# A FOUNDATION FOR THE DESIGN AND ANALYSIS OF ROBOTIC SYSTEMS AND BEHAVIORS

by

ZHANG YING

B.Sc., Zhejiang University, China, 1984
M.Sc., Zhejiang University, China, 1987
M.Sc., The University of British Columbia, 1989

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

We accept this thesis as conforming
to the required standard

...............................................

...............................................

...............................................

...............................................

THE UNIVERSITY OF BRITISH COLUMBIA

September 1994

To the Memory and Spirit of
My Grandmother — Lady *Fire and Ice*

## Abstract

Robots are generally composed of electromechanical parts with multiple sensors and actuators. The overall behavior of a robot emerges from coordination among its various parts and interaction with its environment. Developing intelligent, reliable, robust and safe robots, or real-time embedded systems, has become a focus of interest in recent years. In this thesis, we establish a foundation for modeling, specifying and verifying discrete/continuous hybrid systems and take an integrated approach to the design and analysis of robotic systems and behaviors.

A robotic system in general is a hybrid dynamic system, consisting of continuous, discrete and event-driven components. We develop a semantic model for dynamic systems, that we call Constraint Nets (CN). CN introduces an abstraction and a unitary framework to model discrete/continuous hybrid systems. CN provides aggregation operators to model a complex system hierarchically. CN supports multiple levels of abstraction, based on abstract algebra and topology, to model and analyze a system at different levels of detail. CN, because of its rigorous foundation, can be used to define programming semantics of real-time languages for control systems.

While modeling focuses on the underlying structure of a system — the organization and coordination of its components — requirements specification imposes global constraints on a system's behavior, and behavior verification ensures the correctness of the behavior with respect to its requirements specification. We develop a timed linear temporal logic and timed $\forall$-automata to specify timed as well as sequential behaviors. We develop a formal verification method for timed $\forall$-automata specification, by combining a generalized model checking technique for automata with a generalized stability analysis method for dynamic systems.

A good design methodology can simplify the verification of a robotic system. We develop a systematic approach to control synthesis from requirements specification, by exploring a relation between constraint satisfaction and dynamic systems using constraint methods. With this approach, control synthesis and behavior verification are coupled through requirements specification.

To model, synthesize, simulate, and understand various robotic systems we have studied in this research, we develop a visual programming and simulation environment that we call ALERT: A Laboratory for Embedded Real-Time systems.

# Contents

# List of Figures

# List of Tables

## Acknowledgement

This thesis could not have been a success without the contributions of the members of my supervisory committee: Alan, Peter, Nick, Jeff and Dinesh.

It has been most fruitful and enjoyable for me to have had Alan Mackworth as my thesis supervisor and research collaborator. Alan's perspective and interest inspired me to enter this new world. His open-mindedness and trust provided me with the extreme freedom to explore and to learn, and his continued financial support has made the completion of this thesis possible.

I have learned much from Peter Lawrence during my long graduate studies at UBC, not only about robotics and control theory, but also about how engineers solve problems. Most concepts of this thesis are the result of my regular discussions with Peter over the years.

Nick Pippenger has been acting as an oracle in my research. There is really nothing that Nick has no knowledge of in mathematics. My confidence in my formalisms is rooted in Nick's approval. Nevertheless, I am fully responsible for every mistake I may have made in the thesis.

It was Jeff Joyce who first introduced me to programming semantics, software/hardware co-design, and to formal specification and verification. Jeff always has a crucial insight. His enthusiasm for what should be done and his belief in what can be done have greatly influenced my research.

Dinesh Pai's work on constraint-based robotics has directly stimulated many ideas in this thesis, some of which are the further development of my course project in his exciting computational robotics class. Dinesh is always a good model for me. His wide knowledge across areas in computer science, electrical and mechanical engineering makes him a good example of a good researcher in both theory and practice.

The Laboratory for Computational Intelligence (LCI) has been a supportive environment. Michael Sahota has always been there to lend me help of any kind at any time I needed. Bill Millar has always been willing to discuss with me everything about my thesis. They are the very first readers and the very effective reviewers of my thesis draft. Andrew Csinger was kind enough to provide me with the final proof-reading of my thesis. An aspiring author with a philological bent, Andrew precisely pointed out subtle problems in my use of English. I am nonetheless responsible for any remaining errors, grammatical or otherwise. Valerie Mcrae, our lab secretary, has always been the first person I turned to whenever I had a problem.

I thank Danny Bobrow for providing me with a summer internship and a stimulating environment at XEROX PARC and for referring me to XEROX WRC. It has been a most exciting experience to work with people in both SERA and the responsive environment project.

I thank the University of British Columbia for the Graduate Student Fellowships and NSERC for the Post Doctoral Fellowship they granted me.

I thank my family back in China, my parents, aunts and uncles, for their wishes, belief, understanding and expectation.

Last but not least, I thank Runping Qi, my husband, for his love and support. There is no word that is strong enough to express how much I have received from Runping; life for me would be totally different without him.

My long journey as a graduate student comes to an end. I am looking forward to new challenges in the real world.

*The heaven attained Oneness and became clear.*
*The earth attained Oneness and became settled.*
*The spirit attained Oneness and became numinous.*
*Valleys attained Oneness and became reproductive.*
*All things attained Oneness and became alive.*
— *Tao Teh Ching, Lao Tzu*

*Time attains Oneness and becomes linear.*
*Domains attain Oneness and become universal.*
*Components attain Oneness and become functional.*
*Systems attain Oneness and become alive.*
*Design and analysis attain Oneness and become productive.*
— *Zhang Ying*

# Chapter 1

# Motivation and Introduction

In applications such as nuclear and chemical plants, forest industries, space and undersea exploration, there is a demand for intelligent, reliable, robust and safe robots. Building control systems for autonomous robots working in complex environments is an important challenge for research in computer science, electrical and mechanical engineering.

Robots are generally composed of electromechanical parts with multiple sensors and actuators. Robots should be reactive as well as purposive systems, closely coupled with their environments; they must deal with inconsistent, incomplete and delayed information from various sources. Robots are usually complex, hierarchically organized and physically distributed; each component functions according to its own dynamics. The overall behavior of a robot emerges from coordination among its various parts and interaction with its environment. We call the coupling of a robot and its environment a *robotic system*, and the dynamic relationship of a robot and its environment the *robotic behavior*.

A robot controller (or control system) is a subsystem of a robot, designed to regulate its behavior to meet certain requirements. In general, a robot controller is an integrated software/hardware system implemented on various digital/analog devices. Designing control systems for robots that meet certain requirements has become an active topic studied in many areas, such as reactive systems, intelligent systems, real-time embedded systems and integrated hybrid systems. The issues raised in this interdisciplinary research range from programming languages and software/hardware engineering to control theory and dynamic systems.

In this thesis, we establish a unified foundation for modeling, specifying and verifying discrete/continuous hybrid systems and take an integrated approach to the design and analysis of robotic systems and behaviors.

## 1.1 The Problems

A robotic system is a dynamic system. The study of *dynamic systems* is the study of dynamics and the study of systems. The study of *dynamics* is concerned with how things change over time. The study of *systems* is concerned with how a system's overall behavior is generated through interaction among its components.

From the systemic point of view, a robotic system is a coupling of a robot to its environment, while the robot is a coupling of a controller to its plant (Figure 1.1). The roles of these three

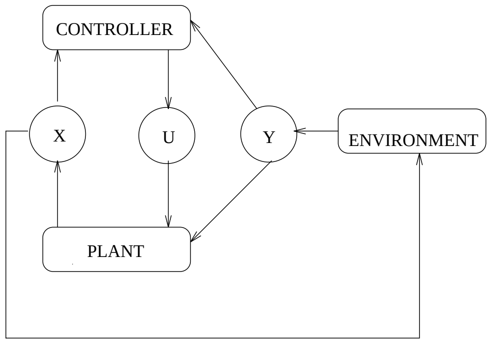


Figure 1.1: A robotic system

subsystems can be characterized as follows:

- *Plant*: a plant is a set of entities that must be controlled to achieve certain requirements. For example, a robot arm with multiple joints, a car with throttle and steering, an airplane or a nuclear power plant can each be considered as the *plant* of some robotic system.

- *Controller*: a controller is a set of sensors and actuators, which, together with software/hardware computational systems, senses the observable states of the plant ($X$) and the environment ($Y$), and computes desired control inputs ($U$) to actuate the plant. For example, an analog circuit, a program in a digital computer, various sensors and actuators might be parts of the *controller* of some robotic system.

- *Environment*: an environment is a set of entities beyond the (direct) control of the controller, with which the plant may interact. For example, obstacles to be avoided, objects to be reached, and rough terrain to be traversed might from part of the *environment* of some robotic system.

From the dynamics point of view, the relationship of a robot and its environment changes over time. In order to develop a robotic system, analyze its behavior and understand its underlying physics, we need a mathematical model for characterizing the behaviors of its components and deriving the behavior of the overall system.

Let us introduce an example that will be used throughout this thesis. In our Laboratory for Computational Intelligence, a testbed has been installed for radio-controlled cars playing soccer [SM94]. Each "soccer player" has a car-like mobile base. It can move forward and backward with a throttle setting, and can make turns by steering its two front wheels. However, it cannot move sideways and its turns are limited by mechanical stops in the steering gear.

Figure 1.2 illustrates the configuration of a car. Let $v$ be the velocity of the car and $\alpha$ be the current steering angle of the front wheels; $v$ and $\alpha$, for now, can be considered as control inputs to the car. The dynamics of the car can be simply modeled by the following differential equations [Lat91]:

$$\dot{x} = v\cos(\theta), \quad \dot{y} = v\sin(\theta), \quad \dot{\theta} = v/R \tag{1.1}$$

where $(x, y)$ is the position of the tail of the car, $\theta$ is the heading direction and $R = L/\tan(\alpha)$ is the turning radius given the length of the car $L$. The controller of such a car is equipped

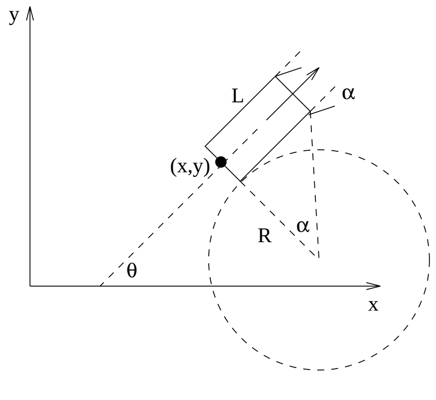


Figure 1.2: The configuration of a car

with both digital and analog devices [SM94].

Although differential equations have been used to model continuous dynamic systems, they are not sufficient to model discrete and event-driven systems. Although the continuous and discrete components of a system can be modeled and analyzed separately, it is essential to use a unitary model for discrete/continuous hybrid systems, in order to derive the behavior of the overall system.

Control systems are designed to meet certain requirements. Typical requirements include safety, reachability and persistence. *Safety* declares that a system should never be in a certain situation. *Reachability* declares that a system should reach a certain goal eventually. *Persistence* declares that a system should approach a certain goal infinitely often. A formal language for requirements specification is essential for characterizing desired properties of a system and a formal method for behavior verification is essential for ensuring the correctness of the behavior of the system with respect to some requirements specification.

Yet another challenging task in the design of a robotic system is control synthesis, i.e., given the dynamics of the plant and the environment, produce a controller so that the behavior of the overall system meets certain requirements.

As a whole, we propose four problems involved in the design and analysis of robotic systems and behaviors:

- How to model a robotic system?
- How to specify desired properties?
- How to synthesize a control system according to its requirements specification?
- How to guarantee the robot will do the right thing?

Figure 1.3 presents an overall picture of the problems and our corresponding solutions that we will develope in this thesis.

## 1.2 The Proposed Solutions

We claim in this thesis that a unified foundation for discrete/continuous hybrid dynamic systems can be established and an integrated approach to the design and analysis of robotic systems and behaviors should be taken.

First, we develop a semantic model for dynamic systems, that we call Constraint Nets (CN). CN introduces an abstraction and a unitary framework to model discrete/continuous hybrid systems. CN provides aggregation operators to model a complex system hierarchically; therefore, the dynamics of the environment as well as the dynamics of the robot can be modeled individually and then integrated. CN supports multiple levels of abstraction, based on abstract algebra and topology, to model and analyze a system at different levels of detail. CN, because of its rigorous foundation, can be used to define programming semantics of real-time languages for control systems.

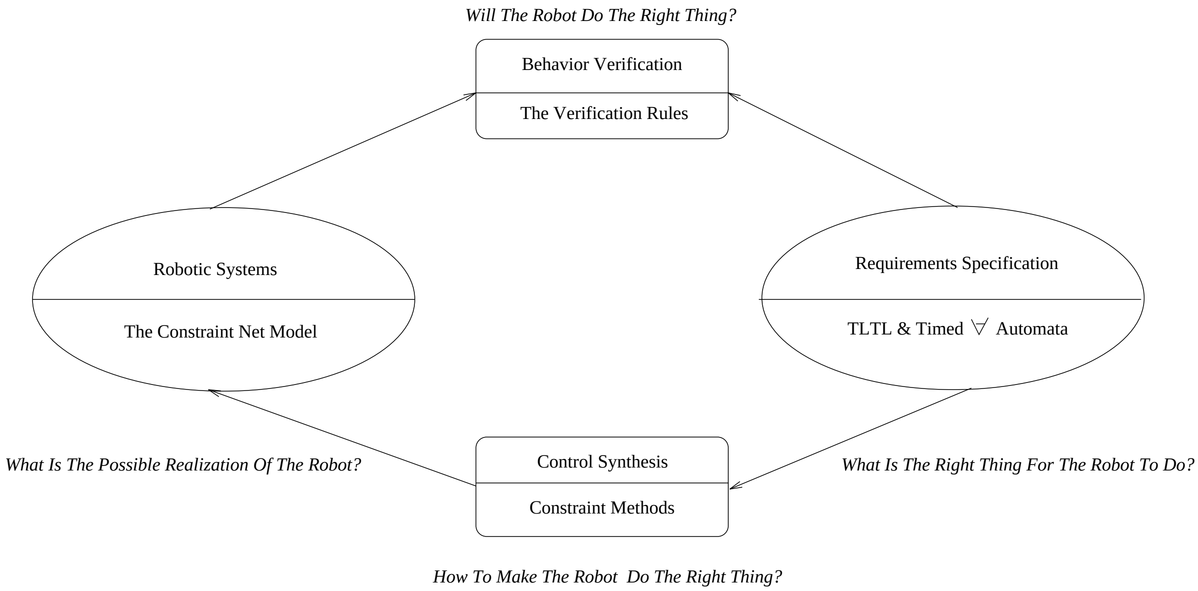


Figure 1.3: The problems and our solutions

Second, we develop a timed linear temporal logic (TLTL) and timed $\forall$-automata as specification languages. TLTL is a linear temporal logic developed on abstract time and domain structures. Timed $\forall$-automata are essentially finite automata that accept timed traces; yet they are powerful enough to specify properties of sequential and timed behaviors of hybrid systems, such as safety, reachability, persistence and real-time response. We develop a formal verification method for timed $\forall$-automata specification, by combining a generalized model checking technique for automata with a generalized stability analysis method for dynamic systems. This verification method can be semi-automated for discrete time systems and further automated for finite domain systems.

Third, we develop a systematic approach to control synthesis from requirements specification, by exploring a relation between constraint satisfaction and dynamic systems using constraint methods. With this approach, control synthesis and behavior verification are coupled through requirements specification. In particular, requirements specification imposes global constraints over a system's behavior and controllers can be synthesized as embedded constraint solvers that solve constraints over time. For complex control systems, we advocate a two-dimensional hierarchical structure. A system with such hierarchical structure will simplify design and analysis significantly.

## 1.3 Semantic Model and Behavior Analysis

In the past decades, models for continuous, discrete and event-driven dynamic systems have been developed and matured. Models for continuous and discrete dynamic systems include differential and difference equations, respectively [Lue79, San90]. Models for event-driven dynamic systems include Mealy-Moore Machines [Mea55, Moo56], Petri Nets [Pet81], Calculus for Communicating Systems (CCS) [MM79] and Communicating Sequential Processes (CSP) [Hoa85]. However, a robotic system in general is a continuous/discrete hybrid dynamic system. First, the plant and the environment of a robotic system are normally modeled in continuous dynamics. Second, most advanced robots today are controlled by distributed and asynchronous processes in digital computer networks, as well as by analog circuits. In order to develop a system whose behavior can be analyzed and understood, a model for hybrid dynamic systems is essential.

In the last two years, hybrid systems have become a focus of interest of a wide community for two reasons. One is that analog computation once again is gaining attention because of the neural net model and analog VLSI technology. Another is that the use of computers to control and monitor continuous dynamic systems shows increasing importance.

Our approach to developing a model for hybrid systems is motivated by the following arguments. First, hybrid systems consist of interacting discrete and continuous components. Instead of fixing a model with particular time and domain structures, a model for hybrid systems should be developed on both abstract time structures and abstract data types. Second, hybrid systems are complex systems with multiple components. A model for hybrid systems should support hierarchy and modularity. Third, hybrid systems are generalizations of basic discrete or continuous systems. A model for hybrid systems should be at least as powerful as existing computational models. In short, a model for hybrid systems should be unitary, modular, and powerful.

In this thesis, we start with a general definition of time. Time is a linearly ordered set. In addition, a metric distance is associated with any two time points and a measure is associated with some intervals of time points. Such a time structure abstracts the notion of event-based as well as discrete and continuous time. We then examine domain structures in abstract algebra and topology so that discrete and continuous domains can be studied in a unitary framework. Given a time structure and a domain structure, we define two basic types of element in dynamic systems: traces that are functions from time to domains, and transductions that are mappings from traces to traces with the causal restriction, viz., the output value at any time is determined only by its input values up to that time. For example, a finite state automaton with an initial state defines a transduction from input traces to state traces, and temporal integration is a typical transduction in continuous dynamics.

We then develop the Constraint Net model on an abstract dynamics structure composed of a multi-sorted set of trace spaces and a set of basic transductions: transliterations (memory-less combinational processes), transport delays and unit delays (sequential processes), and event-driven transductions. Event-driven transductions play an important role in this model, acting as ties between continuous and discrete time components, or as synchronizers among asynchronous components.

Syntactically, a constraint net is a graph with two types of node: locations and transductions, and with a set of connections between locations and transductions. Locations are depicted by circles, transductions by boxes and connections by arcs. A location is an input iff it is not connected to the output of any transduction. A constraint net is open if there is an input location; it is otherwise closed.

Semantically, a constraint net represents a set of equations, with locations as variables and transductions as functions. The semantics of the constraint net, with each location denoting a trace, is the least solution of the set of equations.

A complex system is generally composed of multiple components. We define a module as a constraint net with a set of locations as its interface. A constraint net can be composed hierarchically using modular and aggregation operators on modules. The semantics of a system can be obtained hierarchically from the semantics of its subsystems and their connections.

For example, Equation 1.1 is denoted by an open constraint net, as shown in Figure 1.4 in which $sin$, $cos$, $tan$ and $*$ are transliterations, and $\int$ is a temporal integrator. A module can be defined with locations $v, \alpha, x, y, \theta$ as its interface.

In general, we can model a control system as a module that can be further decomposed into a hierarchy of interactive modules. The higher levels are composed of event-driven transductions and the lower levels are analog control components. Furthermore, the environment of the robot can be modeled as a module as well. A robotic system (Figure 1.1) can be modeled as an integration of a plant, a controller and an environment. Formally, the semantics (or behavior) of the system is the solution of the following equations:

$$X = PLANT(U,Y), \;\; U = CONTROLLER(X,Y), \;\; Y = ENVIRONMENT(X).$$

As we can see here, a robot, composed of a plant and a controller, is an open system, and a robotic system, composed of a robot and its environment, is a closed system.

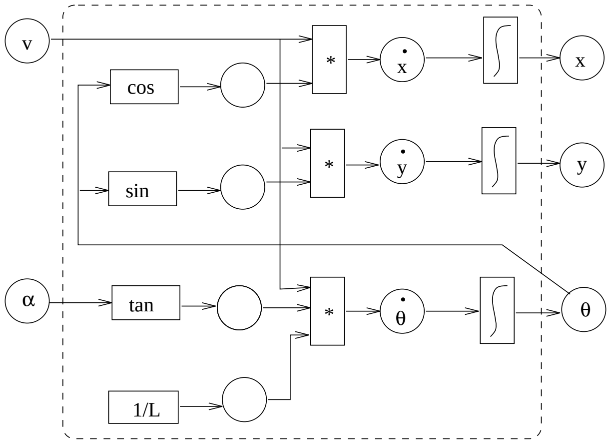


Figure 1.4: The constraint net of Equation 1.1

We finally study the issue of behavior analysis for robotic systems. We define the concepts of abstraction and refinement for time and domains based on homomorphism and quotient algebra, and derive equivalence relations on dynamic systems.

A semantic model for hybrid dynamic systems defines a formal semantics for real-time programming that may involve hardware/software co-design and digital/analog hybrid computation. A formal semantics, in turn, supports the formal analysis of real-time embedded systems.

## 1.4 Requirements Specification and Behavior Verification

A semantic model for a robotic system can be considered an executable specification that defines the underlying structure of the system, i.e., the organization and coordination of the components. Even though a system can be modeled at different levels of abstraction, each component is local in terms of constraints on time and its input/output domains. A requirements specification, in contrast, imposes global constraints on a system's behavior.

Let us consider the car-like robot we introduced previously. We will design control systems for such a robot to perform the following tasks:

1. *Maze Traveler*: traveling in a maze and trying to get out of the maze

   The environment of this system is a maze that is composed of various static obstacles. A requirements specification for this robot is to get out of the maze.

2. *Ball Shooter*: tracking a moving ball and carrying the ball to a target

   The environment of this system is a moving ball, a target and a field with boundaries. A requirements specification for this robot is to eventually kick or carry the ball to the target.

A requirements specification declares what a system should achieve, while an executable specification shows how a system is implemented (at a certain level of abstraction). A formal language for requirements specification is essential for both formal verification and systematic synthesis. Since robotic behaviors are inherently temporal, it is natural to adopt temporal logic as a language for requirements specification.

We first develop a timed linear temporal logic (TLTL) as a specification language, in which "linear" stands for linear orders and "timed" indicates metric distances between time points. Let modal operators $\Diamond$ and $\Box$ denote "eventually" and "always," respectively. One possible control for the maze traveler is to make the robot move in a particular direction persistently in order to escape a maze of finite size. This property can be specified in TLTL as $\Box\Diamond ME$ where $ME$ is a predicate for moving east, or $|\theta| < \delta$ and $v > \epsilon$ for small $\delta > 0$ and $\epsilon > 0$; $\Box\Diamond P$ is normally referred to as liveness or persistence. Kicking or carrying a ball to a target eventually can be specified as $\Diamond\Box BT$ where $BT$ is a predicate for the ball arriving at the target, or $distance(Ball, Target) < \epsilon$; $\Diamond\Box G$ is normally referred to as reachability or goal achievement. In addition, operators can be augmented with metric time so that real-time properties can be specified. For instance, $\Box(E \rightarrow \Diamond^{\tau} R)$ declares that any event $(E)$ will be responded to $(R)$ within time $\tau$.

Even though TLTL can provide a formal specification, there is no general procedure for verifying the behavior of a system. An alternative to temporal logic for representing sequential behaviors is automata. If we take the behavior of a system as a language, then a specification can be represented as an automaton, and the verification checks the inclusion relation between the behavior of the system and the language accepted by the automaton.

We then develop timed $\forall$-automata, a generalization of (discrete) $\forall$-automata [MP87], for requirements specification. $\forall$-automata have been proposed for the specification and verification of concurrent systems; they are essentially finite automata that accept $\omega$-languages, i.e., sets of sequences of infinite length. We extend $\forall$-automata to timed $\forall$-automata to accept timed discrete/continuous traces.

There are two reasons to adopt automata-type languages. First, automata provide graphical representations, which are more illuminating, and, in some cases, simpler than their textual counterparts. The corresponding timed $\forall$-automata specification of $\Box\Diamond ME$, $\Diamond\Box BT$ and $\Box(E \rightarrow \Diamond^{\tau} R)$ are shown in Figure 1.5 (a), (b) and (c), respectively, where nodes are automaton-states and arcs are state transitions; $\Diamond$ denotes a recurrent state, indicating a condition the system should satisfy periodically, and $\Box$ denotes a stable state, indicating a "final condition" the system should satisfy.

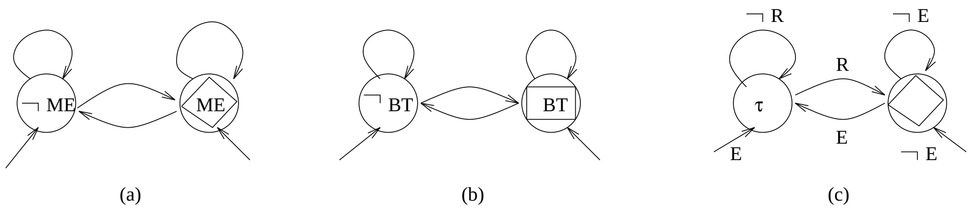


Figure 1.5: Timed $\forall$-automata specification

Second, automata facilitate a formal verification method— a set of sound and complete verification rules— based on a model checking technique and a stability analysis method. Given a constraint net model of a discrete time system, the set of verification rules can be used to deduce a set of state formulas that can be checked using an automatic or interactive theorem prover. If, in addition, the discrete time system is of a finite number of states, the set of verification rules can be used to deduce an automatic verification algorithm that has polynomial time complexity in both the size of the specification and the size of the system.

## 1.5 Control Synthesis and Robotic Architecture

The problem of behavior verification in general is hard. However, a well-organized and structured system will simplify the problem of verification. Therefore, robotic architecture plays an important role in both design and analysis.

We first develop a general framework for the synthesis of control systems from requirements specification in timed $\forall$-automata. In this framework, constraint satisfaction is viewed as a dynamic process approaching the solution set of the given constraints asymptotically. A constraint solver is a constraint net whose semantics corresponds to a dynamic process of this type. Constraint solvers can be systematically synthesized based on various constraint methods. In particular, continuous time constraint solvers are based on gradient methods and discrete time constraint solvers are based on relaxation algorithms in numerical computation. Control synthesis and behavior verification are coupled through requirements specification. While requirements specification imposes constraints over the behavior of a system, the controller is designed as a set of embedded constraint solvers that, together with the dynamics of the plant and the environment, solve the constraints over time.

A control system is a complex system. In this thesis, we advocate a modular and hierarchical robotic architecture. We study two types of hierarchy: composition hierarchy that is the modular or compositional structure of a system, and interaction hierarchy that is the communication or interaction structure of a system. Furthermore, we propose a two-dimensional structure for the interaction hierarchy: abstraction hierarchy that reflects the granularity of time and domain structures, and arbitration hierarchy that reflects constraint priorities.

As a whole, a control system is designed as a set of embedded constraint solvers distributed over the two-dimensional interaction hierarchy. Constraint solvers at lower levels of the abstraction hierarchy are normally either continuous or discrete at fast and fixed sampling rates, while constraint solvers at higher levels are either event-driven or with noticeable computational delays. Constraint solvers at the same level of the abstraction hierarchy are coordinated through various arbitrations, which form an arbitration hierarchy.

## 1.6 How This Thesis Fits In

This thesis provides a foundation for the design of robotic systems and the analysis of robotic behaviors. Robotic systems are integrated hybrid systems and robots are intelligent real-time systems. In this section, we illustrate how this thesis relates to these subjects.

### 1.6.1 Integrated hybrid systems

*Integrated hybrid systems* are systems consisting of a non-trivial mixture of discrete and continuous components, such as a controller realized by a combination of digital and analog circuits, a robot composed of a digital controller and a physical plant, or a robotic system consisting of a computer-controlled robot coupled to a continuous environment. Integrated hybrid systems are more general than traditional real-time systems; the former can be composed of continuous subsystems in addition to discrete or event-controlled components. With the development of computation, control and communication technologies, integrated hybrid systems will come to everyday life, in such things as computer-controlled TVs, autonomous cars and smart buildings.

Integrated hybrid systems engineering is a combination of computer engineering and control engineering. The life cycle for computer engineering includes specification, implementation and verification. The life cycle for control engineering includes modeling, design and analysis. In practice, integrated hybrid systems require novel design principles and development environments for modeling, design and analysis, as well as specification, implementation and verification. From a theoretical point of view, integrated hybrid models, languages, algorithms and programs propose brand new approaches to computation and control.

Research and development in integrated hybrid systems have become very active for the last two years. Typical commercial products for integrated modeling and simulation environments are Simulink [Incc] and SystemBuild [Incb]. Both Simulink and SystemBuild provide graphical modeling environments, simulation and animation tools, for discrete/continuous hybrid systems, as well as linear systems analysis libraries. Both systems support modularity and hierarchy with dataflow-, net- or circuit-like representations. Both systems have their advantages: Simulink is more flexible and simpler, while SystemBuild has more built-in functions. In addition, SystemBuild supports automatic code generation [Inca], which can greatly reduce the cost and time for developing real-time embedded control systems.

Some research on languages of hybrid systems for modeling and simulation has also been proposed: a typical example is SHSML: Standard Hybrid Systems Modeling Language [Tay92]. SHSML is based mostly upon the conceptual definition of a hybrid system that underlies hybrid DSTOOL [GN92] and on the modeling and simulation environment provided by SIMNON [Elm77]. A system modeled by SHSML consists of continuous (continuous time and domain, e.g., differential equations), discrete (discrete time and continuous domain, e.g., difference equations) and logic (discrete time and domain) components. SHSML can be considered as an architecture definition language for software/hardware co-design.

Some theoretical work on hybrid models and topologies has been carried out recently. There are two types of model: models for synchronous systems and models of hybrid automata. SIGNAL [BL90] and LUSTRE [CPHP87] are based on the synchronous models derived from the *Dynamic Network Processes* model [Kah74], with the augmentation of clocks. Synchronous models can be considered as general models for discrete time and hybrid domain dynamic systems. Phase transition systems [MMP91], event-driven hybrid systems [NK93a] and hybrid automata [ACHH93] are automata-based models in which states are differential equations, trajectories, or continuous activities. The theory of topological structures for hybrid domains has been brought up [NK93a], so that continuity, stability and controllability of systems with hybrid domains can be further studied.

Our work contributes to the research and development of integrated hybrid systems in the following ways.

First, Constraint Nets serve as a formal semantic model for hybrid dynamic systems; the mathematical rigor underlies the foundation for both modeling and simulation. Just as with formal semantics for programming languages, formal semantics for modeling, control and simulation will not only bring unambiguousness and precision to existing real-time programming languages and simulation environments like Simulink and SystemBuild, but will also provide insight into the design of new programming languages for hybrid systems.

Second, unlike other efforts to combine discrete and continuous models, we begin by defining concepts of dynamic systems on the abstraction that captures both discrete and continuous time and domains. The Constraint Net model is a model of models, preserving the general structure of dynamic systems. Constraint Nets can be used not only for system design with modeling, control and simulation, but also for behavior analysis with refinement and abstraction.

### 1.6.2 Intelligent real-time systems

*Intelligent real-time systems* are reactive as well as purposive systems, closely coupled with unstructured/unpredictable environments, such as robots that should promptly make correct decisions in various situations, and accurately perform complex tasks in changing environments. Intelligent real-time systems have attracted researchers from both the Artificial Intelligence (AI) and real-time control communities [Sch91]. In the past, AI and control have focused on solving different problems with different interests and applications [DW91]. AI systems focus on high-level activity like planning, reasoning, and inferencing with facts and rules in knowledge base, while control systems involve sensing and acting in real time. Currently, there are two major trends in the cross-fertilization of AI and control: one is to combine AI techniques (planning, knowledge and belief representation, symbolic processing, temporal and qualitative reasoning, inference rules, heuristic search, etc.) with control theory (linear and nonlinear control, adaptive and fuzzy control, etc.), and the other is to experiment with reactive or situated systems. From the AI point of view, the former is revisionary and the latter is revolutionary. The key differences are the understanding of what is intelligence and the methodology of how to realize intelligence in embedded real-time systems.

In cognitive science, intelligence is considered as the ability to plan, reason or apply knowledge to manipulate one's environment. For robots, intelligence reflects ways of acquiring, forming, storing and maintaining knowledge as well as planning and reasoning about actions to achieve desired goals. Much work has been done in AI on knowledge representation, planning and reasoning. However, it has been shown that domain-independent representation, planning and reasoning are difficult to fit in to a real-time framework. Many planning and reasoning problems are computationally intractable [Cha87]. For both planning and reasoning, the more powerful and general the knowledge and action representations are, the less feasible it is that these computations can be realized. For example, universal planning [Sch87], generating plans of mappings from situations to actions (reaction plans), and planning under uncertainty [Qi94], producing plans with maximum expected utilities or minimum expected cost, are in general harder than planning action sequences. For any real applications, some compromise between the complexity of plan representation and the complexity of planning must be achieved. Two typical strategies have been studied: one is to adopt reactive planning, and the other is to apply any-time algorithms. Reactive planning [GL87, RK89] produces a partial planning strategy given current states, so that the plan representation is simple, but planning and execution are tightly coupled to realize reactive and situated behaviors. Any-time algorithms [BD89, Bod91] are algorithms producing results approaching the solution over time, so that a compromise can be made between the accuracy of the results and the time for computation.

In behavior science, real-time interaction with one's environment is considered as the intrinsic characteristic of intelligence. Furthermore, such intelligence is not from deliberate decision, but from distributed constraint satisfaction and cooperation among various components in the system. This view of intelligence is shared by many researchers in AI and psychology (Brooks [Bro91], Meas [Mae89], Agre and Chapman [AC87], Hewitt [Hew91], Minsky [Min86], Beer [Bee90], Braitenberg [Bra84]). Brooks and his colleagues did very interesting work on building artificial creatures [BCN88, Bro88, Con90]. Brooks [Bro86, BC86] proposed a robust, layered control system for mobile robots, called the *subsumption architecture*. Unlike the traditional decomposition of a mobile robot control system into functional modules, Brooks decomposed a mobile robot control system into task-achieving behaviors. Maes [Mae89] suggested that rational action selection could be modeled as an emergent property of an activation/inhibition dynamics among modules. Similarly, Hewitt [Hew91], Minsky [Min86] and researchers in Distributed AI [Huh87] argued that intelligence comes from the interaction between multiple components and their environment. Agre and Chapman [AC88] claimed that pure planning and reasoning are not suitable for dealing with inconsistent, uncertain and immediate situations; rather, reaction and moment-to-moment improvisation play a central role in most activity. From the point of view of an experimental psychologist, Braitenberg [Bra84] studied various incrementally complex life-like systems. Beer [Bee90] performed a series of simulations of an artificial insect with adaptive behavior.

Our work contributes to the research and development of intelligent real-time systems in the following ways.

First, by avoiding the controversial issues surrounding intelligence, we focus on formal methods for specifying properties of behaviors and on systematic approaches to synthesizing control systems. Because there can be no rigorous definition of intelligent or stupid behaviors, we use the concept of *desired properties of behaviors*. Furthermore, behavior equivalence and system robustness are formalized and studied.

Second, instead of advocating one particular type of implementation (knowledge-based or reaction-based) for intelligent real-time systems, we focus on general structures of complex systems and principles for the organization of hybrid dynamic systems. Because Constraint Nets provide a unitary model for components with diversity in both time and domain structures (continuous, discrete or event-based time, and real, integer, logical, or symbolic variables), the behavior of an overall system can be derived and analyzed.

## 1.7 Thesis Outline

This thesis consists of three major parts. Part I presents a mathematical structure of dynamics, the syntax and semantics of the Constraint Net model, and the method of behavior analysis based on algebra and topology. Part II develops two languages, TLTL and timed $\forall$-automata, for requirements specification, and examines formal verification methods for timed $\forall$-automata specification. Part III discusses a relation between behavior verification and control synthesis through requirements specification using constraint satisfaction, and proposes a robotic architecture with hierarchy and modularity. Each part starts with an introduction, and ends with a summary of our approaches and a survey of related work.

Mathematical preliminaries on topology, algebra and analysis are presented whenever necessary; however, most of the proofs are given in Appendix A. A visual programming and simulation environment, ALERT — A Laboratory for Embedded Real-Time systems — has been developed for modeling, synthesizing, simulating, and understanding various robotics systems studied in this research. ALERT and some simple examples are presented in Appendix B. The car-like robot is used as a running example throughout the thesis. Two more complex examples, an elevator system and a hydraulically actuated robot arm, are presented in Appendix C to further illustrate our approaches. A model estimation technique for the car-like robot is discussed in Appendix D.

## 1.8 A Guide to the Reader

We assume that, by now, you have read this chapter, Motivation and Introduction. You also have an overall picture of the problems and our proposed solutions. In the rest of this thesis we will systematically develop these solutions.

We take an integrated approach towards modeling, specification, verification and control synthesis, each of which, nevertheless, is a research topic by itself.

Those who are interested in real-time/hybrid models should start with Part I. Besides using standard techniques in denotational semantics like partial order topologies, we develop topological structures of time, domains and traces. Based on these topological structures, we develop, in series, the concepts of primitive and event-driven transductions, nets, modules, semantics and behaviors. Even though the minimum background for understanding this part is elementary discrete mathematics (set, relation, function) and calculus (integrals and derivatives), knowledge of dynamic systems, general topology, metric space and partial order would be an asset.

Those who are interested in real-time specification/verification should continue onto Part II. The minimum materials from Part I for understanding Part II are topological structures of time, domains and traces (Chapter 3), and general concepts of behaviors and requirements specification (Chapter 6). Besides predicate calculus and the first order logic, knowledge of dynamic systems, temporal/modal logic and regular languages would be an asset.

Those who are interested in planning and control should not miss Part III. The minimum materials from Part I and Part II for understanding Part III are parameterized nets (Chapter 4) and generalized $\forall$-automata (Chapter 10). Knowledge of nonlinear dynamics and constraint methods would be an asset.

Those who are interested in applications of the theory should finish (or start) with the appendixes, where the modeling and simulation environment is discussed, and the methods developed in this thesis are illustrated by examples.

The problems of design and analysis are interesting and challenging enough to spend more time on. We hope everyone, with every kind of background, will find something useful in this thesis at every reading.

# Part I

# Semantic Model and Behavior Analysis

*The* Tao *that can be taught is not the everlasting* Tao.
*The Name that can be named is not the everlasting Name.*
*That which has no name is the origin of heaven and earth.*
*That which has a name is the Mother of all things.*
— *Tao Teh Ching, Lao Tzu*

*A system that can be modeled is not the system itself.*
*A model that can be made is not the absolute model.*
*That which has no model is the origin of a system.*
*That which has a model is the understanding of the system.*
— *Zhang Ying*

# Chapter 2

# Introduction

In this chapter, we present an overview of Part I, Semantic Model and Behavior Analysis. There are four major chapters in Part I. Chapter 3 gives a topological structure of dynamics. Chapter 4 describes the Constraint Net model, its syntax and semantics. Chapter 5 illustrates the modeling aspects of the Constraint Net model and discusses its computational power. Chapter 6 focuses on behavior analysis.

## 2.1 Topological Structure of Dynamics

One important feature of this research is abstraction. The purpose of abstraction is for generalization. Hybrid systems are systems with possibly multiple data types and multiple time structures. Instead of combining different models, we extract the commonalities shared by various models for dynamic systems.

First, we develop a general structure of *time*, capturing linearity, metric and measure properties of time, i.e., for any two time points, there are two important attributes: order and metric distance, and for any interval of time points, there is a measure. Discrete and continuous time can be modeled by this structure uniformly. Two time structures may relate to each other in terms of reference mapping.

Second, we develop a general structure of *domains* that can be either simple or composite. Domains are associated with metrics capturing discreteness or density. They are also associated with partial orders characterizing definedness or information hierarchy.

Third, we develop a general structure of *traces* that are mappings from time to domains. We further formalize event traces as a special kind of trace for modeling event-based time.

Fourth, we define *transductions* as causal mappings from traces to traces. We further characterize two types of transduction: *primitive* transductions and *event-driven* transductions.

A primitive transduction is a functional composition of *transliterations* and *delays* for memoryless processes and sequential processes, respectively. An event-driven transduction is a primitive transduction augmented with an event trace input that defines an event-based time structure for the primitive transduction.

Finally, we define a *dynamics structure*, based on a reference time structure and a domain structure, as a pair consisting of a multi-sorted set of trace spaces and a set of primitive and event-driven transductions.

All structures are defined on two types of topology: partial order topology and metric topology. The preliminary concepts of general topology, partial order and metric space are given first, following which all concepts are defined formally.

## 2.2 The Constraint Net Model

We start with the syntax of constraint nets. A *constraint net* is a bipartite graph, with two types of node: *locations* and *transductions*. A location is an *input* if it is not connected to the output of any transduction; it is otherwise an *output*. A *module* is a constraint net with a set of locations as its *interface* and with the rest of its locations as *hidden* locations. A complex module can be composed hierarchically from simple ones. Also a module can be considered as an abstraction of its net: hidden inputs capture nondeterminism, and hidden outputs capture information encapsulation.

We then develop the semantics of constraint nets using continuous algebra. Locations denote traces and transductions are causal mappings from traces to traces. A constraint net denotes a set of equations, each of which corresponds to a transduction. The semantics of a constraint net is the *least solution* of the set of equations. We further study the well-definedness of constraint nets and modules, and its relationship with algebraic loops. We finally introduce parameterized nets and limiting semantics for temporal integration.

## 2.3 Modeling in Constraint Nets

The Constraint Net model (CN) is an abstraction of dataflow-like models. CN provides a unitary framework to model a hybrid system composed of components of different dynamics.

We first define various event generators and synchronizers. Using event generators and synchronizers, components of different time structures can be coordinated.

We then illustrate the modeling methodology with an example of a typical hybrid system, a maze traveler, whose overall system is composed of both discrete and continuous components.

We finally explore the computational power of constraint nets, in terms of sequential computation and analog computation. We discover that a constraint net can model discrete sequential computation in which the sequential order of a computation is controlled by events, and similarly, that it can model nondeterministic choices and time-out. We prove, for a simple domain structure, that the Constraint Net model is as powerful as the Turing Machine model for sequential computation. We also establish, for analog computation, a relationship of smooth non-hypertranscendental functions and constraint nets of continuous dynamics.

## 2.4 Behavior Analysis

We discuss the basic concepts of behavior analysis. Intuitively, the behavior of a system is the set of observable traces of the system. We characterize two important types of behavior: state-based behavior and time-invariant behavior. We then briefly discuss the following issues: requirements specification, robustness of parameterized nets with respect to requirements specification, and behavioral complexity that is analogous to functional complexity in sequential computation.

Since the Constraint Net model is developed on abstract time and domains, we can model and analyze a system at different levels of abstraction. We first define the concepts of abstraction and refinement for time and domains, and then derive the concepts of abstraction and equivalence for behaviors.

## 2.5 Summary and Related Work

Part I is the kernel and is considered as one of the major contributions of this thesis. It is the first time that a unitary and comprehensive model for discrete/continuous hybrid systems has been proposed. The theory that supports the model is developed from algebra and topology. Even though similar techniques such as continuous algebra and fixpoint theory have been applied to the semantics of sequential or concurrent programs, it is the first time that such techniques are applied to the semantics of dynamic systems.

# Chapter 3

# Topological Structure of Dynamics

In this chapter, we present a topological structure of dynamics. We start with concepts in general topology, then focus on two particular types of topology: partial order topology and metric topology. Based on these two types of topology, we formalize time, domain and trace structures. We then present transductions as causal mappings from traces to traces. Finally, we define abstract dynamics structures.

## 3.1 General Topology, Partial Order and Metric Space

In this section, we summarize some mathematical preliminaries that will be used later. For a more comprehensive introduction, the reader is referred to other sources (e.g., [Gem90, Hen88, Vic89, MA86, War72, Roy88]).

### 3.1.1 General topology

General topology studies the limit-point concept based on which connectivity and continuity can be defined.

**Definition 3.1.1 (Topology and Topological space)** *Let $X$ be a set and $\emptyset$ be an empty set. A collection $\tau$ of subsets of $X$ is said to be a* topology *on $X$ iff the following axioms are satisfied:*

- *$X \in \tau$ and $\emptyset \in \tau$.*
- *If $X_1 \in \tau, X_2 \in \tau$, then $X_1 \cap X_2 \in \tau$.*
- *If $X_i \in \tau$ for all $i \in I$, then $\cup_i X_i \in \tau$, given an arbitrary index set $I$.*

*$\langle X, \tau \rangle$ is called a* topological space.

The members of a topology $\tau$ are said to be $\tau$-*open* subsets of $X$, or merely *open* if no ambiguity arises. A subset $S$ of $X$ is *closed* iff $X - S$ is open. We will use $X$ to denote topological space $\langle X, \tau \rangle$ if no ambiguity arises.

**Proposition 3.1.1** *For any topology on $X$, $X$ and $\emptyset$ are both open and closed.*

Two topologies $\tau_1$ and $\tau_2$ on a set can be compared in the following sense: $\tau_1$ is a *finer* topology than $\tau_2$ iff $\tau_1 \supseteq \tau_2$.

There are two extreme topologies on $X$. The coarsest topology is *trivial topology* in which only $X$ and $\emptyset$ are open and the finest topology is *discrete topology* in which all subsets of $X$ are open.

Let $x \in X$ and $N(x)$ be a $\tau$-open subset of $X$ containing $x$, $N(x)$ is called a *neighborhood* of $x$ w.r.t. $\tau$. A point $x$ of $X$ is a *limit point* of a subset $S$ of $X$ iff every neighborhood of $x$ also contains a point of $S$ distinct from $x$, i.e., $\forall N(x), N(x) \cap S - \{x\} \neq \emptyset$.

Topologies can also be defined in terms of limit points.

**Proposition 3.1.2** *(1) A subset is closed iff it includes all its limit points. (2) A topology is trivial iff every point $x$ is a limit point of any subset with elements distinct from $x$. A topology is discrete iff no point is a limit point of any subset.*

Now we define connectivity and continuity on topological spaces. A topological space is *separated* if it is the union of two disjoint, non-empty open sets; it is otherwise *connected.*

**Proposition 3.1.3** *A topological space is connected iff the only sets that are both open and closed are the empty set and the total set.*

Let $\langle X, \tau \rangle$ and $\langle X', \tau' \rangle$ be topological spaces. A function $f : X \to X'$ is *continuous* iff for any $\tau'$-open subset $S'$ of $X'$, $f^{-1}(S') = \{x | f(x) \in S'\}$ is $\tau$-open.

**Proposition 3.1.4** *(1) Continuous functions are closed under functional composition. (2) A function $f : X \to X'$ is continuous, iff $x \in X$ is a limit point of $S \subset X$ implies that $f(x)$ is a point or a limit point of $f(S) = \{f(x) | x \in S\}$.*

It is natural to ask if there exists any smaller collection of subsets that can be used to represent the open sets. The answer is affirmative, and the following definitions provide such collections.

**Definition 3.1.2 (Basis and Subbasis)** *A subset $\mathcal{B}$ of a topology $\tau$ is said to be a* basis *for $\tau$ iff each member of $\tau$ is the union of members of $\mathcal{B}$. A subset $\mathcal{S}$ of $\tau$ is said to be a* subbasis *for $\tau$ iff the set $\mathcal{B} = \{B | B$ is the intersection of finitely many members of $\mathcal{S}\}$ is a basis for $\tau$.*

We can derive new topologies based on known ones. Subspace topology and product topology are two important types of derived topology.

**Proposition 3.1.5** *Let $\langle X, \tau\rangle$ be a topological space, $X' \subseteq X$ and $\tau' = \{W | W = X' \cap U, U \in \tau\}$. The collection $\tau'$ is a topology on $X'$.*

We call $\tau'$ the *subspace topology* on $X'$, and $\langle X', \tau'\rangle$ a *subspace* of $\langle X, \tau\rangle$.

Let $\{\langle X_i, \tau_i\rangle\}_{i\in I}$ be a family of topological spaces and let $\times_I X_i$ be the product set of $\{X_i\}_{i\in I}$. Let $\mathcal{S} = \{\times_I V_i | V_i = X_i$ for all but one $i \in I$, and $V_i \in \tau_i$ for all $i \in I\}$. We call $\tau$ the *product topology* on $\times_I X_i$ iff $\mathcal{S}$ is a subbasis for $\tau$. We call $\langle \times_I X_i, \tau\rangle$ the *product space* of $\{\langle X_i, \tau_i\rangle\}_{i\in I}$.

If $X_i = X$ with the same topology for all $i \in I$, $\times_I X_i$ is denoted by $X^I$.

**Proposition 3.1.6** *Let $\{X_i\}_{i\in I}$ be a family of topological spaces and $J$ be an arbitrary index set. Then $(\times_I X_i)^J = \times_I X_i^J$.*

A *Hausdorff topologies* is one with the property that for any two points, there are disjoint neighborhoods. The trivial topology is non-Hausdorff and the discrete topology is Hausdorff.

In the next two sections, we will introduce two important types of topology that are between the two extremes: partial order topology and metric topology. We will see that partial order topologies in general are non-Hausdorff and metric topologies are Hausdorff.

### 3.1.2 Partial order

A set and a partial order relation on the set define a partially ordered set, or simply, a partial order.

**Definition 3.1.3 (Partial order)** *Let $A$ be a set. A binary relation $\leq_A \subseteq A \times A$ is called a* partial order relation *iff $\leq_A$ is reflexive, anti-symmetric and transitive. $\langle A, \leq_A\rangle$ is called a* partial order*; it is called a* linear order *iff, in addition, $\forall a_1, a_2 \in A$, either $a_1 \leq_A a_2$ or $a_2 \leq_A a_1$.*

For any partial order relation $\leq_A$, let $\geq_A$ denote the inverse of $\leq_A$, viz., $a_1 \geq_A a_2$ iff $a_2 \leq_A a_1$, and let $<_A$ ($>_A$) denote the strict relation of $\leq_A$ ($\geq_A$), viz., $a_1 <_A a_2$ ($a_1 >_A a_2$) iff $a_1 \leq_A a_2$ ($a_1 \geq_A a_2$) and $a_1 \neq a_2$. We will use $A$ to denote partial order $\langle A, \leq_A\rangle$ if no ambiguity arises.

**Definition 3.1.4 (Subpartial order)** *Let $\langle A, \leq_A\rangle$ be a partial order and $A' \subseteq A$. A partial order relation $\leq_{A'} \subseteq A' \times A'$ is called the* subpartial order relation *on $A'$ iff $a_1 \leq_{A'} a_2$ whenever $a_1 \leq_A a_2$. $\langle A', \leq_{A'}\rangle$ is called a* subpartial order *of $\langle A, \leq_A\rangle$.*

**Definition 3.1.5 (Product partial order)** *Let $\{A_i\}_{i \in I}$ be a set of partial orders and $A = \times_I A_i$. A partial order relation $\leq_A \subseteq A \times A$ is called the* product partial order relation *on $A$ iff $a \leq_A a'$ whenever $a_i \leq_{A_i} a'_i$ for all $i \in I$. $\langle A, \leq_A \rangle$ is called the* product partial order *of $\{\langle A_i, \leq_{A_i} \rangle\}_{i \in I}$.*

A partial order may have a least element and/or a greatest element.

**Definition 3.1.6 (Least (Greatest) element)** *Let $A$ be a partial order. An element $\bot_A$ ($\top_A$) $\in A$ is a* least (greatest) element *in $A$ iff it satisfies $\bot_A \leq_A a$ ($\top_A \geq_A a$) for every $a$ in $A$.*

It follows from the antisymmetry of $\leq_A$ that least (greatest) elements, if they exist, are unique.

Any set $A$ can be extended to a flat partial order by augmenting a least element $\bot_A \notin A$.

**Definition 3.1.7 (Flat partial order)** *A* flat *partial order, written $\overline{A}$, is a set $A$ augmented with a new element $\bot_A$, viz., $\overline{A} = A \cup \{\bot_A\}$ such that $a \leq_{\overline{A}} a'$ implies $a = a'$ or $a = \bot_A$.*

Element $\bot_A$ is the least element of $\overline{A}$. Usually $\bot_A$ means undefined in $A$. With this augmentation, any partial function to $A$ can be extended into a total function to $\overline{A}$, i.e., $f(a) = \bot_A$ if $f$ is not defined at $a$. In this thesis, functions mean total functions unless explicitly stated.

A subset of a partial order may have a least upper bound and/or a greatest lower bound.

**Definition 3.1.8 (Least upper (Greatest lower) bound)** *Let $A$ be a partial order, $D \subseteq A$ and $a \in A$. Then $a$ is an* upper (lower) bound *of $D$ iff $d \leq_A a$ ($d \geq_A a$) for every $d \in D$. Moreover, $a$ is a* least upper bound (lub) *(*greatest lower bound (glb)*) of $D$ iff*

1. *$a$ is an upper (lower) bound of $D$ and*
2. *if $d'$ is an upper (lower) bound of $D$ then $a \leq_A d'$ ($a \geq_A d'$).*

It follows from the antisymmetry of $\leq_A$ that least upper bounds (greatest lower bounds), if they exist, are unique. We use $\bigvee_A D$ ($\bigwedge_A D$) to denote the least upper (greatest lower) bound of $D$ in $A$, when it exists. We will drop the subscript $A$ if it is clear from context. If $A$ is the set of real numbers with arithmetic ordering, we use "sup" and "inf" to denote $\bigvee$ and $\bigwedge$, respectively. If $D$ is finite, we may use "max" and "min" to denote $\bigvee$ and $\bigwedge$, respectively.

One important kind of subset of a partial order is directed subset.

**Definition 3.1.9 (Directed subset)** *Let $A$ be a partial order and $D \subseteq A$. $D$ is* directed *iff $D \neq \emptyset$ and for all $d_1, d_2 \in D$, the set $\{d_1, d_2\}$ has an upper bound in $D$.*

A *chain* in a partial order $A$ is a linearly ordered subset of $A$. A chain is a directed subset.

One important kind of partial order is complete partial order.

**Definition 3.1.10 (Complete partial order (*cpo*))** *A partial order $A$ is* complete *iff:*

1. *it contains a least element, denoted $\perp_A$, and*
2. *every directed subset of $A$ has a least upper bound in $A$.*

Following are two propositions related to *cpo*s.

**Proposition 3.1.7** *A flat partial order is a cpo.*

**Proposition 3.1.8** *The product of cpos is a cpo. Let $\{A_i\}_{i \in I}$ be a set of cpos and $A = \times_I A_i$. The least element of $A$ is $\perp_A$ with $(\perp_A)_i = \perp_{A_i}, \forall i \in I$. Let $D$ be a directed subset of $A$. The least upper bound of $D$ is $\bigvee_A D$ with $(\bigvee_A D)_i = \bigvee_{A_i} D_i, \forall i \in I$, where $D_i$ is the projection of $D$ onto its ith component, i.e., $D_i = \Pi_i D$.*

A topology can be defined from a partial order.

**Definition 3.1.11 (Partial order topology)** *Let $A$ be a partial order. A subset $S$ of $A$ is* open *iff (1) $S$ is* upward closed, *i.e., $a \in S$ implies that $\forall a' \geq_A a$, $a' \in S$, and (2) $S$ is inaccessible from any directed subset $D$ of $A$, i.e., if $\bigvee_A D \in S$, then $\exists a \in D$, such that $a \in S$. This collection of open sets on $A$ forms the* partial order topology *of $A$.*

A partial order $\langle A, \leq_A \rangle$ is *non-trivial* iff there exist two elements $a, a'$ in $A$ such that $a <_A a'$.

**Proposition 3.1.9** *The partial order topology of a non-trivial partial order is non-Hausdorff.*

The following two propositions declare the properties of continuous functions in partial order topologies.

**Proposition 3.1.10** *Any continuous function is monotonic, i.e., if $f : A \to A'$ is continuous, then $a_1 \leq_A a_2$ implies $f(a_1) \leq_{A'} f(a_2)$.*

**Proposition 3.1.11** *Let $A$ and $A'$ be two cpos. Then $f : A \to A'$ is continuous iff for every directed subset $D \subseteq A$,*

1. *$f(D) = \{f(d) | d \in D\}$ is directed and*
2. *$f(\bigvee_A D) = \bigvee_{A'} f(D)$.*

### 3.1.3 Metric space

Metric topology is the most direct generalization of the topology used for real numbers in analysis.

**Definition 3.1.12 (Metric and Metric Space)** *Let $X$ be a set and $\mathcal{R}^+$ be the set of non-negative real numbers. A function $d : X \times X \to \mathcal{R}^+$ is a* metric *on $X$ iff*

- $d(x, y) = d(y, x)$.
- $d(x, y) \leq d(x, z) + d(z, y)$.
- $d(x, y) = 0$ *iff* $x = y$.

$\langle X, d\rangle$ *is called a* metric space.

Let $\langle X, d\rangle$ be a metric space, $x \in X$ and $\epsilon$ be a positive real number. The *spherical $\epsilon$-neighborhood* of $x$ is $\{x' | d(x', x) < \epsilon\}$, denoted $N^{\epsilon}(x)$.

**Definition 3.1.13 (Metric topology)** *The* metric topology *of a metric space is a topology with the set of spherical neighborhoods as a subbasis.*

**Proposition 3.1.12** *Metric topologies are Hausdorff.*

Another important concept used in analysis is measure. Let $X$ be a set. A family $\sigma$ of subsets of $X$ is a *$\sigma$-field* on $X$ iff it contains the empty set, the complement in $X$ of every element in $\sigma$ and the union of every denumerable subcollection. $\langle X, \sigma\rangle$ is called a *measurable space*.

**Definition 3.1.14 (Measure and Measure space)** *Let $\langle X, \sigma\rangle$ be a measurable space. A function $\mu : \sigma \to \mathcal{R}^+ \cup \{\infty\}$ is a* measure *iff $\mu(\emptyset) = 0$, and for any denumerable index set $J$ and any set of mutually disjoint elements $\{X_j\}_J$ of $\sigma$, $\mu(\cup_{j\in J} X_j) = \Sigma_{j\in J}\mu(X_j)$. $\langle X, \sigma, \mu\rangle$ is called a* measure space.

If $\langle X, \tau\rangle$ is a topological space, then the smallest $\sigma$-field containing $\tau$ is called the *Borel field of sets*, denoted $\Sigma_{Borel}(X)$. A measure defined on $\Sigma_{Borel}(X)$ is called a *Borel measure*.

Finishing up this section, we define the concept of limits. Given any linear order $L$ and topological space $X$, $v : L \to X$ is called a *linear set of values*. A limit of $v$ is defined as a generalization of a limit of a sequence.

**Definition 3.1.15 (Limit)** *Let $X$ be a topological space and $v : L \rightarrow X$ be a linear set of values. A point $v^* \in X$ is called a* limit *of $v$, written $v \rightarrow v^*$, iff for every neighborhood $N(v^*)$ of $v^*$, $\exists l_0$, $\forall l \geq_L l_0, v(l) \in N(v^*)$.*

If $L$ has a greatest element $l_0$, then $v \rightarrow v(l_0)$. Therefore, the concept of limits is also a generalization of the "final" value. We will use $\lim_{l \rightarrow \infty} v(l)$ to denote the limit of $v$ if it is unique.

One important property of Hausdorff topologies is the uniqueness of limits.

**Proposition 3.1.13** *If $X$ is of a Hausdorff topology and $v : L \rightarrow X$ is a linear set of values, then $v \rightarrow v_1^*$ and $v \rightarrow v_2^*$ imply $v_1^* = v_2^*$.*

One important property of product topologies is the pointwiseness of limits.

**Proposition 3.1.14** *If $\times_I X_i$ is of the product topology and $v : L \rightarrow \times_I X_i$ is a linear set of values, then $v \rightarrow v^*$ iff $v_i \rightarrow v_i^*$ for all $i \in I$.*

## 3.2 Time Structures

Understanding time is the key to understanding dynamics. We formalize time using an abstract structure that captures its important aspects. A time structure, in general, can be considered as a linearly ordered set with a start time point, an associated metric for "the distance between any two time points" and a measure for "the duration of an interval of time."

**Definition 3.2.1 (Time structure)** *A* time structure *is a triple $\langle \mathcal{T}, d, \mu \rangle$ where*

- *$\mathcal{T}$ is a linearly ordered set $\langle \mathcal{T}, \leq \rangle$ with $\mathbf{0}$ as the least element;*
- *$\langle \mathcal{T}, d \rangle$ forms a metric space with $d$ as a metric satisfying: for all $t_0 \leq t_1 \leq t_2$,*

$$d(t_0, t_2) = d(t_0, t_1) + d(t_1, t_2),$$

  *$\{t | m(t) \leq \tau\}$ has a greatest element and $\{t | m(t) \geq \tau\}$ has a least element for all $0 \leq \tau < \sup\{m(t) | t \in \mathcal{T}\}$ where $m(t) = d(\mathbf{0}, t)$;*
- *$\langle \mathcal{T}, \sigma, \mu \rangle$ forms a measure space with $\sigma$ as the Borel set of topological space $\langle \mathcal{T}, d \rangle$ and $\mu$ as a Borel measure satisfying $\mu([t_1, t_2)) \leq d(t_1, t_2)$ for all $t_1 \leq t_2$ where $[t_1, t_2) = \{t | t_1 \leq t < t_2\}$ and $\mu([t_1, t_2)) = \mu([\mathbf{0}, t_2)) - \mu([\mathbf{0}, t_1))$.*

For simplicity, we will use $\mathcal{T}$ to refer to time structure $\langle \mathcal{T}, d, \mu \rangle$ when no ambiguity arises. For most applications, we have $\mu([t_1, t_2)) = d(t_1, t_2)$. However, if $\mathcal{T}$ is an abstraction of another time structure, it is possible that $\exists t_1, t_2, \mu([t_1, t_2)) < d(t_1, t_2)$. Discussions on time abstraction will be found in Chapter 6, Behavior Analysis.

A time structure $\mathcal{T}$ is *infinite* iff $\mathcal{T}$ has no greatest element and $\mu(\mathcal{T}) = \infty$. $\mathcal{T}$ is *discrete* iff its metric topology is discrete. $\mathcal{T}$ is *continuous* iff its metric space is connected.

For example, the set of natural numbers $\mathcal{N}$ and the set of nonnegative real numbers $\mathcal{R}^+$, with $d(t_1, t_2) = |t_1 - t_2|$ and $\mu([0, t)) = t$, are time structures. $\mathcal{N}$ is discrete and $\mathcal{R}^+$ is continuous. The set $\{1 - \frac{1}{2^n} | n \in \mathcal{N}\}$ with the metric $d$ and the measure $\mu$ also defines a discrete time structure. However, the sets $\{1 - \frac{1}{2^n} | n \in \mathcal{N}\} \cup \{1\}$, $\{0\} \cup \{\frac{1}{2^n} | n \in \mathcal{N}\}$ and $[0, 1] \cup [2, 3]$ with the metric $d$ and the measure $\mu$ form time structures neither discrete nor continuous. The set of rational numbers $\mathcal{Q}$ with the metric $d$ and the measure $\mu$ does not form a time structure.

**Proposition 3.2.1** *(1) For any time structure $\mathcal{T}$, if $T \subset \mathcal{T}$ has an upper bound in $\mathcal{T}$, $T$ has a least upper bound in $\mathcal{T}$.*
*(2) The following properties for a time structure are equivalent:*

- *(a) $\mathcal{T}$ is discrete.*
- *(b) Let $(t_1, t_2) = \{t | t_1 < t < t_2\}$. For all $t$, if $t$ is not the least element of $\mathcal{T}$, then $\exists t' < t$, denoted $pre(t)$, such that $(t', t) = \emptyset$, and for all $t$, if $t$ is not the greatest element of $\mathcal{T}$, then $\exists t' > t$, denoted $suc(t)$, such that $(t, t') = \emptyset$.*
- *(c) $\mathcal{T}$ is* well-founded*, i.e., $\forall t \in \mathcal{T}$, $[\mathbf{0}, t)$ is finite.*

*(3) The following properties for a time structure are equivalent:*

- *(a) $\mathcal{T}$ is continuous.*
- *(b) $\mathcal{T}$ is* dense*, i.e., for all $t_1 < t_2$, there exists $t_0$ such that $t_1 < t_0 < t_2$.*

Intuitively, discrete time is isomorphic to an ordered subset of natural numbers and continuous time is isomorphic to a left-closed interval of a real line. Even though our definition of time structures is general, discrete and continuous time structures are most commonly used.

A time structure $\langle \mathcal{T}, d, \mu \rangle$ may be related to another time structure $\langle \mathcal{T}_r, d_r, \mu_r \rangle$, where $\langle \mathcal{T}_r, \leq_r \rangle$ is a linear order with $\mathbf{0}_r$ as the least element, by a *reference time mapping* $h : \mathcal{T} \to \mathcal{T}_r$ satisfying

- the order among time points is preserved: $t < t'$ implies $h(t) <_r h(t')$,

- the least element is preserved: $h(\mathbf{0}) = \mathbf{0}_r$,
- the distance between two time points is preserved: $d(t_1, t_2) = d_r(h(t_1), h(t_2))$, and
- the measure on any finite time interval is preserved: $\mu([\mathbf{0}, t)) = \mu_r([\mathbf{0}_r, h(t)))$.

$\mathcal{T}_r$ is called a *reference time* of $\mathcal{T}$, and $\mathcal{T}$ is called a *sample time* of $\mathcal{T}_r$. For example, if $h : \mathcal{N} \to \mathcal{R}^+$ is defined as $h(n) = n$, $\mathcal{R}^+$ is a reference time of $\mathcal{N}$. For any time structure $\mathcal{T}$, a reference time of $\mathcal{T}$ is as "dense" as $\mathcal{T}$. Furthermore, the reference relation is transitive:

**Proposition 3.2.2** *If $\mathcal{T}_0$ is a reference time of $\mathcal{T}_1$ and $\mathcal{T}_1$ is a reference time of $\mathcal{T}_2$, then $\mathcal{T}_0$ is a reference time of $\mathcal{T}_2$.*

## 3.3 Domain Structures

As with time, we formalize domains as abstract structures so that discrete and continuous domains are defined uniformly. A domain can be either simple or composite. Simple domains denote simple data types, such as reals, integers, Booleans and characters; composite domains denote structured data types, such as arrays, vectors, strings, objects, structures and records.

**Definition 3.3.1 (Simple domain)** *A* simple domain *is a pair $\langle A \cup \{\perp_A\}, d_A \rangle$ where $A$ is a set, $\perp_A \notin A$ means undefined in $A$, and $d_A$ is a metric on $A$.*

Let $\overline{A} = A \cup \{\perp_A\}$. For simplicity, we will use $\overline{A}$ to refer to simple domain $\langle \overline{A}, d_A \rangle$ when no ambiguity arises. For example, let $\mathcal{R}$ be the set of real numbers, $\overline{\mathcal{R}}$ is a simple domain with a connected metric space; let $\mathcal{B} = \{0, 1\}$, $\overline{\mathcal{B}}$ is a simple domain with a discrete topology on $\mathcal{B}$.

Any simple domain $\overline{A}$ is associated with a partial order relation $\leq_{\overline{A}}$. $\langle \overline{A}, \leq_{\overline{A}} \rangle$ is a flat partial order with $\perp_A$ as the least element. In addition, $\overline{A}$ is associated with a *derived metric topology* $\tau = \tau_A \cup \{\overline{A}\}$ where $\tau_A$ is the metric topology on $A$ derived from the metric $d_A$.

**Proposition 3.3.1** *$\{\perp_A\}$ is not $\tau$-open. The only neighborhood of $\perp_A$ is $\overline{A}$.*

A simple domain $\langle A, d_A \rangle$ can also be represented as a triple $\langle \overline{A}, \leq_{\overline{A}}, \tau \rangle$ where $\leq_{\overline{A}}$ is the partial order relation and $\tau$ is the derived metric topology.

A domain is defined recursively based on simple domains.

**Definition 3.3.2 (Domain)** *$\langle A, \leq_A, \tau \rangle$, with $\leq_A$ as the partial order relation and $\tau$ as the derived metric topology, is a* domain *iff:*

- *it is a simple domain; or*
- *it is a* composite *domain, i.e., it is the product of a family of domains* $\{\langle A_i, \leq_{A_i}, \tau_i\rangle\}_{i\in I}$ *such that* $\langle A, \leq_A\rangle$ *is the product partial order of the family of partial orders* $\{\langle A_i, \leq_{A_i}\rangle\}_{i\in I}$ *and* $\langle A, \tau\rangle$ *is the product space of the family of topological spaces* $\{\langle A_i, \tau_i\rangle\}_{i\in I}$.

Note that there is no restriction on the index set $I$, which can be arbitrary (finite or infinite, countable or uncountable). For simplicity, we will use $A$ to refer to domain $\langle A, \leq_A, \tau\rangle$ when no ambiguity arises. For example, let $n$ be a natural number, then $\overline{\mathcal{R}}^n$ is a composite domain with $n$ components; let $\mathcal{N}$ be the set of natural numbers, then $\mathcal{N} \to \overline{\mathcal{B}}$ (or equivalently, $\overline{\mathcal{B}}^{\mathcal{N}}$) is a composite domain with infinitely many components.

Given a simple domain $\overline{A}$, a value $a \in \overline{A}$ is *well-defined* iff $a \neq \perp_A$. Given a composite domain $\times_I A_i$, a value $a \in \times_I A_i$ is *well-defined* iff $a_i$ is well-defined for all $i \in I$. A value in a domain is *undefined* iff it is the least element of the domain.

Intuitively, for any domain, its partial order topology characterizes the information (or definedness) hierarchies of data and its derived metric topology characterizes the limit properties of data.

**Proposition 3.3.2** *For any domain, its partial order topology is finer than its derived metric topology, and both are non-Hausdorff.*

A signature is a syntactical structure of a multi-sorted set of data with associated functions.

**Definition 3.3.3 (Signature)** *Let* $\Sigma = \langle S, F\rangle$ *be a* signature *where* $S$ *is a set of* sorts *and* $F$ *is a set of* function symbols. $F$ *is equipped with a mapping* type: $F \to S^* \times S$ *where* $S^*$ *denotes the set of all finite tuples of* $S$. *For any* $f \in F$, *type*$(f)$ *is the type of* $f$. *We use* $f : s^* \to s$ *to denote* $f \in F$ *with type*$(f) = \langle s^*, s\rangle$.

For example, the signature of Boolean algebra can be described as: $\Sigma_b = \langle\{b\}, \{\mathsf{0}, \neg, \wedge, \vee\}\rangle$ with $\mathsf{0} :\to b$, $\neg : b \to b$, $\wedge : b, b \to b$, and $\vee : b, b \to b$. $\Sigma_b$ has one sort with a constant $\mathsf{0}$ (nullary function), a unary function $\neg$, and two binary functions $\wedge$ and $\vee$.

A domain structure of some signature is defined as follows.

**Definition 3.3.4 ($\Sigma$-domain structure)** *Let* $\Sigma = \langle S, F\rangle$ *be a signature. A* $\Sigma$-domain structure $A$ *is a pair* $\langle\{A_s\}_{s\in S}, \{f^A\}_{f\in F}\rangle$ *where for each* $s \in S$, $A_s$ *is a domain of sort* $s$, *and for each* $f : s^* \to s \in F$ *with* $s^* : I \to S$ *and* $s \in S$, $f^A : \times_I A_{s^*_i} \to A_s$ *is a function denoted by* $f$, *which is continuous in the partial order topology.*

To be continuous on a domain in its partial order topology is not a real restriction on a function. Strict functions are continuous functions in partial order topologies. A function is *strict w.r.t. an argument* iff its output is undefined whenever its input of that argument is undefined. A function is *strict* iff it is strict w.r.t. all of its arguments.

