A State-Space Based Approach to the Specification and Verification of Hybrid Systems and Its Axiomatic Basis

A Thesis Submitted for the Degree of
Master of Science

by

Wu Dinghao

Advisor: Prof. Lü Jian
Major: Formal Method

Institute of Computer Software
Nanjing University, P. R. China

February 20, 2018

To My Family

Acknowledgements

I would like to thank my advisor Prof. Lü Jian for making this research project possible and for providing remarkable insights into many aspects of my academic life. His moral and financial support, technical ideas, friendship, and high standards of integrity have been inspirational to me and crucial to the completion of this thesis. Many thanks also to Dr. Tao Xianping and Dr. Yang Dajun for numerous interesting discussions and their pleasant companionship on our scientific trips.

A number of other people play an important role in the completion of this thesis. I own much to Liao Yu, who provides me with helpful advice about verification using the PVS specification and verification system. Sincere thanks go to Zhang Ming and Tang Bao for their useful comments and suggestions. I am grateful to Li Xin for his help to my practicing the LaTeX document preparation system. I am indebted to Wang Yan for correcting many spelling and grammar errors. Thanks also to my colleagues at the Institute of Computer Software, Nanjing University.

Moreover, I would like to express my special thanks to my parents and my brothers. It is their support, trust, and love that have enabled me to achieve successes such as this thesis.

Finally, I wish to thank my wife, Lily. Her unfailing love, support, and companionship have made the process of writing this thesis tolerable. In many times I have deeply doubted my work and myself, she has provided the encouragement to push on. I could not have reached this point without her, nor can I image a better soul with whom to face future challenges.

Dinghao Wu
Nanjing, P.R.China

Abstract

In recent years, the study of hybrid control systems has already become an important topic in the fields of computer science and control engineering. Hybrid control systems usually contain two distinct kinds of subsystems, namely time-evolving and event-driven subsystems, which interact with each other and operate in real time. In the field of control systems, this kind of hybridity is becoming more and more popular, as exemplified by robots, redundant flight control systems, and intelligent control systems *etc.* In hybrid control systems we not only need to control the time-evolving subsystems, but also need to control the event-driven subsystems. Therefore, the control in a hybrid control system usually contains two corresponding parts: the digital controllers and decision-maker.

In this thesis, we discuss the design methodology of hybrid control systems, and its theoretical foundation as well. We apply the principle of stepwise refinement to the design and analysis of hybrid control systems and propose a state-space based method for the design of digital controllers and synthesis of the decision-maker through decomposition of the corresponding control laws. The main procedure includes problem description, determination of state-space, partition of state-space and control refinement. The central idea of this method is illustrated by applying it to a typical example of hybrid systems: a water level monitoring system.

To explore the theoretical foundation of the state-space based approach, we analyze the generic characteristics of control laws. We conclude that the important characteristics of control laws are their entering points, leaving points and changing ranges. These three aspects can be specified by

making general assertions about the values that the relevant state variables constituting state-space will take before, after and during the action of control laws. To describe control laws in this sense, we introduce a new notation $P \{ C \mid W \} Q$, where P, R, W, and C are precondition, postcondition, duration-condition and control law, respectively. Furthermore, in order to specify and verify the design procedure, the classical Hoare logic system is extended as an axiomatic basis of this method, and a compositional proof system is formulated. We show that the system is sound in specifying and verifying hybrid control systems by constructing an operational model, Evolution Machine (EM), of the extended Hoare ($\mathcal{EH}$) logic system.

Following the proposed design procedure, we achieve the specifications for the example water level monitoring system using the extended Hoare logic. The correctness of the specifications has been mechanically verified by using the PVS specification and verification system. As a result, the correctness of design procedure can be verified to some extent, and the reliability of hybrid control systems is increased.

Contents

List of Figures

Chapter 1

Introduction

Nowadays, the exponential drop in price of computing power makes computers widely available. This leads to the widely use of computers in most control systems. A computer that acts as a logic decision unit processes input and provides output in digital form, *i.e.*, 0 and 1. It is also known as a discrete-time system. On the contrary, a system that processes only continuous-time data is called a continuous-time system, that is, it can be represented by mathematical functions. The combination of the computer science and control engineering has led to the emergence of computer-aided control engineering as a new research field. Therefore, it is common to have a mixture of both logic and continuous systems. This kind of system is known as *hybrid system* or *hybrid control system*. Unfortunately, this kind of hybridity causes extra complexity of the design, specification and verification of hybrid systems, which leads to the general aim of this thesis: the design, specification and verification of hybrid properties of hybrid systems.

1.1 What Are Hybrid Systems

The term *hybrid system* is used to describe a large and varied class of dynamical systems. A hybrid system consists of a collection of digital programs that interact with each other and with an analogue environment.

Examples of hybrid systems include medical equipment, manufacturing controllers, automotive controllers, and robots. The formal analysis of the mixed digital-analog nature of these systems requires a model that incorporates the discrete behavior of computer programs with the continuous behavior of environment variables, such as temperature and pressure.

A typical hybrid system is arranged in a hierarchy of two (or more) layers [1, 2], as shown in Figure 1.1. At each layer the system is modeled at a different level of abstraction. In the bottom layer the plant model is usually described by means of differential and/or difference equations. This level contains the actual plant and any conventional controllers working at the same level of abstraction. In the top layer the plant description is more abstract. Typical choices of description language at this level are finite state machines, fuzzy logic, Petri nets *etc.* Typically the controllers designed at this level are discrete event supervisory controllers [3]. The two levels communicate by means of an interface that plays the role of a translator between signals and symbols.

In other words, hybrid control systems usually contain two distinct kinds of subsystems, namely time-evolving and event-driven subsystems, which interact with each other through an interface and operate in real time. In the field of control systems, this kind of hybridity is becoming more and more popular, as exemplified by robots [4, 5], redundant flight control systems [6], and intelligent control systems [7] *etc.* Many researchers devote to this field and numerous papers are published, such as [8], [9], [10], [11], and [12] *etc.*

1.2 The Design Framework

In hybrid control systems we not only need to control the time-evolving subsystems, but also need to control the event-driven subsystems. This naturally follows then, that the controller has a hierarchical structure. In order to reflect this character more clearly, an appropriate framework of hybrid control systems has been presented and used in many papers, such as [1], [13] and [14], as shown in Figure 1.2.

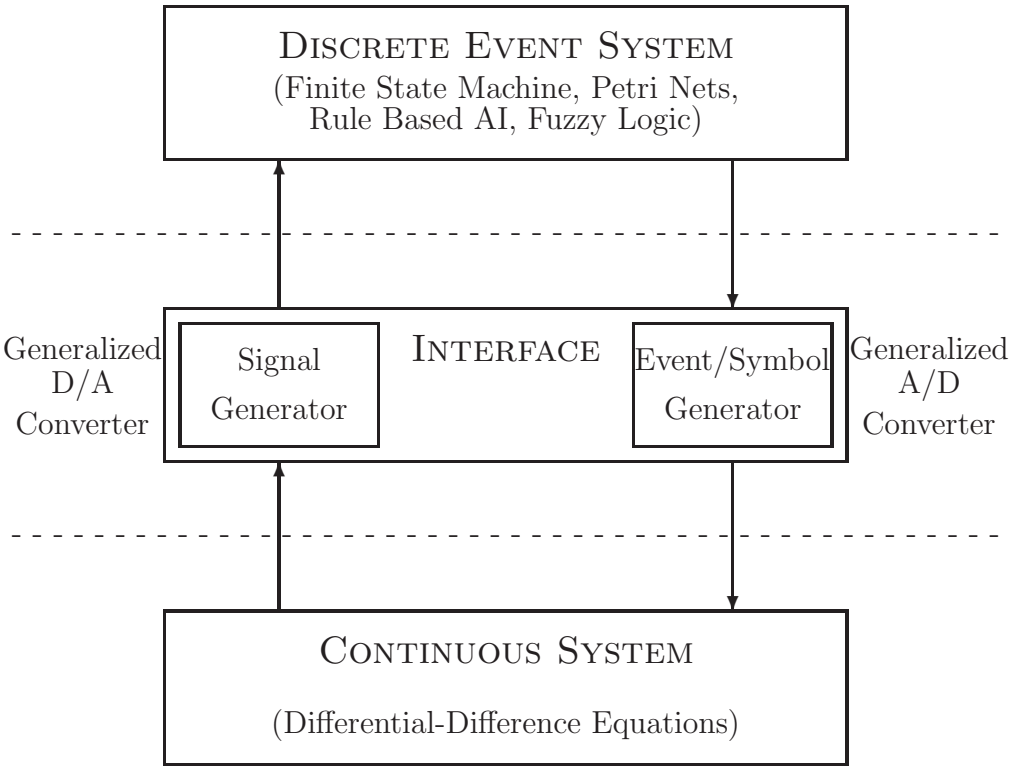

Figure 1.1: The Structure of Hybrid Control Systems

The structure can be divided into three layers. The lowest layer is the continuous time layer which consists of the plant to be controlled, the sensor for measuring the plant states and the actuators to implement the control action. The middle layer is the discrete time layer. It consists of digital controllers to calculate the control signals, which control the plant based on the measured plant states. The top layer is the discrete event layer that makes decisions through symbol manipulation and is driven by discrete events fed from the digital control loop. Among these three layers there are two interfaces: namely A-D and D-S interfaces. A-D interface includes A/D and D/A converters; and D-S interface includes D/S and S/D converters. A/D (D/A) converter transforms Analog (Digital) signals

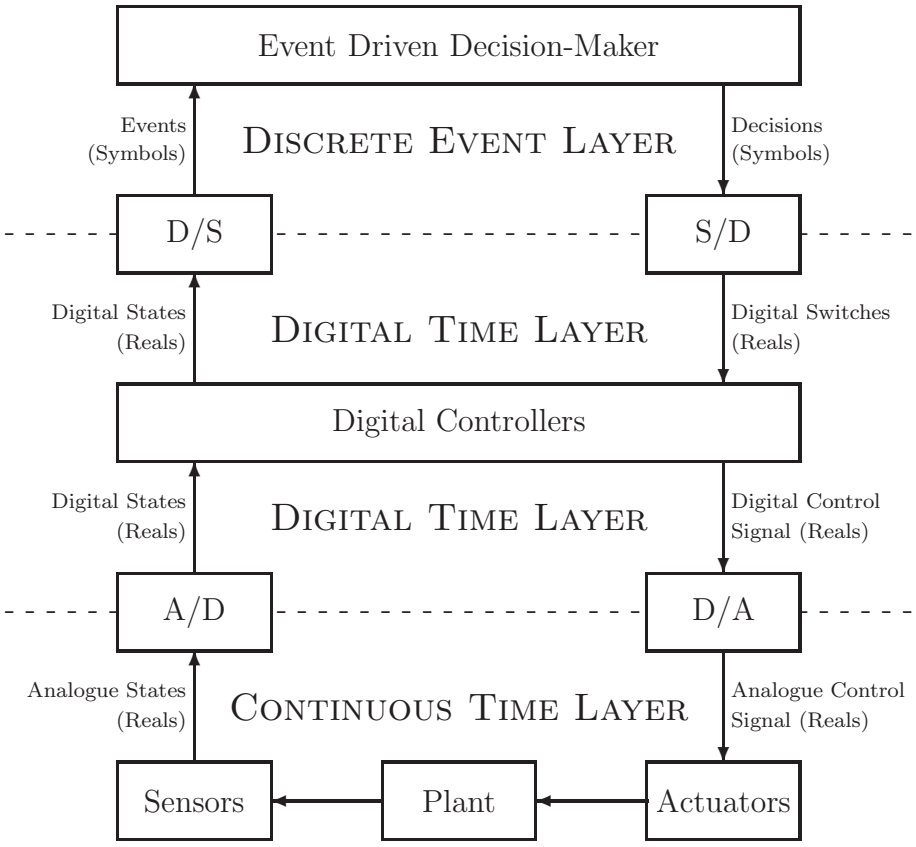

Figure 1.2: The Design Framework

to Digital (Analog) signals, and D/S (S/D) converter transforms Digital (Symbol) signals to Symbol (Digital) signals.

The control in a hybrid control system usually contains two parts: Decision-Maker (Dmer) and Digital Controllers (Dcers). The decision-maker determines the usage of digital controllers under different conditions, while the digital controllers use control laws to control physical systems.

