A Temporal Logic Approach to the Analysis and Synthesis of Discrete Event Systems

by

Jing-Yue Lin

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Ottawa-Carleton Institute for Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
University of Ottawa
Ottawa, Ontario
Canada K1N 6N5

ISBN 0-612-15735-0

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

ABSTRACT

The analysis and synthesis of discrete event systems (DESs) are addressed in this thesis by a temporal logic approach. This approach provides a temporal logic model and a temporal logic language for the modeling and specification, an algorithm for reachability analysis, and a procedure for the controller design and synthesis of DESs. To handle the probabilistic system where the point probability distributions are known, a temporal logic model is defined and a generalized temporal logic language is formulated to include the certainty operators for specifications and verification. An algorithm is developed for computing the reachability set and constructing the reachability graph. Using a process algebra, the composition and synthesis of processes are investigated through the process homomorphism; and a procedure is proposed for the controller synthesis and configuration. Then the optimization problem of DESs is solved by the A^* algorithm via a heuristic search. Based on these results, a software package is developed for the temporal logic evaluation, reasoning and simulating discrete event systems. The software is designed using an object-oriented approach and implemented in Objective-C. The simulation results are reported in terms of the logic evaluation, temporal logic reasoning, and discrete event system simulation. Besides, the examples of applications are also given to convey and motivate the theoretical discussions. The results are compared with related works, particularly, qualitative reasoning, other modeling approaches of DESs, and different temporal logic approaches; and our results are seen more advantageous than them in various aspects.

ACKNOWLEDGEMENT

I wish to express my sincere gratitude to my supervisor, Dr. Dan Ionescu, for his constant guidance, encouragement, and support throughout my research.

I would also like to express my sincere thanks to the members of my advisory committee and thesis examiners: Dr. N.U. Ahmed, Dr. S. Matwin, Dr. J.S. Riordon, and Dr. W.M. Wonham for their encouragement, discussions and suggestions.

I also wish to thank my colleagues and friends, Dr. P. Li, Dr. S.S. Lim, Mr. C. Andronic, Mr. C. Drummond, Mr. T. Damerji, and Mr. A. Sisiruca, for their helps and encouragements.

I would also like to acknowledge the financial supports from the University of Ottawa, from the Natural Science and Engineering Research Council of Canada, and from the Ontario Graduate Scholarship.

Finally, I would like to express my sincere gratitude to my family to whom this thesis is dedicated. It would not have been possible to finish this long-term work without their support.

Contents

List of Tables

List of Figures

Notation

Various symbols, superscripts, subscripts, and abbreviations used frequently in this thesis are summarized below. All notation is fully defined where it first arises in the text.

Symbols

$\mathbf{E}$	the set of events (event labels); and $e \in \mathbf{E}$.
E_s	the enabled event set, a subset of $\mathbf{E}$.
f	the transition function of the plant, or a function letter.
$\bar{f}$	the transition function of the closed-loop system.
f_c	the transition function of the controller.
$\mathbf{F}$	the set of formulas.
$\mathbf{F}^*$	the set of subsets of $\mathbf{F}$.
$J(s_0, s_g)$	the cost function index from s_0 to s_g.
l	the labelling function of the plant.
l_c	the labelling function of the controller.
$\bar{l}$	the labelling function of the closed-loop system.
M	the temporal logic model of the plant, or a model.
M_c	the temporal logic model of the controller.
$\overline{M}$	the temporal logic model of the closed-loop system.
$M \| M'$	the composition of M and M'.
p	the probability distribution function.
$\tilde{p}_s$	the probability distribution on W_s.
$\mathbf{Q}$	the set of states of the controller; and $q \in \mathbf{Q}$.
q_0	the initial state of the controller.
R^+	the set of the positive real numbers.
$R(M, s_0)$	the reachability set for $M = (\mathbf{E}, \mathbf{S}, f, s_0, l, p)$.

$\mathbf{S}$	the set of states of the plant; and $s \in \mathbf{S}$.
$\bar{\mathbf{S}}$	the set of states of the closed-loop system.
V	the event structure.
W_s	the set of all paths of S^* that begins at s.
x	the local variable symbol, or a state.

Greek Letters

α	(also β, γ, and so on) an event symbol.
δ	the local variable of the event sort.
ϵ	the null event.
ϕ	the empty sequence or the empty set.
σ	a sequence of states or events, or a trajectory.
$\sigma^{(k)}$	a sequence of states starting from s_k.
θ	the cost function.
Γ	a set of formulas.

Special Symbols

$\neg$	the logic negation symbol.
$\vee$	the disjunction connective.
$\wedge$	the conjunction connective.
$\Rightarrow$	the conditional implication.
$\Leftrightarrow$	the biconditional implication.
$\bigcirc$	the next operator.
$\Box$	the henceforth operator.
$\Diamond$	the eventually operator.
$\mathcal{P}$	the precedes operator.

$\mathcal{U}$	the until operator.
∇	the certainly operator.
Δ	the possibly operator.
$\cup$	the union of sets.
$\cap$	the intersection of sets.

Acronyms and Definitions

AI	artificial intelligence.
CSP	communicating sequential process.
CWA	closed world assumption.
DES	discrete event system.
DESs	discrete event systems.
DP	dynamic programming.
ITL	interval temporal logic.
QDE	qualitative differential equation.
RTTL	real-time temporal logic.
TL	temporal logic.
TLM	temporal logic model.
TLMs	temporal logic models.
pf	a *pf* part is a part which has been processed for the first time on a machine.

Chapter 1

Introduction

A *discrete event system* (DES) refers to a dynamic system that changes state, in accordance with the occurrence, at possibly unknown time points, of discrete events [106]. For example, an event may correspond to the completion of a task or the failure of a machine in a manufacturing system, the arrival or departure of a packet in a communication system. The characteristic features of *discrete event systems* (DESs) include the following:

- Events occur at discrete times instantaneously;
- States have discrete values which may be non-numerical and symbolic;
- Systems are event-driven rather than clock-driven;
- Operations of processes are nondeterministic in general;
- Systems may have internal dynamic behavior and also interact and react with their environments; and
- Processes may operate concurrently and communicate with each other.

As an introduction, in this chapter we will briefly review the literature of DESs. After that, we will state our research objective and motivations, and then give the outline of the thesis and its contributions.

1.1 A Brief Literature Review

DESs are found in many areas, and even those systems that have not been called as discrete event systems traditionally, from a certain perspective may be thought of or modeled as DESs, for example, manufacturing systems, computer operating systems, communication networks, and data bases.

Before we go to the research of DESs, it is instructive to review the history of the DES research which has been driven by an interplay of computers and the electronic technology, by the growth of control system theories, and by the progress of the digital computation. Perhaps the early usage of discrete event concepts was in the many simulation languages: GPSS, SIMULA, and SIMSCRPT among others [24,104]; and they have played an important role in the the evolution of the object-oriented languages. Another early work was [91] in which Petri formulated a representation of asynchronous sequential processes using net structures that have been known as Petri nets. Aveyard presented in [3] a Boolean model for a class of DESs. Then the DES research got an explosive growth recently. Among the models of DESs developed to date we count the following:

- Automata and formal languages; (to this group we also assign finite state machines);

- Boolean algebraic models;
- Finitely recursive processes; (to this group we also assign communicating sequential processes and other process algebras);
- Generalized semi-Markov processes (GSMP) (this category includes all efforts related to general discrete event simulation languages);
- Markov chains;
- Min-max algebraic models;
- Perturbation analysis;
- Petri nets models;
- Queueing network;
- Temporal logic.