Given any partial or total function $f : \times_I A_i \to A$, a continuous function $\bar{f} : \times_I \overline{A_i} \to \overline{A}$ can be defined as:

$$\bar{f}(a) = \begin{cases} f(a) & \text{if } a \in \times_I A_i \text{ and } f(a) \text{ is defined} \\ \perp_A & \text{otherwise.} \end{cases}$$

We call $\bar{f}$ a *strict extension* of function $f$. We will also use $f$ to denote its strict extension if no ambiguity arises. For example, let $\Sigma_r = \langle\{r\}, \{\mathtt{0}, +, \cdot\}\rangle$ with $\mathtt{0} :\to r$, $+ : r, r \to r$ and $\cdot : r, r \to r$. Then $\langle\{\overline{\mathcal{R}}\}, \{0, +, \cdot\}\rangle$ is a $\Sigma_r$-domain structure, where $+$ and $\cdot$ are strict extensions of addition and multiplication on $\mathcal{R}$, respectively.

However, not every extension of a function that is continuous should also be strict. For example, $\langle\{\overline{\mathcal{B}}\}, \{0, \neg, \wedge, \vee\}\rangle$ is a $\Sigma_b$-domain structure where $\neg$, $\wedge$ and $\vee$ are negation, conjunction and disjunction, respectively. Function $\vee : \overline{\mathcal{B}} \times \overline{\mathcal{B}} \to \overline{\mathcal{B}}$ is continuous but not strict, since $\vee$ is an "or" logic satisfying $1 \vee x = 1$ for all $x \in \overline{\mathcal{B}}$, thus, $1 \vee \perp_{\mathcal{B}} \neq \perp_{\mathcal{B}}$.

The following propositions characterize the general properties of continuous functions on simple domains.

**Proposition 3.3.3** *(1) Function $f : \overline{A} \to \overline{A'}$ is continuous in the partial order topology iff $f$ is strict or constant. (2) If $f : \overline{A} \to \overline{A'}$ is continuous in the derived metric topology, then $f$ is continuous in the partial order topology. (3) Function $f : \overline{A} \to \overline{A'}$ is continuous in the derived metric topology iff $f$ is continuous in the partial order topology and the restriction of $f$ on $A$ and $A'$ is continuous in the metric topology, namely, for any open subset $S$ of $A'$, $f^{-1}(S) \cap A$ is open.*

The properties of continuous functions in partial order topologies can be generalized to composite domains. A function $f : \times_I A_i \to A$ is *continuous w.r.t. an argument* $j$, iff function $\lambda a_j.f(a, a_j)$[1] is continuous for all $a \in \times_{I-\{j\}} A_i$.

**Proposition 3.3.4** *Let $I$ be a finite index set. (1) Function $f : \times_I A_i \to A$ is continuous in the partial order topology iff $f$ is continuous w.r.t. all $i \in I$. (2) If $f : \times_I \overline{A_i} \to \overline{A}$ is continuous in the derived metric topology, then $f$ is continuous in the partial order topology. (3) Function $f : \times_I \overline{A_i} \to \overline{A}$ is continuous in the derived metric topology iff $f$ is continuous in the partial*

[1] $\lambda x.expr(x)$ is a lambda expression of a function $f$, equivalent to $\forall x, f(x) = expr(x)$.

*order topology and the restriction of $f$ on $\times_I A_i$ and $A$ is continuous in the product metric topology, namely, for any open subset $S$ of $A$, $f^{-1}(S) \cap \times_I A_i$ is open.*

A function is *well-defined* iff its output is well-defined whenever its input is well-defined. Both well-definedness and strictness are closed under functional composition, and a function can be both well-defined and strict.

For example, a widely used *conditional function*, $cond : \overline{A} \times \overline{A} \times \overline{A'} \times \overline{A'} \to \overline{A'}$, is defined as follows:

$$cond(x, y, u, v) = \begin{cases} \perp_{A'} & \text{if } x = \perp_A \text{ or } y = \perp_A \\ u & \text{else if } x = y \\ v & \text{otherwise.} \end{cases} \tag{3.1}$$

Function *cond* is continuous in the partial order topology; it is continuous in the derived metric topology if $A$ is of a discrete topology. Furthermore, it is well-defined and strict w.r.t. arguments $x$ and $y$.

## 3.4 Traces and Events

Intuitively, a trace denotes changes of values over time. Formally, a mapping $v : \mathcal{T} \to A$ from time $\mathcal{T}$ to domain $A$ is called a *trace*. A trace $v$ is *well-defined* iff $v(t)$ is well-defined for all $t \in \mathcal{T}$. For example if $\mathcal{T} = \mathcal{R}^+$ and $A = \overline{\mathcal{R}}$, $v_1 = \lambda t. \sin(t)$ and $v_2 = \lambda t. e^{-t}$ are well-defined traces. A trace $v$ is *undefined* iff $v(t)$ is undefined for all $t \in \mathcal{T}$.

A trace provides complete information at every (finite) time point. Values at infinite time points are not represented explicitly, they can, however, be derived when limits are introduced. For example, $\lim_{t\to\infty} \sin(t) = \perp_{\mathcal{R}}$ and $\lim_{t\to\infty} e^{-t} = 0$.

Let $A$ be a domain and $v : L \to A$ be a linear set of values. A value $v^* \in A$ is a *limit* of $v$, written $v \to v^*$, iff $v^*$ is a limit of $v$ in the derived metric topology of $A$. In the rest of this thesis, limits defined on a domain will mean those in its derived metric topology. Limits of $v$ may not be unique. However, the set of limits of $v$ has the following properties.

**Proposition 3.4.1** *Let $v : L \to \overline{A}$ be a linear set of values. Then*

*(1) $v \to \perp_A$, and*

*(2) $v \to v_1^*$ and $v \to v_2^*$ imply that either $v_1^* = v_2^*$ or one of $v_1^*$ and $v_2^*$ is $\perp_A$.*

**Proposition 3.4.2** *Let $v : L \to A$ for $A = \times_I A_i$. Then*

*(1) $v \to v^*$ iff $v_i \to v_i^*$ for all $i \in I$, and*

*(2) the set of limits $\{v^* | v \to v^*\}$ is a directed subset in $\langle A, \leq_A \rangle$ and has a greatest element.*

The *greatest limit* of $v$, written $\lim v$, is defined as the greatest element of the set of limits of $v$, i.e., $\lim v = \bigvee_A \{v^* | v \to v^*\}$. Note that the greatest limit of a linear set of values always exists and is unique. We will call the greatest limit simply the limit if no ambiguity arises.

The following two propositions capture two important properties of the limits.

**Proposition 3.4.3** *Let $v : L \to A$ for $A = \times_I A_i$. Then $(\lim v)_i = \lim v_i, \forall i \in I$.*

**Proposition 3.4.4** *If $v_1, v_2 : L \to A$ and $v_1(l) \leq_A v_2(l)$ for all $l \in L$, then $\lim v_1 \leq_A \lim v_2$.*

Proposition 3.4.3 characterizes the composite property of the limits. Proposition 3.4.4 characterizes the monotonic property of the limits.

Using the concept of the limits, we can complete a trace with its values at limit time points. Given a time structure $\mathcal{T}$, let $\mathcal{T}^\infty$ be the set of *downward closed* intervals, i.e., for any $T \in \mathcal{T}^\infty$, (1) $T \neq \emptyset$ and (2) $t \in T$ implies that for all $t' \leq t$, $t' \in T$. A trace $v : \mathcal{T} \to A$ can be extended to its *completion* $v^\infty : \mathcal{T}^\infty \to A$ as $v^\infty(T) = \lim v_{|T}$ where $v_{|T}$ denotes the restriction of $v$ onto $T$. If $T$ has a greatest element $t_0$, then $v^\infty(T) = v(t_0)$. A trace completion provides values at infinite as well as at finite time points. Note that $\mathcal{T} \in \mathcal{T}^\infty$, for any trace $v : \mathcal{T} \to A$, $v^\infty(\mathcal{T}) = \lim v$ can be considered as the "final" value. For simplicity, we will use $v$ to refer to both $v$ and its completion $v^\infty$ when no ambiguity arises.

Let $T_{<t} = \{t' | t' < t\}$. Then $T_{<t} \in \mathcal{T}^\infty$ whenever $t > \mathbf{0}$. We use $pre(t)$ to denote both $T_{<t}$ and the greatest element of $T_{<t}$, if it exists.

Let $T_{\leq t-\tau} = \{t' | t' < t, d(t, t') \geq \tau\}$ for $\tau > 0$. Then $T_{\leq t-\tau} \in \mathcal{T}^\infty$ whenever $m(t) \geq \tau$.

**Proposition 3.4.5** *For any time structure $\mathcal{T}$, $T_{\leq t-\tau}$ has a greatest element whenever $m(t) \geq \tau$.*

We use $t - \tau$ to denote the greatest element of $T_{\leq t-\tau}$ when $m(t) \geq \tau$.

The set of all possible traces from a time structure to a domain, associated with a partial order relation and a derived metric topology, forms a trace space.

**Definition 3.4.1 (Trace space)** *Given a time structure $\mathcal{T}$ and a domain $\langle A, \leq_A, \tau \rangle$, the* trace space *is a triple $\langle A^\mathcal{T}, \leq_{A^\mathcal{T}}, \Gamma \rangle$ where $A^\mathcal{T}$ is the product set (the set of* all *functions from $\mathcal{T}$ to $A$), $\leq_{A^\mathcal{T}}$ is the product partial order relation constructed from the partial order relation $\leq_A$, and $\Gamma$ is the product topology constructed from the derived metric topology $\tau$.*

For simplicity, we will use $A^\mathcal{T}$ to refer to trace space $\langle A^\mathcal{T}, \leq_{A^\mathcal{T}}, \Gamma \rangle$ when no ambiguity arises.

A trace space is essentially a composite domain. Therefore, limits of a linear set of traces can be defined accordingly. Given a linear set of traces $V : L \to A^\mathcal{T}$, limits and the greatest limit of $V$ are defined as follows. A trace $V^* \in A^{\mathcal{T}}$ is a *limit* of $V$, written $V \to V^*$, iff $V^*$ is a limit of $V$ in the derived metric topology of $A^{\mathcal{T}}$. Similar to the properties of limits of a linear set of values, the properties of limits of a linear set of traces are as follows.

**Proposition 3.4.6** *Let $V : L \to A^{\mathcal{T}}$ for a linear order $L$ and a trace space $A^{\mathcal{T}}$. Then*

*(1) $V \to V^*$ iff $V(t) \to V^*(t)$ for all $t \in \mathcal{T}$, and*

*(2) the set of limits $\{V^* | V \to V^*\}$ is a directed subset in $\langle A^{\mathcal{T}}, \leq_{A^{\mathcal{T}}}\rangle$ and has a greatest element.*

The *greatest limit* of $V$, written $\lim V$, is defined as the greatest element of the set of limits of $V$, $\lim V = \bigvee_{A^{\mathcal{T}}}\{V^* | V \to V^*\}$. We will call the greatest limit simply the limit if no ambiguity arises. Furthermore, the composite property of the limits holds as well.

**Proposition 3.4.7** *Let $V : L \to A^{\mathcal{T}}$. Then* $(\lim V)(t) = \lim V(t), \forall t \in \mathcal{T}$.

The concept of the limit of a linear set of traces will be used when we introduce limiting semantics in the next chapter.

A nonintermittent trace is a special type of trace defined as follows. A trace $v : \mathcal{T} \to \overline{A}$ is *nonintermittent* iff for any $T \in \mathcal{T}^\infty$, $v(T) = \perp_A$ implies that $\forall T' \supset T$, $v(T') = \perp_A$. A trace $v : \mathcal{T} \to \times_I A_i$ is *nonintermittent* iff $v_i$ is nonintermittent for all $i \in I$.

A right-continuous trace is a special type of trace defined as follows. A trace $v : \mathcal{T} \to A$ is *right-continuous* at $t_0$ iff $\forall t > t_0, t \to t_0$ implies $v(t) \to v(t_0)$; $v$ is *right-continuous* iff it is right-continuous at all $t \in \mathcal{T}$. A discrete-time trace is always right-continuous according to this definition.

An *event trace* is a nonintermittent and right-continuous trace whose domain is $\overline{\mathcal{B}}$. An event trace $e : \mathcal{T} \to \overline{\mathcal{B}}$ with $e \neq \lambda t. \perp_{\mathcal{B}}$ generates a structure $\langle \mathcal{T}_e, d_e, \mu_e\rangle$ from $\langle \mathcal{T}, d, \mu\rangle$ where:

- $\mathcal{T}_e \subseteq \mathcal{T}$ is defined as $\mathcal{T}_e = \{\mathbf{0}\} \cup \{t > \mathbf{0} | e(t) \neq \perp_{\mathcal{B}}, e(t) \neq e(pre(t))\}$,
- $d_e = d_{|\mathcal{T}_e \times \mathcal{T}_e}$,
- $\forall t \in \mathcal{T}_e$, $\mu_e([\mathbf{0}, t)) = \mu([\mathbf{0}, t))$, and $\mu_e(\mathcal{T}_e) = \mu(T)$ for $T = \{t | e(t) \neq \perp_{\mathcal{B}}\}$.

**Proposition 3.4.8** *For any time structure $\mathcal{T}$ and any event trace $e$, $\langle \mathcal{T}_e, d_e, \mu_e\rangle$ is a discrete sample time structure of $\mathcal{T}$.*

For any event-based time, each transition point of the event trace defines a time point (Figure 3.1).

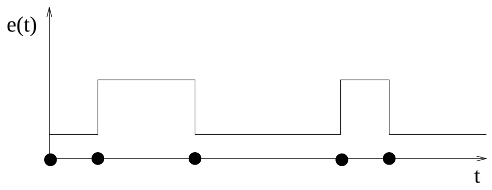


Figure 3.1: An event trace: each dot depicts a time point

The set of all possible event traces on a reference time structure, associated with a partial order relation and a derived metric topology, forms an event space.

**Definition 3.4.2 (Event space)** *An* event space *is a triple* $\langle \mathcal{E}^{\mathcal{T}}, \leq_{\mathcal{E}^{\mathcal{T}}}, \Gamma' \rangle$ *where* $\mathcal{T}$ *is a time structure,* $\mathcal{E}^{\mathcal{T}} \subset \overline{\mathcal{B}}^{\mathcal{T}}$ *is the set of all event traces on* $\mathcal{T}$*,* $\leq_{\mathcal{E}^{\mathcal{T}}}$ *is the sub partial order relation of* $\leq_{\overline{\mathcal{B}}^{\mathcal{T}}}$*, and* $\Gamma'$ *is the subspace topology of* $\Gamma$ *that is the derived metric topology of* $\overline{\mathcal{B}}^{\mathcal{T}}$*.*

## 3.5 Transductions

Transductions are mathematical models of general transformational processes. In this section, we first define general concepts of transductions, then discuss two types of basic transduction: transliterations and delays. Finally, we introduce event-driven transductions for constructing systems with components of different time structures.

### 3.5.1 General concepts

A transduction is a mapping from input traces to output traces that satisfies the causal relationship between its inputs and outputs, i.e., the output value at any time depends only on inputs up to that time. Formally, causality can be defined as follows.

**Definition 3.5.1 (Causality and Transduction)** *Given* $v_1, v_2 \in A^{\mathcal{T}}$ *and* $\tau \in \mathcal{R}^+$*,* $v_1$ *and* $v_2$ *are* coincident *up to* $\tau$ *iff* $\forall t, m(t) \leq \tau, v_1(t) = v_2(t)$*. A mapping* $F : A^{\mathcal{T}} \to A'^{\mathcal{T}'}$ *from a trace space to a trace space is* causal *iff for any* $t' \in \mathcal{T}'$*,* $F(v_1)(t') = F(v_2)(t')$ *whenever* $v_1$ *and* $v_2$ *are coincident up to* $m'(t')$*. A causal mapping on trace spaces is called a* transduction.

For instance, a state automaton with an initial state defines a transduction on a discrete time structure; a temporal integration with a given initial value is a typical transduction on a continuous time structure. Just as nullary functions represent constants, nullary transductions represent traces. Transductions are closed under functional composition.

We characterize two classes of transduction: primitive transductions and event-driven transductions.

### 3.5.2 Primitive transductions

Primitive transductions are defined on a generic time structure $\mathcal{T}$. *Primitive transductions* are functional compositions of two types of *basic* transduction: transliterations and delays.

**Definition 3.5.2 (Transliteration)** *A* transliteration *is a pointwise extension of a function. Formally, let $f : A \to A'$ be a function and $\mathcal{T}$ be a time structure. The pointwise extension of $f$ onto $\mathcal{T}$ is a mapping $f_{\mathcal{T}} : A^{\mathcal{T}} \to A'^{\mathcal{T}}$ satisfying $f_{\mathcal{T}}(v) = \lambda t.f(v(t))$.*

By this definition, $(f \circ g)_{\mathcal{T}} = f_{\mathcal{T}} \circ g_{\mathcal{T}}$. We will also use $f$ to denote transliteration $f_{\mathcal{T}}$ if no ambiguity arises.

Intuitively, a transliteration is a transformational process without memory or internal state, such as a combinational circuit. For example, let $\oplus : \overline{\mathcal{B}} \times \overline{\mathcal{B}} \to \overline{\mathcal{B}}$ be a function defined as $x \oplus y = (\neg x) \wedge y \vee x \wedge (\neg y)$, i.e., an "exclusive or". Then a pointwise extension of $\oplus$ is a transliteration, functioning as the basic "or" logic in asynchronous event control [Sut89] (Figure 3.2). We will discuss more on event logics in Chapter 5, Modeling in Constraint Nets.

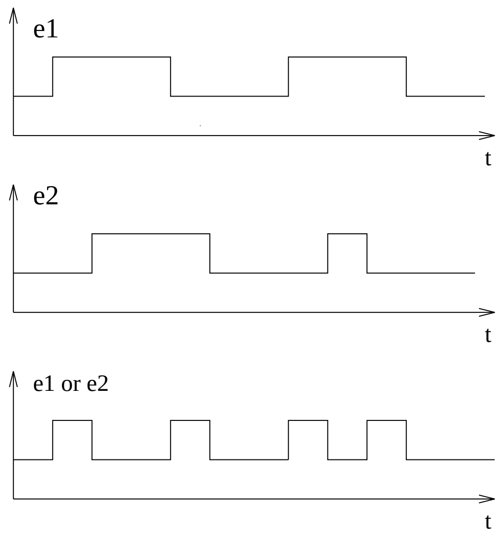


Figure 3.2: Event logic for "or"

There are two types of delay: unit delays and transport delays.

**Definition 3.5.3 (Unit delay)** *Let $A$ be a domain, $v_0$ a well-defined value in $A$, and $\mathcal{T}$ a time structure. A* unit delay *$\delta^A_{\mathcal{T}}(v_0) : A^{\mathcal{T}} \to A^{\mathcal{T}}$ is a transduction defined as*

$$\delta^A_{\mathcal{T}}(v_0)(v) = \lambda t. \begin{cases} v_0 & \textit{if } t = \mathbf{0} \\ v(pre(t)) & \textit{otherwise} \end{cases}$$

*where $v_0$ is called the* initial output value *of the unit delay.*

A unit delay $\delta^A_{\mathcal{T}}(v_0)$ acts as a unit memory for data in domain $A$, given a discrete time structure. We will use $\delta(v_0)$ to denote unit delay $\delta^A_{\mathcal{T}}(v_0)$ if no ambiguity arises.

Unit delays may not be meaningful for non-discrete time structures.

**Definition 3.5.4 (Transport delay)** *Let $A$ be a domain, $v_0$ a well-defined value in $A$, $\mathcal{T}$ a time structure and $\tau > 0$. A* transport delay *$\Delta^A_{\mathcal{T}}(\tau)(v_0) : A^{\mathcal{T}} \to A^{\mathcal{T}}$ is a transduction defined as*

$$\Delta^A_{\mathcal{T}}(\tau)(v_0)(v) = \lambda t. \begin{cases} v_0 & \textit{if } m(t) < \tau \\ v(t-\tau) & \textit{otherwise} \end{cases}$$

*where $v_0$ is called the* initial output value *of the transport delay and $\tau$ is called the* time delay.

We will use $\Delta(\tau)(v_0)$ to denote transport delay $\Delta^A_{\mathcal{T}}(\tau)(v_0)$ if no ambiguity arises. Transport delays are essential for modeling sequential behaviors in dynamic systems.

### 3.5.3 Event-driven transductions

A primitive transduction maps traces to traces with the same time structure. A hybrid system consists of components of different time structures. In this section, we consider event-driven transductions, which are an important component of our model.

We define sample and extension traces as follows. Let $\mathcal{T}_r$ be a reference time of $\mathcal{T}$ with a reference time mapping $h$. The *sample trace* of $v : \mathcal{T}_r \to A$ onto $\mathcal{T}$ is a trace $\underline{v} : \mathcal{T} \to A$ satisfying

$$\underline{v} = \lambda t.v(h(t)).$$

The *extension trace* of $v : \mathcal{T} \to A$ onto $\mathcal{T}_r$ is a trace $\overline{v} : \mathcal{T}_r \to A$ satisfying

$$\overline{v} = \lambda t_r. \begin{cases} v(h^{-1}(t_r)) & \text{if } \exists t \in \mathcal{T}, \mu_r([\mathbf{0}_r, t_r)) \le \mu([\mathbf{0}, t)) \text{ or } \mu_r([\mathbf{0}_r, t_r)) < \mu(\mathcal{T}) \\ \perp_A & \text{otherwise} \end{cases}$$

where $h^{-1}(t_r) = \{t | h(t) \le_r t_r\} \in \mathcal{T}^\infty$.

*Sampling* is a type of transduction whose output is a sample trace of its input. *Extending* is a type of transduction whose output is an extension trace of its input.

An event-driven transduction is a primitive transduction augmented with an extra input which is an event trace; it operates at each event point and the output value holds between two events. The additional event trace input of an event-driven transduction is called the *clock* of the transduction. Intuitively, an event-driven transduction works as follows. First, the input trace with the reference time $\mathcal{T}$ is sampled onto the sample time $\mathcal{T}_e$ generated by the event trace $e$. Then, the primitive transduction is performed on $\mathcal{T}_e$. Finally, the output trace is extended from $\mathcal{T}_e$ back to $\mathcal{T}$.

**Definition 3.5.5 (Event-driven transduction)** *Let $\mathcal{T}$ be a time structure and $F_{\mathcal{T}} : A^{\mathcal{T}} \to A'^{\mathcal{T}}$ a primitive transduction. Let $\mathcal{E}^{\mathcal{T}}$ be the set of all event traces on time structure $\mathcal{T}$. The* event-driven transduction *of $F$ is a mapping $F^{\circ}_{\mathcal{T}} : \mathcal{E}^{\mathcal{T}} \times A^{\mathcal{T}} \to A'^{\mathcal{T}}$ satisfying:*

$$F^{\circ}_{\mathcal{T}}(e, v) = \begin{cases} \lambda t.\ \perp_{A'} & \text{if } e = \lambda t.\ \perp_{\mathcal{B}} \\ \overline{F_{\mathcal{T}_e}(\underline{v})} & \text{otherwise.} \end{cases}$$

We will use $F^{\circ}$ to denote event-driven transduction $F^{\circ}_{\mathcal{T}}$ if no ambiguity arises.

## 3.6 Dynamics Structures

With preliminaries established, we define an abstract structure of dynamics.

**Definition 3.6.1 ($\Sigma$-dynamics structure)** *Let $\Sigma = \langle S, F\rangle$ be a signature. Given a $\Sigma$-domain structure $A$ and a time structure $\mathcal{T}$, a* $\Sigma$-dynamics structure *$\mathcal{D}(\mathcal{T}, A)$ is pair $\langle \mathcal{V}, \mathcal{F}\rangle$ such that*

- *$\mathcal{V} = \{A_s^{\mathcal{T}}\}_{s\in S} \cup \mathcal{E}^{\mathcal{T}}$ where $A_s^{\mathcal{T}}$ is a trace space of sort $s$ and $\mathcal{E}^{\mathcal{T}}$ is the event space;*
- *$\mathcal{F} = \mathcal{F}_{\mathcal{T}} \cup \mathcal{F}^{\circ}_{\mathcal{T}}$ where $\mathcal{F}_{\mathcal{T}}$ is the set of basic transductions, including the set of transliterations $\{f^A_{\mathcal{T}}\}_{f\in F}$, the set of unit delays $\{\delta^{A_s}_{\mathcal{T}}(v_s)\}_{s\in S, v_s\in A_s}$, and the set of transport delays $\{\Delta^{A_s}_{\mathcal{T}}(\tau)(v_s)\}_{s\in S, \tau>0, v_s\in A_s}$, $\mathcal{F}^{\circ}_{\mathcal{T}}$ is the set of event-driven transductions derived from the set of basic transductions, i.e., $\{F^{\circ} | F \in \mathcal{F}_{\mathcal{T}}\}$.*

Finishing up this chapter, let us explore the properties of dynamics structures.

The following propositions establish the fact that the partial order of a trace space and the partial order of an event space are *cpo*s.

**Proposition 3.6.1** *The partial order of a domain is a cpo.*

**Proposition 3.6.2** *The partial order of a trace space is a cpo.*

**Proposition 3.6.3** *The partial order of an event space is a cpo.*

The following propositions characterize the continuity of basic transductions in partial order topologies.

**Proposition 3.6.4** *A transliteration $f_{\mathcal{T}} : A^{\mathcal{T}} \to A'^{\mathcal{T}}$ on any time structure $\mathcal{T}$ is continuous if $f : A \to A'$ is continuous.*

**Proposition 3.6.5** *A unit delay on any discrete time structure is continuous.*

**Proposition 3.6.6** *A transport delay is continuous.*

The following proposition characterizes the continuity of event-driven transductions.

**Proposition 3.6.7** *An event-driven transduction $F^{\circ}$ is continuous if its primitive transduction $F$ on any discrete time structure is continuous.*

The following theorem concludes these properties.

**Theorem 3.6.1 ($\Sigma$-dynamics structure)** *Let $A$ be a $\Sigma$-domain structure and $\mathcal{T}$ a time structure. The $\Sigma$-dynamics structure $\mathcal{D}(\mathcal{T}, A) = \langle \mathcal{V}, \mathcal{F} \rangle$ satisfies (1) $\mathcal{V}$ is a multi-sorted set of cpos and (2) transliterations, transport delays and event-driven transductions in $\mathcal{F}$ are continuous in the partial order topology. If, in addition, $\mathcal{T}$ is discrete, all transductions in $\mathcal{F}$ are continuous in the partial order topology.*

Transductions are functions. The *well-definedness and strictness* of a transduction is the well-definedness and strictness of the function, respectively. The following propositions characterize well-defined and/or strict transductions in dynamics structures.

**Proposition 3.6.8** *A transliteration $f_{\mathcal{T}}$ is well-defined iff function $f$ is well-defined; $f_{\mathcal{T}}$ is strict w.r.t. an argument iff $f$ is strict w.r.t. the argument.*

**Proposition 3.6.9** *Any delay is not strict. A unit delay on any discrete time structure is well-defined. A transport delay is well-defined.*

**Proposition 3.6.10** *An event-driven transduction $F^{\circ}$ is well-defined iff $F$ on any discrete time structure is well-defined; $F^{\circ}$ is strict w.r.t. its event input, and $F^{\circ}$ is strict w.r.t. one of the other input arguments iff $F$ is strict w.r.t. the argument.*

Event traces are nonintermittent and right-continuous. We call a transduction *nonintermittent* iff its output is nonintermittent whenever its input is nonintermittent. We call a transduction *right-continuous* iff its output is right-continuous whenever its input is right-continuous. The following propositions characterize nonintermittent and/or right-continuous transductions in dynamics structures.

**Proposition 3.6.11** *A transliteration $f_{\mathcal{T}}$ is right-continuous if $f$ is continuous in the derived metric topology; $f_{\mathcal{T}}$ with $f : \times_I \overline{A_i} \rightarrow \overline{A}$ is nonintermittent if $f$ is strict, well-defined and continuous in the derived metric topology.*

**Proposition 3.6.12** *A delay is nonintermittent. A transport delay is right-continuous.*

**Proposition 3.6.13** *An event-driven transduction is right-continuous. An event-driven transduction $F^{\circ}$ is nonintermittent if $F$ is nonintermittent.*

For example, the "event or" transduction $\oplus$ (Figure 3.2) is well-defined and strict; it is also right-continuous and nonintermittent. "Event or" is a typical event synchronizer. In Chapter 5, Modeling in Constraint Nets, we will define other event synchronizers that are all nonintermittent and right-continuous.

We have presented a topological structure of dynamics by formalizing time, domains and traces in topological spaces and by characterizing primitive and event-driven transductions. With such a topological structure, continuous/discrete time and domains can be represented uniformly, and hybrid dynamic systems can be studied in a unitary model.

# Chapter 4

# The Constraint Net Model

A hybrid dynamic system can have multiple sorts corresponding to different data types that can be numerical, symbolic or logical. It can have multiple components with different time structures generated by different clocks, and clocks can be generated or synchronized.

In this chapter, we present a formal model for hybrid dynamic systems, that we call Constraint Nets (CN). We first define the syntax of CN. We then provide a fixpoint semantics of CN using the fixpoint theory of partial orders. Finally, we discuss parameterized CN and temporal integration in CN.

## 4.1 Syntax of Constraint Nets

In this section, we introduce the syntax of constraint nets and characterize the composite structure and modularity of the model.

### 4.1.1 Syntax and graphical representation

A constraint net consists of a finite set of locations, a finite set of transductions and a finite set of connections.

**Definition 4.1.1 (Syntax)** *A* constraint net *is a triple* $CN = \langle Lc, Td, Cn \rangle$*, where* $Lc$ *is a finite set of* locations*, each associated with a sort;* $Td$ *is a finite set of labels of* transductions*, each with an* output port *and a set of* input ports*, and each port is associated with a sort;* $Cn$ *is a set of* connections *between locations and ports of the same sort, with the following restrictions: (1) there is at most one output port connected to each location, (2) each port of a transduction connects to a unique location and (3) no location is isolated.*

Intuitively, each location is of fixed sort; a location's value typically changes over time. A location can be regarded as a wire, a channel, a variable, or a memory cell. Each transduction is a causal mapping from inputs to outputs over time, operating according to a certain reference time or activated by external events. Connections relate locations with ports of transductions. A clock is a special kind of location connected to the event ports of event-driven transductions.

A location $l$ is an *output location* of a transduction $F$ iff $l$ connects to the output port of $F$; $l$ is an *input location* of $F$ iff $l$ connects to an input port of $F$. A location is an *output* of the constraint net if it is an output location of a transduction; it is otherwise an *input*. A constraint net is *open* if there is an input location; it is otherwise *closed*. We use $I(CN)$ and $O(CN)$ to denote the set of input locations and the set of output locations, respectively, of a constraint net $CN$.

A constraint net is depicted by a bipartite graph where locations are depicted by circles, transductions by boxes and connections by arcs. For example, the graph in Figure 4.1, where $f$ is a transliteration and $\delta$ is a unit delay, depicts an open net. The net, with a discrete time structure, models a state automaton: $s(0) = s_0$, $s(n) = f(i(n-1), s(n-1))$. The closed net

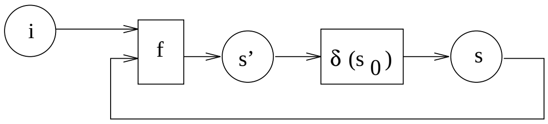


Figure 4.1: The constraint net representing a state automaton

depicted by the graph in Figure 4.2, with a continuous time structure, models a differential equation $\dot{s} = f(s)$.

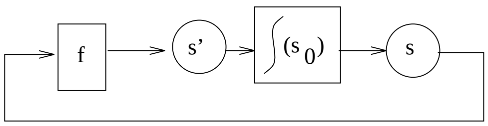


Figure 4.2: The constraint net representing $\dot{s} = f(s)$

### 4.1.2 Modules and composition

A system may be composed of subsystems. In order to capture the hierarchical composition structure of systems, we introduce subnets and modules.

**Definition 4.1.2 (Subnet)** *A constraint net* $CN_1 = \langle Lc_1, Td_1, Cn_1 \rangle$ *is a* subnet *of* $CN_2 = \langle Lc_2, Td_2, Cn_2 \rangle$*, written* $CN_1 \subseteq CN_2$*, iff* $Lc_1 \subseteq Lc_2$*,* $Td_1 \subseteq Td_2$*,* $Cn_1 \subseteq Cn_2$ *and* $I(CN_1) \subseteq I(CN_2)$*.*

**Definition 4.1.3 (Module)** *A* module *is a triple* $\langle CN, I, O \rangle$*, also denoted* $CN(I, O)$*, where* $CN$ *is a constraint net,* $I \subseteq I(CN)$ *and* $O \subseteq O(CN)$ *are subsets of the input and output locations of* $CN$*, respectively;* $I \cup O$ *defines the* interface *of the module.*

A module $CN(I, O)$ is *closed* if $I = \emptyset$; it is otherwise *open*. Locations in $I(CN) - I$ are *hidden inputs* and locations in $O(CN) - O$ are *hidden outputs*. A module will be depicted by a box with rounded corners.

We define three basic operations — union, coalescence and hiding — that can be applied to obtain a new module from existing ones.

The *union* operation generates a new module by putting two modules side by side. Formally, let $CN_1 = \langle Lc_1, Td_1, Cn_1 \rangle$ and $CN_2 = \langle Lc_2, Td_2, Cn_2 \rangle$ be two constraint nets, with $Lc_1 \cap Lc_2 = \emptyset$ and $Td_1 \cap Td_2 = \emptyset$, [1] then the union of $CN_1(I_1, O_1)$ and $CN_2(I_2, O_2)$, written $CN_1(I_1, O_1) \parallel CN_2(I_2, O_2)$, is a new module $CN(I, O)$ where $CN = \langle Lc, Td, Cn \rangle$ is a constraint net with $Lc = Lc_1 \cup Lc_2$, $Td = Td_1 \cup Td_2$ and $Cn = Cn_1 \cup Cn_2$, $I \cup O$ defines its interface with $I = I_1 \cup I_2$ and $O = O_1 \cup O_2$.

The *coalescence* operation coalesces two locations in the interface of a module into one, with the restriction that at least one of these two locations is an input location. Formally, let $CN = \langle Lc, Td, Cn \rangle$ be a constraint net, $l \in I$ and $l' \in I \cup O$ be of the same sort, the coalescence of $CN(I, O)$ for $l$ and $l'$, denoted $CN(I, O)/(l, l')$, is a new module $CN'(I', O')$ with $CN' = \langle Lc[l'/l], Td, Cn[l'/l] \rangle$, $I' = I - \{l\}$ and $O' = O$, where $X[v/x]$ denotes that $x$ in $X$ is replaced by $v$.

The *hiding* operation deletes a location from the interface. Formally, let $CN = \langle Lc, Td, Cn \rangle$ be a constraint net and $l \in I \cup O$, the hiding of $CN(I, O)$ for $l$, denoted $CN(I, O) \backslash l$, is a new module $CN'(I', O')$ with $CN' = CN$, $I' = I - \{l\}$ and $O' = O - \{l\}$.

In addition, we define three combined operations: cascade connection, parallel connection and feedback connection. The *cascade connection* connects two modules in series. The *parallel connection* connects two modules in parallel. The *feedback connection* connects an output of the module to an input of its own.

Figure 4.3 depicts the three operations. The formal definitions of these operations, in terms of basic operations, are as follows.

[1] Note that $Td$ is a set of transduction labels, which can be different for the same transduction.

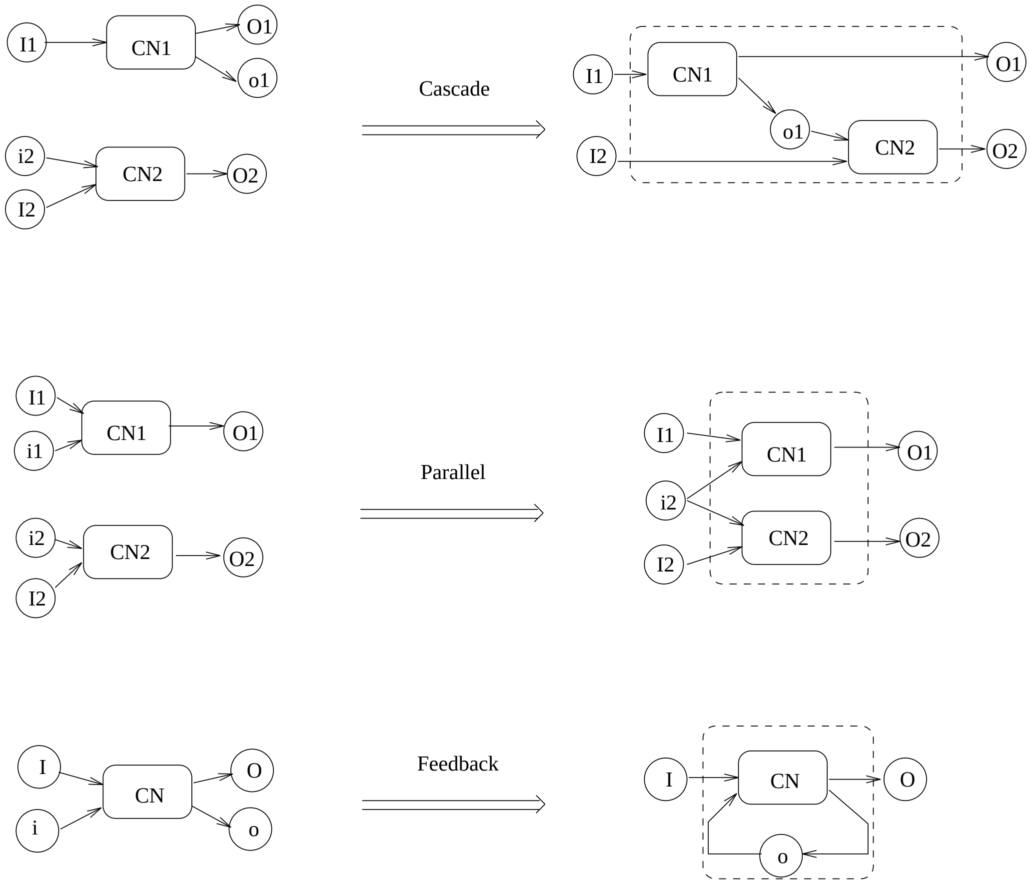


Figure 4.3: Cascade, parallel and feedback connections

Let $o_1 \in O_1$ and $i_2 \in I_2$. A cascade connection of $CN_1(I_1, O_1)$ and $CN_2(I_2, O_2)$, denoted $CN_2(I_2, O_2) \circ CN_1(I_1, O_1)$, produces a new module $CN(I, O)$,

$$CN(I, O) = [(CN_1(I_1, O_1) \parallel CN_2(I_2, O_2))/(i_2, o_1)] \backslash o_1.$$

Let $i_1 \in I_1$ and $i_2 \in I_2$. A parallel connection of $CN_1(I_1, O_1)$ and $CN_2(I_2, O_2)$, denoted $CN_1(I_1, O_1) + CN_2(I_2, O_2)$, produces a new module $CN(I, O)$,

$$CN(I, O) = (CN_1(I_1, O_1) \parallel CN_2(I_2, O_2))/(i_1, i_2).$$

Let $i \in I$ and $o \in O$. A feedback connection of $CN(I, O)$, denoted $\mathcal{F}(CN(I, O))$, produces a new module $CN'(I', O')$,

$$CN'(I', O') = [CN(I, O)/(i, o)] \backslash o.$$

The following relations hold for these syntactic operations.

**Proposition 4.1.1**

$$CN_1(I_1, O_1) \parallel CN_2(I_2, O_2) = CN_2(I_2, O_2) \parallel CN_1(I_1, O_1).$$

$$CN_1(I_1, O_1) \circ (CN_2(I_2, O_2) \circ CN_3(I_3, O_3)) = (CN_1(I_1, O_1) \circ CN_2(I_2, O_2)) \circ CN_3(I_3, O_3)$$

*if both sides are defined.*

$$CN_1(I_1, O_1) + (CN_2(I_2, O_2) + CN_3(I_3, O_3)) = (CN_1(I_1, O_1) + CN_2(I_2, O_2)) + CN_3(I_3, O_3)$$

*if both sides are defined.*

**Proposition 4.1.2** *Following are some properties of subnets:*

*(1) $CN_1$ and $CN_2$ are subnets of $CN_1 \parallel CN_2$.*

*(2) $CN_1$ and $CN_2$ are subnets of $CN_1 + CN_2$.*

*(3) $CN_1$ is a subnet of $CN_2 \circ CN_1$, however, $CN_2$ is not a subnet of $CN_2 \circ CN_1$.*

There are at least three reasons to introduce modules.

First, modules facilitate hierarchical composition structures for complex systems. For example, we can create a state automaton module $SA$ by selecting $\{i, s\}$ or $\{i, s'\}$ as the interface for the constraint net in Figure 4.1. An input/output automaton $IOA$ can be constructed by cascading $SA$ to a transliteration $g$ as shown in Figure 4.4. $IOA$ defines a transduction from input traces to output traces.

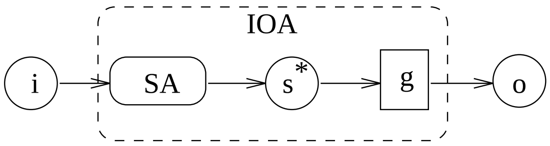


Figure 4.4: An input/output automaton ($s^*$ denotes either $s$ or $s'$)

Second, modules provide a flexible way to generate different systems from the same set of components. To illustrate this idea, let us again consider input/output automata. In general, an input/output automaton is a tuple $\langle \mathcal{I}, \mathcal{S}, s_0, f_s, \mathcal{O}, f_o \rangle$ where $\mathcal{I}$ is the set of input values, $\mathcal{S}$ is the set of states with $s_0 \in \mathcal{S}$ as the initial state, $f_s : \mathcal{I} \times \mathcal{S} \to \mathcal{S}$ is a state transition function, $\mathcal{O}$ is the set of output values and $f_o$ is an output function. However, there are two ways to define an output function, corresponding to two types of input/output automata, $f_o : \mathcal{I} \times \mathcal{S} \to \mathcal{O}$ for Mealy machines [Mea55] and $f_o : \mathcal{S} \to \mathcal{O}$ for Moore machines [Moo56]. In a constraint net model, a Mealy or Moore machine is derived by selecting different output locations as the interface of its state automaton module. If we select $\{i, s'\}$ as the interface of $SA$, then $IOA$ is a Mealy machine with $f_s = f$ and $f_o = g \circ f$. If we select $\{i, s\}$ as the interface of $SA$, then $IOA$ is a Moore machine with $f_s = f$ and $f_o = g$.

Third, modules capture the notion of abstraction through hidden locations. Hidden outputs encapsulate internal structures of a system. However, the role of hidden inputs is not so obvious. Consider again the state automaton in Figure 4.1. By hiding the only input location $i$, we obtain a closed module representing a nondeterministic state transition system. More specifically, the state transition function $f$ defines a state transition relation $R \subseteq \mathcal{S} \times \mathcal{S}$, such that $(s, s') \in R$ iff $\exists i \in \mathcal{I}, s' = f(i, s)$, or equivalently, the set of next possible states of a state $s$ is $\{f(i, s) | i \in \mathcal{I}\}$. In general, any module $CN(I, O)$ with $I \subset I(CN)$ defines a nondeterministic system. Similar concepts have been explored in general systems theory [MT75]. We will discuss more on nondeterministic behaviors of modules in Chapter 6, Behavior Analysis. Furthermore, we can associate hidden locations with random distributions. Thus, while simpler than most inherently nondeterministic models, the Constraint Net model can also incorporate probabilistic and stochastic analysis.

## 4.2 Semantics of Constraint Nets

We have presented the syntactical structure of constraint nets, which is graphical and modular. However, syntax only serves as a mechanism for creating a model, the meaning of which is not provided. There are many models with syntax similar to constraint nets (Petri Nets [Pet81] for example) that have totally different interpretations.

Since transductions are mappings from traces to traces, a constraint net denotes a set of equations with locations as variables and transductions as functions; the semantics of the constraint net should be a solution of the set of equations.

A set of equations may have no solution, or exactly one solution, or more than one solution. For example, if $x \in \mathcal{R}$, $x = x - 2$ has no solution, $x = 0.5x - 2$ has one solution $(-4)$, and $x = x^2 - 2$ has two solutions ($-1$ and $2$). The fixpoint theory of partial orders has been applied to provide denotational semantics for programming languages and models [Hen88]: a program or a model defines a function $f$ and its semantics is the least solution of $x = f(x)$, or the least fixpoint of $f$.

In this section, we will first present the fixpoint theory of partial orders and then apply this theory to provide a fixpoint semantics for the Constraint Net model.

### 4.2.1 Fixpoint theory of partial orders

A fixpoint of a function $f$ can be considered as a solution of the equation $x = f(x)$. The least fixpoint is the least element in the fixpoint set.

**Definition 4.2.1 (Fixpoint and Least fixpoint)** *Let $f : A \to A$ be a function on a partial order $A$. An element $a \in A$ is a* fixpoint *of $f$ iff $a = f(a)$. It is the* least fixpoint *of $f$ iff, in addition, $a \leq_A a'$ for every fixpoint $a'$ of $f$.*

Least fixpoints, if they exist, are unique. The least fixpoint of $f$ will be denoted by $\mu.f$.

The first fixpoint theorem is stated as follows.

**Theorem 4.2.1 (Fixpoint Theorem I)** *Let $A$ be a cpo. Every continuous function $f : A \to A$ has a least fixpoint.*

We shall provide the proof of this theorem next, since the proof itself is to construct the least fixpoint.

Proof: Define $x_f^n$ by induction on $n$:

$$x_f^0 = \perp_A,$$
$$x_f^{n+1} = f(x_f^n).$$

$x_f^0 \leq x_f^1$ because $x_f^0$ is the least element in $A$. Since $f$ is monotonic (Proposition 3.1.10), we have $f(x_f^0) \leq f(x_f^1)$, i.e., $x_f^1 \leq x_f^2$. Continuing this we have a chain

$$x_f^0 \leq x_f^1 \leq x_f^2 \ldots \leq x_f^n \leq \ldots.$$

Since $A$ is a *cpo*, this chain has a least upper bound $\bigvee_A\{x_f^n|n \geq 0\}$, which we denote by $x_f$.

Since $f(x_f) = \bigvee_A\{f(x_f^n)|n \geq 0\} = \bigvee_A\{x_f^n|n \geq 1\} = x_f$ (Proposition 3.1.11), then $x_f$ is a fixpoint of $f$.

We now show that $x_f$ is the least fixpoint. Suppose $y$ is a fixpoint of $f$. We have: $x_f^0 \leq y$ because $x_f^0$ is $\perp_A$. Furthermore, suppose $x_f^n \leq y$, then $x_f^{n+1} = f(x_f^n) \leq f(y) = y$. Therefore $x_f^k \leq y$ for any $k$ by induction. Thus, $y$ is an upper bound for the chain $\{x_f^n|n \geq 0\}$. Hence, $x_f \leq y$.

Therefore, for a continuous function $f : A \rightarrow A$, $\mu.f = \bigvee_A\{f^n(\perp_A)|n \geq 0\}$. □

By extending $f$ to a function of two arguments, we have the second fixpoint theorem.

**Theorem 4.2.2 (Fixpoint Theorem II)** *Let $A$ and $A'$ be two cpos. If $f : A \times A' \rightarrow A'$ is a continuous function, then there exists a unique continuous function $\mu.f : A \rightarrow A'$, such that for all $a \in A$, $(\mu.f)(a)$ is the least fixpoint of $\lambda x.f(a, x)$, or equivalently, $\forall a \in A, (\mu.f)(a) = f(a, (\mu.f)(a))$.*

The continuous function $\mu.f : A \rightarrow A'$ is called the *least fixpoint* of function $f : A \times A' \rightarrow A'$ or the *least solution* of the equation $y = f(x, y)$.

Now we further investigate general properties of equations in complete partial orders.

**Proposition 4.2.1** *Let $I \subseteq J$ be an index set. If $f : \times_I A_i \rightarrow A$ is a continuous function, then the extension of $f$, $f' : \times_J A_j \rightarrow A$ satisfying $f'(a) = f(a_{|I})$, is a continuous function.*

**Proposition 4.2.2** *Let $\{f_k : \times_J A_j \rightarrow A_k\}_{k \in K}$ be a family of continuous functions. Then $\vec{f} : \times_J A_j \rightarrow \times_K A_k$ with $\vec{f}(a)_k = f_k(a)$ is a continuous function.*

**Proposition 4.2.3** *If $\vec{f} : \times_J A_j \rightarrow \times_K A_k$ is a continuous function, $K \subseteq J$ and $I = J - K$, then $\vec{f}$ has a least fixpoint $\mu.\vec{f} : \times_I A_i \rightarrow \times_K A_k$.*

**Proposition 4.2.4** *Let $X$ be a set of variables and $O \subseteq X$ a set of output variables. Let $\{f_o : \times_{I_o} A_i \rightarrow A_o\}_{o \in O}$ be a set of continuous functions. Then the set of equations $\{o = f_o(\vec{x})\}_{o \in O}$ with $\vec{x} : I_o \rightarrow X$ has a least solution.*

A set of equations can also be written as $\vec{o} = \vec{f}(\vec{i}, \vec{o})$ where $\vec{i}$ is a tuple of input variables and $\vec{o}$ is a tuple of output variables. If $\vec{f}$ is continuous, then its least fixpoint is a continuous function, denoted $\mu.\vec{f}$.

### 4.2.2 Semantics of constraint nets

In this section, we define the fixpoint semantics of constraint nets. Let $\Sigma = \langle S, F\rangle$ be a signature and $c \in S$ be a special sort for clocks. A constraint net with signature $\Sigma$ is a triple $CN_\Sigma = \langle Lc, Td, Cn\rangle$ where

- each location $l \in Lc$ is associated with a sort $s \in S$, the sort of location $l$ is written as $s_l$;
- each transduction $F \in Td$ is a basic transduction or an event-driven transduction, the sorts of the input and output ports of $F$ are as follows:
  1. if $F$ is a transliteration of a function $f : s^* \to s \in F$, the sort of the output port is $s$ and the sort of the input port $i$ is $s^*(i)$;
  2. if $F$ is a unit delay $\delta^s$ or a transport delay $\Delta^s$, the sort of both input and output ports is $s$;
  3. if $F$ is an event-driven transduction, the sort of the event input port is $c$, the sorts of the other ports are the same as its primitive transduction.

Let $\mathcal{D}(\mathcal{T}, A) = \langle \mathcal{V}, \mathcal{F}\rangle$ be a $\Sigma$-dynamics structure. $CN_\Sigma$ on $\langle \mathcal{V}, \mathcal{F}\rangle$ denotes a set of equations $\{o = F_o(\vec{x})\}_{o\in O(CN)}$, such that for any output location $o \in O(CN)$,

- $F_o$ is a continuous transduction in $\mathcal{F}$ whose output port connects to $o$,
- $\vec{x}$ is the tuple of input locations of $F_o$, i.e., the input port $i$ of $F_o$ connects to location $\vec{x}(i)$.

The semantics of a constraint net is defined as follows.

**Definition 4.2.2 (Semantics)** *The* semantics *of a constraint net $CN$ on a dynamics structure $\langle \mathcal{V}, \mathcal{F}\rangle$, denoted $[\![CN]\!]$, is the least solution of the set of equations $\{o = F_o(\vec{x})\}_{o\in O(CN)}$, given that $F_o$ is a continuous transduction in $\mathcal{F}$ for all $o \in O(CN)$; it is a continuous transduction from the input trace space to the output trace space, i.e., $[\![CN]\!] : \times_{I(CN)} A_{s_i}^{\mathcal{T}} \to \times_{O(CN)} A_{s_o}^{\mathcal{T}}$.*

Given any set of output locations $O$, the restriction of $[\![CN]\!]$ onto $O$, denoted $[\![CN]\!]_{|O} : \times_{I(CN)} A_{s_i}^{\mathcal{T}} \to \times_O A_{s_o}^{\mathcal{T}}$, is called *the semantics of $CN$ for $O$*. For example, the constraint net in Figure 4.1 denotes equations $s' = f(i, s)$ and $s = \delta(s_0)(s)$. Given a discrete time structure $\mathcal{N}$, a domain $\overline{\mathcal{I}}$ for inputs and a domain $\overline{\mathcal{S}}$ for states, the semantics for $s$ is $F : \overline{\mathcal{I}}^{\mathcal{N}} \to \overline{\mathcal{S}}^{\mathcal{N}}$ such that $F(v)(0) = s_0$ and $F(v)(n) = f(v(n-1), F(v)(n-1))$.

The nonintermittent and right-continuous transductions are closed under all types of composition.

**Proposition 4.2.5** *If a constraint net is composed of nonintermittent transductions, then its semantics is nonintermittent. If a constraint net is composed of right-continuous transductions, then its semantics is right-continuous.*

The semantics of a subnet can be extended.

**Proposition 4.2.6** *If $CN'$ is a subnet of $CN$, $[\![CN]\!]_{|O(CN')}(\vec{i}) = [\![CN']\!](\vec{i}_{|I(CN')})$.*

### 4.2.3 Semantics of modules

We have defined the semantics of a constraint net as a transduction. We now define the semantics of a module as a set of transductions.

**Definition 4.2.3 (Semantics of modules)** *Given that the semantics of a constraint net $CN$ is $[\![CN]\!] : \times_{I(CN)} A_{s_i}^{\mathcal{T}} \to \times_{O(CN)} A_{s_o}^{\mathcal{T}}$, the* semantics of a module *$CN(I, O)$ is $[\![CN(I, O)]\!] = \{F_u : \times_I A_{s_i}^{\mathcal{T}} \to \times_O A_{s_o}^{\mathcal{T}}\}_{u \in U}$ where $F_u(i) = [\![CN]\!]_{|O}(u, i)$ and $U \subset \times_{I(CN)-I} A_{s_i}^{\mathcal{T}}$ is the set of well-defined hidden input traces.*

For example, if locations $i$ and $s'$ in Figure 4.1 are hidden, the semantics of the module is a set of traces $\{F(i)\}_{i \in \mathcal{I}^{\mathcal{N}}}$.

The semantics of a composite module can be derived from the semantics of its components.

**Proposition 4.2.7** *Following are some properties associated with module operations:*

- Union*: If $CN(I, O) = CN_1(I_1, O_1) \parallel CN_2(I_2, O_2)$, then*

$$[\![CN(I, O)]\!] = [\![CN_1(I_1, O_1)]\!] \times [\![CN_2(I_2, O_2)]\!].$$

- Cascade connection*: If $CN(I, O) = CN_2(I_2, O_2) \circ CN_1(I_1, O_1)$, then*

$$[\![CN(I, O)]\!] = \{F_2 \circ F_1 | F_1 \in [\![CN_1(I_1, O_1)]\!], F_2 \in [\![CN_2(I_2, O_2)]\!]\}.$$

- Parallel connection*: If $CN(I, O) = CN_1(I_1, O_1) + CN_2(I_2, O_2)$, then*

$$[\![CN(I, O)]\!] = \{\langle F_1, F_2 \rangle | F_1 \in [\![CN_1(I_1, O_1)]\!], F_2 \in [\![CN_2(I_2, O_2)]\!]\}$$

  *where $\langle F_1, F_2 \rangle_{|O_1}(i) = F_1(i_{|I_1})$ and $\langle F_1, F_2 \rangle_{|O_2}(i) = F_2(i_{|I_2})$.*

- Feedback connection*: If $CN'(I', O') = \mathcal{F}(CN(I, O))$, then*

$$[\![CN'(I', O')]\!] = \{\mu.F | F \in [\![CN(I, O)]\!]\}$$

  *where $\mu.F$ is the the least fixpoint of $F$.*

Now we discuss the well-definedness of systems. A constraint net $CN$ is *well-defined* iff its semantics, transduction $[\![CN]\!]$, is well-defined. For example, the constraint net in Figure 4.1, given a well-defined function $f$ and with a discrete time structure, is well-defined. A module is *well-defined* iff all the transductions in its semantics are well-defined. If a constraint net is well-defined, all its modules are well-defined.

The well-definedness of modules is closed under some module operations.

**Proposition 4.2.8** *If $CN_1(I_1, O_1)$ and $CN_2(I_2, O_2)$ are well-defined modules, then $CN_1(I_1, O_1) \parallel CN_2(I_2, O_2)$, $CN_1(I_1, O_1) \circ CN_2(I_2, O_2)$ and $CN_1(I_1, O_1) + CN_2(I_2, O_2)$ are well-defined modules.*

However, well-definedness is not closed under the feedback operation.

There is a relationship between the well-definedness of a constraint net and the strictness of transductions in the constraint net, which is derived from the following property of strict continuous functions.

**Proposition 4.2.9** *Let $A$ and $A'$ be two cpos. If $f : A \times A' \to A'$ is a strict continuous function w.r.t. its second argument, then the least fixpoint of $f$, or the least solution of the equation $o = f(i, o)$, is undefined.*

For example, let $+, \cdot : \overline{\mathcal{R}} \times \overline{\mathcal{R}} \to \overline{\mathcal{R}}$ be strict extensions of $+$ and $\cdot$, respectively. Let $+, \cdot : \overline{\mathcal{R}}^{\mathcal{T}} \times \overline{\mathcal{R}}^{\mathcal{T}} \to \overline{\mathcal{R}}^{\mathcal{T}}$ be the corresponding transliterations. The least solution of $x = 0.5x + 2$ on $\mathcal{D}(\mathcal{T}, \overline{\mathcal{R}})$ is undefined, even though $\lambda t.4$ is a well-defined solution.

In general, a net is not well-defined if there is an algebraic loop.

**Definition 4.2.4 (Algebraic loop)** *Let $CN$ be a constraint net. A location $l$ is* strictly dependent *on a location $l'$ in $CN$, written $l \leftarrow l'$, iff: (1) there is a transduction $F$ in $CN$ such that $l$ is the output location of $F$, $l'$ is an input location of $F$, and $F$ is strict w.r.t. the input port (indicating an input argument) that connects with $l'$; or (2) $\exists l'' : l \leftarrow l'', l'' \leftarrow l'$. $CN$ has an* algebraic loop *on a location $l$ iff $l \leftarrow l$.*

**Proposition 4.2.10** *A module $CN(I, O)$ is not well-defined if there is an output location $l \in O$ such that $CN$ has an algebraic loop on $l$.*

A common strategy to break an algebraic loop is to insert a delay. For example, by inserting a unit delay $\delta(0)$ to the equation $x = 0.5x + 2$, we have $y = 0.5x + 2, x = \delta(0)(y)$. Let $\mathcal{N}$ be the time structure. The semantics of the net for $x$ is a sequence $0, 2, 3, 3.5, 3.75, \ldots$ and

$\lim_{n\to\infty} x(n) = 4$. Note that 4 is a solution of $x = 0.5x + 2$ on $\mathcal{R}$. In general, a well-defined solution of $x = f(x)$ for a continuous function $f$ can be computed via a relaxation method: $x(n+1) = f(x(n)) = f^{n+1}(x(0))$ if $\lim_{n\to\infty} f^n(x_0)$ is well-defined, and any relaxation method can be modeled as a state automaton in constraint nets. We will discuss this type of computation further in Part III.

### 4.2.4 Parameterized nets

In this section, we introduce parameterized nets and discuss the limiting semantics of parameterized nets.

A system may have qualitatively different properties with respect to different parameters. A *parameter* is a variable in a transduction whose value does not change over time. For example, mass, friction coefficient, initial state, time delay, gain and threshold are typical parameters of robotic systems. Let $CN$ be a constraint net and $P$ be a set of parameters in $CN$. We use $CN^P$ and $CN^P(I,O)$ to denote a *parameterized net* and a *parameterized module*, respectively. Associated with each parameter $p \in P$ is a set of values $D_p$; $\times_P D_p$ is called the *parameter space*. The semantics of a parameterized net $CN^P$ is defined as follows.

**Definition 4.2.5 (Semantics of parameterized nets)** *The* semantics *of a parameterized net $CN^P$, denoted $[\![CN^P]\!]$, is a mapping from the parameter space to the set of transductions, i.e., $[\![CN^P]\!] : \times_P D_p \to (\times_{I(CN)} A_{s_i}^{\mathcal{T}} \to \times_{O(CN)} A_{s_o}^{\mathcal{T}})$ such that for any parameter tuple $v \in \times_P D_p$, $[\![CN^P]\!](v) = [\![CN[v/P]]\!]$ where $CN[v/P]$ denotes that each $p \in P$ in $CN$ is replaced by its corresponding value $v(p)$.*

The *semantics* of a parameterized module $CN^P(I,O)$, denoted $[\![CN^P(I,O)]\!]$, is a function of parameters as well: $[\![CN^P(I,O)]\!](v) = [\![CN(I,O)[v/P]]\!]$.

There are at least two reasons to introduce parameterized nets.

First, a system can be modeled and analyzed against its parameters. A property of a system may change qualitatively when the value of its parameters changes from one to another. For example, let $k$ be a gain parameter with $D_k = \mathcal{R}$, and $y = kx + 2, x = \delta(0)(y)$ be a net on dynamics structure $\mathcal{D}(\mathcal{N}, \overline{\mathcal{R}})$. The semantics for $x$ is a sequence $0, 2, 2k+2, \ldots$. If $|k| < 1$, we have $\lim_{n\to} x(n) = \frac{2}{1-k}$; if $|k| \geq 1$, we have $\lim_{n\to\infty} x(n) = \perp_{\mathcal{R}}$. In general, $\lim_{n\to\infty} f^n(x_0)$ exists in $\mathcal{R}$, if $f$ is a contractor [MA86], i.e., $\exists k < 1, |f(x) - f(y)| \leq k|x - y|$. A qualitative property is *stable* w.r.t. its parameter iff the parameter region that supports the property is open. In the previous example, the convergent property is stable since $\{k | k \in \mathcal{R}, |k| < 1\}$ is open. Intuitively, a stable property means that a small change in the value of its parameters will not cause a qualitative change of the property.

Second, limiting semantics can be defined. Let $P$ be a set of parameters, $\times_P D_p$ be the parameter space, and $\leq_{\times_P D_p}$ be a partial order relation. If $\langle \times_P D_p, \leq_{\times_P D_p} \rangle$ is a linear order, and $CN^P$ is a closed parameterized net whose semantics is a mapping $[\![CN^P]\!] : \times_P D_p \to \times_{Lc} A_{s_l}^{\mathcal{T}}$, the *limiting semantics* of $CN^P$ w.r.t. the parameter set $P$, written $[\![CN^*]\!]$, is defined as the limit of the linear set of traces $[\![CN^P]\!]$, i.e., $[\![CN^*]\!] = \lim[\![CN^P]\!]$.

Infinitesimal is an important parameter for limiting semantics. Let $\epsilon$ be a parameter with $D_\epsilon = (0,1) \subset \mathcal{R}$. Let $\leq_{D_\epsilon}$ be a partial order relation such that $\epsilon_1 \leq_{D_\epsilon} \epsilon_2$ iff $\epsilon_2 \leq_{\mathcal{R}} \epsilon_1$. $\langle D_\epsilon, \leq_{D_\epsilon} \rangle$ is a linear order. The limiting semantics of $CN^\epsilon$ w.r.t. $\epsilon$ is $\lim_{\epsilon \to 0}[\![CN^\epsilon]\!]$. We call such a parameter $\epsilon$ an *infinitesimal*. For example, let $CN^\epsilon$, with parameter $\epsilon$ as an infinitesimal, be a closed parameterized net denoting $y = f(x), x = \Delta(\epsilon)(x_0)(y)$ on $\mathcal{D}(\mathcal{R}^+, \overline{\mathcal{R}})$. If $f = \lambda x.x$, then $x = \lambda t.x_0$; if $f = \lambda x.(-x)$ and $x_0 \neq 0$, then $x(t) = \perp_{\mathcal{R}}$ for all $t > 0$.

### 4.2.5 Temporal integration

So far we have no definition for temporal integration, the most important type of transduction on continuous time structures. We now define temporal integration on vector spaces and provide the semantics of constraint nets with temporal integration using limiting semantics.

A *vector space* [War72] is a set $X$ associated with the functions sum and product: $+ : X \times X \to X$ and $\cdot : \mathcal{R} \times X \to X$ and with $0_X \in X$ satisfying the following conditions:

$$x + y = y + x, (x + y) + z = x + (y + z),$$

$$\alpha(x + y) = \alpha x + \alpha y, (\alpha + \beta)x = \alpha x + \beta x,$$

$$\alpha(\beta x) = (\alpha\beta)x, x + 0_X = x, 0x = 0_X, 1x = x.$$

Let $\Sigma_I x_i$ denote the sum of all elements in $\{x_i\}_{i \in I}$. A *topological vector space* is a vector space with a topology such that $+$ and $\cdot$ are continuous functions.

Let $U$ be a vector space with functions $+ : U \times U \to U$ and $\cdot : \mathcal{R} \times U \to U$ continuous in metric topology. *Temporal integration* $\int(s_0) : \overline{U}^{\mathcal{T}} \to \overline{U}^{\mathcal{T}}$ with an initial state $s_0 \in U$ can be defined as follows.

Let $+$ and $\cdot$ be strict extensions.

Given that $\mathcal{T}$ is a discrete time structure, for all $t > \mathbf{0}$, $pre(t)$ denotes the previous time point. Temporal integration is defined as follows:

$$\int(s_0)(u) = \lambda t. \begin{cases} s_0 & \text{if } t = \mathbf{0} \\ \Sigma_{\mathbf{0} < t' \leq t} \mu([pre(t'), t')) \cdot u(pre(t')) & \text{otherwise.} \end{cases}$$

We can represent $\int(s_0)$ as the least solution of the following equation

$$s = \delta(s_0)(s) + dt \cdot \delta(0)(u)$$

where

$$dt = \lambda t. \begin{cases} 0 & \text{if } t = \mathbf{0} \\ \mu([pre(t), t)) & \text{otherwise.} \end{cases}$$

This equation can be represented by a constraint net that computes temporal integration on discrete time structures.

Given that $\mathcal{T}$ is an arbitrary time structure, temporal integration is defined as follows: Let $\mathcal{T}_e$ be a discrete sample time of $\mathcal{T}$, generated by an event trace $e$ with $e = \Delta(\epsilon)(0)(\neg e)$ for an infinitesimal $\epsilon$. Let $int_{s_0}(u, s) = \delta(s_0)(s) + dt \cdot \delta(0)(u)$. Temporal integration $\int(s_0)$ can be computed by a module $CN(u, s)$ where $CN$ denotes the following two equations:

$$s = int^{\circ}_{s_0}(e, u, s), \;\; e = \Delta(\epsilon)(0)(\neg e)$$

with $\epsilon > 0$ as an infinitesimal.

This definition can be considered as derived by the forward Euler method; however, we are interested in semantics, rather than numerical simulation of differential equations.

As an example, let us investigate the limiting semantics of the net in Figure 4.2 with $U$ as $\mathcal{R}$, $\mathcal{T}$ as $\mathcal{R}^+$ and $f : \overline{\mathcal{R}} \to \overline{\mathcal{R}}$ where $f = \lambda s.(-s)$ is a strict function. This closed net is represented by three equations:

$$s = int^{\circ}_{s_0}(e, u, s), \;\; e = \Delta(\epsilon)(0)(\neg e), \;\; u = -s.$$

The solution for $e$ is:

$$e = \lambda t. \begin{cases} 0 & \text{if } \lfloor \frac{t}{\epsilon} \rfloor \text{ is even} \\ 1 & \text{otherwise.} \end{cases}$$

The solution for $s$ is the least solution of $s = int^{\circ}_{s_0}(e, -s, s)$. Following the proof of **Fixpoint Theorem I**, let $s^0 = \lambda t.\perp_{\mathcal{R}}$ be the least element, then we have

$$\begin{aligned} s^1 &= int^{\circ}_{s_0}(e, -s^0, s^0) = \lambda t. \begin{cases} s_0 & \text{if } t < \epsilon \\ \perp_{\mathcal{R}} & \text{otherwise,} \end{cases} \\ s^2 &= int^{\circ}_{s_0}(e, -s^1, s^1) = \lambda t. \begin{cases} s_0 & \text{if } t < \epsilon \\ s_0 - \epsilon s_0 & \text{if } \epsilon \le t < 2\epsilon \\ \perp_{\mathcal{R}} & \text{otherwise,} \end{cases} \\ &\vdots \end{aligned}$$

$$s^{k+1} = int^{\circ}_{s_0}(e, -s^k, s^k) = \lambda t. \begin{cases} s_0 & \text{if } t < \epsilon \\ s_0 - \epsilon s_0 & \text{if } \epsilon \leq t < 2\epsilon \\ \vdots & \\ (\Sigma_{i=0}^{k}(-1)^i C_k^i \epsilon^i)s_0 & \text{if } k\epsilon \leq t < (k+1)\epsilon \\ \perp_{\mathcal{R}} & \text{otherwise.} \end{cases}$$

Let $s = \bigvee_{\overline{\mathcal{R}}^{\mathcal{R}^+}}\{s^k\}$. Then $s = \lambda t.s^{\lfloor \frac{t}{\epsilon} \rfloor + 1}(t)$ is the least solution of the equation $s = int^{\circ}_{s_0}(e, -s, s)$. The limiting semantics of the net for $s$ is $s^* = \lambda t. \lim_{\epsilon \to 0} s(t) = \lambda t. \lim_{\epsilon \to 0}(\Sigma_{i=0}^{k}(-1)^i \frac{k!}{i!(k-i)!}\epsilon^i)s_0$ where $k = \lfloor \frac{t}{\epsilon} \rfloor$, i.e., $s^* = \lambda t.(\Sigma_{i=0}^{\infty}(-1)^i \frac{t^i}{i!})s_0 = \lambda t.s_0 e^{-t}$, which is the solution of $\dot{s} = -s$.

Some remarks follow about this semantics of constraint nets with temporal integration.

First, limiting semantics only applies to a closed parameterized net and is not composite. For a constraint net with more than one temporal integrator, we will use a single infinitesimal for all the temporal integrators.

Second, temporal integration in constraint nets is defined on any time structure, discrete or continuous, and any vector space, numerical or symbolic.

Third, in general, a set of differential equations can have no solution or more than one solutions. The limiting semantics produces a unique solution in any case, which might not be well-defined. For example, $\dot{x} = 2\sqrt{x}$ with $x(0) = 0$ on dynamics structure $\mathcal{D}(\mathcal{R}^+, \overline{\mathcal{R}})$ has infinitely many solutions; two significant ones are $x = \lambda t.0$ and $x = \lambda t.t^2$. However, the limiting semantics gives only $x = \lambda t.0$. For another example, $\dot{x} = \frac{1}{2x}$ with $x(0) = 0$ on dynamics structure $\mathcal{D}(\mathcal{R}^+, \overline{\mathcal{R}})$ has two normal solutions $x = \lambda t.\sqrt{t}$ and $x = -\lambda t.\sqrt{t}$. However, the limiting semantics gives an undefined one $x = \lambda t. \perp_{\mathcal{R}}$. In the next chapter, we will come back to this issue and discuss the conditions under which the constraint net produces the "correct" solution.

We can also define three variations of temporal integration: (1) temporal integration with bounds, (2) temporal integration with reset, and (3) integration against another trace on domain $\overline{\mathcal{R}}$.

A *bounded temporal integration*, denoted $\int_{\langle m,M \rangle}(s_0)$, ensures that the output values at all time points are between $m$ and $M$, i.e., $\forall u, t, m \leq \int_{\langle m,M \rangle}(s_0)(u)(t) \leq M$. We can realize this restriction by simply letting

$$int_{s_0}(u, s) = \min(\max(\delta(s_0)(s) + dt \cdot \delta(0)(u), m), M)$$

where "min" and "max" are strict continuous extensions of conventional "min" and "max," respectively.