1.3 Motivation

As stated above, there are two parts of control in a hybrid control system. The digital controller is located in the middle layer of the framework, which controls the plant through A-D interface, sensors and actuators. This part is very familiar to control experts. There are mature theories to support it. The other part is the decision-maker, which is located in the decision-maker box. It makes decision events according to the events fed from the digital control loops and conveys them into the digital control loops. This part is familiar to computers and control experts.

These two parts are interrelated and interact with each other. The decision-maker considers the main problems in a global view without any triviality, while the digital controller deals with such concrete problems as how to realize the decisions fed from the decision-maker, how to design the controllers and how to select the control parameters *etc.* No matter good or bad decisions will directly affect the realization of the digital control loops, in return the design of the digital controllers can offer valuable information to the decision-maker for making decisions.

For digital control systems, we have mature approaches based on the differential equation theory to support their analysis and design; and for discrete event systems, we also have several approaches like Ramadge and Wonham's Supervisory Control Theory [15, 16], Petri Nets *etc.*, to support their analysis and design. However, there are virtually less theories to support the control problem in hybrid systems which contain decision-maker and digital controllers [17].

In general, the design objective of hybrid control systems is to find a correct control strategy that restricts the behavior of the plant to some desired states. Therefore, an important problem is how to design systematically and correctly the decision-maker and digital controllers from the problem description in the above framework.

Many papers have been presented to discuss this problem. For instance, Dang Van Hung and Wang Ji [18] use I/O automata to discuss the method for decision-maker synthesis and analyze the timing properties of hybrid systems, and in [13], P. J. Antsaklis *et al.* propose a general plant

model and discuss its intelligent control. However, less attention is paid to the problem of systematic acquisition of the correct decision-maker and digital controllers from the problem description. In this thesis, we apply the principle of stepwise refinement to the design and analysis of hybrid control systems. As a result, a state-space based approach for systematic acquisition of the digital controllers and the corresponding decision-maker from practical problem is proposed. The key idea of this method is to decompose the whole system into several subsystems step by step through the partition of state-space and its control refinement with respect to the original control requirements until control laws can be designed directly. The hierarchical decomposition will make the design of control laws and the synthesis of the corresponding decision-maker easier. In order to guarantee the correctness of the decision-maker, Hoare logic is extended as the logical basis for the method. The main ideas are illustrated by a process control example of a water level monitoring system. Additionally, the correctness of specifications has been mechanically verified by using the PVS specification and verification system.

1.4 The Main Work

The achievements of the work presented in this thesis can be summarized as follows:

1. A state-space based approach to the design and analysis of hybrid systems. The main design procedure includes problem description, determination of state-space, partition of state-space and control refinement.

2. The extended Hoare logic system $\mathcal{EH}$, which is an axiomatic basis of the state-space based approach and is used to specify hybrid systems. By carefully scrutinizing the control laws, we conclude that the important characteristics of control laws are their entering points, leaving points and changing ranges. These three aspects can be specified by making general assertions about the values that the relevant state

variables constituting state-space will take before, after and during the action of control laws. Therefore, we extend the Hoare triple $P \{ S \} Q$ with a third predicate, called duration-condition, which expresses the behavior of control laws during the changing range. This leads to the extended Hoare logic formula $P \{ C \mid W \} Q$, where C is a control law; P, W, and Q are precondition, duration-condition, and postcondition, respectively.

3. The development of an $\mathcal{EH}$ proof assistant in PVS. The proof assistant is used to verify the specification of the example water level monitoring system.

1.5 The Plan of This Thesis

The rest of this thesis is organized as follows. In Chapter 2, using the principle of stepwise refinement, we develop a hierarchical design and synthesis method based on partition of the state-space and decomposition of the control laws. The design procedure is illustrated by a simple example. In Chapter 3, our extension of Hoare logic as the axiomatic basis of the method is introduced. Furthermore, we construct its operational model and thus its soundness is proved. Then in Chapter 4, we discuss the specification and verification of hybrid systems using the extended Hoare logic. In Chapter 5, the correctness of specifications is mechanically verified by using PVS. Finally, the concluding remarks are presented in Chapter 6.

Chapter 2

State-Space Partition and Design of Control Laws

A typical example of a hybrid system is a real-time computer system that controls physical processes. In this chapter, we propose a state-space based approach to the design of hybrid systems [19] and illustrate the design procedure by the analysis and design of a water level monitoring system as shown in Figure 2.1, which is taken from [20], and has been used in many papers such as [21] and [22].

2.1 Problem Description: The Water level Monitoring System

Consider a water level decision-maker that opens and closes a valve regulating the outflow of water from a container. The water container has an input vent, through which water flows in at a constant rate a, and an output vent, on which there is a valve. When the valve is full open, there is an outflow at a rate $c > a$, leading the decrease of overall water level $-b = c - a$ per unit time. The intention is to design a control system which keeps the water level inside the vessel between certain critical values 68 and 76. Besides, the initial value of water level is assumed to be 0 or

h. The water level in the vessel can be measured by means of a sensor and the level can be influenced by switching the valve on or off. Therefore, the main work is to design a decision-maker to decide the changes of the valve states (*i.e.* open or close) such that the water level is maintained between 68 and 76. Here, we use a sensor to measure the water level and an actuator to change the status of the valve. Both of them are assumed to be given.

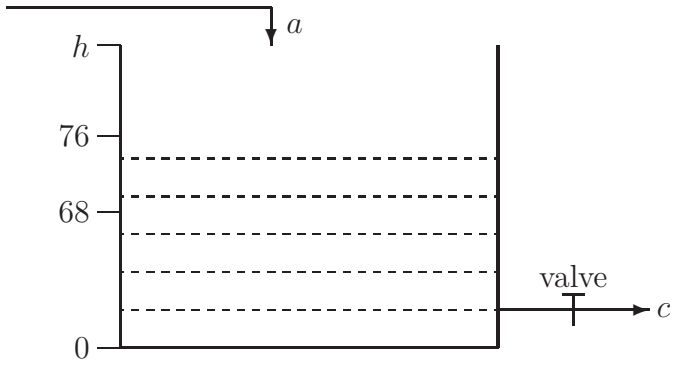

Figure 2.1: Water Container

2.2 Determination of State-Space

A state-space is a multi-dimensional space, of which each coordinate axis corresponds to a considered state variable. Therefore, all state variables and their value domains related to the practical problem should be determined. According to the example described previously, there are two factors that can be used to reflect the state of the problem at hand:

- Water level w: its domain is $W = [0,\ h]$
- State of valve d: its domain is $D = \{open,\ close\}$

So, the state space is $W \times D$.

2.3 Partition of State-Space

By state-space refinement, we mean that some reasonable cutting points are introduced and therefore a large space can be divided into several smaller spaces so that the design of control laws on the large space can be decomposed into the design of control laws on the smaller spaces.

The requirement in the problem description says "water level should be maintained between 68 and 76." Therefore, according to the requirement, two cutting points 68 and 76 are introduced. As a result, W can be naturally divided into three parts: $W' = \{[0,\ 68],\ [68,\ 76],\ [76,\ h]\}$. And state-space $W \times D$ is refined into $W' \times D$ as shown in Figure 2.2.

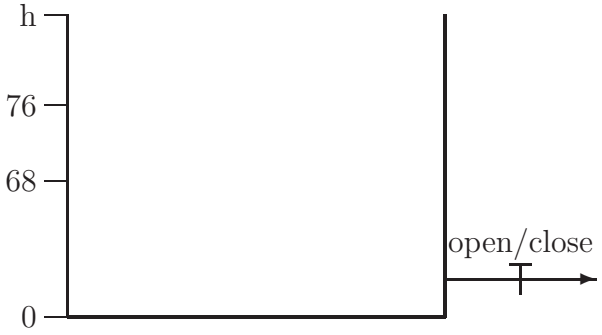

Figure 2.2: Partition of State-Space

2.4 Control Refinement

Based on the refined state-space $W' \times D$, we state the control refinement strategy as follows.

2.4.1 Choosing primary factors

Firstly, some primary factors are chosen and their control relations to their neighbors are studied. Then, other factors are considered when they are

needed. For example, the water level w is chosen as a primary factor in our problem because the aim of control is to maintain w between the certain values.

Based on the primary factors, the state-space is partitioned into several sub state-spaces and the adjacent relations between them are defined. For our example, we have three sub state-spaces:

$$S_1 : [0,\ 68] \times D$$

$$S_2 : [68,\ 76] \times D$$

$$S_3 : [76,\ h] \times D$$

The relation between them is

$$R = \{(S_1, S_2),\ (S_2, S_1),\ (S_2, S_3),\ (S_3, S_2)\}.$$

Therefore, we will consider the following six sub control laws C_1, C_2, C_3, C_4, C_5 and C_6 as shown in Figure 2.3 and study the control relations between them in next step.

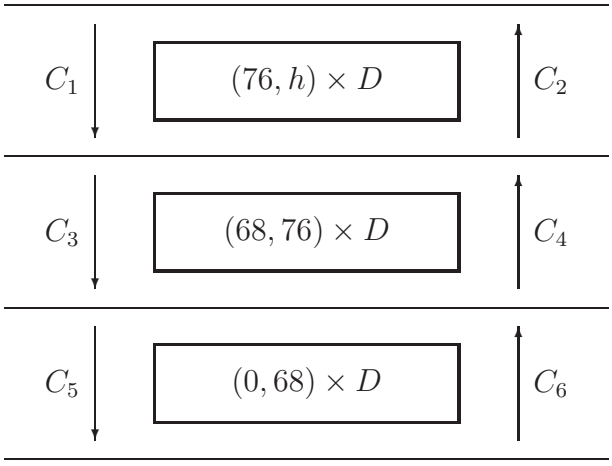

Figure 2.3: Control Laws and Their Relations

2.4.2 Identifying key sub state-space

In this step, some key sub state-spaces are identified and the control transformation from non-key sub state-spaces to key sub state-spaces is studied. For our example, $[68,\ 76] \times D$ is chosen as a key sub state-space because the requirement says that water level should be maintained between 68 and 76. Based on the key sub state-space, the aim of the control laws (or corresponding digital controllers) for non-key sub state-spaces is to reduce the corresponding non-key sub state-spaces to the key sub state-space.

Using this design strategy, the control laws C_1, C_3, C_4, and C_6 are needed only as shown in Figure 2.4. As a result, the original control requirement is refined into several small control requirements.

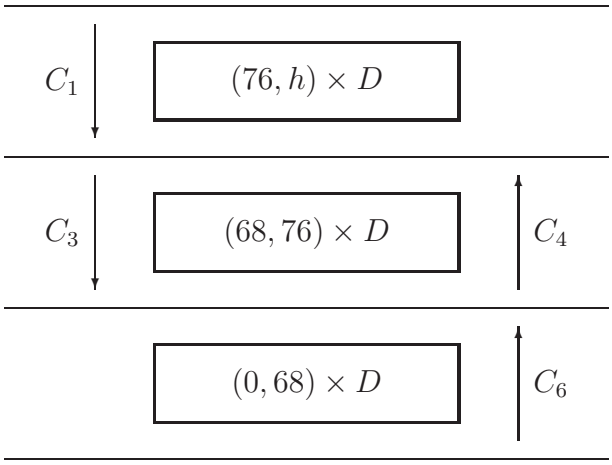

Figure 2.4: Desirable Control Laws and Their Relations

2.4.3 Synthesis of Decision-Maker

Each control law in the sub state-spaces can be specified by a triple: $\langle$*entering point*, *changing range*, *leaving point*$\rangle$. The entering point gives the initial condition that the corresponding control law can act on.

The leaving point gives the resulting description when the corresponding control law leaves the sub state-space. And the changing range gives the range of state change when the control law acts.

According to the above description method, four desirable control laws C_1, C_3, C_4, and C_6 are specified as follows:

$$C_1 : \langle w = h,\ w \in [76, h],\ w = 76 \rangle$$

$$C_3 : \langle w = 76,\ w \in [68, 76],\ w = 68 \rangle$$

$$C_4 : \langle w = 68,\ w \in [68, 76],\ w = 76 \rangle$$

$$C_6 : \langle w = 0,\ w \in [0, 68],\ w = 68 \rangle$$

Based on the specifications for four control laws C_1, C_3, C_4, and C_6, the decision-maker can be described informally as follows: Firstly, we use C_1 and C_6 to reduce the non-key sub state-spaces to the key state-space. Then, we use C_3 and C_4 continuously to keep water level between 68 and 76.