While there have been a lot of attempts to construct general models, there is a lack of a universal framework for DESs. Thus far, there is no model which has the potential to eventually serve as the analog of the differential equation for continuous variable dynamical systems [103]. These approaches are simply different means of modeling and analysis of different aspects of the system behavior. Although seeming diverse, they have a common connection through the properties of time and determinism as shown in Table 1.1.

Efforts on modeling DESs have been aimed in two directions. One is concerned with deterministic issues in logical approach or algebraic approach. The

other addressed nondeterministic issues in stochastic formalism. Untimed logical behavior can be modeled by automata and formal languages, finite state machines and Petri nets. Temporal behavior can be described by temporal logic, and timed Petri nets. The algebraic models describe DESs by mathematical equations in minimax algebraic models for timed behavior; Boolean algebraic models, finitely recursive processes and algebraic theories for untimed behavior. Stochastic aspects are emphasized in queuing networks, Markov chains, and perturbation analysis. The classification in Table 1.1 is primary and is not complete. For instance, timed Petri nets and stochastic Petri nets are not included. Furthermore, there is a nondeterministic issue for almost every approach.

	timed $->$	$<-$ untimed
Logical	Temporal logic;	Automata & formal languages; Petri nets;
Algebraic	Min-max algebra;	Finitely recursive process; Boolean algebra;
Performance	Perturbation analysis; Queueing networks; Markov chains; GSMP;	
	Stochastic =>	<= Deterministic

Table 1.1: Classification of DES models

In the sequel of this section, we review briefly each model listed in Table 1.1. We begin our brief review with the untimed logical approaches including automata

and formal languages, and Petri nets.

The theory of automata and formal languages for DES models was initiated by Ramadge and Wonham [95,96]. The objective of this approach has been to examine properties specific for a control theory approach such as controllability, observability, and hierarchical control for DESs from a qualitative viewpoint. In particular, the concepts of dynamics, such as time constants, time and frequency responses, controllability and observability, have played important roles in the development of this approach.

In this approach, a DES is formally a 5-tuple

$$G = (\Sigma, Q, \delta, q_0, Q_m)$$

Here Σ is the (finite) set of event labels; Q is the (possibly infinite) state set; $\delta : \Sigma \times Q \rightarrow Q$ is the transition function; q_0 is the initial state and Q_m is the subset of marker states. Control of a DES G can be exercised by a supervisor S consisting of an automaton equipped with a state-determined output function. The closed-loop system is again a DES, denoted by S/G, the *action* of S on G.

This theory studies a number of qualitative issues such as the existence, uniqueness, and structure of supervisors for simple control tasks. In addition, algorithms have been developed for the synthesis of the designed supervisors. In the use of a formal language, the model is similar to the work of Shaw [101] on flow expressions and the work in [6] on using automata models to study process synchronization; and there are certain points of similarity with the linguistic ap-

proach of Hoare to the specification of concurrent processes [35]. The theory has been proven useful in the theoretical analysis of a number of basic supervisory control problems [95,112]; and it has been extended to cover modular [96], observability and distributed control [56,14], decentralized and hierarchical control [54], and optimal control [55,100,95].

Petri nets have been used as a structure model in the analysis and design of DESs [90,39]. Reachability, observability, and controller synthesis problems have been studied [21,36]. The advantage of using Petri nets is that the model can be described in a graphical form and hence it is possible to visualize the intersections among different components of a complex system. However, Petri-net theories have been criticized in [47] as lacking satisfactory verification methods for liveness properties and (concurrent) data structures.

One of the timed logical models is temporal logic. There are known examples of the applications of temporal logic to the modeling and analysis of DESs. In [27], Fusaoka et al. described a methodology to deal with the behavior of a dynamic system including plant controllers in the framework of temporal logic. Thistle and Wonham used a linear-time temporal logic framework in [105] for the specifications and verification of control systems. For real-time DESs, Ostroff introduced a Real-Time Temporal Logic framework (RTTL) in [85] with the extended state machine models. The above two approaches are based on the work of Manna and Pnueli [72] so that a sound proof system based on their work is immediately available. RTTL is more expressive than Thistle's in real-time aspects. The dif-

ference between two systems is that RTTL is not strongly sound and Thistle's is not strongly complete. This is resulted from adopting the Insertion Rule in RTTL but not in Thistle's. As a result, state formulas are interpreted with respect to the initial state of a sequence in Thistle's, and to all states of a sequence in RTTL.

Linearity is a most welcome property in the system theory since it significantly simplifies the system analysis [1,111]. However, DESs encountered in applications are usually nonlinear. Generally, coping with nonlinearity consists of considering a system as locally linear under prescribed behavior conditions. But linearization is not the only thinkable approach for DESs. In [16], Cohen et al. established a "linear" model for a class of deterministic DESs in the sense of an unusual algebra already known in the literature under names such as "minimax algebra" [17], or "path algebra" [12]. A state-space representation of DESs was given to efficiently calculate the periodical steady state of the closed DES, involving a finite set of activities repeatedly performed by means of a limited amount of available resources.

We have extended Cohen's work to the nondeterministic DESs with known point probability distributions [57,58,61]. Under such extensions, our model of the nondeterministic DES is linear in the sense that minimax algebra is combined with the usual classical algebra. The minimax algebra is used to process the treatment of time sequences; and the usual classical algebra is used to introduce a Markov structure that assigns point probabilities to events. A hybrid dynamical state-space representation of the nondeterministic DES is then established to determine the quantitative structured features of the control problem. As a direct result of

this approach, the output feedback control is generated. To predict the long range performance of the DESs, the asymptotic behavior of the system is established.

The "pseudo-linear" algebras in the above timed algebraic models make them similar to the usual linear dynamic systems. This analogy has been proved very useful, since notions such as eigenvalue, eigenvectors, and their relations with what can be called the "modes" [83] of their counter-part in the $(max, +)$ algebra. Therefore, they are mathematically and computationally attractive. Their disadvantage is that they can not deal with the systems whose state values are non-numerical or symbolic; and this restricts the scope of applications for DESs. This approach was abandoned in this thesis due to the disadvantage mentioned above.

The untimed algebraic models include Boolean algebraic models and finite recursive processes. Boolean models of DESs have been given in [3] for the deterministic DESs using a Boolean algebra. The state of the model is represented by a vector of Boolean variables, and enabling conditions and state transitions are embodied in a Boolean matrix resulting in a difference equation formulation for the model. In the Boolean difference equation model, the next state is determined from a knowledge of the present state and the firing variable input. A simple conveyor system was used as a vehicle for relating the development of the Boolean matrix equation model to a physical system.

Finite recursive processes have been developed in [40] for DESs by using

Hoare's deterministic process theory, and a general approach toward constructing model algebras has also been proposed in [41]. These models are proposed to help in the specification, implementation and simulation of communication protocols and supervisory control strategies.