A *reset temporal integration*, denoted $\int_r(s_0)$, is a transduction of two arguments with the second argument as an event input. $\int_r(s_0)(u, c)$ sets the output value back to $s_0$ whenever there is an event at $c$. A reset temporal integration can be realized as follows. Let

$$int_{s_0}(u, c, s) = cond(c, \delta(0)(c), \delta(s_0)(s) + dt \cdot \delta(0)(u), s_0)$$

where *cond* is the conditional function defined in Equation 3.1. The reset temporal integration $\int_r(s_0)$ can be computed by a module $CN(\{u, c\}, s)$ where $CN$ denotes the following two equations:

$$s = int^{\circ}_{s_0}(e, u, c, s), \;\; e = \Delta(\epsilon)(0)(\neg e)$$

where $\epsilon > 0$ is an infinitesimal.

A *trace-based temporal integration*, denoted $\int_t(s_0)$, is a transduction of two arguments with the second argument as a trace on domain $\overline{\mathcal{R}}$. $\int_t(s_0)(u, v)$, also denoted $\int(s_0)(u)d(v)$, integrates $u$ against the changes of $v$. A trace-based temporal integration can be realized as follows. Let

$$int_{s_0}(u, v, s) = \delta(s_0)(s) + dv \cdot \delta(0)(u)$$

where

$$dv = \lambda t. \begin{cases} 0 & \text{if } t = \mathbf{0} \\ v(t) - v(pre(t)) & \text{otherwise.} \end{cases}$$

The trace-based temporal integration $\int_t(s_0)$ can be computed by a module $CN(\{u, v\}, s)$ where $CN$ denotes the following two equations:

$$s = int^{\circ}_{s_0}(e, u, v, s), \;\; e = \Delta(\epsilon)(0)(\neg e)$$

where $\epsilon > 0$ is an infinitesimal.

## 4.3 Summary

We have presented CN, a formal model for hybrid dynamic systems. The syntax of CN is graphical and modular, and the semantics of CN is denotational and composite. The modular aspect of CN not only provides hierarchical structures of system composition, but also provides a simple and general concept for nondeterminism. The fixpoint semantics provides a rigorous and straightforward interpretation for the meaning of CN. Furthermore, parameterized nets and temporal integration increase the representational power of CN. As a result, CN can be used to model a discrete/continuous hybrid dynamic system with various event-driven components, while events can be generated and synchronized within the system. In the next chapter, we will focus on some typical types of event computation and then discuss modeling aspects of CN via examples.

# Chapter 5

# Modeling in Constraint Nets

A dynamic system is defined on a dynamics structure $\mathcal{D}(\mathcal{T}, A)$ where $\mathcal{T}$ is a time structure and $A$ is a domain structure; the time and domain structures can be either continuous or discrete. Table 5.1 shows examples of the four basic types of model for dynamic systems. We call a dynamic system composed of components of more than one basic type a *hybrid system*.

We have developed Constraint Nets (CN) as a formal model for hybrid dynamic systems. A hybrid dynamic system consists of modules with different time structures, with its domain structure multi-sorted. A typical hybrid domain structure would include a continuous domain $\overline{\mathcal{R}}$ and a discrete or finite domain $\overline{S}$, with associated functions. A typical reference time for a hybrid dynamic system is the set of nonnegative real numbers $\mathcal{R}^+$. Event-driven modules can be associated with different clocks, characterizing different sample time structures generated by event traces. An event trace can be either of fixed sampling rate, or created by some event generator that responses to changes of its inputs. Multiple event traces can also be combined to generate other event traces. Typical event interactions are "event or," "event and," and "event select" that can be defined in terms of event logics. With event logic modules, asynchronous components can be coordinated.

In this chapter, we first focus on some general issues on event control logics and typical event generators and synchronizers. We then illustrate constraint net modeling via an example

Table 5.1: Basic types of model for dynamic systems

| *Dynamic Systems* | *Discrete Time* | *Continuous Time* |
|---|---|---|
| *Discrete Domain* | Finite State Machines | Asynchronous Circuits |
| *Continuous Domain* | Difference Equations | Differential Equations |

that characterizes the features of CN. Finally, we discuss the power of CN in terms of both discrete and continuous computation.

## 5.1 Event Generators and Synchronizers

Introducing event-driven transductions makes a simple and unitary model for arbitrary event-triggered components as well as for various components with fixed sampling rates. Furthermore, events can be generated and synchronized within the model. In this section, we discuss some typical event generators and synchronizers for modeling, programming and design.

### 5.1.1 Event generators

An *event generator* is a transduction whose output is an event trace. For example, $e = \Delta(t_s)(0)(\neg e)$ is an event generator whose output is an event trace of fixed sampling rate. There are event generators with its output capturing the changes of its input. For example, a transition is generated whenever a certain property becomes true.

We introduce some basic modules that will be used mostly for event control.

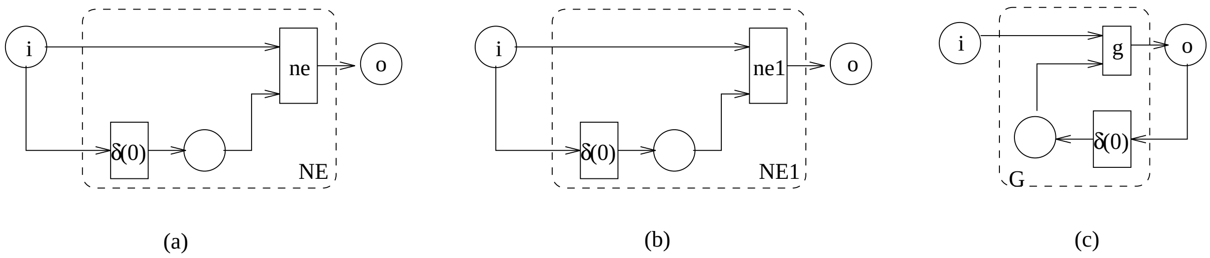


Figure 5.1: Basic modules for event logics

Let *cond* be the conditional function defined in Equation 3.1.

- Module $NE(i, o)$ (Figure 5.1(a)) is composed of a unit delay and a transliteration $ne$ where $ne : \overline{\mathcal{B}} \times \overline{\mathcal{B}} \to \overline{\mathcal{B}}$ is defined as $ne(x, y) = cond(x, y, 0, 1)$.
- Module $NE1(i, o)$ (Figure 5.1(b)) is the same as $NE(i, o)$ except that $ne$ is replaced by $ne1 : \overline{\mathcal{B}} \times \overline{\mathcal{B}} \to \overline{\mathcal{B}}$, $ne1(x, y) = cond(x, y, 0, cond(x, 1, 1, 0))$.
- Module $G(i, o)$ (Figure 5.1(c)) is composed of a unit delay and a transliteration $g$ where $g : \overline{\mathcal{B}} \times \overline{\mathcal{B}} \to \overline{\mathcal{B}}$ is defined as $g(x, y) = cond(x, 0, y, \neg y)$.

(As a matter of fact, both $ne$ and $g$ are $\oplus$, an "exclusive or".) If the reference time is not discrete, unit delays in these modules are performed at a fast (relative to its inputs and/or outputs) fixed sampling rate.

Note that both $NE$ and $NE1$ are nonintermittent and right-continuous. Furthermore, $G$ is an event generator, and any cascade connection to $G$ is an event generator. For example, "rising transition" — an event generator that generates an event whenever its input changes from 0 to 1 — is a cascade connection of $NE1$ to $G$, i.e., $G \circ NE1$.

### 5.1.2 Event synchronizers

An *event synchronizer* is a transduction that maps event traces to new event traces. For example, "event or" (Figure 3.2) is an event synchronizer that merges events in its two input traces as long as no two events happen at the same time.

Now let us consider "event and" (Figure 5.2), another important event synchronizer. The

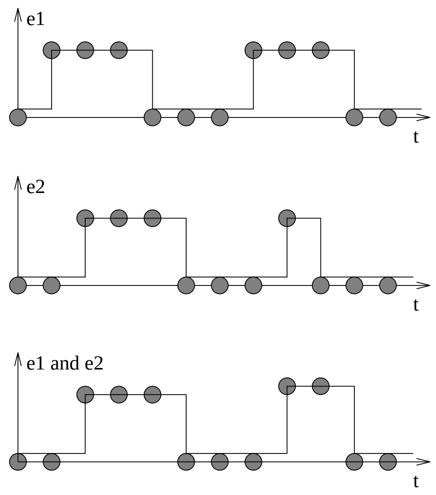


Figure 5.2: Event logic for "and"

Muller C-element [Sut89] acts as the "and" for events: if both of its inputs are of the same value, the output and its next state are copies of that value, otherwise the output and its next state are unchanged. The Muller C-element can be modeled as a state automaton (Figure 4.1) with a state transition function $mc : \overline{\mathcal{B}} \times \overline{\mathcal{B}} \times \overline{\mathcal{B}} \to \overline{\mathcal{B}}$, $mc(i_1, i_2, s) = cond(i_1, i_2, i_1, s)$. The Muller C-element is a module with $i_1, i_2$ and $s'$ as the interface, i.e., a Mealy machine. We can verify that the transduction of the Muller C-element is indeed nonintermittent and right-continuous; therefore, its output is an event trace as long as its inputs are event traces.

We should also notice, according to the definition, that the Muller C-element works as "event and" only for inputs with the following properties:

1. both inputs start at the same value (0 or 1), and
2. the order of events in two inputs are paired such that exactly one event in each pair is produced by one of its inputs.

Only after an event takes place on both of its inputs will the output produce an event, i.e., an event in the output corresponds to the second event in a pair of input events. The Muller C-element generalizes easily to three or more inputs. Such elements are also called *rendezvous* elements [Sut89].

Although the absolute value (from 0 to 1, or 1 to 0) of a transition in a single event trace does not matter, the value relative to other related traces does matter. Thus, it is sometimes important to invert transition signals. We use the standard "and" logic symbol with a "C" inside it to represent Muller C-elements and "bubbles" on input or output ports to represent inversions.

"Event and" elements have been used to coordinate asynchronous events in distributed systems [Sut89]. Consider a simple 1-buffered producer-consumer problem. Both producer and consumer are processes that repeat the following two steps: ask the synchronizer to grant permission for an action (to produce or to consume), and then whenever the request is granted, do the action (the producer produces or the consumer consumes a product).

In Figure 5.3, $R1$ is the request from the producer and $R2$ is the request from the consumer. We use clock $C1$ to grant the producer and clock $C2$ to grant the consumer. Assume that either producing or consuming takes time $\tau$. Two negated Muller C-elements (with initial state 0) and two transport delays are used to synchronize events.

Requests from $R1$ and $R2$ may arrive asynchronously. Given that $R1$ starts at 0 and $R2$ starts at 1, we can check by hand that $C1$ generates a new event iff there is a transition at $R1$ and the buffer is empty ($C1 = C2$). $C2$ generates a new event iff there is a transition at $R2$ and the buffer is full ($C1 \neq C2$). In Part II, we will provide formal specification languages for declaring desired properties of a given system and explore formal verification methods for checking the correctness of the given system.

"Event filter" is an event synchronizer that selects events from its two event inputs according to the value in its first input. Figure 5.4 is a module for an "event filter" element that is composed of basic modules $NE$ and $G$ as well as a transliteration $f$ defined as $f(b, x, y) = cond(b, 0, x, y)$.

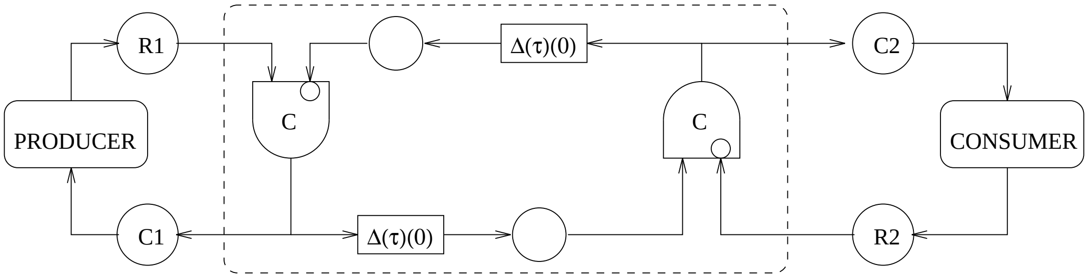


Figure 5.3: A producer-consumer event synchronizer

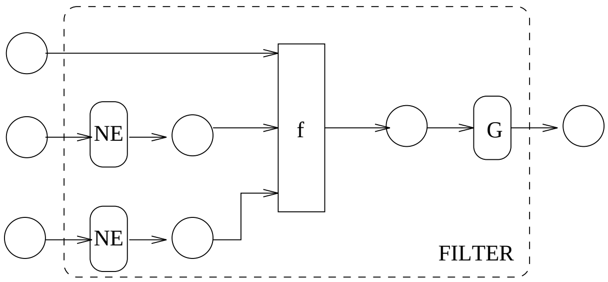


Figure 5.4: An event filter

Similarly, "event select" is an event synchronizer that steers events in its second input to one of two of its outputs according to the value in its first input. Figure 5.5 is a module for an "event select" element that is composed of basic modules $NE$ and $G$ as well as a transliteration $s$ defined as $s(b, x) = cond(b, 0, \langle x, 0\rangle, \langle 0, x\rangle)$.

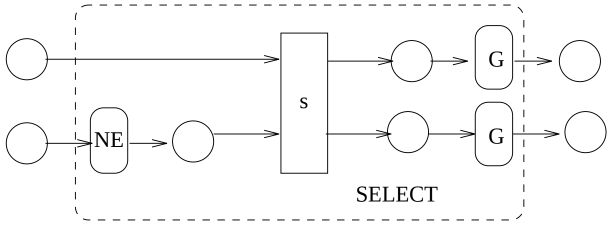


Figure 5.5: An event select

FILTER (resp. SELECT) can be extended to three or more input (resp. output) event traces.

In this way, we can also model all the event logic elements described in Sutherland's paper [Sut89], such as "Switch," "Event-Controlled Storage Element" (ECSE), "Toggle," "Arbiter," etc.

## 5.2 Modeling Hybrid Systems

A robotic system is a hybrid system in general, which is an integration of a plant with continuous dynamics, a continuous/discrete hybrid controller, and a possibly changing environment (Figure 1.1).

Let us consider an example, a car-like maze traveler. Suppose a maze is composed of blocks of bounded size placed on an unbounded plane. A car-like robot with two touch sensors, forward sensor $SF$ and right-side sensor $SR$ (Figure 5.6(a)), is required to traverse the maze from west to east (Figure 5.6(b)).

As any robotic system, this system consists of a plant, a controller and an environment. The plant is the body of the car-like robot, which can move forward/backward by setting a speed $v$ and can make turns by steering two front wheels to some angle $\alpha$. The environment is the maze, and the controller connects sensor signals and motor commands (Figure 5.7).

The plant of the robot has been modeled as a constraint net in Figure 1.2 on dynamics structure $\mathcal{D}(\mathcal{R}^+, \overline{\mathcal{R}})$. The environment can be modeled as a transliteration that maps any

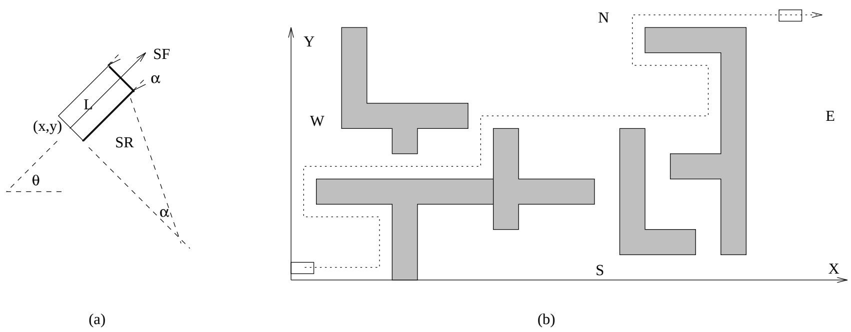


Figure 5.6: (a) The car-like robot (b) Traveling through a maze

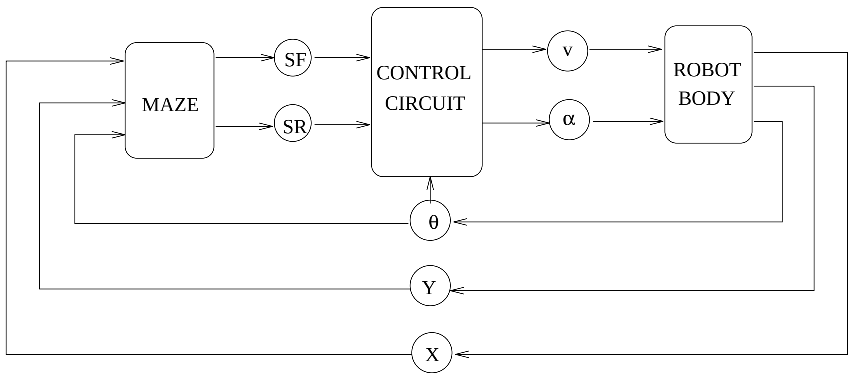


Figure 5.7: The maze traveler robotic system

configuration of the car-like robot ($\langle x, y, \theta \rangle \in \overline{\mathcal{R}} \times \overline{\mathcal{R}} \times \overline{\mathcal{R}}$) to sensor signals ($SF, SR \in \overline{\mathcal{B}}$) over time (continuously). If the robot is facing (or to the left of) a wall within some distance, the forward (or right) sensor $SF$ (or $SR$) is on, i.e., $SF = 1$ (or $SR = 1$); otherwise it is off, i.e., $SF = 0$ (or $SR = 0$).

The simplest strategy for a robot to move out of a maze is to follow a wall with one side (e.g., the right side) [Ad81]. Starting at any position with the correct heading $|\theta| < \delta$ (e.g., east), the robot is always moving forward until it hits a wall ($SF$ becomes on). Whenever it hits a wall, it turns left ($\theta = \theta + \frac{\pi}{2}$), with its right side against the wall, and moves forward. Whenever the right side is off the wall ($SR$ becomes off) and the heading is not correct ($|\theta - \frac{\pi}{2}k| < \delta, k > 0$), it turns right ($\theta = \theta - \frac{\pi}{2}$), again with its right side against the wall, and moves forward, $\cdots$. This strategy can be modeled as a transliteration that maps the heading of the car and the sensor signals $\langle \theta, SF, SR \rangle$ to a control signal $c \in \{0, -1, 1\}$ where 0 means "continuing in the current direction," $-1$ means "turning right" and 1 means "turning left:"

$$
\begin{array}{rll}
|\theta - \frac{\pi}{2}k| < \delta, k > 0 : \text{if} & SR & = 0 \text{ then } c = -1 \\
\text{elseif} & SF & = 1 \text{ then } c = 1 \\
\text{else} & c & = 0 \\
|\theta| < \delta : \text{if} & SF & = 1 \text{ then } c = 1 \\
\text{else} & c & = 0
\end{array}
$$

We will see that (in Part II) if the car is not in a closed block and if there is always enough space for the robot to turn, the robot will move in the correct direction ($|\theta| < \delta$) persistently.

This strategy is made in discrete time, but without any fixed sampling rate, since it may not be known how long the car takes to turn to the next direction, and how long before it hits a wall or moves off a wall. Therefore, the strategy should be event-driven. There are three types of event: (1) $\theta$ enters $\{(\frac{\pi}{2}k - \delta, \frac{\pi}{2}k + \delta) | k = 0, 1, 2 \ldots\}$, (2) $SF$ changes from 0 to 1 or (3) $SR$ changes from 1 to 0. "Rising transition" elements are used to generate these events and "event or" elements are used to synchronize these events. An event generator (Figure 5.8(a)) is created by combining these elements.

As a result, the control circuit is composed of the event generator, the event-driven strategy module and an actualizer (Figure 5.8(b)), which, for simplicity, is set to be $v = 1$ and $\alpha = \frac{\pi}{4}c$.

Even though it is a simple hybrid system, in order for the system to work properly, we have to consider the interface between discrete and continuous domains carefully.

- The "event or" logic works correctly only when no two events happen at the same time.

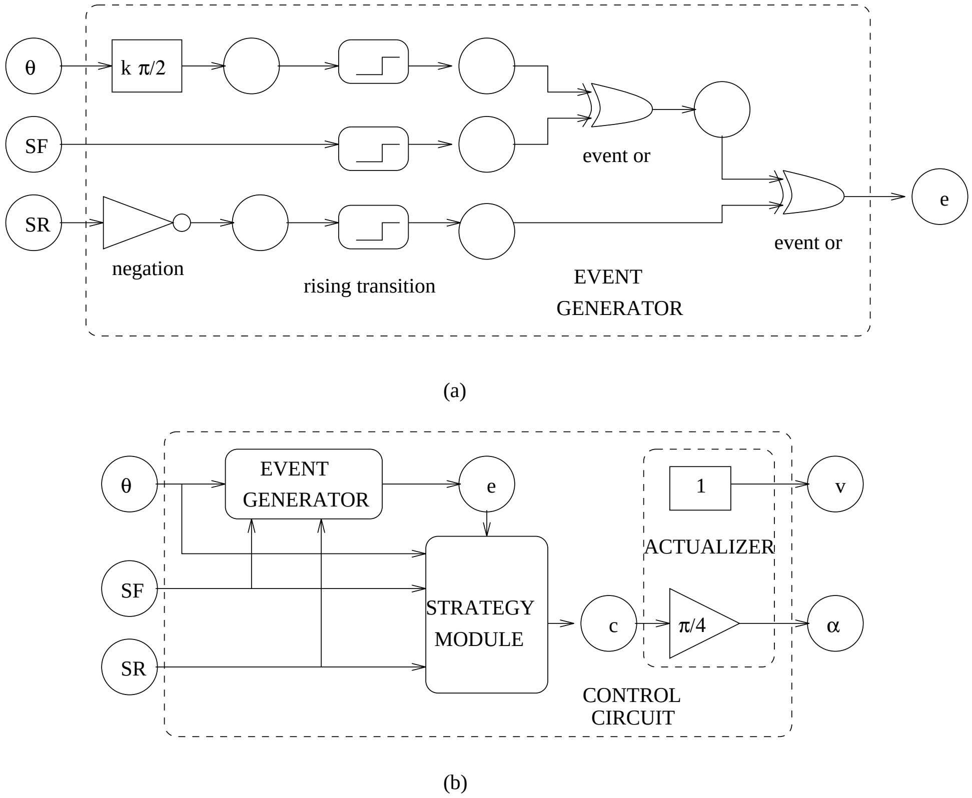


Figure 5.8: (a) Event generator (b) Control circuit

In this example, we assume that the sizes of the blocks and the spaces between the blocks are much larger than the size of the car.

- The error angle $\delta$ should be assigned based on the sizes of the blocks and the turning radius. Given that the steering angle $\alpha$ is $\pi/4$ and the length of the car is $L$, the turning radius $R$ becomes $L$ (since $R = L/\tan\alpha$). Let the maximum size of blocks be $M$. We have $\delta < L/M$ so that the car will not hit the right wall when it moves forward with some error $\delta$ in its heading.

- The front and right sensor ranges are designed according to the the sizes of the blocks, the turning radius and the error angle. Let the turning radius be $L$ and $\delta < L/M$. If the initial distance from the right wall is $L$, the distance from the right wall will always be less than $2L$ when it moves forward with some error $\delta$ in its heading. Therefore, suppose $DF$ is the distance between the front of the car and the front wall, and $DR$ is the distance between the right side of the car and the right wall, we have $SF = DF \leq L$ and $SR = DR < 2L$ (so that $SR$ will not be off because of error $\delta$ in its heading).

These problems seem particular to this special design and the solutions seem *ad hoc*. However, similar situations, such as choosing errors, thresholds, gains, sampling rates, etc., would be encountered in the design of every hybrid system. In Appendix C, we will study more examples of hybrid system design and analysis. In Part III, we will design a more complex control system for the car-like robot.

## 5.3 Power of Constraint Nets

Any computational model is suitable for representing a certain type of computation. For example, Turing machines are used to represent sequential computation and analog circuits are used to represent parallel and continuous computation. The Constraint Net model (CN) is an abstraction of models for dynamic systems. Even though CN is inherently parallel, sequential computation can also be modeled. In this section, we first focus on sequential computation in CN and then discuss continuous computation in CN.

### 5.3.1 Sequential computation

We model any sequential computation as a module with an event input indicating the start of a computation and an event output indicating the end of the computation (Figure 5.9). The time duration between the start and the end of the computation is variable, depending on the input data. We call such a module a *sequential module*.

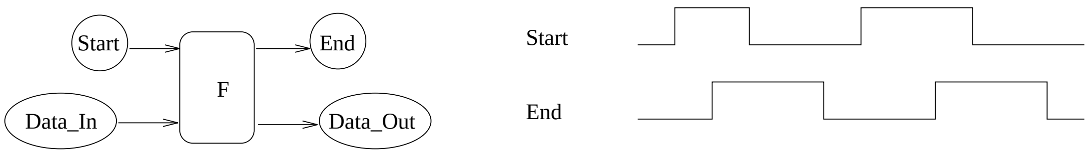


Figure 5.9: A sequential module

A transliteration $f$ is modeled as a sequential module with $End = Start$ and $Data\_Out = f(Data\_In)$, i.e., there is no time delay in a transliteration. A functional composition of two sequential computations is modeled as a cascade connection of the two sequential modules (Figure 5.10).

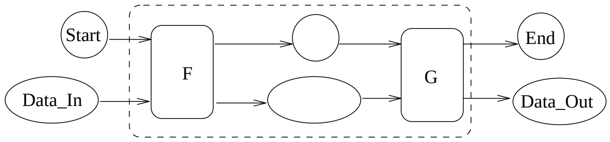


Figure 5.10: A functional composition $G \circ F$

Let $g : A \rightarrow A'$ and $h : \overline{\mathcal{N}} \times A \times A' \rightarrow A'$ be functions. A *recursive function* $f : \overline{\mathcal{N}} \times A \rightarrow A'$ based on $g$ and $h$ can be defined as $f(0, x) = g(x), f(n+1, x) = h(n, x, f(n, x)))$. Given that $g$ and $h$ are computed by sequential modules G and H, respectively, a sequential module for $f$ can be constructed as follows.

Let COUNTER be a module with two event inputs and one output on domain $\overline{\mathcal{N}}$. The first event input resets the output to zero and the second event input increases the output value by one. COUNTER($\{c1, c2\}, n$) (Figure 5.11) is composed of module $NE$ (Figure 5.1(a)), two transliterations *suc* and *cond*, and an event-driven unit delay $\delta^{\circ}(0)$, where

$$suc(n) = \begin{cases} \perp_{\mathcal{N}} & \text{if } n = \perp_{\mathcal{N}} \\ n+1 & \text{otherwise} \end{cases}$$

is a successor function.

The sequential module for $f$ (Figure 5.12) is composed of sequential modules G and H, modules COUNTER, FILTER and SELECT, and transliteration *cond*. Unit delays are also

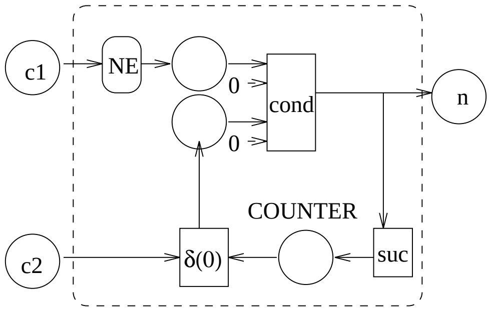


Figure 5.11: An event counter

added to avoid algebraic loops. We can see that in order to compute $f(n, x)$, the sequential module G will be triggered initially and the sequential module H will be triggered up to $n - 1$ times.

A function $f : A \rightarrow \overline{\mathcal{N}}$ is defined using the *minimization operation* on a function $g : \overline{\mathcal{N}} \times A \rightarrow \overline{\mathcal{N}}$ if:

$$f(x) = \begin{cases} \min\{n | g(n, x) = 0\} & \text{if the set is not empty} \\ \perp_{\mathcal{N}} & \text{otherwise.} \end{cases}$$

Thus, $f(x)$ is defined as the smallest $n$ for which $g(n, x) = 0$ if there is such an $n$; it is otherwise undefined. Given that $g$ is computed by a sequential module G, a sequential module for $f$ can be constructed as in Figure 5.13. If $f(x) = n \in \mathcal{N}$, the sequential module G will be triggered $n + 1$ times, otherwise G will be triggered infinitely many times and there will be no event generated in $End$.

Therefore, given a set of basic functions and their sequential modules, the set of functions closed under functional composition, recursive schemes and minimization operations can be computed by sequential modules. In fact, this set is large enough to include all the computable functions given a small set of basic functions. It is well known that the set of Turing computable functions is equal to the set of partial recursive functions. A function $f$ is defined *partial-recursively* iff [Yas71]:

- it is the constant 0, the successor function $suc$, or a projection function $proj_i$,

$$proj_i(x_1, \ldots, x_i, \ldots, x_n) = x_i;$$

  or

- it is defined as a functional composition of functions defined partial-recursively; or

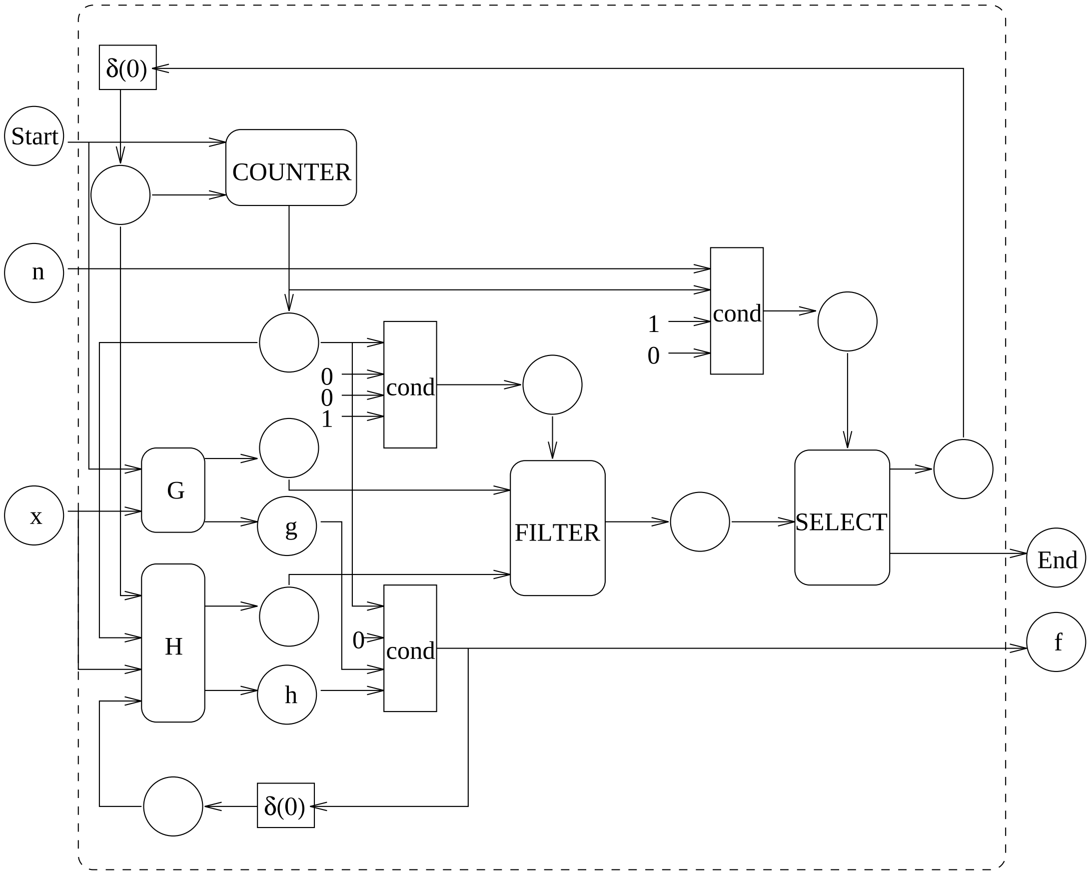


Figure 5.12: A sequential module for a recursive function

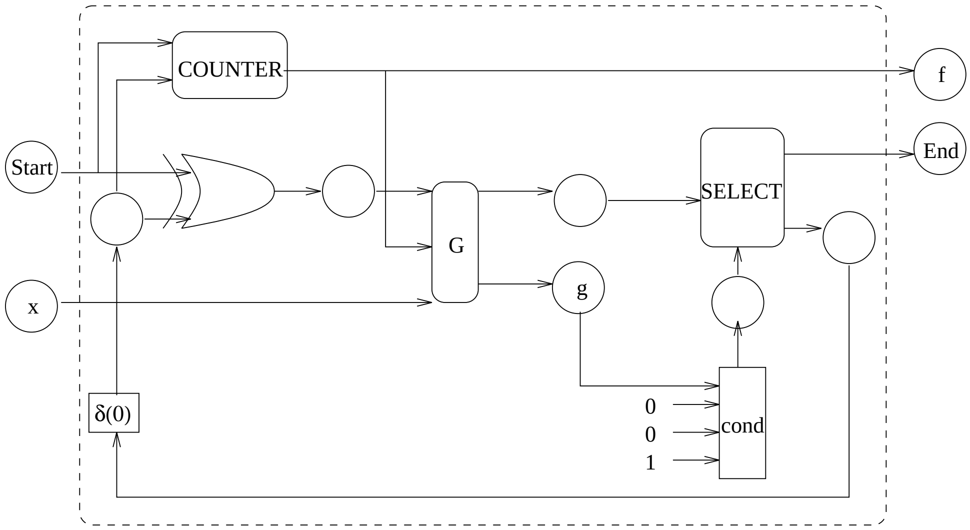


Figure 5.13: A sequential module for the minimization operation

- it is defined as a recursive function based on functions defined partial-recursively; or
- it is defined using the minimization operation on a function defined partial-recursively.

A function $f$ is a *partial recursive function* if it equals a function that is defined partial-recursively.

**Theorem 5.3.1** *Let* $\Sigma_n = \langle \{n\}, \{\mathtt{0}, \mathtt{suc}, \mathtt{cond}\} \rangle$ *be a signature. A partial recursive function can be computed by a sequential module on* $\Sigma_n$*-dynamics structure* $\mathcal{D}(\mathcal{N}, \overline{\mathcal{N}})$ *where* $\overline{\mathcal{N}}$ *denotes the* $\Sigma_n$*-domain structure* $\langle \{\overline{\mathcal{N}}\}, \{0, suc, cond\} \rangle$.

Finishing up this section on sequential computation, we give two more examples used mostly in concurrency and real-time models.

- *Internal choice* $A + B$: either $A$ or $B$ will be computed. Figure 5.14 is a sequential module for this scheme, where $id$ is an event-driven transliteration of an identity function $id = \lambda x.x$ and location $n$ is a hidden input with Boolean domain.

  Internal choices are often used for modeling nondeterminism in concurrent systems.

- *External choice* $C \to A|D \to B$: if an event in $C$ comes before an event in $D$, $A$ is computed, otherwise $B$ is computed. Figure 5.15 is a sequential module for this scheme,

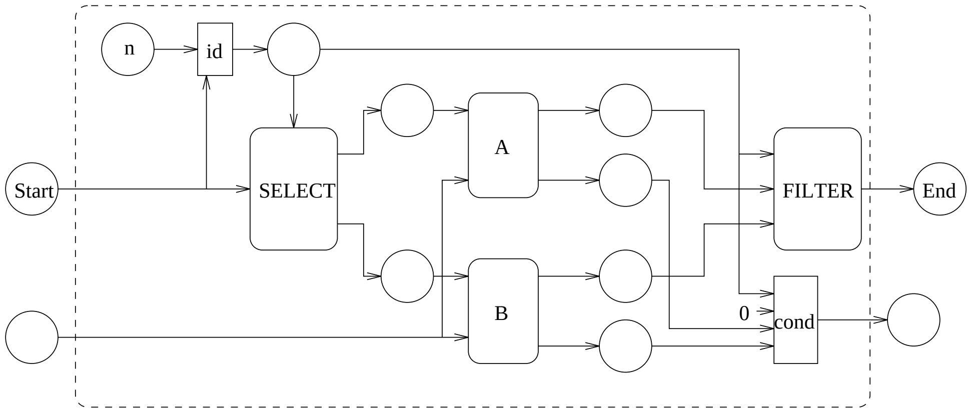


Figure 5.14: A sequential module for internal choice $A + B$

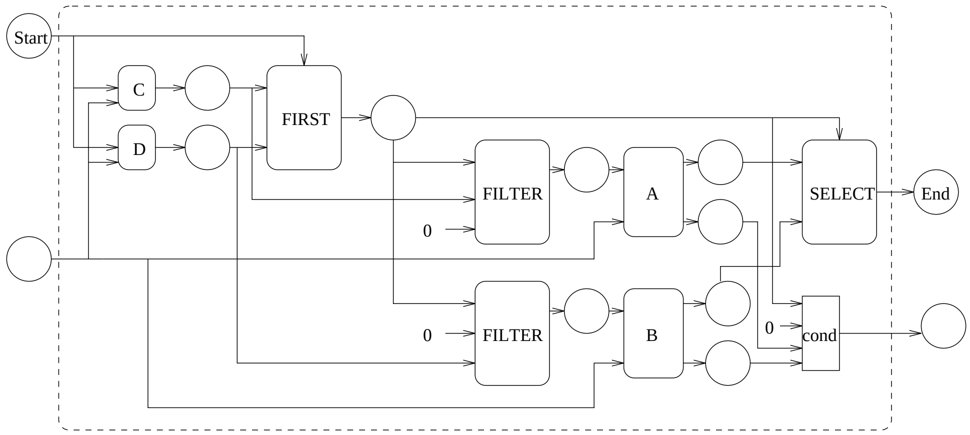


Figure 5.15: A sequential module for external choice $C \rightarrow A|D \rightarrow B$

where FIRST is a module that outputs 0 if an event in $C$ comes first and 1 if an event in $D$ comes first.

Module FIRST (Figure 5.16) is composed of a transliteration *cond* for resetting the state whenever there is an event in $Start$, a transliteration of state transition function $f$ defined as $f(\langle c, d\rangle, \langle s_c, s_d\rangle) = \langle s_c \vee c \wedge \neg s_d, s_d \vee d \wedge \neg s_c\rangle$, and a transliteration of an output function $g$ defined as $g(\langle s_c, s_d\rangle) = \neg s_c \wedge s_d$. Note that events in $c$ have higher priorities for this definition of $g$, i.e., if events in $c$ and $d$ come at the same time, the event in $c$ will be selected.

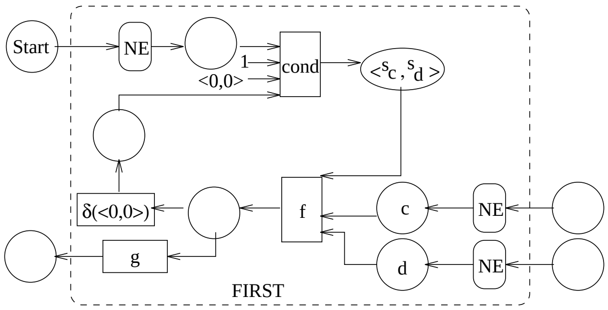


Figure 5.16: The FIRST module

External choices are often used for modeling time-out in real-time systems. For example, if $C$ is a module that generates time-out events (Figure 5.17), $C \to A|D \to B$ means that if $D$ generates an event before time-out, $B$ will be executed, otherwise $A$ will be executed.

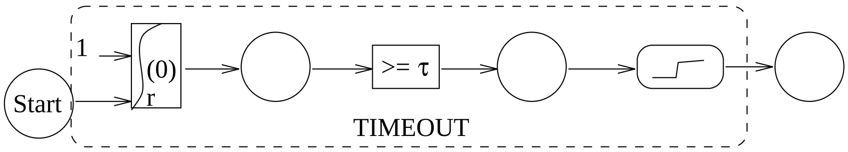


Figure 5.17: The TIMEOUT module

### 5.3.2 Analog computation

We have seen that the Constraint Net model (CN) can represent sequential computation as well, by using events to coordinate the order of computation. However, in general, CN is used for modeling computation over time, i.e., relationships between input traces and output traces. If a constraint net $CN$ is closed, the semantics of $CN$ is simply a trace, i.e., a function of time. Many functions that are not easy to model in sequential computation are easy to compute as traces. For example, $\lambda t.Ce^{kt}$ is the solution of a constraint net $x = \int(C)(kx)$; $\lambda t.\langle\sin(t), \cos(t)\rangle$ is the solution of a constraint net $x = \int(0)(y), y = \int(1)(-x)$. In the rest of this section, we will ask two questions. First, given a set of basic functions on $\mathcal{R}$, say $+$ and $\cdot$, what is the set of traces that can be represented as solutions of differential equations? Second, given differential equations modeled in constraint nets, what is the relationship between the semantics of constraint nets and the solutions of the differential equations?

The first question was answered by Shannon. Here we present a variation of the results in [Sha41]. Let $\mathcal{T} = [t_0, t_1] \subset \mathcal{R}$. A trace $x : \mathcal{T} \to \mathcal{R}$ can be obtained as a solution of a set of differential equations composed of only $+$ and $\cdot$, iff $x = x_1$ can be written as:

$$\dot{x_k} = P_k(t, x_1, \ldots, x_n) = \Sigma_i a_i t^{i_0} x_1^{i_1} \ldots x_n^{i_n} \qquad k = 1, \ldots, n \tag{5.1}$$

where the $i$'s denote natural numbers. We use $P$'s to denote polynomial functions. The question is then reduced to: what is the set of functions that can be written in Equation 5.1? It has been shown [Sha41] that this set is equal to the set of non-hypertranscendental functions.

A function $x = \lambda t.f(t)$ is *non-hypertranscendental* iff it can be written as

$$P(t, x, \dot{x}, \ddot{x}, \ldots, x^{(n)}) = \Sigma_i a_i t^{i_0} x^{i_1} (\dot{x})^{i_2} (\ddot{x})^{i_3} \ldots (x^{(n)})^{i_{n+1}} = 0. \tag{5.2}$$

**Proposition 5.3.1** [Sha41] *Equations 5.1 and 5.2 are equivalent, i.e., a function written in one form can be transformed into another.*

Most common analytic functions are non-hypertranscendental [Sha41] such as exponential and logarithmic, trigonometric and hyperbolic, Bessel functions, elliptic functions, probability functions, and solutions of an algebraic equation in terms of a parameter. Non-hypertranscendental functions are also closed under various operations.

**Proposition 5.3.2** [Sha41] *If $x = \lambda t.f(t)$ is non-hypertranscendental, then its derivative $y = \lambda t.f'(t)$, its integral $z = \lambda t.\int_{t_0}^{t} f(t)dt$, and its inverse $w = \lambda t.f^{-1}(t)$ are non-hypertranscendental.*

**Proposition 5.3.3** [Sha41] *Non-hypertranscendental functions are closed under functional composition.*

The second question is that given a trace as a solution of a set of differential equations, can that trace be computed as the limiting semantics of the constraint net representing the set of differential equations? This question is further decomposed into two questions: first, does the constraint net have a well-defined limiting semantics? second, does the set of differential equations have a unique solution? If the answers to both questions are positive, the trace can be computed by the constraint net.

**Proposition 5.3.4** *Given a constraint net of differential equations $\dot{x}_k = f_k(\vec{x}), k = 1, \ldots, n$ with $x_k(t_0) \in \mathcal{R}$ and $f_k : \mathcal{R}^n \to \mathcal{R}$ as partial or total functions, and given that all $f_k$ are smooth at $\vec{x}(t_0)$, the limiting semantics of the constraint net, based on the forward Euler method, is well-defined over $\mathcal{T} = [t_0, t_1]$ for some $t_1 > t_0$. In particular, $x = \lambda t.\Sigma_{n=0}^{\infty} \frac{x^{(n)}(t_0)}{n!}(t - t_0)^n$.*

If $f_k$ is a polynomial function, then $f_k$ is smooth over $\mathcal{R}^n$. If, in addition, the initial value for $\vec{x}$ is well-defined, the limiting semantics of the constraint net is well-defined.

It has been shown that a sufficient condition for differential equations $\dot{\vec{x}} = \vec{f}(\vec{x})$ to have a unique solution is the Lipschitz condition [MA86]. The Lipschitz condition is defined as follows. Given $\langle \mathcal{R}^n, d \rangle$ as a metric space, we say that $\vec{f} : \mathcal{R}^n \to \mathcal{R}^n$ satisfies a *Lipschitz condition uniformly with respect to* $t \in [t_0, t_1]$ iff there exists a number $K > 0$ such that

$$d(\vec{f}(\vec{x}(t)), \vec{f}(\vec{y}(t))) \leq K d(\vec{x}(t), \vec{y}(t)) \quad \text{for all} \quad t \in [t_0, t_1].$$

Let $|\vec{x}|_2 = \sqrt{\Sigma_{i=1}^n x_i^2}$ and $d(\vec{x}, \vec{y}) = |\vec{x} - \vec{y}|_2$. If $\vec{f}$ is a linear function, i.e., $\vec{f}(\vec{x}) = A\vec{x}$, $|\vec{f}(\vec{x}(t)) - \vec{f}(\vec{y}(t))|_2 \leq |A|_2 |\vec{x}(t) - \vec{y}(t)|_2$, for all $t$. Therefore, linear functions always satisfy the Lipschitz condition, and linear differential equations always have a unique solution.

A more general result is as follows.

**Theorem 5.3.2** *Let $\Sigma_r = \langle \{r\}, \{+, \cdot\} \rangle$ be a signature. A non-hypertranscendental function that is defined and smooth over a closed segment $\mathcal{T} = [t_0, t_1]$ can be computed by a constraint net of differential equations on $\Sigma_r$-dynamics structure $\mathcal{D}(\mathcal{T}, \overline{\mathcal{R}})$, where $\overline{\mathcal{R}}$ denotes the $\Sigma_r$-domain structure $\langle \{\overline{\mathcal{R}}\}, \{+, \cdot\} \rangle$.*

# Chapter 6

# Behavior Analysis

We have presented the Constraint Net model, its syntax and semantics, and its power in modeling dynamics and computation. In this chapter, we relate systems to their behaviors. We start with some preliminaries in abstract algebra on equivalence and abstraction. We then present a formal definition of behaviors and discuss various properties of behaviors. Finally, we formalize the concept of behavior abstraction at different levels of granularity, and the meaning of system equivalence with respect to a certain type of abstraction.

## 6.1 Abstraction, Quotient and Homomorphism

This section introduces some basic, but important, concepts in abstract algebra. These concepts are related to the question of how to generate an abstraction of a given system.

Intuitively, equivalence induces partitions, and partitions induce abstraction. An *algebraic system* is a set with an associated structure, i.e., a set of functions and relations. A structure that is consistent with a partition can be abstracted to a quotient structure on the partition. An algebraic system $A'$ is a quotient of an algebraic system $A$ if $A'$, with the quotient structure, is a partition of $A$. A quotient of an algebraic system can be considered as an abstraction of the algebraic system. An algebraic system $A$ is homomorphic to an algebraic system $A'$, if there is a surjective (onto) mapping from $A$ to $A'$ that is consistent with the associated structure; it is isomorphic if the mapping has an inverse. On the other hand, a homomorphic mapping induces a partition and a quotient structure. An algebraic system is homomorphic to its quotient. Given algebraic systems $A$ and $A'$, if $A$ is homomorphic to $A'$, $A'$ is isomorphic to the quotient of $A$ induced by the homomorphic mapping. We present these concepts more formally in the rest of this section.

Equivalence relations are characterized as congruences. A binary relation $\simeq_A$ over a set $A$

is a *congruence* iff it is reflexive, transitive and symmetric.

A congruence induces a partition. A *partition* of a set $A$ induced by a congruence $\simeq_A$, written $A/\simeq_A$, is a set of sets $\{A_i\}$ such that (1) $A = \cup_i A_i$ (2) $\forall i \neq j, A_i \cap A_j = \emptyset$ and (3) $a_1 \simeq_A a_2$ and $a_1 \in A_i$ imply $a_2 \in A_i$. We use $[a]$ to denote the set that $a$ belongs to. Intuitively, a partition of a set can be considered as an *abstraction* of the set; $[a]$ is an *abstraction* of $a$.

If a congruence is consistent with a function $f$, it is called an $f$-congruence. Let $f : A \rightarrow A'$ be a function. A congruence $\simeq$ over $A \cup A'$ is an *$f$-congruence* iff $a_1 \simeq a_2$ implies $f(a_1) \simeq f(a_2)$. Given $f : A \rightarrow A'$ and $\simeq$ as $f$-congruence, an abstraction of $f$ can be defined on the partition of $A$ and $A'$ induced by the $f$-congruence. Let $f : A \rightarrow A'$ be a function and $\simeq$ be an $f$-congruence over $A \cup A'$. The *quotient function* of $f$ w.r.t. $\simeq$, written $f^{\simeq} : A/\simeq \rightarrow A'/\simeq$, is defined as $f^{\simeq}([a]) = [f(a)]$. We also say that function $f$ is *abstractable* w.r.t. $\simeq$ when $\simeq$ is an $f$-congruence.

The concepts of abstraction and quotient structures can be extended to multi-sorted algebra. Let $\Sigma = \langle S, F \rangle$ be a signature and $A$ be an arbitrary $\Sigma$-algebra. A $\Sigma$-*congruence* on $A$ is an $S$-sorted relation $\simeq$, $\simeq = \{\simeq_s\}_{s \in S}$ satisfies

1. for each $s \in S$, $\simeq_s$ is a congruence on $A_s$, and

2. for any $f : s_1, \ldots s_n \rightarrow s \in F$ and all $a_1, a'_1 \in A_{s_1}, \ldots, a_n, a'_n \in A_{s_n}$, if $a_1 \simeq_{s_1} a'_1, \ldots$, and $a_n \simeq_{s_n} a'_n$ hold, then $f^A(a_1, \ldots, a_n) \simeq_s f^A(a'_1, \ldots, a'_n)$. Namely, $\simeq$ is an $f^A$-congruence for all $f \in F$.

Given a $\Sigma$-congruence $\simeq$ on a $\Sigma$-algebra $A$, we can define a quotient algebra $A/\simeq$. Let $\simeq$ be a $\Sigma$-congruence over a $\Sigma$-algebra $A$. The *quotient $\Sigma$-algebra* $A/\simeq$ of $A$ is defined as $(A/\simeq)_s = A_s/\simeq_s$ and for $f : s_1, \ldots, s_n \rightarrow s \in F$, $a_1 \in A_{s_1}, \ldots, a_n \in A_{s_n}$, $f^{A/\simeq}([a_1], \ldots, [a_n]) = [f^A(a_1, \ldots, a_n)]$. The quotient $\Sigma$-algebra $A/\simeq$ of $A$ can be considered as an *abstraction* of $A$ induced by the $\Sigma$-congruence $\simeq$.

The relationship between an algebra and its quotient algebras can be characterized by homomorphism. In general, a homomorphism on $\Sigma$-algebras is defined as follows. Let $A$, $A'$ be two $\Sigma$-algebras. A $\Sigma$-*homomorphism* $h : A \rightarrow A'$ is a family of surjective (onto) mappings $h = \{h_s : A_s \rightarrow A'_s\}_{s \in S}$ such that for each $f : s_1, \ldots, s_n \rightarrow s \in F$ and each $a_1 \in A_{s_1}, \ldots, a_n \in A_{s_n}$, $h_s(f^A(a_1, \ldots, a_n)) = f^{A'}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$. It is a $\Sigma$-*isomorphism* if $h$ is a bijection. If $A'$ is a quotient algebra of $A$, there exists a homomorphism $h$ from $A$ to $A'$ with $h(a) = [a]$. On the other hand, if there is a homomorphism $h$ from $A$ to $A'$, $A'$ is isomorphic to a quotient algebra of $A$. To see this, let us define a congruence $\simeq_h$ on $A$ as follows: $a_1 \simeq_h a_2$ iff $h(a_1) = h(a_2)$.

Since $h$ is a $\Sigma$-homomorphism from $A$ to $A'$, $\simeq_h$ is an $f^A$-congruence on $A$ for any $f \in F$. Therefore, $A'$ is isomorphic to the quotient algebra of $A$ induced by $\simeq_h$. And, $A'$, which is isomorphic to a quotient of $A$, is also considered as an abstraction of $A$.

## 6.2 Behavior Analysis: General Concepts

Now we discuss the relationship between dynamic systems and their behaviors. Intuitively, the behavior of a dynamic system is the set of observable input/output traces of the system. Formally, let $CN(I,O)$ be a module. An input-output pair $\langle i, o\rangle$ is an *observable trace* of $CN(I,O)$ iff $\exists F \in [\![CN(I,O)]\!]$ such that $o = F(i)$. The *behavior* of $CN(I,O)$ is the set of all observable traces of $CN(I,O)$. We will also use $[\![CN(I,O)]\!]$ to denote the behavior of $CN(I,O)$ if no ambiguity arises. We will use $[\![CN]\!]$ as an abbreviation of $[\![CN(I,O)]\!]$ if $I = I(CN)$ and $O = O(CN)$. Two modules are *equivalent*, written $CN_1(I,O) \simeq CN_2(I,O)$, iff they have the same behavior, i.e., $[\![CN_1(I,O)]\!] = [\![CN_2(I,O)]\!]$. For example, two state transition modules are equivalent if they have the same initial state and the same state transition relation. A behavior $\mathcal{B}$ is *deterministic* iff for any pair of traces $\langle i_1, o_1\rangle, \langle i_2, o_2\rangle \in \mathcal{B}$, $i_1 = i_2$ implies $o_1 = o_2$; it is otherwise *nondeterministic*. In general, a module $CN(I,O)$ will exhibit a nondeterministic behavior if there are hidden inputs, i.e., $I \subset I(CN)$.

Two important types of behavior are state-based behavior and time-invariant behavior.

State-based behavior is formalized as follows. Let $\mathcal{B}$ be a behavior. Given any time point $t \in \mathcal{T}$, two traces $v_1, v_2 \in \mathcal{B}$ are *coincident up to* $t$, written $v_1 \simeq_{\leq t} v_2$, iff $\forall t' \leq t$, $v_1(t') = v_2(t')$. Let $[v]_t = \{v'_{|>t} | v' \simeq_{\leq t} v\}$ where $v_{|>t}$ denotes the restriction of $v$ onto $\{t' \in \mathcal{T} | t' > t\}$. $\mathcal{B}$ is *state-based* iff for all $v_1, v_2 \in \mathcal{B}$ and $t \in \mathcal{T}$, $v_1(t) = v_2(t)$ implies $[v_1]_t = [v_2]_t$, i.e., the behavior in the future is fully determined by the current snapshot.

Time-invariant behavior is formalized as follows. Let $\mathcal{B} = \{v | v : \mathcal{T} \to A\}$ be a behavior. For any $a_1, a_2 \in A$, let $a_1 \prec a_2$ iff $\exists v \in \mathcal{B}, t_1 < t_2$ such that $a_1 = v(t_1)$ and $a_2 = v(t_2)$. $\mathcal{B}$ is *time-invariant* iff $\prec$ is transitive, i.e., $\prec$ is independent of time.

A state automaton in Figure 4.1 exhibits a state-based and time-invariant behavior. However, an input/output automaton in Figure 4.4 may not exhibit a state-based and time-invariant behavior. Any state-based and time-invariant behavior of discrete time corresponds to a state transition system.

A *state transition system* is a pair $\langle S, \to\rangle$ where $S$ is a set of states and $\to \subset S \times S$ is a transition relation between two states. For any discrete time $\mathcal{T}$, $v : \mathcal{T} \to S$ is a trace of $\langle S, \to\rangle$ iff $\forall t > \mathbf{0}$, $v(pre(t)) \to v(t)$. A behavior $\mathcal{B}$ *corresponds to* a state transition system $\langle S, \to\rangle$ iff $\mathcal{B}$

is equal to the set of traces of $\langle S, \rightarrow\rangle$. State transition systems can be considered as a compact representation of state-based and time-invariant behaviors.

A *requirements specification* $\mathcal{R}$ for a system $CN(I,O)$ is a set of allowable input/output traces of the system: $\mathcal{R} \subseteq \times_{I \cup O} A_l^{\mathcal{T}}$. $CN(I,O)$ *satisfies* a requirements specification $\mathcal{R}$, written $[\![CN(I,O)]\!] \models \mathcal{R}$, iff $[\![CN(I,O)]\!] \subseteq \mathcal{R}$. With the formal definition of requirements specification, robustness and complexity can be formally defined.

The robustness of systems is defined on parameterized nets. A parameterized system $CN_1^P(I,O)$ is *less robust* than $CN_2^P(I,O)$ w.r.t. a requirements specification $\mathcal{R}$, written $CN_1^P(I,O) \preceq_{\mathcal{R}} CN_2^P(I,O)$, iff $[\![CN_1^P(I,O)]\!](v) \subseteq \mathcal{R}$ implies $[\![CN_2^P(I,O)]\!](v) \subseteq \mathcal{R}$, for all $v \in \times_P D_p$. Two parameterized systems $CN_1^P(I,O)$ and $CN_2^P(I,O)$ are equivalent w.r.t. a requirements specification $\mathcal{R}$, written $CN_1^P(I,O) \simeq_{\mathcal{R}} CN_2^P(I,O)$, iff $CN_1^P(I,O) \preceq_{\mathcal{R}} CN_2^P(I,O)$ and $CN_2^P(I,O) \preceq_{\mathcal{R}} CN_1^P(I,O)$.

Behavioral complexity is defined with respect to some kind of measurement on the size of a dynamic system: the number of transductions, the number of delay elements, or the maximum number of delay elements in any path. Let $|CN|_m$ denote the size of $CN$ w.r.t. measurement $m$. The *complexity of behaviors* satisfying $\mathcal{R}$ w.r.t. $m$, denoted $|\mathcal{R}|_m$, is the minimum realization of the dynamic systems satisfying $\mathcal{R}$ w.r.t. $m$, i.e., $|\mathcal{R}|_m = \min\{|CN|_m\}_{[\![CN(I,O)]\!] \models \mathcal{R}}$.

**Proposition 6.2.1** *If $\mathcal{R}_1 \subseteq \mathcal{R}_2$, $|\mathcal{R}_1|_m \geq |\mathcal{R}_2|_m$.*

In Part II, we will present two formal requirements specification languages and a formal method for behavior verification.

## 6.3 Time and Domain Abstraction

We have introduced reference and sample time for modeling multiple time structures of a hybrid dynamic system. Here we study another kind of mapping between two time structures, which is for modeling dynamic systems at different levels of detail.

A time structure $\langle \mathcal{T}, d, \mu\rangle$ may be related to another time structure $\langle \mathcal{T}', d', \mu'\rangle$ by a *homomorphic time mapping* $h : \mathcal{T} \rightarrow \mathcal{T}'$ where $h$ is a surjective partial or total function, or $h : \mathcal{T} \rightarrow \mathcal{T}' \cup \{\perp\}$ is a surjective function, such that

- it is monotonic: $t_1 \leq_{\mathcal{T}} t_2$ implies $h(t_1) \leq_{\mathcal{T}'} h(t_2)$ if both sides are defined ($\neq\perp$),
- the least element is preserved: $h(\mathbf{0}) = \mathbf{0}'$,
- the metrics are preserved: $m'(t') = \inf\{m(t)|h(t) = t'\}$,

- it is continuous: for any open $T' \subseteq \mathcal{T}'$ in its metric topology, $h^{-1}(T')$ is open in its metric topology, and
- the measures are preserved: $\mu'(T') = \mu(h^{-1}(T'))$.

$\mathcal{T}'$ is an *abstraction* of $\mathcal{T}$, and $\mathcal{T}$ is a *refinement* of $\mathcal{T}'$. For example, let $h : \mathcal{R}^+ \to \mathcal{N}$ be a partial mapping with

$$h(t) = \begin{cases} 0 & \text{if } t < 1 \\ n & \text{else if } n < t < n+1. \end{cases}$$

Function $h$ is a homomorphic time mapping. $\mathcal{N}$ is an abstraction of $\mathcal{R}^+$, and $\mathcal{R}^+$ is a refinement of $\mathcal{N}$.

A domain $A$ may be related to a domain $A'$ by a *homomorphic domain mapping* $h : A \to A'$ where $h$ is surjective and continuous in the derived metric topology. $A'$ is an *abstraction* of $A$, and $A$ is a *refinement* of $A'$. For example, let $h : \overline{\mathcal{R}} \to \overline{\mathcal{S}}$, where $\mathcal{S} = \{-1, 1\}$, be a mapping with

$$h(x) = \begin{cases} -1 & \text{if } r < 0 \\ 1 & \text{if } r > 0 \\ \perp_{\mathcal{S}} & \text{if } r = 0 \text{ or } r = \perp_{\mathcal{R}}. \end{cases} \tag{6.1}$$

Function $h$ is a homomorphic domain mapping. $\overline{\mathcal{S}}$ is an abstraction of $\overline{\mathcal{R}}$, and $\overline{\mathcal{R}}$ is a refinement of $\overline{\mathcal{S}}$.

Let $\Sigma = \langle S, F \rangle$ be a signature. A $\Sigma$-domain structure $A$ may be related to a $\Sigma$-domain structure $A'$ by a *homomorphic domain structure mapping* $h = \{h_s : A_s \to A'_s\}_{s \in S}$ where (1) $A'_s$ is an abstraction of $A_s$ for all $s \in S$, and (2) $h(f^A(x_1, \ldots, x_n)) = f^{A'}(h(x_1), \ldots, h(x_n))$ for all $f \in F$. $A'$ is an *abstraction* of $A$, and $A$ is a *refinement* of $A'$.

The condition for a homomorphic domain structure mapping is very strong, since the congruence induced by the mapping must be a $\Sigma$-congruence. For example, let $\Sigma_r = \langle \{r\}, \{+, \cdot\} \rangle$ and $\langle \{\overline{\mathcal{R}}\}, \{+, \cdot\} \rangle$ be a $\Sigma_r$-domain structure. The mapping defined in Function 6.1 is not a homomorphic domain structure mapping, since $\simeq_h$ is not a $+$-congruence. For another example, let $\Sigma = \langle \{s\}, \{\mathtt{0}, \mathtt{f}, \mathtt{g}\} \rangle$ be a signature with $0 :\to s, f : s \to s$ and $g : s, s \to s$, and $\langle \{\overline{\mathcal{N}}\}, \{0, suc, +\} \rangle$ and $\langle \{\overline{\mathcal{B}}\}, \{0, \neg, \oplus\} \rangle$ be $\Sigma$-domain structures. Let $h : \overline{\mathcal{N}} \to \overline{\mathcal{B}}$ be a mapping with

$$h(n) = \begin{cases} 0 & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \\ \perp_{\mathcal{B}} & \text{if } n = \perp_{\mathcal{N}}. \end{cases}$$

Function $h$ is a homomorphic domain structure mapping. $\langle \{\overline{\mathcal{B}}\}, \{0, \neg, \oplus\} \rangle$ is an abstraction of $\langle \{\overline{\mathcal{N}}\}, \{0, suc, +\} \rangle$, and $\langle \{\overline{\mathcal{N}}\}, \{0, suc, +\} \rangle$ is a refinement of $\langle \{\overline{\mathcal{B}}\}, \{0, \neg, \oplus\} \rangle$.

Because it is hard to satisfy the strong condition on homomorphic domain structure mappings for most domain structures, in many cases a weaker version of abstraction may apply. So called *qualitative algebra/dynamics* [Wd90, Wil91] in AI belong to this category. Let $\Sigma = \langle S, F\rangle$ be a signature. A $\Sigma$-domain structure $A$ may be related to a $\Sigma$-domain structure $A'$ by a *domain structure mapping* $h = \{h_s : A_s \to A'_s\}_{s\in S}$ where (1) $A'_s$ is an abstraction of $A_s$ for all $s \in S$, and (2) $f^{A'}(x'_1, \ldots, x'_n) = \bigwedge_{A'}\{h(f(x_1, \ldots, x_n))|h(x_1) = x'_1, \ldots, h(x_n) = x'_n\}$ for all $f \in F$. $A'$ is a *qualitative domain structure* of $A$, and $A$ is a *quantitative domain structure* of $A'$. We should point out here that the partial order structure for any domain is a semilattice, i.e., any two elements have a lower bound, and if $A$ is a domain, the greatest lower bound $\bigwedge_A$ is defined for any subset of $A$. This definition is similar to the definition in [Wil91], except that we enforce continuity in domain mapping. For the previous example, let $\langle\{\overline{\mathcal{S}}\}, \{+, \cdot\}\rangle$ be a $\Sigma_r$-domain structure, with $+$ and $\cdot$ defined as:

$$
\begin{aligned}
x + y &= \begin{cases} 1 & \text{if } x = y = 1 \\ -1 & \text{if } x = y = -1 \\ \perp_{\mathcal{S}} & \text{otherwise, and} \end{cases} \\
x \cdot y &= \begin{cases} 1 & \text{if } x = y = 1 \text{ or } x = y = -1 \\ -1 & \text{if } x = 1, y = -1 \text{ or } x = -1, y = 1 \\ \perp_{\mathcal{S}} & \text{otherwise.} \end{cases}
\end{aligned}
$$

The mapping $h$ defined in Function 6.1 is a domain structure mapping. $\langle\{\overline{\mathcal{S}}\}, \{+, \cdot\}\rangle$ is a qualitative domain structure of $\langle\{\overline{\mathcal{R}}\}, \{+, \cdot\}\rangle$, and $\langle\{\overline{\mathcal{R}}\}, \{+, \cdot\}\rangle$ is a quantitative domain structure of $\langle\{\overline{\mathcal{S}}\}, \{+, \cdot\}\rangle$. However, $h$ is not a homomorphic domain structure mapping, since $h(x + y) = h(x) + h(y)$ does not hold for all $x, y \in \overline{\mathcal{R}}$. Qualitative algebra, along with qualitative diagnosis and qualitative physics [Wd90], has been a major area in AI. In this thesis, we focus only on abstraction with quotient structures.

## 6.4 Behavior Abstraction and Equivalence

A trace is a function from a time structure to a domain. Given $\mathcal{T}'$ as an abstraction time of $\mathcal{T}$ with mapping $h_{\mathcal{T}}$ and $A'$ as an abstraction domain of $A$ with mapping $h_A$, a trace $v : \mathcal{T} \to A$ is *abstractable* to a trace $v' : \mathcal{T}' \to A'$ iff $h_{\mathcal{T}}(t_1) = h_{\mathcal{T}}(t_2) \neq \perp$ implies $h_A(v(t_1)) = h_A(v(t_2))$. The *abstraction trace* of $v$ w.r.t. $h = \{h_{\mathcal{T}}, h_A\}$ is $v' = \lambda h_{\mathcal{T}}(t).h_A(v(t))$. Two traces $v_1$ and $v_2 : \mathcal{T} \to A$ are *equivalent* w.r.t. $h$, written $v_1 \sim_h v_2$, iff $v_2$ and $v_2$ are abstractable to the same abstraction trace w.r.t. $h$. We should point out here that $\sim_h$ is not a congruence since $\sim_h$ is not reflexive (not every trace is abstractable). For example, let $v_1, v_2 : \mathcal{R}^+ \to \overline{\mathcal{R}}$, with

$v_1 = \lambda t.\sin(\pi t)$ and $v_2 = sign(v_1)$ where $sign : \overline{\mathcal{R}} \rightarrow \overline{\mathcal{R}}$ is a function defined as the sign of its argument. Traces $v_1$ and $v_2$ are both abstractable to a trace $v' : \mathcal{N} \rightarrow \overline{\mathcal{S}}$, with $v'(0) = 1$ and $v'(n+1) = -v'(n)$ (Figure 6.1).

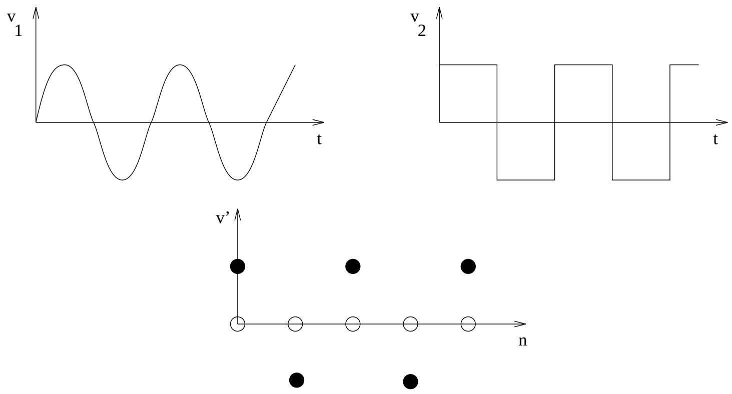


Figure 6.1: Equivalent traces and their abstraction

Consider again the example of the car-like maze traveler. The heading trace of the car $\theta : \mathcal{R}^+ \rightarrow \overline{\mathcal{R}}$ (Figure 6.2(a)) can be abstracted to a discrete trace (Figure 6.2(b)). Notice that the "ambiguous directions" during the turnings are abstracted away.

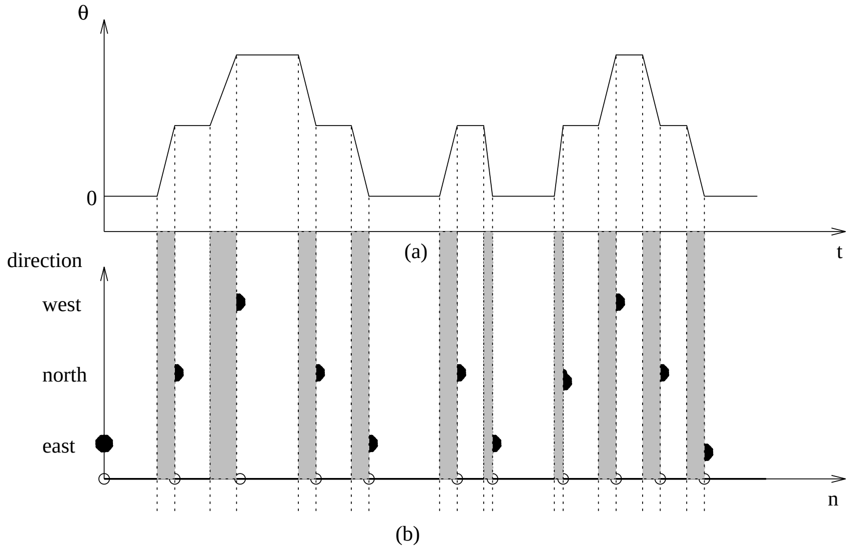


Figure 6.2: The heading of a maze traveler and its abstraction

Similar to the abstraction and equivalence defined for traces, abstraction and equivalence for transductions are defined as follows. A transduction $F : A_1^{\mathcal{T}_1} \to A_2^{\mathcal{T}_2}$ is *abstractable* to a transduction $F' : {A'}_1^{\mathcal{T}_1'} \to {A'}_2^{\mathcal{T}_2'}$ w.r.t. $h = \{h_1, h_2\}$ iff $F(v) \sim_{h_2} F(w)$ whenever $v \sim_{h_1} w$. If there is no input, an abstractable transduction reduces to an abstractable trace. The *abstraction* of $F$ w.r.t. $h = \{h_1, h_2\}$ is $F'(h_1(v)) = h_2(F(v))$. Two transductions $F_1$ and $F_2$ are *equivalent* w.r.t. $h$, written $F_1 \sim_h F_2$, iff $F_1$ and $F_2$ are abstractable to the same abstraction transduction w.r.t. $h$.

Abstraction and equivalence for behaviors are based on the abstraction and equivalence for traces. A behavior $\mathcal{B}$ is *abstractable* w.r.t. $h$ iff for all $\langle i, o\rangle \in \mathcal{B}$, $o$ is abstractable w.r.t. $h$ whenever $i$ is abstractable w.r.t. $h$. If there is no input, an abstractable behavior reduces to the set of abstractable traces. The *abstraction* of $\mathcal{B}$ w.r.t. $h$ is $\mathcal{B}' = \{\langle i', o'\rangle | \langle i, o\rangle \in \mathcal{B} \text{ and } i \text{ is abstractable}\}$. Two behaviors are *equivalent* w.r.t. $h$, written $\mathcal{B}_1 \sim_h \mathcal{B}_2$, iff $\mathcal{B}_1$ and $\mathcal{B}_2$ are abstractable to the same abstraction behavior w.r.t. $h$. Two modules $CN_1(I, O)$ and $CN_2(I, O)$ are *equivalent* w.r.t. $h$, written $CN_1(I, O) \sim_h CN_2(I, O)$, iff $[\![CN_1(I, O)]\!] \sim_h [\![CN_2(I, O)]\!]$.