2.5 Towards a Hierarchical Design and Synthesis Method

Up to now, the design of control laws and synthesis of decision-maker of the original control problem can be reduced to the design of control laws and synthesis of decision-maker of several sub problems. The specifications for control laws in the sub state-spaces can be used as new control problems in the following design. Therefore, the above design procedure can be applied recursively until the control specification can be directly satisfied by a control law. Consequently, we have a hierarchical design method for the design of control laws and synthesis of the corresponding decision-maker.

Chapter 3

An Axiomatic Basis for the State-Space Based Method

In the bottom layer of hybrid system architecture, control laws or physical laws are usually described by means of differential and/or difference equations. In this way, it is difficult to formalize control laws or to compose a more complicated control law from some simpler ones. As stated above, if viewed at a high abstract level, each control law can be specified by a triple $\langle$*entering point*, *changing range*, *leaving point*$\rangle$. In other words, a control law is composed of three parts: an initiate state, a final state and an evolving process, which may satisfy some certain conditions, from the initiate state to the final state. Therefore, we can take control laws as the elementary design units of hybrid systems, and acquire more complicated control laws through composition of simpler control laws.

To make the formalization of control laws easier, we consider a simple sequential language. In Section 3.1 we give the syntax of this language and informal meaning of syntactic domains. Next, in Section 3.2, Hoare logic is extended as the logical basis of the state-space based method, and furthermore, we define the axiomatic semantics of the language by it. Then, in Section 3.3, in order to construct a model of the extended Hoare logic system we describe the operational semantics of the language. Finally, in Section 3.4, we prove the soundness of the extended Hoare logic system.

3.1 Syntax and Informal Meaning

The syntactic domains and their informal meaning are given below:

- **Atomic controller set** $Actrl$ When designing hybrid control system by using the state-space based approach stated previously, we decompose the complicated control laws until they are simple enough to be designed directly. The depth of the decomposition is, however, related to the certain problem. Therefore, the sub control laws of the deepest level of the decomposition are assumed to be given. The corresponding digital controllers conforming to them are called atomic controllers, which are denoted as A. And $Actrl$ is the set of atomic controllers, that is, $A \in Actrl$.

- **Boolean expression set** $Bexp$ is the set of Boolean expressions, which are denoted as B. Hence, $B \in Bexp$.

- **Controller set** $Ctrl$ The elements of the controller set $Ctrl$ are defined inductively by the BNF expression as follows:

  $$C ::= \mathsf{skip} \mid A \mid C; C \mid (\textbf{if } B \textbf{ then } C \textbf{ else } C) \mid (\textbf{while } B \textbf{ do } C)$$

  Hence, simple controllers can compose complicated controller by compositional, conditional and iterative rules.

3.2 The Extended Hoare Logic System $\mathcal{EH}$

In order to verify the correctness of the design of control laws and synthesis of the corresponding decision-maker, the proof system of Hoare logic (see [23], [24], and [25]), which is conventionally called Hoare system $\mathcal{H}$, is extended as logical basis for this method.

The important characteristics of the above control laws are their entering points (or initiate states), leaving points (or final states) and changing ranges (or called evolving processes). These three aspects can be specified

by making general assertions about the values that the relevant state variables constituting state-space will take before, after and during the action of control laws. To describe the control law in this sense, we introduce a new notation

$$P \{ C \mid W \} Q$$

where P, Q, and W are predicates. It has the following interpretation:

If execution of C is begun in a proper state satisfying P, then W is satisfied during the action of C, and Q will be satisfied on its completion. In the case when Q is a contradiction which is false for all states, the control law C acts and the assertion W is satisfied forever.

P is called the precondition or input assertion of C; Q the postcondition or output assertion; and W the duration-condition. Corresponding to the classical Hoare logic system $\mathcal{H}$, we have the following proof axioms and rules of extended Hoare logic formal system $\mathcal{EH}$ where P, Q, R, Inv and W are assertions and C is a control law:

(i) $P\{\mathsf{skip}|\mathsf{false}\}P$
This rule states that execution of skip does nothing.

(ii) $P\{A|W\}Q$
As mentioned earlier, atomic control laws are assumed to be given. Therefore, the semantics of them are also given. In general, an inference rule has a list of hypotheses and a conclusion separated by a horizontal line. Thus we have

(iii) Rule of Composition

$$\frac{P\{C_1|W_1\}R,\ R\{C_2|W_2\}Q}{P\{C_1;C_2|W_1;W_2\}Q}$$

where $C_1;C_2|W_1;W_2$ means that there exists a cutting point in the states sequence produced by the control law $C_1;C_2$, such that W_1 holds for every state before the cutting point and W_2 holds for every

state after the cutting point. Suppose the cutting point were the state that between the final state of C_1 and the initial state of C_2 , thus W_1 holds for every state during the action of C_1 and W_2 holds for every state during the action of C_2.

(iv) Conditional Rule

$$\frac{P \wedge B \ \{C_1|W_1\} \ Q, \ P \wedge \neg B \ \{C_2|W_2\} \ Q}{P \ \{(\textbf{if} \ B \ \textbf{then} \ C_1 \ \textbf{else} \ C_2)|W_1 \vee W_2\} \ Q}$$

(v) Iteration Rule

$$\frac{Inv \wedge B \ \{C|W\} \ Inv}{Inv \ \{(\textbf{while} \ B \ \textbf{do} \ C) \ |W\} \ Inv \wedge \neg B}$$

where Inv is an invariant which is true before and after each iteration of the loop.

(vi) Rule of Consequence

$$\frac{P \Rightarrow P_1, \ P_1\{C|W\}Q}{P\{C|W\}Q}$$

$$\frac{P\{C|W_1\}Q, \ W_1 \Rightarrow W}{P\{C|W\}Q}$$

$$\frac{P\{C|W\}Q_1, \ Q_1 \Rightarrow Q}{P\{C|W\}Q}$$

The above three rules can be abbreviated to

$$\frac{P \Rightarrow P_1, \ P_1\{C|W_1\}Q_1, \ W_1 \Rightarrow W, \ Q_1 \Rightarrow Q}{P\{C|W\}Q}$$

Compared with the Hoare logic, extension of the above logic made exists in two aspects. Firstly, the object that the above logic attempts to specify is the control law instead of statement. The second difference is the above logic extends Hoare logic with another component W called duration-condition, which expresses the behavior of the control law during the changing range.

The soundness of the above logic can be understood informally and formally. Informally speaking, the soundness of the above logic can be obtained according to the intuitive meaning of $P\{C|W\}Q$ and various control structures. Formally speaking, the soundness of the above logic can be established by showing that it has a model. Actually, this model can be defined as the formal operational semantics of $P\{C|W\}Q$ and various control structures. This is discussed in Section 3.3 and 3.4.

3.3 Operational Semantics

In order to describe the behavior or semantics of controllers and construct the model of the extended Hoare system $\mathcal{EH}$ introduced above, we extend the operational approach presented in [26] and further developed in [27] as follows.

3.3.1 Evolution Relation

To interpret the controllers of the language specified in Section 3.1, firstly an abstract evolution machine is introduced. Next, an *evolution relation* $\rightsquigarrow$ between so-called *configurations* of the abstract machine is specified, then the meaning or semantics of programs is defined with the help of $\rightsquigarrow$. Depending on the definition of configurations, the evolution relation $\rightsquigarrow$ can model executions at various levels of details.

We choose here a "high level" view of an execution, where a configuration is simply a pair $\langle C, s\rangle$ consisting of a controller C and a state s. Let $State$ be the set of all the states, and let $Ctrl$ be the set of all controllers. Thus $C \in Ctrl$, and $s \in State$. The state consists of current values of

the control variables, and the configuration consists of the controller to be executed and the current state. When the execution of a program has completed, the controller remaining to be executed of the configuration is empty. In this case, the configuration consists of only a state. So, let

$$Con = (Ctrl \times State) \cup State$$

denote the set of all configurations, and let *con* be a member of Con, thus $con \in Con$.

To illustrate the execution process of a controller, we introduce the concept of evolution between two configurations. Evolution means a *continuous* evolving process from one configuration to another. Usually, the evolution process of a controller is composed of several evolution processes of sub-controllers. Thus, the execution process of a controller can be described by means of evolution between configurations, and the action of abstract evolution machine by means of the evolution rules between configurations. Intuitively, an *evolution*

$$\langle C_1, s_1 \rangle \ \rightsquigarrow \ \langle C_2, s_2 \rangle$$

means: executing C_1 for a certain time in a proper state s_1 can lead to the state s_2 with C_2 being the remainder of C_1 that still to be executed. And in the meantime, the evolving process will satisfy certain control laws. Besides, an evolution is a continuous process, which is the main difference compared with the *transition relation* presented in [27].

Consequently, we propose a formal proof system, called an *evolution system*, which consists of axioms and rules about evolutions. Using it, we specify the evolution relation $\langle C, s \rangle$ by induction on the structure of programs. Correspondingly, we have the following evolution axioms and rules where s is a proper state:

(i) $\langle \mathsf{skip}, s \rangle \ \rightsquigarrow \ s$

This rule states that execution of skip does nothing. For the atomic controller A, we have

(ii) $\langle A, s\rangle \rightsquigarrow \mathcal{A}[\![A]\!]$
where $\mathcal{A}$ is the semantic "function" of the atomic controllers.

(iii) Rule of Composition

a) $$\frac{\langle C_1,s\rangle \rightsquigarrow \langle C_1',s'\rangle}{\langle C_1;C_2,s\rangle \rightsquigarrow \langle C_1';C_2,s'\rangle}$$

b) $$\frac{\langle C_1,s\rangle \rightsquigarrow s'}{\langle C_1;C_2,s\rangle \rightsquigarrow \langle C_2,s'\rangle}$$

The above two rules state that, a composition controller $C_1; C_2$ is executed by first executing C_1 and then executing C_2. They can be abbreviated to the following rule:

$$\frac{\langle C_1, s\rangle \rightsquigarrow \langle C_1', s'\rangle \mid s'}{\langle C_1; C_2, s\rangle \rightsquigarrow \langle C_1'; C_2, s'\rangle \mid \langle C_2, s'\rangle}$$

This rule shows the principle of stepwise refinement. Using it we can design control laws by decomposition.

(iv) Conditional Rule

a) **if** $\mathcal{B}[\![B]\!](s) = \mathsf{true}$, then
$\langle(\textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2), s\rangle \rightsquigarrow \langle C_1, s\rangle$,

b) **if** $\mathcal{B}[\![B]\!](s) = \mathsf{false}$, then
$\langle(\textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2), s\rangle \rightsquigarrow \langle C_2, s\rangle$.

where $\mathcal{B}$ is the semantic function of Boolean expressions.

(v) Iteration Rule

a) **if** $\mathcal{B}[\![B]\!](s) = \mathsf{true}$, then
$\langle(\textbf{while } B \textbf{ do } C), s\rangle \rightsquigarrow \langle C; (\textbf{while } B \textbf{ do } C), s\rangle$,

b) **if** $\mathcal{B}[\![B]\!](s) = \mathsf{false}$, then
$\langle(\textbf{while } B \textbf{ do } C), s\rangle \rightsquigarrow \langle \mathsf{skip}, s\rangle$.

An evolution $\langle C_1, s_1\rangle \rightsquigarrow \langle C_2, s_2\rangle$ is possible if and only if it can be deduced in the above evolution system. Note that the skip controller, atomic controllers and evaluation of Boolean expressions are all executed in one step. This "high level" view abstracts from all details of the evaluation of expressions in the execution of atomic controllers. Consequently, this is a high-level semantics. In other words, we can verify the design steps with the specifications of components by using it but without knowing their implementation.

To describe the effect of finite evolution sequences we use the transitive, reflexive closure $\rightsquigarrow*$ of the evolution relation $\rightsquigarrow$:

$$\langle C, s\rangle \rightsquigarrow* \langle C', s'\rangle$$

holds when there exist configurations $\langle C_1, s_1\rangle, \cdots, \langle C_n, s_n\rangle$ with $n \geq 0$ such that

$$\langle C, s\rangle = \langle C_1, s_1\rangle \rightsquigarrow \cdots \rightsquigarrow \langle C_n, s_n\rangle = \langle C', s'\rangle$$

holds. In the case when $n = 0$, $\langle C, s\rangle = \langle C', s'\rangle$ holds. The transitive, reflexive closure states the reachability of the evolution relation. Therefore, we have the following axiom and rule.