The stochastic approaches include the perturbation analysis, queueing theory, and Markov chains. Sample paths of DESs which contain a considerable amount of information have been exploited in a perturbation analysis context– given several system parameters, one may predict the behavior of the system when the values of these parameters are perturbed. Some of the early work in this area were limited to "infinitesimal" perturbation for continuous parameters such as service rates of DES resources (*e.g.* machines, processors, network switches), and have come to be known as Infinitesimal Perturbation analysis. Advances in this approach have been seen in [103,34], and a good overview may be found in [33].

Queueing network models have been used to describe the synchronization constraint problem of resource sharing and have provided a variety of quantitative insights into the system performance [4]. It examines a class of DESs, and gives a variety of results, related to the performance analysis. A stochastic DES has been modeled as a continuous-time Markov process and then a Markov chain [13] can be obtained from the system. A generalized semi-Markov process (GSMP) [29] defines a particular type of stochastic processes. It captures the essential dynamic structure of a DES and then gives the qualitative theory and numerical algorithms.

No single approach to the modeling and analysis of DESs will suffice for all problems of interest. Each of the above models has its own applications, virtues, and limitations. In all models, there is a need for more expressive descriptions of system dynamics, for a concise means of system and problem specifications, for the study of interesting subclass of systems with special structures, and for optimization algorithms and procedures.

1.2 Research Objective and Motivations

Our research objective is to build a framework for the analysis and synthesis of DESs. It focuses on the idea of how to ensure the orderly and efficient flow of events. This can be achieved within a hierarchy of levels: higher levels dealing with decision making and optimization, lower levels with the logical aspects of the problem specification, modeling and controller design, and the lowest levels addressing implementation issues such as the software implementation.

As many DES problems arise in computer systems, communication networks, and computer-related applications, we choose a temporal logic approach for modeling DESs. This has the following motivations:

- Temporal logic is well-developed and continuous to be an active area of the research in Computer Science [28]. It has found applications in AI, especially in temporal reasoning of simulating actions [102], planning systems and expert systems [44].

- The notation of DESs is captured by temporal logic in an intuitively appealing and simple manner [86]. Thus, the use of temporal logic as a model for DESs is of growing interest [105]. Also, temporal logic is proving to be of some use in the area of specifying the behavior of real-time DESs [85].

- Temporal logic has been applied to the specifications of communication protocols [99], verification of concurrent programming [72], hardware specification [9], and automatic verification of sequential circuits [10]. The use of temporal logic as a formal method is growing, particularly its applications to the software design and development. Working in this approach could find a way to generate a controller software from the temporal logic specifications.

As seen from the above review, the temporal logic framework used in control problems needed to be more expressive in order to deal with the probabilistic features in DESs. This gives us a motivation to define a temporal logic model and formulate a generalized temporal logic framework including the certainty operators for the probabilistic systems. Then the framework is applied to the specification and verification of the properties for the nondeterministic DESs where the point probability distributions are known. As a framework, it is necessary to have notions of dynamics. This motivates us to proceed to the reachability analysis for the dynamic behavior of the system. For the temporal logic models, the composition of them is investigated by a process algebra and then extended to the synthesis problems leading to the construction of suitable controllers. The fact that computational algorithms are needed in the temporal logic approach gives us a motivation to develop an optimization procedure via a heuristic search. Within such a temporal logic framework, a software package is developed for logic evaluation,

reasoning and simulating DESs. It is designed in an object-oriented approach and implemented in Objective-C. Then three examples of simulation are demonstrated.

1.3 Outline of the Thesis and Contributions

The sequel of this thesis is organized as follows:

In Chapter 2, we will introduce a temporal logic framework for the specifications and verification of properties for the control problems of DESs.

In Chapter 3, we will define a temporal logic model based on an event structure and formulate a generalized temporal logic framework which includes the certainty operators to specify the probabilistic features of DESs. The syntax, semantics, and proof system of the language will be described.

In Chapter 4, we will apply the generalized temporal logic framework to specify and verify the properties of the nondeterministic DESs where the point probability distributions are known.

In Chapter 5, we will analyze the dynamics of DESs and define reachability of temporal logic models. The relationship between validity of logic formulas and reachability of states will be given and the reachability analysis of DESs will be developed by constructing a reachability graph and computing the reachability set.

In Chapter 6, we will compose temporal logic models through a process algebra and then propose a procedure for the reachability synthesis and controller design of a given DES. As a result, we also discuss the relationship between the controller synthesis procedure and the verification.

In Chapter 7, we will discuss the active controls and a semi-metric space, and then develop a procedure of optimization for DESs via a heuristic search and use the A^* algorithm to solve the optimization problem.

In Chapter 8, we will develop a software package for the logic evaluation, for reasoning about DESs, and for the DES simulation. The software is designed using an object-oriented approach and implemented in Objective-C.

In Chapter 9, we will demonstrate three examples of the simulation: a logic evaluation, a gun shooting example, and a manufacturing system.

In Chapter 10, we will summarize the results of the thesis and discuss the related research works. After that, the suggestions for further research will be given at the end.

The original contributions of the thesis include:

1. The temporal logic model based on an event structure; the generalized temporal logic which includes the certainty operators for the case where the point probability distributions are known, the proof system, Soundness Theorem,

and Completeness Theorem, and these are generalizations of those in the framework given by Manna and Pnueli [72] and used by Thistle and Wonham [105]. see Sections 3.2, 3.3, 3.4, [59,60,62,66].

2. The general forms of specifications of the control problem of the DESs; Procedure 4.2.1; verification of a nondeterministic DES with point probability distributions known. see Sections 4.2, 4.3. [59,60,66].

3. Theorem 5.2.1; Theorem 5.3.2; Corollary 5.3.3 that the validity of logic formulas is equivalent to the reachability of states in a given temporal logic model; Algorithm 5.3.1, the algorithm for constructing a reachability graph and computing the reachability set. see Sections 5.2, and 5.3. [62,68,70,71].

4. The composition of temporal logic through a process algebra, Theorems 6.3.1, 6.3.3, 6.3.4; Corollary 6.4.1; Procedure 6.4.1, the procedure for the controller design and reachability synthesis; and Theorem 6.4.2. see Sections 6.3, 6.4. [63,68,43,71].

5. Definition 7.3.1, the definition of a semi-metric space; the statement of the optimization problem for DESs; Theorem 7.5.1, the solution to the optimization by A^* algorithm with an up-bounded semi-metric function via a heuristic search. see Sections 7.3, 7.4, 7.5. [67,64].

6. The object-oriented design of the software package; its implementation in Objective-C; and simulation of applications. see Sections 8.2, 8.3, 8.4, 9.2, 9.3, 9.4. [69,42].

Chapter 2

A Temporal Logic Framework for the Control Problem of DESs

Temporal logic has many applications [28,102,44] and some of them will be introduced briefly in the next section. After an introduction, we will present a temporal logic framework which has been applied to the specifications and verification of the properties for the control problem of DESs in [105].

2.1 Introduction

Temporal logic has its origins in philosophy where it was used to analyze the structure or topology of time [97]. Philosophers introduced special temporal operators such as $\Box$ (henceforth) and $\Diamond$ (eventually), for the analysis of temporal connectives in languages. As a valuable tool for analyzing the topology of time, various types of semantics can be given to the temporal logic operators.