We should notice that behavior abstraction may not preserve the property of being state-based or time-invariant. Now we investigate the abstractable condition of a state transition system. A state transition system $\langle S, \to\rangle$ is *abstractable* w.r.t. a congruence $\simeq$ on $S$ iff $s_1 \to s_2, s_2' \to s_3$ and $s_2 \simeq s_2'$ imply that $\exists s \in [s_2]$ such that $s_1 \to s \to s_3$. Let $\langle S/\simeq, \to^{\simeq}\rangle$ be a state transition system defined as $[s_1] \to^{\simeq} [s_2]$ iff $\exists s \in [s_1], s' \in [s_2]$ such that $s \to s'$. If $\langle S, \to\rangle$ is abstractable w.r.t. $\simeq$, $\langle S/\simeq, \to^{\simeq}\rangle$ is called the *abstraction* of $\langle S, \to\rangle$ w.r.t. $\simeq$; otherwise, it is called the *approximate abstraction* of $\langle S, \to\rangle$ w.r.t. $\simeq$.

**Proposition 6.4.1** *(1) If $\langle S', \to'\rangle$ is an abstraction of $\langle S, \to\rangle$, the behavior corresponding to $\langle S', \to'\rangle$ is the abstraction of the behavior corresponding to $\langle S, \to\rangle$. (2) If $\langle S', \to'\rangle$ is an approximate abstraction of $\langle S, \to\rangle$, the behavior corresponding to $\langle S', \to'\rangle$ is a superset of the abstraction of the behavior corresponding to $\langle S, \to\rangle$.*

## 6.5 Summary

We have presented formal definitions of the behavior of a system and requirements specification, and a formal relationship between the behavior of a system and a requirements specification. Within this framework, the robustness of parameterized systems and the complexity of behaviors can be studied. We have also presented a systematic approach to the study of behavior abstraction and equivalence using concepts from abstract algebra.

# Chapter 7

# Summary and Related Work

We have presented a semantic model for hybrid dynamic systems modeling and behavior analysis in this modeling framework. In this chapter, we summarize the results of Part I and discuss some related work on models for dynamic systems.

## 7.1 Summary

In this section, we summarize the Constraint Net model for design and analysis in terms of its power and limitations.

### 7.1.1 Power

The Constraint Net model is powerful in the following aspects.

- *Power of Abstraction*: The Constraint Net model is based on the abstract notions of time and domains. With this abstraction, both continuous and discrete time and domains can be represented in a uniform framework. Given abstract structures of time and domains, an abstract structure of dynamics can be derived based on the abstract notion of traces and transductions. Developed on abstract algebra and topology, a system can be represented at different levels of abstraction. Quotient and qualitative dynamics can be formalized, behavior abstraction and equivalence can be studied.

- *Power of Expression*: The syntax of the Constraint Net model is graphical and modular, and its semantics is denotational and composite. Nondeterministic and stochastic systems can be represented with hidden inputs. Parameterized systems and various forms of temporal integration can be incorporated into the model.

- *Power of Computation*: The Constraint Net model is an abstraction and generalization of dataflow-like models, so that hybrid systems — systems with components in different domain and time structures — can be modeled. Furthermore, both sequential computation and analog computation are special types of dynamic system that can be modeled with a simple domain and time structure.

### 7.1.2 Limitations

The Constraint Net model is limited in the following sense.

- *Limitations of Abstraction*: The Constraint Net model is based on the abstract notion of traces and transductions, while transductions are causal mappings from input traces to output traces. Not every physical process can be considered as a transduction. For example, a frequency bandwidth filter is not a transduction, since the output at any time may depend on the whole input trace. Furthermore, partial differential equations cannot be modeled.

- *Limitations of Expression*: The Constraint Net model is developed on the principles of simplicity and generality. There is no inherent notion of nondeterminism, which must be explicitly expressed by hidden inputs. There is no inherent notion of synchronization for communicating systems nor that of time-out for real-time systems, which must be explicitly modeled by event generators and synchronizers. Sequential computation must be explicitly represented via event coordinations.

- *Limitations of Computation*: We call our model *Constraint* Nets for two reasons. First, semantically, a constraint net is a set of equations, each of which imposes a constraint on traces. The semantics of a constraint net is the least solution of the set of equations. Second, we will see, in Part III, constraint satisfaction can be viewed as a dynamic process that can be modeled by a constraint net. Such a constraint net may approach a stable equilibrium that is the solution set of the constraint satisfaction problem. However, not every constraint satisfaction problem can be solved using the Constraint Net model. Furthermore, since the semantics of a constraint net is the least solution of the equations, any constraint net with algebraic loops may result in an undefined solution. From the computational point of view, algebraic loops represent infinite amount of computation in any instant of time. A well-defined constraint net performs only a finite amount of computation in any instant of time.

## 7.2 Related Work

Various models for concurrent and distributed systems [FF84] have been developed in the theory, AI and systems communities. Roughly speaking, these models can be characterized as belonging to one of the three categories: (1) Automata or State Transition Models, (2) Communicating Processes or Multi-agent Architectures, and (3) Nets, Circuits or Dataflow Structures. Models in any of these forms can be equivalent in computational power (as with sequential models). The selection of models depends on applications. Typical criteria for model seletion are:

- Simple and Uniform,
- Modular and Composite,
- Parallel or Concurrent,
- Sequential or Synchronous,
- Nondeterministic or Probabilistic.

Some of these criteria are opposed to each other.

Most of these models can be augmented with the notion of time for modeling real-time and/or hybrid systems. There are also constraint-based models and biology-based models. We survey some typical models in every category and their extensions to real-time and/or hybrid models, then we discuss the relationship between the Constraint Net model and other existing models.

### 7.2.1 Automata or state transition models

Automata or state transition models are typical for studying discrete event systems [Hol82], and most recently, for modeling hybrid systems [GNRR93]. However, for complex systems with multiple components, global state description will cause the exponential growth of the number of states. Nevertheless, modeling global transitions of a system is important for analyzing the system's overall behavior. Although nondeterminism can be expressed by this type of model inherently, automata or state transition models go to the extreme for simplicity and global analysis, with little concern for modularity and parallelism.

Examples of this type of model are Mealy/Moore Machines and Statecharts. Various forms of timed and hybrid automata have been studied recently.

### Mealy/Moore Machines

Mealy/Moore machines [Mea55, Moo56] are the simplest form of input/output transducer for event control systems [CWG88]. Various adaptations can be made to particular domains. For example, Rosenschein [RK87, Ros89] proposed the situated-automata approach, which seeks to analyze knowledge in terms of relations between the states of a machine and the states of its environment over time. This approach, in contrast to the interpreted-symbolic-structure approach that has prevailed in AI for decades, provides a way of compromising between the representational power and real-time execution of AI systems. A situated automaton is in fact a variation of a Moore machine [Moo56]. The Requirement State Machine (RSM) [JLHM91], a special form of Mealy machine [Mea55], has been proposed as a software requirement analysis language for real-time process-control systems.

### Statecharts

The Statechart method was introduced [HP85] as a visual formalism for specifying the behavior of complex reactive systems. It describes a system's behavior in terms of states, events and conditions, with combinations of the latter two causing the transitions between the former. Both states and transitions can be associated in various ways with output events, called activities, which can be triggered either by executing a transition or by entering, exiting, or simply being in a state. A system's inputs are thus the events and its outputs are the activities; their union comprises the interface set.

In Statecharts, the exponential growth of states is avoided by defining higher-level states. States in a Statechart can be repeatedly combined into higher-level states using AND and OR modes of clustering.

### Timed and Hybrid Automata

Much work has been done recently on introducing real-time concepts into formal models of concurrency [dHdR91]. For example, Merritt et al. [MMT91] augmented the input-output automaton model with a notion of time that allows to reason about timed behaviors. Alur and Dill [AD91] developed the theory of timed automata to reason about timed behaviors. Henzinger et al. [HMP91b] incorporated time into an interleaving model of concurrency in which upper and lower bounds on time delay are associated with each transition. None of these models, however, are able to represent continuous change.

Some effort has been made recently to develop models for hybrid systems [GNRR93], systems with both discrete and continuous components. By generalizing timed transition systems to phase transition systems [MMP91, NS91], computation consist of alternating phases of discrete transitions and continuous activities. More specifically, Nerode and Kohn [NK93a] present a model consisting of two automata: a digital control automaton and a plant automaton. The plant automaton can be modeled as a state transition system over intervals. The inputs of a plant automaton are control signals and disturbances, while its states are the solutions of the set of differential equations of the plant for the given control signals and disturbances.

### 7.2.2 Processes or multi-agent architectures

Models in this category represent a system with multiple processes or agents that communicate with each other via channels or shared memories. In most cases, agents and channels can be created dynamically and communication patterns are not fixed at run time. Modularity, compositionality, as well as nondeterminism are features of this type of model. This type of model can be very complex with various communication and synchronization operators; both parallel and sequential computation can be incorporated. Even though discrete time structures can be added to these models, they are concurrent rather than real-time models.

Examples of this type of model are algebraic processes, the Actor model and the cc family.

#### Algebraic Processes

Much work has been done in algebraic processes. Typical models of this type are CSP and CCS. C.A.R. Hoare's Communicating Sequential Processes (CSP) [Hoa85] is a model describing concurrent and distributed computation. A CSP program is a static set of explicit processes. Pairs of processes communicate by naming each other in input and output statements. Communication is nonbuffered and synchronous with unidirectional information flow. Guarded commands are used to introduce indeterminacy. Some work has been done for specifying a robot control system in CSP and formally verifying some properties [LD89]. However, it is hard to capture the essential structure of an analog control system and the dynamics of robot manipulators in CSP.

The work of George Milne and Robin Milner [MM79] is an attempt to describe a mathematical semantics for concurrent computation and communication. Their goal is a formal calculus of concurrent computation, much as the lambda calculus is a formal calculus of uniprocess computation. Their model, Calculus for Communicating Systems (CCS), has explicit processes that communicate synchronously and bidirectionally over labeled channels. The number of processes and their communication connections can change dynamically. Syntactically, a system modeled by CCS is a flowgraph with composition, restriction and relabeling. The semantics of CCS is based on the theory of sets, powerdomains and fixpoint of continuous algebra. Though CCS allows the analysis of the temporal ordering of events, there is no way to specify the relative speeds of events. Synchronous CCS (SCCS) has been studied by Milner [Mil83], in which events are synchronized by timesteps. Timed CCS (TCCS) has also been proposed [MT90, MT91] as a tool for real-time analysis, which introduces willing-to-delay and forcing-to-delay operators.

Many basic tools for communication protocol specification and verification are CCS-like languages [QAF89, Atl89]. Some more general work on the semantics of communicating processes has been presented by Hennessy [Hen88]. His approach relies heavily on abstract algebra — $\Sigma$-algebras and the fixpoint theory of continuous functions — which shows that algebraic theory is a powerful tool for programming semantics.

### Actors

The Actor model was proposed by Hewitt for developing highly parallel machines and open systems [Hew88]. The Actor model takes the theme of object-oriented computation seriously and to an extreme. In an Actor system, everything is an actor (object). Actors communicate by sending each other messages, which are themselves actors. Every actor has a script (program) and acquaintance (data, local storage). When a message arrives at an actor, the actor's script is applied to that message. Clinger [Cli81] gave a denotational semantics for an Actor-like system based on powerdomains and fixpoint theory, and also defined a set of laws that are meant to restrict Actor systems to those that can be physically implemented. Agha [Agh85] further gave a structured operational semantics for an Actor language and discussed compositionality and abstraction from irrelevant detail.

The Robot Schema ($\mathcal{RS}$) Model is a variation of the Actor model, where a schema can be considered as a class of object. $\mathcal{RS}$ is a special model of computation for sensory-based robot programming [LA89]. $\mathcal{RS}$ is a typical concurrent object-oriented model, in which a schema is a general specification and a schema instance is a concurrent object. Each object can be created and terminated by other objects. Therefore, a network is created and changed during computation. Objects communicate with each other through input and output channels. The concept of $\mathcal{RS}$ can be implemented via any concurrent object-oriented language [Zha89, Zha90]. However, the formal semantics of $\mathcal{RS}$ is very complicated, due to various interpretations of the composition, communication and nondeterminism. Furthermore, again, continuous dynamics cannot be represented in this model.

#### The cc Family

Saraswat [Sar89] has developed a framework of concurrent constraint programming, that he called the cc Family. In this paradigm, computation emerges from the interaction of the concurrently executing agents that place, check and instantiate constraints on shared variables that range over some domain of discourse.

Constraints are partial specifications of (possibly infinite) sets of values, and the agents may either collaborate or compute in placing constraints. The major form of concurrency control in the system is through the notion of Atomic Tell and Blocking Ask. The former allows an agent to (instantaneously) place constraints only if they are consistent with the constraints that have already been placed. The latter forces an agent to block when it checks a relationship that is not yet known to hold.

This paradigm is a generalization of research in concurrent logic programming languages [Sha87]. It has been shown that concurrent logic programming languages are good candidates for open systems [KM88] and for the simulation of robot behaviors [ZM92]. However, they are not real-time languages, since their computation time is unpredictable.

A timed extension of the cc family, timed cc, has been proposed [SJG94] in which real-time requirements (such as time-out) can be expressed.

### 7.2.3 Nets or dataflow structures

Unlike state transition models that represent flow of control, computation in dataflow structures is data-driven. Unlike process-based systems in which processes and communication can be created dynamically, operators and connections in dataflow models are fixed. The advantages of a dataflow model are its inherent parallelism or concurrency, its locality (modularity), its graphical orientation, and most importantly, its generality and simplicity. Nondeterminism is inherent for interleaving concurrency models. However, neither sequential computation nor synchronization is explicitly represented.

Examples of this type of model are Petri Nets, circuit models, communicating state machines and operator nets.

### Petri Nets

The Petri Net model is a formal modeling technique that encodes the states of a dynamical system as the markings of tokens on a graph [Pet81]. The graph is a bipartite, directed multigraph that has two kinds of node, places and transitions, and arcs connect places and transitions.

A marked Petri net is the association of a number with each place (the number of tokens on that place), which is not bounded but is always finite. A transition is enabled if every place connected to that transition with $k$ arcs has at least $k$ tokens. A transition may fire at any time if it is enabled. When a transition fires, it moves tokens from its input places to its output places. If multiple transitions are enabled at that time, it nondeterministically choose one to fire.

The Petri Net model of a system can be used to prove properties such as mutual exclusion, liveness and reachability. Various extensions (for example, inhibition) of Petri Nets have been proposed to make it Turing equivalent. The Time Petri Net model is a current area in Petri Net theory research [Pet86, BD91].

### Circuit Models

Circuit models are a typical kind of dataflow model. There are digital circuit models and analog circuit models. Analog circuits are basic systems for analog control. Analog circuits may include resistors, capacitors, amplifiers, differential or integral elements. Digital circuits include synchronous and asynchronous models.

Synchronous circuits (sequential circuits) are the building blocks of most digital computer systems. A synchronous circuit consists of a set of basic gates (e.g., *and*, *or* and *not*) and all the gates operate at the same sampling rate controlled by a single clock. The idea of asynchronous circuits was demonstrated by Sutherland's Turing Award paper "Micropipelines" [Sut89]. Sutherland discards the clocked-logic conceptual framework and thinks instead about a different but equally simple form of control called transition signaling. The basic elements of asynchronous circuits are the exclusive *or* ($xor$) element that acts as the "or" element for events, and the Muller C-element that acts as the "and" element for events. Asynchronous circuits have advantages in hardware design, software and system development.

Variations of circuit models have been adapted in AI. For example, the action network [Nil89] is composed of a forest of logical gates that select actions in response to sensory and stored data. The elementary unit of an action net implements a logical *and* gate.

### Communicating State Machines

Communicating state machines are networks of state machines, each of which has a set of input ports and a set of output ports. Typical examples of this type are the the Augmented Finite State Machine, the Extended State Machine, and temporal automata.

The Augmented Finite State Machine (AFSM) [Bro88] was used as the model for the subsumption architecture. Each AFSM has a set of registers and a set of timers, or alarm clocks, connected to a conventional finite state machine that can control a combinatorial network fed by registers. Registers can be written by attaching input wires to them and receiving messages from other machines. The arrival of a message, or the expiration of a timer, can trigger a change of state in the interior finite state machine. The finite state machine can wait on some event, conditionally dispatch to one of two other states based on some combinational predicate on the registers, or compute a combinatorial function of registers directing the result either back to one of the registers or to an output of the augmented finite state machine.

The Extended State Machine (ESM) [Ost89] is a framework for modeling systems composed of real-time discrete event processes. ESM can be used to model the processes and devices of a plant, as well as the software tasks of controllers implemented as real-time software. Each ESM description of a process will have a distinguished variable called an activity variable that ranges over a set of activities. In addition, an ESM may have a set of data variables to store numerical or quantitative information. States in ESM refer to values of all the activities and data variables. In addition, each ESM has a set of event labels, a set of communication channels and a set of basic actions. The occurrence of an ESM event causes an instantaneous change from the current activity to some new activity, as well as causing a change in the values of the data variables.

The Temporal Automaton model [LS90] is closer to dataflow models than to automata. A temporal automaton has the characteristics of explicit representation of process time, symmetric representation of a machine and of the environment in which it operates, the wiring together of asynchronous automata, and the ability to aggregate individual machines to form one machine at a coarser level of granularity. Temporal automata are defined on entities and transductions. Entities associate time with data domains and transductions induce causal relationships between entities. Two temporal automata can be connected by wires to form a new temporal automaton. A temporal automaton with empty input entities defines a closed system, it otherwise defines a causal system.

#### Operator Nets

The Operator Net model is a generalized deterministic dataflow model [Ash86]. A graphical language is defined that is syntactically extremely simple and that is mainly uninterpreted, i.e., using operator symbols rather than particular operators. This uninterpreted graphical language can then be interpreted in several different ways, by starting with different (continuous) sequence algebras. A mathematical semantics is given by the fixpoint theory that is referred to as Kahn's Principle [Kah74].

Different possible sequence algebras form families, each of which is based on a different continuous data algebra. If $A$ is a data algebra, then $I(A)$ is a sequence algebra based on pointwise extensions of functions in $A$, and $E(A)$ is an enlargement of $I(A)$ by the addition of a set of continuous operators that are not pointwise based, e.g., *next*, *merge*, *follow-by*, etc.

SIGNAL [BL90] and LUSTRE [CPHP87] are specializations of the Operator Net model. Both of them augment the notion of clocks that are represented by streams of Booleans. Each operator can be associated with a clock such that the operator is performed at the clock's sampling rate. This type of model can be considered as a general model for real-time systems and for discrete time and hybrid domain dynamic systems.

### 7.2.4 Constraint-based and biology-based models

Models in this category are motivated by physical and biological natural systems. They are not mainly for providing the syntax or semantics of a programming language. Instead, they can be considered as philosophical or mathematical structures of natural systems.

Most natural systems are constraint-based, following some natural laws or keeping certain relationships. There are two types of relationship, dynamic or algebraic. Constraint-based models explore relations rather than causalities.

There are various biology-based models, such as neural nets and cerebellar models. The categorical theory of biological systems has also been proposed.

#### Constraint-based Models

The constraint paradigm [Ste80] is a model of computation in which values are deduced whenever possible, under the limitation that deductions must be local in a certain sense. One may visualize a constraint "program" as a network of devices connected by wires. Data values may flow along the wires, and computation is performed by the devices. A device computes using only locally available information and places newly derived values on other locally attached wires. In this way computed values are propagated.

An advantage of the constraint paradigm is that a single relationship can be used in more than one direction. The connections to a device are not labeled as inputs and outputs; a device will compute with whatever values are available, and produce as many new values as it can. A disadvantage is that it can only deal with very limited classes of constraint satisfaction problem.

Differential (resp. difference) algebraic equations (DAE) can be considered as taking both dynamic (causal) and algebraic (relational) constraints in one framework. In general, a dynamic system in continuous time (resp. discrete time) is a set of differential (resp. difference) algebraic equations:

$$\dot{x} = f(x, y) \quad (resp.\ x((n+1)\delta) = f(x(n\delta), y(n\delta))),$$

$$y = g(x, y).$$

### Biology-based Models

The Neural Net model is motivated by the principle in physics, i.e., minimizing the energy of a system. Such minimization is performed dynamically by changing the parameters of the system, that is parallel and distributed in general. A neural net can solve a constraint satisfaction problem [RM86] if the energy function is defined according to the degree of satisfaction. The advantages of the Neural Net model for solving constraints are that it can solve soft constraints and that it involves dynamics that is important in behavior simulation and animation [Pla89].

The Cerebellar Model Arithmetic Computer (CMAC) is motivated by the structure and function of the various cells and fiber types in the cerebellum [Alb81]. CMAC is defined by a series of mappings, $S \to M \to A \to P$, where $S$ is a set of input vectors, $M$ is a set of mossy fiber used to encode $S$, $A$ is a set of granule cells contacted by $M$, and $P$ is a set of outputs. The overall mapping $S \to P$ is a function that represents the causal relationship between the input and the output. Feedback is introduced in the model so that the system can learn. Furthermore, CMAC can simulate finite state automata, as well as compute integrals and other general functions. Hierarchical structures can be used for modeling complex systems.

The categorical theory of biological systems was studied by mathematical biologists [Ros85]. Using the categorical theory, the dynamics of a composition system, quotient dynamics, and hierarchies can be studied formally and abstractly.

### 7.2.5 Relationships with the Constraint Net Model

A distinguished feature of the Constraint Net model (CN), comparing with all the existing models, is abstraction. CN is an abstraction and generalization of dataflow-like models. With abstract time and domain structures, CN models dynamic systems with components of different dynamics. It is the first time that the programming semantics techniques are applied to dynamic systems modeling.

Some important concepts in CN are influenced by Temporal Automata and Operator Nets. Comparing with Temporal Automata, CN is defined on more general and abstract structures of time and domains, based on which, traces, event-driven as well as primitive transductions are formalized. In addition, CN has a more rigorous semantics based on fixpoint theory. Comparing with Operator Nets, CN introduces reference time structures that can be continuous as well as discrete. In addition, events in event traces are transitions so that Sutherland's event logic is adopted.

CN is a net-oriented model, while a component in a net can be an automaton or a state transition system. Processes or components in CN cannot be created or destroyed, and interconnections are fixed. However, such effects can be achieved by event-driven computation. CN can model synchronous, asynchronous and analog circuits. Even though CN does not directly represent synchronous communication and sequential computation, such mechanisms can be generated by event synchronization using the event logic. The syntactic structure of CN is similar to that of Petri Nets, i.e., a bipartite directed graph. However, the semantics of CN is for maximum parallelism, while the semantics of Petri Nets is for concurrency. CN is an inherently deterministic model, while nondeterminism can be captured by hidden inputs. CN can efficiently model differential and difference equations, Neural Nets and CMAC. CN can also simulate constraint-based models, given the underlying dynamics that keeps the relationship as a stable state. Since CN is based on algebraic theory, homomorphism and quotient dynamics can be studied under this model.

In summary, the major contributions of CN are: (1) CN models asynchronous and synchronous components, as well as coordination among components with different time structures; (2) CN supports abstract data types and functions, as well as algebraic specification; (3) CN can provide a programming semantics for the design and analysis of hybrid real-time embedded systems; (4) CN serves as a foundation for the specification and verification of hybrid systems.

# Part II

# Requirements Specification and Behavior Verification

*The way of human follows the way of earth.*
*The way of earth follows the way of heaven.*
*The way of heaven follows the way of* Tao.
*The way of* Tao *follows the way of Nature.*
*— Tao Teh Ching, Lao Tzu*

*Implementations follow algorithms.*
*Algorithms follow specifications.*
*Specifications follow ideas.*
*Ideas follow the way of Nature.*
*— Zhang Ying*

# Chapter 8

# Introduction

We have developed a semantic model for dynamic systems. A model of a dynamic system represents the whole system as a set of components and their connections. However, the behavior of the system is not explicitly represented, since most dynamic systems have no closed form solutions at all. On the other hand, most design requirements can be expressed by qualitative properties and can be satisfied by many models. As a simple example, $\dot{x} = -x$ is a model of a dynamic system, which fortunately has a closed form solution: $x = \lambda t.x_0 e^{-t}$. A requirements specification may simply be a limit property $\lim_{t \to \infty} x(t) = 0$. In this case, the model $\dot{x} = -x$ satisfies the specification $\lim_{t \to \infty} x(t) = 0$. In Part II, we propose and answer the following two questions: What is an appropriate requirements specification language? How to verify the behavior of a system against certain requirements specification?

In this chapter, we present an overview of Part II, Requirements Specification and Behavior Verification. There are three major chapters in Part II. Chapter 9 develops timed linear temporal logic. Chapter 10 develops timed $\forall$-automata. Chapter 11 develops a formal method for ensuring that the behavior of a system satisfies a timed $\forall$-automata specification.

## 8.1 Timed Linear Temporal Logic

Since we consider time as a linearly ordered set with a least element, linear temporal logic is the simplest specification language for sequential (dynamic) behaviors.

First, we develop a propositional linear temporal logic (PLTL). As with other temporal logics, we define the basic temporal operators $\mathcal{U}$ and $\mathcal{S}$; $F_1 \,\mathcal{U}\, F_2$ indicates that $F_1$ is $true$ after the current time until $F_2$ becomes $true$, and $F_1 \,\mathcal{S}\, F_2$ indicates that $F_1$ is $true$ up to the current time since $F_2$ becomes $true$. From these basic operators, we further define $\Diamond$ (eventually), $\Box$ (always), $\bigcirc$ (next), $\ominus$ (previous), etc. Unlike other temporal logics, PLTL is defined for arbitrary time structures, with discrete and continuous time as special instances.

Second, we extend PLTL with two real-time operators $\mathcal{U}^\tau$ (real-time until) and $\mathcal{S}^\tau$ (real-time since) where $\tau > 0$ is any positive real number. The resultant language is called a propositional timed linear temporal logic (PTLTL), where "timed" indicates the representation of metric or measure properties of time. From these two basic real-time operators, we further define other real-time operators such as $\Diamond^\tau$ (real-time eventually) and $\Box^\tau$ (real-time always).

Third, we define FTLTL, a first order TLTL. FTLTL is strongly typed, i.e., its domain is a multi-sorted $\Sigma$-algebra. Terms of FTLTL are defined on the signature $\Sigma$ and predicates are associated with types too. Furthermore, any global variable (variable whose value is a constant over time) can be quantified. RFTLTL, a restricted version of FTLTL, is also defined, in which quantifiers are restricted to state formulas (formulas without temporal or real-time operators). FTLTL is strictly more powerful than RFTLTL, however, RFTLTL gains its advantage in the simplicity of verification.

Finally, we propose the concept of open state specification and briefly discuss the importance and the use of open state specification.

## 8.2 Timed $\forall$-automata

An alternative to linear temporal logic for representing sequential behaviors is automata. Consider an automaton as a language recognizer that accepts a set of traces. If a trace is accepted by the automaton, the trace satisfies the specification defined by the automaton. The simplest automata are finite state automata.

First, we present discrete $\forall$-automata, adopted from the definition given by Manna and Pnueli [MP87]. Discrete $\forall$-automata are finite state automata accepting infinite sequences, i.e., traces of discrete time. $\forall$-automata have a graphical representation that is useful and illuminating. Furthermore, it has been shown [MP87] that discrete $\forall$-automata are strictly more powerful than PLTL.

Second, we extend discrete $\forall$-automata to discrete timed $\forall$-automata, by augmenting time bounds on automaton-states. With this augmentation, various types of real-time property can be specified.

Finally, we generalize discrete timed $\forall$-automata to timed $\forall$-automata. Timed $\forall$-automata can accept traces of arbitrary time structures, with discrete and continuous time structures as special cases.

## 8.3 Behavior Verification

We start with the concepts of behavior verification in general and the discussion of the theorem proving approach to verification in particular. The rest of the chapter focuses on verification techniques for timed $\forall$-automata specification.

One of the important advantages of timed $\forall$-automata specification is that there exists a formal verification procedure. This verification procedure is derived from the integration of a model checking technique and a stability analysis method.

For verifying state-based and time-invariant behaviors of discrete time systems, we modify the verification rules developed by Manna and Pnueli [MP87] in the following ways:

- Ranking functions are replaced by Liapunov functions that generalize the functions for stability analysis in dynamic systems.
- Verification rules for real-time bounds are augmented so that real-time properties can be verified.

We apply the verification rules to the semi-automatic verification of constraint nets on discrete time structures. A verification of this type reduces to a set of first order state formulas that can be checked by a theorem prover.

We translate the verification rules into an algorithm for finite domain and discrete time dynamic systems. The algorithm has a polynomial time complexity in both the size of the model and the size of the specification. With the concept of state transition abstraction, further savings in complexity can be explored.

Finally, we generalize the verification rules so that behaviors with continuous as well as discrete time structures can be formally verified.

## 8.4 Summary and Related Work

The novelty in specification languages includes: a temporal logic defined on abstract time and domains, a timed extension to finite automata, and a generalized version of finite automata that accepts traces of continuous time. The novelty in behavior verification includes a semi-automatic verification method for discrete constraint nets, an efficient algorithm for finite domain systems, and a formal verification method for behaviors of hybrid systems.

# Chapter 9

# Timed Linear Temporal Logic

Temporal logic provide a simple and precise specification for sequential behaviors [Eme90]. We develop timed linear temporal logic (TLTL) for specifying desired properties of system behaviors, where "linear" refers to linearly ordered time structures and "timed" implies metric distances. First we generalize the propositional linear temporal logic to specifying properties of arbitrary traces (instead of finite or infinite sequences). Then we augment real-time modal operators so that real-time properties (e.g., real-time response) can be specified. Finally, we develop a first order TLTL for arbitrary time and domain structures.

## 9.1 Propositional Linear Temporal Logic (PLTL)

The simplest temporal logic is the propositional linear temporal logic (PLTL). In this section, we present a form of PLTL that can incorporate both discrete and continuous time, so that properties of arbitrary traces can be specified and reasoned about.

### 9.1.1 PLTL: syntax and semantics

The basic form of the propositional linear temporal logic (PLTL) is the classical propositional logic extended with *temporal operators.* Formally, the syntax of the logic is defined as follows.

**Definition 9.1.1 (Syntax of PLTL)** *Let $\Phi$ be a set of propositions. The basic* syntax *can be defined using BNF:*

$$F ::= false \mid p \mid F_1 \to F_2 \mid F_1 \,\mathcal{U}\, F_2 \mid F_1 \,\mathcal{S}\, F_2$$

*where $p \in \Phi$ is a proposition, $\to$ is a logical connective denoting "implication," $\mathcal{U}$ is a temporal operator denoting "until" and $\mathcal{S}$ is a temporal operator denoting "since."*

We will use the convention that temporal operators have higher priorities than logical connectives, and unary connectives (operators) have higher priorities than binary connectives (operators).

A *frame* of PLTL is a triple $\langle \mathcal{T}, A, V\rangle$ where $\mathcal{T}$ is a time structure, $A$ is a domain, and $V : \Phi \rightarrow 2^A$ is an *interpretation* that assigns to each proposition $p \in \Phi$ a subset $V(p)$ of $A$. We will use $a \models p$ or $p(a)$ to denote $a \in V(p)$.

A *model* of PLTL is a pair $\langle \mathcal{F}, v\rangle$ where $\mathcal{F} = \langle \mathcal{T}, A, V\rangle$ is a frame and $v : \mathcal{T} \rightarrow A$ is a trace.

Formally, the semantics of the logic is defined as follows.

**Definition 9.1.2 (Semantics of PLTL)** *Let $\mathcal{F} = \langle \mathcal{T}, A, V\rangle$ be a frame and $\langle \mathcal{F}, v\rangle$ be a model of PLTL. Let $F$ be a PLTL formula. Then $v \models_t F$* denotes *that $v$ satisfies $F$ at time $t$:*

- $v \not\models_t false$.
- $v \models_t p$ *for* $p \in \Phi$ *iff* $v(t) \models p$.
- $v \models_t F_1 \rightarrow F_2$ *iff* $v \models_t F_1$ *implies* $v \models_t F_2$.
- $v \models_t F_1 \,\mathcal{U}\, F_2$ *iff* $\exists t' > t, v \models_{t'} F_2$ *and* $\forall t''$, $t < t'' < t', v \models_{t''} F_1$.
- $v \models_t F_1 \,\mathcal{S}\, F_2$ *iff* $\exists t' < t, v \models_{t'} F_2$ *and* $\forall t''$, $t' < t'' < t, v \models_{t''} F_1$.

We will use $v \models F$ to denote that $v$ satisfies $F$ initially, i.e., $v \models_{\mathbf{0}} F$. $F$ is *valid* over a frame $\mathcal{F}$, iff for any model $\langle \mathcal{F}, v\rangle$, $v \models F$. $F$ is *valid*, iff for any frame $\mathcal{F}$, $F$ is *valid* over $\mathcal{F}$. $F$ is *satisfiable* over a frame $\mathcal{F}$, iff for some model $\langle \mathcal{F}, v\rangle$, $v \models F$. $F$ is *satisfiable*, iff for some frame $\mathcal{F}$, $F$ is *satisfiable* over $\mathcal{F}$.

### 9.1.2 PLTL: extensions

More logical connectives and temporal operators can be defined using the basic logic connective $\rightarrow$ and the basic temporal operators $\mathcal{U}$ and $\mathcal{S}$.

Some commonly used logical connectives are defined as follows:

- *Negation*: $\neg F \equiv F \rightarrow false$.
- *True*: $true \equiv \neg false$.
- *Disjunction*: $F_1 \vee F_2 \equiv \neg F_1 \rightarrow F_2$.
- *Conjunction*: $F_1 \wedge F_2 \equiv \neg(F_1 \rightarrow \neg F_2)$.

- *Equivalence*: $F_1 \leftrightarrow F_2 \equiv (F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)$.

Some commonly used temporal operators are defined as follows:

- *Eventually*: $\Diamond F \equiv F \vee true\, \mathcal{U}\, F$.
- *Always*: $\Box F \equiv \neg\Diamond\neg F$.
- *Next*: $\bigcirc F \equiv F\, \mathcal{U}\, F$.
- *Previous*: $\ominus F \equiv F\, \mathcal{S}\, F$.
- *Wait*: $F_1\, \mathcal{W}\, F_2 \equiv \Box F_1 \vee F_1 \wedge (F_1\, \mathcal{U}\, F_2) \vee F_2$.

Various stronger and weaker variations of these temporal operators [Eme90] can also be defined.

The semantics of these logical connectives and temporal operators can be derived from their definitions. Let $\mathcal{F} = \langle \mathcal{T}, A, V\rangle$ be a frame and $\langle \mathcal{F}, v\rangle$ be a model of PLTL. Let $F$ be an extended PLTL formula:

- $v \models_t \neg F$ iff $v \not\models_t F$.
- $v \models_t true$.
- $v \models_t F_1 \vee F_2$ iff $v \models_t F_1$ or $v \models_t F_2$.
- $v \models_t F_1 \wedge F_2$ iff $v \models_t F_1$ and $v \models_t F_2$.
- $v \models_t \Diamond F$ iff $\exists t' \geq t,\ v \models_{t'} F$.
- $v \models_t \Box F$ iff $\forall t' \geq t, v \models_{t'} F$.
- $v \models_t \bigcirc F$ iff $\exists t' > t, \forall t'', t < t'' \leq t', v \models_{t''} F$.
- $v \models_t \ominus F$ iff $\exists t' < t, \forall t'', t' \leq t'' < t, v \models_{t''} F$.
- $v \models_t F_1\, \mathcal{W}\, F_2$ iff $\forall t' \geq t,\ v \models_{t'} F_1$, or $\exists t' > t,\ v \models_{t'} F_2$ and $\forall t'',\ t \leq t'' < t', v \models_{t''} F_1$, or $F_2$.

We should note that the temporal operators $\bigcirc$ and $\ominus$ are generalizations of the "next" and "previous" operators, respectively, from discrete to arbitrary time. However, $\neg(\bigcirc F) \wedge \neg(\bigcirc\neg F)$ and $\neg(\ominus F) \wedge \neg(\ominus\neg F)$ are satisfiable, and $\bigcirc F \leftrightarrow \bigcirc(\bigcirc F)$ and $\ominus F \leftrightarrow \ominus(\ominus F)$ are valid, for any frame with dense time.

For the maze traveler example in Part I, let $ME$ be a proposition denoting that the robot is moving east. A desired property of the maze traveler is $\Box\Diamond ME$, i.e., moving east infinitely often, which ensures the escape of the robot from any finite maze, for the given design and environment.

We can define some more abbreviations that are more convenient to use in many situations.

- $final \equiv \neg \bigcirc true$.
- $initial \equiv \neg \ominus true$.
- $rise(p) \equiv (\neg p \wedge \bigcirc p) \vee (\ominus\neg p \wedge p)$.
- $change(p) \equiv rise(p) \vee rise(\neg p)$.
- $event(p) \equiv (\ominus\neg p \wedge p) \vee (\ominus p \wedge \neg p)$.

Some important properties of behaviors can be specified using PLTL.

- *Safety*: If $B$ is a proposition denoting a bad situation, $\Box\neg B$.
- *Goal achievement*: If $G$ is a proposition denoting a final goal, $\Diamond\Box G$.
- *Persistence*: If $P$ is a proposition denoting a persistent condition, $\Box\Diamond P$.
- *Precedence* $Q\mathcal{B}R$: $Q$ happens before $R$, i.e. $\neg R\mathcal{W}(\neg R \wedge Q)$.
- *Interleaving* $Q\mathcal{I}R$: $Q$ and $R$ interleave, i.e. $\Box(R \rightarrow Q\mathcal{B}R) \wedge \Box(Q \rightarrow R\mathcal{B}Q)$.

Now we can formally specify desired properties of the producer-consumer circuit in Figure 5.3. The first desired property is that producing precedes consuming, i.e.,

$$event(C1)\,\mathcal{B}\,event(C2).$$

The second desired property is that producing and consuming interleave, i.e.,

$$event(C1)\,\mathcal{I}\,event(C2).$$

## 9.2 Propositional TLTL

In order to specify the metric properties of time, we develop Timed Linear Temporal Logic (TLTL). In this section, we introduce propositional TLTL (PTLTL), and in the next section, we present the first order TLTL (FTLTL).

The basic syntax and semantics of PTLTL are the same as those of PLTL. In addition, we augment the basic form of PLTL with two real-time operators. Let $\tau > 0$ be a positive real number, $T_{t+\tau} = \{t' | t < t', d(t,t') \leq \tau\}$ and $T_{t-\tau} = \{t' | t' < t, d(t',t) \leq \tau\}$. Two *real-time operators* are defined as follows:

- $v \models_t F_1 \mathcal{U}^\tau F_2$ iff $\exists t' \in T_{t+\tau}$, $v \models_{t'} F_2$ and $\forall t'', t < t'' < t', v \models_{t''} F_1$.
- $v \models_t F_1 \mathcal{S}^\tau F_2$ iff $\exists t' \in T_{t-\tau}$, $v \models_{t'} F_2$ and $\forall t'', t' < t'' < t, v \models_{t''} F_1$.

Other real-time and temporal operators can be defined using the two basic real-time operators.

- $\Diamond^\tau F \equiv true \mathcal{U}^\tau F$.
- $\Box^\tau F \equiv \neg(\Diamond^\tau \neg F)$.
- $\Diamond_\tau F \equiv true \mathcal{S}^\tau F$.
- $\Box_\tau F \equiv \neg(\Diamond_\tau \neg F)$.

The semantics of these real-time operators can be derived as follows:

- $v \models_t \Diamond^\tau F$ iff $\exists t' \in T_{t+\tau}$, $v \models_{t'} F$.
- $v \models_t \Box^\tau F$ iff $\forall t' \in T_{t+\tau}$, $v \models_{t'} F$.
- $v \models_t \Diamond_\tau F$ iff $\exists t' \in T_{t-\tau}$, $v \models_{t'} F$.
- $v \models_t \Box_\tau F$ iff $\forall t' \in T_{t-\tau}$, $v \models_{t'} F$.

With real-time operators, real-time properties can be specified, for example, real-time response can be specified as $\Box(E \rightarrow \Diamond^\tau R)$.

## 9.3 First Order TLTL

We present FTLTL and its restricted version RFTLTL. RFTLTL imposes a constraint that quantifiers are associated only with state formulas (formulas without temporal and real-time operators).

To define the syntax for FTLTL, we shall first define terms. Let $\Sigma = \langle S, F \rangle$ be a signature, $X_l$ be a set of trace variables, also called *local variables*, and $X_g$ be a set of parameter variables, also called *global variables*. $X = X_l \cup X_g$ is the set of $S$-sorted variables. The set of *terms* of sort $s \in S$ induced by $\Sigma$ and $X$, denoted $T(\Sigma, X)_s$, is the least set of strings that satisfies one of the following:

- if $x \in X_s$, then $x \in T(\Sigma, X)_s$,
- if $x \in X_l \cap X_s$, then $pre(x), x - \tau \in T(\Sigma, X)_s$ for $\tau > 0$,
- if $f \in F$ with type $\rightarrow s$, then $f \in T(\Sigma, X)_s$,
- if $f \in F$ with type $s^* \rightarrow s$ where $s^* : I \rightarrow S$, then $f(T) \in T(\Sigma, X)_s$ where $T : I \rightarrow T(\Sigma, X)$ with $T_i \in T(\Sigma, X)_{s_i^*}$.

Given $\Sigma = \langle S, F \rangle$ as a signature, let $\Phi$ be a set of $S$-sorted predicate symbols, such that for each $p \in \Phi$, the *type* of $p$ is a tuple $s^* : I \rightarrow S$. The syntax of FTLTL can be defined given $\Sigma$ and $\Phi$.

**Definition 9.3.1 (Syntax of FTLTL)** *The basic* syntax *of FTLTL can be defined as:*

$$F ::= \; false \,|\, T_s^1 = T_s^2 \,|\, p(T) \,|\, F_1 \rightarrow F_2 \,|\, F_1 \,\mathcal{U}\, F_2 \,|\, F_1 \,\mathcal{S}\, F_2 \,|\, F_1 \,\mathcal{U}^\tau\, F_2 \,|\, F_1 \,\mathcal{S}^\tau\, F_2 \,|\, \exists x F$$

*where $T_s \in T(\Sigma, X)_s$ is a term of sort $s$, $p \in \Phi$ is a predicate symbol with type $s^* : I \rightarrow S$, $T : I \rightarrow T(\Sigma, X)$ with $T_i \in T(\Sigma, X)_{s_i^*}$, and $x \in X_g$ is a global variable.*

A *frame* of FTLTL is a triple $\langle \mathcal{T}, A, V \rangle$ where $\mathcal{T}$ is a time structure, $A$ is a $\Sigma$-domain structure and $V$ is an interpretation that assigns to each predicate symbol $p \in \Phi$ a subset $V(p)$ of $\times_I A_{s_i^*}$, given that the type of $p$ is $s^* : I \rightarrow S$.

A *model* of FTLTL is a pair $\langle \mathcal{F}, \sigma \rangle$ where $\mathcal{F} = \langle \mathcal{T}, A, V \rangle$ is a frame and $\sigma = \langle \sigma_l, \sigma_g \rangle$ is a valuation for $X = X_l \cup X_g$, i.e., $\sigma_g : X_g \rightarrow A$ and $\sigma_l : X_l \rightarrow (\mathcal{T} \rightarrow A)$. By extending the valuation $\sigma$ from variables to terms, we have $\sigma : T(F, X) \rightarrow (\mathcal{T} \rightarrow A)$, such that for any $t \in \mathcal{T}$:

- $\sigma(x)(t) = \sigma_g(x)$ for any $x \in X_g$,
- $\sigma(x)(t) = \sigma_l(x)(t), \sigma(pre(x))(t) = \sigma_l(x)(pre(t)), \sigma(x - \tau)(t) = \sigma_l(x)(t - \tau)$ for any $x \in X_l$,
- $\sigma(f(T))(t) = f^A(\sigma(T)(t))$ for any $f \in F$.

**Definition 9.3.2 (Semantics of FTLTL)** *Let $\mathcal{F} = \langle \mathcal{T}, A, V \rangle$ be a frame and $\langle \mathcal{F}, \sigma \rangle$ be a model of FTLTL. Let $F$ be an FTLTL formula, $\sigma \models_t F$* denotes *that $\sigma$ satisfies $F$ at time $t$:*

- *$\sigma \not\models_t false$.*

- $\sigma \models_t T_s^1 = T_s^2$ *iff* $\sigma(T_s^1)(t) = \sigma(T_s^2)(t)$.
- $\sigma \models_t p(T), p \in \Phi$ *iff* $\sigma(T)(t) \in V(p)$.
- $\sigma \models_t F_1 \rightarrow F_2$ *iff* $\sigma \models_t F_1$ *implies* $\sigma \models_t F_2$.
- $\sigma \models_t F_1 \mathcal{U} F_2$ *iff* $\exists t' > t$, $\sigma \models_{t'} F_2$ *and* $\forall t'', t < t'' < t', \sigma \models_{t''} F_1$.
- $\sigma \models_t F_1 \mathcal{S} F_2$ *iff* $\exists t' < t$, $\sigma \models_{t'} F_2$ *and* $\forall t'', t' < t'' < t, \sigma \models_{t''} F_1$.
- $\sigma \models_t F_1 \mathcal{U}^\tau F_2$ *iff* $\exists t' \in T_{t+\tau}$, $\sigma \models_{t'} F_2$ *and* $\forall t'', t < t'' < t', \sigma \models_{t''} F_1$.
- $\sigma \models_t F_1 \mathcal{S}^\tau F_2$ *iff* $\exists t' \in T_{t-\tau}$, $\sigma \models_{t'} F_2$ *and* $\forall t'', t' < t'' < t, \sigma \models_{t''} F_1$.
- $\sigma \models_t \exists x F$, $x \in X_s$ *iff there is a value* $a$ *in* $A_s$, $\sigma \models_t F[a/x]$, *where* $F[a/x]$ *stands for substitution of* $x$ *in* $F$ *by* $a$.

We will use $\sigma \models F$ to denote that $\sigma$ satisfies $F$ initially, i.e., $\sigma \models_{\mathbf{0}} F$. $F$ is *valid* over a frame $\mathcal{F}$, iff for any model $\langle \mathcal{F}, \sigma \rangle$, $\sigma \models F$. $F$ is *valid* , iff any frame $\mathcal{F}$, $F$ is *valid* over $\mathcal{F}$. $F$ is *satisfiable* over a frame $\mathcal{F}$, iff for some model $\langle \mathcal{F}, \sigma \rangle$, $\sigma \models F$. $F$ is *satisfiable*, iff for some frame $\mathcal{F}$, $F$ is *satisfiable* over $\mathcal{F}$.

Various logical connectives, temporal and real-time operators can be defined as for PTLTL. In addition, let $\forall$ be the dual of $\exists$, i.e., $\forall x F \equiv \neg \exists x \neg F$.

If we restrict quantifiers to state formulas (formulas without temporal and real-time operators), we have RFTLTL, a restricted version of FTLTL. Formally, a *state formula* is defined as

$$F_s ::= false \,|\, T_s^1 = T_s^2 \,|\, p(T) \,|\, F_s^1 \rightarrow F_s^2 \,|\, \exists x F_s$$

where $T_s \in T(\Sigma, X)_s$ is a term of sort $s$, $p \in \Phi$ is a predicate symbol with type $s^* : I \rightarrow S$, $T : I \rightarrow T(F, X)$ with $T_i \in T(F, X))_{s_i^*}$ and $x \in X_g$. Let $FV(F_s)$ be the set of free variables in $F_s$. A state formula $F_s$ is a *state proposition* iff $FV(F_s) \subseteq X_l$.

A RFTLTL formula can be defined as

$$F ::= F_s \,|\, F_1 \rightarrow F_2 \,|\, F_1 \mathcal{U} F_2 \,|\, F_1 \mathcal{S} F_2 \,|\, F_1 \mathcal{U}^\tau F_2 \,|\, F_1 \mathcal{S}^\tau F_2$$

where $F_s$ is any state formula.

Every RFTLTL formula is also a FTLTL formula, but not vice versa. FTLTL is strictly more expressive than RFTLTL. For example, $\lim_{t \rightarrow \infty} x(t) = 0$ can be expressed by FTLTL as $\forall \epsilon, \epsilon > 0 \rightarrow \Diamond \Box |x| < \epsilon$. However, there is no equivalent RFTLTL formula.

A RFTLTL formula with all free variables as local variables can be interpreted as a PTLTL formula, with domain $\times_{X_l} A_{s_l}$ and state propositions. For example, we may use state proposition $|\theta| < \delta \wedge v > \epsilon$ to represent proposition $ME$, where $\theta$ is the heading and $v$ is the velocity of the car.

## 9.4 Open State Specification

Now we discuss an important issue for requirements specification, the *openness* of state formulas. If $F_s$ is a state formula and $FV(F_s)$ is the set of free variables in $F_s$, let $V(F_s)$ be the set of tuples satisfying $F_s$, i.e., $V(F_s) = \{a : FV(F_s) \to A | a \models F_s\}$.

A state formula $F_s$ is *open* (*closed*) in $A$ iff $V(F_s)$ is open (closed) in the derived metric topology. The following properties are directly from the definition of general topology: (1) State formulas $true$ and $false$ are both open and closed; and (2) if $F$, $F_1$, $F_2$ are open (closed), then:

- $F_1 \vee F_2$ is open (closed);
- $F_1 \wedge F_2$ is open (closed);
- $\neg F$ is closed (open);
- $\exists x F$ is open ($\forall x F$ is closed).

We will further discuss the openness of state formulas in the next chapter. Now we consider the meaning of open state formulas for the definedness of information. If a predicate $p$ on $\times_I \overline{A_i}$ is open, $\Pi_i V(p)$ is either a set of well-defined values or a total set. Extra attention should be paid to this property. For example, let $>$ on $\overline{\mathcal{R}} \times \overline{\mathcal{R}}$ be defined as $\{\langle x, y\rangle | x \in \mathcal{R}, y \in \mathcal{R}, x > y\}$; it is an open predicate that is $true$ only on well-defined tuples $\mathcal{R} \times \mathcal{R}$. Similarly, let $\leq$ on $\overline{\mathcal{R}} \times \overline{\mathcal{R}}$ be defined as $\{\langle x, y\rangle | x \in \mathcal{R}, y \in \mathcal{R}, x \leq y\}$; it is a predicate neither open nor closed that holds only on $\mathcal{R} \times \mathcal{R}$ too. We should notice that for the domain $\overline{\mathcal{R}} \times \overline{\mathcal{R}}$, an obvious relation $x > y \leftrightarrow \neg(x \leq y)$ does not hold any more, since both $\perp_{\mathcal{R}} > \perp_{\mathcal{R}}$ and $\perp_{\mathcal{R}} \leq \perp_{\mathcal{R}}$ are false.

Open state specification is important for requirements specification. For example, for a safety requirements specification $\Box\neg B(x)$ where $B$ is a predicate, $B$ should be closed, so that $\neg B$ is open. Otherwise, if $B$ is open and $\neg B$ is closed, an undefined value will satisfy the safety property. That is usually *not* what safety means.

# Chapter 10

# Timed $\forall$-Automata

An alternative to temporal logic for specifying sequential behaviors is automata. Consider traces as a generalization of (finite or infinite) sequences. A desired property of traces can be specified by an automaton; a trace satisfies the specification iff the automaton accepts the trace.

In this chapter, we develop extensions of $\forall$-automata, proposed by Manna and Pnueli [MP87] for the specification and verification of concurrent programs. We start with an introduction to basic $\forall$-automata that are defined for sequences, or traces with discrete time structures. Then, we augment discrete $\forall$-automata to discrete timed $\forall$-automata by specifying real-time constraints on automaton-states. Finally, we generalize discrete timed $\forall$-automata to timed $\forall$-automata whose time structure can be arbitrary. The relationship between timed $\forall$-automata and TLTL will also be discussed.

## 10.1 Discrete $\forall$-Automata

Discrete $\forall$-automata are non-deterministic finite state automata over infinite sequences. These automata were originally proposed as a formalism for the specification and verification of temporal properties of concurrent programs [MP87]. We briefly introduce discrete $\forall$-automata, but in the role of specifying discrete time traces rather than concurrent programs.

Formally, a $\forall$-automaton is defined as follows.

**Definition 10.1.1 (Syntax of $\forall$-automata)** *A* $\forall$-automaton $\mathcal{A}$ *is a quintuple* $\langle Q, R, S, e, c\rangle$ *where* $Q$ *is a finite set of* automaton-states*,* $R \subseteq Q$ *is a set of* recurrent states *and* $S \subseteq Q$ *is a set of* stable states*. With each* $q \in Q$*, we associate a state proposition* $e(q)$*, which characterizes the* entry condition *under which the automaton may start its activity in* $q$*. With each pair* $q, q' \in Q$*, we associate a state proposition* $c(q, q')$*, which characterizes the* transition condition *under which the automaton may move from $q$ to $q'$.*

$R$ and $S$ are the generalization of *accepting* states to the case of infinite inputs. We denote by $B = Q - (R \cup S)$ the set of *non-accepting (bad)* states.

A ∀-automaton is called *complete* iff the following requirements are met:

- $\bigvee_{q \in Q} e(q)$ is valid.
- For every $q \in Q$, $\bigvee_{q' \in Q} c(q, q')$ is valid.

We will restrict ourselves to complete automata. This is not a substantial restriction, since any automaton can be transformed to a complete automaton by introducing an additional error state $q_E \in B$, with the entry condition:

$$e(q_E) = \neg(\bigvee_{q \in Q - \{q_E\}} e(q)),$$

and the transition conditions:

$$\begin{aligned} c(q_E, q_E) &= true \\ c(q_E, q) &= false \quad \text{for each } q \in Q - \{q_E\} \\ c(q, q_E) &= \neg(\bigvee_{q' \in Q - \{q_E\}} c(q, q')) \quad \text{for each } q \in Q - \{q_E\}. \end{aligned}$$

Let $\mathcal{T}$ be a discrete time structure, $A$ be a domain and $v : \mathcal{T} \to A$ be a trace. A *run* of $\mathcal{A}$ over $v$ is a mapping $r : \mathcal{T} \to Q$ such that (1) $v(\mathbf{0}) \models e(r(\mathbf{0}))$; and (2) for all $t > \mathbf{0}$, $v(t) \models c(r(pre(t)), r(t))$.

A complete automaton guarantees that any discrete trace has a run over it, and that any partial run[1] can always be extended to a total run.

If $r$ is a run, let $Inf(r)$ be the set of automaton-states appearing infinitely many times in $r$, i.e., $Inf(r) = \{q | \forall t \exists t_0 \geq t, r(t_0) = q\}$. If $\mathcal{T}$ has a greatest element $t_0$, $Inf(r) = \{r(t_0)\}$. Therefore, $Inf(r)$ is a generalization of the "final value."

A run $r$ is defined to be *accepting* iff:

1. $Inf(r) \cap R \neq \emptyset$, i.e., *some* of the states appearing infinitely many times in $r$ belong to $R$, or
2. $Inf(r) \subseteq S$, i.e., *all* the states appearing infinitely many times in $r$ belong to $S$.

[1]Consider a run as a function.

**Definition 10.1.2 (Semantics of ∀-automata)** *A ∀-automaton $\mathcal{A}$* accepts *a trace $v$, written $v \models \mathcal{A}$, iff* all *possible runs of $\mathcal{A}$ over $v$ are accepting.*

One of the advantages of using automata as a specification language is its graphical representation. It is useful and illuminating to represent ∀-automata by diagrams. The basic conventions for such representations are the following:

- The automaton-states are depicted by nodes in a directed graph.
- Each initial automaton-state ($e(q) \neq false$) is marked by a small arrow, an *entry arc*, pointing to it.
- Arcs, drawn as arrows, connect some pairs of automaton-states.
- Each recurrent state is depicted by a diamond shape inscribed within a circle.
- Each stable state is depicted by a square inscribed within a circle.

Nodes and arcs are labeled by state propositions. A node or an arc that is left unlabeled is considered to be labeled with *true*. The labels define the entry conditions and the transition conditions of the associated automaton as follows.

- Let $q \in Q$ be a node in the diagram corresponding to an initial automaton-state. If $q$ is labeled by $\psi$ and the entry arc is labeled by $\varphi$, the entry condition $e(q)$ is given by $e(q) = \varphi \wedge \psi$. If there is no entry arc, $e(q) = false$.
- Let $q, q'$ be two nodes in the diagram corresponding to automaton-states. If $q'$ is labeled by $\phi$, and arcs from $q$ to $q'$ are labeled by $\varphi_i, i = 1 \cdots n$, the transition condition $c(q, q')$ is given by $c(q, q') = (\varphi_1 \vee \cdots \vee \varphi_n) \wedge \psi$. If there is no arc from $q$ to $q'$, $c(q, q') = false$.

A diagram representing an incomplete automaton is interpreted as a complete automaton by introducing an error state and associated entry and transition conditions.

Some examples of ∀-automata are shown in Figure 10.1. Figure 10.1(a) accepts any trace that satisfies $\neg G$ only finitely many times, Figure 10.1(b) accepts any trace that never satisfies $B$, and Figure 10.1(c) accepts any trace that will satisfy $R$ in the finite future whenever it satisfies $E$.

Now we give a definition of open specification. A ∀-automata specification is *open* iff $\forall q \in R \cup S, e(q)$ is open and $c(q', q)$ is open for any $q' \in Q$. For discrete domains, open specification implies the well-definedness of accepting states; for continuous domains, open specification

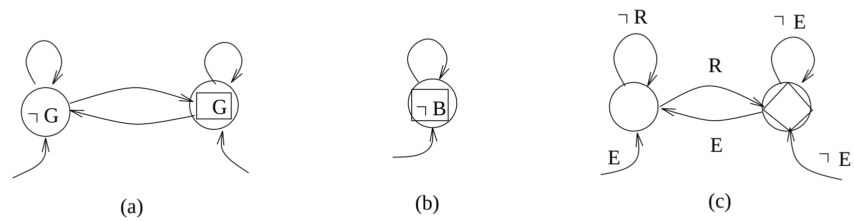


Figure 10.1: ∀-automata: (a) goal achievement (b) safety (c) bounded response

provides a relaxed representation for asymptotic behaviors. For example, a relaxed representation for $\lim_{t\to\infty} x(t) = 0$ is an automaton in Figure 10.1 (a) with $G \equiv |x| < \epsilon$ for some $\epsilon > 0$. We will see that openness should be imposed for any useful requirements specification.

∀-automata may provide a more compact representation than TLTL. For example, the two desired properties of the producer-consumer synchronizer, precedence and interleaving, can be specified by one ∀-automaton in Figure 10.2 (a), where $E(Ci)$ indicates there is an event in $Ci$ and $NE(Ci)$ indicates there is no event in $Ci$. $E(Ci)$ and $NE(Ci)$ can be represented as state propositions as follows. Let $Qi$ be the hidden location of the Muller C-element with output location $Ci$, $E(Ci) \equiv neq(Ci, Qi)$ and $NE(Ci) \equiv eq(Ci, Qi)$ with both $neq$ and $eq$ open. The persistent property of the maze traveler can be represented by a ∀-automaton in Figure 10.2 (b), meaning that the robot will persistently move east.

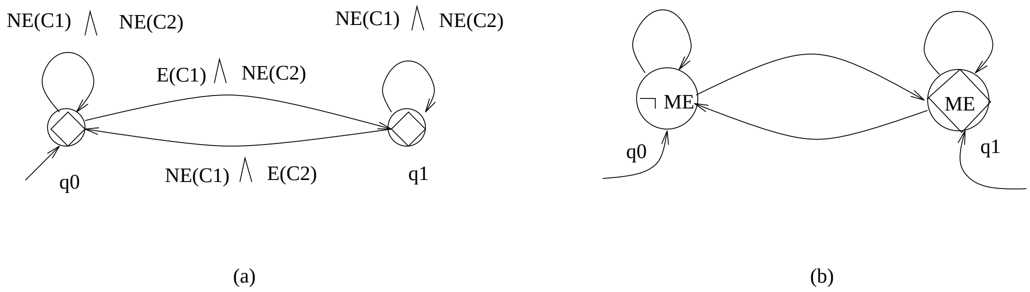


Figure 10.2: The specification of (a) the producer-consumer problem (b) the maze traveler

It has been shown [MP87] that discrete ∀-automata have the same expressive power as Buchi automata [Tho90] and the extended temporal logic (ETL) [Wol83], which are strictly more powerful than the propositional linear temporal logic (PLTL) [Tho90, Wol83].

## 10.2 Discrete Timed ∀-Automata

In order to represent timeliness, we develop timed ∀-automata. Timed ∀-automata are ∀-automata augmented with timed automaton-states and time bounds. Formally, a timed ∀-automaton is defined as follows.

**Definition 10.2.1 (Syntax of timed ∀-automaton)** *A* timed ∀-automaton $\mathcal{TA}$ *is a triple* $\langle \mathcal{A}, T, \tau \rangle$ *where* $\mathcal{A} = \langle Q, R, S, e, c \rangle$ *is a ∀-automaton,* $T \subseteq Q$ *is a set of* timed *automaton-states and* $\tau : T \cup \{bad\} \to \mathcal{R}^+ \cup \{\infty\}$ *is a* time function.

A ∀-automaton is a special timed ∀-automaton with $T = \emptyset$ and $\tau(bad) = \infty$. Graphically, a $T$-state is denoted by a nonnegative real number indicating its time bound. The conventions for complete ∀-automata are adopted for timed ∀-automata.

Let $v : \mathcal{T} \to A$ be a trace. A *run* $r$ of $\mathcal{TA}$ over $v$ is a run of $\mathcal{A}$ over $v$; $r$ is *accepting* for $\mathcal{TA}$ iff

1. $r$ is accepting for $\mathcal{A}$ and
2. $r$ satisfies the time constraints. If $I \subseteq \mathcal{T}$ is an interval of $\mathcal{T}$ and $q^* : I \to Q$ is a segment of run $r$, i.e., $q^* = r_{|I}$, let $\mu(q^*)$ denote the measure of $q^*$, i.e., $\mu(q^*) = \mu(I) = \Sigma_{t \in I}\mu(t)$ since $I$ is discrete. Furthermore, let $\mu_B(q^*)$ denote the measure of bad automaton-states in $q^*$, i.e., $\mu_B(q^*) = \Sigma_{t \in I, q^*(t) \in B}\mu(t)$. Let $Sg(q)$ be the set of segments of consecutive $q$'s in $r$, i.e., $q^* \in Sg(q)$ implies $\forall t \in I, q^*(t) = q$. Let $BS$ be the set of segments of consecutive $B$ and $S$-states in $r$, i.e., $q^* \in BS$ implies $\forall t \in I, q^*(t) \in B \cup S$. The run $r$ satisfies the time condition iff
   (a) (local time constraint) $\forall q \in T, q^* \in Sg(q)$, $\mu(q^*) \leq \tau(q)$ and
   (b) (global time constraint) $\forall q^* \in BS$, $\mu_B(q^*) \leq \tau(bad)$.

**Definition 10.2.2 (Semantics of timed ∀-automaton)** *A timed ∀-automaton* $\mathcal{TA}$ accepts *a trace* $v$*, written* $v \models \mathcal{TA}$*, iff* all *possible runs of* $\mathcal{TA}$ *over* $v$ *are accepting.*

For example, the real-time response $\Box(E \to \Diamond^{\tau} R)$ is depicted by the timed ∀-automaton in Figure 10.3, meaning that any event will be responded to within time $\tau$ (assuming $d(t_1, t_2) = \mu([t_1, t_2)))$.

We should notice that timed ∀-automata are closed under conjunction and disjunction, but not under complementation. Even though discrete ∀-automata are strictly more expressive than PLTL, discrete timed ∀-automata and PTLTL are not strictly more expressive than each other, since PTLTL is closed under complementation.

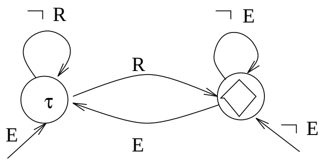


Figure 10.3: Real-time response

## 10.3 Timed ∀-Automata

Now we generalize discrete timed ∀-automata to timed ∀-automata that can accept general traces, with discrete time traces as special cases. The syntax and semantics of timed ∀-automata are the same as those of discrete timed ∀-automata, except for the definitions of runs and accepting runs.

The important concept of general runs is the generalization of the consecution condition. Let $\mathcal{T}$ be a time structure and $t < \infty$ denote that $t$ is not the greatest element of $\mathcal{T}$. Let $v : \mathcal{T} \to A$ be a trace. A *run* of $\mathcal{A}$ over $v$ is a trace $r : \mathcal{T} \to Q$ satisfying

1. *Initiality*: $v(\mathbf{0}) \models e(r(\mathbf{0}))$;
2. *Consecution*:
   - inductivity: $\forall t > \mathbf{0}, \exists q \in Q, t' < t, \forall t'', t' \leq t'' < t, r(t'') = q$ and $v(t) \models c(r(t''), r(t))$ and
   - continuity: $\forall t < \infty, \exists q \in Q, t' > t, \forall t'', t < t'' < t', r(t'') = q$ and $v(t'') \models c(r(t), r(t''))$.

When $\mathcal{T}$ is discrete, the two conditions in *Consecution* are reduced to one, i.e., $\forall t > \mathbf{0}, v(t) \models c(r(pre(t)), r(t))$; and if, in addition, $\mathcal{A}$ is complete, every trace has a run. However, if $\mathcal{T}$ is not discrete, even if $\mathcal{A}$ is complete, not every trace has a run. For example, a trace with infinite transitions among $Q$ within a finite interval has no run. A trace $v$ is *specifiable* by $\mathcal{A}$ iff there is a run of $\mathcal{A}$ over $v$. For example, if $\mathcal{T}$ and $A$ are $[0, 1]$, trace $v : \mathcal{T} \to \overline{A}$ with $v = \lambda t.t$ is not specifiable by the automaton in Figure 10.4.

The definition of accepting runs for ∀-automata is the same as that for discrete cases. A run $r$ is defined to be *accepting* for $\mathcal{A}$ iff:

1. $Inf(r) \cap R \neq \emptyset$, i.e., *some* of the states appearing infinitely many times in $r$ belong to $R$, or

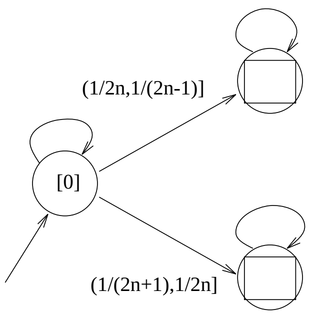


Figure 10.4: A generalized ∀-automaton

2. $Inf(r) \subseteq S$, i.e., *all* the states appearing infinitely many times in $r$ belong to $S$.

We should notice that dense ∀-automata is no longer more powerful than PLTL, since the ability of counting in automata [MP71] is lost when time is dense. In other words, meaningful dense automata are counter-free only, since for any transition between two automaton-states, there is a self-loop at one of the automaton-states.

The definition of accepting runs for timed ∀-automata is similar to that for discrete cases, except for the measures of segments. If $I \subseteq \mathcal{T}$ is an interval of $\mathcal{T}$ and $q^* : I \rightarrow Q$ is a segment of run $r$, i.e., $q^* = r_{|I}$, let $\mu(q^*)$ denote the measure of $q^*$, i.e., $\mu(q^*) = \mu(I) = \int_I dt$. Furthermore, let $\mu_B(q^*)$ denote the measure of bad automaton-states in $q^*$, i.e., $\mu_B(q^*) = \int_I \chi_B(q^*(t))dt$, where $\chi_B$ is the characterization function for set $B$. A run $r$ is *accepting* for a timed ∀-automaton iff

1. $r$ is accepting for its ∀-automaton and
2. $r$ satisfies the time constraints. Let $Sg(q)$ be the set of segments of consecutive $q$'s in $r$, i.e., $q^* \in Sg(q)$ implies $\forall t \in I, q^*(t) = q$. Let $BS$ be the set of segments of consecutive $B$ and $S$-states in $r$, i.e., $q^* \in BS$ implies $\forall t \in I, q^*(t) \in B \cup S$. The run $r$ satisfies the time condition iff
   (a) (local time constraint) $\forall q \in T, q^* \in Sg(q)$, $\mu(q^*) \leq \tau(q)$ and
   (b) (global time constraint) $\forall q^* \in BS$, $\mu_B(q^*) \leq \tau(bad)$.

Timed ∀-automata are powerful enough to represent various temporal and timed properties of dynamic systems, such as persistence or liveness, goal achievement or reachability, safety and real-time response. More importantly, there is a formal verification method based on a model checking technique and a stability analysis method.

# Chapter 11

# Behavior Verification

While modeling focuses on the underlying structure of a system — the organization and coordination of its components — requirements specification imposes global constraints on a system's behavior, and behavior verification checks the relationship between the behavior of a system and a requirements specification. In this chapter, we first discuss general issues of behavior verification, then focus on a formal verification method for timed $\forall$-automata specification.

## 11.1 Behavior Verification: General Issues

We have defined the behavior of a dynamic system as the set of observable input/output traces. Given $\mathcal{B}$ as the behavior of a dynamic system and $\mathcal{R}$ as a requirements specification, the behavior satisfies requirements, written $\mathcal{B} \models \mathcal{R}$, iff $\forall v \in \mathcal{B}, v \models \mathcal{R}$. The *verification* procedure is to certify the relationship $\mathcal{B} \models \mathcal{R}$ for any given behavior $\mathcal{B}$ and requirements specification $\mathcal{R}$.

It is not hard to see that there is no automatic verification procedure for behaviors of discrete time and domain dynamic systems and TLTL specification in general. We have seen that any partial recursive function $f$ can be computed by a constraint net. And whether or not $f$ is defined for an input value $n$ (the halting problem) can be represented by a specification $\Box[(Data\_In = n) \wedge E(Start) \rightarrow \Diamond E(End)]$, where $E(X)$ indicates that there is an event at $X$. There are, as we will see, automatic verification procedures for discrete time and finite domain dynamic systems and PLTL specification.

There are generally three methods for system verification: simulation, theorem proving and model checking. Simulation is a procedure of generating partial traces[1] by executing the model, and then checking the set of partial traces against its specification. However, simulation

[1]Note that time might be infinite.

is like program testing, which can only discover errors, but cannot guarantee correctness[2]. Both theorem proving and model checking are formal methods for ensuring correctness.

*Theorem proving* is based on syntactic deduction in a formal system. A *formal system* $\Lambda$ is a pair $\langle A, R\rangle$ consisting of a set of *axioms* $A$ and a set of *rules* $R$ each of which has the form $P_1, \ldots, P_l \Rightarrow P$. A formula $F$ is a *theorem* in $\Lambda$, written $\vdash_\Lambda F$, iff (a) $F$ is an axiom in $A$ or (b) there exists a sequence of theorems $F_1, \ldots, F_m, F$ such that either $F_i$ is an axiom or $F_i$ can be derived from $\{F_1, \ldots, F_{i-1}\}$ using a rule in $R$, namely, there is some $P_1, \ldots, P_l \Rightarrow P$ such that $P = F_i$ and $\{P_1, \ldots, P_l\} \subseteq \{F_1, \ldots, F_{i-1}\}$.

A frame $\mathcal{F}$ is *axiomatizable* iff $\mathcal{F}$ can be captured by a formal system, also denoted by $\mathcal{F}$, such that $F$ is valid over the frame $\mathcal{F}$ iff $\vdash_\mathcal{F} F$, i.e., there is a sound and complete axiomatization.