(vi) Rule of Reachability

a) $con \rightsquigarrow* con$,

b) $$\frac{con \rightsquigarrow* con',\ con' \rightsquigarrow con''}{con \rightsquigarrow* con''}.$$

3.3.2 Properties of Evolution Relation

The evolution relation $\rightsquigarrow$, as defined above, satisfies several simple properties that are used in the sequel.

Lemma 3.3.1 (Determinism) *For any controller C and a proper state s , there is exactly one configuration evolved from the configuration $\langle C, s\rangle$ one step. That is, if $con \rightsquigarrow con_i$ $(i = 1, 2)$ then $con_1 = con_2$.*

Proof. Due to the form of the above evolution system, it is sufficient to prove that all axioms and proof rules of it are deterministic. Then the result can be followed by the induction on the length of proofs. The details of the proof are omitted here.

However, the reachability relation $\rightsquigarrow *$ is not a deterministic relation, though it has the Church-Rosser property [28]. For any controller C which is terminable and a proper state s, there is one "final" configuration, which cannot evolve anymore, *i.e.*, cannot evolve from the configuration $\langle C, s\rangle$ one or several steps. In the case when the controller C is not terminable, we introduce a configuration "$\perp$" to denote this state. Note that "$\perp$" is also a proper state or configuration. Therefore, we have the following lemma.

Lemma 3.3.2 (Church-Rosser property) *if $con \rightsquigarrow * con_i$ $(i = 1, 2)$ then there exists one configuration con', such that $con_i \rightsquigarrow * con'$ $(i = 1, 2)$.*

3.3.3 Evolution Machine

Based on the above properties of the evolution relation $\rightsquigarrow$, an *Evolution Machine (EM)* is introduced.

Definition 3.3.1 (Evolution Machine) *For any controller C,*

$$EM(C)(s) = s' \text{ iff } \langle C, s\rangle \rightsquigarrow * s'.$$

By the Lemma 3.3.2, there exists only one configuration s', no matter whether it is a final configuration or the interminable configuration "$\perp$", such that $\langle C, s\rangle \rightsquigarrow * s'$. This ensures the unique result of $EM(C)(s)$. According to the above definition, we have the following properties of EM. Since the proofs are obvious, they are omitted here.

Theorem 3.3.1 $EM(\mathsf{skip}) = \mathsf{id}$, *where* id *is the identity function of the states set State, that is,* $\mathsf{id}(s) = s$.

Theorem 3.3.2 $EM(A) = \mathcal{A}[\![A]\!](s)$.

Theorem 3.3.3 $EM(C_1; C_2) = EM(C_2) \circ EM(C_1)$. *For any controllers* C_1 *and* C_2, *we define the composition* $EM(C_2) \circ EM(C_1)$, *provided that* $EM(C_1)$ *ends in the same case that* $EM(C_2)$ *starts with, as follows:*

$EM(C_2) \circ EM(C_1)(s) = s'$ iff
$\exists s''.\ s'' = EM(C_1)(s),$ *and* $s' = EM(C_2)(s'')$.

Theorem 3.3.4

$$EM(\textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2) = \mathsf{cond}(\mathcal{B}[\![B]\!],\ EM(C_1),\ EM(C_2)).$$

where the notation cond, *called conditional operator, is defined as follows:*

$$\mathsf{cond}(b,\ EM_1,\ EM_2)(s) = \begin{cases} EM_1(s), & \text{if } b(s) = \mathsf{true}, \\ EM_2(s), & \text{if } b(s) = \mathsf{false}. \end{cases}$$

Theorem 3.3.5

$$EM(\textbf{while } B \textbf{ do } C) = \mathsf{cond}(\mathcal{B}[\![B]\!],\ EM(C; \textbf{while } B \textbf{ do } C),\ \mathsf{id}).$$

3.4 Soundness of the Extended Hoare Logic System $\mathcal{EH}$

Up to now, the meaning of inductive propositions and the soundness of the extended Hoare logic formal system $\mathcal{EH}$ are both informally illuminated. According to the model theory, a formal system is sound if it has a model. We show that the Evolution Machine (EM), which is introduced in Section 3.3, is a model of the extended Hoare system $\mathcal{EH}$. Thus, the soundness of $\mathcal{EH}$ is proved. This goal is reached if we can show that provability of a correct formula in the proof system $\mathcal{EH}$ implies its truth.

The assertions such as P, Q, and W *etc.* used previously constitute an assertion language $\mathcal{L}$ which is included in the proof system $\mathcal{EH}$. Let $\mathcal{I}$ be an interpretation of $\mathcal{L}$, and we might suppose

$$\mathcal{I}[\![B]\!] = \mathcal{B}[\![B]\!] \quad (B \in Bexp)$$

as well. In the case when the controller is interminable, we have

$$\mathcal{I}[\![\bot]\!] = \mathcal{B}[\![\bot]\!] = \mathsf{true}$$

We express the timing behavior of a hybrid system from a viewpoint of an external observer with his own clock. Thus, the observable behavior of a system is described in terms of a single, conceptual, global clock. Here we use a time domain Time that equals the nonnegative real numbers. The operational interpretation $\mathcal{O}$ of system $\mathcal{EH}$ is defined by

Definition 3.4.1 $\mathcal{O}[\![P\{C|W\}Q]\!] \stackrel{def}{=}$
$$\forall\ s, s'.\ (\mathcal{I}[\![P]\!](s) \wedge s' = EM(C)(s)) \Rightarrow \mathcal{I}[\![Q]\!](s')$$
$$\wedge \forall\ t \in \textsc{Time}.\ t_s < t < t_{s'} \Rightarrow \mathcal{I}[\![W]\!](s_t),$$

where, s is a proper state; P is an assertion; $\mathcal{I}[\![P]\!](s)$ is the abbreviation of $\mathcal{I}[\![P]\!][s[x_i]/x_i]_{i=1}^n$ ($x_1, x_2, \cdots, x_n$ are all variables of the assertion P), and the expression $P[s[x_i]/x_i]_{i=1}^n$ is formed by simultaneously substituting $s[x_i]$, which are the values of x_i in the proper state s, for all of free variables x_i

of the assertion P respectively; EM is the evolution machine; $t \in \textsc{Time}$ is a time variable; $t_s \in \textsc{Time}$ is the time of the proper state s, in particular, $t_\perp = \infty$; s_t is a proper state at time t.

Particularly, in the case when $\mathcal{I}[\![W]\!] = \mathcal{B}[\![W]\!] = \mathsf{false}$, the controller C has no operations. Hence, we have

Definition 3.4.2 $\mathcal{O}[\![P\{C \mid \mathsf{false}\}Q]\!] \stackrel{def}{=} P \Rightarrow Q$.

Definition 3.4.3 $\models P\{C|W\}Q \quad \text{iff} \quad \mathcal{O}[\![P\{C|W\}Q]\!] \equiv \mathsf{true}$.

Theorem 3.4.1 (Soundness) $\vdash P\{C|W\}Q \quad \text{implies} \quad \models P\{C|W\}Q$.

The proof of the soundness is given in Appendix A.

3.5 Summary

This chapter is concerned with the theoretical basis of the analysis and design approach we proposed in Chapter 2. We extend conventional Hoare logic as an axiomatic basis of the design approach and prove the soundness of the extended Hoare ($\mathcal{EH}$) logic system [29, 30, 31]. In the next chapter, specifications are achieved by using the design approach and $\mathcal{EH}$.

Chapter 4

Specification and Verification

Following the analysis and design procedure presented in Chapter 2 and the extended Hoare logic introduced in Chapter 3, the control refinement and formal specifications of the example water level monitoring system are achieved in this chapter; additionally, the correctness of control refinement and specifications are verified in the extended Hoare logic system $\mathcal{EH}$.

4.1 Requirement Specifications

Following the state-space based approach and based on the analysis in Chapter 2, the original control problem of the example water level monitoring system can be formalized as follows:

- State-Space
  $\mathit{Water\text{-}level} = \{w | w \in \mathit{Real} \ \wedge \ 0 \leq w \leq h\}$
  $\mathit{Valve\text{-}state} = \{\mathit{open}, \ \mathit{close}\}$
  $\mathit{State\text{-}space} = \ \mathit{Water\text{-}level} \times \ \mathit{Valve\text{-}state}$
- Initial-states: $w = 0 \vee w = h$
- Control requirement: $CR_1; CR_2$

where

$w = 0 \vee w = h \ \{CR_1 | w \in [76,\ h] \vee w \in [0,\ 68]\}\ w = 68 \vee w = 76$
$w = 68 \vee w = 76 \ \{CR_2 | w \in [68,\ 76]\}$ false

4.2 Specifications for the Sub Control Requirements

In Chapter 2, we have obtained four key sub control laws C_1, C_3, C_4 and C_6. Using the new notation, $P \ \{ \ C \mid W \ \} \ Q$, introduced in the extended Hoare logic system $\mathcal{EH}$ (Section 3.2), the four key sub control laws can be specified as follows:

$w = h \ \{C_1 | w \in [76,\ h]\}\ w = 76$
$w = 76 \ \{C_3 | w \in [68,\ 76]\}\ w = 68$
$w = 68 \ \{C_4 | w \in [68,\ 76]\}\ w = 76$
$w = \ 0 \ \{C_6 | w \in [\ 0,\ 68]\}\ w = 68$

4.3 Specification for the Decision-Maker

The synthesized decision-maker can be described formally by using the small language including sequential, conditional and loop statements which was introduced in Section 3.1. According to the control refinement strategy and the informal analysis in Section 2.4, we specify the decision-maker to the following program.

if $w = h$ **then** C_1 **else** C_6;
while true **do**
    begin
        if $w = 68$ **then** C_4 **else** C_3
    end.

4.4 Verification of Control Refinement

By means of the formal axioms and rules in the extended Hoare logic system $\mathcal{EH}$, it can be shown that the total effects of the control laws C_1, C_3, C_4 and C_6 under the control of the above decision-maker satisfy the requirement specifications.

By Consequence Rule and C_1, we have

$$(w = 0 \vee w = h) \wedge w = h$$
$$\{C_1 | w \in [76,\ h]\}$$
$$(w = 68 \vee w = 76)$$

By consequence Rule and C_6, we have

$$(w = 0 \vee w = h) \wedge w \neq h$$
$$\{C_6 | w \in [0,\ 68]\}$$
$$(w = 68 \vee w = 76)$$

Using Conditional Rule, We have

$$(w = 0 \vee w = h)$$
$$\{(\textbf{if } w = h \textbf{ then } C_1 \textbf{ else } C_6) | w \in [0, 68] \vee w \in [76, h]\}$$
$$(w = 68 \vee w = 76)$$

So CR_1 is satisfied.

Using Conditional Rule, we have

$$(w = 68 \vee w = 76)$$
$$\{(\textbf{if } w = 68 \textbf{ then } C_4 \textbf{ else } C_3) | w \in [68, 76]\}$$
$$(w = 68 \vee w = 76)$$

By Iteration Rule, we have

$(w = 68 \vee w = 76)$
$\{(\textbf{while } \mathsf{true} \textbf{ do if } w = 68 \textbf{ then } C_4 \textbf{ else } C_3 \textbf{ end}) | w \in [68, 76]\}$
false

So CR_2 is satisfied.

4.5 Summary

In Chapter 2, we have described the design procedure and got a hierarchical design method. Following it, we obtain the specification of the example water level monitoring system in this chapter. The correctness of specifications has been verified by means of the extended Hoare logic system. However, the proof of correctness is not mechanically checked. In the next chapter, the correctness of specification is mechanically proved by using PVS specification and verification system.

Chapter 5

Verification Using PVS

In the earlier chapters, we have described our approach to the formalization of hybrid system behavior and illustrated it by a typical example. Further, the correctness proofs for the specifications have been sketched. These proofs are mechanically verified by using the PVS specification and verification system [32, 33] in this chapter. Firstly, we specify the extended Hoare ($\mathcal{EH}$) logic system, as given in Chapter 3, and water level monitoring system to PVS specifications; then prove that the specifications satisfy the control requirements using PVS.

5.1 An Introduction to PVS

PVS stands for "Prototype Verification System," and as the name suggests, it is a prototype environment for specification and verification based on high-order logic. It consists of a specification language integrated with support tools and a theorem prover. PVS tries to provide the mechanization needed to apply formal methods both rigorously and productively.