One of the areas in which Computer Science has found a use for temporal logic is *Artificial Intelligence* (AI) [28]. To duplicate human intelligence or merely to simulate its effects, it is important that human intelligence manifests itself both in what we do and what we say, so that much AI research has been directed towards the tasks of simulating action (event) and simulating language. Both of these tasks need a proper treatment of time. For this purpose, Bruce [11] presented a formal model of temporal references in a natural language. After that, Kahn and Gorry [45] introduced a time manager to handle temporal matters in the problem-solving routines. To serve as a framework for programs that must deal with time, McDermott proposed a temporal logic for reasoning about processes and plans [77]. In the AI approach to temporal logic, Allen elaborated what he calls a theory of time [2] as a part of a foundation for the sorts of temporal reasoning required in the AI applications.

Reasoning about time typically involves drawing conclusions from the given information. In general, the given information is incomplete because of ignorance, nondeterminism, and indecision. Despite the lack of complete information, predictions have to be made to pursue hypotheses and to plan for the future. A computational approach to temporal reasoning was presented by T. Dean et al. in [20,19], called temporal database management. This approach centers around techniques for managing a data base of assertions corresponding to the occurrence of events and the persistence of their effects over time. The temporal reason maintenance was performed by keeping track of dependency information about the truth of facts.

In recent years, temporal logic has found applications in the areas of software verifications and knowledge based systems [28,44]. For example, to specify program behavior, the structure of states is the key concept that makes temporal logic suitable for program specifications. A formula, containing temporal logic operators, is interpreted over a structure of states. In programming languages, the structure represents the computations executed by a program. Hence, temporal logic has been proposed as a means of formalizing assertions expressing how the state of a program changes with time; and it has been useful for software verifications [72,73,75,15].

As temporal logic allows a high level description of program states and the way they change with time, it may provide a useful formal approach to the control problem of DESs; and indeed, it has already applied in this area in [27,105]. In the sequel of this chapter, we will introduce a temporal logic framework to the control problem of DESs [105]. It is a discrete time temporal logic; and its language and proof system are adapted from those used in the verification of concurrent programs by Manna and Pnueli [72,73]. In Section 2.2, we describe the syntax and semantics of the temporal logic language. In Section 2.3 we present a system of proof rules, and then its soundness and completeness in Section 2.4. In Section 2.5, we apply this framework to the specifications and verification for the control problem of a DES. Following that, we talk about the extension of temporal logic for real-time systems. Finally in Section 2.7, we discuss the advantages and disadvantages of this framework.

2.2 The Language, Its Syntax and Semantics

A few symbols of elementary set theory are used in the language. In what follows, we consider the axiomatics of the set theory as known. We denote a set by a bold or capital letter such as **S** or Γ, or embracing the members of the set with curly braces such as $\{w\}$, $\{w_1, w_2\}$. Also, we use the ideas of set-membership $\in$, of set union $\cup$, and of set intersection $\cap$.

The symbols of the language include individual constant symbols; individual variable symbols; definitional identity $=$; function letters; predicate letters; logical connectives: $\neg$ (the negation symbol) and $\vee$ (the disjunction connective); and temporal operators: $\bigcirc$ (the next operator), $\Box$ (the henceforth operator), and $\mathcal{U}$ (the until operator).

There are different time points which may yield different truth values of formulas in the language. We assume that the set of time points is infinite, discrete, and linearly ordered by the usual precedence relations $<$ and $\leq$. Hence we use the positive natural numbers as a set of time points.

The language is a many-sorted one. Following Enderton [23], we assume that we have a non-empty set **I**, whose members are called sorts. There is a countable number of constant symbols, and also of variable symbols, of each sort i. For each $n > 0$ and each $(n+1)$-tuple of sorts $(i_1, i_2, ..., i_{n+1})$, there is a countable set of $n-$ary function letters said to be of sort $(i_1, i_2, ..., i_{n+1})$. Also, for each $n > 0$ and each n-tuple of sorts $(i_1, i_2, ..., i_n)$, there is a countable set of $n-$ary predicate

letters said to be of sort $(i_1, i_2, ..., i_n)$.

For any sort i, the terms of sort i are defined as those of a standard first order logic. In addition

- the expression $(\bigcirc t)$, for any term t of sort i, is a term.

Definition 2.2.1: The well-formed formulas (or simply, formulas) of the language are defined as follows:

(i): for any terms $t_1, \cdots, t_n$, any predicate symbol P, $P(t_1, \cdots, t_n)$ is a formula;

(ii): for any formula w, $(\neg w)$, $(\bigcirc w)$, and $(\Box w)$ are formulas;

(iii): for any formulas v and w, $(v \vee w)$ and $(v \mathcal{U} w)$ are formulas.

Informally, $(\bigcirc w)$ means that w will be true at the next time instant; $(\Box w)$ has the English paraphrase: w will be true henceforth; and $(v \mathcal{U} w)$ has the intuitive interpretation: v will be true until w is (and w will indeed eventually be true).

In the language, a state, an interpretation, and a structure are defined as follows:

Definition 2.2.2: Let D_i be the domain containing x, y, z, $\cdots$ for every sort i. A state is an assignment of an element of D_i to each variable symbol of sort i. An interpretation M specifies a domain D_i for each sort i, and assigns to each constant symbol of the sort i an element of D_i, to each function letter of sort $(i_1, i_2, ..., i_{n+1})$ a function $f : D_{i_1} \times D_{i_2} \times \cdots \times D_{i_n} \rightarrow D_{i_{n+1}}$ and to each predicate letter of sort $(i_1, i_2, ..., i_n)$ a relation $P \subset D_{i_1} \times D_{i_2} \times \cdots \times D_{i_n}$.

Definition 2.2.3: A structure is defined by (M, σ) where

- M is an interpretation; and
- σ is an infinite sequence of states $s_0, s_1, s_2, ...$

The formulas are evaluated with respect to the structure (M, σ). For the structure (M, σ) with sequence $\sigma = s_0 s_1 s_2 \cdots$, and for any integer $k \geq 0$, $M^{(k)}$ represents the structure $(M, \sigma^{(k)})$ with sequence $\sigma^{(k)} = s_k s_{k+1} \cdots$, and $M^{(0)} = M$. The following notations are used:

- $M[t]$ represents the value assigned to the term t by M;
- $M[C]$ represents the value assigned to any global constant symbol C by M;
- $M[x]$ represents the value assigned to any local variable symbol x under the initial state of σ by M;
- f^M represents the function assigned to the function letter f by M;
- P^M represents the relation assigned to the predicate letter P by M.

Proposition 2.2.1: For any term t, $M[(\bigcirc t)] = M^{(1)}[t]$.

Intuitively, the states of the sequence of a structure represent the state of affairs at successive instants of time. So, the global constant symbols, predicate letters and function letters have interpretations that are independent of time, while the local variable symbols represent quantities that change with time.

Definition 2.2.4: A state formula is any well-formed first order formula which is evaluated with its truth value depending only on the first state of the path at which it is evaluated. A temporal formula is a formula constructed from state formulas to which some temporal operators are applied.

