automata. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pages 105–117. Springer-Verlag, 1994.

[Ner92] J. Guckenheimerand A. Nerode. Simulation for hybrid systems and nonlinear control. In *CDC92*, pages 2980–2981, 1992.

[Ner93] A. Nerode. Hybrid system games: extraction of control automata with small topologies. Technical Report Technical Report 93-102, Mathematical Science Institute, Cornell University, 1993.

[NK93a] A. Nerode and W. Kohn. Models for hybrid systems: automata, topologies, controllability, observability. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 317–356. Springer-Verlag, 1993.

[NK93b] A. Nerode and W. Kohn. Multiple-agent hybrid control architecture. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 297–316. Springer-Verlag, 1993.

[NOSY93] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In R.L. Grossman,

A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 149–178. Springer-Verlag, 1993.

[NSY93] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30:181–202, 1993.

[NW88] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

[OSY94] A. Olivero, J. Sifakis, and S. Yovine. Using abstractions for the verification of linear hybrid systems. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pages 81–94. Springer-Verlag, 1994.

[PV94] A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pages 95–104. Springer-Verlag, 1994.

[PV95a] A. Puri and P. Varaiya. Driving safely in smart cars. Technical Report UCB-ITS-PRR-95-24, California PATH Research Report, August 1995.

[PV95b] A. Puri and P. Varaiya. Verification of hybrid systems using abstractions. To appear, 1995.

[QS81] J. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *Fifth International Symposium on Programming*, Lecture Notes in Computer Science 137, pages 337–351. Springer-Verlag, 1981.

[Rok93] T. G. Rokicki. *Representing and Modeling Circuits*. PhD thesis, Stanford University, 1993.

[RRH93] A.P. Ravn, H. Rischel, and K.M. Hansen. Specifying and verifying requirements of real-time systems. *IEEE Trans. on Software Eng.*, 1993.

[Seg93] C.-J. H. Seger. Voss – a formal hardware verification system user's guide. Technical Report Technical report 93-45, Department of Computer Science, The university of British Columbia, Vancouver, B.C., Canada, 1993.

[Var93] P. Varaiya. Smart cars on smart roads: problems of control. *IEEE Transactions on Automatic Control*, 38(2):195–207, 1993.

[WME92] F. Wang, A.K. Mok, and E.A. Emerson. Real-time distributed system specification and verification in asynchronous propositional temporal logic. In *Proceedings of the 12th International Conference on Software Engineering*, 1992.

[Wol88] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley Publishing Company, 1988.

[WT94] Howard Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems*. PhD thesis, Department of Computer Science, Stanford University, CA, December 1994.

[ZM95] Y. Zhang and A.K. Mackworth. Constraint nets: a semantic model for hybrid dynamic systems. *Theoretical Computer Science*, 138(1):211–239, 1995.