The primary purpose of PVS is to provide formal support for conceptualization and debugging in the early stages of the lifecycle of the design of a hardware or software system. In these stages, both the requirement and designs are expressed in abstract terms that are not necessarily executable.

The best way to analyze such an abstract specification is to attempt proofs for the desirable consequences of the specification. In this regard, PVS has been experienced that such attempted proofs of *putative theorems* very quickly highlight even subtle errors and infelicities. These would be costly to detect and correct at later stages of the design lifecycle.

The specification language of PVS is built on high-order logic; *i.e.*, functions can take functions as arguments and return them as values, and qualification can be applied to function variables. There is a rich set of built-in types and type-constructs, as well as a powerful notion of subtype. Specifications can be constructed by using definitions, axioms, or a mixture of the two. Specifications are logically organized into parameterized *theories* and *datatypes*. Theories are linked by *import* and *export* lists. Specifications for many foundational and standard theories are preloaded into PVS as *prelude* theories that are always available and do not need to be explicitly imported. Details on PVS language may be found in *PVS language reference* [34].

PVS has a powerful theorem prover, or proof checker [35]. It is both interactive and highly mechanized: the user chooses each step that is to be applied and PVS performs it, displays the result, and then waits for the next command. PVS is, therefore, directly controlled by the user. On the other hand, PVS can invoke powerful decision procedures for arithmetic, automatic rewriting, and induction. The automation underlying PVS serves to ensure that the process of verification yields human insights that can be easily communicated to other human, and encapsulated for future verification. PVS therefore pays a lot of attention to simplifying the process of developing, debugging, maintaining, and presenting proofs.

PVS is implemented in Common Lisp with ancillary functions provided in C, Tcl/Tk, and LaTeX [36], but it is not necessary to know Common Lisp to effectively use the system. The GNU Emacs [37] or Xemacs display editor provides the interface to PVS.

5.2 Specification for the $\mathcal{EH}$ system

A PVS specification consists of a collection of *theories*. Each theory consists of a *signature* for the type names and constants introduced in the theory, and the axioms, definitions, and theorems associated with the signature. The $\mathcal{EH}$ system is specified as a theory `EH`; the axioms and rules (see Section 3.2) of the $\mathcal{EH}$ system are specified as axioms in the theory `EH`. The type `CONTROL_LAW` is defined *uninterpreted*, as shown below.

```
CONTROL_LAW : TYPE
```

The notion of system can be captured by the following PVS declarations.

```
P, P1, Q, Q1, W, W1, W2, R, B, Inv : VAR bool
C, C1, C2 : VAR CONTROL_LAW
skip, A :CONTROL_LAW

Assertion : [CONTROL_LAW, bool, bool, bool -> bool]
```

The above function `Assertion` signifies the new notation $P \{ C \mid W \} Q$ we introduced in Section 3.2. The axioms and rules of $\mathcal{EH}$ system are defined as axioms of theory `EH` as follows.

```
Skip: AXIOM Assertion(skip, P, FALSE, P)
Composition: AXIOM Assertion(C1, P, W1, R)
                AND Assertion(C2, R, W2, Q)
            IMPLIES Assertion(SequenceControl(C1, C2), P,
                              SequenceCondition(W1, W2), Q)
Conditional: AXIOM Assertion(C1, (P AND B), W1, Q)
                AND Assertion(C2, (P AND (NOT B)), W2, Q)
            IMPLIES Assertion(ConditionControl(B, C1, C2),
                              P, (W1 OR W2), Q)
```

```
Iteration: AXIOM Assertion(C, (Inv AND B), W, Inv)
         IMPLIES Assertion(WhileControl(B, C), Inv, W,
                                    (Inv AND (NOT B)))
Consequence: AXIOM ((P IMPLIES P1)
                AND (W1 IMPLIES W)
                AND (Q1 IMPLIES Q)
                AND Assertion(C, P1, W1, Q1))
           IMPLIES Assertion(C, P, W, Q)
```

The complete specification for the $\mathcal{EH}$ system appears in Appendix B.1.

5.3 Specification for the Water Level Monitoring System

As stated in the previously, we obtain four desirable control laws C_1, C_3, C_4, and C_6 for the water level monitoring system example as below:

$$C_1 : \langle w = h,\ w \in [76, h],\ w = 76 \rangle$$

$$C_3 : \langle w = 76,\ w \in [68, 76],\ w = 68 \rangle$$

$$C_4 : \langle w = 68,\ w \in [68, 76],\ w = 76 \rangle$$

$$C_6 : \langle w = 0,\ w \in [0, 68],\ w = 68 \rangle$$

We specify the water level monitoring system as theory `WaterContainer`, and the above four control laws as the axioms in the theory. The control requirements `CR1` and `CR2` (see Section 4.1) are specified as theorems in the theory `WaterContainer` to be verified. The details are shown in Appendix B.2.

The system behavior is specified to `WaterContainer` as uninterpreted type `CONTROL_LAW` shown below.

```
WaterContainer : CONTROL_LAW =
  SequenceControl(ConditionControl(w = h, C1, C6),
  WhileControl(TRUE, ConditionControl((w=68), C4, C3)))
```

where `SequenceControl, ConditionControl` and `WhileControl` are sequential, conditional and iteration control laws defined in the theory `EH` according to the composition, conditional and iteration rules of $\mathcal{EH}$ system, respectively.

The four control laws C_1, C_3, C_4, and C_6 have the following PVS specifications.

```
ControlLaw1:AXIOM Assertion(C1,(w=h ),(w>=76 AND w<=h ),(w=76))
ControlLaw3:AXIOM Assertion(C3,(w=76),(w>=68 AND w<=76),(w=68))
ControlLaw4:AXIOM Assertion(C4,(w=68),(w>=68 AND w<=76),(w=76))
ControlLaw6:AXIOM Assertion(C6,(w=0 ),(w>=0  AND w<=68),(w=68))
```

As for axiom `ControlLaw1`, it signifies control law C_1 with the entering point satisfying proposition $(w = h)$, the changing range satisfying proposition $(w >= 76 \wedge w <= h)$, and the leaving point satisfying proposition $(w = 76)$. Then, we define two theorem `CR1` and `CR2`:

```
CR1: THEOREM Assertion(ConditionControl(w=h,C1,C6),(w=0 OR w=h),
        ((w>=76 AND w<=h) OR (w>=0 AND w<=68)), (w=68 OR w=76))

CR2: THEOREM Assertion(WhileControl(TRUE, ConditionControl
        ((w=68), C4, C3)),(w=68 OR w=76),
        ((w>=68 AND w<=76) OR (w>=68 AND w<=76)), FALSE)
```

These signify the two control requirements CR_1 and CR_2 (see Section 4.1), which are to be verified.

5.4 Verification Using PVS

The next step is to verify the correctness of specifications. Although a broad understanding of the specification correctness for our example can be obtained fairly readily, detailed proof of its theorems requires attention to a mass of details and an astonishingly intricate argument. In the course of the proof, PVS builds up a tree of sequents where each sequent is a subgoal generated from its parent sequent by a PVS proof command. At any point in the proof attempt, the control is at a leaf sequent of such a proof tree. The proof is completed when there are no remaining unproved leaf sequents in the proof tree. A complete proof of control requirement CR_1 using PVS is shown below.

```
(CR1 "" (LEMMA "ControlLaw1")
 (("" (LEMMA "ControlLaw6")
 (("" (LEMMA "Consequence")
 (("" (LEMMA "Conditional")
 (("" (GRIND)
   (("1" (CASE "w=0" "w=h")
     (("1" (USE "height") (("1" (ASSERT) NIL)))
      ("2" (USE "height") (("2" (ASSERT) NIL)))
      ("3" (ASSERT)
       (("3" (CASE "w=h")
         (("1" (ASSERT) (("1" (USE "False") NIL)))
          ("2" (ASSERT) (("2" (USE "False") NIL)))))))))
   ("2" (CASE "w=h")
    (("1" (USE "height") (("1" (ASSERT) NIL)))
     ("2" (CASE "w=0")
      (("1" (USE "height") (("1" (ASSERT) NIL)))
       ("2" (GRIND) (("2" (USE "False") NIL)))))))))))))))))
```

This is the form in which PVS proofs are stored for later replay. In each proof step PVS applies a proof command. For examples, the first step of the above proof is the proof command `(LEMMA "ControlLaw1")`, which

means use the lemma (axiom) `ControlLaw1` as an *antecedent* formula of the current PVS sequent. The complete proofs and proof trees of requirements CR_1 and CR_2 are shown in Appendix B.3 and B.4.

5.5 Summary

This chapter is concerned with the formal verification of correctness of the specification obtained in the previous chapters. This goal is achieved by means of PVS specification and verification system. Using PVS, we have mechanically verified the correctness of specifications.

Chapter 6

Concluding Remarks

The design, specification and verification of hybrid systems is becoming increasingly important both in fields of computer science and control engineering. This trend is being driven by two factors: the nature of hybrid systems, which reflects the characteristics of most control systems nowadays, and the wide use of computers.

This thesis has shown that the design of hybrid systems is simplified using the state-space based approach (see Chapter 2), and that the reliability is increased by specification using extended Hoare logic (see Chapter 3 and Chapter 4) and verification using PVS (see Chapter 5).

This chapter begins in Section 6.1 with a brief description of the contributions made in the course of demonstrating the thesis. This work has also uncovered many avenues of further inquiry; these are presented in Section 6.2.

6.1 Contributions of This Thesis

In this thesis, we propose a state-space based approach to the design of control laws and synthesis of decision-maker. We apply the principle of stepwise refinement to the design and analysis of hybrid control systems and propose a state-space based method for the design of digital controllers

and synthesis of the decision-maker through decomposition of the corresponding control laws. The main design procedure includes problem description, determination of state-space, partition of state-space and control refinement. The central idea of this approach is illustrated by applying it to a typical example of hybrid systems—a water level monitoring system. Additionally, this method has been used to solve a more complex problem—control of the inverted pendulum [38, 39]. Since the inverted pendulum control system is a non-linear system, it is difficult to get its control law by solving the corresponding equations. Several approaches, such as adaptive control and optimal control *etc.*, have been adopted to deal with this problem. The state-space based method gives a different approach to solving this problem.

To explore the theoretical foundation of the state-space based approach, we analyze the generic characteristics of control laws. We conclude that the important characteristics of control laws are their entering points, leaving points and changing ranges. These three aspects can be specified by making general assertions about the values that the relevant state variables constituting state-space will take before, after and during the action of control laws. To describe control laws in this sense, we introduce a new notation $P \{ C \mid W \} Q$, where P, R, W, and C are precondition, postcondition, duration-condition and control law, respectively. Furthermore, in order to specify and verify the design procedures, the classical Hoare logic system has been extended as an axiomatic basis of the method and a compositional proof system has been formulated. Then, we show that the system is sound in specifying and verifying hybrid control systems by constructing an operational model, Evolution Machine (EM), of the extended Hoare ($\mathcal{EH}$) logic system.

Following the proposed design procedure, we achieve the specification for the example water level monitoring system using the extended Hoare logic. The correctness of specifications has been mechanically verified by using the PVS specification and verification system. As a result, the correctness of design procedure can be verified to some extent, and the reliability of hybrid control systems is increased.

6.2 Future Work

The spatial and timing properties are two important aspects of a hybrid system. In this thesis, we apply the principle of stepwise refinement and the decomposition of control laws to the design and analysis of hybrid control systems, and pay more attention to the spatial properties of hybrid systems than to their timing properties; that is, we pay more attention to the systematic acquisition of the correct digital controllers and the corresponding decision-maker from the problem description using the state-space based approach. However, it is an important topic to analyze the timing properties of hybrid systems. Many methods, such as the Interval Temporal Logic (ITL) [40], Duration Calculus community [41, 42, 43, 44, 45], Hybrid Temporal Logic [20], VDM++ [46, 47], Metric Temporal Logic [48, 49], Hybrid CSP [50, 51] and Clocked Transition System (CTS) [52, 53] *etc.*, for the specification and verification of hybrid systems and real-time systems discuss this problem in different ways. In our method, the timing properties of the hybrid system are expressed implicitly. The next step is to express the timing properties more directly.