Definition 2.2.5: A formula w is said to be satisfied by a structure (M, σ), written by $\models_M w$, if it is defined as follows:

(i). for any formula of the form $P(t_1, t_2, ..., t_n)$ where P is a predicate letter and $t_1, t_2, ..., t_n$ are terms, $\models_M P(t_1, t_2, ..., t_n)$ if and only if

$$(M[t_1], M[t_2], ..., M[t_n]) \in P^M$$

(ii). $\models_M (\neg w)$ if and only if it is not the case that $\models_M w$;

(iii). $\models_M (v \vee w)$ if and only if either $\models_M v$ or $\models_M w$;

(iv). $\models_M (\bigcirc w)$ if and only if $\models_{M^{(1)}} w$;

(v). $\models_M (\Box w)$ if and only if for all $i \geq 0$, $\models_{M^{(i)}} w$;

(vi). $\models_M (v \mathcal{U} w)$ if and only if there exists an $n \geq 0$ such that $\models_{M^{(n)}} w$ and for all k, $0 \leq k < n$, $\models_{M^{(k)}} v$.

(ii) in Definition 2.2.5 is the definition of that the negation of a formula w is true; and it is defined as not the case of that w is true. It is equivalent to the Closed World Assumption (CWA) [44] which says what we do not know can not be true. In a sense of CWA, we assume that the state descriptions completely characterize the positive facts of a model. For example, we use $fire(GUN)$ to represent the fact of that the trigger of the gun is pulled. If we say that the trigger of the gun is not pulled at time 1, then this means it is not the case of

$fire(GUN)$, that is $\neg fire(GUN)$ is true, at time 1. So if $fire(GUN)$ is not given in the description, then it implies that $\neg fire(GUN)$ is true. Another example, if $W = P$ is not in the formula, then this means that it is not the case of $W = P$, that is, $\neg W = P$ is true (we usually write this as $W \neq P$).

The following are some convenient abbreviations to be used in the sequel:

(i). conditional implication: $v \Rightarrow w$ abbreviates $(\neg v \vee w)$;

(ii). conjunction connective: $v \wedge w$ abbreviates $(\neg(\neg v \vee \neg w))$;

(iii). biconditional implication: $v \Leftrightarrow w$ abbreviates

$$(v \Rightarrow w) \wedge (w \Rightarrow v);$$

(iv). eventually operator: $(\Diamond w)$ abbreviates $(\neg(\Box(\neg w)))$ and has the English paraphrase: w will eventually be true;

(v). precedes operator: $(v \mathcal{P} w)$ abbreviates $(\neg((\neg v)\mathcal{U} w))$ and has the intuitive interpretation: v will become true before w does.

Definition 2.2.6: A formula w is said to be valid, written by $\models w$, if $\models_M w$ for all M.

2.3 The Proof System

The proof system has three parts: the general part, the formal mathematical reasoning, and the description of events. The general part deals only with the formal temporal and logical reasoning.

2.3.1 The general part

Axiom schemata

If a formula w has the form of one of the following schemata, then w and $\Box w$ are axioms:

(A0) w, where w is an instance of a propositional tautology;

(A1) $\bigcirc(w_1 \Rightarrow w_2) \Rightarrow (\bigcirc w_1 \Rightarrow \bigcirc w_2)$;

(A2) $\neg \bigcirc w \Leftrightarrow \bigcirc \neg w$;

(A3) $\Box(w_1 \Rightarrow w_2) \Rightarrow (\Box w_1 \Rightarrow \Box w_2)$;

(A4) $w_1 \mathcal{U} w_2 \Rightarrow \Diamond w_2$;

(A5) $\Box w \Rightarrow w \wedge \bigcirc \Box w \wedge \bigcirc w$;

(A6) $w_1 \mathcal{U} w_2 \Leftrightarrow [w_2 \vee (w_1 \wedge \bigcirc(w_1 \mathcal{U} w_2))]$;

(A7) $\Box(w \Rightarrow \bigcirc w) \Rightarrow (w \Rightarrow \Box w)$;

(A8) $t = t$, for any term t;

(A9) $t_1 = t_2 \Rightarrow (w(t_1, t_1) \Leftrightarrow w(t_1, t_2))$, for any terms t_1 and t_2 and any formula $w(t_1, t_1)$ and $w(t_1, t_2)$, where $w(t_1, t_2)$ is obtained from $w(t_1, t_1)$ by replacing with t_2 some of the occurrences of t_1 that are not within the scope of any temporal operator;

(A10) for any $n-$ary predicate letter P and terms $t_1, t_2, ..., t_n$

$$\bigcirc P(t_1, t_2, ..., t_n) \Leftrightarrow P((\bigcirc t_1), (\bigcirc t_2), ..., (\bigcirc t_n))$$

(A11) $\bigcirc f(t_1, t_2, ..., t_n) = f((\bigcirc t_1), (\bigcirc t_2), ..., (\bigcirc t_n))$ for any $n-$ary function letter f, terms $t_1, t_2, ..., t_n$ and equality symbol $=$; and

$C = (\bigcirc C)$, for any constant symbol C and equality symbol $=$.

Inference rule:

Modus ponens: for any formulas w_1 and w_2

$$\frac{w_1,\ w_1 \Rightarrow w_2}{w_2}$$

Here we use the notation

$$\frac{w_1, w_2, ..., w_n}{w_{n+1}}$$

to mean that one may infer the formula w_{n+1} from the formulas $w_1, w_2, ..., w_n$. Also the notation $\vdash w$ means that the formula w is a theorem of our proof system, and $\Gamma \vdash w$ means that w can be deduced from the set Γ of formulas.

Generalization rule:

If $\{w_1, w_2, ..., w_n\} \vdash w_{n+1}$, then $\{\Box w_1, \Box w_2, ..., \Box w_n\} \vdash \Box w_{n+1}$.

Propositional reasoning (PR):

If

$$\wedge_{i=1}^{n} w_i \Rightarrow w_{n+1}$$

is an instance of a tautology, then

$$\frac{w_1, w_2, ..., w_n}{w_{n+1}}$$

Deduction theorem:

$\Gamma \cup \{w_1\} \vdash w_2$ if and only if $\Gamma \vdash w_1 \Rightarrow w_2$ for any set of formulas $\Gamma \cup \{w_1, w_2\}$.

Derived computational induction rule:

$$\frac{\Box(w_1 \Rightarrow (w_2 \wedge \bigcirc w_1))}{\Box(w_1 \Rightarrow \Box w_2)}$$

Right until introduction:

$$\Box(w_1 \wedge w_3 \Rightarrow \bigcirc w_2)$$

$$w_1 \Rightarrow \Diamond w_3$$

$$\frac{\Box(w_1 \Rightarrow (\bigcirc w_2 \vee \bigcirc w_1))}{w_1 \Rightarrow (w_1 \mathcal{U} w_2)}$$

Right precedes introduction:

$$\frac{\Box(w_1 \Rightarrow \neg w_3 \wedge (w_2 \vee \bigcirc w_1))}{\Box(w_1 \Rightarrow w_2 \mathcal{P} w_3)}$$

$\mathcal{P}$−chain:

$$\frac{\Box\{v_i \Rightarrow \neg w \wedge [\bigvee_{j<i} v_j \vee \bigcirc(\bigvee_{k=1}^{i} v_k)]\} \quad \textit{for all } i, \ \ 0 < i \leq n}{\Box\{\bigvee_{i=1}^{n} v_i \Rightarrow v_0 \mathcal{P} w\}}$$

Frame Theorem:

$$\vdash \Box[\bigcirc w(y_1, y_2, ..., y_n) \Leftrightarrow w((\bigcirc y_1), (\bigcirc y_2), ..., (\bigcirc y_n))]$$

for any formula $w(y_1, y_2, ..., y_n)$ where y_1, y_2, $\cdots$, y_n are the only local variable symbols in $w(y_1, y_2, ..., y_n)$ and formula $w((\bigcirc y_1), (\bigcirc y_2), ..., (\bigcirc y_n))$ is obtained by replacing each of these local variable symbols y_i in w by $(\bigcirc y_i)$.