If we can represent a constraint net $CN$ by a formula, also denoted by $CN$, in the formal system of the specification language $\mathcal{F}$, the behavior of $CN$ satisfies requirements $\mathcal{R}$, written $[\![CN]\!] \models \mathcal{R}$, iff $\vdash_\mathcal{F} CN \rightarrow \mathcal{R}$. For example, a state automaton $s' = f(i, s), s = \delta(s_0)(s')$ in Figure 4.1 can also be represented by a FTLTL formula $\Box(s' = f(i, s)) \bigwedge (s = s_0) \wedge \bigcirc\Box(s = pre(s'))$.

There are some inherent difficulties with the theorem proving approach. First, to be axiomatizable is a strong condition. In fact, according to Goedel's incompleteness theorem, there is no sound and complete axiomatization for any set as complex as natural numbers. Second, even the frame is axiomatizable, there might be no computable decision procedure for an infinite frame. Third, even for finite frames, the problem of checking the validity of a formula is hard in general.

However, in many cases, a proof theoretic approach can assist the verification process. One can always have a set of sound axioms and rules describing the properties of the frame and the logic [Ost89, MP92]. With an interactive theorem prover like HOL — a higher order logic theorem prover developed by Cambridge University and SRI International — one can add more sound axioms and rules for any particular problem at hand. In addition, the reasoning mechanism of theorem proving based on natural deduction might be easier for human to follow.

In conclusion, there are three levels of formal specification for the theorem proving approach:

- *frame specification*: a set of axioms and rules of the temporal logic for the given time structure, a set of axioms and rules characterizing $\Sigma$-domain structure, a set axioms and rules for the given set of predicates;
- *model specification*: a set of formulas specifying the equations of a constraint net;

[2]Symbolic simulation [BS87] is a different procedure that generates symbolic representations of behaviors.

- *requirements specification*: a set of formulas specifying the desired temporal relations on the interface of the module.

We will not discuss further in this thesis the issues on the theorem proving approach, rather, in the rest of this chapter, we will focus on the model checking approach for timed $\forall$-automata specification. Model checking is a formal procedure of verifying behaviors of models. Given the behavior of a system and a timed $\forall$-automaton, *model checking* is to certify the inclusion relation between the behavior and the language accepted by the automaton.

First, we develop a formal verification method for state-based and time-invariant behaviors of discrete time, modified from Manna & Pnueli's verification rules [MP87]. Then, we apply the method to construct a semi-automatic verification procedure for constraint nets with discrete time structures, and translate the verification rules into an automatic algorithm for finite domain systems. Finally, we generalize the verification rules for behaviors of hybrid dynamic systems.

## 11.2 Verification for Behaviors of Discrete Time Systems

Manna & Pnueli [MP87] gave a formal method for checking the validity of a $\forall$-automata specification over a concurrent program. We modify the method to verify state-based and time-invariant behaviors of discrete time. First, we generalize ranking functions to Liapunov functions. Then, we augment timing functions to verify real-time behaviors.

A state-based and time-invariant behavior $\mathcal{B}$ of discrete time corresponds to a state transition system $\langle S_{\mathcal{B}}, \rightarrow\rangle$ with $\Theta$ denoting the initial set of states.

We write $n(s, s')$ iff $s \rightarrow s'$, and $\{\varphi\}\mathcal{B}\{\psi\}$ iff the consecutive condition:

$$\varphi(s) \wedge n(s, s') \rightarrow \psi(s')$$

is valid.

Let $\mathcal{A} = \langle Q, R, S, e, c\rangle$ be a $\forall$-automaton. A set of propositions $\{\alpha_q\}_{q\in Q}$ is called a set of *invariants* for $\mathcal{B}$ and $\mathcal{A}$ iff

- *Initiality*: $\forall q \in Q, \Theta \wedge e(q) \rightarrow \alpha_q$.
- *Consecution*: $\forall q, q' \in Q, \{\alpha_q\}\mathcal{B}\{c(q, q') \rightarrow \alpha_{q'}\}$.

**Proposition 11.2.1** *Let $\{\alpha_q\}_{q\in Q}$ be invariants for $\mathcal{B}$ and $\mathcal{A}$. If $r$ is a run of $\mathcal{A}$ over a trace $v \in \mathcal{B}$, then $\forall t \in \mathcal{T}, v(t) \models \alpha_{r(t)}$.*

Let $\{\alpha_q\}_{q\in Q}$ be a set of invariants for $\mathcal{B}$ and $\mathcal{A}$. A set of partial functions $\{\rho_q\}_{q\in Q}$ is called a set of *Liapunov functions* for $\mathcal{B}$ and $\mathcal{A}$ iff $\rho_q : S_\mathcal{B} \rightarrow \mathcal{R}^+$ satisfies the following conditions:

- *Definedness*: $\forall q \in Q, \alpha_q \rightarrow \exists w, \rho_q = w$.
- *Non-increase*: $\forall q \in S, q' \in Q, \{\alpha_q \wedge \rho_q = w\}\mathcal{B}\{c(q,q') \rightarrow \rho_{q'} \leq w\}$.
- *Decrease*: $\exists \epsilon > 0, \forall q \in B, q' \in Q, \{\alpha_q \wedge \rho_q = w\}\mathcal{B}\{c(q,q') \rightarrow \rho_{q'} - w \leq -\epsilon.\}$

The first two conditions are derived from [MP87]. The last condition generalizes the decrease condition for ranking functions on discrete domains [MP87].

**Proposition 11.2.2** *Let $\{\alpha_q\}_{q\in Q}$ be a set of invariants for $\mathcal{B}$ and $\mathcal{A}$ and $r$ be a run of $\mathcal{A}$ over a trace $v \in \mathcal{B}$. If $\{\rho_q\}_{q\in Q}$ is a set of Liapunov functions for $\mathcal{B}$ and $\mathcal{A}$, then*

- $\rho_{r(t)}(v(t)) \leq \rho_{r(pre(t))}(v(pre(t)))$ *when* $r(pre(t)) \in S$,
- $\rho_{r(t)}(v(t)) - \rho_{r(pre(t))}(v(pre(t))) \leq -\epsilon$ *when* $r(pre(t)) \in B$, *and*
- *if $BS$ is the set of segments of consecutive $B$ and $S$-states in $r$, then $\forall q^* \in BS, q^*$ has a finite number of $B$-states.*

Let $\mathcal{TA} = \langle \mathcal{A}, T, \tau \rangle$. Corresponding to two types of time bound, we define two timing functions. Without loss of generality, we assume that the measurement of time is encoded in the state transition system and let $\mu : S_\mathcal{B} \rightarrow \mathcal{R}^+$ be a function of time measure on states.

Let $\{\alpha_q\}_{q\in Q}$ be a set of invariants for $\mathcal{B}$ and $\mathcal{A}$. A set of partial functions $\{\gamma_q\}_{q\in T}$ is called a set of *local timing functions* for $\mathcal{B}$ and $\mathcal{TA}$ iff $\gamma_q : S_\mathcal{B} \rightarrow \mathcal{R}^+$ satisfies the following conditions:

- *Boundedness*: $\forall q \in T, \alpha_q \rightarrow \mu \leq \gamma_q \leq \tau(q)$.
- *Decrease*: $\forall q \in T, \{\alpha_q \wedge \gamma_q = w \wedge \mu = u\}\mathcal{B}\{c(q,q) \rightarrow \gamma_q - w \leq -u\}$.

A set of partial functions $\{\gamma'_q\}_{q\in Q}$ is called a set of *global timing functions* for $\mathcal{B}$ and $\mathcal{TA}$ iff $\gamma'_q : S_\mathcal{B} \rightarrow \mathcal{R}^+$ satisfies the following conditions:

- *Definedness*: $\forall q \in Q, \alpha_q \rightarrow \exists w, \gamma'_q = w$.
- *Boundedness*: $\forall q \in B, \alpha_q \rightarrow \gamma'_q \leq \tau(bad)$.
- *Non-increase*: $\forall q \in S, q' \in Q, \{\alpha_q \wedge \gamma'_q = w\}\mathcal{B}\{c(q,q') \rightarrow \gamma'_{q'} \leq w\}$.
- *Decrease*: $\forall q \in B, q' \in Q, \{\alpha_q \wedge \gamma'_q = w \wedge \mu = u\}\mathcal{B}\{c(q,q') \rightarrow \gamma'_{q'} - w \leq -u\}$.

**Proposition 11.2.3** *Let $\{\alpha_q\}_{q\in Q}$ be a set of invariants for $\mathcal{B}$ and $\mathcal{A}$ and $r$ be a run of $\mathcal{A}$ over a trace $v \in \mathcal{B}$. If there exist local and global timing functions for $\mathcal{B}$ and $\mathcal{TA}$, then*

- *if $Sg(q)$ is the set of segments of consecutive $q$'s in $r$, then $\forall q \in T, q^* \in Sg(q)$, $\mu(q^*) \leq \tau(q)$, and*
- *if $BS$ is the set of segments of consecutive $B$ and $S$-states in $r$, then $\forall q^* \in BS$, $\mu_B(q^*) \leq \tau(bad)$.*

Following is the set of *verification rules* for a behavior $\mathcal{B}$ and a timed automaton $\mathcal{TA} = \langle \mathcal{A}, T, \tau \rangle$:

(I) Associate with each automaton-state $q \in Q$ a state formula $\alpha_q$, such that $\{\alpha_q\}_{q\in Q}$ is a set of invariants for $\mathcal{B}$ and $\mathcal{A}$.

(L) Associate with each automaton-state $q \in Q$ a partial function $\rho_q$, such that $\{\rho_q\}_{q\in Q}$ is a set of Liapunov functions for $\mathcal{B}$ and $\mathcal{A}$.

(T) Associate with each timed automaton-state $q \in T$ a partial function $\gamma_q$, such that $\{\gamma_q\}_{q\in T}$ is a set of local timing functions for $\mathcal{B}$ and $\mathcal{TA}$. Associate with each automaton-state $q \in Q$ a partial function $\gamma'_q$, such that $\{\gamma'_q\}_{q\in Q}$ is a set of global timing functions for $\mathcal{B}$ and $\mathcal{TA}$.

**Theorem 11.2.1** *For any state-based and time-invariant behavior $\mathcal{B}$ with an infinite time structure and a complete timed $\forall$-automaton $\mathcal{TA}$, the verification rules are sound and complete, i.e., $\mathcal{B} \models \mathcal{TA}$ iff there exist a set of invariants, Liapunov functions and timing functions.*

We shall provide the proof of this theorem next, since the proof itself will be used later in the verification algorithm for behaviors of finite state systems.

Proof: The construction of these rules guarantees the soundness of the verification method. For any trace $v$, there is a run because $\mathcal{TA}$ is complete. For any run $r$ over $v$, if any automaton-state in $R$ appears infinitely many times in $r$, $r$ is accepting. Otherwise, there is a time point $t_0 \in \mathcal{T}$, the sub-sequence $r$ on $I = \{t \in \mathcal{T} | t \geq t_0\}$, denoted $q^*$, has only bad and stable automaton-states. If there exist a set of invariants and a set of Liapunov functions, $q^*$ has only a finite number of $B$-states. Since time is infinite, all the automaton-states appearing infinitely many times in $r$ belong to $S$; so $r$ is accepting too. Therefore, every trace is accepting for the automaton. If there exists a set of local and global timing functions, every trace satisfies the timing constraints.

On the other hand, if $\mathcal{TA}$ is valid over $\mathcal{B}$, then there exist a set of invariants, a set of Liapunov functions, and a set of local and global timing functions that satisfy the requirements. The construction of invariants and functions will be used later for the verification algorithm.

For any state $s$ and proposition $\alpha$, we write $\alpha(s)$ iff $s \models \alpha$. The invariants can be constructed as the fixpoint of the set of equations:

$$\alpha_{q'}(s') = (\exists q, s, \alpha_q(s) \wedge n(s, s') \wedge c(q, q')(s')) \bigvee (\Theta(s') \wedge e(q')(s')). \tag{11.1}$$

We can verify that $\{\alpha_q\}_{q \in Q}$ is a set of propositions over $S_{\mathcal{B}}$ and satisfies the requirements of initiality and consecution. Furthermore, $s \models \alpha_q$ iff $\langle q, s\rangle$ is a reachable pair for $\mathcal{TA}$ and $\mathcal{B}$.

Given the constructed invariants $\{\alpha_q\}_{q \in Q}$, a set of Liapunov functions $\{\rho_q\}_{q \in Q}$ and a set of global timing functions $\{\gamma'_q\}_{q \in Q}$ can be constructed as follows:

- $\forall q \in R, s \models \alpha_q$, let $\rho_q(s) = 0$ and $\gamma'_q(s) = 0$.
- $\forall q \notin R, s \models \alpha_q$, $\rho_q(s)$ and $\gamma'_q(s)$ are defined as follows. Construct a directed graph $G = \langle V, E\rangle$, such that $\langle q, s\rangle \in V$ iff $q \notin R, s \models \alpha_q$, and $\langle q, s\rangle \rightarrow \langle q', s'\rangle$ in $E$ iff $n(s, s') \wedge c(q, q')(s')$. For any path $p$ starting at $\langle q, s\rangle$, let $|p|_B$ be the number of $B$-states in $p$ and $\mu_B(p)$ be the measure of $B$-states in $p$. Let $\rho_q(s) = \sup\{|p|_B\}$ and $\gamma'_q(s) = \sup\{\mu_B(p)\}$.

We can verify that $\{\rho_q\}_{q \in Q}$ is a set of Liapunov functions, and that $\{\gamma'_q\}_{q \in Q}$ is a set of global timing functions.

Similarly, a set of local timing functions $\{\gamma_q\}_{q \in T}$ can be constructed as follows. For all $q \in T$, construct a directed graph $G = \langle V, E\rangle$, such that $s \in V$ iff $s \models \alpha_q$, and $s \rightarrow s'$ in $E$ iff $n(s, s') \wedge c(q, q)(s')$. For any path $p$ starting at $s$, let $\mu(p)$ be the measure of the path. Let $\gamma_q(s) = \sup\{\mu(p)\}$. We can verify that $\{\gamma_q\}_{q \in T}$ is a set of local timing functions. □

This verification method for behaviors of discrete time systems will be the basis of verification for behaviors of hybrid dynamic systems. On the other hand, many hybrid systems can be verified at different levels of implementation. If a system has an event-driven component, we can verify, using this method, the discrete time behavior, where the time is generated by events. For the maze traveler example, the persistent property — the robot moves to the east infinitely many times represented by the $\forall$-automaton in Figure 10.2 — can be verified at the strategy level. We can construct a state transition system $\langle S, \rightarrow\rangle$ such that $S$ is the set of configurations of the car and $\rightarrow$ is the state transition relation derived from the strategy. Formally, let $\langle x, y\rangle \in \mathcal{R} \times \mathcal{R}$ and $\theta \in \mathcal{R}$ be the position and the orientation of the car, respectively, and let $\langle x, y, \theta\rangle \rightarrow \langle x', y', \theta\rangle$ iff $\langle x', y', \theta'\rangle$ is the configuration of the car at the next event according to the strategy. Associate with $q_0$ and $q_1$ the state proposition $\neg(|\theta| < \delta)$ and $|\theta| < \delta$, respectively; $q_0$ and $q_1$ are invariants. Associate with $q_0$ a function $\rho : \mathcal{R} \times \mathcal{R} \times \mathcal{R} \rightarrow \mathcal{R}^+$ such that $\rho(x, y, \theta)$ is the distance between the current configuration and the "desired" configuration with heading $|\theta| < \delta$. Associate with $q_1$ a constant function 0. Given that the block sizes are finite, $\rho$ and 0 are are Liapunov functions for $q_0$ and $q_1$, respectively. Therefore, the maze traveler controlled by the strategy will satisfy the desired property.

### 11.2.1 Semi-automatic verification

Now we apply the verification rules to constraint nets with discrete time structures. Let $CN = \langle Lc, Td, Cn \rangle$ be a constraint net composed of transliterations and unit delays only. $CN$ can be represented by two sets of *domain equations*, each of the form $l_0' = l$, if $l_0$ is an output location of a unit delay with the input location $l$, or $l_0 = f(l_1, \ldots, l_n)$, if $l_0$ is an output location of a transliteration $f$ with the input location tuple $\langle l_1, \ldots, l_n \rangle$. For example, consider the producer-consumer circuit in Figure 5.3, and assume that any delay is unit (if not, it can be modeled by a finite number of unit delays), the domain equations for the control circuit are:

$$C1 = mc(R1, \neg Q2, Q1), \qquad C2 = mc(Q1, \neg R2, Q2) \tag{11.2}$$

$$Q1' = C1, \qquad Q2' = C2. \tag{11.3}$$

Let $\mathcal{T}$ be a discrete time structure and $\mathcal{A}$ be a domain structure. The behavior of $CN$ on dynamics structure $\mathcal{D}(\mathcal{T}, A)$ corresponds to a state transition system $\langle S, \rightarrow \rangle$ where (1) $S \subseteq \times_{Lc} A_l$ and $s \in S$ iff for every equation of the form $l_0 = f(l_1, \ldots, l_n)$, $s(l_0) = f(s(l_1), \ldots, s(l_n))$, and (2) $s \rightarrow s'$ iff for every equation of the form $l_0' = l$, $s'(l_0) = s(l)$. However, the behavior of $CN$ can be verified without generating its state transition system.

Let $CN_t \equiv \bigwedge\{l_0 = f(l_1, \ldots, l_n)\}$ and $CN_d \equiv \bigwedge\{l_0' = l\}$. Let $\varphi$ and $\psi$ be state formulas with a subset of $Lc$ as local variables. We use $[\varphi]CN[\psi]$ to denote that the consistent condition:

$$\varphi \wedge CN_t \rightarrow \psi$$

is valid, and $\{\varphi\}CN\{\psi\}$ to denote that the consecutive condition:

$$\varphi \bigwedge CN_t \wedge CN_d \wedge CN_t[l'/l] \rightarrow \psi[l'/l]$$

is valid, where $x'/x$ denotes the replacement of $x$ by $x'$.

Let $\Theta$ be a state formula imposing constraints on the set of initial states of $CN$. Let $\mathcal{A} = \langle Q, R, S, e, c \rangle$ be a $\forall$-automaton. A set of state propositions $\{\alpha_q\}_{q \in Q}$ is called a set of *invariants* for $CN$ and $\mathcal{A}$ iff

- *Initiality*: $\forall q \in Q, [\Theta \wedge e(q)]CN[\alpha_q]$.
- *Consecution*: $\forall q, q' \in Q, \{\alpha_q\}CN\{c(q, q') \rightarrow \alpha_{q'}\}$.

Let $\{\alpha_q\}_{q \in Q}$ be a set of invariants for $CN$ and $\mathcal{A}$. A set of partial functions $\{\rho_q\}_{q \in Q}$ is called a set of *Liapunov functions* for $CN$ and $\mathcal{A}$ iff $\rho_q : \times_{Lc} A_{s_l} \rightarrow \mathcal{R}^+$, satisfies the following conditions:

- *Definedness*: $\forall q \in Q, [\alpha_q]CN[\exists w, \rho_q = w]$.
- *Non-increase*: $\forall q \in S, q' \in Q, \{\alpha_q \wedge \rho_q = w\}CN\{c(q, q') \rightarrow \rho_{q'} \leq w\}$.
- *Decrease*: $\exists \epsilon > 0, \forall q \in B, q' \in Q, \{\alpha_q \wedge \rho_q = w\}CN\{c(q, q') \rightarrow \rho_{q'} - w \leq -\epsilon.\}$

Let $\mathcal{TA} = \langle \mathcal{A}, T, \tau \rangle$. Corresponding to two types of time bound, we define two timing functions. Without loss of generality, we assume that the measurement of time is encoded in a location and let $\mu : \times_{Lc} A_{s_l} \rightarrow \mathcal{R}^+$ be a function of time measure. Let $\{\alpha_q\}_{q \in Q}$ be a set of invariants for $CN$ and $\mathcal{A}$. A set of partial functions $\{\gamma_q\}_{q \in T}$ is called a set of *local timing functions* for $\mathcal{B}$ and $\mathcal{TA}$ iff $\gamma_q : \times_{Lc} A_{s_l} \rightarrow \mathcal{R}^+$ satisfies the following conditions:

- *Boundedness*: $\forall q \in T, [\alpha_q]CN[\mu \leq \gamma_q \leq \tau(q)]$.
- *Decrease*: $\forall q \in T, \{\alpha_q \wedge \gamma_q = w \wedge \mu = u\}CN\{c(q, q) \rightarrow \gamma_q - w \leq -u\}$.

A set of partial functions $\{\gamma'_q\}_{q \in Q}$ is called a set of *global timing functions* for $CN$ and $\mathcal{TA}$ iff $\gamma'_q : \times_{Lc} A_{s_l} \rightarrow \mathcal{R}^+$ satisfies the following conditions:

- *Definedness*: $\forall q \in Q, [\alpha_q]CN[\exists w, \gamma'_q = w]$.
- *Boundedness*: $\forall q \in B, [\alpha_q]CN[\gamma'_q \leq \tau(bad)]$.
- *Non-increase*: $\forall q \in S, q' \in Q, \{\alpha_q \wedge \gamma'_q = w\}CN\{c(q, q') \rightarrow \gamma'_{q'} \leq w\}$.
- *Decrease*: $\forall q \in B, q' \in Q, \{\alpha_q \wedge \gamma'_q = w \wedge \mu = u\}CN\{c(q, q') \rightarrow \gamma'_{q'} - w \leq -u\}$.

We say that the verification method based on this set of rules is *semi-automatic* because given the invariants, Liapunov functions and timing functions, the method is reduced to checking the validity of a set of formulas in the domain structure $A$. If there is a first order theorem prover for the domain structure $A$, the procedure can be done semi-automatically.

Now we illustrate the verification method using an example. Some other examples are also studied [ZM94].

A desired property of the asynchronous event controller has been expressed by the $\forall$-automaton in Figure 10.2(a). The automaton is not complete. To make it complete, introduce an error state $q_E$ with $e(q_E) = false, c(q_E, q_E) = true, c(q_E, q_i) = false$, and let $c(q_0, q_E)$ be $(Q1 =\perp) \vee (Q2 =\perp) \vee (C1 =\perp) \vee (C2 =\perp) \vee neq(C2, Q2)$ and $c(q_1, q_E)$ be $(Q1 =\perp) \vee (Q2 =\perp) \vee (C1 =\perp) \vee (C2 =\perp) \vee neq(C1, Q1)$. The domain equations of the controller have been expressed in Equations 11.2 and 11.3.

Let the initial condition $\Theta$ be $Q1 = Q2 = 0, R1 = 0, R2 = 1$, and assume that values at $R1$ and $R2$ are always well-defined. Let $AEC_t$ denote the conjunction of domain equations in 11.2 with $\neg(R1 =\perp)$ and $\neg(R2 =\perp)$, and $AEC_d$ denote the conjunction of domain equations in 11.3. Furthermore, let $AEC$ denote the conjunction of all domain equations, $AEC_t \wedge AEC_d \wedge AEC_t[l'/l]$.

(I) Associate with $q_0, q_1, q_E$ the state propositions $eq(C1, C2)$, $neq(C1, C2)$ and $false$, respectively. The following verification conditions are satisfied:

- *Initiality:*

$$q_0 : \Theta \wedge true \wedge AEC_t \rightarrow eq(C1, C2).$$
$$q_1 : \Theta \wedge false \wedge AEC_t \rightarrow neq(C1, C2).$$
$$q_E : \Theta \wedge false \wedge AEC_t \rightarrow false.$$

- *Consecution:*

$$(q_0, q_0) : eq(C1, C2) \wedge AEC \rightarrow (eq(C1', Q1') \wedge eq(C2', Q2') \rightarrow eq(C1', C2')).$$
$$(q_0, q_1) : eq(C1, C2) \wedge AEC \rightarrow (neq(C1', Q1') \wedge eq(C2', Q2') \rightarrow neq(C1', C2')).$$
$$(q_0, q_E) : eq(C1, C2) \wedge AEC \rightarrow$$
$$((Q1' =\perp) \vee (Q2' =\perp) \vee (C1' =\perp) \vee (C2' =\perp) \vee neq(C2', Q2') \rightarrow false).$$

...

Therefore, $eq(C1, C2)$, $neq(C1, C2)$ and $false$ are invariants for $q_0, q_1$ and $q_E$, respectively.

(L) Since $q_0, q_1 \in R$ and the invariant of $q_2 \in B$ is $false$, any set of functions is a set of Liapunov functions for $q_0, q_1$ and $q_E$.

Therefore, according to the verification rules, the behavior of the constraint net satisfies its requirements specification.

### 11.2.2 Automatic verification

The existence of the semi-automatic verification method for constraint nets presented in the previous section does not necessarily imply the existence of an automatic procedure. First, the invariants, Liapunov functions and the timing functions are defined separately, and not automatically generated. Second, there is, of course, no decision procedure for determining the validity of a first-order formula in general.

However, for finite constraint nets — nets with finite domains — we can automate the verification process against a timed $\forall$-automata specification. Derived from the verification rules, the algorithm consists of three phases:

1. Invariant Generation,

2. Boundedness and Global Timing, and

3. Local Timing.

Let $CN = \langle Lc, Td, Cn \rangle$ be a constraint net composed of transliterations and unit delays only. We write $CN(s)$ iff for every equation of the form $l_0 = f(l_1, \ldots, l_n)$, $s(l_0) = f(s(l_1), \ldots, s(l_n))$, and $CN(s, s')$ iff $CN(s), CN(s')$, and for every equation of the form $l_0' = l$, $s'(l_0) = s(l)$.

Invariant generation is a process that produces all reachable pairs of $(q, s)$, denoted $a(q, s)$, where $q \in Q$ and $s \in \times_{Lc} A_{s_l}$. According to Equation 11.1, this fixpoint operation can be efficiently realized in two steps:

1. *Initiality*: Generate $a(q, s)$ if $\Theta(s), e(q)(s), CN(s)$.

2. *Consecution*: Generate $a(q', s')$ if $a(q, s), CN(s, s'), c(q, q')(s')$.

The algorithm is shown in Figure 11.1, where $start(s)$ denotes $\Theta(s)$.

We write $bstate(q, s)$ iff $a(q, s)$ and $q \in B$, and $sstate(q, s)$ iff $a(q, s)$ and $q \in S$. Let $\langle V, E \rangle$ be the state transition graph where $V$ is the set of pairs $(q, s)$ satisfying $sstate(q, s)$ or $bstate(q, s)$, $E$ is the set of transitions $(q, s, q', s')$ between two states in $V$, $(q, s, q', s') \in E$ if $CN(s, s')$ and $c(q, q')(s')$. Boundedness checks whether or not there is a loop consisting of $bstate(q, s)$ in the state transition graph. Global timing checks whether or not there is a path $p$ in the state transition graph whose time measure of $bstate(q, s)$, denoted $m(p)$, is greater than the time bound $\tau(bad)$, denoted $time(bad)$. The algorithm is shown in Figure 11.2.

For each $q \in T$ let $\langle V, E \rangle$ be the state transition graph where $V$ is the set of $s$ satisfying $a(q, s)$ and $E$ is the set of transitions $(s, s')$ between two states in $V$, $(s, s') \in E$ if $CN(s, s')$ and

```
Algorithm: Invariant Generation
Qs = [];
Rs = [];
for all q, s do /* Initiality */
    if start(s) and e(q)(s) and CN(s)
       { Qs = [a(q, s)|Qs];
         Rs = [a(q, s)|Rs];
       }
while Qs = [a(q, s)|Qs1] do /* Consecution */
{
    NRs = [];
    for all q', s' do
    if a(q, s) and CN(s, s') and c(q, q')(s')
       and a(q', s') not in Rs
    NRs = [a(q', s')|NRs];

    Rs = append(Rs, NRs);
    Qs = append(Qs1, NRs);
}
```

Figure 11.1: The algorithm for invariant generation

```
Algorithm: Boundedness and Global Timing
1. /* Generate state transition graph <V,E> */
   for all q in B do
    for all s do
     if a(q, s) put bstate(q, s) in V
   for all q in S do
    for all s do
     if a(q, s) put sstate(q, s) in V
   for all (q, s), (q', s') in V do
     if CN(s, s') and c(q, q')(s')
     put (q, s, q', s') in E
2. /* Check the acyclicity of bstate */
   for all bstate(q, s) in V do
     for all path p starting from (q, s) do
        if p ends at (q, s) return false
   /* Check the time bound of bstate */
   for all bstate(q, s) in V do
     for all path p starting from (q, s) do
        if m(p) > time(bad) return false
   return true
```

Figure 11.2: The algorithm for boundedness and global timing

$c(q,q)(s')$. Local timing checks whether or not there is a path $p$ in the state transition graph whose time measure, denoted $m(p)$, is greater than the time bound $\tau(q)$, denoted $time(q)$. The algorithm is shown in Figure 11.3.

```
Algorithm: Local Timing
for all q in T do
    if not ttest(q) return false
return true
ttest(q):
    1. /* Generate state transition graph <V,E> */
       for all s do
        if a(q, s) put s in V
       for all s, s' in V do
        if CN(s, s') and c(q, q)(s')
          put (s, s') in E
    2. /* Check the longest path */
       for all s in V with no input edges do
          for all path p starting from s do
            if m(p) > time(q) or p has loop return false
       return true
```

Figure 11.3: The algorithm for local timing

The complexity of the verification algorithm is obtained as follows. The invariant generation can be done in polynomial time in $|Q| \times_{Lc} |A_{s_l}|$, which is the total number of $(q,s)$ pairs. For each $bstate(q,s)$, searching for a loop including $bstate(q,s)$ or a longest bad state path starting at $bstate(q,s)$ is linear in the number of transitions in the state transition graph, since each state needs to be visited only its outdegree number of times in the search algorithm. Therefore, both checking boundedness and global timing are polynomial in $|Q| \times_{Lc} |A_{s_l}|$. Similarly, checking local timing is in polynomial in $|Q| \times_{Lc} |A_{s_l}|$.

As a result, the verification algorithm is polynomial in both the size of the model and the size of the specification. This result seems a little surprising, since it is well-known [Eme90] that model checking for the linear propositional temporal logic is $PSPACE$-complete in the length of the formula. However, we should notice that, in the worst case, the size of a $\forall$-automaton may be exponential in the length of its equivalent linear propositional temporal logic formula.

On the other hand, for many system properties, such as safety, liveness, reachability and bounded response, $\forall$-automata do have size equivalent to the length of their corresponding linear propositional temporal logic formulas. However, the number of automaton-states in the complement of a $\forall$-automaton may be exponential in the number of automaton-states in the original $\forall$-automaton [Tho90]. This suggests that we should choose the simpler $\forall$-automaton, $\mathcal{A}$ or $\neg\mathcal{A}$, as a basis to verify finite systems.

However, we should also notice that even though the complexity of the algorithm is polynomial in the size of the model, it is exponential in the number of local variables or locations of the constraint net. In most cases, a property of a system is expressed by only a small subset of locations, for example, locations in the interface of a module. If the algorithm can explore only this small portion of the system, there is an exponential savings in complexity.

For a constraint net $CN = \langle Lc, Td, Cn \rangle$, let $\rightarrow_{Lc}$ denote the transition relation of $CN$, i.e., $s_1 \rightarrow_{Lc} s_2$ iff $CN(s_1, s_2)$. For a subset of locations $U \subset Lc$, let $\rightarrow_U$ denote the projected relation, i.e., $s'_1 \rightarrow_U s'_2$ iff $\exists s_1, s_2,\ s'_1 = h(s_1)$ and $s'_2 = h(s_2)$, such that $CN(s_1, s_2)$, where $h = \lambda s.s_{|U}$. $U$ is an *abstraction* of the set of locations $Lc$ for $CN$ iff $\langle \times_{Lc} A_{s_l}, \rightarrow_{Lc} \rangle$ is abstractable to $\langle \times_U A_{s_l}, \rightarrow_U \rangle$. The following proposition provides an equivalent definition of this concept.

**Proposition 11.2.4** *Given $Lc$ as the set of locations and $U \subseteq Lc$, $U$ is an abstraction of $Lc$ iff $[\![CN(U)]\!]$ is state-based and time-invariant.*

The following propositions underpin the application of this concept of abstraction.

**Proposition 11.2.5** *If $U$ is an abstraction of $Lc$, any property restricted on relations on $U$ can be verified by exploring the abstraction transition system, $\langle \times_U A_{s_l}, \rightarrow_U \rangle$.*

**Proposition 11.2.6** *If $CN_s$ is a subnet of $CN$, the set of locations of $CN_s$ is an abstraction.*

**Proposition 11.2.7** *The set of output locations of unit delays is an abstraction.*

**Proposition 11.2.8** *The set of input locations of unit delays is an abstraction.*

**Proposition 11.2.9** *If $U$ is an abstraction and $I \subseteq I(CN)$, $U \cup I$ or $U - I$ is still an abstraction.*

We have implemented the verification algorithm in Prolog, where the model is represented by the initial state predicate $start(s)$ and the state transition predicate $cn(s, s')$, the specification is represented by the entry condition predicate $e(q, s)$ and the consecution condition predicate $c(q, q', s)$. For simplicity, each state is assumed to take one unit time. Examples of the producer-consumer synchronizer, with an interleaving property, and an elevator system (in Appendix C), with a real-time response property, have been verified in this implementation.

## 11.3 Verification for Behaviors of Hybrid Dynamic Systems

Now we generalize the verification rules for behaviors of hybrid dynamic systems.

The set of verification rules is the same as that for behaviors of discrete time systems, however, the definitions of invariants, Liapunov functions and timing functions are generalized.

For any trace $v : \mathcal{T} \to A$, let $\{\varphi\}v\{\psi\}$ denote the validity of the following two consecutive conditions:

- $\{\varphi\}v^{-}\{\psi\}$: for all $t > \mathbf{0}$, $\exists t' < t, \forall t'', t' \leq t'' < t, v(t'') \models \varphi$ implies $v(t) \models \psi$.
- $\{\varphi\}v^{+}\{\psi\}$: for all $t < \infty$, $v(t) \models \varphi$ implies $\exists t' > t, \forall t'', t < t'' < t', v(t'') \models \psi$.

If $\mathcal{T}$ is discrete, these two conditions are reduced to one, i.e., $\forall t > \mathbf{0}, v(pre(t)) \models \varphi$ implies $v(t) \models \psi$.

Given $\mathcal{B}$ as a behavior, let $\Theta = \{v(\mathbf{0}) | v \in \mathcal{B}\}$ denote the set of initial values in $\mathcal{B}$. Let $\mathcal{A} = \langle Q, R, S, e, c\rangle$ be a $\forall$-automaton. A set of propositions $\{\alpha_q\}_{q\in Q}$ is called a set of *invariants* for $\mathcal{B}$ and $\mathcal{A}$ iff

- *Initiality*: $\forall q \in Q, \Theta \wedge e(q) \to \alpha_q$.
- *Consecution*: $\forall v \in \mathcal{B}$, $\forall q, q' \in Q, \{\alpha_q\}v\{c(q, q') \to \alpha_{q'}\}$.

**Proposition 11.3.1** *Let $\{\alpha_q\}_{q\in Q}$ be invariants for $\mathcal{B}$ and $\mathcal{A}$. If $r$ is a run of $\mathcal{A}$ over $v \in \mathcal{B}$, $\forall t \in \mathcal{T}, v(t) \models \alpha_{r(t)}$.*

Without loss of generality, we assume that time is encoded in domain $A$ by $t_c : A \to \mathcal{T}$. Given that $\{\alpha_q\}_{q\in Q}$ is a set of invariants for $\mathcal{B}$ and $\mathcal{A}$, a set of partial functions $\{\rho_q\}_{q\in Q} : A \to \mathcal{R}^+$ is called a set of *Liapunov functions* for $\mathcal{B}$ and $\mathcal{A}$ iff the following conditions are satisfied:

- *Definedness*: $\forall q \in Q, \ \ \alpha_q \to \exists w, \rho_q = w$.
- *Non-increase*: $\forall v \in \mathcal{B}$, $\forall q \in S, q' \in Q$,

  $$\{\alpha_q \wedge \rho_q = w\}v^{-}\{c(q, q') \to \rho_{q'} \leq w\}$$

  and $\forall q \in Q, q' \in S$,

  $$\{\alpha_q \wedge \rho_q = w\}v^{+}\{c(q, q') \to \rho_{q'} \leq w\}.$$

- *Decrease*: $\forall v \in \mathcal{B}$, $\exists \epsilon > 0$, $\forall q \in B, q' \in Q$,

$$\{\alpha_q \wedge \rho_q = w \wedge t_c = t\} v^- \{c(q, q') \rightarrow \frac{\rho_{q'} - w}{\mu([t, t_c))} \leq -\epsilon\}$$

  and $\forall q \in Q, q' \in B$,

$$\{\alpha_q \wedge \rho_q = w \wedge t_c = t\} v^+ \{c(q, q') \rightarrow \frac{\rho_{q'} - w}{\mu([t, t_c))} \leq -\epsilon\}.$$

**Proposition 11.3.2** *Let $\{\alpha_q\}_{q \in Q}$ be invariants for $\mathcal{B}$ and $\mathcal{A}$ and $r$ be a run of $\mathcal{A}$ over a trace $v \in \mathcal{B}$. If $\{\rho_q\}_{q \in Q}$ is a set of Liapunov functions for $\mathcal{B}$ and $\mathcal{A}$, then*

- $\rho_{r(t_2)}(v(t_2)) \leq \rho_{r(t_1)}(v(t_1))$ *when* $\forall t_1 \leq t \leq t_2, r(t) \in B \cup S$,
- $\frac{\rho_{r(t_2)}(v(t_2)) - \rho_{r(t_1)}(v(t_1))}{\mu([t_1, t_2))} \leq -\epsilon$ *when* $t_1 < t_2$ *and* $\forall t_1 \leq t \leq t_2, r(t) \in B$, *and*
- *if $BS$ is the set of segments of consecutive $B$ and $S$-states in $r$, then $\forall q^* \in BS, \mu_B(q^*)$ is finite.*

Let $\mathcal{TA} = \langle \mathcal{A}, T, \tau \rangle$. Corresponding to two types of time bound, we define two timing functions. Let $\{\alpha_q\}_{q \in Q}$ be invariants for $\mathcal{B}$ and $\mathcal{A}$. A set of partial functions $\{\gamma_q\}_{q \in T}$ is called a set of *local timing functions* for $\mathcal{B}$ and $\mathcal{TA}$ iff $\gamma_q : A \rightarrow \mathcal{R}^+$ satisfies the following conditions:

- *Boundedness*: $\forall v \in \mathcal{B}$, $\forall q \in Q, q' \in T$,

$$\{\alpha_q\} v^- \{\gamma_{q'} \leq \tau(q')\}$$

  and $\forall q \in T, q' \in Q$,

$$\{\alpha_q \wedge t_c = t \wedge \gamma_q = w\} v^- \{w \geq \mu([t, t_c))\}.$$

- *Decrease*: $\forall v \in \mathcal{B}$, $\forall q \in T, \{\alpha_q \wedge \gamma_q = w \wedge t_c = t\} v \{c(q, q) \rightarrow \frac{\gamma_q - w}{\mu([t, t_c))} \leq -1\}$.

A set of partial functions $\{\gamma'_q\}_{q \in Q}$ is called a set of *global timing functions* for $\mathcal{B}$ and $\mathcal{TA}$ iff $\gamma'_q : A \rightarrow \mathcal{R}^+$ satisfies the following conditions:

- *Definedness*: $\forall q \in Q, \alpha_q \rightarrow \exists w, \gamma'_q = w$.
- *Boundedness*: $\forall q \in B, \alpha_q \rightarrow \gamma'_q \leq \tau(bad)$.
- *Non-increase*: $\forall v \in \mathcal{B}$, $\forall q \in S, q' \in Q$,

$$\{\alpha_q \wedge \gamma'_q = w\} v^- \{c(q, q') \rightarrow \gamma'_{q'} \leq w\}$$

  and $\forall q \in Q, q' \in S$,

$$\{\alpha_q \wedge \gamma'_q = w\} v^+ \{c(q, q') \rightarrow \gamma'_{q'} \leq w\}.$$

- *Decrease*: $\forall v \in \mathcal{B}, \forall q \in B, q' \in Q$,

$$\{\alpha_q \wedge \gamma'_q = w \wedge t_c = t\} v^- \{c(q,q') \to \frac{\gamma'_{q'} - w}{\mu([t,t_c))} \leq -1\}$$

  and $\forall q \in Q, q' \in B$,

$$\{\alpha_q \wedge \gamma'_q = w \wedge t_c = t\} v^+ \{c(q,q') \to \frac{\gamma'_{q'} - w}{\mu([t,t_c))} \leq -1\}.$$

**Proposition 11.3.3** *Let $\{\alpha_q\}_{q \in Q}$ be invariants for $\mathcal{B}$ and $\mathcal{A}$ and $r$ be a run of $\mathcal{A}$ over a trace $v \in \mathcal{B}$. If there exist local and global timing functions for $\mathcal{B}$ and $\mathcal{TA}$, then*

- *if $Sg(q)$ is the set of segments of consecutive $q$'s in $r$, then $\forall q \in T, q^* \in Sg(q)$, $\mu(q^*) \leq \tau(q)$, and*
- *if $BS$ is the set of segments of consecutive $B$ and $S$-states in $r$, then $\forall q^* \in BS$, $\mu_B(q^*) \leq \tau(bad)$.*

The following theorem is a generalization of the soundness and completeness of the set of verification rules.

**Theorem 11.3.1** *The verification rules (I), (L) and (T) are sound if the following conditions on $\mathcal{B}$ and $\mathcal{TA}$ are satisfied:*

- *$\mathcal{T}$ is an infinite time structure.*
- *All traces in $\mathcal{B}$ are specifiable by $\mathcal{TA}$.*

*The verification rules are complete if the following conditions on $\mathcal{B}$ and $\mathcal{TA}$ are satisfied:*

- *$\{\langle v, r\rangle | v \in \mathcal{B}, r$ is a run over $v\}$ is time-invariant.*
- *All transitions from $R$ to non-$R$-states are left-closed, i.e., if $r$ is a run, and there is a transition from a $R$-state to a $B$-state or a $S$-state at $t$, then $r(t) \in B \cup S$.*

The conditions for the completeness of the rules are imposed so as to be able to define Liapunov functions for a behavior and an automaton, as long as the behavior satisfies the automaton. The second condition for completeness is always satisfied for traces with discrete time structures. More generally, the following proposition may apply.

**Proposition 11.3.4** *All transitions from $R$ to non-$R$-states are left-closed, if the following conditions are satisfied:*

- $\mathcal{TA}$ *is open and complete.*
- $\forall q \in R$, $q_1 \notin R$ *and* $q_2 \in R$, $c(q, q_1) \wedge c(q, q_2)$ *is not satisfiable.*
- *All traces in* $\mathcal{B}$ *are right-continuous.*

This formal method has no practical use yet; we aim at understanding the concept of behavior verification for hybrid systems. In part III, we will discuss an important class of behavior with asymptotic properties. By characterizing certain types of hybrid system and property, we may obtain a semi-automatic verification method, similar to the one for discrete time systems. There is much more left to be explored than what we have already understood.

# Chapter 12

# Summary and Related Work

We have developed two requirements specification languages, TLTL and timed $\forall$-automata, for representing desired global properties of dynamic systems. We have also developed a set of formal verification rules for timed $\forall$-automata specification. In this chapter, we summarize the results of Part II and discuss some related work on specification and verification.

## 12.1 Summary

In this section, we summarize the specification languages and the verification procedures, then discuss their power and limitations.

### 12.1.1 Specification

Timed Linear Temporal Logic (TLTL) has the following properties:

- Simple properties of dynamic systems (such as safety, reachability and persistence) can be specified.
- Some metric or measure properties of dynamic systems (such as real-time response) can be specified.
- TLTL is defined for arbitrary time and domain structures; therefore, continuous as well as discrete time dynamic systems can be specified in a unitary framework.

Timed $\forall$-automata have the following properties:

- They are a simple alternative, though not equivalent in expressive power, to TLTL.
- They have a graphical representation.

- They are powerful enough to specify many important properties of sequential and timed behaviors.
- They are simple enough to have a formal verification procedure for behaviors of hybrid dynamic systems, a semi-automatic verification procedure for discrete time systems, and an automatic verification procedure for discrete time and finite domain systems.

### 12.1.2 Verification

The verification procedures have the following properties:

- A model checking technique and a stability analysis method are integrated.
- The automatic algorithm derived from the verification rules has a polynomial time complexity in both the size of the model and the size of the specification.
- The generalized verification rules can be used to formally verify behaviors of hybrid dynamic systems.

### 12.1.3 Power and limitations

Both TLTL and timed $\forall$-automata are powerful enough to specify various properties of sequential and timed behaviors. However, there are still many important behaviors that cannot be specified in these languages, such as

- energy minimization over time, i.e., $\min \int_{\mathcal{T}} \mathcal{E} dt$, where $\mathcal{E}$ is a function of states,
- probabilistic or stochastic properties, and
- timed properties on intervals.

However, we should also point out that the power of specification and the simplicity of verification are in conflict with each other. The more powerful the specification language is, the more complex is the verification procedure. A compromise between these two should be made for any application.

Although most research in this area mixes modeling and specification languages, we claim that two different kinds of language are necessary for specifying two different aspects of systems and behaviors: composite structures and global functionalities. We have not yet worked on axiomization for TLTL, since we focused on model checking, rather than theorem proving, for behavior verification.

## 12.2 Related Work

Various languages for specification, verification, and reasoning about concurrent, distributed and timed behaviors have been developed in the theory, AI and systems communities. Roughly speaking, these languages can be characterized as belonging to one of the three categories: (1) Automata, (2) Point Time Temporal Logics, and (3) Interval Time Temporal Logics. In any of these languages, there are always two ways to introduce real-time (metric time). One is to embed metric time in modal operators, the other is to use an explicit time variable. Different languages can have different expressive power; some of them may have no formal verification procedures at all.

We survey some typical examples in every category, and discuss their relationships with TLTL and timed $\forall$-automata.

### 12.2.1 Automata-based approaches

Automata play two kinds of role: as an input/output transducer modeling on-line computation (e.g., Mealy/Moore machines), or as a language recognizer (e.g., $\forall$-automata). We have surveyed some related work on automata for modeling in Part I. Here we emphasize their roles for specification and verification.

The simplest form of an automata-based representation for sequential behaviors is Buchi automata [Tho90]. Buchi automata are finite state automata for defining $\omega$-languages, languages consisting of infinite sequences. The expressive power of Buchi automata is the same as that of $\forall$-automata [MP87]. In fact, a restricted version of $\forall$-automata is a dual of Buchi automata [MP87].

Timed Buchi Automata (TBA) has been proposed [AD90] to express constant bounds on timing delays between system events. These automata accept languages of timed traces, traces in which each event has an associated real-valued time of occurrence. A TBA is a Buchi-automaton associated with a finite set of (real-valued) clocks. A clock can be set to zero simultaneously with any transition of the automaton. At any instant, the reading on a clock equals the time elapsed since the last time it was set. With each transition, there is an enabling condition that compares the current values of clocks with time constants. TBAs are not closed under complementation and it is undecidable whether the language of one automaton is a subset of the language of another. However, there exists a subclass represented by Deterministic Timed Muller Automata (DTMA) closed under all Boolean operations, and there is a decidable computation to check the subset relation for this class.

Hybrid automata [ACHH93] can be viewed as a generalization of timed automata, in which the behavior of variables is governed in each state by a set of differential equations. The reachability problem is undecidable even for very restricted classes of hybrid automata. However, there exist semi-decision procedures for verifying safety properties of piecewise-linear hybrid automata, in which all variables change at constant rates.

In both cases, explicit variables are introduced to reason about time bounds and changes. The extra time variables, however, will increase both the expressive power of the representation and the complexity of the verification.

Similar developments along this line include timed Statecharts, timed transition systems, hybrid Statecharts and phase transition systems [MMP91], etc.

State Transition Assertions (STA) developed by Gordon [Gor, Gor92] are variations of Hoare logic for real-time specification. A state transition assertion is a quadruple $\langle A, B, P, Q\rangle$ where $A, B$ are predicates on states, called state precondition and postcondition, respectively, $P, Q$ are predicates on state sequences, called input precondition and output postcondition, respectively. A machine $\mathcal{M}$ satisfies a state transition assertion $\langle A, B, P, Q\rangle$ as follows: if $\mathcal{M}$ is in a state satisfying $A$ and a sequence of inputs arrives that satisfies $P$, then a state satisfying $B$ will be reached and the sequence of intermediate states will satisfy $Q$. Some laws for combining STAs are analogous to rules of Hoare logic.

In contrast to state transition systems, where states and possible transitions are predefined, the situation calculus [MH69] defines states on the results of actions. Similar to most temporal logics, propositions and functions are interpreted over states (fluents in the situation calculus). Fluents at any state can be computed by frame axioms. The advantage of the situation calculus, namely, states with no structures, is also its disadvantage because of (1) the frame problem [MH69] and (2) the computation cost that may increase with time as the action list gets longer and longer.

### 12.2.2 Point time temporal logics

There are, in general, two kinds of point time temporal logic: linear time temporal logic and branching time temporal logic. A model of a linear time temporal logic is a trace, and a model of a branching time temporal logic is a tree.

Computation Tree Logic (CTL) is a typical modal branching time temporal logic [Eme90]. In CTL, temporal operators occur only in pairs consisting of $A$ (all paths) or $E$ (exists some path), followed by $F$ (eventually), $G$ (always), $U$ (until) or $X$ (next time). CTL has efficient model-checking algorithms, however, it loses some expressive power [Eme90]. CTL has been used for symbolic model checking of circuits [McM92].

In the rest of this section, we will focus on linear time temporal logics and their timed extensions. There are two kinds of linear time temporal logic: modal logic in which temporal operators are introduced, and the first order logic in which a special time variable is introduced.

PLTL is a basic form of modal linear time temporal logic. It has been shown that model checking for PLTL is linear in the size of the model [LP85]. Various timed extensions are based on PLTL. Again, there are two kinds of extension: real-time operators and time variables. The former is simpler and more elegant, but the latter can be more powerful. Temporal proof methodologies for both explicit and implicit time have been studied [HMP91a].

Extended Temporal Logic (ETL) [Wol83] is an extended linear (and discrete) time temporal logic, which is strictly more powerful than (discrete) PLTL and has the same expressive power as Buchi Automata. ETL defines temporal operators generated by right-linear grammars, so that (countable) properties such as $even(p)$ ($p$ is $true$ at even time points) can be specified, which, however, cannot be expressed in PLTL.

Metric Temporal Logic (MTL) [MMP91] introduces various types of real-time operator, such as $\Box_{<u}$ and $\Diamond_{\leq u}$ where $u$ is a nonnegative real number.

Real Time Temporal Logic (RTTL) [Ost89] is a first-order temporal logic, with one of the state variables representing time. For instance, $w_1 \wedge t = T \rightarrow \Diamond(w_2 \wedge t < T + 4)$ may be read as: "if $w_1$ is $true$ at time $T$ then $w_2$ must happen before the clock reads $T + 4$," where $T$ is a parameter (global variable). The problem with this specification language is that the unquantified global variables about time ($T$ in the above example) may lead to opacity [AH89].

Timed Propositional Temporal Logic (TPTL) [AH89] is the adoption of temporal operators as quantifiers over state variables; every modality binds a variable to the time(s) it refers to. For instance, "if $w_1$ is $true$ at time $T$ then $w_2$ must happen before the clock reads $T + 4$" can be represented as $\Box x.(w_1 \rightarrow \Diamond y.(w_2 \wedge y < x + 4))$. A tableau-based decision procedure was developed for TPTL. Introducing extra time variables increases the flexibility of expressing time constraints, and simultaneously, the complexity of verification.

The Temporal Logic of Actions (TLA) [Lam91] is a logic for specifying and reasoning about concurrent systems. Systems and their properties are represented in the same logic, so the assertion that a system meets its specification and the assertion that one system implements another are both expressed by logical implication. TLA introduces a concept called "action," which is any boolean-valued expression from variables, primed variables and values. An action represents a relation between old states and new states, where the unprimed variables refer to the old state and the primed variables refer to the new state. TLA imposes some constraints for representing actions such that action $\mathcal{A}$ can only appear in the form $\Box[\mathcal{A}]_f \equiv \Box(\mathcal{A} \vee (f' = f))$, where $f$ is a state tuple. [Lam91] shows that TLA is powerful enough for representing properties such as liveness and fairness, with a simple set of axioms and rules for the proof system. A real time version of TLA was proposed by introducing an explicit time variable *now* [Lam93].

Most temporal logics are defined for discrete time systems, i.e., with models as state sequences. It was suggested [BKP86] that linear temporal logic with the time structure of the (non negative) real numbers provides a more abstract logic than that of the natural numbers. Temporal Logic of Reals (TLR) is a logic defined on dense time. For each trace $v$ there exists a denumerable sequence $0 = t_0 < t_1 < t_2 \cdots$ with $t_n \to \infty$ such that $v(t)$ is uniform in TLR within each open interval $(t_i, t_{i+1})$. The difference between TLR and discrete time temporal logics is that there is no predetermined sampling rate. TLR would be best suited for asynchronous event control systems.

Besides modal linear temporal logics, there are first order temporal logics. McDermott [McD90] developed a first-order temporal logic, in which it is possible to name and prove things about facts, events, plans, and world histories. In particular, the logic provides the analysis of causality, continuous change in quantities, the persistence of facts and the relationship between tasks and actions. Shoham [Sho88, Sho87] generalized McDermott's temporal logic and defined a clean syntax and semantics. Finer distinctions of fact/event/process trichotomy are allowed under this framework.

### 12.2.3 Interval time temporal logics

Unlike point time temporal logics, formulas of Interval Temporal Logics (ITL) are defined on intervals of state sequences. One distinguished advantage of ITL is that it can represent lengths of intervals, and therefore it can represent time easily. ITL has been applied to multilevel reasoning about hardware properties [Mos85] such as delay and stability of digital circuits. ITL has also been used for the specification of real-time systems [Hal90]. There are properties that can be represent by ITL but not by LTL. For instance, $\Box(E \to R\mathrm{w}ithin\{time(\tau)\mathrm{w}henS\}$ is an ITL formula [Hal90] representing that whenever $E$ is *true*, $R$ will be *true* within an interval that $S$ holds for time $\tau$ in total.

The duration calculus [HZ91] is a kind of interval temporal logic defined on continuous time structures. The duration calculus uses the integral of a predicate to formalize critical duration constraints. For example, "a bad situation cannot happen more often than 5 percent of the time over any time interval" can be represented as $\Box(\int B \leq 0.05l)$ where $l$ indicates the length of the interval. This property is hard to specify in a simple form of linear temporal logic.

Besides modal interval temporal logics, there are first order interval temporal logics. Allen [All90] proposed a framework in which time is represented by intervals. The relationships between two time intervals are characterized (before, equal, meets, overlaps, during, starts, finishes) and the properties of facts (that hold in an interval), events (that occur over an interval) and processes (that are occurring over an interval) are examined by logic axioms. Various types of action can be represented in this logic.

### 12.2.4 Relationships with TLTL and timed $\forall$-automata

TLTL is a powerful and simple specification language for sequential and timed behaviors. Unlike most specification languages, it is based on abstract time and domain structures. For simplicity, TLTL introduces only two basic real-time operators $\mathcal{U}^\tau$ and $\mathcal{S}^\tau$, while other real-time operators can be derived from these basic operators. TLTL is powerful enough to represent properties such as "if $w_1$ is $true$ at time $T$ then $w_2$ must happen before the clock reads $T+4$." In fact, this property can be represented by FTLTL without real-time operators as $\forall T \Box(w_1 \wedge t = T \rightarrow \Diamond(w_2 \wedge t < T+4))$, or simply by PTLTL as $\Box(w_1 \rightarrow \Diamond^4 w_2)$. TLTL can be considered as a generalization of TLR. However, there is no axiomization for TLTL yet, since any axiomization is defined for a particular time structure. FTLTL is more expressive than TLA since terms of FTLTL can as well include $pre(x)$ and $x - \tau$ for any local variable $x$, and TLTL has no restriction on formulas with these variables.

Timed $\forall$-automata are generalizations of $\forall$-automata to represent timed or continuous behaviors. A local timing constraint in (discrete) timed $\forall$-automata can also be specified in TBA. However, global timing constraints cannot be specified within TBA, since it is not possible to stop a clock except by resetting it. On the other hand, there are properties of timed behaviors that can be specified by TBA but cannot be specified by timed $\forall$-automata. Some interval time properties that are hard to represent in TLTL, are easy to represent in timed $\forall$-automata. For example, $\Box(E \rightarrow R\,within\{time(\tau)\,when\,S\}$ can be specified in a timed $\forall$-automaton with a global time bound $\tau$. An example of this type of specification will be discussed in Appendix C.

# Part III

# Control Synthesis and Robotic Architecture

*Attain utmost emptiness.*
*Maintain profound tranquility.*
*All things are running concurrently,*
*cycle follows cycle.*

*Activity overcomes cold.*
*Tranquility overcomes heat.*
*Peace and quiet is the true path in the world.*
*— Tao Teh Ching, Lao Tzu*

*Attain utmost stability.*
*Maintain minimum energy.*
*All things are running concurrently,*
*cycle follows cycle.*

*Constraints overcome chaos.*
*Stability overcomes disturbance.*
*Peace and quiet is the true path in the world.*
*— Zhang Ying*

# Chapter 13

# Introduction

We have developed a semantic model for dynamic systems and two requirements specification languages for dynamic behaviors. We have also developed a formal method for verifying the behavior of a dynamic system against its requirements specification. Verification in general is hard. However, a good design methodology can result in a well-structured system, which, in turn, may simplify the verification greatly. In Part III, we present a framework of control synthesis with a simple principle. We consider a robotic system as a constraint-based dynamic system and the robot controller as a regulator that, together with the dynamics of the plant and the environment, solves the constraints on-line. We then propose a two-dimensional hierarchical structure for control systems.

In this chapter, we present an overview of Part III, Control Synthesis and Robotic Architecture. There are three major chapters in Part III. Chapter 14 studies constraint-based dynamic systems. Chapter 15 proposes a framework for control synthesis. Chapter 16 discusses structures of control systems.

## 13.1 Constraint-Based Dynamic Systems

We view constraint satisfaction as a dynamic process that approaches the solution set of the given constraints asymptotically. Generalizing, we view a constraint-based dynamic system as a dynamic system that approaches the solution set of the given constraints persistently.

We first introduce dynamic processes, stable equilibria and attractors. We then define Liapunov functions with respect to dynamic processes and stable states, and study the relationship of a Liapunov function and the stability of a dynamic process.

We consider a constraint solver as a constraint net whose behavior is a dynamic process that is asymptotically stable at the solution set of the given constraints.

We show that various discrete and continuous time constraint methods for solving discrete/continuous optimization and global consistency problems can be modeled in constraint nets and analyzed using Liapunov functions.

We consider constraint-based dynamic systems as a generalization of constraint solvers, whose behaviors can be specified by $\forall$-automata.

## 13.2 Control Synthesis

We define the problem of control synthesis as follows. Given a requirements specification and the models of the plant and the environment, produce a model of the controller that, together with the plant and the environment, satisfies the requirements specification.

Control synthesis in general is hard. However, we show that there is a systematic approach to control synthesis using constraint methods for constraint-based specification; typical constraint-based specification includes safety requirements, goal achievement and persistent properties.

We illustrate, by two examples, that various control algorithms, from simple linear control to complex nonlinear and adaptive control, can be synthesized and analyzed in this framework.

## 13.3 Robotic Architecture

Any complex system should have some kind of hierarchical structure. We consider here two kinds of hierarchy: composition hierarchy and interaction hierarchy. The interaction hierarchy can be further decomposed into a two-dimensional structure: abstraction hierarchy and arbitration hierarchy. The abstraction hierarchy characterizes the multiple levels of control strategy in a system; the arbitration hierarchy characterizes the priority of constraints to be satisfied within the same abstraction level.

## 13.4 Summary and Related Work

The major contribution of this part includes a unified framework for constraint satisfaction and a unified framework for control synthesis based on a simple principle — on-line constraint satisfaction or energy minimization. Hybrid control systems can be designed and analyzed in this framework.

# Chapter 14

# Constraint-Based Dynamic Systems

In this chapter, we start with the basic concepts of dynamic processes, equilibria and stability, then discuss two basic types of constraint solver, discrete state transitions and differential state integrations. Furthermore, we study some typical discrete and continuous time constraint methods for both global consistency and optimization. Finally, we introduce constraint-based dynamic systems.

## 14.1 Asymptotic Stability

In this section, we study properties of dynamic processes in metric space.

Given a metric space $\langle X, d\rangle$, we can define the distance between a point and a set of points as $d(x, X^*) = \inf_{x^* \in X^*}\{d(x, x^*)\}$. For $x^* \in X$ and $\epsilon > 0$, let $N^\epsilon(x^*)$ be the spherical $\epsilon$-neighborhood of $x^*$ and for $X^* \subset X$, let $N^\epsilon(X^*) = \bigcup_{x^* \in X^*} N^\epsilon(x^*)$ be the spherical $\epsilon$-neighborhood of $X^*$. A neighborhood of $X^*$ is *strict* iff it is a strict superset of $X^*$.

Let $\mathcal{T}$ be a time structure, $X$ be a metric space, and $v : \mathcal{T} \to X$ be a function from time to the metric space. We say $v$ *approaches a point* $x^* \in X$ iff $\lim_{t\to\infty} d(v(t), x^*) = 0$; $v$ *approaches a set* $X^* \subset X$ iff $\lim_{t\to\infty} d(v(t), X^*) = 0$.

**Definition 14.1.1** *A* dynamic process *is a mapping* $p : X \to X^{\mathcal{T}}$*, satisfying the following conditions:*

1. $p(x)(\mathbf{0}) = x, \forall x \in X$,
2. *$p$ is state-based, i.e.,* $\forall t$, $p(x)(t) = p(y)(t)$ *implies that* $\forall t' \geq t$, $p(x)(t') = p(y)(t')$.
3. *$p$ is time-invariant, i.e.,* $\{p(x) | x \in X\}$ *is a time-invariant behavior.*

Let $\phi_p(x) = \{p(x)(t)|t \in \mathcal{T}\}$ and $\phi_p(X^*) = \bigcup_{x \in X^*} \phi_p(x)$ for $X^* \subset X$. A point $x^* \in X$ is an *equilibrium* (or *fixpoint*) of a process $p$ iff $\forall t, p(x^*)(t) = x^*$, or $\phi_p(x^*) = \{x^*\}$. A set $X^* \subset X$ is an *equilibrium* of a process $p$ iff $\phi_p(X^*) = X^*$. An equilibrium $X^*$ is *stable* [MT75] iff $\forall \epsilon \exists \delta, \phi_p(N^\delta(X^*)) \subseteq N^\epsilon(X^*)$, i.e., $\phi_p$ is continuous at $X^*$.

A set $X^* \subset X$ is an *attractor* [San90] of a process $p$ iff there exists a strict neighborhood $N(X^*)$ such that $\forall x \in N(X^*)$, $p(x)$ approaches $X^*$. The largest neighborhood of $X^*$ satisfying this property is called the *attraction basin* of $X^*$. $X^*$ is an *attractor in the large* iff $\forall x \in X$, $p(x)$ approaches $X^*$, that is the attraction basin of $X^*$ is $X$. If $X^*$ is an attractor (in the large) and $X^*$ is a stable equilibrium, $X^*$ is called an *asymptotically* stable equilibrium (in the large).

**Proposition 14.1.1** *If $\{X_i\}_{i \in I}$ are ((asymptotically) stable) equilibria, then $\bigcup_I X_i$ is an ((asymptotically) stable) equilibrium.*

Let $\langle X, d \rangle$ be a metric space, $p : X \to X^{\mathcal{T}}$ be a dynamic process and $X^* \subset X$. A *Liapunov function* for $p$ and $X^*$ is a function $V : \Omega \to \mathcal{R}$, where $\Omega$ is a strict neighborhood of $X^*$, satisfying:

1. $V$ is continuous, i.e., $d(x, x') \to 0$ implies $|V(x) - V(x')| \to 0$.

2. $V$ has its unique minimum within $\Omega$ on $X^*$.

3. $\forall x \in \Omega, \forall t, V(p(x)(t)) \leq V(x)$.

The following two theorems are analogous to the theorems of sound and complete verification rules in Part II.

**Theorem 14.1.1** *$X^* \subset X$ is a stable equilibrium of a process $p$ iff there exists a Liapunov function $V$ for $p$ and $X^*$.*

**Theorem 14.1.2** *$X^* \subset X$ is an asymptotically stable equilibrium of a process $p$ iff there exists a Liapunov function $V : \Omega \to \mathcal{R}$ for $p$ and $X^*$, such that $\forall x \in \Omega, \lim_{t \to \infty} V(p(x)(t)) = V(X^*)$. Furthermore, if $\Omega = X$, $X^*$ is an asymptotically stable equilibrium in the large.*

## 14.2 Constraint Solvers

We view a *constraint* as a possibly implicit relation on a set of variables. The *constraint satisfaction problem* is defined as follows. Given a set of variables $V$ with the associated domains $\{D_v\}_{v \in V}$ and a set of constraints $\{C_j\}_{j \in J}$ each on a subset of the variables, i.e., $C_j \subseteq \times_{V_j} D_v$

where $V_j \subseteq V$, find an explicit relation tuple $x \in \times_V D_v$ that satisfies all the given constraints, i.e., for all $j \in J$, $x_{|V_j} \in C_j$ where $x_{|S}$ denotes the restriction of $x$ onto $S \subseteq V$. If $C = \{C_j\}_{j \in J}$ is a set of constraints, we use $sol(C)$ to denote the set of solutions, called *the solution set.*

A constraint solver for a constraint satisfaction problem is a closed parameterized net whose behavior is a dynamic process approaching the solution set of the constraints.

**Definition 14.2.1 (Constraint solver)** *A closed parameterized net $CS^V$ is a* constraint solver *for a constraint satisfaction problem $C$ on domain $X = \times_V D_v$ iff (1) the semantics of $CS^V$ for $V$ is a dynamic process $[\![CS^V]\!] : X \to X^{\mathcal{T}}$ and (2) $sol(C)$ is an asymptotically stable equilibrium of $[\![CS^V]\!]$. $CS^V$* solves $C$ globally *iff $sol(C)$ an asymptotically stable equilibrium of $[\![CS^V]\!]$ in the large.*

**Proposition 14.2.1** *If a constraint solver $CS^V$ solves a set of constraints $C$ on variables $V$ globally, every equilibrium of $[\![CS^V]\!]$ is a solution of $C$.*

As an application of the concept of robustness for parameterized nets, two constraint solvers $CS_1$ and $CS_2$ for the set of constraints $C$ can be compared as follows. $CS_1$ is *more robust* than $CS_2$ iff the attraction basin of $sol(C)$ in $CS_1$ is a superset of that in $CS_2$.

We discuss here two basic types of constraint solver: state transition systems for discrete methods and state integration systems for continuous methods.

Let $S$ be a set of states and $f : S \to S$ be a state transition function. $\langle S, f\rangle$ forms a *state transition system* $\langle S, \to\rangle$ with $s \to s'$ iff $s' = f(s)$. Such a state transition system can be represented by a closed parameterized net with a transliteration $f$ and a unit delay $\delta(s_0)$ where $s_0$ is the initial state parameter. The semantic of this net on the discrete time structure $\mathcal{N}$ is a dynamic process $p : S \to S^{\mathcal{N}}$ with $p(s_0)(n) = f^n(s_0)$. A state $s^* \in S$ is an equilibrium of $\langle S, f\rangle$ iff $s^* = f(s^*)$.

**Proposition 14.2.2** *If $V : \Omega \to \mathcal{R}$ is a Liapunov function for $\langle S, f\rangle$ and $S^* = \{s^* | s^* = f(s^*)\} \subset \Omega$, then $V(f(x)) \leq V(x), \forall x \in \Omega$. In addition, if $f$ is continuous and $V(f(x)) < V(x), \forall x \notin S^*$, $S^*$ is an asymptotically stable equilibrium.*

For continuous time structures and domains, integration is used to replace the unit delay. A *state integration system* is a differential equation $\dot{s} = f(s)$ that can be represented by a closed parameterized net with a transliteration $f$ and an integration $\int(s_0)$ where $s_0$ is the initial state parameter (Figure 4.2). The semantic of this net on the continuous time structure $\mathcal{R}^+$ is a dynamic process $p : S \to S^{\mathcal{R}^+}$ with $p(s_0)$ as the solution of $\dot{s} = f(s)$ and $s(0) = s_0$. A state $s^* \in S$ is an equilibrium of $\dot{s} = f(s)$ iff $f(s^*) = 0$.

**Proposition 14.2.3** *A set $S^* = \{s^* | f(s^*) = 0\} \subset \Omega$ is an asymptotically stable equilibrium of a state integration system if $f$ is continuous at $S^*$ and $S^*$ is the unique minimum of $-\int f(s)ds$ in $\Omega$. If $\Omega = S$, $S^*$ is an asymptotically stable equilibrium in the large.*

## 14.3 Constraint Methods

Various *constraint methods* fit into our framework of constraint satisfaction. In this section, we examine some typical constraint methods and their dynamic properties. We discuss two types of constraint satisfaction problem, namely, global consistency and optimization, for linear, convex and nonlinear relations in $n$-dimensional Euclidean space $\langle \mathcal{R}^n, d_n \rangle$, where $d_n(x,y) = |x - y| = \sqrt{\Sigma_{i=1}^n (x_i - y_i)^2}$. Constraint methods for finite domain constraint satisfaction have been presented in [ZM93a, ZM93b].

The problem of *global consistency* is to find a solution tuple that satisfies all the given constraints. The problem of *unconstrained optimization* is to minimize a function $\mathcal{E} : \mathcal{R}^n \to \mathcal{R}$. Global consistency corresponds to solving hard constraints and unconstrained optimization corresponds to solving soft constraints. A problem of the first kind can be translated into one of the second by introducing an energy function representing the degree of global consistency. For example, given a set of equations $g_i(x) = 0, i = 1 \cdots n$, let $\mathcal{E}_g(x) = \Sigma_{i=1}^k w_i g_i^2(x)$ where $w_i > 0$ and $\Sigma_{i=1}^k w_i = 1$. If a constraint solver $CS$ solves $\min \mathcal{E}_g(x)$, $CS$ solves $g(x) = 0$. Inequality constraints can be transformed into equality constraints. There are two approaches. Let $g_i(x) \leq 0$ be an inequality constraint: the equivalent equality constraint is (i) $\max(0, g_i(x)) = 0$ or (ii) $g_i(x) + z^2 = 0$ where $z$ is introduced as an extra variable.

*Constrained optimization* is a problem of solving (soft) constraints subject to the satisfaction of a set of hard constraints, or solving a constraint satisfaction problem within a subspace characterized by a set of hard constraints.

There are two types of constraint method, discrete relaxation, which can be implemented as state transition systems, and differential optimization, which can be implemented as state integration systems. In the rest of this section, we demonstrate the use of both types of constraint method.

### 14.3.1 Discrete methods

We discuss here two typical discrete constraint methods, the projection method for global consistency, and Newton's method for unconstrained optimization.

### Projection method

The projection method [GPR67] can be used for solving convex constraints. A function $f : \mathcal{R}^n \to \mathcal{R}$ is *convex* iff for any $\lambda \in (0,1)$, $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$; it is *strictly convex* iff the inequality is strict. A strictly convex function has a unique minimal point. Linear functions are convex, but not strictly convex. A quadratic function $x^T M x + c^T x$ is convex if $M$ is semi-positive definite; it is strictly convex if $M$ is positive definite. A set $R \subseteq \mathcal{R}^n$ is *convex* iff for any $\lambda \in (0,1), x, y \in R$ implies $\lambda x + (1-\lambda)y \in R$. If $g$ is a convex function, $\{x | g(x) \leq 0\}$ is a convex set.

A *projection* of a point $x$ to a set $R$ in a metric space $\langle X, d\rangle$ is a point $P_R(x) \in R$, such that $d(x, P_R(x)) = d(x, R)$. Projections in the $n$-dimensional Euclidean space $\langle \mathcal{R}^n, d_n\rangle$ share the following properties.