To this end, the extended Hoare logic notation $P \{ C \mid W \} Q$ should be modified to real-time; the predicates P, W, and Q should be extended to temporal-like logic. Consider, for instance, the formula

$$(time = 3) \{ \textbf{delay } 4 \mid W \} (time = 7).$$

In the precondition the variable *time* specifies the starting time of the program, whereas in the postcondition *time* denotes the termination time. Furthermore, we can use logical variables to relate pre-, duration- and postcondition. For instance, with the variable t, the specification

$$(time = t) \{ C \mid W \} (t + 3 < time < t + 7)$$

or

$$P \{ C \mid 3 < duration < 7 \} Q$$

expresses that if C terminates then it takes between 3 and 7 time units.

Another interesting subject for further research is the *chop* (;) operator in the compositional rule of the proof system for the extended Hoare logic. Because *chop* is a temporal operator, the temporal logic is implicit in the compositional rule. The coexistence of the explicit temporal operator *chop* with implicit expression style of the timing properties in $\mathcal{EH}$ system may cause some theoretical problems and requires further research. A putative way is to introduce the temporal-like logic into the $\mathcal{EH}$ system.

Appendix A

Proof for Soundness of the Proof System $\mathcal{EH}$

Theorem 3.4.1 (Soundness) $\vdash P\{C|W\}Q$ implies $\models P\{C|W\}Q$.

Proof. Due to the form of the proof system $\mathcal{EH}$, it is sufficient to prove that all axioms and proof rules of $\mathcal{EH}$ are sound. Then the result follows by the induction on the length of proofs. We consider all axioms and proof rules in turn.

(i) SKIP
By the definition 3.4.2

$$\begin{aligned} & \mathcal{O}[\![P\{\mathsf{skip}|\mathsf{false}\}P]\!] \\ \equiv\ & P \Rightarrow P \\ \equiv\ & \mathsf{true} \end{aligned}$$

So, by the definition 3.4.3

$$\models P\{\mathsf{skip}|\mathsf{false}\}P$$

(ii) ATOMIC CONTROLLER
For atomic controllers, the semantics of them is assumed to be given soundly. Thus

$$\models P\{A|W\}Q$$

(iii) COMPOSITION

Suppose now that
$\models P\{C_1|W_1\}R$
and
$\models R\{C_2|W_2\}Q$
We prove that
$\models P\{C_1; C_2|W_1; W_2\}Q$

By the definition 3.4.1

$$
\begin{aligned}
& \mathcal{O}[\![P\{C_1; C_2|W_1; W_2\}Q]\!] \\
\equiv\ & \forall\ s, s'.\ (\mathcal{I}[\![P]\!](s) \wedge s' = EM(C_1; C_2)(s)) \Rightarrow \mathcal{I}[\![Q]\!](s') \\
& \wedge \forall\ t \in \textsc{Time}.\ t_s < t < t_{s'} \Rightarrow \mathcal{I}[\![W_1; W_2]\!](s_t) \\
\equiv\ & \forall\ s, s'.\ (\mathcal{I}[\![P]\!](s) \wedge s' = EM(C_2) \circ EM(C_1)(s)) \Rightarrow \mathcal{I}[\![Q]\!](s') \\
& \wedge \forall\ t \in \textsc{Time}.\ t_s < t < t_{s'} \Rightarrow \mathcal{I}[\![W_1; W_2]\!](s_t) \\
\equiv\ & \forall\ s, s'.\ \exists s''.\ (\mathcal{I}[\![P]\!](s) \wedge s'' = EM(C_1)(s) \wedge s' = EM(C_2)(s'')) \\
& \qquad \Rightarrow \mathcal{I}[\![Q]\!](s') \\
& \wedge \forall\ t \in \textsc{Time}.\ t_s < t < t_{s'} \\
& \qquad \Rightarrow (t < t_{s''} \Rightarrow \mathcal{I}[\![W_1]\!](s_t) \wedge t > t_{s''} \Rightarrow \mathcal{I}[\![W_2]\!](s_t)
\end{aligned}
$$

Since $\models P\{C_1|W_1\}R$, we have
$\mathcal{I}[\![P]\!](s) \wedge s'' = EM(C_1)(s) \Rightarrow \mathcal{I}[\![R]\!](s'')$
$\wedge(t_s < t < t_{s''} \Rightarrow \mathcal{I}[\![W_1]\!](s_t)$

Since $\models R\{C_2|W_2\}Q$, we have
$\mathcal{I}[\![R]\!](s'') \wedge s' = EM(C_2)(s'') \Rightarrow \mathcal{I}[\![R]\!](s')$
$\wedge(t_{s''} < t < t_{s'} \Rightarrow \mathcal{I}[\![W_2]\!](s_t)$

So, $\mathcal{O}[\![P\{C_1; C_2|W_1; W_2\}Q]\!] \equiv \mathsf{true}$

Hence by the definition 3.4.3
$\models P\{C_1; C_2|W_1; W_2\}Q$

(iv) CONDITIONAL

Suppose that
$\models P \wedge B\{C_1|W_1\}Q$
and
$\models P \wedge \neg B\{C_2|W_2\}Q$
We prove that
$\models P\{(\textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2)|W_1 \vee W_2\}Q$

By the definition 3.4.1
$\mathcal{O}[\![P\{(\textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2)|W_1 \vee W_2\}Q]\!]$

$$\begin{aligned}
\equiv\ & \forall\ s, s'.\ (\mathcal{I}[\![P]\!](s) \wedge s' = EM((\textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2) \\
& \qquad |W_1 \vee W_2)(s)) \Rightarrow \mathcal{I}[\![Q]\!](s') \\
& \wedge \forall\ t \in \textsc{Time}.\ t_s < t < t_{s'} \Rightarrow \mathcal{I}[\![W_1 \vee W_2]\!](s_t)
\end{aligned}$$

$$\begin{aligned}
\equiv\ & \forall\ s, s'.\ (\mathcal{I}[\![P]\!](s) \wedge s' = \mathsf{cond}(\mathcal{I}[\![B]\!](s), EM(C_1|W_1), \\
& \qquad EM(C_2|W_2))(s) \Rightarrow \mathcal{I}[\![Q]\!](s') \\
& \wedge \forall\ t \in \textsc{Time}.\ t_s < t < t_{s'} \Rightarrow \mathcal{I}[\![W_1 \vee W_2]\!](s_t)
\end{aligned}$$

$$\begin{aligned}
\equiv\ & \forall\ s, s'.\ (\mathcal{I}[\![P]\!](s) \wedge ((\mathcal{I}[\![B]\!](s) \wedge s' = EM(C_1|W_1)(s)) \vee (\mathcal{I}[\![\neg B]\!](s) \\
& \wedge s' = EM(C_2|W_2)(s)))) \Rightarrow \mathcal{I}[\![Q]\!](s') \\
& \wedge \forall\ t \in \textsc{Time}.\ t_s < t < t_{s'} \Rightarrow \mathcal{I}[\![W_1 \vee W_2]\!](s_t)
\end{aligned}$$

$$\begin{aligned}
\equiv\ & \forall\ s, s'.\ (\mathcal{I}[\![P]\!](s) \wedge \mathcal{I}[\![B]\!](s) \wedge s' = EM(C_1|W_1)(s)) \Rightarrow \mathcal{I}[\![Q]\!](s') \\
& \wedge (\mathcal{I}[\![P]\!](s) \wedge \mathcal{I}[\![\neg B]\!](s) \wedge s' = EM(C_2|W_2)(s)) \Rightarrow \mathcal{I}[\![Q]\!](s') \\
& \wedge \forall\ t \in \textsc{Time}.\ t_s < t < t_{s'} \Rightarrow \mathcal{I}[\![W_1 \vee W_2]\!](s_t)
\end{aligned}$$

Since $\models P \wedge B\{C_1|W_1\}Q$, we have
$(\mathcal{I}[\![P]\!](s) \wedge \mathcal{I}[\![B]\!](s) \wedge s' = EM(C_1|W_1)(s)) \Rightarrow \mathcal{I}[\![Q]\!](s'))$
$\wedge \forall\ t \in \textsc{Time}.\ t_s < t < t_{s'} \Rightarrow \mathcal{I}[\![W_1]\!](s_t)$

Since $\models P \wedge B\{C_1|W_1\}Q$, we have
$(\mathcal{I}[\![P]\!](s) \wedge \mathcal{I}[\![\neg B]\!](s) \wedge s' = EM(C_2|W_2)(s)) \Rightarrow \mathcal{I}[\![Q]\!](s'))$
$\wedge \forall\, t \in \textsc{Time}.\ t_s < t < t_{s'} \Rightarrow \mathcal{I}[\![W_2]\!](s_t)$

So, $\quad \mathcal{O}[\![P\{(\textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2)|W_1 \vee W_2\}Q]\!] \ \equiv\ \mathsf{true}$

Hence by the definition 3.4.3
$\models P\{(\textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2)|W_1 \vee W_2\}Q$

The proofs of rule (v) and (vi) are similar to the above, and thus are omitted here.

Appendix B

PVS Specifications and Proofs

B.1 Theory EH

The source code of the specification for $\mathcal{EH}$ system, Theory EH, is as follows:

```
EH : THEORY
  BEGIN

  CONTROL_LAW: TYPE

  P, P1, Q, Q1, W, W1, W2, R, B, Inv: VAR bool
  C, C1, C2: VAR CONTROL_LAW
  skip, A:CONTROL_LAW
  P_A, W_A, Q_A:bool

  Assertion: [CONTROL_LAW, bool, bool, bool -> bool]
  SequenceControl: [CONTROL_LAW, CONTROL_LAW -> CONTROL_LAW]
  SequenceCondition: [bool, bool -> bool]
  WhileControl: [bool, CONTROL_LAW -> CONTROL_LAW]
  ConditionControl: [bool, CONTROL_LAW, CONTROL_LAW
                -> CONTROL_LAW]
```

```
    False: AXIOM Assertion(C, FALSE, W, Q)
    True: AXIOM Assertion(C, TRUE, TRUE, TRUE)
    Skip: AXIOM Assertion(skip, P, FALSE, P)
    Atomic: AXIOM Assertion(A, P_A, W_A, Q_A)
    Composition: AXIOM Assertion(C1, P, W1, R)
                    AND Assertion(C2, R, W2, Q)
                IMPLIES Assertion(SequenceControl(C1, C2), P,
                                  SequenceCondition(W1, W2), Q)
    Conditional: AXIOM Assertion(C1, (P AND B), W1, Q)
                    AND Assertion(C2, (P AND (NOT B)), W2, Q)
                IMPLIES Assertion(ConditionControl(B, C1, C2),
                                  P, (W1 OR W2), Q)
    Iteration: AXIOM Assertion(C, (Inv AND B), W, Inv)
              IMPLIES Assertion(WhileControl(B, C), Inv, W,
                                          (Inv AND (NOT B)))
    Consequence: AXIOM ((P IMPLIES P1)
                    AND (W1 IMPLIES W)
                    AND (Q1 IMPLIES Q)
                    AND Assertion(C, P1, W1, Q1))
                IMPLIES Assertion(C, P, W, Q)

END EH
```

The LaTeX output of the Theory `EH` is shown in Figure B.1.

B.2 Theory `WaterContainer`

The source code of the specification for the example water level monitoring system, Theory `WaterContainer`, is as follows:

```
WaterContainer : THEORY

BEGIN
```

```
IMPORTING EH

C1, C3, C4, C6: CONTROL_LAW
w: nat
h: nat

WaterContainer: CONTROL_LAW =
           SequenceControl(ConditionControl(w = h, C1, C6),
           WhileControl(TRUE, ConditionControl((w=68), C4, C3)))
ControlLaw1: AXIOM Assertion(C1,(w=h ),(w>=76 AND w<=h ),(w=76))
ControlLaw3: AXIOM Assertion(C3,(w=76),(w>=68 AND w<=76),(w=68))
ControlLaw4: AXIOM Assertion(C4,(w=68),(w>=68 AND w<=76),(w=76))
ControlLaw6: AXIOM Assertion(C6,(w=0 ),(w>=0  AND w<=68),(w=68))
height: AXIOM h > 76

CR1: THEOREM Assertion(ConditionControl(w=h,C1,C6),(w=0 OR w=h),
         ((w>=76 AND w<=h) OR (w>=0 AND w<=68)),(w=68 OR w=76))

CR2: THEOREM Assertion(WhileControl(TRUE,
         ConditionControl((w=68), C4, C3)), (w=68 OR w=76),
         ((w>=68 AND w<=76) OR (w>=68 AND w<=76)), FALSE)

END WaterContainer
```

The LaTeX output of the Theory `EH` is shown in Figure B.2.