2.3.2 Formal mathematical reasoning

Manna and Puneli [72] include in their proof system a 'domain part' to do formal mathematical reasoning about the domains of their interpretation. Since the main interest for DESs here is temporal logic reasoning, all of those formulas that are true under intended evaluation are axioms:

(A12) if w is a formula that is true under the obviously intended evaluation, then w and $\Box w$ are axioms.

We call such axioms domain axioms. In order to further simplify our formal mathematical reasoning, we state the following derived rule:

Mathematical reasoning (MR):

If the formula $(\wedge_{i=1}^{n} w_i) \Rightarrow w_{n+1}$ is a domain axiom, then

$$\frac{w_1, w_2, ..., w_n}{w_{n+1}}$$

2.3.3 The description of events

One of the sorts of the language is used to represent events which include various transitions such as actions and operations. The global constant symbols of this sort are called *event symbols* and the symbol δ is included among the local variables of that sort. Thus, the formula $\delta = \alpha$ means that the event represented by the event symbol α is about to occur.

The sequence of a structure is infinite, but it may be that a system eventually stops making state transitions because a goal has been achieved and no further action is required, or because a deadlock has occurred and no further transitions are possible. In order that such situations can be modeled, the symbol ϵ is included among the event symbols and stands for the null event. This event leaves unchanged the values of all local variable symbols; and it gives the last axiom schema of the proof system as follows.

(A13) $\Box[\delta = \epsilon \Rightarrow (\bigcirc x) = x]$ where x is any local variable symbol.

2.4 Soundness and Completeness

As mentioned in Subsection 2.3.1, the notation $\vdash w$ means that the formula w is a theorem of the proof system, and $\Gamma \vdash w$ means that w can be deduced from

the set Γ of formulas. The notation $\Gamma \models w$ means w is a consequence of Γ; and it is defined by using Definition 2.2.6 as follows: $\Gamma \models w$ if $\models v$ for every $v \in \Gamma$ implies $\models w$.

The proof system given in the previous section is sound, strongly sound, and complete; but not strongly complete as given in the below.

Definition 2.4.1: A proof system is sound iff $\vdash w$ implies $\models w$ for all w; and it is strongly sound iff $\Gamma \vdash w$ implies $\Gamma \models w$ for any Γ and w.

Theorem 2.4.1: The proof system given in Section 2.3 is sound and strongly sound [105].

Definition 2.4.2: A proof system is complete iff $\models w$ implies $\vdash w$ for all w; and it is strongly complete iff $\Gamma \models w$ implies $\Gamma \vdash w$ for any Γ and w.

Theorem 2.4.2: The proof system given in Section 2.3 is complete but not strongly complete [105].

This completes the descriptions of the language and proof system of the temporal logic framework which will be the fundamental base of the dissertation. For a DES, if a structure satisfies the given plant specification, then it represents a physically possible trajectory of the system. If, in addition, it satisfies the controller specification, then it represents a possible trajectory of the closed-loop

system. Hence, the temporal logic framework provides a way to verify the properties in the analysis process of deterministic DESs. Examples of applications of this framework have been shown in [27,105], and a simple example will illustrate this approach in the next section.

2.5 An Example

In this section, a simple example of a DES [62] will be given to illustrate the analysis of a DES in a temporal logic and to outline the formal specification and verification of some properties of the system.

Let's consider a simple machine shop processing parts: The machine shop waits until a part comes and then processes the part and sends it out for delivery. Assuming that there are N parts to be processed, we can write down a temporal logic description of the plant.

The global constant symbols O, W and P will represent the three possible states of the N parts: those of being outside the machine shop, waiting to be processed, and being processed on machine, respectively. The local variable symbols $x_1, x_2, \cdots, x_N$ will be used to represent the states of the N parts; and these symbols are of the same sort as the global symbols O, W and P.

For any i, $0 < i \leq N$, the following event symbols will be used: $arrive_i$: the arrival of part i; $begin_i$: the commencement of processing of part i; end_i: the

departure of part i; and ϵ stands for the null event. By using these symbols, we can give the specifications of the dynamics of the plant as follows:

Dynamics of the Plant

$$\Box[\vee_{i=1}^{N}(\delta = arrive_i \vee \delta = begin_i \vee \delta = end_i) \vee \delta = \epsilon] \qquad (2.5.P1)$$

$$\Box[O \neq W \wedge O \neq P \wedge P \neq W] \qquad (2.5.P2)$$

Formula (2.5.P1) simply describes the set of possible events; it means that the only events that can occur are those listed above. Formula (2.5.P2) means that the three possible states of a part are distinct.

$$\Box\{\wedge_{i=1}^{N}[\delta = arrive_i \Rightarrow x_i = O \wedge (\bigcirc x_i) = W \wedge [\wedge_{i\neq j=1}^{N}(\bigcirc x_j) = x_j]]\} \qquad (2.5.P3)$$

$$\Box\{\wedge_{i=1}^{N}[\delta = begin_i \Rightarrow x_i = W \wedge (\bigcirc x_i) = P \wedge [\wedge_{i\neq j=1}^{N}(\bigcirc x_j) = x_j]]\} \qquad (2.5.P4)$$

$$\Box\{\wedge_{i=1}^{N}[\delta = end_i \Rightarrow x_i = P \wedge (\bigcirc x_i) = O \wedge [\wedge_{i\neq j=1}^{N}(\bigcirc x_j) = x_j]]\} \qquad (2.5.P5)$$

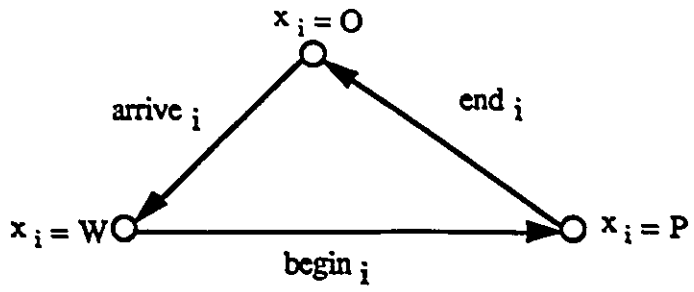


Figure 2.1: Graph of the part processing.

Formula (2.5.P3) describes the effect of the arrival of part i: its state changes from O to W, while the states of the other parts remain the same. (2.5.P4) and (2.5.P5) are similar to (2.5.P3), describing the commencement of processing, and the departure, respectively, of part i.