**Proposition 14.3.1** [GPR67] *Let $R \subset \mathcal{R}^n$ be closed and convex. The projection $P_R(x)$ of $x$ to $R$ exists and is unique for every $x$, and $(x - P_R(x))^T(y - P_R(x)) \leq 0$ for any $y \in R$.*

Suppose we are given a system of convex and closed sets, $\{X_i\}_{i\in I}$, each representing a constraint. The problem is to solve $\{X_i\}_{i\in I}$, or to find $\cap_I X_i$. Let $P(x) = P_{X_l}(x)$ be a projection of $x$ to a least satisfied set $X_l$, i.e., $d(x, X_l) = \max_I d(x, X_i)$. The *projection method* [GPR67] for this problem defines a state transition system $\langle \mathcal{R}^n, f\rangle$ where $f(x) = x + \lambda(P(x) - x)$ for $0 < \lambda < 2$.

Let PM be a constraint net representing the projection method. The following theorem is derived from [GPR67].

**Theorem 14.3.1** *PM solves $\{X_i\}_{i\in I}$ globally if all the $X_i$'s are convex.*

The projection method can be used to solve a set of inequality constraints, i.e., $X_i = \{x | g_i(x) \leq 0\}$, where each $g_i$ is a convex function. Linear functions are convex. Therefore, the projection method can be applied to a set of linear inequalities $Ax \leq b$, where $x = \langle x_1, \ldots, x_n\rangle \in \mathcal{R}^n$. Let $A_i$ be the $i$th row of $A$. The projection of a point $x$ to a half space $A_i x - b_i \leq 0$ is defined as

$$P_i(x) = \begin{cases} x & \text{if } A_i x - b_i \leq 0 \\ x - cA_i^T & \text{otherwise} \end{cases}$$

where $c = (A_i x - b_i)/|A_i^T|^2$. This reduces to the method described in [Agm54]. Without any modification, this method can be also applied to a set of linear equalities, by simply replacing each linear equality $g_i(x) = 0$ with two linear inequalities: $g_i(x) \leq 0$ and $-g_i(x) \leq 0$.

There are various ways to modify this method for faster convergence. For instance, a simultaneous projection method is given in [CE82], in which $f(x) = x + \lambda\Sigma_{j\in J}w_j(P_j(x) - x)$ where $J \subseteq I$ is an index set of violated constraints, $w_j > 0$ and $\Sigma_{j\in J}w_j = 1$. A similar method is given in [YM] in which $f(x) = x + \lambda(P_S(x) - x)$ where $S = \{x|\Sigma_{j\in J}w_j g_j(x) \leq 0\}$, with the same assumption about $J$ and $w_j$. Furthermore, for a large set of inequalities, the problem can be decomposed into a set of $K$ subproblems with $f_k$ corresponding to the transition function of the $k$th subproblem. The whole problem can be solved by combining the results of $\{f_1, \ldots, f_K\}$.

**Newton's method**

Newton's method [San90] minimizes a second-order approximation of the given function, at each iterative step. Let $\Delta\mathcal{E} = \frac{\partial\mathcal{E}}{\partial x}$ and $J$ be the Jacobian of $\Delta\mathcal{E}$. At each step with current point $x^{(k)}$, Newton's method minimizes the function:

$$\mathcal{E}_a(x) = \mathcal{E}(x^{(k)}) + \Delta\mathcal{E}^T(x^{(k)})(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T J(x^{(k)})(x - x^{(k)}).$$

Let $\frac{\partial\mathcal{E}_a}{\partial x} = 0$, we have:

$$\Delta\mathcal{E}(x^{(k)}) + J(x^{(k)})(x - x^{(k)}) = 0.$$

The solution of the above equation becomes the next point, i.e.,

$$x^{(k+1)} = x^{(k)} - J^{-1}(x^{(k)})\Delta\mathcal{E}.$$

*Newton's method* defines a state transition system $\langle\mathcal{R}^n, f\rangle$ where $f(x) = x - J^{-1}(x)\Delta\mathcal{E}(x)$.

Let NM be a constraint net representing Newton's method. The following theorem specifies conditions under which NM solves the problem of local minimization of a function $\mathcal{E}$.

**Theorem 14.3.2** *Let $X^* \in \mathcal{R}^n$ be the set of local minima of $\mathcal{E}$. NM solves the problem if $|J(x^*)| \neq 0$, $\forall x^* \in X^*$. i.e., $\mathcal{E}$ is strictly convex at each local minimal point. NM solves the problem globally if, in addition, $\mathcal{E}$ is convex.*

Here we assume that the Jacobian and its inverse are obtained off-line. Newton's method can also be used to solve a nonlinear equation $g(x) = 0$ by replacing $\Delta\mathcal{E}$ with $g$.

For example, consider Newton's method for solving $x^2 = 2$. Newton's method for solving $g(x) = 0$ can be represented by a constraint net with domain equation: $x' = x - \frac{g(x)}{g'(x)}$. In our example, $g(x) = x^2 - 2$, $x' = x - \frac{x^2-2}{2x} = \frac{x}{2} + \frac{1}{x}$. NM solves $x^2 = 2$ since $g'(x^*) = 2x^* \neq 0$ for both $x^* = \sqrt{2}$ and $x^* = -\sqrt{2}$. The attraction basin of $\sqrt{2}$ is $\{x|x > 0\}$ and the attraction basin of $-\sqrt{2}$ is $\{x|x < 0\}$.

### 14.3.2 Continuous methods

We discuss here some typical continuous constraint methods: the gradient method for unconstrained optimization, the penalty method and the Lagrange multiplier method for constrained optimization.

#### Gradient method

The gradient method [Pla89] is based on the gradient descent algorithm, where state variables slide downhill in the direction opposed to the gradient. Formally, if the function to be minimized is $\mathcal{E}(x)$ where $x = \langle x_1, \cdots, x_n \rangle$, then at any point, the vector that points in the direction of maximum increase of $\mathcal{E}$ is the gradient of $\mathcal{E}$. Therefore, the following gradient descent equations model the *gradient method* :

$$\dot{x}_i = -k_i \frac{\partial \mathcal{E}}{\partial x_i}, \quad k_i > 0. \tag{14.1}$$

Let $\mathcal{E} : \mathcal{R}^n \rightarrow \mathcal{R}$ be a function. Let GM be a constraint net representing the gradient descent equation (Equation 14.1). The following theorem specifies conditions under which GM solves the problem of local minimization of $\mathcal{E}$.

**Theorem 14.3.3** *Let $X^*$ be the set of local minima of $\mathcal{E}$. GM solves the problem if $\frac{\partial \mathcal{E}}{\partial x}$ is continuous at $X^*$. GM solves the problem globally if, in addition, $\mathcal{E}$ is convex.*

Consider again the example of solving $x^2 = 2$. Let $\mathcal{E}(x) = \frac{1}{4}(x^2 - 2)^2$. Let the constraint solver GM be $\dot{x} = -\frac{\partial \mathcal{E}}{\partial x} = -x(x^2 - 2)$. GM solves $x^2 = 2$ since $-x(x^2 - 2)$ is continuous. The attraction basin of $\sqrt{2}$ is $\{x | x > 0\}$ and attraction basin of $-\sqrt{2}$ is $\{x | x < 0\}$.

#### Penalty and Lagrange multiplier methods

The prototypical constrained optimization problem can be stated as [Pla89]: locally minimize $f(x)$, subject to $g(x) = 0$, where $g(x) = 0$ is a set of equations describing a manifold of the state space. There are various methods for solving the constrained optimization problem. Here we focus on methods derived from the gradient method. During constrained optimization, the state $x$ should be attracted to the manifold $g(x) = 0$ and slide along the manifold until it reaches the locally smallest value of $f(x)$ on $g(x) = 0$.

Different methods arise from the design of the energy function $\mathcal{E}$ for minimizing $f(x)$ under constraints $g_k(x) = 0$ for $k = 0 \ldots m$. Let $\mathcal{E}_c$ be the energy function generated from the constraints, i.e., $\mathcal{E}(x) = f(x) + \mathcal{E}_c(x)$.

- *Penalty Methods*: The penalty method constructs an energy term that penalizes violations of the constraints, i.e., $\mathcal{E}_c(x) = \Sigma_{k=0}^m c_k g_k^2(x)$.
- *Lagrange Multipliers*: The Lagrange multiplier method introduces a Lagrange multiplier $\lambda$ for each constraint and $\lambda$ varies as long as its constraint is not satisfied, i.e., $\mathcal{E}_c(x) = \Sigma_{k=0}^m \lambda_k g_k(x)$. In addition, there is a set of differential equations for $\lambda$, i.e., $\dot{\lambda}_k = g_k(x)$.

The advantage of the penalty method is its simplicity; however, the constrained optimization problem may not be solved with finite $c_i$. The advantage of the Lagrange multiplier method is its ability to satisfy the hard constraints.

Let LM be a constraint net representing the Lagrange multiplier method. The following theorem specifies a condition under which LM solves the constrained optimization problem globally.

**Theorem 14.3.4** *Let $A$ be a matrix where $A_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} + \Sigma_{k=0}^m \lambda_k \frac{\partial^2 g_k}{\partial x_i \partial x_j}$. If $A$ is positive definite, LM solves the constrained optimization problem* $\min f(x)$ *subject to* $g_k(x) = 0$ *globally.*

Consider a simple example. Given a function $f(x, y) = x^2 + y^2$ to be minimized, subject to constraint $x + y - 1 = 0$, it is easy to check that the solution to this problem is $(0.5, 0.5)$. The constrained optimization based on the penalty method proceeds as follows: the energy function is $\mathcal{E}(x, y) = x^2 + y^2 + c(x + y - 1)^2$ where $c$ is a constant. Using the gradient method, let

$$\frac{dx}{dt} = -0.5\frac{\partial \mathcal{E}}{\partial x} = -(x + c(x + y - 1)),$$

$$\frac{dy}{dt} = -0.5\frac{\partial \mathcal{E}}{\partial y} = -(y + c(x + y - 1)).$$

The process is asymptotically stable at $(\frac{c}{2c+1}, \frac{c}{2c+1})$. When $c \to \infty$, the state $(x, y)$ approaches $(0.5, 0.5)$. The constraint optimization based on the Lagrange multipler method proceeds as follows: the energy function is $\mathcal{E}(x, y) = x^2 + y^2 + \lambda(x + y - 1)$. Using the gradient method, let

$$\frac{dx}{dt} = -\frac{\partial \mathcal{E}}{\partial x} = -(2x + \lambda),$$

$$\frac{dy}{dt} = -\frac{\partial \mathcal{E}}{\partial y} = -(2y + \lambda).$$

In addition

$$\frac{d\lambda}{dt} = (x + y - 1).$$

The process is asymptotically stable at $(0.5, 0.5)$ in the large.

## 14.4 Summary

We have presented here a framework for constraint satisfaction. Figure 14.1 illustrates the overall approach. First, we view constraints as relations and constraint satisfaction as a dynamic

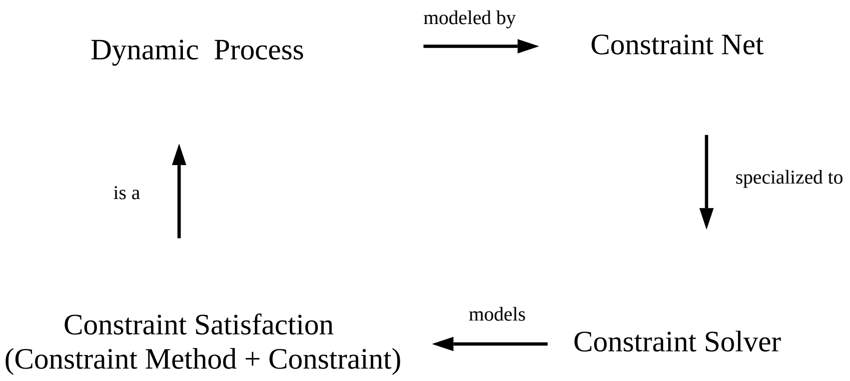


Figure 14.1: A framework for constraint satisfaction

process of approaching the solution set of the constraints. Then, we explore the relationship between constraint satisfaction and constraint nets through constraint solvers.

Within this framework, *constraint programming* is seen as the creation of a constraint solver that solves the set of constraints. A constraint solver "solves" a set of constraints in the following sense (Figure 14.2). Given a constraint satisfaction problem $C$, and a discrete or continuous (time) constraint method, a constraint solver $CS$ is generated. Starting from any initial state in the attraction basin of $sol(C)$, $CS$ will approach $sol(C)$ asymptotically. In this framework, constraint programming is off-line and constraint satisfaction is on-line.

We have also studied various continuous and discrete time constraint methods, which can be realized by state integration systems and state transition systems, respectively.

This framework for constraint satisfaction has two advantages. First, the definition of constraint solvers relaxes the condition of solving constraints from finite computation to asymptotic stability. For example, many relaxation methods with the local convergence property are in fact "solvers" under this definition and many problems become "semi-computable" in this sense. This concept is very useful in practice and can be used for generalizing Turing computability from discrete domains to continuous domains. Second, dynamic constraints can be solved in this framework as well. This characteristic will be important later in control synthesis.

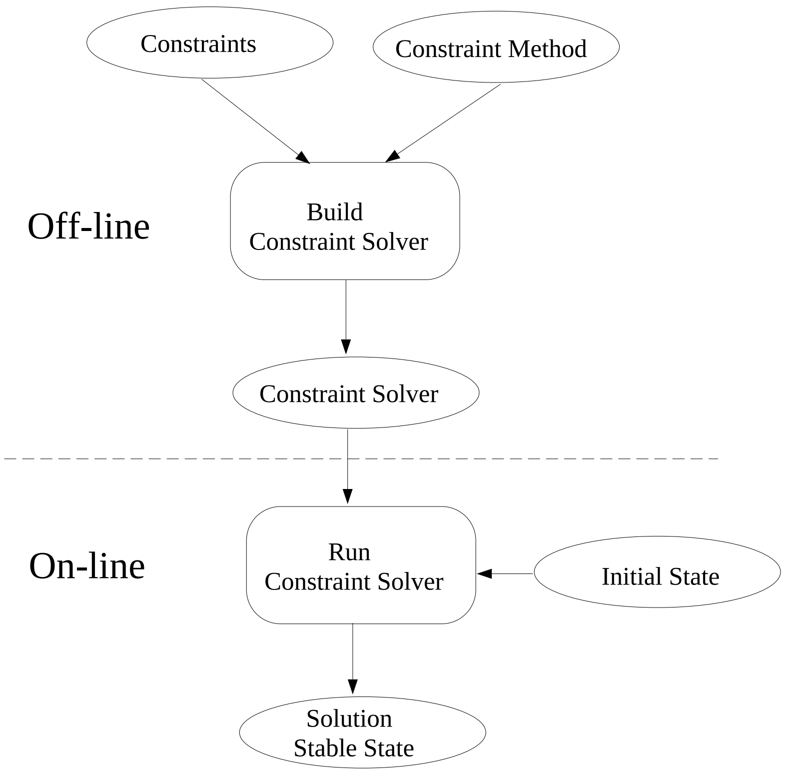


Figure 14.2: Constraint solvers and constraint satisfaction

## 14.5 Constraint-Based Dynamic Systems

Given a set of constraints $C$ on variables $V$, let $C^\epsilon$ denote the assertion that is true on the $\epsilon$-neighborhood of its solution set $N^\epsilon(sol(C)) \subset \times_V D_v$. Let $\mathcal{A}(C^\epsilon; \Box)$ stand for the $\forall$-automaton in Figure 14.3(a).

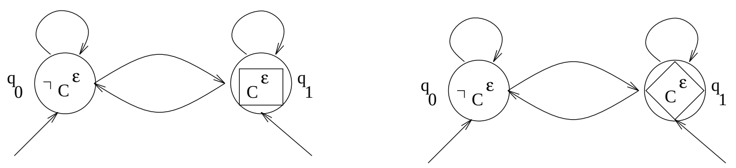


Figure 14.3: Specification for (a) Constraint solver (b) Constraint-based dynamic system

**Proposition 14.5.1** *A constraint solver $CS^V$ solves $C$ iff there exists an initial condition $\Theta \supset sol(C)$ such that $\forall \epsilon > 0$, $[\![CS^V(\Theta)]\!] \models \mathcal{A}(C^\epsilon; \Box)$. $CS$ solves $C$ globally when $\Theta = \times_V D_v$.*

For example, let $C^\epsilon$ be $|x - \sqrt{2}| < \epsilon$ or $|x + \sqrt{2}| < \epsilon$. In order to prove that Newton's method for solving $x^2 = 2$ satisfies $\mathcal{A}(C^\epsilon; \Box)$, we do the following. Let $\Theta$ be $|x| > 0$.

(I) Associate with automaton-state $q_0$ and $q_1$ state propositions $\Theta \wedge \neg C^\epsilon$ and $\Theta \wedge C^\epsilon$, respectively. It is easy to check that the following conditions are satisfied.

- *Initiality*: $q_0 : \Theta \wedge \neg C^\epsilon \rightarrow \Theta \wedge \neg C^\epsilon$ and $q_1 : \Theta \wedge C^\epsilon \rightarrow \Theta \wedge C^\epsilon$.
- *Consecution*: Let $f_s = \lambda x.(\frac{x}{2} + \frac{1}{x})$.

$$q_0, q_0 \ : \ \{\Theta \wedge \neg C^\epsilon\} x' = f_s(x) \{\neg C^\epsilon \rightarrow \Theta \wedge \neg C^\epsilon\}.$$
$$q_0, q_1 \ : \ \{\Theta \wedge \neg C^\epsilon\} x' = f_s(x) \{C^\epsilon \rightarrow \Theta \wedge C^\epsilon\}.$$
$$q_1, q_0 \ : \ \{\Theta \wedge C^\epsilon\} x' = f_s(x) \{\neg C^\epsilon \rightarrow \Theta \wedge \neg C^\epsilon\}.$$
$$q_1, q_1 \ : \ \{\Theta \wedge C^\epsilon\} x' = f_s(x) \{C^\epsilon \rightarrow \Theta \wedge C^\epsilon\}.$$

Therefore, $\Theta \wedge \neg C^\epsilon$ and $\Theta \wedge C^\epsilon$ are invariants for $q_0$ and $q_1$, respectively.

(L) Associate with automaton-state $q_0$ and $q_1$ a partial function $\rho$:

$$\rho = \lambda x. \begin{cases} x^2 - 2 & \text{if } |x| \geq \sqrt{2} \\ 1 + \rho(\frac{x}{2} + \frac{1}{x}) & \text{otherwise.} \end{cases}$$

It is easy to check that they satisfy the definedness and non-increase conditions. Furthermore, since $\Theta \wedge \neg C^\epsilon$ and $x' = f_s(x)$ imply that $\rho(x') - \rho(x) \leq -\min(1, \epsilon_0)$ where $\epsilon_0 = \rho(\sqrt{2} + \epsilon) - \rho(f_s(\sqrt{2} + \epsilon))$, the decrease condition is satisfied. Therefore, it is a Liapunov function.

According to the verification rules, Newton's method for solving $x^2 = 2$ satisfies $\mathcal{A}(C^\epsilon; \Box)$. We should notice the importance of open specification for the asymptotic goal achievement property; Newton's method for solving $x^2 = 2$ does not satisfy $\mathcal{A}(sol(C); \Box)$.

For another example, the gradient method for solving $x^2 = 2$ satisfies the $\mathcal{A}(C^\epsilon; \Box)$ for any $\epsilon > 0$ as well. To see this, let $\Theta$ be $|x| > 0$. Associate with automaton-state $q_0$ and $q_1$ state propositions $\Theta \wedge \neg C^\epsilon$ and $\Theta \wedge C^\epsilon$, which are invariants for $q_0$ and $q_1$, respectively. Associate with automaton-state $q_0$ and $q_1$ the function $\mathcal{E}(x) = \frac{1}{4}(x^2 - 2)^2$. For any initial state $x_0 \in \Theta$, $\dot{\mathcal{E}}(x) = -x^2(x^2 - 2)^2 \leq -min(2, x_0^2)\epsilon^4$ whenever $x \in \Theta \wedge \neg C^\epsilon$; $\mathcal{E}$ is a Liapunov function.

However, when constraints are dynamic, approaching the solution set asymptotically is still too stringent for a constraint satisfaction problem with disturbance and uncertainty in its data variables over time. If we consider the solution set of a set of constraints as the "goal" for the system to achieve, a relaxed property for a constraint solver is to make the system approach the goal persistently. In other words, if the system diverges from the goal by some disturbance, the system should always be able to be regulated back to its goal. We call a system $CB$ *constraint-based* with respect to a set of constraints $C$, iff there exists an initial condition $\Theta \supset sol(C)$ such that $\forall \epsilon > 0, [\![CB(\Theta)]\!] \models \mathcal{A}(C^\epsilon; \Diamond)$ where $\mathcal{A}(C^\epsilon; \Diamond)$ stands for the $\forall$-automaton in Figure 14.3(b). In other words, a dynamic system is constraint-based iff it approaches the solution set of the constraints persistently. Since $\Box\Diamond G \rightarrow \Diamond\Box G$, a constraint solver is a constraint-based system as well.

We may relax this condition further and define constraint-based systems with errors. We call a system $CB$ *constraint-based* w.r.t. a set of constraints $C$ with *error* $\delta$, iff $\forall \epsilon > \delta, [\![CB(\Theta)]\!] \models \mathcal{A}(C^\epsilon; \Diamond)$; $\delta$ is called the *steady-state error* of the system. Normally, steady-state errors are caused by uncertainty and disturbance of the data variables.

If $\mathcal{A}(C^\epsilon; \Box)$ is considered as an open specification of a constraint-based *computation* for a closed system, $\mathcal{A}(C^\epsilon; \Diamond)$ can been seen as an *open specification* of a constraint-based *control* for an open or embedded system.

# Chapter 15

# Control Synthesis

Given a constraint-based specification for a controller, the design of the controller is the synthesis of an embedded constraint solver that, together with the dynamics of the plant, solves constraints on-line. Various constraint methods can be applied to control synthesis under this framework. More importantly, most constraint methods are associated with some type of Liapunov function, which can be directly used by the verification method. In this chapter, we start with general issues of control synthesis and then focus on constraint-based control design and analysis. Finally we illustrate this approach via examples.

## 15.1 Control Synthesis: General Issues

A robotic system, in general, consists of a plant, a controller and an environment (Figure 1.1). The robotic behavior is the set of observable robot/environment traces of the system. A requirements specification is a subset of all the possible robot/environment traces. The problem of *control synthesis* can be formalized as follows: Given a requirements specification $\mathcal{R}$, the model of the plant $PLANT$ and the model of the environment $ENVIRONMENT$, synthesize a model of the controller $CONTROLLER$, such that

$$[\![X = PLANT(U,Y),\ U = CONTROLLER(X,Y),\ Y = ENVIRONMENT(X)]\!] \models \mathcal{R}.$$

Both planning and control problems can be seen as instances of this formalization.

The planning problem is a special case of the control synthesis problem, with the restriction that the controller is an 0-ary transduction (a trace), instead of a transduction in general, and the requirements specification only imposes conditions on the "final state" of the system. If the integration of the plant and the environment is a finite state automaton, with the control output as the input, planning is the generation a path in the state transition graph, given the initial state. The complexity of this problem is linear in the size of the state transition graph.

This simple form of the planning problem can be considered as an open-loop control synthesis problem. It has been shown in control system theory (and in practice) that open-loop control is not robust. A direct generalization is then synthesizing the controller to behave as a transliteration, i.e., a reactive (universal) plan [Sch87]. Given that $\mathcal{S}$ is the space of the robot/environment state tuples and $\mathcal{U}$ is the set of possible control values, the number of possible reactive controllers will be $|\mathcal{U}|^{|\mathcal{S}|}$.

In general, requirements specification may impose other forms of constraints on traces. For example, safety and persistence are typical requirements, other than reachability, for dynamic systems. Some aggregation evaluation of the system, such as the minimum overall energy, is also an important kind of specification. When uncertainty is concerned, minimum overall expected cost is normally imposed as a constraint [Qi94].

Approaching a final goal and minimizing a global function over time (for example, energy) can both be considered as constraints over traces; the former is a typical planning problem and the latter is a typical control problem. Planning and control have been studied as different problems over the years. The *planning problem* [DW91] is defined as using a model to formulate sequences of actions (or more generally, to composite descriptions of actions over time) to achieve a certain goal. The *control problem* [DW91] is considered as finding a policy to achieve a goal or minimizing a functional. Planning is normally restricted to symbolic domains in discrete time; while control is often for numerical domains, particularly $n$-dimensional Euclidean spaces, in either discrete or continuous time. The result of a planning problem (traditionally) is a trace (sequence) of inputs to a plant for approaching a final goal; the solution to a control problem (closed-loop control) is a transduction from the sensor traces to the command traces for minimizing a required functional, such as time, energy, cost for approaching a goal. Search algorithms and theorem proving are basic techniques for planning; calculus of variations and optimization are basic techniques for control. In our framework of control synthesis, planning and control can be studied together, and techniques developed for one problem may be used for the other.

Control synthesis in general, like verification, is hard. Furthermore, there does not exist a uniform algorithm for different control synthesis problems. In the rest of this chapter, we focus on a systematic approach to designing and analyzing constraint-based control systems.

## 15.2 Constraint-Based Control

We restrict requirements specification to constraint-based specification. Most robotic systems are constraint-based, since physical limitations, environmental restrictions and task requirements can be specified as constraints. We have developed a framework of viewing constraint satisfaction as a dynamic process. An important consequence of this framework is to be able to design control systems as *embedded constraint solvers*. Such an embedded constraint solver is an open system with inputs as observable traces of the plant and the environment. The embedded constraint solver together with the rest of the robotic system satisfies the desired constraint-based specification (Figure 15.1).

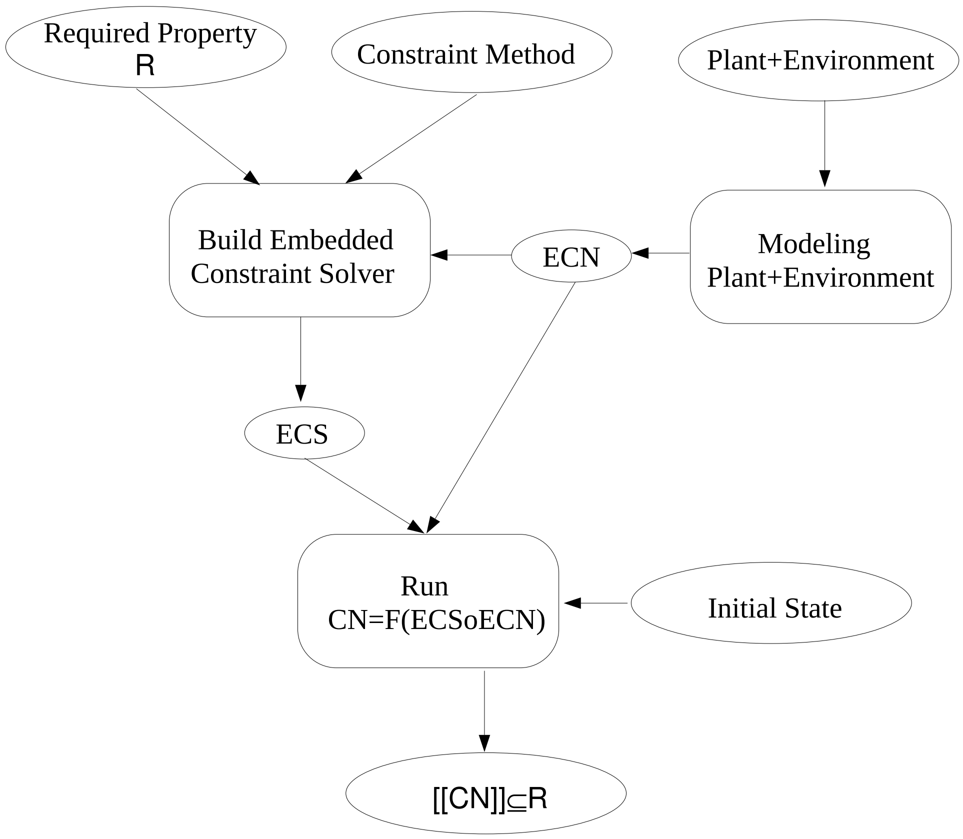


Figure 15.1: Embedded constraint solvers

Let $C$ be a set of constraints and $C^\epsilon$ be an $\epsilon$-neighborhood of $sol(C)$. Typical types of constraint-based specification are:

- *safety requirement*: $\Box C^{\epsilon}$;
- *goal achievement*: $\Diamond\Box C^{\epsilon}$;
- *persistence*: $\Box\Diamond C^{\epsilon}$.

The safety requirement is the strongest and the persistence is the weakest, since $\Box C^{\epsilon} \rightarrow \Diamond\Box C^{\epsilon}$ and $\Diamond\Box C^{\epsilon} \rightarrow \Box\Diamond C^{\epsilon}$.

Embedded constraint solvers can be either discrete or continuous according to the constraint methods. Continuous solvers, based on energy functions, generalize potential functions. Discrete solvers, based on relaxation methods in numerical computation, are more flexible in many applications.

The design of an energy function depends on the type of constraint. For goal achievement or persistence constraints, the energy function defines the degree of satisfaction of the constraints; for safety constraints, the energy function defines the degree of satisfaction of the constraints within $C^{\epsilon}$ and infinity outside of $C^{\epsilon}$. For example, given a requirement specification $\Box\Diamond C^{\epsilon}$ with $C$ defined as $f(x) = 0$, an energy function for this specification can be $\frac{1}{2}f^2(x)$. If $\Box(f(x) > 0)$ is required, an energy function can be $\max(-\ln\frac{f(x)}{\epsilon}, 0)$, i.e., if $f(x) \geq \epsilon$, then $\mathcal{E}(x) = 0$, if $0 < f(x) < \epsilon$, then $\mathcal{E} > 0$, and if $f(x) \rightarrow 0$, then $\mathcal{E}(x) \rightarrow \infty$. Using these types of energy function, we have designed controllers for a two-link robot arm tracking targets (persistence) and/or avoiding obstacles (safety); details are presented in Appendix C.

## 15.3 Examples

Various existing controllers, from simple linear control to complex nonlinear adaptive control or potential field methods, can be derived and analyzed in this framework. We analyze two simple examples here to illustrate the approach. The first is on the design and analysis of linear controllers, the second is on the design and analysis of a nonlinear controller for a car-like robot.

### 15.3.1 Linear control

Linear controllers are most widely used in real systems. Even though there are many advanced control strategies in theory, linear controllers are still the most robust and reliable ones.

A linear *proportional and derivative* (PD) controller has the form $u = k_p e + k_d \dot{e}$ where $u$ is the control signal, $e = x_d - x$ is the current error between the desired position $x_d$ and the actual position $x$, $k_p$ is a proportional gain and $k_d$ is a derivative gain. A desired property for a PD controller can be $\Diamond\Box(|e| < \epsilon)$. However, in many cases, we would also like to trade position errors for low oscillation or frequency. A more appropriate property for a PD controller should be $\Diamond\Box(e^2 + \lambda\dot{e}^2 < \epsilon)$ where $\lambda > 0$ denotes a trade-off between position and velocity errors. If $\lambda \to 0$, only position errors are taken into account.

We can synthesize a PD controller using an energy function $\mathcal{E} = \frac{1}{2}(e^2 + \lambda\dot{e}^2)$. The controller, together with the dynamics of the plant, is to make $\mathcal{E}$ go to its minimum. Let $\dot{\mathcal{E}} = e\dot{e} + \lambda\dot{e}\ddot{e} = \dot{e}(e + \lambda\ddot{e})$. If we let $e + \lambda\ddot{e} = -k\dot{e}$ for $k > 0$, we have $\dot{\mathcal{E}} \leq 0$, a desired property for the controller. Therefore, we want $-\ddot{e} = \frac{1}{\lambda}(e + k\dot{e})$. In most cases, $\ddot{x}_d = 0$, so $\ddot{x} = \frac{1}{\lambda}(e + k\dot{e})$. If the dynamics of the plant is $u = \ddot{x}$, let $u = \frac{1}{\lambda}(e + k\dot{e})$, which is a PD controller with $k_p = \frac{1}{\lambda}$ and $k_d = \frac{k}{\lambda}$. This design tells us that if $\lambda \to 0$, then $k_p, k_d \to \infty$, and there will be possibly high oscillation since the constraint on $\dot{e}$ is neglected. A compromise between the position error and the oscillation frequency should be made for any application.

Furthermore, if the dynamics of the plant is $u = m\ddot{x}$, the PD controller $u = \frac{1}{\lambda}(e + k\dot{e})$ will make $\mathcal{E} = \frac{1}{2}(e^2 + m\lambda\dot{e}^2)$ go to its minimum. If the dynamics of the plant is not fully known, we can still get a good estimation of the control parameters. Since $\dot{\mathcal{E}} = \dot{e}(e + \lambda\ddot{e}) = (\lambda u - e)(e + \lambda\ddot{e})/k$, if $\lambda|u| \leq |e|$ and $\lambda|\ddot{e}| \leq |e|$, we have $\dot{\mathcal{E}} \leq 0$. Therefore, let $\lambda = \frac{|e|_{\min}}{\max(|u|_{\max}, |\ddot{e}|_{\max})}$ where $|e|_{\min}$ is the steady state error, and $|u|_{\max}$ and $|\ddot{e}|_{\max}$ can be estimated even when the dynamics of the plant is unknown. If $u$ can be estimated on-line, $\lambda$ can be adapted over time, and better performance can be achieved.

We can design and analyze nontrivial control strategies using the same simple principle — on-line constraint satisfaction or energy minimization.

### 15.3.2 Nonlinear control

Linear PD controllers are simple and easy to analyze. However, they may not fit on to systems with complex nonlinear dynamics. Consider a tracking system for the car-like robot. Let $v$ be the velocity of the car and $\alpha$ be the current steering angle of the wheels; $v$ and $\alpha$ can be considered as control inputs to the car. The dynamics of the car can be modeled by following differential equations:

$$\dot{x} = v\cos(\theta), \;\; \dot{y} = v\sin(\theta), \;\; \dot{\theta} = v/R$$

where $(x, y)$ is the position of the tail of the car, $\theta$ is the heading direction and $R = L/\tan(\alpha)$ is the turning radius given the length of the car $L$ (Figure 1.2). A *tracking problem* for the car-like robot is to design a controller, given a target trace and an actual trace of the configuration of the car up to the current time, produce the control inputs to the car so that the car tracks the target over time.

If the target is constant (for example, parallel parking), the problem can be decomposed into two subproblems: path planning and control. The path planning is to produce a set of consecutive circle and line segments that connect the current and the target configuration of the car (Figure 15.2). Although there are more complex tracking algorithms in practice [SM94],

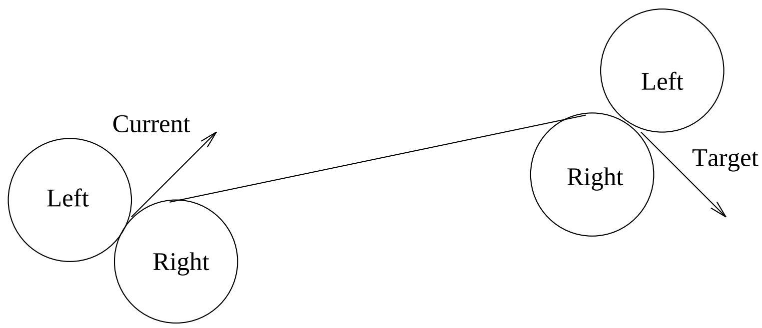


Figure 15.2: Path planning

the simplest tracking algorithm is as follows. For tracking on the line segments, set $\alpha = 0$; and for tracking on the circle segments, set $\alpha$ to be a nonzero constant (Left: $+\tan^{-1}\frac{L}{R}$ and Right: $-\tan^{-1}\frac{L}{R}$). In either case, the velocity can be set to a constant or to be controlled by a linear proportional controller.

When the target is moving, the path planning can be either applied at a fixed sampling rate or event-driven, where an event indicates a substantial change of the target. However, there is a simple control strategy for tracking a dynamic target, so that the path planning problem can be simplified, if not eliminated.

Let $C$ denote the constraint for the tracking problem: $(x = x_d) \wedge (y = y_d) \wedge (\theta = \theta_d)$. The desired property for tracking is persistence that can be expressed as $\Box\Diamond C^\epsilon$. We define an energy function for the controller as

$$\mathcal{E} = \frac{k_p}{2}(x_d - x)^2 + \frac{k_p}{2}(y_d - y)^2 + \frac{k_t}{2}(\theta_d - \theta)^2.$$

The controller is designed to make $\mathcal{E}$ go to its minimum.

Let $p = \int v dt$ be the length of the path. We have

$$v = \dot{p}, \quad \alpha = \tan^{-1}(\frac{L}{v}\dot{\theta}).$$

Using the gradient method, we would like to have

$$\dot{p} = -k_1\frac{\partial\mathcal{E}}{\partial p}, \quad \dot{\theta} = -k_2\frac{\partial\mathcal{E}}{\partial\theta}$$

where $\frac{\partial \mathcal{E}}{\partial p}$ and $\frac{\partial \mathcal{E}}{\partial \theta}$ can be computed as follows:

$$-\frac{\partial \mathcal{E}}{\partial p} = k_p(x_d - x)\frac{\partial x}{\partial p} + k_p(y_d - y)\frac{\partial y}{\partial p} + k_t(\theta_d - \theta)\frac{\partial \theta}{\partial p}$$

where

$$\frac{\partial x}{\partial p} = \frac{\dot{x}}{v} = \cos(\theta), \;\; \frac{\partial y}{\partial p} = \frac{\dot{y}}{v} = \sin(\theta), \;\; \frac{\partial \theta}{\partial p} = \frac{\dot{\theta}}{v} = \frac{\tan(\alpha)}{L}$$

and

$$-\frac{\partial \mathcal{E}}{\partial \theta} = k_p(x_d - x)\frac{\partial x}{\partial \theta} + k_p(y_d - y)\frac{\partial y}{\partial \theta} + k_t(\theta_d - \theta)$$

where

$$\frac{\partial x}{\partial \theta} \doteq -v\sin(\theta), \;\; \frac{\partial y}{\partial \theta} \doteq v\cos(\theta).$$

Let $d = \sqrt{(x_d - x)^2 + (y_d - y)^2}$ and $\theta' = \tan^{-1}(y_d - y, x_d - x)$. The control law for the tracking problem is:

$$\begin{aligned} v &= k_1[k_p d\cos(\theta' - \theta) + k_t(\theta_d - \theta)\frac{\tan(\alpha)}{L}] \\ \alpha &= \tan^{-1}(\frac{L}{v}k_2(k_p v d\sin(\theta' - \theta) + k_t(\theta_d - \theta))). \end{aligned}$$

Now we are able to analyze the stability of this control law. We argue that the control law is stable, since

$$\begin{aligned} \dot{\mathcal{E}} &= -[k_p(x_d - x)\dot{x} + k_p(y_d - y)\dot{y} + k_t(\theta_d - \theta)\dot{\theta}] \\ &= -[k_p(x_d - x)\cos(\theta) + k_p(y_d - y)\sin(\theta) + k_t(\theta_d - \theta)\frac{\tan(\alpha)}{L}]v \\ &= -\frac{1}{k_1}v^2 \leq 0. \end{aligned}$$

However, there are local minima or singularities. If $|\theta' - \theta| = \frac{\pi}{2}k$ and $\theta_d = \theta$ we get $v = 0$ even when $d \neq 0$. We can prove that they are the only singularities of this control law.

**Proposition 15.3.1** *This control law satisfies the condition that $v = 0$ iff*

$$(d = 0 \vee |\theta' - \theta| = \frac{\pi}{2}k) \bigwedge (\theta_d = \theta).$$

We have applied this control strategy to the soccer-playing robot car with high level target generation and low level target tracking. For the real car, the throttles and steering angles are limited to certain ranges, errors appear in both sensing and control. Gains in the control law are any positive reals in theory but should be chosen for the best performance in the practice. The model of the car-like robot, with the dynamics of forces, frictions and mechanical delays, has been developed. Even though the development itself is not closely related to the content of this thesis, the method may have a general interest for other applications. We describe the theory behind this model estimation method in Appendix D.

## 15.4 Summary

Constraint-based control synthesis and analysis provide a unitary framework for developing continuous/discrete hybrid control systems. However, we are not aiming either to subsume or to replace existing control theory, rather to formalize the underlying principles that are used informally in practice.

Local minima and/or singularities are the major problem for this type of controller. Normally singularities can be avoided if a higher level control strategy is used to detect singularities and to produce a sequence of intermediate configurations between the actual and the target configurations. Such a higher level control strategy becomes more important when the robot is embedded in a complex environment. In general, any complex robot control system should be developed and organized hierarchically. In the rest of Part III, we will propose a hierarchical robotic architecture.

# Chapter 16

# Robotic Architecture

We propose two kinds of hierarchy in a robot control system: one is composition hierarchy, the other is interaction hierarchy. Both of these hierarchies should be used as systematic mechanisms for building, organizing and analyzing a complex system incrementally.

The Constraint Net model supports composition hierarchies with modules, that has a set of inputs and outputs and performs a transduction from input traces to output traces. The *composition hierarchy* characterizes the hierarchy of composing complex modules from simple ones. The composition hierarchy of a system has a tree structure, in which the root is the whole net, and leaves are basic transductions. A complex module can be incrementally composed of simpler ones. A system can be tested and verified structurally.

The *interaction hierarchy* imposes the hierarchy of interaction or communication between modules. In the rest of this chapter, we focus on interaction hierarchies. We present a two-dimensional hierarchical structure, one is abstraction (or vertical) hierarchy and the other is arbitration (or horizontal) hierarchy.

## 16.1 Abstraction Hierarchy

A control system, in general, is implemented in a vertical hierarchy [Alb81] (Figure 16.1) corresponding to a hierarchical abstraction of time and domains (Figure 16.1). The bottom level sends control signals to various actuators, and at the same time, senses the state of actuators. Control signals flow down and the sensing signals flow up. Sensing signals from the environment are distributed over levels. Each level is a black box that represents the causal relationship between the inputs and the outputs. The inputs consist of the control signals from the higher level, the sensing signals from the environment and the current states from the lower level. The outputs consist of the control signals to the lower level and the current states to the higher

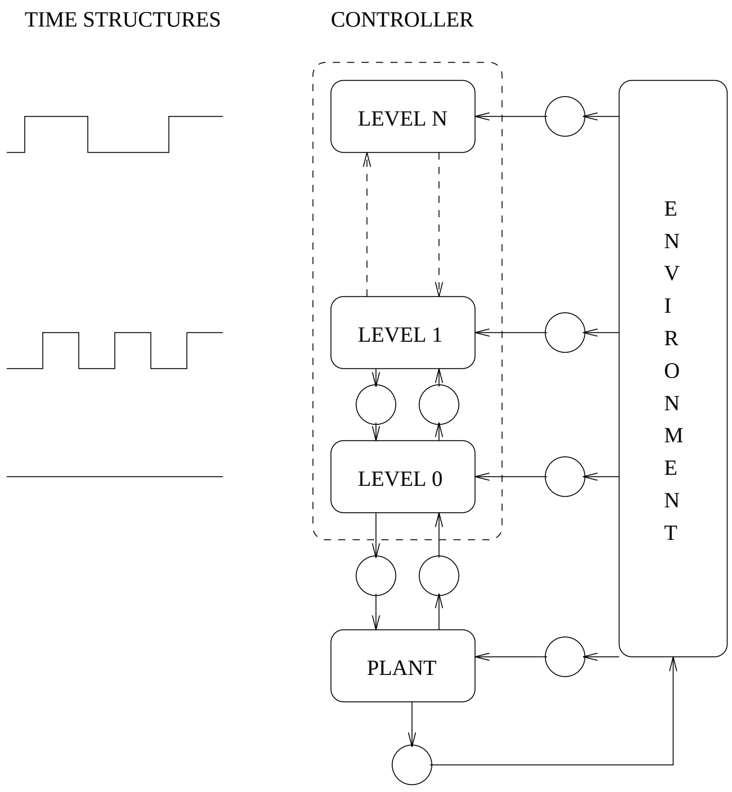


Figure 16.1: Abstraction hierarchy

level. Usually, the bottom level is implemented by analog circuits that function in continuous dynamics and the higher levels are realized by distributed computing networks.

In our framework of control synthesis, constraints are specified at different levels on different domains, with the higher levels more abstract and the lower levels more plant-dependent. For example, a multi-joint arm can be specified by two levels: the low level on joint space and the high level on task space.

A control system can be synthesized as a hierarchy of interactive embedded constraint solvers, that form the *abstraction hierarchy.* Each abstraction level solves constraints on its state space and produces the input to the lower level. The higher levels are composed of digital/symbolic event-driven control derived from discrete constraint methods and the lower levels are analog control based on continuous constraint methods.

## 16.2 Arbitration Hierarchy

Various constraints at same level of the abstraction hierarchy may form a constraint hierarchy. For example, safety requirements may always have the highest priority for satisfaction and persistence properties the lowest.

In our framework of control synthesis, constraint solvers at the same level of the abstraction hierarchy are coordinated via various arbitrations to compromise among different kinds of constraint, which form the *arbitration hierarchy.* (Figure 16.2).

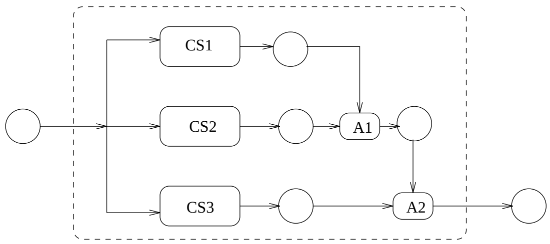


Figure 16.2: Arbitration hierarchy (CS's and A's denote solvers and arbiters respectively)

One type of arbitration can be modeled by the subsumption architecture [Bro86]. An output of a module in a higher layer can be *subsumed* by an output of a module in a lower layer. An input of a module in a lower layer can be *inhibited* by an output of a module in a higher layer.

Some other forms of subsumption and inhibition mechanism have been proposed in terms of compound synapses in neural activities [Bee90]. There are two different interaction functions: *gated* synapses where

$$f_g(I_S, I_G) = (U + I_G)I_S$$

and *modulated* synapses where

$$f_m(I_S, I_M) = \begin{cases} (1 + I_M)I_S & \text{if } I_M \geq 0 \\ I_S/(1 + |I_M|) & \text{otherwise.} \end{cases}$$

We can define some other arbitration functions:

- *Subsume*:

$$f_s(L, U) = \begin{cases} L & \text{if } L \neq 0 \\ U & \text{otherwise.} \end{cases}$$

- *Conditional pass*:

$$f_c(C, I) = \begin{cases} I & \text{if } C \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

- *Compromise*:

$$f_w(I_1, I_2) = w_1 I_1 + w_2 I_2, \quad w_1, w_2 > 0, w_1 + w_2 = 1.$$

In most cases, arbitration functions are nonlinear.

In general, multiple embedded constraint solvers are distributed and coordinated via various arbiters, which implement constraint hierarchies with the subsumption architecture or with some forms of compromise.

We have developed a control system for a hydraulically actuated arm with a low level PD controller and a high level end-point tracking and obstacle avoidance. Obstacle avoidance has a higher priority for satisfaction than end-point tracking. Both levels can be considered as applications of constraint-based control. This control system is a typical example of a hierarchical control system. The model of the arm and the hydraulic actuators, and the joint-level and end-point level control strategies are described in detail in Appendix C.

We have also developed a modeling and simulation environment, called ALERT (A Laboratory for Embedded Real-Time systems), in which all the examples described in this thesis have been experimented. In addition to the existing linear and nonlinear modules, we develop event, logic and arbitration modules for constructing complex hybrid control systems. ALERT and some examples are presented in Appendix B.

# Chapter 17

# Summary and Related Work

We have developed a systematic approach to control synthesis: a framework for constraint-based control and a framework for robotic architecture. In this chapter, we summarize this approach in terms of its power and limitations, and discuss some related work on constraint-based control and robotic architecture.

## 17.1 Summary

In this section, we summarize our framework for control synthesis and robotic architecture.

### 17.1.1 Power

Most robotic systems are constraint-based dynamic systems. Systems with adaptivity and learning exhibit this type of property as well. Constraint-based control synthesis provides a simple principle, on-line constraint satisfaction or energy minimization, that has been used implicitly in many existing control laws. With this framework, both discrete and continuous control strategies can be derived and analyzed, and many existing constraint methods can be applied to control. With this synthesis principle, verification can be simplified as well.

### 17.1.2 Limitations

Similar to the limitations of $\forall$-automata for representing dynamic behaviors, constraint-based specification cannot represent probabilistic or stochastic performance, or minimization of total cost over time (for example, energy cost for control). Constraint-based control differs from optimal control: the former is an on-line optimization that uses on-line constraint satisfaction, and the latter is an off-line optimization that uses calculus of variations [Lue79, War72, NK93a].

Constraint-based control synthesis is a methodology, a framework or a concept for a systematic development of control systems, rather than a new technique for the automatic generation of control systems. We will work on automatic or semi-automatic control synthesis for special classes of system in the future.

## 17.2 Related Work

Much work has been done on control synthesis. In this section, we survey only the most related and influential work. We consider two classes of work: one is on control strategies and the other is on control structures.

### 17.2.1 Constraint-based control

Early work on constraint-based control includes potential functions and the least constraint framework.

Potential functions generalize the conceptions of potential fields and forces, so that intention and action are intrinsically bound together in the description of the robot's task [Kod89]. Potential functions are used in obstacle avoidance and target tracking in unstructured environments [Kha86]. Various control methods, from PD controllers to adaptive control and neural nets, can be considered as applications of potential functions [Kod89].

The least constraint framework was proposed [Pai89, Pai91] to program robots with a high degree of freedom in changing environments. In this framework, sensed and actuated variables are related via a set of inequality or equality constraints, possibly changing over time. Constraints are satisfied at run time by a set of real-time constraint methods. This framework can deal with redundancy and the partial specification of motion, at the same time supporting modularity and parallelism.

Some recent work on auction-based control [CH93] can be considered as constraint-based control with the objective as the minimization of standard deviation.

To the best of our knowledge, there has been no research on formal requirements specification for control synthesis.

### 17.2.2 Robotic architecture

Much work has been done on robot control structures. Our concept of a two-dimensional interaction hierarchy derives from the work done by Albus and Brooks.

From the point of view of robotic systems design, Albus [Alb81] studied the hierarchical goal-directed behavior and proposed the sensory-processing hierarchy. In this structure, high-level goals are decomposed through a succession of levels, each producing strings of more specific commands to the next lower level. The bottom level generates the drive signals to the robot, such as joints and grippers. Each control level is a separate process with a limited scope of responsibility, independent of the details at other levels. Thus, such a structure provides a foundation for future modular, "plug compatible" hardware and software for robots and real-time sensory interactive control applications.

Brooks [Bro86] proposed a robust layered control system for mobile robots, called the subsumption architecture, Unlike the traditional decomposition of a mobile robot control system into functional modules, Brooks decomposed a mobile robot control system into task-achieving behaviors. Such a decomposition meets the requirements of multiple goals, multiple sensors and robustness.

Many real control systems use the concept of hierarchy. For example, Sahota and Mackworth [SM94] developed a hierarchical control structure for a soccer playing robot, with high level behavior bidding and path generation and low level path tracking. Zhao [Zha91] developed a synthesis method for nonlinear control systems with high level path planning and navigation in phase spaces and low level path tracking using linear control.

Nerode and Kohn [NK93b] proposed a multiple agent hybrid control architecture. The key capabilities of the architecture are: reactive and adaptive mechanisms, distributed structures with coordination, dynamic hierarchization, provable correctness and real-time response. The central mechanism for providing these capabilities is an on-line restricted automated theorem prover associated with each agent. Extensibility and robustness are also considered in this architecture. Some other work on hybrid control systems [GNRR93] has also been done recently.

# Part IV

# Conclusions and Further Research

*The greatest accomplishment seems unfinished,*
*yet its applications are endless.*
*The greatest fullness seems empty,*
*yet its applications are never exhausted.*
— *Tao Teh Ching, Lao Tzu*

*The greatest conclusion seems stuttering,*
*yet its implications are endless.*
*The greatest future work seems crude,*
*yet its fruits are never exhausted.*
— *Zhang Ying*

# Chapter 18

# Conclusions and Further Research

We have taken an integrated approach to the design and analysis of robotic systems and behaviors by establishing a foundation for modeling, analyzing, specifying, verifying and synthesizing complex artifacts that interact with changing environments. We have developed a semantic model for hybrid dynamic systems, two languages for requirements specification, a formal method for behavior verification, and a systematic approach to control synthesis.

In this chapter, we review what has been achieved in this research, and point out possible topics for the future.

## 18.1 Conclusions

We have decomposed the problem of design and analysis into four phases: modeling, specification, synthesis and verification. We have developed formal methods for each individual phase, and the relationships among all the phases.

First, we have developed a semantic model for hybrid dynamic systems, that we call Constraint Nets (CN). Based on abstract algebra and topology, we have represented both time and domains in abstract forms, and uniformly formalized basic elements of dynamic systems in terms of traces and transductions. We have studied both primitive and event-driven transductions.

CN is an abstraction and generalization of dataflow networks, while the behavior of a system (the semantics of a model) is formally obtained using the fixpoint theory of continuous algebra. In particular, CN models a dynamic system as a set of interconnected transductions, while the behavior of the system is the set of input/output traces of the system satisfying all the relationships imposed by the transductions. CN models a hybrid system using event-driven transductions, while the events are generated and synchronized within the system.

The motivation for developing CN is for modeling hybrid dynamic systems. However, we have shown that CN is as powerful as existing computational models so that both sequential and analog computations can be modeled. In order to study system behaviors formally, we have defined abstraction and equivalence of systems and behaviors using homomorphism and quotient algebra.

Second, we have developed two languages, TLTL and timed $\forall$-automata, for requirements specification. TLTL is a linear temporal logic extended with real-time modal operators. Timed $\forall$-automata are nondeterministic finite state automata augmented with local and global time bounds. As with CN, both languages are defined on abstract time and domains.

Third, we have developed a formal method, based on model checking and stability analysis, for behavior verification. This verification method is semi-automatic if the time structure is discrete, and is automatic, if, in addition, the domains are finite as well; the time complexity of the resulting verification algorithm is polynomial in both the size of the model and the size of the specification.

Fourth, we have developed a systematic approach to control synthesis. In this approach, desired properties of behaviors are specified with various forms of constraints using timed $\forall$-automata, such that the accepting automaton-states of the $\forall$-automata represent the neighborhoods of the solution set of the given constraints. Constraint-based control is then synthesized as embedded constraint solvers that, together with the dynamics of the plant and the environment, solve the constraints on-line. For the purposes of both design and analysis, we advocate a two-dimensional hierarchical structure for control systems.

As a whole, we have established a theoretical foundation for developing robotic systems and analyzing robotic behaviors (Figure 18.1).

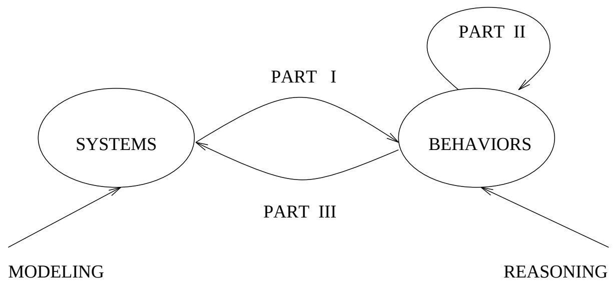


Figure 18.1: Summary

The major contributions of this thesis are summarized as follows:

- Constraint Nets for hybrid systems modeling and analysis

  CN possesses the essential properties of a desired model for robotic systems (modified from [LS90]), namely:

  > *Real-Time*: time is explicitly represented,
  >
  > *Symmetrical*: the dynamics of environments as well as the dynamics of plants and control can be modeled,
  >
  > *Hybrid*: multiple time and domain structures are uniformly formalized,
  >
  > *Hierarchical*: multiple levels of abstraction are provided, and
  >
  > *Formal*: formal syntax and semantics are defined, and formal analysis is facilitated.

- TLTL and timed $\forall$-automata for requirements specification

  TLTL specifies discrete/continuous sequential/timed behaviors uniformly; timed $\forall$-automata provide a simple alternative to TLTL, which is illuminating, and, in some cases, more powerful.

- a formal method for behavior verification

  This method applies to behaviors of hybrid systems in general, and is semi-automatic for discrete time systems and automatic for discrete time and finite domain systems.

- constraint-based requirements specification and control synthesis

  This approach proposes a general framework for control synthesis with a simple principle. Control synthesis and system verification are coupled via requirements specification.

- an integrated approach to the design and analysis of robotic systems and behaviors

  This thesis decomposes the problems in the design and analysis of robotic systems and behaviors, and focuses on the relationships among modeling, specification, synthesis and verification.

## 18.2 Further Research

We propose further research in both theory and practice.

### 18.2.1 Theory

We have proposed a foundation for the design and analysis of robotic systems and behaviors. There are more questions than answers; all we have done is to take the first step in a long journey. Further work includes:

- modeling and analyzing probabilistic and stochastic systems and behaviors

  Many robotic systems cannot be modeled exactly, due to the lack of knowledge of, or to the uncertainty in, the dynamics of the plant and the environment. It is important to model systems under uncertainty and to analyze their behaviors with probabilities.

- more expressive specification languages

  There are behaviors that are not expressible using TLTL or timed $\forall$-automata, such as maximizing global utilities and timed behaviors over intervals. Other specification languages, with more expressive power and pertaining formal verification procedures, are yet to be explored. For example, we can extend time bounds on timed automaton-states to both lower and upper bounds, while keeping the verification rules simple.

- (semi-)automatic verification for special classes of hybrid system

  There are simple hybrid systems that have algorithmic verification [ACHH93]. More work along this line can be done. For example, a finite automaton coupled to a linear continuous system is a special class of hybrid system that might have simpler verification procedures.

- (semi-)automatic synthesis and analysis of controllers for special classes of system

  For finite domain systems, controllers can be synthesized automatically, though with a high complexity. For linear systems, stability can be analyzed semi-automatically. More work along this line can be done. For example, it is possible to develop an algorithm that can (semi-)automatically synthesize and analyze a finite automaton that controls a linear continuous system.

- more extensive study on behavior abstraction

  We have provided the notion of behavior abstraction based on homomorphism. Other notions of abstraction can be defined; for example, implication can be considered as a type of abstraction where $A \rightarrow B$ means $B$ is an abstraction of $A$. (Under this definition, a requirements specification is an abstraction of the system model; a nondeterministic model is an abstraction of the deterministic system.) Given this notion of abstraction, the properties of behavior equivalence can be further studied.

### 18.2.2 Practice

We have already developed, based on our semantic model, a visual programming and simulation environment called ALERT: A Laboratory for Embedded Real-Time systems. Further work includes:

- a programming language with a real-time semantics

  CN is an abstraction of dataflow models for hybrid systems, with abstract data types and abstract reference time. An instantiation of the data types and the reference time results in a programming language, which can be used for both modeling and programming (control). ALERT is such a language for modeling.

- a specification and verification environment based on our methods

  Timed $\forall$-automata have a graphical representation, which can be implemented on a graphical user interface. The formal verification method for discrete time systems can be implemented on an interactive theorem prover.

- an integrated design and analysis environment for developing robotic systems

  Both CN and timed $\forall$-automata can be implemented on a graphical user interface, resulting in an integrated environment that facilitates both verification and simulation.

- more extensive study on some real machines to uncover more design problems

  This thesis establishes a theoretical foundation for the problem of design and analysis, which, nevertheless, are abstracted from our experiences on real machines. Our research aims, not to invent, but to understand, discover, formalize and solve new problems. The guiding research principle is "from practice to theory, and from theory to practice."

# Bibliography


[AC87] P. E. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *IJCAI-87*, pages 268–272, 1987.

[AC88] P. E. Agre and D. Chapman. What are plans for? Technical Report A.I. Memo 1050, MIT AI Lab, September 1988.

[ACHH93] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. 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*, number 736 in Lecture Notes on Computer Science, pages 209 – 229. Springer-Verlag, 1993.

[Ad81] H. Abelson and A. A. diSessa. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. MIT Press, 1981.

[AD90] R. Alur and D. Dill. Automata for modeling real-time systems. In M. S. Paterson, editor, *ICALP90: Automata, Languages and Programming*, number 443 in Lecture Notes on Computer Science, pages 322 – 335. Springer-Verlag, 1990.

[AD91] R. Alur and D. Dill. The theory of timed automata. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 45 – 73. Springer-Verlag, 1991.

[Agh85] G. A. Agha. Actor: A model of concurrent computation in distributed systems. Technical Report 844, MIT AI LAB, 1985.

[Agm54] S. Agmon. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6:382–392, 1954.

[AH89] R. Alur and T. A. Henzinger. A really temporal logic. In *30th Annual Symposium on Foundations of Computer Science*, pages 164 – 169, 1989.

[Alb81] J. S. Albus. *Brains, Behavior, and Robotics.* BYTE Publications, 1981.

[All90] J. F. Allen. Towards a general theory of action and time. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 464 – 479. Morgan Kaufmann Publishers Inc., 1990.

[Ash86] E. A. Ashcroft. Dataflow and eduction: Data-driven and demand-driven distributed computation. In J. W. deBakker, W.P. deRoever, and G. Rozenberg, editors, *Current Trends in Concurrency*, number 224 in Lecture Notes on Computer Science, pages 1 – 50. Springer-Verlag, 1986.

[Atl89] M. Atlevi. $SDT$ — a real-time $CASE$ tool for the $CCITT$ specification language $SDL$. In *FORTE*, pages 9 – 13, 1989.

[BC86] R. A. Brooks and J. H. Connell. Asynchronous distributed control system for a mobile robot. *SPIE Mobile Robots*, 727, 1986.

[BCN88] R. A. Brooks, J. H. Connell, and Peter Ning. Herbert: A second generation mobile robot. Technical report, MIT AI Lab, January 1988. A. I. Memo 1016.

[BD89] M. Boddy and T. Dean. Solving time-dependent planning problems. In *IJCAI-89*, pages 979 – 984, 1989.

[BD91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259 – 273, March 1991.

[Bee90] R. D. Beer. *Intelligence as Adaptive Behavior: An Experiment in Computational Neuroethology.* Academic Press, 1990.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A really abstract concurrent model and its temporal logic. In *Thirteenth Annual ACM Symposium on Principles of Programming Languages*, 1986.

[BL90] A. Benveniste and P. LeGuernic. Hybrid dynamical systems theory and the SIGNAL language. *IEEE Transactions on Automatic Control*, 35(5):535 – 546, May 1990.

[Bod91] M. Boddy. Anytime problem solving using dynamic programming. In *AAAI-91*, pages 738 – 743, 1991.

[Bra84] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology.* MIT Press, 1984.

[Bro86] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), March 1986.

[Bro88] R. A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network, September 1988.

[Bro91] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1 – 3), January 1991.

[BS87] J. A. Brzozowski and C. J. Seger. A characterization of ternary simulation of gate networks. *IEEE Transactions on Computers*, 36(11), November 1987.

[CE82] Y. Censor and T. Elfving. New method for linear inequalities. *Linear Algebra and Its Applications*, 42:199–211, 1982.

[CH93] S. H. Clearwater and B. A. Huberman. Thermal markets for controlling building environments. Technical report, Dynamics of Computation Group, Xerox Palo Alto Research Center, September 1993.

[Cha87] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[Cli81] W. D. Clinger. Foundations of actor semantics. Technical Report 633, MIT AI LAB, May 1981.

[Con90] J. Connell. *A Colony Architecture for an Artificial Creature.* Academic Press, 1990.

[CPHP87] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. LUSTRE: A declarative language for programming synchronous systems. In *ACM Proceedings on Principles of Programming Languages*, pages 178 – 188, 1987.

[Cra86] J. J. Craig. *Introduction to Robotics.* Addison-Wesley Publishing Company, Inc., 1986.

[CWG88] P. E. Caines, S. Wang, and R. Greiner. Dynamical (default) logic observers for finite automata. In *Conference on Information Sciences and Systems, Princeton*, March 1988.

[dHdR91] J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors. *Real-Time: Theory in Practice*. Number 600 in Lecture Notes on Computer Science. Springer-Verlag, 1991.

[DW91] T. Dean and M. Wellman. *Planning and Control*. Morgan Kaufman, 1991.

[Elm77] H. Elmqvist. SIMNON – an interactive simulation program for nonlinear systems. In *Proc. of Simulation 77*, 1977.

[Eme90] E. Emerson. Temporal and modal logic. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. Elsevier, MIT Press, 1990.

[FF84] R. E. Filman and D. P. Friedman. *Coordinated Computing: : Tools and Techniques for Distributed Software*. McGraw-Hill Book Company, 1984.

[FT89] I. Foster and S. Taylor. *Strand: New Concepts in Parallel Programming*. Prentice Hall, 1989.

[Gem90] M. C. Gemignani. *Elementary Topology*. Dover Publications, Inc., 1990.

[GL87] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *AAAI-87*, pages 677 – 682, 1987.

[GN92] J. Guckenheimer and A. Nerode. Simulation for hybrid systems and nonlinear control. In *Proc. IEEE Conference on Decision and Control*, pages 2980–2981, December 1992.

[GNRR93] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors. *Hybrid Systems*. Number 736 in Lecture Notes on Computer Science. Springer-Verlag, 1993.

[Gor] M. Gordon. A formal method for hard real-time programming. manuscript.

[Gor92] M. Gordon. Verifying real-time programs: A case study. In J. Bowen, editor, *Towards Verified Systems*. 1992. To appear.

[GPR67] L. G. Gubin, B. T. Polyak, and E. V. Raik. The method of projections for finding the common point of convex sets. *U.S.S.R. Computational Mathematics and Mathematical Physics*, pages 1–24, 1967.

[Hal90] R. Hale. Using temporal logic for prototyping: The design of a lift controller. In H.S.M. Zedan, editor, *Real-Time Systems, Theory and Applications.* Elsvier Science Publishers B.V. (North-Holland), 1990.

[Hen88] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[Hew88] C. Hewitt. Offices are open systems. In B.A. Huberman, editor, *The Ecology of Computation.* Elsevier Science Publisher B.V.(North-Holland), 1988.

[Hew91] C. Hewitt. Open information systems semantics for DAI. *Artificial Intelligence*, 47(1 – 3), January 1991.

[HMP91a] T. A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies. In *Proceedings of the 18th Annual ACM Symposium on Principles of Programming Languages*, 1991.

[HMP91b] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 226–251. Springer-Verlag, 1991.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Pretice-Hall, 1985.

[Hol82] W. M. L. Holcombe. *Algebraic Automata Theory.* Cambridge University Press, 1982.

[HP85] D. Harel and A. Pnueli. On the development of reactive system. In K.R. Apt, editor, *Logics and Models of Concurrent Systems.* Springer-Verlag Belin Heidelberg, 1985.

[Hub88] B. A. Huberman. The ecology of computation. In B. A. Huberman, editor, *The Ecology of Computation.* Elsevier Science Publishers B.V.(North-Holland), 1988.

[Huh87] M. N. Huhns, editor. *Distributed Artificial Intelligence.* Research Notes in Artificial Intelligence. Pitman, London, 1987.

[HZ91] M. R. Hansen and C. Zhou. Semantics and completeness of duration calculus. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 209 – 225. Springer-Verlag, 1991.

[Inca] Integrated Systems Inc. AutoCode User's Guide.

[Incb] Integrated Systems Inc. SystemBuild User's Guide.

[Incc] The MathWorks Inc. Similink User's Guide.

[JLHM91] M. S. Jaffe, N. G. Leveson, M. P. E. Heimdahl, and B. E. Melhart. Software requirements analysis for real-time process-control systems. *IEEE Transactions on Software Engineering*, 17(3):241 – 257, March 1991.

[Kah74] G. Kahn. The semantics of a simple language for parallel processing. In *Proceedings of IFIP Congress 74*, pages 471 – 475, 1974.

[Kha86] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90 – 99, 1986.

[Khi61] G. F. Khilmi. *Qualitative Methods in the Many Body Problem.* Science Publishers Inc. New York, 1961.

[KM88] K. M. Kahn and M. S. Miller. Language design and open systems. In B. A. Huberman, editor, *The Ecology of Computation.* Elsevier Science Publishers B.V.(North-Holland), 1988.

[Kod89] D. E. Koditschek. Robot planning and control via potential functions. In J. Craig O. Khatib and T. Lozano-Perez, editors, *The Robotic Review 1.* MIT Press, 1989.

[LA89] D. M. Lyons and M. A. Arbib. A formal model of computation for sensory-based robotics. *IEEE Transactions on Robotics and Automation*, 5(3):280 – 293, June 1989.

[Lam91] L. Lamport. The temporal logic of actions. Technical Report 79, Digital Systems Research Center, Palo Alto, California, December 1991.

[Lam93] L. Lamport. Hybrid systems in $tla^+$. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, number 736 in Lecture Notes on Computer Science, pages 77 – 102. Springer-Verlag, 1993.

[Lat91] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[LD89] Y.K.H. Lau and R.W. Daniel. A csp model for distributed control software design. Technical Report OUEL 1789/89, Robotics Research Group, Department of Engineering Science, University of Oxford, 1989.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finit-state concurrent programs satisfy their linear specification. In *Proc. 12th Ann. ACM Sym. on Principles of Programming Languages*, pages 97 – 107, 1985.

[LS90] J. Lavignon and Y. Shoham. Temporal automata. Technical Report STAN-CS-90-1325, Robotics Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305, 1990.

[Lue79] D. G. Luenberger. *Introduction to Dynamic Systems: Theory, Models and Applications.* John Wiley & Sons, 1979.

[MA86] E. G. Manes and M. A. Arbib. *Algebraic Approaches to Program Semantics.* Springer-Verlag, 1986.

[Mae89] P. Maes. The dynamics of action selection. In *IJCAI-89*, Detroit, 1989.

[McD90] D. McDermott. A temporal logic for reasoning about processes and plans. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 436 – 463. Morgan Kaufmann Publishers Inc., 1990.

[McM92] Kenneth L. McMillan. Symbolic model checking. Technical Report CMU-CS-92-131, Department of Computer Science, Carnegie Mellon, 1992.

[Mea55] G. H. Mealy. A method for synthesizing sequential circuits. *Bell Sys. Tech. Journal*, 34:1045 – 1079, 1955.

[MH69] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267 – 310, 1983.

[Min86] M. Minsky. *The Society of the Mind.* Simon and Schuster, New York, 1986.

[MM79] G. Milne and R. Milner. Concurrent processes and their syntax. *JACM*, (2):302 – 321, April 1979.

[MMP91] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de-Bakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 448 – 484. Springer-Verlag, 1991.

[MMT91] M. Merritt, F. Modugno, and M.R. Tuttle. Time-constrained automata. In J.C.M. Baeten and J.F. Groote, editors, *CONCUR-91*, number 527 in Lecture Notes on Computer Science, pages 393 – 407. Springer-Verlag, 1991.

[Moo56] E. F. Moore. Gedanken-experiments on sequential machines. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.

[Mos85] B. Moszkowski. A temporal logic for multilevel reasoning about hardware. *Computer*, 18(2), February 1985.

[MP71] R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.

[MP87] Z. Manna and A. Pnueli. Specification and verification of concurrent programs by $\forall$-automata. In *Proc. 14th Ann. ACM Symp. on Principles of Programming Languages*, pages 1–12, 1987.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[MT75] M. D. Mesarovic and Y. Takahara. *General Systems Theory: Mathematical Foundations*. Academic Press, 1975.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR-90*, number 458 in Lecture Notes on Computer Science, pages 401 – 415. Springer-Verlag, 1990.

[MT91] F. Moller and C. Tofts. Relating processes with respect to speed. In J.C.M. Baeten and J.F. Groote, editors, *CONCUR-91*, number 527 in Lecture Notes on Computer Science. Springer-Verlag, 1991.