B.3 Proofs for Control Requirements

B.3.1 Proof for CR_1

```
(CR1 "" (LEMMA "ControlLaw1")
 (("" (LEMMA "ControlLaw6")
```

```
(("" (LEMMA "Consequence")
(("" (LEMMA "Conditional")
(("" (GRIND)
  (("1" (CASE "w=0" "w=h")
    (("1" (USE "height") (("1" (ASSERT) NIL)))
     ("2" (USE "height") (("2" (ASSERT) NIL)))
     ("3" (ASSERT)
      (("3" (CASE "w=h")
        (("1" (ASSERT) (("1" (USE "False") NIL)))
         ("2" (ASSERT) (("2" (USE "False") NIL)))))))))
   ("2" (CASE "w=h")
    (("1" (USE "height") (("1" (ASSERT) NIL)))
     ("2" (CASE "w=0")
      (("1" (USE "height") (("1" (ASSERT) NIL)))
       ("2" (GRIND)
              (("2" (USE "False") NIL)))))))))))))))
```

B.3.2 Proof for CR_2

```
(CR2 "" (LEMMA "ControlLaw4")
 (("" (LEMMA "ControlLaw3")
 (("" (LEMMA "Consequence")
 (("" (LEMMA "Iteration")
 (("" (GRIND)
 (("" (DELETE 2)
 (("" (LEMMA "Consequence")
 (("" (LEMMA "Conditional")
 (("" (GRIND)
   (("1" (LEMMA "Consequence")
     (("1" (GRIND)
       (("1" (CASE "w=68" "w=76")
         (("1" (ASSERT) NIL) ("2" (ASSERT) NIL)
```

```
        ("3" (ASSERT)
         (("3" (CASE "w=76")
           (("1" (ASSERT) (("1" (USE "True") NIL)))
            ("2" (ASSERT) NIL))))))))))))
("2" (CASE "w=76")
 (("1" (ASSERT) NIL)
  ("2" (CASE "w=68")
   (("1" (ASSERT) (("1" (USE "True") NIL)))
    ("2" (ASSERT) NIL))))))))))))))))))))
```

B.4 Proof Trees

B.4.1 Proof Tree for CR_1

Graphical display of the proof Tree for CR_1 is shown in Figure B.3.

B.4.2 Proof Tree of CR_2

Graphical display of the proof Tree for CR_1 is shown in Figure B.4.

EH : THEORY
BEGIN

CONTROL_LAW : TYPE

$P, P_1, Q, Q_1, W, W_1, W_2, R, B, \text{Inv}$: VAR bool

C, C_1, C_2 : VAR CONTROL_LAW

skip, A : CONTROL_LAW

Assertion : [CONTROL_LAW, bool, bool, bool → bool]

SequenceControl : [CONTROL_LAW, CONTROL_LAW → CONTROL_LAW]

SequenceCondition : [bool, bool → bool]

WhileControl : [bool, CONTROL_LAW → CONTROL_LAW]

ConditionControl : [bool, CONTROL_LAW, CONTROL_LAW → CONTROL_LAW]

FALSE : AXIOM Assertion(C, FALSE, W, Q)

TRUE : AXIOM Assertion(C, TRUE, TRUE, TRUE)

Skip : AXIOM Assertion(skip, P, FALSE, P)

Composition : AXIOM
Assertion(C_1, P, W_1, R) ∧ Assertion(C_2, R, W_2, Q) ⊃
Assertion(SequenceControl(C_1, C_2), P, SequenceCondition(W_1, W_2), Q)

Conditional : AXIOM
Assertion($C_1, (P \wedge B), W_1, Q$) ∧ Assertion($C_2, (P \wedge (\neg B)), W_2, Q$) ⊃
Assertion(ConditionControl(B, C_1, C_2), P, $(W_1 \vee W_2)$, Q)

Iteration : AXIOM
Assertion(C, (Inv ∧ B), W, Inv) ⊃
Assertion(WhileControl(B, C), Inv, W, (Inv ∧ $(\neg B)$))

Consequence : AXIOM
$((P \supset P_1) \wedge$
$(W_1 \supset W) \wedge$
$(Q_1 \supset Q) \wedge$ Assertion(C, P_1, W_1, Q_1)) ⊃
Assertion(C, P, W, Q)

END EH

Figure B.1: LaTeX-Printed Version of the Theory EH

```
WaterContainer : THEORY
  BEGIN

    IMPORTING EH

    C_1, C_3, C_4, C_6 : CONTROL_LAW

    w, h : nat

    WaterContainer : CONTROL_LAW =
      SequenceControl(ConditionControl(w = h, C_1, C_6),
                      WhileControl(TRUE, ConditionControl((w = 68), C_4, C_3)))

    ControlLaw1 : AXIOM
      Assertion(C_1, (w = h), (w ≥ 76 ∧ w ≤ h), (w = 76))

    ControlLaw3 : AXIOM
      Assertion(C_3, (w = 76), (w ≥ 68 ∧ w ≤ 76), (w = 68))

    ControlLaw4 : AXIOM
      Assertion(C_4, (w = 68), (w ≥ 68 ∧ w ≤ 76), (w = 76))

    ControlLaw6 : AXIOM
      Assertion(C_6, (w = 0), (w ≥ 0 ∧ w ≤ 68), (w = 68))

    heigh : AXIOM h > 76

    CR1 : THEOREM
      Assertion(ConditionControl(w = h, C_1, C_6), (w = 0 ∨ w = h),
                ((w ≥ 76 ∧ w ≤ h)∨
                   (w ≥ 0 ∧ w ≤ 68)),
                (w = 68 ∨ w = 76))

    CR2 : THEOREM
      Assertion(WhileControl(TRUE, ConditionControl((w = 68), C_4, C_3)),
                (w = 68 ∨ w = 76),
                ((w ≥ 68 ∧ w ≤ 76)∨
                   (w ≥ 68 ∧ w ≤ 76)),
                FALSE)

    END WaterContainer
```

Figure B.2: LaTeX-Printed Version of the Theory WaterContainer

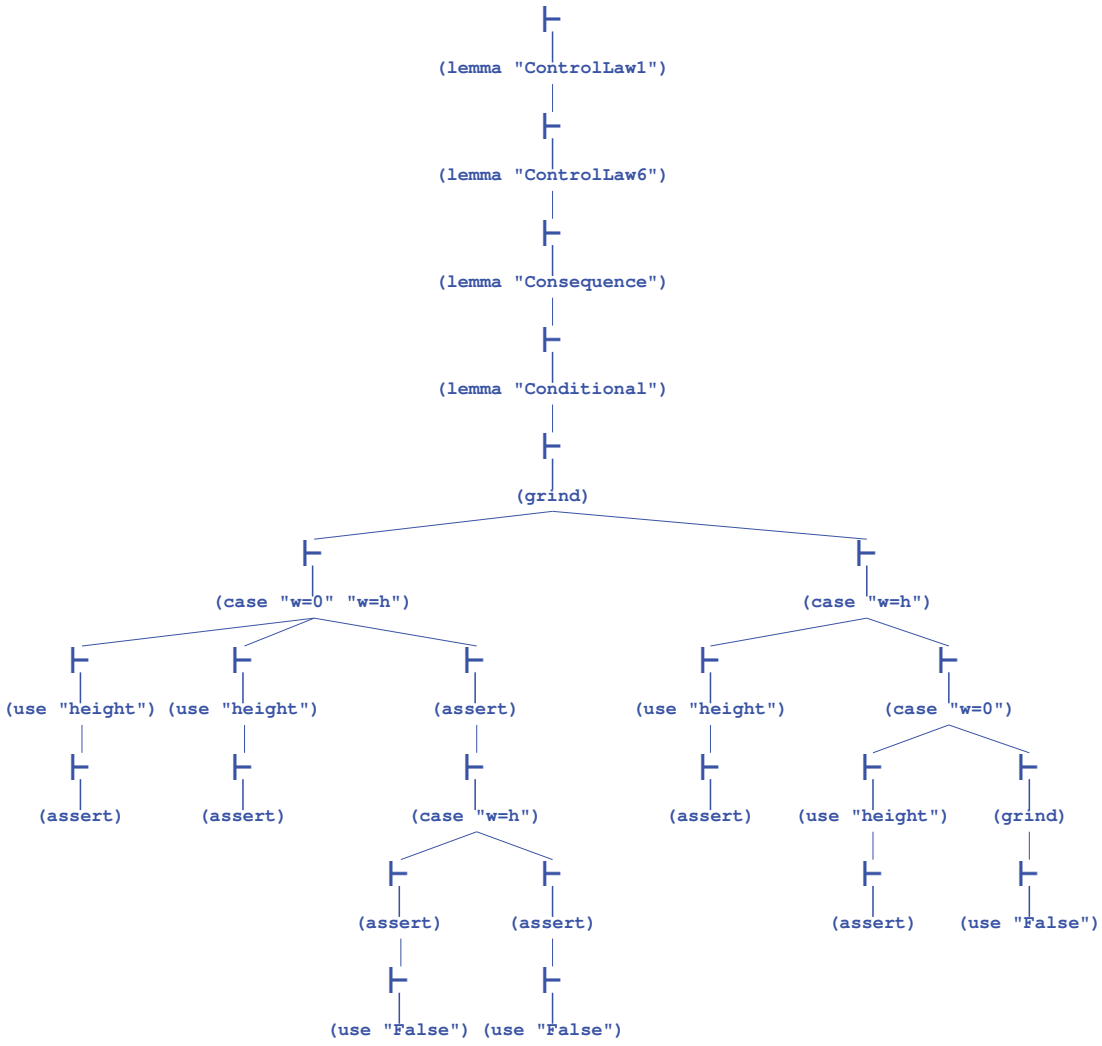

Figure B.3: Graphical Display of the Proof Tree for CR_1

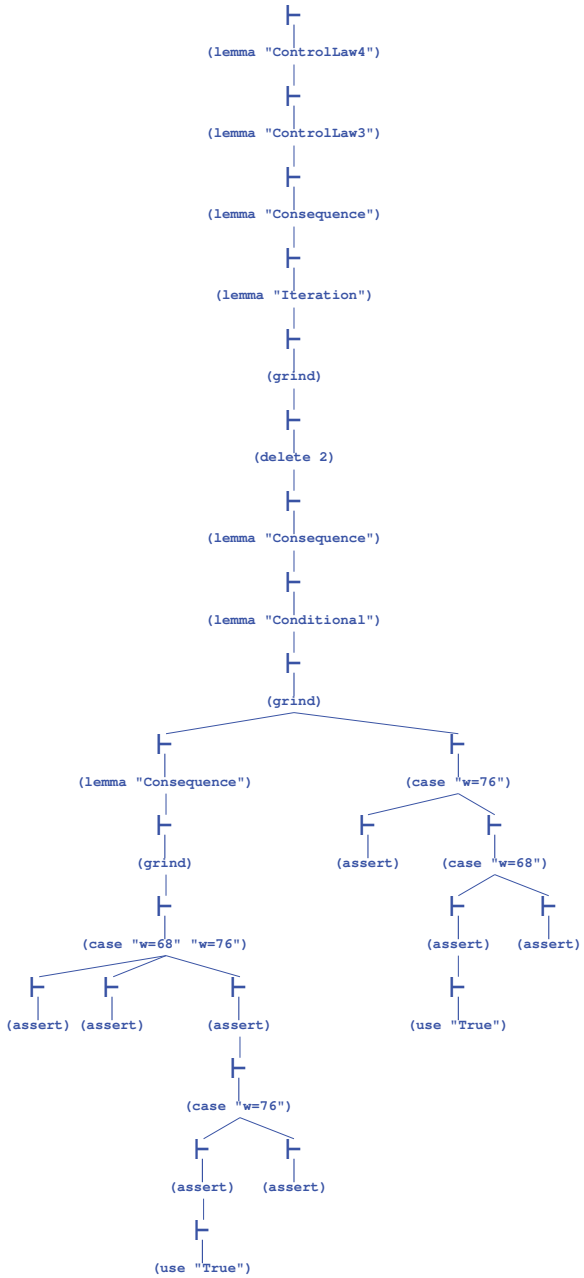

Figure B.4: Graphical Display of the Proof Tree for CR_2

Bibliography

[1] A. Nerode and W. Kohn. Models for hybrid systems: Automata, topologies, controllability and observability. In Grossman et al. [2], pages 317–356.

[2] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors. *Hybrid Systems*. Volume 736 of Grossman et al. [2], 1993.