Initial condition

$$\wedge_{i=1}^{N} x_i = O \qquad (2.5.P6)$$

It means that initially there are no parts at the machine shop.

The graph of the part processing is shown in Figure 2.1.

Now we express the required behavior of the composite system as that: Parts are to be processed one at a time and in the order in which they arrive. This gives the following specifications:

$$\Box[\wedge_{i=1}^{N}(x_i = P \Rightarrow [\wedge_{i\neq j=1}^{N} x_j \neq P])] \qquad (2.5.CL1)$$

$$\Box\{\wedge_{i,j=1;i\neq j}^{N}[x_i \neq W \Rightarrow (\delta = arrive_i \mathcal{P}\delta = arrive_j \Rightarrow \delta = begin_i \mathcal{P}\delta = begin_j)]\} \qquad (2.5.CL2)$$

The next step is to construct a controller. A first-in-first-out queue should be maintained for each machine in order to keep track of parts and the order in which they arrived. When a part arrives at the machine shop, it should be pushed onto the tail of the queue. It must then be prevented from putting on the machine until it has reached the head of the queue. We use the symbol q to represent the queue. In any state, q is assigned a value that is either some finite sequence of integers between 1 and N or the empty sequence, which is denoted by the symbol ϕ. The events in the plant and in the controller are selected for synchronization. Then, the specifications of the controller are as follows.

Dynamics of the Controller

$$\Box\{\wedge_{i=1}^{N}[\delta = arrive_i \Rightarrow (\bigcirc q) = q * i]\} \qquad (2.5.C1)$$

where $*$ represents concatenation of sequences and $q * i$ means that i is pushed onto the tail of queue q.

$$\Box\{\wedge_{i=1}^{N}[\delta = begin_i \Rightarrow i \propto q \wedge (\bigcirc q) = q]\} \qquad (2.5.C2)$$

where $i \propto q$ means that i is an initial element of q.

$$\Box\{\wedge_{i=1}^{N}[\delta = end_i \Rightarrow (\bigcirc q) = q|^1]\} \qquad (2.5.C3)$$

where $q|^1$ represents the sequence obtained by deleting the first element of q.

Initial Condition of the Controller

$$q = \phi \qquad (2.5.C4)$$

That the resulting composite system has the required behavior is checked by verifying the following.

Theorem: The closed-loop system specifications (2.5.CL1)-(2.5.CL2) can be deduced from the plant specifications (2.5.P1)-(2.5.P6) and the controller specifications (2.5.C1)-(2.5.C4).

This theorem can be proven with the help of the following lemma which has been partly in Lemmas 3.1.1 and 3.1.2 in [105] respectively.

Lemma: The following formulas can be deduced from (2.5.P1)-(2.5.P6) and (2.5.C1)-(2.5.C4):

$$\Box[\wedge_{i=1}^{N}(x_i = P \Rightarrow i \propto q)] \qquad (2.5.L1)$$

$$\Box[\wedge_{i=1}^{N}(x_i = O \Leftrightarrow i \notin q)] \qquad (2.5.L2)$$

This example is a very simple one; and rationale behind the control logic is clear and it is easy to see that the closed-loop system has the desired properties. The formal proof is also straightforward. It has been shown that temporal logic leads to easily understood high-level specifications.

2.6 Temporal Logic for Real-Time Systems

Real-time systems are characterized by quantitative timing properties relating occurrences of events. For example, the exact time between events, and the maximal or minimal time between events. In [94], Pnueli and Harel gave a brief account of some attempts to use temporal logic for the specification of real-time systems. The computational model used is a timed interleaving model where enabled transitions have associated lower and upper bounds within which they must be taken. It considers two possible extensions of temporal logic to deal with real-time. The first adds a global clock as an explicit variable to which the specification may refer. The second approach introduces quantitative temporal operators. For specifying synchronous systems it recommends the use of a discrete time domain such as the natural numbers and for asynchronous systems a dense time domain such as the rational numbers.

One of the methods using the first approach is that of Ostroff [85,86] for the control of real-time DESs. It introduces a distinguished variable t representing

the clock. A typical formula of his logic is the following:

$$\varphi \wedge t = T \rightarrow \Diamond(\psi \wedge t \leq T + 5) \tag{2.6.1}$$

where T is a global variable [85]. (2.6.1) says that if φ is true now and the clock reads T ticks, then within $T + 5$ clock ticks ψ must become true. Thus, once φ becomes true, ψ must become true no more than 5 ticks later. The explicit clock variable allows the easy reference to times when specifying the properties of the real-time. However, the explicit clock variable is against the original philosophy of temporal logic to abstract from time as much as possible.

An example using the second approach is that of Koymans [46]. It extends temporal logic with metric operators from their qualitative versions. For example, the semantics of (2.6.1) can be written equivalently by the metric temporal logic as follows:

$$\varphi \rightarrow \Diamond_{\leq 5}\psi \tag{2.6.2}$$

The metric operators have been applied to the formal specifications of real-time systems. However, there is no proof system in the metric temporal logic so that it lacks means of verification.

2.7 Discussion

In this chapter, we have introduced the research background as a starting point of this thesis. As a general background, we have briefly introduced the development of applications of temporal logic to the AI reasoning, database man-

agement, software verification and high-level specifications.

We have introduced a discrete linear-time temporal logic framework and its applications to the specification and verification of the properties for the control problem of DESs. The framework is a starting point of this thesis; and we will extend it into a generalized temporal logic framework in the next chapter.

For the real-time applications, we have talked about the development on the real-time aspects of temporal logic. We have emphasized on two approaches: Ostroff's Real-Time Temporal Logic and Koymans's Metric Temporal Logic. We will come back to the real-time issue later on.

The temporal logic framework presented in Sections 2.2-2.5 uses a linear-time temporal logic adapted from that of Manna and Pnueli [72] so that a proof system is available based on their work although it is slightly different from their proof system. In this framework a sort of the temporal logic is set aside for the events of the system. This allows for easy references to events in the specifications.

A major disadvantage of this approach is that the full plant-controller system must be available before verification can proceed. To overcome this disadvantage, we will develop the method to do an early verification, *i.e.* to verify the specification of the DES's plant before or without designing a controller. Furthermore, we will provide a theoretical basis for the controller design and synthesis in a temporal logic approach.

The framework introduced here can be improved to be more expressive to deal with the probabilistic features of DESs. In the next chapter, we will extend this framework by including the certainty operators to a class of nondeterministic DESs where the point probability distributions are known.

Chapter 3

A Generalized Temporal Logic

In this chapter, we will generalize the discrete linear-time temporal logic presented in Chapter 2 by including the certainty operators for reasoning of the nondeterministic systems with point probability distributions known, *i.e.* the probabilistic systems.

3.1 Introduction

The principal features of DESs are that they are discrete and nondeterministic [85,95]. The nondeterministic DES models have been proposed in [14,95] using nondeterministic automata without explicit mention of probabilities, and have been studied using minimax algebra in [57,58,61,82] for the nondeterministic DES with known probability distributions.