[Nil89] N. Nilsson. Action networks. In J. Tenenberg et. al, editor, *Proceedings from the Rochester Planning workshop: From Formal System to Practical Systems*, University of Rochester, New York, 1989.

[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*, number 736 in Lecture Notes on Computer Science, 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*, number 736 in Lecture Notes on Computer Science. Springer-Verlag, 1993.

[NS91] X. Nicollin and J. Sifakis. From ATP to timed graphs and hybrid systems. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 549 – 572. Springer-Verlag, 1991.

[Ost89] J. S. Ostroff. *Temporal Logic For Real-Time Systems.* John Wiley & Sons Inc., 1989.

[Pai89] D. K. Pai. Programming parallel distributed control for complex systems. In *IEEE International Symposium on Intelligent Control*, pages 426 – 432, 1989.

[Pai91] D. K. Pai. Least constraint: A framework for the control of complex mechanical systems. In *Proceedings of American Control Conference*, pages 426 – 432, Boston, 1991.

[Pet81] J. L. Peterson. *Petri-Net Theory and the Modeling of Systems.* Prentice-Hall, Inc., Englewood Cliffs, 1981.

[Pet86] C. A. Petri. "Forgotten topics" of net theory. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, number 255 in Lecture Notes on Computer Science, pages 500 – 514. Springer-Verlag, 1986.

[Pla89] J. Platt. Constraint methods for neural networks and computer graphics. Technical Report Caltech-CS-TR-89-07, Department of Computer Science, California Institute of Technology, 1989.

[QAF89] J. Quemada, A. Azcorra, and D. Frutos. A timed calulus for lotos. In *FORTE89*, pages 245 – 263, 1989.

[Qi94] R. Qi. Decision graphs: Algorithms and applications to influence diagram evaluation and high-level path planning under uncertainty, 1994. Ph.D. thesis, forthcoming.

[RK87] S. J. Rosenschein and L. P. Kaelbling. The synthesis of digital machines with provable epistemic properties. Technical Report Technical Note 412, SRI International, April 1987.

[RK89] S. J. Rosenschein and L. P. Kaelbling. Integrating planning and reactive control, 1989.

[RM86] D. E. Rumelhart and J. L. McClelland, editors. *Parallel Distributed Processing — Exploration in the Microstructure of Cognition*. MIT Press, 1986.

[Ros85] R. Rosen, editor. *Theoretical Biology and Complexity*. Academic Press, Inc., 1985.

[Ros89] S. J. Rosenschein. Synthesizing information–tracking automata from environment description. In *First International Conference on Reasoning and Knowledge Representation, Toronoto*, pages 386 – 393, 1989.

[Roy88] H. L. Royden. *Real Analysis, 3rd edition*. Macmillan Publishing Company, 1988.

[San90] J. T. Sandfur. *Discrete Dynamical Systems: Theory and Applications*. Clarendon Press, 1990.

[Sar89] V. Saraswat. Concurrent constraint programming languages. Technical report, Computer Science Department, Carnegie–Mellon University, 1989. Ph. D. thesis.

[Sch87] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *IJCAI-87*, pages 1039–1046, 1987.

[Sch91] M. Schoppers, editor. *Communications of ACM*. ACM, August 1991. Special Section on Real-Time Knowledge-Based Control Systems.

[SDLS90] N. Sepehri, G.A.M. Dumont, P.D. Lawrence, and F. Sassani. Cascade control of hydraulically actuated manipulators. *Robotica*, 8:207 – 216, 1990.

[Sha41] C. E. Shannon. Mathematical theory of the differential analyzer. *Journal of Mathematics and Physics*, 20:337 – 354, 1941.

[Sha87] E. Shapiro, editor. *Concurrent Prolog*. MIT press, 1987.

[Sho87] Y. Shoham. Temporal logics in ai: Semantical and ontological considerations. *Artificial Intelligence*, 33:89—104, 1987.

[Sho88] Y. Shoham. *Reasoning about Change*. MIT Press, 1988.

[Shu88] N. C. Shu. *Visual Programming*. Van Nostrand Reinhold Company Inc., 1988.

[SJG94] V. Saraswat, R. Jagadeesan, and V. Gupta. Programming in timed concurrent constraint languages. In B. Mayoh, E. Tyugu, and J. Penjam, editors, *Constraint Programming*, NATO Advanced Science Institute Series, Series F: Computer And System Sciences. 1994.

[SM94] M. Sahota and A. K. Mackworth. Can the situated robot play soccer. In *1994 Canadian Artificial Intelligence, Banff, Alberta*, May 1994.

[Ste80] Jr. G. L. Steele. The definition and implementation of a computer programming language based on constraints. Technical Report AI-TR-595, MIT AI Lab, August 1980.

[Sut89] I. E. Sutherland. Micropipeline. *Communication of ACM*, 32(6):720 – 738, June 1989.

[Tay92] J. H. Taylor. Software requirements specification for modeling design, development, and evaluation of distributed, hybrid, intelligent contol. Technical report, Odyssey Research Associates, Ithaca, NY, April 1992.

[Tho90] W. Thomas. Automata on infinite objects. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science*. MIT Press, 1990.

[Vic89] S. Vickers. *Topology via Logic*. Cambridge University Press, 1989.

[War72] J. Warga. *Optimal Control of Differential and Functional Equations*. Academic Press, 1972.

[Wd90] D. S. Weld and J. deKleer, editors. *Qualitative Reasoning About Physical Systems*. Morgan Kaufmann Publishers, Inc., 1990.

[Wil91] B. C. Williams. A theory of interactions: Unifying qualitative and quantitive algebraic reasoning: Extended report. Technical Report P91-00127, SSL-91-03, Palo Alto Research Center, 1991.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72 – 99, 1983.

[Yas71] A. Yasuhara. *Recursive Function Theory and Logic.* Academic Press, 1971.

[YM] K. Yang and K. G. Murty. New iterative methods for linear inequalities. Unpublished.

[Zha89] Y. Zhang. Transputer-based behavioral module for multi-sensory robot control. In Mike Reeve and Steven Ericsson Zenith, editors, *Parallel Processing and Artificial Intelligence*, Communication Process Architecture. Wiley, 1989.

[Zha90] Y. Zhang. Object oriented modeling for sensor-guided real-time robot control. In Alan S. Wagner, editor, *Transputer Research and Applications 3.* IOS Press, 1990.

[Zha91] F. Zhao. Phase space navigator: Towards automating control synthesis in phase spaces for nonlinear control systems. In *Proc. of the 3rd IFAC International workshop on Artificial Intelligence in Real Time Control.* Pergamon Press, 1991.

[ZM92] Y. Zhang and A. K. Mackworth. Modeling behavioral dynamics in discrete robotic systems with logical concurrent objects. In S.G. Tzafestas and J.C. Gentina, editors, *Robotics and Flexible Manufacturing Systems.* Elsevier Science Publishers B.V., 1992.

[ZM93a] Y. Zhang and A. K. Mackworth. Constraint programming in constraint nets. In *First Workshop on Principles and Practice of Constraint Programming*, pages 303–312, 1993. A revised version will appear in a book with the same title in MIT Press, 1995.

[ZM93b] Y. Zhang and A. K. Mackworth. Parallel and distributed constraint satisfaction: Complexity, algorithms and experiments. In Laveen N. Kanal, editor, *Parallel Processing for Artificial Intelligence.* Elsevier/North Holland, 1993.

[ZM94] Y. Zhang and A. K. Mackworth. Will the robot do the right thing? In *Proc. Artificial Intelligence 94*, pages 255 – 262, Banff, Alberta, May 1994.

# Part V

# Appendixes

# Appendix A

# Proofs of Theorems

In this appendix, we prove all the propositions and theorems in this thesis.

## A.1 Topological Structure of Dynamics

**Proposition 3.1.1** *For any topology on $X$, $X$ and $\emptyset$ are both open and closed.*
Proof: $X$ ($\emptyset$) is closed since $\emptyset$ ($X$) is open. □

**Proposition 3.1.2** *(1) A subset is closed iff it includes all its limit points. (2) A topology is trivial iff every point $x$ is a limit point of any subset with elements distinct from $x$. A topology is discrete iff no point is a limit point of any subset.*
Proof: (1) If a subset $S$ of $X$ is closed, $X - S$ is open and there is no point in $X - S$ that is a limit point of $S$. If there is no point in $X - S$ that is a limit point of $S$, $S$ is closed, since if $S$ is not closed, $X - S$ is not open. If $X - S$ is not open, there is at least one point in $X - S$ that is a limit point of $S$, otherwise every point in $X - S$ has a neighborhood in $X - S$, thus $X - S$ is open.

(2) If a topology is trivial, any point has only one neighborhood, the total set. If every point $x$ is a limit point of any subset with elements distinct from $x$, the topology is trivial since otherwise there is an open set $S \subset X$ and no point in $S$ is a limit point of $X - S$, contradiction. If a topology is discrete, any point is a neighborhood of itself, thus cannot be a limit point of any subset. If no point is a limit point of any subset, the topology is discrete since otherwise there is a point that is not open, which is a limit point of the total set, contradiction. □

**Proposition 3.1.3** *A topological space is connected iff the only sets that are both open and closed are the empty set and the total set.*

Proof: If there is $\emptyset \subset X' \subset X$ that is both open and closed, both $X'$ and $X - X'$ are non-empty open sets. Therefore, $X$ is separated. $\Box$

**Proposition 3.1.4** *(1) Continuous functions are closed under functional composition. (2) A function $f : X \to X'$ is continuous, iff $x \in X$ is a limit point of $S \subset X$ implies that $f(x)$ is a point or a limit point of $f(S) = \{f(x)|x \in S\}$.*
Proof: The first property is deduced directly from the definition of continuous functions. The second property is deduced from an equivalent definition of continuous functions, i.e., a function is continuous iff the inverse image of any closed subset is closed, and from the property that a closed subset includes all its limit points. $\Box$

**Proposition 3.1.5** *Let $\langle X, \tau \rangle$ be a topological space, $X' \subseteq X$ and $\tau' = \{W|W = X' \cap U, U \in \tau\}$. The collection $\tau'$ is a topology on $X'$.*
Proof: Deduced from the definition of topology. $\Box$

**Proposition 3.1.6** *Let $\{X_i\}_{i \in I}$ be a family of topological spaces and $J$ be an arbitrary index set. Then $(\times_I X_i)^J = \times_I X_i^J$.*
Proof: $\times_J(\times_I X_{ij})$ and $\times_I(\times_J X_{ij})$ are isomorphic. $\Box$

**Proposition 3.1.7** *A flat partial order is a cpo.*
Proof: $\perp_A$ is the least element and every directed subset is a chain with a greatest element. $\Box$

**Proposition 3.1.8** *The product of cpos is a cpo. Let $A = \times_I A_i$. The least element of $A$ is $\perp_A$ with $(\perp_A)_i = \perp_{A_i}, \forall i \in I$. Let $D$ be a directed subset of $A$. The least upper bound of $D$ is $\bigvee_A D$ with $(\bigvee_A D)_i = \bigvee_{A_i} D_i, \forall i \in I$, where $D_i$ is the projection of $D$ onto its ith component, i.e., $D_i = \Pi_i D$.*
Proof: According to the definition of least elements and least upper bounds. $\Box$

**Proposition 3.1.9** *The partial order topology of a non-trivial partial order is non-Hausdorff.*
Proof: For any $a <_A a'$, every neighborhood of $a$ includes $a'$. $\Box$

**Proposition 3.1.10** *Any continuous function is monotonic, i.e., if $f : A \to A'$ is continuous, then $a_1 \leq_A a_2$ implies $f(a_1) \leq_{A'} f(a_2)$.*

Proof: Suppose $f(a_1) \not\leq_{A'} f(a_2)$, there is an open set $S \subseteq A'$ including $f(a_1)$ but not $f(a_2)$. Therefore, $f^{-1}(S) \subseteq A$ is an open set including $a_1$ but not $a_2$. So $a_1 \not\leq_A a_2$. □

**Proposition 3.1.11** *Let $A$ and $A'$ be two cpos. Then $f : A \to A'$ is continuous iff for every directed subset $D \subseteq A$,*

1. *$f(D) = \{f(d)|d \in D\}$ is directed and*
2. *$f(\bigvee_A D) = \bigvee_{A'} f(D)$.*

Proof: The only if part: If $f$ is continuous, $f$ is monotonic (Proposition 3.1.10). Therefore, if $d$ is an upper bound of $d_1$ and $d_2$, $f(d)$ is an upper bound of $f(d_1)$ and $f(d_2)$. Therefore, if $D$ is directed, then $f(D)$ is directed and $f(\bigvee_A D) \geq_{A'} \bigvee_{A'} f(D)$. We now prove that $f(\bigvee_A D) \leq_{A'} \bigvee_{A'} f(D)$. If $f(\bigvee_A D) \not\leq_{A'} \bigvee_{A'} f(D)$, there is an open set $S \subseteq A'$ including $f(\bigvee_A D)$ but not $\bigvee_{A'} f(D)$. Therefore, $f^{-1}(S) \subseteq A$ is an open set including $\bigvee_A D$ but not any $d \in D$, contradicting to the definition of open sets in partial order topologies.

The if part: If conditions (1) and (2) are satisfied, $f$ is monotonic. Therefore, for any upward closed set $S$, $f^{-1}(S)$ is also upward closed. Since $f(\bigvee_A D) = \bigvee_{A'} f(D)$, if $S$ is inaccessible from any directed subset $f(D)$, then $f^{-1}(S)$ is inaccessible from any directed subset $D$. Therefore, $f$ is continuous since for any open set $S$, $f^{-1}(S)$ is open. □

**Proposition 3.1.12** *Metric topologies are Hausdorff.*
Proof: Given any two elements $x, x'$ with $l = d(x, x')$, $N^{l/2}(x) \cap N^{l/2}(x') = \emptyset$. □

**Proposition 3.1.13** *If $X$ is of a Hausdorff topology and $v : L \to X$ is a linear set of values, then $v \to v_1^*$ and $v \to v_2^*$ imply $v_1^* = v_2^*$.*
Proof: If $v_1^* \neq v_2^*$, There exist $N(v_1^*)$ and $N(v_2^*)$ such that $N(v_1^*) \cap N(v_2^*) = \emptyset$. Since $v \to v_1^*$ and $v \to v_1^*$, there is $l_0$, for all $l \geq_L l_0$, $v(l) \in N(v_1^*) \cap N(v_2^*)$, contradiction. □

**Proposition 3.1.14** *If $\times_I X_i$ is of the product topology and $v : L \to \times_I X_i$ is a linear set of values, then $v \to v^*$ iff $v_i \to v_i^*$ for all $i \in I$.*
Proof: If $v \to v^*$, then $v_i \to v_i^*$ for all $i \in I$ since for every neighborhood in the subbasis, $N^i(v^*) = \{\times_I V_j | \text{for all } j \neq i, V_j = X_j\}$, there is $l_0$, for all $l \geq_L l_0$, $v(l) \in N^i(v^*)$. If $v_i \to v_i^*$ for all $i \in I$, then $v \to v^*$ since every neighborhood $N(v^*)$ is the union of a set of neighborhood in the basis and for every neighborhood in the basis $N^J(v^*) = \{\times_I V_i | \text{for all } i \notin J, V_i = X_i\}$ with

a finite subset $J \subseteq I$, there is $l_0$, for all $l \geq_L l_0$, $v(l) \in N^J(v^*)$. □

**Proposition 3.2.1** *(1) For any time structure $\mathcal{T}$, if $T \subset \mathcal{T}$ has an upper bound in $\mathcal{T}$, $T$ has a least upper bound in $\mathcal{T}$.*
*(2) The following properties for a time structure are equivalent:*

*(a) $\mathcal{T}$ is discrete.*

*(b) Let $(t_1, t_2) = \{t | t_1 < t < t_2\}$. For all $t$, if $t$ is not the least element of $\mathcal{T}$, then $\exists t' < t$, denoted $pre(t)$, such that $(t', t) = \emptyset$, and for all $t$, if $t$ is not the greatest element of $\mathcal{T}$, then $\exists t' > t$, denoted $suc(t)$, such that $(t, t') = \emptyset$.*

*(c) $\mathcal{T}$ is* well-founded, *i.e., $\forall t \in \mathcal{T}$, $[\mathbf{0}, t)$ is finite.*

*(3) The following properties for a time structure are equivalent:*

*(a) $\mathcal{T}$ is continuous.*

*(b) $\mathcal{T}$ is* dense, *i.e., for all $t_1 < t_2$, there exists $t_0$ such that $t_1 < t_0 < t_2$.*

Proof: (1) For any $T \subset \mathcal{T}$ with an upper bound $t \in \mathcal{T}$, let $\tau = \inf\{m(t) | t$ is an upper bound of $T\}$. Since $\mathcal{T}$ is a time structure, $\{t | m(t) \leq \tau\}$ has a greatest element $t_0$. Since $T \subseteq \{t | m(t) \leq \tau\}$, $t_0$ is the least upper bound of $T$.

(2) (a) $\rightarrow$ (b): For any $t$, $t$ is not the least element of $\mathcal{T}$, let $\tau = \sup\{m(t') | t' < t\}$. Since $\mathcal{T}$ is a time structure, $\{t' | m(t') \leq \tau\}$ has a greatest element, denoted $t_0$. Since $\mathcal{T}$ is discrete, $t_0 < t$. However, $(t_0, t) = \emptyset$. For any $t$, $t$ is not the greatest element of $\mathcal{T}$, let $\tau = \inf\{m(t') | t' > t\}$. Since $\mathcal{T}$ is a time structure, $\{t' | m(t') \geq \tau\}$ has a least element, denoted $t_0$. Since $\mathcal{T}$ is discrete, $t_0 > t$. However, $(t, t_0) = \emptyset$.

(b) $\rightarrow$ (a): Every point has a neighborhood including no other points but itself. So every point is an open (or closed) set. Therefore, $\mathcal{T}$ is of discrete metric topology.

(b) $\rightarrow$ (c): If $\mathcal{T}$ is not well-founded, there is $t \in \mathcal{T}$, $[\mathbf{0}, t)$ is infinite. Therefore, $T = \{suc^n(\mathbf{0}) | n \in \mathcal{N}\} \subseteq [\mathbf{0}, t) \subset \mathcal{T}$. According to (1), $t_0 = \bigvee T \in \mathcal{T}$. However there is no $t < t_0$ such that $(t, t_0) = \emptyset$, contradiction.

(c) $\rightarrow$ (b): For any $t > \mathbf{0}$, there exists $t' < t$, $(t', t) = \emptyset$ since $[\mathbf{0}, t)$ is finite. For any $t$, $t$ is not the greatest element, there exists $t' > t$, $(t, t') = \emptyset$ since otherwise for any $t' > t$, $[\mathbf{0}, t')$ is infinite.

(3) (a) $\rightarrow$ (b) (Not Dense $\rightarrow$ Not Continuous): If $\mathcal{T}$ is not dense, there exist $t_1$ and $t_2$ such that $(t_1, t_2) = \emptyset$. Then $\mathcal{T}$ is separated (or not continuous) since $\mathcal{T}$ is the union of two disjoint, non-empty open sets $\{t|m(t) < m(t_1) + d(t_1, t_2)/2)\}$ and $\{t|m(t) > m(t_2) - d(t_1, t_2)/2\}$.

(b) $\rightarrow$ (a) (Not Continuous $\rightarrow$ Not Dense): If $\mathcal{T}$ is not continuous, $\mathcal{T}$ is the union of two disjoint, non-empty open (or closed) sets $T_1$ and $T_2$. Let $\tau_1 = \sup\{m(t)|t \in T_1\}$ and $\tau_2 = \inf\{m(t)|t \in T_2\}$. Since $\mathcal{T}$ is a time structure, $\{t|m(t) \leq \tau_1\}$ has a greatest element $t_1$ and $\{t|m(t) \geq \tau_2\}$ has a least element $t_2$. Since $T_1$ and $T_2$ are closed, $t_1 \in T_1$ and $t_2 \in T_2$. Therefore, $(t_1, t_2) = \emptyset$. $\Box$

**Proposition 3.2.2** *If $\mathcal{T}_0$ is a reference time of $\mathcal{T}_1$ and $\mathcal{T}_1$ is a reference time of $\mathcal{T}_2$, then $\mathcal{T}_0$ is a reference time of $\mathcal{T}_2$.*

Proof: According to the definition of a reference time structure. $\Box$

**Proposition 3.3.1** *$\{\perp_A\}$ is not $\tau$-open. The only neighborhood of $\perp_A$ is $\overline{A}$.*

Proof: According to the definition of topology. $\Box$

**Proposition 3.3.2** *For any domain, its partial order topology is finer than its derived metric topology, and both are non-Hausdorff.*

Proof: Trivial. $\Box$

**Proposition 3.3.3** *(1) Function $f : \overline{A} \rightarrow \overline{A'}$ is continuous in the partial order topology iff $f$ is strict or constant. (2) If $f : \overline{A} \rightarrow \overline{A'}$ is continuous in the derived metric topology, then $f$ is continuous in the partial order topology. (3) Function $f : \overline{A} \rightarrow \overline{A'}$ is continuous in the derived metric topology iff $f$ is continuous in the partial order topology and the restriction of $f$ on $A$ and $A'$ is continuous in the metric topology, namely, for any open subset $S$ of $A'$, $f^{-1}(S) \cap A$ is open.*

Proof: (1) If $f$ is strict or a constant, $f$ is continuous. If $f$ is continuous and $f$ is not strict, $f$ is constant since $\perp_A \leq a$ implies that $f(\perp_A) = f(a)$ for any $a$ if $f(\perp_A) \neq \perp_{A'}$.

(2) If $f$ is continuous in the derived metric topology, $f$ is strict or constant, since $\perp_A$ is a limit point of any $\{a\}$ and $f(\perp_A)$ is a point or a limit point of $\{f(a)\}$.

(3) If $f$ is strict or constant, and the restriction of $f$ on $A$ and $A'$ is continuous in the metric topology, then $f$ is continuous in the derived metric topology, since for any open set $S$ of $\overline{A'}$, $f^{-1}(S)$ is open. If $f$ is continuous in the derived metric topology, $f$ is strict or constant, since

in either case, the restriction of $f$ on $A$ and $A'$ must also be continuous. $\Box$

**Proposition 3.3.4** *Let $I$ be a finite index set. (1) Function $f : \times_I A_i \to A$ is continuous in the partial order topology iff $f$ is continuous w.r.t. all $i \in I$. (2) If $f : \times_I \overline{A_i} \to \overline{A}$ is continuous in the derived metric topology, then $f$ is continuous in the partial order topology. (3) Function $f : \times_I \overline{A_i} \to \overline{A}$ is continuous in the derived metric topology iff $f$ is continuous in the partial order topology and the restriction of $f$ on $\times_I A_i$ and $A$ is continuous in the product metric topology, namely, for any open subset $S$ of $A$, $f^{-1}(S) \cap \times_I A_i$ is open.*

Proof: (1) Let $I = \{1, 2\}$. If a function $f : A_1 \times A_2 \to A$ is continuous, it is right continuous since $\bigvee_A f(a, D) = f(a, \bigvee_{A_2} D)$. Similarly, it is left continuous. On the other hand, if $f$ is both left and right continuous, $f(\bigvee_{A_1 \times A_2} D) = f(\bigvee_{A_1} D_1, \bigvee_{A_2} D_2) = \bigvee_A f(D_1, \bigvee_{A_2} D_2) = \bigvee_A f(D_1, D_2) = \bigvee f(D)$ ([Hen88]). $I$ can be extended to any finite index set.

(2) If $f : \times_I \overline{A_i} \to \overline{A}$ is continuous in the derived metric topology, $f$ is continuous in the derived metric space w.r.t. any argument $i \in I$, $f$ is continuous in the partial order w.r.t. any argument $i \in I$ (Proposition 3.3.3 (2)), $f$ is continuous in the partial order (Proposition 3.3.4 (1)).

(3) If $f$ is strict or constant, and the restriction of $f$ on $\times_I A_i$ and $A$ is continuous in the metric topology, $f$ is continuous in the derived metric topology, since for any open set $S$ of $\overline{A}$, $f^{-1}(S)$ is open.

If $f$ is continuous in the derived metric topology, $f$ is strict or constant w.r.t. argument $i$ for all $i \in I$. In either case, the restriction of $f$ on $\times_I A_i$ and $A$ must also be continuous, since for any open set $S$ of $A$, either $f^{-1}(S) \subseteq \times_I A_i$ or the projection onto the $i$-th argument is $\overline{A_i}$ for any $i$. Therefore, $f^{-1}(S) \cap \times_I A_i$ is open. $\Box$

**Proposition 3.4.1** *Let $v : L \to \overline{A}$ be a linear set of values. Then*

*(1) $v \to \perp_A$, and*

*(2) $v \to v_1^*$ and $v \to v_2^*$ imply that either $v_1^* = v_2^*$ or one of $v_1^*$ and $v_2^*$ is $\perp_A$.*

Proof: (1) The only neighborhood of $\perp_A$ is $\overline{A}$. Therefore, $v(l) \in N(\perp_A)$ for all $l$. (2) If $v_1^* \neq v_2^*$, then one of them must be $\perp_A$, since the metric topology is Hausdorff with unique limits (Proposition 3.1.12, Proposition 3.1.13). $\Box$

**Proposition 3.4.2** *Let $v : L \to A$ for $A = \times_I A_i$. Then*

*(1) $v \to v^*$ iff $v_i \to v_i^*$ for all $i \in I$, and*

*(2) the set of limits $\{v^*|v \to v^*\}$ is a directed subset in $\langle A, \leq_A \rangle$ and has a greatest element.*
Proof: (1) follows from Proposition 3.1.14. (2) If $v : L \to \overline{A}$, then $\{v^*|v \to v^*\}$ has a greatest element. If the set of limits of $v_i : L \to A_i$ has a greatest element $v_i^*$, then the set of limits of $v : L \to \times_I A_i$ has a greatest element $v^*$ with $(v^*)_i = v_i^*$ for all $i \in I$. □

**Proposition 3.4.3** *Let $v : L \to A$ for $A = \times_I A_i$. Then* $(\lim v)_i = \lim v_i, \forall i \in I$.
Proof: $(\bigvee_A D)_i = \bigvee_{A_i} D_i$ where $D_i = \Pi_i D$. □

**Proposition 3.4.4** *If $v_1, v_2 : L \to A$ and $v_1(l) \leq_A v_2(l)$ for all $l \in L$, then* $\lim v_1 \leq_A \lim v_2$.
Proof: If $A$ is flat, $v_1^* \neq \perp_A$ implies that $v_2^* \neq \perp_A$. If $A$ is a product, $\lim {v_1^*}_i \leq_{A_i} \lim {v_2^*}_i$. Therefore, $\lim v_1^* \leq_A \lim v_2^*$. □

**Proposition 3.4.5** *For any time structure $\mathcal{T}$, $T_{\leq t-\tau}$ has a greatest element whenever $m(t) \geq \tau$.*
Proof: (1) $T_{\leq t-\tau} = \{t'|t' < t, d(t,t') \geq \tau\} = \{t'|t' < t, m(t') \leq m(t) - \tau\} = \{t'|m(t') \leq m(t) - \tau\}$ since $\tau > 0$. If $m(t) \geq \tau$, then $0 \leq m(t) - \tau < \sup m(\mathcal{T})$. Since $\mathcal{T}$ is a time structure, $T_{\leq t-\tau}$ has a greatest element. □

**Proposition 3.4.6** *Let $V : L \to A^{\mathcal{T}}$ for a linear order $L$ and a trace space $A^{\mathcal{T}}$. Then*

*(1) $V \to V^*$ iff $V(t) \to V^*(t)$ for all $t \in \mathcal{T}$, and*

*(2) the set of limits $\{V^*|V \to V^*\}$ is a directed subset in $\langle A^{\mathcal{T}}, \leq_{A^{\mathcal{T}}} \rangle$ and has a greatest element.*
Proof: Similar to the proof of Proposition 3.4.2. □

**Proposition 3.4.7** *Let $V : L \to A^{\mathcal{T}}$. Then* $(\lim V)(t) = \lim V(t), \forall t \in \mathcal{T}$.
Proof: Similar to the proof of Proposition 3.4.3. □

**Proposition 3.4.8** *For any time structure $\mathcal{T}$ and any event trace $e$, $\langle \mathcal{T}_e, d_e, \mu_e \rangle$ is a discrete sample time structure of $\mathcal{T}$.*
Proof: For any $t_e \in \mathcal{T}_e$ and $0 \leq \tau < \sup m(\mathcal{T}_e)$, let $T_e = \{t'_e|m(t'_e) \leq \tau, t'_e \in \mathcal{T}_e\}$ and $T = \bigcup_{t'_e \in T_e} \{t|t \leq t'_e\}$. If $T_e$ has no greatest element, $T$ has no greatest element. Furthermore, $e(T)$ is not defined, otherwise $\exists t_0 \in T, e$ is constant on $\{t|t \geq t_0, t \in T\}$ and $t_0$ would be an upper bound of $T_e$ in $\mathcal{T}_e$. However, if $e(T)$ is not defined, there will be no $t_e \in \mathcal{T}_e$ with $m(t_e) > \tau$, since $e$ is nonintermittent. Therefore, for any $t_e \in \mathcal{T}_e$ and $0 \leq \tau < \sup m(\mathcal{T}_e)$,

$T_e = \{t'_e | m(t'_e) \le \tau, t'_e \in \mathcal{T}_e\}$ has a greatest element.

For any $t_e \in \mathcal{T}_e$ and $0 \le \tau < \sup m(\mathcal{T}_e)$, let $T_e = \{t'_e | m(t'_e) \ge \tau, t'_e \in \mathcal{T}_e\}$ and $T = \bigcup_{t'_e \in T_e} \{t | t \ge t'_e\}$. Let $\tau' = \inf\{m(t) | t \in T\}$ and $t_0$ be the least element of $\{t | m(t) \ge \tau'\}$. If $T_e$ has no least element, $e(t_0)$ is not defined since $e$ is right-continuous. However, since $e$ is also non-intermittent, $e(t)$ is not defined $\forall t > t_0$, contradiction. Therefore, for any $t_e \in \mathcal{T}_e$ and $0 \le \tau < \sup m(\mathcal{T}_e)$, $T_e = \{t'_e | m(t'_e) \ge \tau, t'_e \in \mathcal{T}_e\}$ has a least element.

Therefore, $\mathcal{T}_e$ is a time structure.

For any $t_e \in \mathcal{T}_e$, $t_e > \mathbf{0}$, let $pre(t_e) = \{t'_e | t'_e < t_e, t'_e \in \mathcal{T}_e\}$ and $T = \bigcup_{t'_e \in pre(t_e)} \{t | t \le t'_e\}$. If $pre(t_e)$ has no greatest element, $T$ has no greatest element. Furthermore, $e(T)$ is not defined, otherwise $\exists t_0 \in T, e$ is constant on $\{t | t \ge t_0, t \in T\}$ and $t_0$ would be an upper bound of $pre(t_e)$ in $\mathcal{T}_e$. However, if $e(T)$ is not defined, $e(t_e)$ will not be defined since $e$ is nonintermittent. Therefore, $pre(t_e)$ has a greatest element.

For any $t_e \in \mathcal{T}_e$, $t_e$ is not the greatest element of $\mathcal{T}_e$, let $suc(t_e) = \{t'_e | t'_e > t_e, t'_e \in \mathcal{T}_e\}$ and $T = \bigcup_{t'_e \in suc(t_e)} \{t | t \ge t'_e\}$. Let $\tau = \inf\{m(t) | t \in T\}$ and $t_0$ be the least element of $\{t | m(t) \ge \tau\}$. If $suc(t_e)$ has no least element, $e(t_0)$ is not defined since $e$ is right-continuous. However, since $e$ is also non-intermittent, $e(t)$ is not defined $\forall t > t_0$, contradiction. Therefore, $suc(t_e)$ has a least element.

Therefore $\mathcal{T}_e$ is discrete. $\Box$

**Proposition 3.6.1** *The partial order of a domain is a cpo.*

Proof: A flat partial order is a *cpo*. The product partial order of *cpo*s is a *cpo*. $\Box$

**Proposition 3.6.2** *The partial order of a trace space is a cpo.*

Proof: The product partial order of *cpo*s is a *cpo*. $\Box$

**Proposition 3.6.3** *The partial order of an event space is a cpo.*

Proof: We first prove that the subpartial order with the set of nonintermittent and right-continuous traces of a trace space is a *cpo*.

Let $\mathcal{V} \subset \overline{A}^{\mathcal{T}}$ be the set of nonintermittent and right-continuous traces on a simple domain. The least element in $\mathcal{V}$ is $\lambda t.\ \bot_A$. The least upper bound of a directed subset $D$ of $\mathcal{V}$ is $\bigvee_{\mathcal{V}} D = \lambda t. \bigvee_{\overline{A}} D(t)$, which is also in $\mathcal{V}$ for the following reasons: First, according to Proposition 3.4.4, $(\bigvee_{\mathcal{V}} D)(T) \ge_{\overline{A}} \bigvee_{\overline{A}} D(T)$, if $(\bigvee_{\mathcal{V}} D)(T)$ is $\bot_A$, $d(T)$ is $\bot_A$ for all $d \in D$. Second, for any $t \in \mathcal{T}$, if $(\bigvee_{\mathcal{V}} D)(t) = \bot_A$, $(\bigvee_{\mathcal{V}} D)$ is right-continuous at $t$; if $(\bigvee_{\mathcal{V}} D)(t) = a \in A$, there is $d \in D$,

$d(t) = a$. Since $d$ is right-continuous at $t$, $(\bigvee_{\mathcal{V}} D)$ is right-continuous at $t$.

Because of the composite properties of nonintermittent traces and limits, nonintermittent and right-continuous traces are closed under least upper bounds for traces on composite domains as well.

Therefore, the partial order of an event space is a *cpo*. □

**Proposition 3.6.4** *A transliteration $f_{\mathcal{T}} : A^{\mathcal{T}} \to A'^{\mathcal{T}}$ on any time structure $\mathcal{T}$ is continuous if $f : A \to A'$ is continuous.*

Proof: Let $D \subseteq A^{\mathcal{T}}$ be directed, and $v^*$ be the least upper bound of $D$. We will prove that $f_{\mathcal{T}}(\bigvee_{A^{\mathcal{T}}} D) = \bigvee_{A'^{\mathcal{T}}} f_{\mathcal{T}}(D)$, i.e., for any $t$, $f_{\mathcal{T}}(v^*)(t) = (\bigvee_{A'^{\mathcal{T}}} f_{\mathcal{T}}(D))(t)$.

$$\begin{aligned} f_{\mathcal{T}}(v^*)(t) &= f(v^*(t)) = f(\bigvee_{A}\{v(t) | v \in D\}) \\ &= \bigvee_{A'}\{f(v(t)) | v \in D\} \quad \text{since } f \text{ is continuous} \\ &= \bigvee_{A'}\{f_{\mathcal{T}}(v)(t) | v \in D\} = \bigvee_{A'^{\mathcal{T}}} f_{\mathcal{T}}(D)(t). \end{aligned}$$

□

**Proposition 3.6.5** *A unit delay on any discrete time structure is continuous.*

Proof: Let $D \subseteq A^{\mathcal{T}}$ be directed and $v^*$ be the least upper bound of $D$. Since $\mathcal{T}$ is discrete, $pre(t)$ has a greatest element, which is denoted by $pre(t)$.

$$\begin{aligned} \delta^A_{\mathcal{T}}(v_0)(v^*)(t) &= \begin{cases} v_0 & \text{if } t = \mathbf{0} \\ v^*(pre(t)) = \bigvee_A\{v(pre(t)) | v \in D\} & \text{otherwise} \end{cases} \\ &= \bigvee_{A}\{\delta^A_{\mathcal{T}}(v_0)(v)(t) | v \in D\} \\ &= (\bigvee_{A^{\mathcal{T}}} \delta^A_{\mathcal{T}}(v_0)(D))(t). \end{aligned}$$

□

**Proposition 3.6.6** *A transport delay is continuous.*

Proof: Similar to the proof of Proposition 3.6.5. Since $\mathcal{T}$ is a time structure, for any $\tau > 0$, $t - \tau$ has a greatest element when $m(t) \geq \tau$. □

**Proposition 3.6.7** *An event-driven transduction $F^\circ$ is continuous if its primitive transduction $F$ on any discrete time structure is continuous.*

Proof: First, we prove sampling and extending are continuous. Let $\mathcal{T}$ be a time structure and $\mathcal{T}_r$ be a reference time structure of $\mathcal{T}$ with a reference time mapping $h$. Sampling is a transduction $S_{\mathcal{T},\mathcal{T}_r} : A^{\mathcal{T}_r} \to A^{\mathcal{T}}$. We prove that it is continuous.

Let $D \subseteq A^{\mathcal{T}_r}$ be directed and $v^*$ be the least upper bound of $D$. Let $\underline{v}$ be $S_{\mathcal{T},\mathcal{T}_r}(v)$.

$$\underline{v^*}(t) = v^*(h(t)) = \bigvee_A \{v(h(t))|v \in D\} = \bigvee_A \{\underline{v}(t)|v \in D\} = (\bigvee_{A^{\mathcal{T}}} \{\underline{v}|v \in D\})(t).$$

Therefore, $\underline{\bigvee_{A^{\mathcal{T}_r}} D} = \bigvee_{A^{\mathcal{T}}} \underline{D}$.

Similarly, extending is continuous since $h^{-1}(t_r) = \{t|m(t) \leq m_r(t_r)\}$ has a greatest element if $\exists t \in \mathcal{T}, \mu_r([\mathbf{0}_r, t_r)) \leq \mu([\mathbf{0}, t))$ or $\mu_r([\mathbf{0}_r, t_r)) < \mu(\mathcal{T})$.

The proof is divided into two steps. First, $F^\circ$ is continuous w.r.t. the second argument if $F$ is continuous on discrete time structures, since any event-based time is discrete, both sampling and extending are continuous, and continuity is closed under functional composition. Second, $F^\circ$ is continuous w.r.t the first argument. Therefore, according to Proposition 3.3.4 (1), $F^\circ$ is continuous.

Now we prove that it is continuous w.r.t. the first argument. Let $\mathcal{T}$ be any time structure and $v \in A^{\mathcal{T}}$ be fixed. For any directed subset $D$ of $\mathcal{E}^{\mathcal{T}}$, $D$ is a chain. According to the definition, $F^\circ_{\mathcal{T}}(D, v)$ is a chain too, i.e., a directed subset. Furthermore, for any $t$ if $(\bigvee_{\mathcal{E}^{\mathcal{T}}} D)(t) \neq \perp_{\mathcal{B}}$, there is $d \in D$ such that for all $t' \leq t, d(t') = (\bigvee_{\mathcal{E}^{\mathcal{T}}} D)(t')$, i.e., $\bigvee_{A'^{\mathcal{T}}} F^\circ_{\mathcal{T}}(D, v) \geq F^\circ_{\mathcal{T}}(\bigvee_{\mathcal{E}^{\mathcal{T}}} D, v)$. On the other hand, $F^\circ_{\mathcal{T}}$ is monotonic w.r.t. the first argument, i.e., $\bigvee_{A'^{\mathcal{T}}} F^\circ_{\mathcal{T}}(D, v) \leq F^\circ_{\mathcal{T}}(\bigvee_{\mathcal{E}^{\mathcal{T}}} D, v)$. Therefore, $\bigvee_{A'^{\mathcal{T}}} F^\circ_{\mathcal{T}}(D, v) = F^\circ_{\mathcal{T}}(\bigvee_{\mathcal{E}^{\mathcal{T}}} D, v)$, it is continuous w.r.t. the first argument. □

**Theorem 3.6.1** *Let $A$ be a $\Sigma$-domain structure and $\mathcal{T}$ a time structure. The $\Sigma$-dynamics structure $\mathcal{D}(\mathcal{T}, A) = \langle \mathcal{V}, \mathcal{F} \rangle$ satisfies (1) $\mathcal{V}$ is a multi-sorted set of cpos and (2) transliterations, transport delays and event-driven transductions in $\mathcal{F}$ are continuous in the partial order topology. If, in addition, $\mathcal{T}$ is discrete, all transductions in $\mathcal{F}$ are continuous in the partial order topology.*

Proof: Follows from Propositions 3.6.1 – 3.6.7. □

**Proposition 3.6.8** *A transliteration $f_{\mathcal{T}}$ is well-defined iff function $f$ is well-defined; $f_{\mathcal{T}}$ is strict w.r.t. an argument iff $f$ is strict w.r.t. the argument.*

Proof: According to the definitions of well-definedness and strictness. □

**Proposition 3.6.9** *Any delay is not strict. A unit delay on any discrete time structure is* *well-defined. A transport delay is well-defined.*

Proof: According to the definitions of well-definedness and strictness. □

**Proposition 3.6.10** *An event-driven transduction $F^\circ$ is well-defined iff $F$ on any discrete time structure is well-defined; $F^\circ$ is strict w.r.t. its event input, and $F^\circ$ is strict w.r.t. one of the other input arguments iff $F$ is strict w.r.t. the argument.*

Proof: Event-based time is discrete, and sampling and extending are well-defined. □

**Proposition 3.6.11** *A transliteration $f_{\mathcal{T}}$ is right-continuous if $f$ is continuous in the derived metric topology; $f_{\mathcal{T}}$ with $f : \times_I \overline{A_i} \to \overline{A}$ is nonintermittent if $f$ is strict, well-defined and continuous in the derived metric topology.*

Proof: For any neighborhood $N(f(v(t)))$, there is a neighborhood $N(v(t))$, such that $x \in N(v(t))$ implies $f(x) \in N(f(v(t)))$. For any neighborhood $N(v(t))$, there is $T = (t, t')$, $t'' \in T$ implies $v(t'') \in N(v(t))$. Therefore, for neighborhood $N(f(v(t)))$, there is $T = (t, t')$, $t'' \in T$ implies $f(v(t'')) \in N(f(v(t)))$.

If $f$ is strict and well-defined, $v(t)$ is well-defined implies that $f_{\mathcal{T}}(v)(t)$ is well-defined, $v(t)$ is not well-defined implies that for all $t' \geq t$ $f_{\mathcal{T}}(v)(t')$ is undefined. If, in addition, $f$ is continuous in the derived metric topology, $\lim v_T$ is well-defined implies that $\lim f(v)_{|T}$ is well-defined, $\lim v_T$ is not well-defined implies that $\lim f(v)_{|T}$ is undefined. □

**Proposition 3.6.12** *A delay is nonintermittent. A transport delay is right-continuous.*

Proof: The output of a delay is nonintermittent if its input is nonintermittent.

The output of a transport delay is right-continuous if its input is right-continuous. □

**Proposition 3.6.13** *An event-driven transduction is right-continuous. An event-driven transduction $F^\circ$ is nonintermittent if $F$ is nonintermittent.*

Proof: Any trace on a discrete time structure is right-continuous. Any extension of a discrete-time trace is right-continuous.

Both sampling and extending are nonintermittent and nonintermittent transductions are closed under functional composition. □

## A.2 The Constraint Net Model

**Proposition 4.1.1**

$$CN_1(I_1, O_1) \parallel CN_2(I_2, O_2) = CN_2(I_2, O_2) \parallel CN_1(I_1, O_1).$$

$$CN_1(I_1, O_1) \circ (CN_2(I_2, O_2) \circ CN_3(I_3, O_3)) = (CN_1(I_1, O_1) \circ CN_2(I_2, O_2)) \circ CN_3(I_3, O_3)$$

*if both sides are defined.*

$$CN_1(I_1, O_1) + (CN_2(I_2, O_2) + CN_3(I_3, O_3)) = (CN_1(I_1, O_1) + CN_2(I_2, O_2)) + CN_3(I_3, O_3)$$

*if both sides are defined.*

Proof: According to the definition of basic and combined operations. □

**Proposition 4.1.2** *Following are some properties of subnets:*

*(1) $CN_1$ and $CN_2$ are subnets of $CN_1 \parallel CN_2$.*

*(2) $CN_1$ and $CN_2$ are subnets of $CN_1 + CN_2$.*

*(3) $CN_1$ is a subnet of $CN_2 \circ CN_1$, however, $CN_2$ is not a subnet of $CN_2 \circ CN_1$.*

Proof: According to the definition of basic and combined operations. □

**Theorem 4.2.2** *Let $A$ and $A'$ be two cpos. If $f : A \times A' \to A'$ is a continuous function, then there exists a unique continuous function $\mu.f : A \to A'$, such that for all $a \in A$, $(\mu.f)(a)$ is the least fixpoint of $f_a : A' \to A'$, where $f_a = \lambda x.f(a, x)$, or equivalently, $\forall a \in A, (\mu.f)(a) = f(a, (\mu.f)(a))$.*

Proof: Let $F^0(a) = f(a, \perp_{A'})$ and $F^{k+1}(a) = f(a, F^k(a))$. Since $f$ is continuous, it is continuous w.r.t. the second argument. A continuous function in any partial order is also monotonic. Therefore,

$$F^0(a) \leq_{A'} F^1(a) \leq_{A'} F^2(a) \ldots \leq_{A'} F^k(a) \leq \ldots.$$

Let $\mu.f(a) = \bigvee_{A'}\{F^k(a) | k \geq 0\}$. Clearly $\mu.f(a)$ is the least fixpoint of $f_a : A' \to A'$.

Next we prove that $\mu.f$ is continuous. Clearly for every $k$, $F^k$ is continuous since $f$ is continuous and continuity is closed under functional composition. Therefore, for any directed subset $D$ of $A$,

$$\begin{aligned} \mu.f(\bigvee_A D) &= \bigvee_{A'}\{F^k(\bigvee_A D) | k \geq 0\} \\ &= \bigvee_{A'}\{\bigvee_{A'}\{F^k(D)\} | k \geq 0\} \end{aligned}$$

$$= \bigvee_{A'}\{\bigvee_{A'}\{F^k(a)|k \geq 0\}|a \in D\}$$
$$= \bigvee_{A'} \mu.f(D).$$

□

**Proposition 4.2.1** *Let $I \subseteq J$ be an index set. If $f : \times_I A_i \to A$ is a continuous function, then the extension of $f$, $f' : \times_J A_j \to A$ satisfying $f'(a) = f(a_{|I})$, is a continuous function.*
Proof: According to the definitions of continuous functions and product topologies. □

**Proposition 4.2.2** *Let $\{f_k : \times_J A_j \to A_k\}_{k\in K}$ be a family of continuous functions. Then $\vec{f} : \times_J A_j \to \times_K A_k$ with $\vec{f}(a)_k = f_k(a)$ is a continuous function.*
Proof: According to the definitions of continuous functions and product topologies. □

**Proposition 4.2.3** *If $\vec{f} : \times_J A_j \to \times_K A_k$ is a continuous function, $K \subseteq J$ and $I = J - K$, then $\vec{f}$ has a least fixpoint $\mu.\vec{f} : \times_I A_i \to \times_K A_k$.*
Proof: According to **Fixpoint Theorem II.** □

**Proposition 4.2.4** *Let $X$ be a set of variables and $O \subseteq X$ a set of output variables. Let $\{f_o : \times_{I_o} A_i \to A_o\}_{o\in O}$ be a set of continuous functions. Then the set of equations $\{o = f_o(\vec{x})\}_{o\in O}$ with $\vec{x} : I_o \to X$ has a least solution.*
Proof: Derived from Proposition 4.2.1, 4.2.2 and 4.2.3. □

**Proposition 4.2.5** *If a constraint net is composed of nonintermittent transductions, then its semantics is nonintermittent. If a constraint net is composed of right-continuous transductions, then its semantics is right-continuous.*
Proof: Both nonintermittent and right-continuous transductions are closed under least upper bounds. □

*If $CN'$ is a subnet of $CN$, $[\![CN]\!]_{|O(CN')}(\vec{i}) = [\![CN']\!](\vec{i}_{|I(CN')})$.*
Proof: Trivial. □

**Proposition 4.2.7** *Following are some properties associated with module operations:*

- Union*: If* $CN(I, O) = CN_1(I_1, O_1) \parallel CN_2(I_2, O_2)$*, then*

$$[\![CN(I,O)]\!] = [\![CN_1(I_1,O_1)]\!] \times [\![CN_2(I_2,O_2)]\!].$$

- Cascade connection*: If* $CN(I, O) = CN_2(I_2, O_2) \circ CN_1(I_1, O_1)$*, then*

$$[\![CN(I,O)]\!] = \{F_2 \circ F_1 | F_1 \in [\![CN_1(I_1,O_1)]\!], F_2 \in [\![CN_2(I_2,O_2)]\!]\}.$$

- Parallel connection*: If* $CN(I, O) = CN_1(I_1, O_1) + CN_2(I_2, O_2)$*, then*

$$[\![CN(I,O)]\!] = \{\langle F_1, F_2\rangle | F_1 \in [\![CN_1(I_1,O_1)]\!], F_2 \in [\![CN_2(I_2,O_2)]\!]\}.$$

- Feedback connection*: If* $CN'(I', O') = \mathcal{F}(CN(I, O))$*, then*

$$[\![CN'(I',O')]\!] = \{\mu.F | F \in [\![CN(I,O)]\!]\}$$

  *where* $\mu.F$ *is the the least fixpoint of* $F$*.*

Proof: According to the definition of the semantics of modules. □

**Proposition 4.2.8** *If* $CN_1(I_1, O_1)$ *and* $CN_2(I_2, O_2)$ *are well-defined modules, then* $CN_1(I_1, O_1)$ $\parallel CN_2(I_2, O_2)$*,* $CN_1(I_1, O_1) \circ CN_2(I_2, O_2)$ *and* $CN_1(I_1, O_1) + CN_2(I_2, O_2)$ *are well-defined modules.*

Proof: According to the definition of the well-definedness of modules. □

**Proposition 4.2.9** *Let* $A$ *and* $A'$ *be two cpos. If* $f : A \times A' \to A'$ *is a strict continuous function w.r.t. its second argument, then the least fixpoint of* $f$*, or the least solution of the equation* $o = f(i, o)$*, is undefined.*

Proof: $\mu.f = \lambda x.\perp_{A'}$. □

**Proposition 4.2.10** *A module* $CN(I, O)$ *is not well-defined if there is an output location* $l \in O$ *such that* $CN$ *has an algebraic loop on* $l$*.*

Proof: If $l \to l$, $l$ results in an undefined trace. However, the inverse is not true. If there exists a not well-defined transduction, the net may not be well-defined either. □

## A.3 Modeling in Constraint Nets

**Theorem 5.3.1** *Let $\Sigma_n = \langle\{n\}, \{\mathtt{0}, \mathtt{suc}, \mathtt{cond}\}\rangle$ be a signature. A partial recursive function can be computed by a sequential module in $\Sigma_n$-dynamics structure $\mathcal{D}(\mathcal{N}, \overline{\mathcal{N}})$ where $\overline{\mathcal{N}}$ denotes the $\Sigma_n$-domain structure $\langle\{\overline{\mathcal{N}}\}, \{0, suc, cond\}\rangle$.*
Proof: For any partial recursive function $f$, there is a sequential module $CN$ defined on the given dynamics structure. If $f(x)$ is defined, for any start event, there is an end event indicating the completion of the computation. □

**Proposition 5.3.1** [Sha41] *Equations 5.1 and 5.2 are equivalent, i.e., a function written in one form can be transformed into another.*
Proof: Refer to [Sha41]. Differentiate Equations 5.1 $n-1$ times we have a total of $n^2$ equations, from which we my eliminate the $n^2-1$ variables $x_2, \dot{x}_2, \ldots, x_2^{(n)}; \ldots; x_n, \dot{x}_n, \ldots, x_n^{(n)}$.

Equation 5.2 can be written as Equations 5.1 as follows. Differentiate both sides w.r.t. $t$ we obtain

$$\frac{\partial P}{\partial t} + \frac{\partial P}{\partial x}\dot{x} + \frac{\partial P}{\partial \dot{x}}\ddot{x} + \ldots + \frac{\partial P}{\partial x^{(n)}}x^{(n+1)} = 0$$

and

$$x^{(n+1)} = -\frac{\frac{\partial P}{\partial t} + \frac{\partial P}{\partial x}\dot{x} + \frac{\partial P}{\partial \dot{x}}\ddot{x} + \ldots + \frac{\partial P}{\partial x^{(n-1)}}x^{(n)}}{\frac{\partial P}{\partial x^{(n)}}} = -\frac{P_1(t, x, \dot{x}, \ldots, x^{(n)})}{P_2(t, x, \dot{x}, \ldots, x^{(n)})}.$$

Let $x_1 = x, x_2 = \dot{x}, \ldots, x_{n+1} = x^{(n)}, x_{n+2} = x^{(n+1)}$. We have

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\ldots & \\
\dot{x}_{n+1} &= x_{n+2} \\
\dot{x}_{n+2} &= x_0 P_1(t, x_1, x_2, \ldots, x_{n+1}) \\
\dot{x}_0 &= x_0^2 P_2'(t, x_1, x_2, \ldots, x_{n+1})
\end{aligned}$$

where

$$P_2'(t, x_1, x_2, \ldots, x_{n+1}) = \frac{\partial P_2}{\partial t} + \frac{\partial P_2}{\partial x_1}x_2 + \frac{\partial P_2}{\partial x_2}x_3 + \ldots + \frac{\partial P_2}{\partial x_{n+1}}x_{n+2}.$$

□

**Proposition 5.3.2** [Sha41] *If $x = \lambda t.f(t)$ is non-hypertranscendental, then its derivative $y = \lambda t.f'(t)$, its integral $z = \lambda t. \int_{t_0}^{t} f(t)dt$, and its inverse $w = \lambda t.f^{-1}(t)$ are non-hypertranscendental.*
Proof: Refer to [Sha41]. □

**Proposition 5.3.3** [Sha41] *Non-hypertranscendental functions are closed under functional composition.*
Proof: Refer to [Sha41]. □

**Proposition 5.3.4** *Given a constraint net of differential equations $\dot{x}_k = f_k(\vec{x}), k = 1, \ldots, n$ with $x_k(t_0) \in \mathcal{R}$ and $f_k : \mathcal{R}^n \to \mathcal{R}$ as partial or total functions, and given that all $f_k$ are smooth at $\vec{x}(t_0)$, the limiting semantics of the constraint net, based on the forward Euler method, is well-defined over $\mathcal{T} = [t_0, t_1]$ for some $t_1 > t_0$. In particular, $x = \lambda t.\Sigma_{n=0}^{\infty} \frac{x^{(n)}(t_0)}{n!}(t - t_0)^n$.*
Proof: If all $f_k$ are smooth, $x^{(n)}(t_0)$ exists, the semantics results in a Taylor expansion. □

**Theorem 5.3.2** *Let $\Sigma_r = \langle\{r\}, \{+, \cdot\}\rangle$ be a signature. A non-hypertranscendental function that is defined and smooth over a closed segment $\mathcal{T} = [t_0, t_1]$ can be computed by a constraint net of differential equations in $\Sigma_r$-dynamics structure $\mathcal{D}(\mathcal{T}, \overline{\mathcal{R}})$, where $\overline{\mathcal{R}}$ denotes the $\Sigma_r$-domain structure $\langle\{\overline{\mathcal{R}}\}, \{+, \cdot\}\rangle$.*
Proof: A non-hypertranscendental function that is defined and smooth over a closed segment $\mathcal{T} = [t_0, t_1]$ can be written as Equations 5.1 with $x(t_0)$ well-defined. Therefore, the constraint net has a well-defined solution. On the other hand, for any polynomial function $P$, $P(x) - P(y) = (x - y)P'(x, y)$ and $P'(x, y)$ is a polynomial that is bounded in any closed interval. Therefore, Lipschitz condition is satisfied. □

## A.4 Behavior Analysis

**Proposition 6.2.1** *If $\mathcal{R}_1 \subseteq \mathcal{R}_2$, $|\mathcal{R}_1|_m \geq |\mathcal{R}_2|_m$.*
Proof: Trivial. □

**Proposition 6.4.1** *(1) If $\langle S', \rightarrow'\rangle$ is an abstraction of $\langle S, \rightarrow\rangle$, the behavior corresponding to $\langle S', \rightarrow'\rangle$ is the abstraction of the behavior corresponding to $\langle S, \rightarrow\rangle$. (2) If $\langle S', \rightarrow'\rangle$ is an approximate abstraction of $\langle S, \rightarrow\rangle$, the behavior corresponding to $\langle S', \rightarrow'\rangle$ is a superset of the abstraction of the behavior corresponding to $\langle S, \rightarrow\rangle$.*
Proof: Trivial. □

## A.5 Behavior Verification

**Proposition 11.2.1** *Let $\{\alpha_q\}_{q\in Q}$ be invariants for $\mathcal{B}$ and $\mathcal{A}$. If $r$ is a run of $\mathcal{A}$ over a trace $v \in \mathcal{B}$, then $\forall t \in \mathcal{T}, v(t) \models \alpha_{r(t)}$.*

Proof: For any trace $v$, $v(\mathbf{0})$ is an initial state, therefore, $v(\mathbf{0}) \models \Theta$. In addition, $r$ is a run, $v(\mathbf{0}) \models e(r(\mathbf{0}))$. Therefore, $v(\mathbf{0}) \models e(r(\mathbf{0})) \wedge \Theta$. Since $e(r(\mathbf{0})) \wedge \Theta \rightarrow \alpha_{r(\mathbf{0})}$, we have $v(\mathbf{0}) \models \alpha_{r(\mathbf{0})}$.

Assume that $v(pre(t)) \models \alpha_{r(pre(t))}$. Therefore, $v(t) \models c(r(pre(t)), r(t)) \rightarrow \alpha_{r(t)}$ since $n(v(pre(t)), v(t))$. In addition, $v(t) \models c(r(pre(t)), r(t))$. Therefore, $v(t) \models \alpha_{r(t)}$.

Use the induction principle for well-founded sets, $v(t) \models \alpha_{r(t)}$ for all $t$. □

**Proposition 11.2.2** *Let $\{\alpha_q\}_{q\in Q}$ be a set of invariants for $\mathcal{B}$ and $\mathcal{A}$ and $r$ be a run of $\mathcal{A}$ over a trace $v \in \mathcal{B}$. If $\{\rho_q\}_{q\in Q}$ is a set of Liapunov functions for $\mathcal{B}$ and $\mathcal{A}$, then*

- *$\rho_{r(t)}(v(t)) \leq \rho_{r(pre(t))}(v(pre(t)))$ when $r(pre(t)) \in S$,*
- *$\rho_{r(t)}(v(t)) - \rho_{r(pre(t))}(v(pre(t))) \leq -\epsilon$ when $r(pre(t)) \in B$, and*
- *if $BS$ is the set of segments of consecutive $B$ and $S$-states in $r$, $\forall q^* \in BS, q^*$ has a finite number of $B$-states.*

Proof: According to the conditions of Liapunov functions. □

**Proposition 11.2.3** *Let $\{\alpha_q\}_{q\in Q}$ be a set of invariants for $\mathcal{B}$ and $\mathcal{A}$ and $r$ be a run of $\mathcal{A}$ over a trace $v \in \mathcal{B}$. If there exist local and global timing functions for $\mathcal{B}$ and $\mathcal{TA}$, then*

- *if $Sg(q)$ is the set of segments of consecutive $q$'s in $r$, $\forall q \in T, q^* \in Sg(q)$, $\mu(q^*) \leq \tau(q)$, and*
- *if $BS$ is the set of segments of consecutive $B$ and $S$-states in $r$, $\forall q^* \in BS, \mu_B(q^*) \leq \tau(bad)$.*

Proof: Let $s_i, i = 1 \cdots n$ be a sequence of $q$-states. Since

$$\gamma_q(s_2) - \gamma_q(s_1) \leq -\mu(s_1)$$

$$\gamma_q(s_3) - \gamma_q(s_2) \leq -\mu(s_2)$$

$$\cdots$$

$$\gamma_q(s_n) - \gamma_q(s_{n-1}) \leq -\mu(s_{n-1})$$

we have

$$\gamma_q(s_n) - \gamma_q(s_1) \leq -\Sigma_{i=1}^{n-1}\mu(s_i).$$

Since $\mu(s_n) \leq \gamma_q(s_n)$ and $\gamma_q(s_1) \leq \tau(q)$, we have $\Sigma_{i=1}^{n}\mu(s_i) \leq \tau(q)$.

Let $s_i, i = 1 \cdots n$ be a sub-sequence of $B$-states in a $BS$ segment. Since

$$\gamma'_{q'_1}(s'_1) - \gamma'_{q_1}(s_1) \leq -\mu(s_1)$$

$$\gamma'_{q'_2}(s'_2) - \gamma'_{q_2}(s_2) \leq -\mu(s_2)$$

$$\ldots$$

$$\gamma'_{q'_n}(s'_n) - \gamma'_{q_1}(s_n) \leq -\mu(s_n)$$

and

$$\gamma'_{q'_i}(s'_i) \geq \gamma'_{q_{i+1}}(s_{i+1})$$

we have

$$\gamma'_{q'_n}(s'_n) - \gamma'_{q_1}(s_1) \leq -\Sigma_{i=1}^{n}\mu(s_i).$$

Since $\gamma'_{q_1}(s_1) \leq \tau(bad)$ and $\gamma'_{q'_n}(s'_n) \geq 0$, we have $\Sigma_{i=1}^{n}\mu(s_i) \leq \tau(bad)$. □

**Proposition 11.2.4** *Given $Lc$ as the set of locations and $U \subseteq Lc$, $U$ is an abstraction of $Lc$ iff $[\![CN(U)]\!]$ is state-based and time-invariant.*
Proof: Trivial. □

**Proposition 11.2.5** *If $U$ is an abstraction of $Lc$, any property restricted on relations on $U$ can be verified by exploring the subspace transition system, $\langle \times_U A_{s_l}, \rightarrow_U \rangle$.*
Proof: $s'_1 \rightarrow_U s'_2 \rightarrow_U \ldots \rightarrow_U s'_n$ iff $s_1 \rightarrow_{Lc} s_2 \rightarrow_{Lc} \ldots \rightarrow_{Lc} s_n$. □

**Proposition 11.2.6** *If $CN_s$ is a subnet of $CN$, the set of locations of $CN_s$ is an abstraction.*
Proof: $CN_s$ can be considered as an independent subsystem, viz. $h(s_1) = h(s_2)$, $s_1 \rightarrow_{Lc} s'_1$ and $s_2 \rightarrow_{Lc} s'_2$ imply that $h(s'_1) = h(s'_2)$. According to the definition, the set of locations of $CN_s$ is an abstraction. □

**Proposition 11.2.7** *The set of output locations of unit delays is an abstraction.*
Proof: The set of output locations of unit delays induces a state transition system. □

**Proposition 11.2.8** *The set of input locations of unit delays is an abstraction.*

Proof: The set of input locations of unit delays induces a state transition system. □

**Proposition 11.2.9** *If $U$ is an abstraction and $I \subseteq I(CN)$, $U \cup I$ or $U - I$ is still an abstraction.*

Proof: Add or delete an input location does not change the property of abstraction. □

**Proposition 11.3.1** *Let $\{\alpha_q\}_{q \in Q}$ be invariants for $\mathcal{B}$ and $\mathcal{A}$. If $r$ is a run of $\mathcal{A}$ over $v \in \mathcal{B}$, $\forall t \in \mathcal{T}, v(t) \models \alpha_{r(t)}$.*

Proof: In order to prove this proposition, we shall introduce a variation of the method of continuous induction [Khi61]. A property $\Gamma$ is *inductive* on a time structure $\mathcal{T}$ iff for all $t_0 \in \mathcal{T}$, $\Gamma$ is satisfied at all $t < t_0$ implies that $\Gamma$ is satisfied at $t_0$. $\Gamma$ is *continuous* iff $\Gamma$ is satisfied at a non-greatest element $t \in \mathcal{T}$ implies that $\exists t' > t, \forall t < t'' < t'$, $\Gamma$ is satisfied at $t''$. Note that when $\mathcal{T}$ is discrete, any property is continuous. The theorem of continuous induction [Khi61] says:

**Theorem A.5.1** *If the property $\Gamma$ is inductive and continuous on a time structure $\mathcal{T}$ and $\Gamma$ is satisfied at $\mathbf{0}$, $\Gamma$ is satisfied at all $t \in \mathcal{T}$.*

We prove that the property $v(t) \models \alpha_{r(t)}$ is satisfied at $\mathbf{0}$ and is both inductive and continuous on any time structure $\mathcal{T}$.

- Initiality: Since $v(\mathbf{0}) \models \Theta$ and $v(\mathbf{0}) \models e(r(\mathbf{0}))$, we have $v(\mathbf{0}) \models \Theta \wedge e(r(\mathbf{0}))$. According to the *Initiality* condition of invariants, we have $v(\mathbf{0}) \models \alpha_{r(\mathbf{0})}$.

- Inductivity: Suppose $v(t) \models \alpha_{r(t)}$ is saisfied at $\mathbf{0} \leq t < t_0$. Since $r$ is a run over $v$, $\exists q \in Q$ and $t_1' < t_0, \forall t, t_1' \leq t < t_0$, $r(t) = q$ and $v(t_0) \models c(q, r(t_0))$. According to the *Consecution* condition of the invariants, $\exists t_2' < t_0, \forall t, t_2' \leq t < t_0$, $v(t) \models \alpha_q$ implies $v(t_0) \models c(q, r(t_0)) \rightarrow \alpha_{r(t_0)}$. Therefore, $\forall t, \max(t_1', t_2') \leq t < t_0$, $r(t) = q$, $v(t) \models \alpha_q$ (assumption), $v(t_0) \models c(q, r(t_0)) \rightarrow \alpha_{r(t_0)}$ and $v(t_0) \models c(q, r(t_0))$. Thus, $v(t_0) \models \alpha_{r(t_0)}$.

- Continuity: Suppose $v(t_0) \models \alpha_{r(t_0)}$. Since $r$ is a run over $v$, $\exists q \in Q$ and $t_1' > t_0, \forall t, t_0 < t < t_1'$, $r(t) = q$ and $v(t) \models c(r(t_0), q)$. According to the *Consecution* condition of the invariants, $\exists t_2' > t_0, \forall t, t_0 < t < t_2'$, $v(t_0) \models \alpha_{r(t_0)}$ implies $v(t) \models c(r(t_0), q) \rightarrow \alpha_q$. Therefore, $\forall t, t_0 < t < \min(t_1', t_2')$, $r(t) = q$, $v(t_0) \models \alpha_{r(t_0)}$ (assumption), $v(t) \models c(r(t_0), q) \rightarrow \alpha_q$ and $v(t) \models c(r(t_0), q)$. Thus, $\forall t, t_0 < t < \min(t_1', t_2')$, $v(t) \models \alpha_{r(t)}$.

□

**Theorem A.5.1**

Proof: We call a time point $t \in \mathcal{T}$ *regular* iff $\Gamma$ is satisfied at all $t'$, $\mathbf{0} \leq t' \leq t$. Let $T$ denote the set of all regular time points. $T$ is not empty since $\Gamma$ is satisfied at $\mathbf{0}$. We prove the theorem by contradiction, i.e., assume that $\Gamma$ is not satisfied at all $t \in \mathcal{T}$. Therefore, $T \subset \mathcal{T}$ is bounded above; let $t_0 = \bigvee T \in \mathcal{T}$ be the least upper bound of $T$ ($t_0$ exists according to Proposition 3.2.1). Since $t_0$ is the least upper bound, it follows that $\Gamma$ is satisfied at all $t$, $\mathbf{0} \leq t < t_0$. Since $\Gamma$ is inductive, it is satisfied at time $t_0$. Therefore, $t_0 \in T$.

Since $T \subset \mathcal{T}$, $t_0$ is not the greatest element in $\mathcal{T}$. Let $T' = \{t | t > t_0\}$. There are two cases: (1) if $T'$ has a least element $t'$, since $\Gamma$ is inductive, $t' \in T$ is a regular time point. (2) otherwise, for any $t' \in T'$, $\{t | t_0 < t < t'\} \neq \emptyset$. Since $\Gamma$ is also continuous, we can find a $t' \in T'$ such that $\Gamma$ is satisfied at all $T'' = \{t | t_0 < t < t'\}$. Therefore, $t$ is a regular time point $\forall t \in T''$. Both cases contradict the fact that $t_0$ is the least upper bound of the set $T$. □

**Proposition 11.3.2** *Let $\{\alpha_q\}_{q \in Q}$ be invariants for $\mathcal{B}$ and $\mathcal{A}$ and $r$ be a run of $\mathcal{A}$ over a trace $v \in \mathcal{B}$. If $\{\rho_q\}_{q \in Q}$ is a set of Liapunov functions for $\mathcal{B}$ and $\mathcal{A}$, then*

- $\rho_{r(t_2)}(v(t_2)) \leq \rho_{r(t_1)}(v(t_1))$ *when* $\forall t_1 \leq t \leq t_2, r(t) \in B \cup S$,
- $\frac{\rho_{r(t_2)}(v(t_2)) - \rho_{r(t_1)}(v(t_1))}{\mu([t_1, t_2))} \leq -\epsilon$ *when* $t_1 < t_2$ *and* $\forall t_1 \leq t \leq t_2, r(t) \in B$, *and*
- *if $BS$ is the set of segments of consecutive $B$ and $S$-states in $r$, then $\forall q^* \in BS, \mu_B(q^*)$ is finite.*

Proof: For any run $r$ over $v$ and for any segments $q^*$ of $r$ with only bad and stable states, $\rho$ on $q^*$ is nonincreasing, i.e., let $I$ be the time interval of $q^*$, for any $t_1 < t_2 \in I$, $\rho_{r(t_1)}(v(t_1)) \geq \rho_{r(t_2)}(v(t_2))$, and the decreasing speed at the bad states is no less than $\epsilon$. Let $m$ be the upper bound of $\{\rho_{r(t)}(v(t)) | t \in I\}$. Since $\rho_q \geq 0$, $\mu_B(q^*) \leq m/\epsilon < \infty$. □

**Proposition 11.3.3** *Let $\{\alpha_q\}_{q \in Q}$ be invariants for $\mathcal{B}$ and $\mathcal{A}$ and $r$ be a run of $\mathcal{A}$ over a trace $v \in \mathcal{B}$. If there exist local and global timing functions for $\mathcal{B}$ and $\mathcal{TA}$, then*

- *if $Sg(q)$ is the set of segments of consecutive $q$'s in $r$, then $\forall q \in T, q^* \in Sg(q)$, $\mu(q^*) \leq \tau(q)$, and*
- *if $BS$ is the set of segments of consecutive $B$ and $S$-states in $r$, then $\forall q^* \in BS$, $\mu_B(q^*) \leq \tau(bad)$.*

Proof: Similar to the proofs of Proposition 11.2.3 and Proposition 11.3.2. □

**Theorem 11.3.1** *The verification rules (I), (L) and (T) are sound if the following conditions of $\mathcal{B}$ and $\mathcal{TA}$ are satisfied:*

- *$\mathcal{T}$ is an infinite time structure.*
- *All traces in $\mathcal{B}$ are specifiable by $\mathcal{TA}$.*

*The verification rules are complete if the following conditions of $\mathcal{B}$ and $\mathcal{TA}$ are satisfied:*

- *$\{\langle v, r\rangle | v \in \mathcal{B}, r$ is a run over $v\}$ is time-invariant.*
- *All transitions from $R$ to non-$R$-states are left-closed, i.e., if $r$ is a run, and there is a transition from a $R$-state to a $B$-state or a $S$ state at $t$, then $r(t) \in B \cup S$. (For discrete time structures, this condition is always satisfied.)*

Proof: Soundness is derived from Propositions 11.3.1, 11.3.2 and 11.3.3. For any trace $v$, there is a run since $v$ is specifiable by $\mathcal{TA}$. For any run $r$ over $v$, if any automaton-state in $R$ appears infinitely many times in $r$, $r$ is accepting. Otherwise there is a time point $t_0$, the sub-sequence $r$ on $I = \{t \in \mathcal{T} | t \geq t_0\}$, denoted $q^*$, has only bad and stable automaton-states. If there exist a set of invariants and a set of Liapunov functions, $\mu_B(q^*)$ is finite. Since time is infinite, all the automaton-states appearing infinitely many times in $r$ belong to $S$; $r$ is accepting too. Therefore, every trace is accepting for the automaton. If there exists a set of local and global timing functions, every trace satisfies the timing constraints.