[3] P. J. G. Ramadge and W. M. Wonham. The control of discrete event dynamical systems. *Proceedings of IEEE*, 77(1):81–98, 1989.

[4] D. Bjørner. A formal model of robots: Geometry and kinematics. Technical Report No. 6, International Institute for Software Technology, The United Nations University (UNU/IIST), P.O. Box 3058, Macau, March 1993.

[5] J. E. Hopcroft. The impact of robotics on computer science. *Communications of the ACM*, 29(6):486–498, June 1986.

[6] S. Sastry, G. Meyer, C. Tomlin, J. Lygeros, D. Godbole, and G. Pappas. Hybrid systems in air traffic control. In *IEEE Control and Decision Conference*, pages 1478–1483, 1995.

[7] R. F. Stengel. Intelligent flight control systems. In *IMA Conference on Aerospace Vehicle Dynamics*, September 1992.

[8] Hans Langmaack, Willem-Paul de Roever, and Jan Vytopil, editors. *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant*

Systems (FTRTFT '94), volume 863 of *Lecture Notes in Computer Science*, Lübeck, Germany, September 1994. Working Group Provably Correct Systems—ProCoS, Springer-Verlag.

[9] Orna Grumberg, editor. *Proceedings of 9th International Conference on Computer Aided Verification (CAV '97)*, volume 1254 of *Lecture Notes in Computer Science*, Haifa, Israel, June 1997. Springer-Verlag.

[10] Roger Shaw, editor. *Safety and Reliability of Software Based Systems: Twelfth Annual CSR Workshop*, Bruges, September 1995. Springer-Verlag London Limited.

[11] *Proceedings of 10th International Conference on Computer Aided Verification (CAV '98)*, Lecture Notes in Computer Science. Springer-Verlag, June 1998.

[12] *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT '98)*, Lecture Notes in Computer Science, Lyngby, Denmark, September 1998. Springer-Verlag.

[13] P. J. Antsaklis, J. A. Stiver, and M. Lemmon. Hybrid system modeling and autonomous control systems. In Grossman et al. [2], pages 366–392.

[14] Chen Zongji, Wang Ji, Yu Xinyao, and Zhou Chaochen. An abstraction of hybrid control systems. In *Proceedings of IEEE Singapore International Conference on Intelligent Control and Instrumentation*, pages 1–6. IEEE, Singapore, 1995.

[15] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event systems. *SIAM Journal of Control Optimization*, 25(1):206–230, January 1987.

[16] P. J. Ramadge and W. M. Wonham. Modular feedback logic for discrete event systems. *SIAM Journal of Control Optimization*, 25(5):1202–1218, September 1987.

[17] Yang Zhenyu, Yu Xinyao, and Chen Zongji. Modeling and control of hybrid control systems. Research report, Department of Automatic Control, Beijing University of Aeronautics and Astronautics, Beijing, P. R. China, 1996.

[18] Dang Van Hung and Wang Ji. On the design of hybrid control systems using I/O automaton models. Research Report No. 35, International Institute for Software Technology, The United Nations University (UNU/IIST), P.O. Box 3058, Macau, November 1994.

[19] Lü Jian, Yang Dajun, and Yang Zhenyu. A state-space based approach to the design of hybrid systems and its logic basis. *Chinese Journal of Advanced Software Research*, 5(4):296–303, 1999.

[20] T. A. Henzinger, Z. Manna, and A. Pnueli. Towards refining temporal specifications into hybrid systems. In Grossman et al. [2], pages 60–76.

[21] J. Hooman. A compositional approach to the design of hybrid systems. In Grossman et al. [2], pages 121–148.

[22] Henny B. Sipma and Zohar Manna. Specification and verification of controlled systems. In Langmaack et al. [8], pages 641–659.

[23] C. A. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 583, October 1969.

[24] C. A. R. Hoare and C. B. Jones. *Essays in Computing Science*. Prentice-Hall International Ltd., 1989.

[25] Krzysztof R. Apt and Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. Graduate Texts in Computer Science. Springer-Verlag New York Inc., second edition, 1997.

[26] M. C. B. Hennessy and G. D. Plotkin. Full abstraction for a simple programming language. In *Proceedings of Mathematical Foundations of Computer Science*, volume 74 of *Lecture Notes in Computer Science*, pages 108–120. Spring-Verlag, New York, 1979.

[27] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN 19, Department of Computer Science, Aarhus University, 1981.

[28] Jan van Leeuwen, editor. *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*. Elsevier Science Publishers and The MIT Press, 1994.

[29] Wu Dinghao and Lü Jian. A state-space based approach to the design of hybrid systems. In *Proceedings of 7th National Conference of Young Computer Scientists (NCYCS '98)*, pages 48–53, Shanghai, China, October 1998. CCF, Shanghai Sci-Tech Press.

[30] Wu Dinghao and Lü Jian. On theoretical basis of a state-space based approach to the design of hybrid systems. *Journal of Nanjing University (Natural Sciences)*, 35(5), 1999.

[31] Wu Dinghao and Lü Jian. A state-space based approach to the specification and verification of hybrid systems and its axiomatic basis. Research report, Institute of Computer Software, Nanjing University, Nanjing, P. R. China, 1999.

[32] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS System Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1998.

[33] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Proceedings of 11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, 1992. Springer-Verlag, New York.

[34] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1998.

[35] N. Shankar, S. Owre, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Prover Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1998.

[36] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Publishing Company, Reading, MA, 2nd edition, 1994.

[37] Richard M. Stallman. *GNU Emacs Manual*. Free Software Foundation, 675 Massachusetts Ave., Cambridge, MA, 13th edition, July 1997.

[38] Yang Zhenyu, Lü Jian, and Chen Zongji. A state-space based approach to acquisition of I/O automaton—a case study for the design of hybrid control systems. In *IEE Proceedings of the Workshop on Discrete Event Systems (WODES96)*, pages 202–207, Edinburgh, U. K., 1996.

[39] Yang Zhenyu, Lü Jian, and Chen Zongji. Control analysis, synthesis and verification in hybrid control systems using I/O automata—a case study. In *IEEE International Symposium on Computer-Aided Control System Design*, Michigan, USA, September 1996.

[40] B. Moszkowski. A temporal logic for multi-level reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.

[41] Zhou Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, May 1991.

[42] Z. Chaochen, A. P. Ravn, and C. A. R. Hoare. An extended duration calculus for hybrid real-time systems. In Grossman et al. [2], pages 36–59.

[43] A. P. Ravn, H. Rischel, and K. M. Hansen. Specifying and verifying requirements of real-time systems. *IEEE Transaction on Software Engineering*, 19(1):41–55, 1993.

[44] Z. Chaochen. Duration calculi: An overview. Research Report No. 10, International Institute for Software Technology, The United Nations University (UNU/IIST), P.O. Box 3058, Macau, June 1993.

[45] Michael R. Hansen and Zhou Chaochen. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 3(1), 1997.

[46] C. B. Jones. *Systematic Software Development Using VDM*. Prentice Hall International (UK), second edition, 1990.

[47] Eugè Dürr, Stephen Goldsack, and Nico Plat. Rigorous development of current object-oriented systems. In *Proceedings of TOOLS94*, 1994.

[48] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[49] J. Hooman. Specification and verification of real-time systems using metric temporal logic. In *ICYCS '91*, pages 300–304. International Academic Publishers, China, 1991.

[50] Jifeng He. From CSP to hybrid systems. In *A Classical Mind*. Prentice-Hall, 1994.

[51] Wang Ji, Yu Xinyao, and Zhou Chaochen. Refinement of digital dynamic systems. Research Report No. 20, International Institute for Software Technology, The United Nations University (UNU/IIST), P.O. Box 3058, Macau, February 1994.

[52] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J. W. de Baker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 226–251. Springer-Verlag, 1992.

[53] Zohar Manna and Amir Pnueli. Clocked transition systems. In *Proceedings of Workshop on Verification and Control of Hybrid Systems*, New Brunswick, NJ, October 1995.

[54] R. Gotzhein, M. Kronenburg, and C. Peper. Specifying and reasoning about generic real-time requirements—a case study. Research Report No. 15/96, University of Kaiserslautern, 1996.

[55] R. Eschbach, T. Dei, and M. Kronenburg. A framework for the analysis of formal description techniques for timed systems. Research Report No. 05/98, University of Kaiserslautern, 1998.

[56] M. Kronenburg, T. Dei, and R. Eschbach. Temporal logics as examples of formal description techniques for timed systems. Research Report No. 07/98, University of Kaiserslautern, 1998.

[57] David Y. W. Park, Jens U. Skakkebk, Mats P. E. Heimdahl, Barbara J. Czerny, and David L. Dill. Checking properties of safety critical specifications using efficient decision procedures. In *FMSP'98: Second Workshop on Formal Methods in Software Practice*, March 1998.

[58] J. Hooman. *Specification and Compositional Verification of Real-Time Systems*, volume 558 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[59] Judy Crow, Sam Owre, John Rushby, Natarajan Shankar, and Mandayam Srivas. A tutorial introduction to PVS. In *Workshop on Industrial-Strength Formal Specification Techniques, WIFT '95*, Boca Raton, Florida, April 1995.

[60] N. Shankar, S. Owre, and J. M. Rushby. *A Tutorial on Specification and Verification Using PVS (Beta Release)*. Computer Science Laboratory, SRI International, Menlo Park, CA 94025, USA, March 1993.

[61] Sam Owre and Natarajan Shankar. Abstract datatypes in PVS. Technical Report SRI-CSL-93-9R, Computer Science Laboratory, SRI International, Menlo Park, CA, December 1993.

[62] John Rushby and David W. J. Stringer-Calvert. A less elementary tutorial for the PVS specification and verification system. Technical

Report SRI-CSL-95-10, Computer Science Laboratory, SRI International, Menlo Park, CA, June 1995. Revised, July 1996. Available, with specification files, at
`http://www.csl.sri.com/csl-95-10.html`.

[63] Radu Grosu, Thomas Stauner, and Manfred Broy. A modular visual model for hybrid systems. In *Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT '98)*. Springer-Verlag, 1998.

[64] Amir Pnueli. Development of hybrid systems. In Langmaack et al. [8], pages 77–85.

[65] Xinyao Yu, Ji Wang, Chaochen Zhou, and Paritosh K. Pandya. Formal design of hybrid systems. In Langmaack et al. [8], pages 738–755.

[66] S. Nadjm-Tehrani and J.-E. Stromberg. From physical modeling to compositional models of hybrid systems. In Langmaack et al. [8], pages 583–604.

[67] S. Bradley, W. D. Henderson, D. Kendall, and A. P. Robson. Designing and implementing correct real-time systems. In Langmaack et al. [8], pages 228–246.

[68] Martin Fränzle. Synthesizing controllers from duration calculus. In Jonsson and Parrow [69], pages 168–187.

[69] Bengt Jonsson and Joachim Parrow, editors. *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT '96)*, volume 1135 of *Lecture Notes in Computer Science*. Springer-Verlag, New York, 1996.

[70] Arjun Kapur, Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Proving safety properties of hybrid systems. In Langmaack et al. [8], pages 431–454.

[71] N. Shankar. Verification of real-time systems using PVS. In Costas Courcoubetis, editor, *Proceedings of Computer Aided Verification, CAV '93*, volume 697 of *Lecture Notes in Computer Science*, pages

280–291, Elounda, Greece, June/July 1993. Springer-Verlag, New York.

[72] S. A. Marshall. *Introduction to Control Theory*. The Macmillan Press Ltd., 1978.

[73] P. K. Pandya, Wang Hanpin, and Xu Qiwen. Towards a theory of sequential hybrid programs. Research Report No. 125, International Institute for Software Technology, The United Nations University (UNU/IIST), P.O. Box 3058, Macau, October 1997.

[74] Gavin Lowe. Formal development of aircraft control software: A case study in the specification, design and scheduling of a real-time system. Research Report No. PRG-TR-15-94, Programming Research Group, Oxford University Computing Laboratory, October 1994.

[75] Zhou Chaochen, Dang Van Hung, and Li Xiaoshan. A duration calculus with infinite intervals. Research Report No. 40, International Institute for Software Technology, The United Nations University (UNU/IIST), P.O. Box 3058, Macau, February 1995.

[76] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTech: A model checker for hybrid systems. In Grumberg [9].

[77] N. Shankar and S. Owre. The formal semantics of PVS. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, August 1997.