On the other hand, if $\mathcal{TA}$ is valid over $\mathcal{B}$, there exist a set of invariants, a set of Liapunov functions, and a set of local and global timing functions that satisfy the requirements.

The set of invariants can be constructed as follows: $\forall s \forall q, s \models \alpha_q$ iff the pair $\langle q, s\rangle$ is reachable, i.e., $\exists r, v, t, r(t) = q \wedge v(t) = s$. We shall prove that $\{\alpha_q\}_{q \in Q}$ is a set of invariants.

- Initiality: if $\Theta(s) \wedge e(q)(s)$, $\exists r, v, r(\mathbf{0}) = q$ and $v(\mathbf{0}) = s$. Therefore, $s \models \alpha_q$.
- Inductivity: $\forall v, t$, if $\exists t' < t, \forall t' \leq t'' < t$, $\exists r, r(t'') = q$ $(v(t'') \models \alpha_q)$, then $\exists r$, $\exists t'_0 < t, \forall t'_0 \leq t'' < t$, $r(t'') = q$. If $v(t) \models c(q, q')$, then $r(t) = q'$, i.e., $v(t) \models \alpha_{q'}$. Therefore, $v(t) \models c(q, q') \rightarrow \alpha_{q'}$.
- Continuity: $\forall v, t$, if $\exists r, r(t) = q$ $(v(t) \models \alpha_q)$, and $\exists t' > t \exists q', \forall t' < t'' < t, v(t'') \models c(q, q')$, $\forall t' < t'' < t, r(t'') = q'$. Therefore, $\exists t' > t, \forall t' < t'' < t, c(q, q') \rightarrow \exists r, r(t'') = q'$.

Given the above constructed invariants, a set of Liapunov functions can be constructed as follows:

- $\forall q \in R$ and $s \models \alpha_q$, let $\rho_q(s) = 0$.
- $\forall q \notin R$ and $s \models \alpha_q$, the Liapunov function is defined as follows. For any $r, v, t$ with $r(t) = q$ and $v(t) = s$, let $q^*$ be a segment of $r$ with only bad and stable states starting at $q$, and $\mu_B(q^*)$ be the measure of $B$-states in $q^*$. Let $\rho_q(s)$ be the longest such measure for all $r, v, t$ with $r(t) = q$ and $v(t) = s$, i.e., $\rho_q(s) = \sup\{\mu_B(q^*)\}$.

We shall prove that $\{\rho_q\}_{q\in Q}$ is a set of Liapunov functions and global timing functions. For $q, q' \notin R$, let $\langle q, s\rangle \prec \langle q', s'\rangle$ iff $\exists r, v, t < t'$, $\forall t < t'' < t', r(t'') \notin R$, $r(t) = q, v(t) = s$ and $r(t') = q'$ and $v(t') = s'$. Since $\{\langle v, r\rangle\}$ is time-invariant, $\prec$ is transitive. Therefore, $\langle q, s\rangle \prec \langle q', s'\rangle$ implies $\rho_q(s) \geq \rho_{q'}(s')$.

- *Definedness*: $\forall q \in Q, s \models \alpha_q$, $\rho_q$ is defined at $s$.
- *Non-increase*: $\forall v \in \mathcal{B}$, $\forall q \in S, q' \in R$,

  $$\{\alpha_q \wedge \rho_q = w\} v^- \{c(q, q') \rightarrow \rho_{q'} \leq w\}$$

  is trivially satisfied. $\forall q \in S, q' \in B \cup S$,

  $$\{\alpha_q \wedge \rho_q = w\} v^- \{c(q, q') \rightarrow \rho_{q'} \leq w\}$$

  is satisfied since $\langle q, s\rangle \prec \langle q', s'\rangle$.

  $\forall v \in \mathcal{B}$, $\forall q \in B \cup S, q' \in S$,

  $$\{\alpha_q \wedge \rho_q = w\} v^+ \{c(q, q') \rightarrow \rho_{q'} \leq w\}$$

  is satisfied since $\langle q, s\rangle \prec \langle q', s'\rangle$. $\forall q \in R, q' \in S$, $c(q, q')$ is $false$ since all transitions from $R$ to non-$R$-states are left-closed.

- *Decrease*: $\forall v \in \mathcal{B}$, $\forall q \in B, q' \in Q$,

  $$\{\alpha_q \wedge \rho_q = w \wedge t_c = t\} v^- \{c(q, q') \rightarrow \frac{\rho_{q'} - w}{\mu([t, t_c))} \leq -1\}.$$

  $\forall q \in R, q' \in B$,

  $$\{\alpha_q \wedge \rho_q = w \wedge t_c = t\} v^+ \{c(q, q') \rightarrow \frac{\rho_{q'} - w}{\mu([t, t_c))} \leq -1\}$$

  is trivially satisfied since $c(q, q')$ is $false$. $\forall q \in B \cup S, q' \in B$,

  $$\{\alpha_q \wedge \rho_q = w \wedge t_c = t\} v^+ \{c(q, q') \rightarrow \frac{\rho_{q'} - w}{\mu([t, t_c))} \leq -1\}.$$

The local timing functions can be defined similarly. □

**Proposition 11.4.4** *All transitions from $R$ to non-$R$-states are left-closed, if the following conditions are satisfied:*

- *$\mathcal{TA}$ is open and complete.*
- *$\forall q \in R$, $q_1 \notin R$ and $q_2 \in R$, $c(q, q_1) \wedge c(q, q_2)$ is not satisfiable.*
- *All traces in $\mathcal{B}$ are right-continuous.*

Proof: Since $\mathcal{TA}$ is open, $\forall q \in Q, q' \in R$, $c(q, q')$ is open. Therefore, $\forall q \in Q, \bigvee_{q' \in R} c(q, q')$ is open. Since $\forall q \in R$, $q_1 \notin R$ and $q_2 \in R$, $c(q, q_1) \wedge c(q, q_2)$ is not satisfiable, $(\bigvee_{q' \in R} c(q, q')) \wedge (\bigvee_{q' \in B \cup S} c(q, q'))$ is not satisfiable. Since $\mathcal{TA}$ is complete, $\bigvee_{q' \in R} c(q, q')$ and $\bigvee_{q' \in B \cup S} c(q, q')$ are complementary. Therefore, $\bigvee_{q' \notin R} c(q, q')$ is closed. Since all traces in $\mathcal{B}$ are right-continuous, for all $v$, $t$, if $t$ is a limit point to the right time points $T$, $v(t)$ is a point or a limit point of $v(T)$. If $\exists t' > t, \forall t < t'' < t', v(t'') \in \bigvee_{q' \notin R} c(q, q')$, $v(t) \in \bigvee_{q' \notin R} c(q, q')$. Therefore, all transitions from $R$ to non-$R$-states are left-closed. □

## A.6 Constraint-Based Dynamic Systems

**Proposition 14.1.1** *If $\{X_i\}_{i \in I}$ are ((asymptotically) stable) equilibria, then $\bigcup_I X_i$ is an ((asymptotically) stable) equilibrium.*
Proof: Trivial. □

**Theorem 14.1.1** *$X^* \subset X$ is a stable equilibrium of a process $p$ iff there exists a Liapunov function $V$ for $p$ and $X^*$.*
Proof: If there exists a Liapunov function $V$, $X^* \subset X$ is a stable equilibrium. First of all, $X^*$ is an equilibrium since $V$ takes the unique minimum at $X^*$. Suppose $\Omega$ is the domain of $V$. Given any $\epsilon$, let $\epsilon' \leq \epsilon$ such that $N^{\epsilon'}(X^*) \subseteq \Omega$. Let $\gamma$ be the minimum over the boundary of $N^{\epsilon'}(X^*)$; $\gamma > V(X^*)$ since $X^*$ is the unique minimum. Because $V$ is continuous, there exists a $\delta$-neighborhood $N^\delta(X^*)$ such that $\forall x \in N^\delta(X^*), V(x) < \gamma$. Therefore, $\phi_p(N^\delta(X^*)) \subseteq N^{\epsilon'}(X^*) \subseteq N^\epsilon(X^*)$.

If $X^* \subset X$ is a stable equilibrium of a process $p$, let $V(x) = \sup_{x' \in \phi_p(x)}\{d(x', X^*)\}$. We have (1) $V(X^*) = 0$ since $X^*$ is an equilibrium, (2) $V(p(x)(t)) \leq V(x)$ since $\phi_p(p(x)(t)) \subseteq \phi_p(x)$, and (3) $V$ is continuous since $X^*$ is stable. □

**Theorem 14.1.2** *$X^* \subset X$ is an asymptotically stable equilibrium of a process $p$ iff there exists a Liapunov function $V : \Omega \to \mathcal{R}$ for $p$ and $X^*$, such that $\forall x \in \Omega, \lim_{t\to\infty} V(p(x)(t)) = V(X^*)$. Furthermore, if $\Omega = X$, $X^*$ is an asymptotically stable equilibrium in the large.*
Proof: Since $X^*$ is the unique minimum in $\Omega$, $p(x)$ approaches $X^*, \forall x \in \Omega$. Given $V$ defined as the same as that in the previous proof, if $X^*$ is an asymptotically stable equilibrium, $V(p(x)(t))$ approaches $V(X^*)$. □

**Proposition 14.2.1** *If a constraint solver $CS^V$ solves a set of constraints $C$ on variables $V$ globally, every equilibrium of $[\![CS^V]\!]$ is a solution of $C$.*
Proof: Trivial. □

**Proposition 14.2.2** *If $V : \Omega \to \mathcal{R}$ is a Liapunov function for $\langle \mathcal{S}, f\rangle$ and $\mathcal{S}^* = \{s^* | s^* = f(s^*)\} \subset \Omega$, then $V(f(x)) \le V(x), \forall x \in \Omega$. In addition, if $f$ is continuous and $V(f(x)) < V(x), \forall x \notin \mathcal{S}^*$, $\mathcal{S}^*$ is an asymptotically stable equilibrium.*
Proof: If $\lim_{n\to\infty} V(f^n(s)) = \epsilon > V(S^*)$, let $X = \{s | V(s) \le \epsilon\} \supset S^*$, $f^n(s)$ approaches $X$. If $f$ is continuous, however, $f^n(s)$ approaches $f(X) \subset X$ and $\lim_{n\to\infty} V(f^n(s)) < \epsilon$, contradiction. □

**Proposition 14.2.3** *A set $\mathcal{S}^* = \{s^* | f(s^*) = 0\} \subset \Omega$ is an asymptotically stable equilibrium of a state integration system if $f$ is continuous at $\mathcal{S}^*$ and $\mathcal{S}^*$ is the unique minimum of $-\int f(s)ds$ in $\Omega$. If $\Omega = \mathcal{S}$, $\mathcal{S}^*$ is an asymptotically stable equilibrium in the large.*
Proof: Let $V(s) = -\int f(s)ds$ be defined on a neighborhood of $S^*$. $V$ is a Liapunov function for $\dot{s} = f(s)$ and $S^*$ since $\dot{V}(s) = -f^2(s) \le 0$. Furthermore, $\dot{V}(s) < 0, \forall s \notin S^*$ since $f(s) \ne 0$. □

**Proposition 14.3.1** *Let $R \subset \mathcal{R}^n$ be closed and convex. The projection $P_R(x)$ of $x$ to $R$ exists and is unique for every $x$, and $(x - P_R(x))^T(y - P_R(x)) \le 0$ for any $y \in R$.*
Proof: Refer to [GPR67]. □

**Theorem 14.3.1** *PM solves $\{X_i\}_{i\in I}$ globally if all the $X_i$'s are convex.*
Proof: Let $X^* = \cap_I X_i$ be the solution set of the problem. First of all, it is easy to see that if $x^* \in X^*$ is a solution, then $x^* = f(x^*)$, i.e., $x^*$ is an equilibrium. Moreover, we can prove that

$|f(x) - x^*| \leq |x - x^*|$ for any $x$ and $x^* \in X^*$ as follows.

$$\begin{aligned} |f(x) - x^*|^2 &= |x + \lambda(P(x) - x) - x^*|^2 \\ &= |x - x^*|^2 + \lambda^2 |P(x) - x|^2 + 2\lambda(x - x^*)^T(P(x) - x) \\ &= |x - x^*|^2 + (\lambda^2 - 2\lambda)|P(x) - x|^2 + 2\lambda(P(x) - x)^T(P(x) - x^*) \\ &\leq |x - x^*|^2 - \lambda(2 - \lambda)|P(x) - x|^2 \quad \text{according to Proposition 14.3.1} \\ &\leq |x - x^*|^2 \quad \text{since } 0 < \lambda < 2. \end{aligned}$$

Therefore, let $V(x) = d(x, X^*)$, we have $V(f(x)) \leq V(x)$. Thus, $X^*$ is stable.

Furthermore, $|f^k(x) - x^*|$ is nonincreasing and bounded below. Therefore, $|f^k(x) - x^*|$ has a limit and $\max_I d(f^k(x), X_i)$ approaches 0. According to [GPR67], $\lim_{k\to\infty} d(f^k(x), X^*) = 0$, since $\mathcal{R}^n$ is finite dimensional. As a result, $\lim_{k\to\infty} V(f^k(x)) = 0 = V(X^*)$. Thus, $X^*$ is an asymptotically stable equilibrium of PM in the large, i.e., PM solves the problem globally. □

**Theorem 14.3.2** *Let $X^* \in \mathcal{R}^n$ be the set of local minima of $\mathcal{E}$. NM solves the problem if $|J(x^*)| \neq 0$, $\forall x^* \in X^*$. i.e., $\mathcal{E}$ is strictly convex at each local minimal point. NM solves the problem globally if, in addition, $\mathcal{E}$ is convex.*

Proof: First, we prove that $\forall x^* \in X^*$, $x^* = f(x^*)$ and $|J(x^*)| \neq 0$ implies that $x^*$ is asymptotically stable. Let $R$ be the Jacobian of $f$. It is easy to check that $|R(x^*)| = 0$. There exists a neighborhood of $x^*$, $N^\epsilon(x^*)$, for any $x \in N^\epsilon(x^*)$, $|f(x) - f(x^*)| \leq \lambda|x - x^*|$ for $0 < \lambda < 1$. Therefore, $\lim_{k\to\infty} |f^k(x) - x^*| = 0$ and $x^*$ is asymptotically stable. Therefore, $X^*$ is an asymptotically stable equilibrium. If $\mathcal{E}$ is convex, $x^*$ is the unique minimal point, which is an attractor in the large. □

**Theorem 14.3.3** *Let $X^*$ be the set of local minima of $\mathcal{E}$. GM solves the problem if $\frac{\partial \mathcal{E}}{\partial x}$ is continuous at $X^*$. GM solves the problem globally if, in addition, $\mathcal{E}$ is convex.*

Proof: According to Proposition 14.2.3, a local minimum is an asymptotically stable equilibrium. A set of local minima is also an asymptotically stable equilibrium. If $\mathcal{E}$ is convex, $X^*$ is the unique minimal set, which is an attractor in the large. □

**Theorem 14.3.4** *Let $A$ be a matrix where $A_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} + \Sigma_{k=0}^m \lambda_k \frac{\partial^2 g_k}{\partial x_i \partial x_j}$. If $A$ is positive definite, LM solves the constrained optimization problem* $\min f(x)$ *subject to $g_k(x) = 0$ globally.*

Proof: Let

$$V(x) = \frac{1}{2}\Sigma_i \dot{x}_i^2 + \frac{1}{2}\Sigma_k g_k^2(x).$$

It has been shown in [Pla89] that

$$\dot{V} = -\Sigma_{i,j}\dot{x}_i A_{ij}\dot{x}_j.$$

Therefore, $V$ is a Liapunov function. □

**Proposition 14.5.1** *A constraint solver $CS^V$ solves $C$ iff there exists an initial condition $\Theta \supset sol(C)$ such that $\forall \epsilon > 0$, $[\![CS^V(\Theta)]\!] \models \mathcal{A}(C^\epsilon; \Box)$. $CS$ solves $C$ globally when $\Theta = \times_V D_v$.*

Proof: According to the definition of constraint solvers, $CS^V$ solves $C$, iff $[\![CS^V]\!]$ is asymptotically stable at $sol(C)$, i.e., $\exists\Theta \supset sol(C)$, $\forall x \in \Theta$, $[\![CS^V]\!](x)$ approaches $sol(C)$ asymptotically. In other word, for any $\epsilon$, $\exists t_0$, $\forall t \geq t_0$, $[\![CS^V]\!](x)(t) \in C^\epsilon$. Therefore, $[\![CS^V]\!](x) \in \mathcal{A}(C^\epsilon; \Box)$ for all $x \in \Theta$.

On the other hand, if $[\![CS^V]\!](x) \in \mathcal{A}(C^\epsilon; \Box)$ for any $\epsilon > 0$, $[\![CS^V]\!](x)$ approaches $sol(C)$ asymptotically. Therefore, $CS^V$ solves $C$. □

## A.7 Control Synthesis

**Proposition 15.3.1** *This control law satisfies the condition that $v = 0$ iff*

$$(d = 0 \vee |\theta' - \theta| = \frac{\pi}{2}k) \bigwedge (\theta_d = \theta).$$

Proof: According to the control law for $\alpha$, $v = 0$ implies $\theta_d = \theta$. According to the control law for $v$, $v = 0$ implies $d\cos(\theta' - \theta) = 0$. □

# Appendix B

# ALERT

We have developed a visual programming and simulation environment called ALERT ( A Laboratory for Embedded Real-Time Systems) based on the Constraint Net model. In this appendix, we first describe the current version of ALERT, then give some simple examples to illustrate the process of analysis.

## B.1 Visual Programming with Constraint Nets

*Visual Programming* means the use of meaningful graphic representations in the process of programming [Shu88]. Visual programming has gained momentum in recent years primarily because the falling cost of graphical-related hardware and software has made it feasible to use pictures as a means of communicating with computers. CN has inherent graphical tokens and the characteristics of hierarchy, which make it an ideal model for visual programming.

CN is a generalization of models for dynamic systems. As a first step, we have developed ALERT on Simulink [Incc]. Simulink, based on Matlab, is a visual programming and simulation environment for both continuous and discrete dynamic systems.

Each Simulink window consists of five pop-up menus: `File` ( open and save files), `Edit` (cut, copy and paste graphical tokens), `Options` (group, mask, flip or rotate modules), `Simulation` (start, pause, and parameters for simulations) and `Style` (color, font and position).

Simulink provides various built-in modules such as linear and nonlinear transductions. In addition, it provides test signals, output viewing windows, and various signal analysis tools.

Programming in Simulink is simply by choosing a set of modules from the given libraries, setting up parameters and making connections. A system can be developed hierarchically by group and mask operations. On the other hand, a module can be opened by an unmask operation and then be modified accordingly.

Even though Simulink supports the integration of discrete and continuous modeling, the internal semantics is different from that of CN. Instead of holding values between sampling points (as does the semantics of Constraint Nets), Simulink assumes linear interpolation. Furthermore, Simulink does not support event-driven transductions, which are the most important aspect of CN.

However, Simulink is a flexible open environment so that new modules can be added easily using Matlab functions and programs. We have extended Simulink with various event-driven transductions and event logics, as well as with various arbitrations. In particular, we have added four new libraries to Simulink (see Figure B.1); they are *logics*, *events*, *arbiters* and *solvers*.

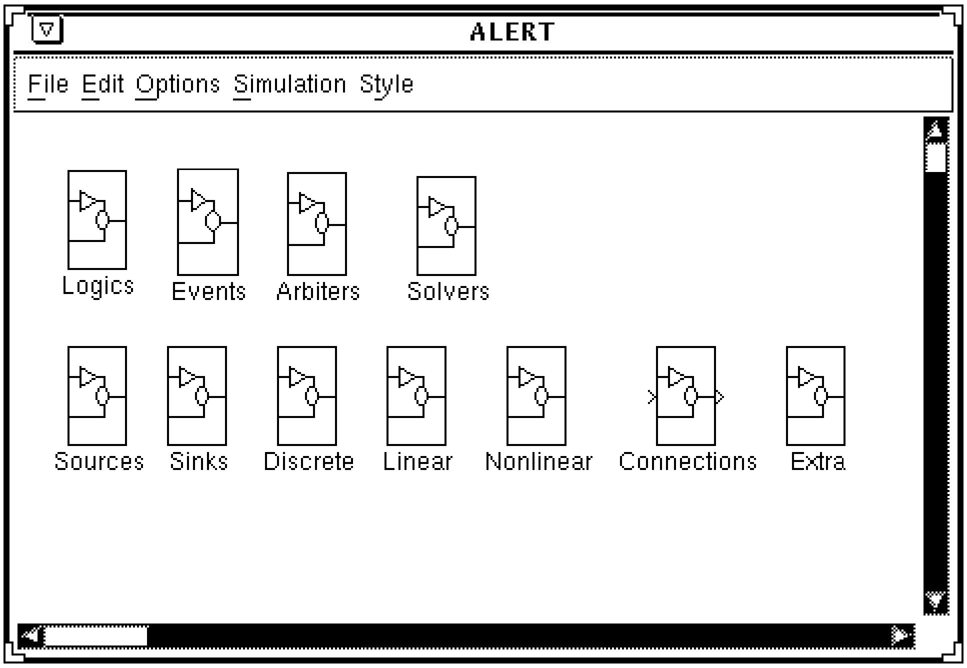


Figure B.1: ALERT

The basic functionalities of these new libraries are:

- *Logics*: This library (Figure B.2) includes various event logics, such as event synchronization elements, "flip-flop," etc.

- *Events*: This library (Figure B.3) includes an event generator and various event-driven transductions.

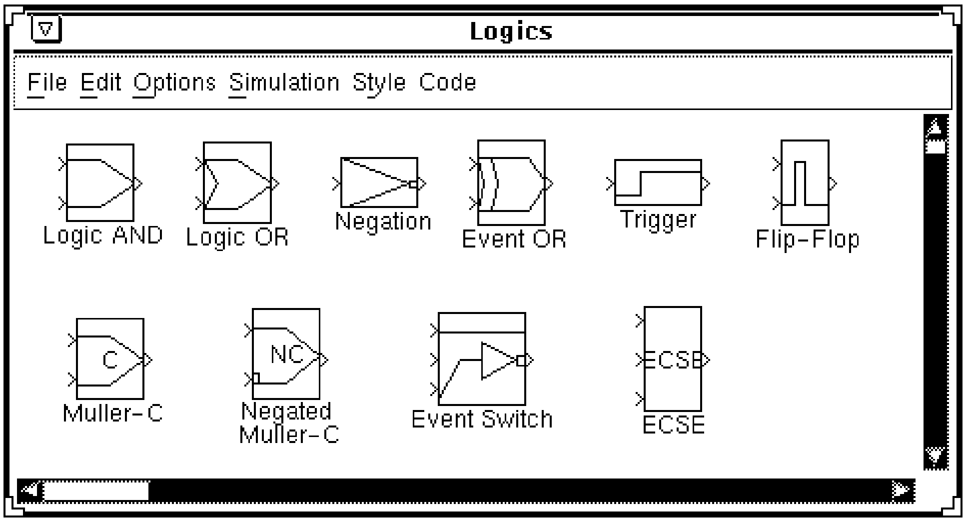


Figure B.2: Logic modules

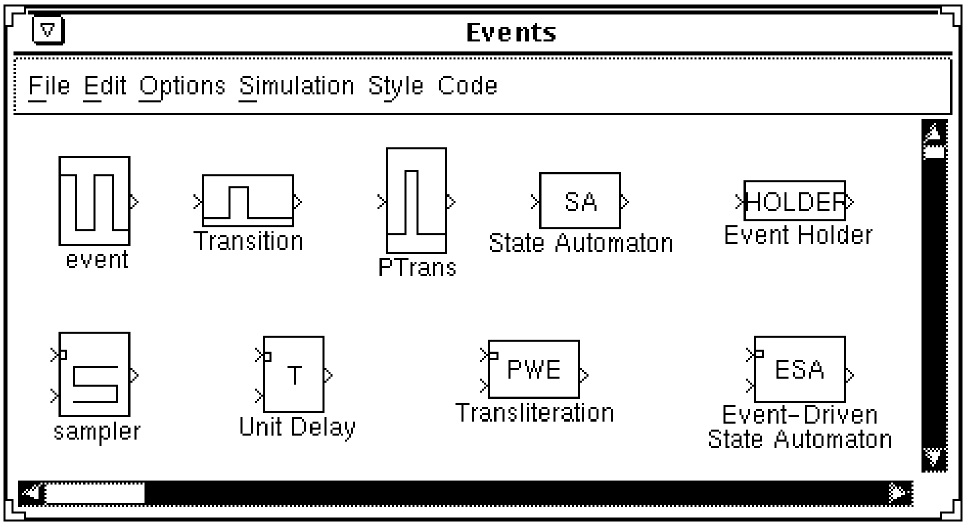


Figure B.3: Event modules

- *Arbiters*: This library includes various arbiters so that arbitration hierarchies can be constructed.
- *Solvers*: This library includes constraint solvers with various constraint methods (where constraints can be given by functions defined in Matlab).

## B.2 Simulation and Animation

A robotic system is a complex dynamic system in general; it is nonlinear in the following sense:

- the dynamics of the plant or the environment is nonlinear for any realistic modeling,
- the control is nonlinear if we model event-driven transductions or arbitration hierarchies.

For a nonlinear system, the behavior of the system is unpredictable in general, and parameters of the system (e.g., latencies and sampling rates) play an important role in the overall behaviors.

ALERT is an integrated environment for modeling, programming and analyzing robotic systems. Such an environment is important for building a system with a certain degree of "correctness." Even though a real system's behavior can not be guaranteed in advance, the more accurate the model is, the more information can be obtained in the simulation. On the other hand, the more robust the control is, the more relaxed the accuracy of the model can be.

ALERT provides an environment for simulation that, in general, is the only approach to analyzing nonlinear dynamic systems. Visualization can be added to the current version of ALERT, using Matlab plot functions. Animation can be done either on-line in Simulink, which is slow, or by saving the traces and down-loading to an SGI machine.

Now we present two simple examples to illustrate the use of ALERT.

In the first example, we analyze the effect of latencies on stability (Figure B.4). The solution of $\dot{x} = -kx$ is $x(t) = x_0 e^{-kt}$, which is asymptotically stable at state 0. If we assume latency $\delta$ for signal $x$, the solution of $\dot{x} = -k(x - \delta)$ is not trivial, and it may become unstable at 0. For this simple equation, we are able to analyze the solution by hand [Hub88]. Let $e^{-\lambda t}$ be a solution. We have $-\lambda e^{-\lambda t} = -ke^{-\lambda(t-\delta)}$, i.e., $\lambda = ke^{\lambda\delta}$. Since $\min\{\frac{ke^{\lambda\delta}}{\lambda}\} = \delta ke$, for any real number $\lambda$, we have $\delta ke \leq 1$, i.e., $\delta k \leq 1/e$. If $\delta k > 1/e$, $\lambda$ must be a complex number, and therefore the solution has oscillation. In general, for a stable system, if latency is introduced, it may become unstable (Figure B.5, B.6).

In the second example, we show that the sampling of data can cause unstability too (Figure B.7). For the same system, let the sampling rate be $\delta$. We have $\dot{x} = -k\overline{u}$ where $u(\delta n) = x(\delta n)$ for any integer $n$. The solution is not stable if $|1 - \delta k| > 1$, i.e., $\delta k > 2$ (Figure B.8, B.9).

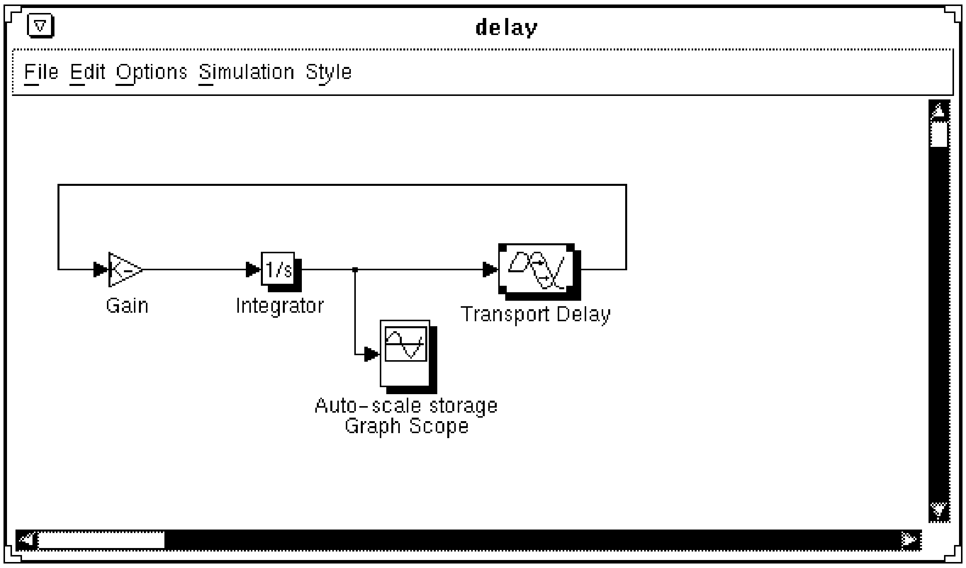


Figure B.4: Circuit with latency

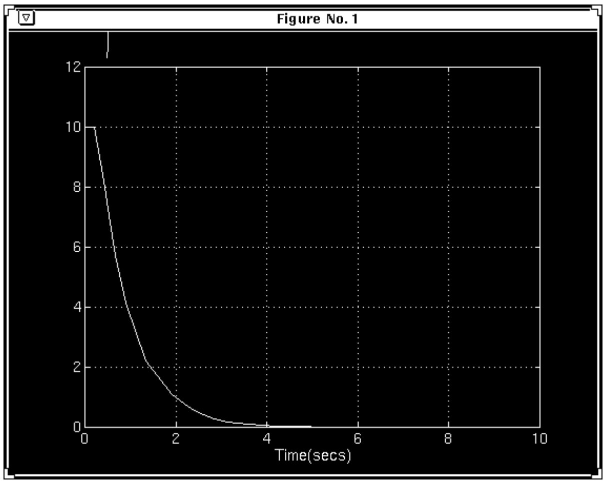


Figure B.5: Latency with $\delta k = 0.25$

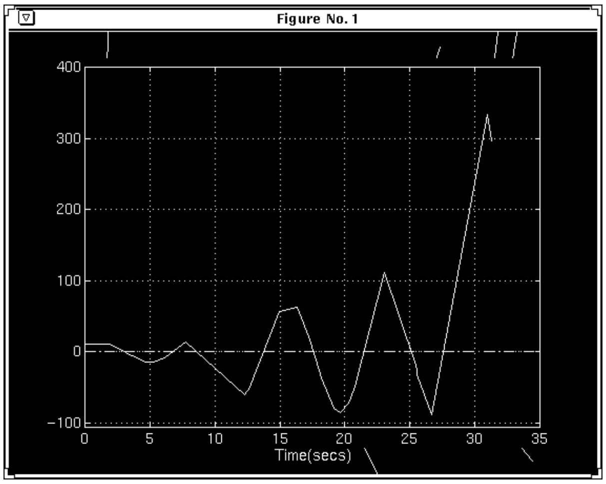


Figure B.6: Latency with $\delta k = 2$

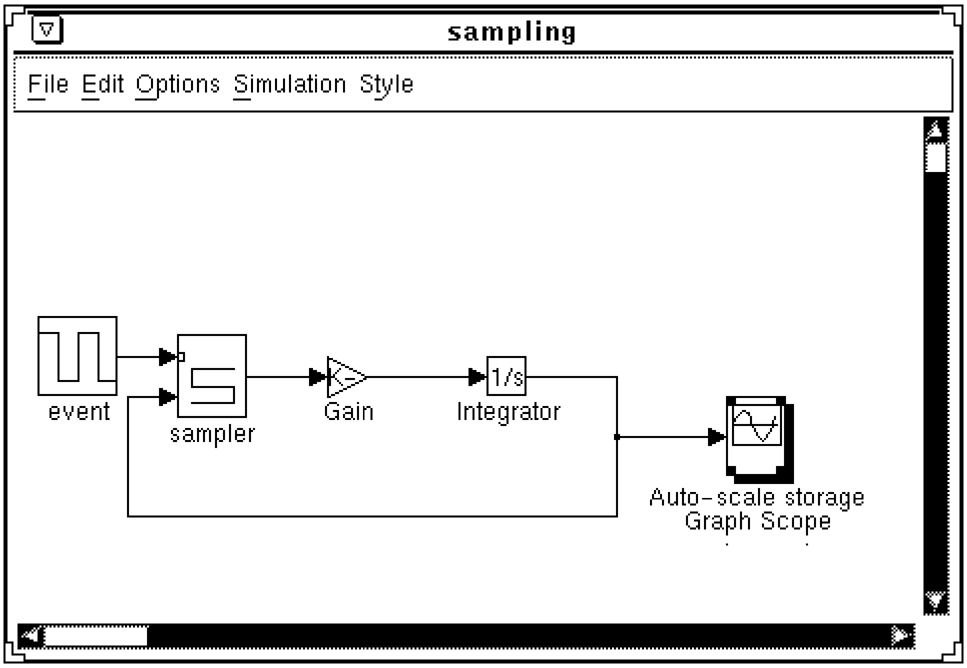


Figure B.7: Circuit with sampling

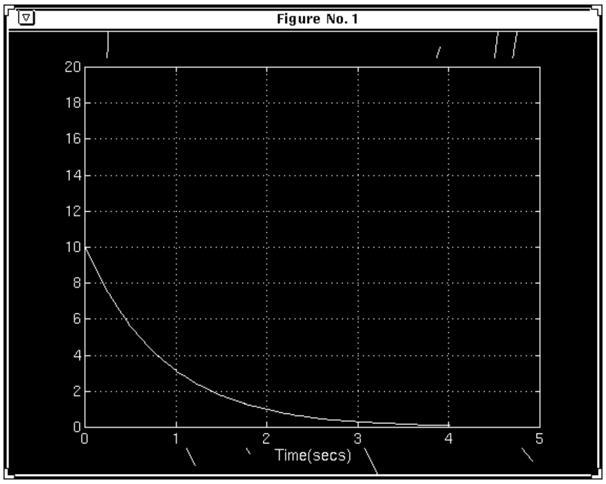


Figure B.8: Sampling with $\delta k = 0.25$

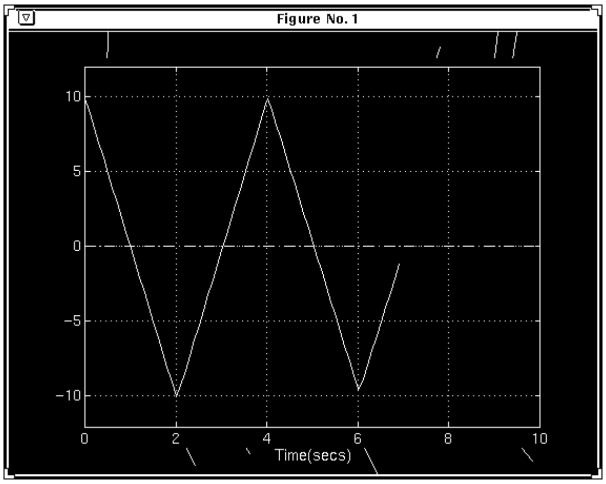


Figure B.9: Sampling with $\delta k = 2$

In general, parameters like $k$ and $\delta$ play important roles in control systems design: $k$ is the parameter for the speed control, and $\delta$ is introduced by unavoidable computation and device latency, or the digital sampling rate. For instance, if $\delta$ is known, we may choose $k$ to achieve fast convergence yet maintaining stability.

## B.3 The Maze Traveler

We conclude this appendix with the maze traveler example.

Figure B.10 depicts the overall structure of the system. Figure B.11 shows the animation window. The model and the controller of the car are given in Figures B.12 and B.13, respectively. The event generator is depicted in Figure B.14.

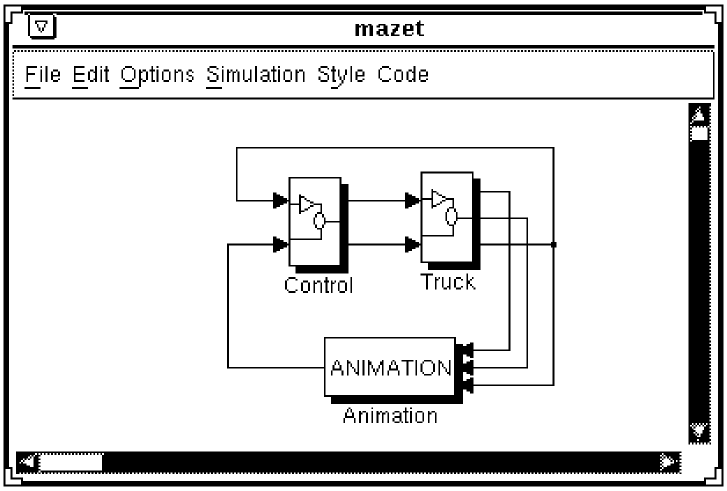


Figure B.10: The overall structure of the maze traveler system

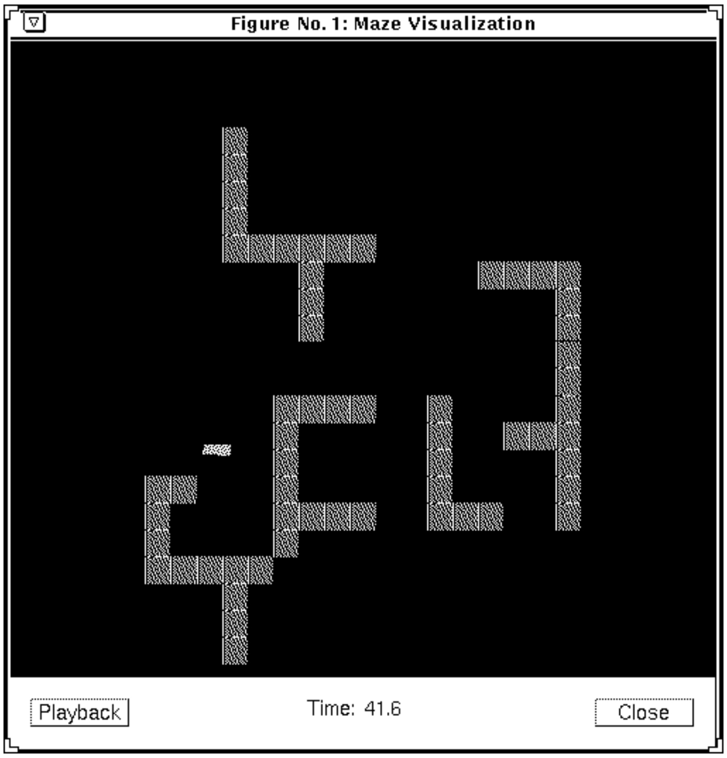


Figure B.11: Animation of the maze traveler

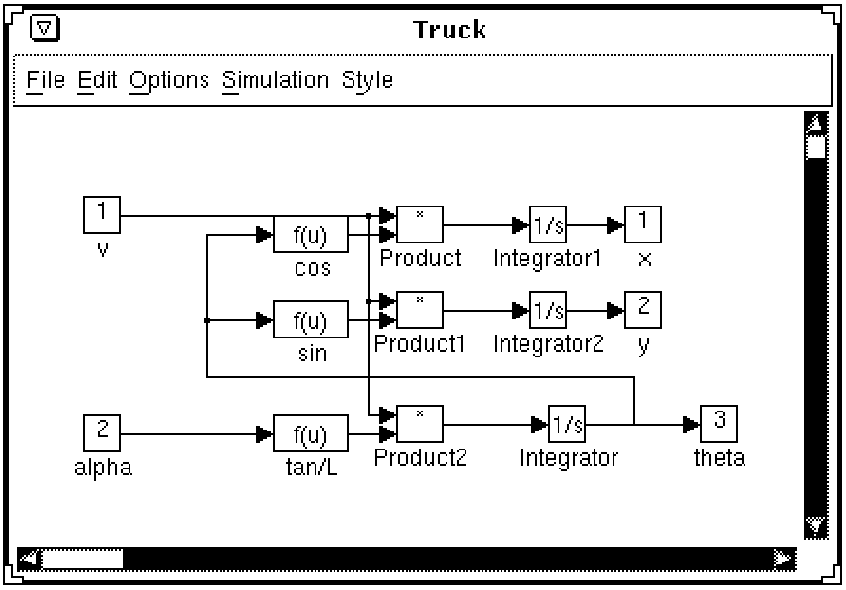


Figure B.12: The car model

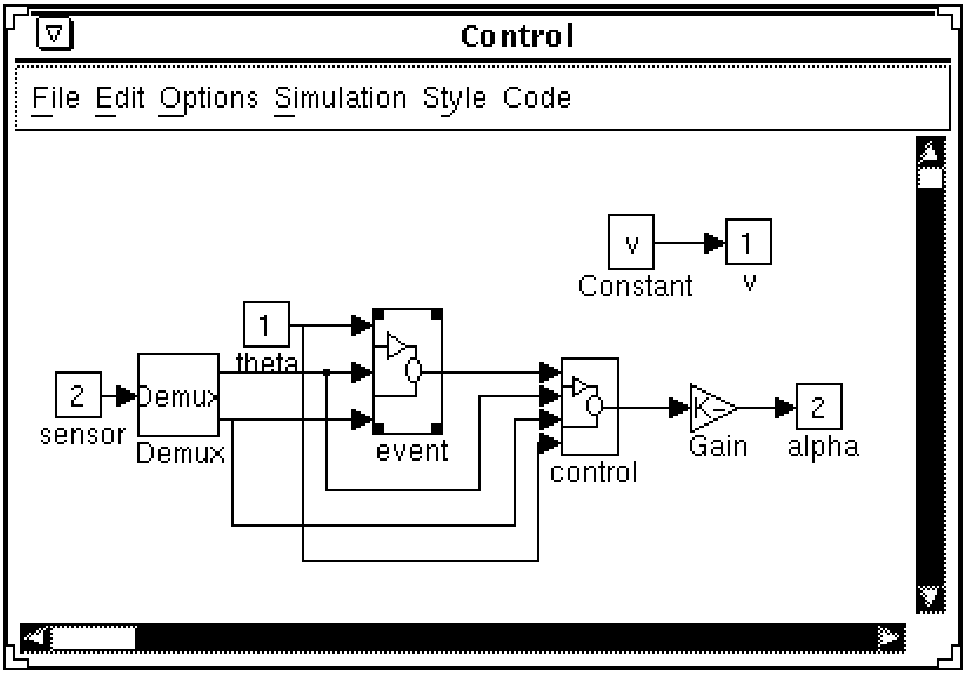


Figure B.13: The control module

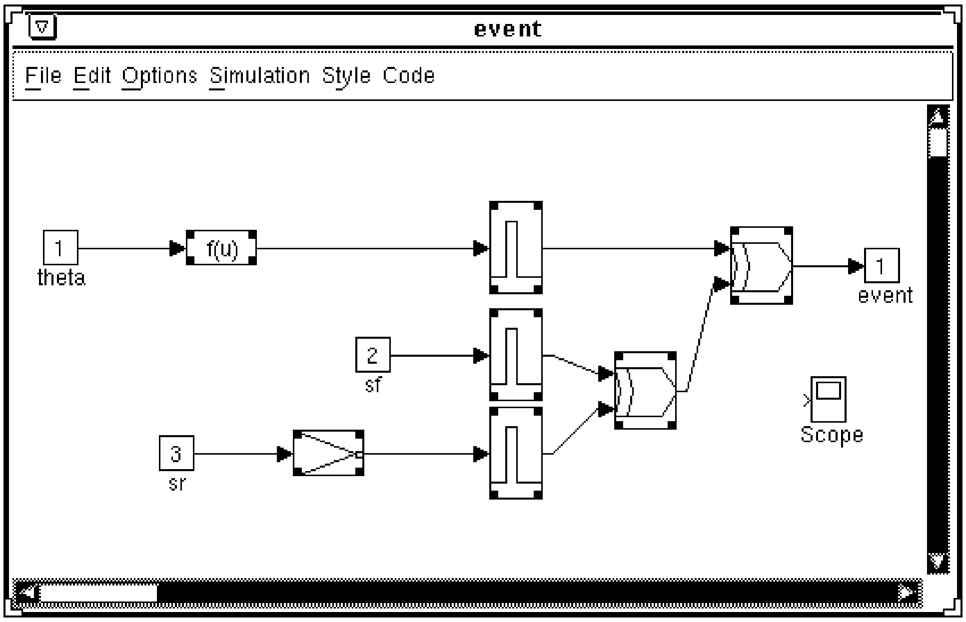


Figure B.14: The event module

# Appendix C

# Examples of Design and Analysis

We present in this appendix two complete examples of the design and analysis of robotic systems and behaviors. One is an hydraulically actuated robot arm and the other is an elevator system.

## C.1 Modeling and Control of an Hydraulically Actuated Arm

Figure C.1 depicts a two-link robot arm. For simplicity, we assume that the mass distribution of the two-link arm is extremely simple: All mass exists as a point mass at the distal end of each link.

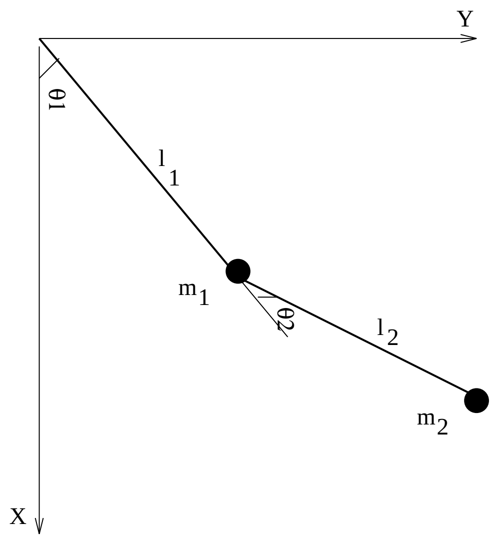


Figure C.1: A two-link arm

The dynamics of the arm is modeled by the following equations [Cra86]:

$$
\begin{aligned}
\tau_1 &= [(m_1+m_2)l_1^2 + m_2 l_2^2 + 2m_2 l_1 l_2 \cos(\theta_2)]\ddot{\theta}_1 + [m_2 l_2^2 + m_2 l_1 l_2 \cos(\theta_2)]\ddot{\theta}_2 \\
&\quad -2m_2 l_1 l_2 \sin(\theta_2)\dot{\theta}_1\dot{\theta}_2 - m_2 l_1 l_2 \sin(\theta_2){\dot{\theta}_2}^2 + (m_1+m_2) g l_1 \sin(\theta_1) + m_2 g l_2 \sin(\theta_1+\theta_2), \\
\tau_2 &= [m_2 l_2^2 + m_2 l_1 l_2 \cos(\theta_2)]\ddot{\theta}_1 + m_2 l_2^2 \ddot{\theta}_2 + m_2 l_1 l_2 \sin(\theta_2){\dot{\theta}_1}^2 + m_2 g l_2 \sin(\theta_1+\theta_2).
\end{aligned}
$$

For simplicity, we further assume $m_1 = m_2 = m$ and $l_1 = l_2 = l$. Let $d_1 = \dot{\theta}_1$ and $d_2 = \dot{\theta}_2$, the arm model is a set of equations with state variables $\theta_1$, $\theta_2$, $d_1$ and $d_2$:

$$
\begin{aligned}
x &= [\tau_1 + ml^2 \sin(\theta_2)\dot{\theta}_2^2 + 2ml^2 \sin(\theta_2)\dot{\theta}_1\dot{\theta}_2 - 2mlg\sin(\theta_1) - mlg\sin(\theta_1+\theta_2) \\
&\quad -(1+\cos(\theta_2))(\tau_2 - ml^2\sin(\theta_2)\dot{\theta}_1^2 - mlg\sin(\theta_1+\theta_2))]/(1+\sin^2(\theta_2)) \\
\dot{d}_1 &= x/ml^2 \\
\dot{d}_2 &= [\tau_2 - ml^2\sin(\theta_2)\dot{\theta}_1^2 - mlg\sin(\theta_1+\theta_2) - (1+\cos(\theta_2))x]/ml^2 \\
\dot{\theta}_1 &= d_1 \\
\dot{\theta}_2 &= d_2
\end{aligned}
$$

where $m$ and $l$ are parameters.

The joints of the arm are actuated by hydraulic actuators [SDLS90]. Valves are devices that control the fluid power. The most widely used valve is the sliding valve with spool type construction. The inputs required to model such a valve are the spool displacement ($-0.5 \leq X_v \leq 0.5$), the supply pressure ($P_{sup}$), the return presure ($P_{res}$) and the lines pressure ($P_{in}$ and $P_{out}$). The governing nonlinear equations are:

$$
Q_{in} = \begin{cases} K_v X_v \sqrt{P_{sup} - P_{in}} & \text{if } X_v > 0 \\ K_v X_v \sqrt{P_{in} - P_{res}} & \text{if } X_v < 0, \end{cases}
$$

$$
Q_{out} = \begin{cases} K_v X_v \sqrt{P_{sup} - P_{out}} & \text{if } X_v > 0 \\ K_v X_v \sqrt{P_{out} - P_{res}} & \text{if } X_v < 0, \end{cases}
$$

where $K_v$ is a parameter, and

$$
\dot{P}_{in} = \frac{\beta}{V}(Q_{in} - D_m\dot{\theta}),
$$

$$
\dot{P}_{Out} = \frac{\beta}{V}(D_m\dot{\theta} - Q_{out}),
$$

where $D_m$ is the volumetric displacement of the hydraulic motor and $\frac{V}{\beta}$ is the hydraulic compliance. The torque generated by the controller is:

$$
\tau = D_m(P_{in} - P_{out}).
$$

Assume that the low level controller for a hydraulically actuated joint is a simple PD control that produces a spool displacement $X_v$ given $\theta$, $\dot{\theta}$ and $\theta_d$:

$$X_v = B[(\theta_d - \theta) - A\dot{\theta}]$$

We select $A$ and $B$ by experiment, given the set of other parameters.

After we get a stable PD controller for joint tracking, a high level controller for end-point tracking is then developed as follows. Let $\langle x, y\rangle$ be the coordinate of the end-point of the arm. The constraints for the end-point tracking are $x = x_d$ and $y = y_d$ where $\langle x_d, y_d\rangle$ is the desired position. Let $\mathcal{E} = \frac{1}{2}(x_d - x)^2 + \frac{1}{2}(y_d - y)^2$ be the energy function. We have

$$\begin{aligned} -\frac{\partial \mathcal{E}}{\partial \theta_1} &= (x_d - x)\frac{\partial x}{\partial \theta_1} + (y_d - y)\frac{\partial y}{\partial \theta_1} \\ -\frac{\partial \mathcal{E}}{\partial \theta_2} &= (x_d - x)\frac{\partial x}{\partial \theta_2} + (y_d - y)\frac{\partial y}{\partial \theta_2} \end{aligned}$$

where

$$\begin{aligned} x &= l\cos(\theta_1) + l\cos(\theta_1 + \theta_2) \\ y &= l\sin(\theta_1) + l\sin(\theta_1 + \theta_2) \\ \frac{\partial x}{\partial \theta_1} &= -l\sin(\theta_1) - l\sin(\theta_1 + \theta_2) \\ \frac{\partial y}{\partial \theta_1} &= l\cos(\theta_1) + l\cos(\theta_1 + \theta_2) \\ \frac{\partial x}{\partial \theta_2} &= -l\sin(\theta_1 + \theta_2) \\ \frac{\partial y}{\partial \theta_2} &= l\cos(\theta_1 + \theta_2) \end{aligned}$$

Using the gradient method, we have:

$$\dot{\theta}_d = -k\frac{\partial \mathcal{E}}{\partial \theta}.$$

Then we use $\theta_d$ as the input to the low level PD controller. We can consider this end-point tracking controller as a variation of the transpose Jacobian controller [Cra86].

Similarly, a high level controller for avoiding obstacles is developed as follows. Let $\langle x_o, y_o\rangle$ be the coordinate of the obstacle and $\mathcal{E}(d) = \max(-\frac{1}{2}\ln(d^2/m^2), 0)$ where $m$ is the minimum distance between the obstacle and the arm. Let the energy function for avoiding the obstacle be:

$$\mathcal{E} = \mathcal{E}(d_{j1}) + \mathcal{E}(d_{l1}) + \mathcal{E}(d_{j2}) + \mathcal{E}(d_{l2})$$

where

$$
\begin{aligned}
d_{j1}^2 &= (x_o - l\cos(\theta_1))^2 + (y_o - l\sin(\theta_1))^2 \\
|d_{l1}| &= |y_o\cos(\theta_1) - x_o\sin(\theta_1)| \\
d_{j2}^2 &= (x_o - l\cos(\theta_1) - l\cos(\theta_1+\theta_2))^2 + (y_o - l\sin(\theta_1) - l\sin(\theta_1+\theta_2))^2 \\
|d_{l2}| &= |(y_o - l\sin(\theta_1))\cos(\theta_1+\theta_2) - (x_o - l\cos(\theta_1))\sin(\theta_1+\theta_2)|
\end{aligned}
$$

The obstacle avoiding controller is then designed using the gradient method.

We can combine these two high level controllers with some arbiters, such as the subsume function, to make the obstacle avoiding control have a higher priority.

The models of the high level controllers and the PD controller as well as the models of the arm and the hydraulic actuator are all developed in ALERT; both simulation and animation are supported.

## C.2 Modeling and Verification of an Elevator System

A simple elevator system for an $n$-floor building consists of one elevator. Inside the elevator there is a board with $n$ *floor buttons*, each associated with one floor. Outside the elevator there are two *direction buttons* for service call on each floor, except the first floor and the top floor where only one button is needed (see Figure C.2). Any button can be pushed at any

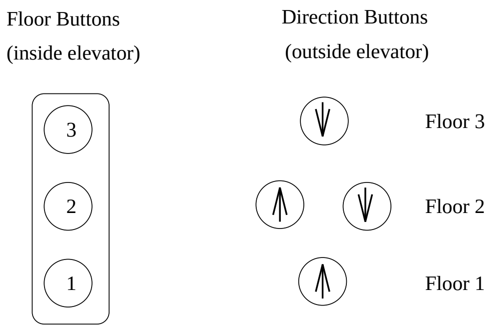


Figure C.2: The interface of a simple 3-floor elevator

time. After being pushed, a floor button will be on until the elevator stops at the floor, and a direction button will be on until the elevator stops at the floor and is going to move at the same direction. (Note that a more complex elevator has open and close door buttons, alarm or emergency buttons which, for simplicity, we will not model.) The atomic actions of an elevator consist of move-up or move-down one floor, serve-a-floor (stop at the floor, open and close the door) and stay-idle. The complete elevator system consists of ELEVATOR BODY, ELEVATOR CONTROL and USER INTERFACE as shown in Figure C.3.

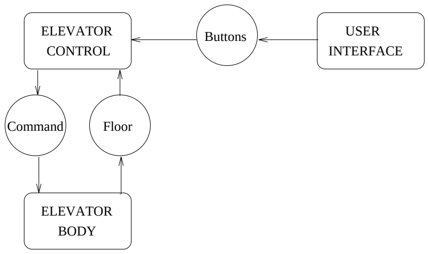


Figure C.3: The complete elevator system

### C.2.1 Discrete modeling and verification

First we present a discrete model of the elevator system, in which each atomic action takes some finite time.

The elevator body is modeled by a transliteration and a unit delay:

$$\begin{aligned} nf &= \begin{cases} \min(f+1, n) & \text{if } cc = up \\ \max(f-1, 1) & \text{if } cc = down \\ f & \text{otherwise.} \end{cases} \\ f' &= nf \end{aligned}$$

where $cc$ is the current command from the controller with domain $\{up, down, serve, idle\}$, and $f, nf$ are the current and next floor numbers, respectively, with domain $\{1, 2, \ldots, n\}$.

The command from the controller is modeled as a function of the current floor number, the current request state and the last control state. Let the request state be a tuple $\langle ub, db, fb\rangle$ where $ub, db, fb \in \{0,1\}^n$ with $ub(n) = 0$ and $db(1) = 0$; let the last control state be $ls$ with domain $\{up, down, idle\}$. Let $ur, dr \in \{0,1\}$ denote the up and down requests, respectively, i.e.,

- $ur$ indicates whether or not there is a request for the elevator to go up:

$$ur = ub(f) \bigvee_{i>f} (ub(i) \vee db(i) \vee fb(i)).$$

- $dr$ indicates whether or not there is a request for the elevator to go down:

$$dr = db(f) \bigvee_{i<f} (ub(i) \vee db(i) \vee fb(i)).$$

The current control state $cs$ is determined as follows:

$$\begin{aligned} cs &= \begin{cases} up & \text{if } ur \wedge (ls \neq down \vee \neg dr) \\ down & \text{if } (\neg ur \wedge f > 1) \vee (dr \wedge ls = down) \\ idle & \text{otherwise} \end{cases} \\ ls' &= cs. \end{aligned}$$

In English, if there is a request for the elevator to go up and either the last state is up or there is no request to go down, the elevator will be in the up state; if there is no request to go up and the elevator is not at the first floor, or the last state is down and there is a request to go down, then the elevator will be in the down state; otherwise the elevator will be idle, that is, the elevator will be parked at the first floor if there are no more requests.

Let $cr$ indicate whether or not there is a request for the elevator to stop and serve the current floor:

$$cr = \begin{cases} db(f) \vee fb(f) & \text{if } cs = down \\ ub(f) \vee fb(f) & \text{otherwise.} \end{cases}$$

In English, if there is an internal request to arrive at this floor or there is an external request to go in the same direction as the elevator, there is a request at this floor.

The current command can be defined as follows:

$$cc = \begin{cases} serve & \text{if } cr \\ cs & \text{otherwise.} \end{cases}$$

In English, if there is a request at this floor, the elevator will stop to serve the floor (open the door, let passengers go in and out, then close the door), otherwise the elevator will pass this floor without stopping.

Furthermore, the request state $\langle ub, db, fb \rangle$ is determined based on two factors: the user's input and the internal reset when a request has been served. Let $s$ denotes $u$, $d$ or $f$, we have

$$\begin{aligned} sb &= isb \vee (\neg rsb \wedge lsb) \\ lsb' &= sb \end{aligned}$$

where $isb$, $rsb$ and $lsb$ are the user's input, the reset and the last request state, respectively. The reset state $rsb$ indicates which requests have been served:

$$\begin{aligned} rsb' &= csb \\ cub(i) &= (f = i) \wedge (cc = serve) \wedge (cs = up) \\ cdb(i) &= (f = i) \wedge (cc = serve) \wedge (cs = down) \\ cfb(i) &= (f = i) \wedge (cc = serve) \end{aligned}$$

We have implemented the discrete model of the elevator system in Strand88 [FT89], a concurrent logic programming language. It is easy to simulate discrete time constraint nets in Strand88, since both transliterations and unit delays can be represented:

```
%f(+in, -out) is a function. fT(+in_trace, -out_trace) is a transliteration.
fT([I|Is], OS) :- f(I, O), OS := [O|Os], fT(Is, Os).
%delay(+init, +in_trace, -out_trace)
delay(Init, In, Out) :- Out := [Init|In].
```

where a trace is represented as an infinite list.

A well-designed elevator system should guarantee that any request will be served within some bounded time. We can specify such requirements in timed $\forall$-automata, and show that the constraint net model of the elevator system satisfies the timed $\forall$-automaton specification.

There are three kinds of request: to go to a particular floor after entering the elevator, or to go up or down when waiting for the elevator. Following are some examples of the state propositions.

- $R2 : (fb(2) = 1) \wedge (cfb(2) = 0)$ denotes that "there is a request to go to the second floor."
- $R2S : cfb(2) = 1$ denotes that "the request to go to the second floor is served."
- $RU2 : (ub(2) = 1) \wedge (cub(2) = 0)$ denotes that "there is a request to go up at the second floor."
- $RU2S : cub(2) = 1$ denotes that "the request to go up at the second floor is served."

Bounded time responses "the request to go to the second floor will be served in finite time" and "the request to go up at the second floor will be served in finite time" are represented as Figure C.4 (a) and (b), respectively.

Let 7 be associated with $q_0$ in Figure C.4 (a) and 11 be associated with $q_0$ in Figure C.4 (b). The two timed $\forall$-automata specify the properties: "the request to go to the second floor will be served within 7 time units" and "the request to go up to the second floor will be served within 11 time units," respectively.

These specifications can be checked using the verification algorithm. If $n = 4$, the state transition graphs of the elevator system with respect to the specifications in Figure C.4 (a) and (b) are shown in Figure C.5 (a) and (b), respectively, where the dotted transitions are disabled in our control strategy and the number associated with the state indicates the length of the

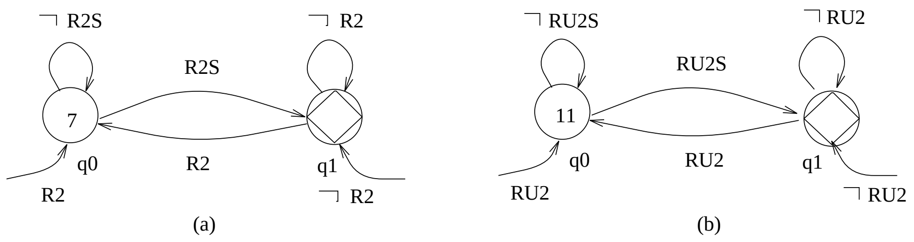


Figure C.4: Specifications of real-time response

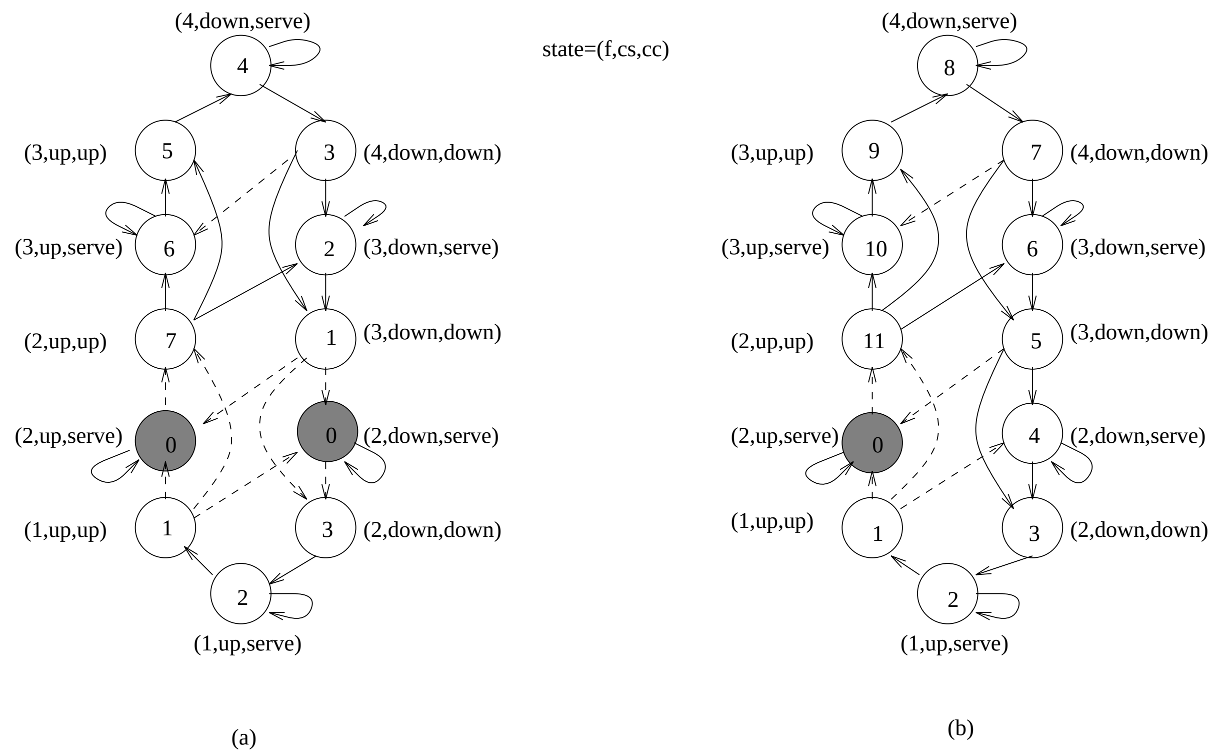


Figure C.5: State transition graphs

longest path from the state to the desired states, if there are no self-loop transitions. If the user is not allowed to issue a new request when the same request has just been served, these specifications will be satisfied. If, however, no such a restriction is imposed, an elevator may stop at a floor forever; therefore, these specifications will not be satisfied.

A more realistic specification for the elevator system is that any request should be served within bounded time of motion. Such a specification cannot be expressed by TLTL, however, it can be expressed by a timed $\forall$-automaton. For example, "the elevator will serve the second floor within 4 unit time of motion" can be depicted by the timed $\forall$-automaton in Figure C.6, with $MV : (cc \neq serve)$, $S2 : (f = 2) \wedge (cc = serve)$, $SN2 : (f \neq 2) \vee (cc = serve)$ and

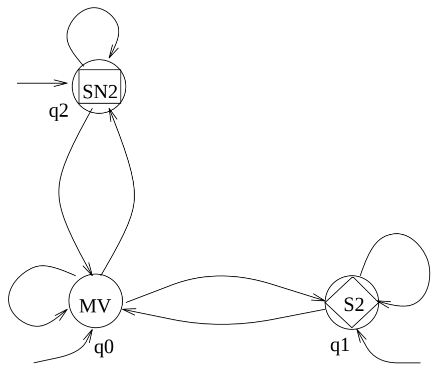


Figure C.6: A more realistic specification

$\tau(bad) = 4$. If $n = 4$, and $fb(2) = 1$ initially, the specification can be satisfied. This example has been verified by the verification procedure written in Prolog.

### C.2.2 Continuous modeling and verification

In the previous modeling of the elevator system, atomic actions are primitives. Now we shall model how these actions are carried out by the low level control system, which is realized as an analog controller. Furthermore, the user's request can come at any time on a continuous time line. A continuous model of the elevator system should be developed for the design of the low level control system and for the analysis of the overall behavior of the system.

First of all, the plant of the elevator is modeled by a second order differential equation following the Newton's Law

$$F - K\dot{h} = \ddot{h}$$

where $F$ is the motor force, $K$ is the friction coefficient and $h$ is the height of the elevator. We assume that the mass is 1 since it can be scaled by $F$ and $K$. We ignore gravity since we assume that it can be added to $F$ to compensate the effect.

A low level PD controller is then designed to produce the force to the elevator, given the action command (up, down or stop) and the height trace:

$$F = \begin{cases} F_0 & \text{if } up \\ -F_0 & \text{if } down \\ K_p d_s - K_v \dot{h} & \text{if } stop \end{cases}$$

where $d_s$ is the distance between the current height and the desired height of the elevator. Let the height of each floor be $H$ and the current floor of the elevator be $f$. We have $f = [h/H] + 1$ and $d_s = (f-1)H - h$ where $[x]$ indicates the closest integer of $x$.

We use the control strategy developed for the discrete model as a high level control. However, this control strategy is activated by events generated from the user interface or within the elevator itself. There are three basic types of event: (1) a user pushes a button at the elevator's idle state, (2) the elevator becomes close to a floor ($d_s \leq 15cm$, for instance) and (3) a user's request has been served (it takes $5s$ to serve a request, for instance). The "event or" of these three events triggers the high level controller to produce a new output. Furthermore, both user's requests and the internal reset are processed at a fast sampling rate ($0.1s$, for instance).

We have verified that the high level control strategy satisfies the desired properties. Now we have to guarantee that the low level control system does the right thing, i.e., accomplishes every goal that the high level strategy sets. Basically, we have to choose $F_0$, $K_p$ and $K_v$. Suppose that the friction coefficient $K$ is 1, the height of each floor is $2m$, and the elevator is said to be at a floor when $d_s \leq 15cm$. One basic request is that if a stop command is issued when the elevator is crossing a floor, the elevator will remain at the floor as long as the command does not change.

We choose $F_0$ to be 0.5 so that both the maximum velocity and acceleration will be 0.5, and it takes at least $4s$ to move up or down a floor. We choose $K_p = 0.5/0.15 = 3.3$ so that the initial acceleration for stop will be no larger than 0.5. Finally, we choose $K_v$ large enough so that the elevator will not over-shoot. In this case $K_v = 10$. We have modeled and simulated this complete elevator system in ALERT, and found that the system works correctly.

# Appendix D

# Model Estimation for the Car

We present here a method of model estimation for the car-like robot. We have modeled the plant of the car-like robot using the following set of equations:

$$\dot{x} = v\cos(\theta), \quad \dot{y} = v\sin(\theta), \quad \dot{\theta} = v\frac{\tan\alpha}{L}$$

where $\langle x, y, \theta \rangle$ is the configuration tuple of the car, $v$ and $\alpha$ are the control inputs to the car. However, for a real car, the velocity $v$ is controlled by the gas throttle $g_s$ and the turning angle $\alpha$ has its inherent mechanical delay. This two effects can be modeled by the following two equations:

$$\begin{aligned} \dot{v} &= \begin{cases} 0 & \text{if } g_s < g_m \text{ and } v = 0 \\ k_g g_s - k_v v & \text{otherwise,} \end{cases} \\ \dot{\alpha} &= k_a(\alpha_d - \alpha) \end{aligned}$$

so that $g_s$ and $\alpha_d$ are the real control inputs to the car and $g_m$, $k_g$, $k_v$ and $k_a$ are parameters to be estimated.

The minimum static gas $g_m$ is easy to estimate, by simply increasing the gas throttle of the stopped car until the car moves.

Parameter estimation for a dynamic system with equation

$$\dot{x} = k_1(k_2 - x)$$

can proceed as follows. Start with $x = 0$, the system will asymptotically approach $x = k_2$. Suppose $x$ can be sensed within error $\epsilon$. Then let $k_2 = x(t)$ as soon as $|x(t) - x(t+\tau)| \leq \epsilon$ for all $\tau > 0$. Then $k_1$ can be estimated as follows. Let $y = x - k_2$. The solution of $\dot{y} = -k_1 y$ is $y = y_0 e^{-k_1 t}$. Since $y_0 = k_2$ and $y \doteq \epsilon$, we have $k_1 = \ln(\frac{k_2}{\epsilon})/t$.

The gain factor $k_g$ and the friction coefficient $k_v$ can be estimated by the above procedure. By apply a constant throttle $g_s > g_m$ to a initially stopped car, we have $\dot{v} = k_v(k_g g_s/k_v - v)$. For example, if $g_s = 0.2$, $v \to 50$ and $\epsilon = 2$, we have $k_v \doteq 3/t$ and $k_g \doteq 750/t$.

The delay factor $k_a$ can be estimated similarly, except that $\alpha$ has to be sensed via $\dot{\theta}$. Since $\dot{\theta} = v\frac{\tan(\alpha)}{L}$, $d\dot{\theta} = \frac{v}{L}\frac{1}{\cos^2(\alpha)}d\alpha$. We first apply a constant $g_s$ to a car until it moves in a constant velocity; then, at time $t_0$, we apply a constant $\alpha_d$ until $|\dot{\theta}(t) - \dot{\theta}(t+\tau)| \leq \epsilon$ for all $\tau > 0$. We have $k_a = \ln(\frac{\alpha_d}{\epsilon_\alpha})/(t - t_0) = \ln(\frac{v\alpha_d}{\cos^2(\alpha_d)L\epsilon})/(t - t_0)$. For example, if $v = 50$, $\alpha_d = \pi/5$, $L = 12$ and $\epsilon = 2$, we have $k_a \doteq 1/(t - t_0)$.

We can also apply different $g_s$ and $\alpha_d$ to the car and average the results.

# Index