Relatively little work has been done for the nondeterministic DES by temporal logic. In [72,73,74] and Chapter 2, it has been shown that temporal logic is useful

for specifying and verifying safety and liveness properties of DESs. However, its most significant limitation is that it can not express properties of the probabilistic behaviors – for example, that a process has probability 0.9 of terminating [52] The motivation for this chapter stems from the need for a generalized temporal logic to handle the probability distributions in the nondeterministic DESs [14,52].

In this chapter, a linear-time temporal logic will be generalized to handle the probability distributions in the nondeterministic DESs. We characterize the behavior of DESs by an event structure in which an event may result in several states with a probability distribution, and then define a temporal logic model which consists of mainly a set of events and a set of logic formulas describing the enable relations of the events. After defining the syntax and semantics of the language, we present a system of rules and its soundness and completeness.

The organization of this chapter is as follows. The following section introduces an event structure for the nondeterministic DES with known probabilities. In Section 3.3, a temporal logic model is defined based on the event structure. In Section 3.4, temporal logic is generalized to include the modal operators about certainty: its language and its proof system are described; and soundness and completeness of the proof system are also discussed. Finally, the conclusion of this chapter is provided in Section 3.5.

3.2 Discrete Event Dynamical Behavior

Dynamical behavior specification of a DES consists of the tracing of occurrences of events. Presently there is a large number of models for representing systems which exhibit a discrete event behavior. Fundamental to all of these models is that the behavior of the system is described by a sequence of events. The temporal logic approach of discrete event processes concentrates on two primitive concepts: events and conditions.

In the following, it is assumed that the events of interest to us are simple and discrete. Here we use that events are simple to imply that we do not worry about what kinds of events they are and that the events of interest to us come from a set **E** of events or more strictly event occurrences.

A condition is one of the premises under which an event can occur. It is a predicate or logical description of the states and/or the inputs of the system. Such a condition may either hold or not hold. For an event to occur, it may be necessary for certain conditions to hold. These are the preconditions of the event and these preconditions must hold before the event can occur. Thus, the set of possible events which can occur at a given time moment is determined by the sequence of events which have occurred up to that time moment since some initial time point and the set of preconditions.

Events are generally assumed to occur asynchronously and instantaneously, and their effects are immediately registered. The occurrence of an event at a state

of the system may cause the preconditions to cease to hold and may result in a set of post-conditions; and therefore it changes the states of the system. Let $\mathbf{E}$ be the set of events. If for every $e \in \mathbf{E}$, the occurrence of e will result in a unique state, then the system is called a deterministic DES. Otherwise, it is called a nondeterministic DES, that is, there exists $e \in \mathbf{E}$, such that the occurrence of e will result in a set of states (more than one state). If the probability distributions of resulting in these states are known, then the system is called a nondeterministic DES with known probability distributions, or a probabilistic DES.

There is a usage of nondeterminism for the physical possibility of the occurrences of events. We will not discuss this issue in this thesis.

Let $\mathbf{S}$ denote the set of all states (state space). A state $s \in \mathbf{S}$ may be a part of preconditions of an event; and it results from another event. If the occurrence of $e \in \mathbf{E}$ changes the system from state $s \in \mathbf{S}$ to another state or a set of states, then we call this as that event e has an outcome on state s.

Definition 3.2.1: An event is said to be enabled at state s if it has an outcome on s; otherwise it said to be disabled at s.

Let E_s denote the set of events which are enabled at state s. The occurrence of an enabled event $e \in E_s$ results in a new state or some postconditions. A state s is called a terminal if no event $e \in \mathbf{E}$ is enabled at it, *i.e.* E_s is empty (it contains only the null event). Now we can define the following event structure to describe

the discrete event behavior of DESs.

Definition 3.2.2: An event structure V is defined by a 4-tuple $V = (\mathbf{S}, \mathbf{E}, f, p)$ where S is the set of states; E is a set of events; f is a set-valued mapping from $E \times S$ into S such that for all $s \in S$ and $e \in E_s$, $f(e,s)$ is defined where $f(e,s)$ may be a set of (more than one) elements with a probability distribution for the elements; p is the probability distribution function from $S \times S$ to $[0,1]$ such that $p(s,t) = 1$ if $t = f(e,s)$ is a single element and $\sum_{j \in J} p(s,t_j) = 1$ and $p_j = p(s,t_j) > 0$ if $f(e,s)$ is a set of J elements t_j, $j \in \mathbf{J} = \{1,2,...J\}$ and $\{p_1, p_2, ..., p_J\}$ is its probability distribution.

In the above definition, if for every $s \in S$ and $e \in E_s$ $f(e,s)$ is a single state, then it is a deterministic structure similar to that given in [109].

Given $e \in E_s$ for $s \in \mathbf{S}$, then $f(e,s)$ is defined. For the deterministic DES, there exists a unique $t \in \mathbf{S}$ such that $t = f(e,s)$. However, for the probabilistic DES, the set-valued mapping $f(e,s)$ may be a set which has the probability distribution p. Hence, the following result can be obtained easily from Definitions 3.2.1 and 3.2.2.

Theorem 3.2.1: For any $s \in \mathbf{S}$, an event $e \in E_s$ if and only if $f(e,s)$ is defined. If e is enabled, then there exists $s' \in \mathbf{S}$ such that $s' = f(e,s)$ for the deterministic DES or $f(e,s)$ is a set for the probabilistic DES. ■

Two sequences result from the execution of an event structure: the sequence of states $(s_0, s_1, s_2, ..., s_n)$ and the sequence of events $(e_1, e_2, ..., e_n)$, which were executed. These two sequences are related by the relationship $f(e_{k+1}, s_k) = s_{k+1}$ for $k = 0, 1, 2, ..., n-1$.

3.3 Temporal Logic Models

As defined in the event structure in the previous section, a nondeterministic DES with known point probability distributions consists of a set of events $\mathbf{E}$, and the constraints of the structure. The constraints characterize the conditions under which occurrences of events may result in different states with certain probability distributions. They may be described by a set of logical formulas, denoted by $\mathbf{F}$. A state of the system can be labelled by a subset of $\mathbf{F}$. Now we define temporal logic models (TLMs) based on the event structure defined above.

Definition 3.3.1: Given the set $\mathbf{F}$, a temporal logic model (TLM) M is a 4-tuple $(V, \mathbf{F}^*, s_0, l)$, where $V = (\mathbf{S}, \mathbf{E}, f, p)$ is an event structure; $\mathbf{F}^*$ is the set of all subsets of $\mathbf{F}$; $s_0 \in \mathbf{S}$ is the initial state; l: $\mathbf{E} \times \mathbf{S} \rightarrow \mathbf{F}^*$ is a function labelling to every pair (e, s), $e \in E_s$, a set of formulas in $\mathbf{F}^*$ that hold in (e, s).

Using Definition 3.2.2, a temporal logic model can be represented as $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, p, s_0)$ which will be used as the form of a TLM in the sequel. It is a generalization of Definitions 1 and 2 in [53]. When J is a singleton $\{j\}$ in Definition 3.3.1, $p_j = 1$. It is exactly the deterministic case. Hence, a deterministic

model is a special case of the above definition.

For the deterministic case, our temporal logic model is a slight modification of the computational model in [74]. First of all, our S is the same as their S and E the same as their T. We have the same definition of $e \in \mathbf{E}$ to be enabled as that of $\tau \in T$ in theirs; and this is where our enabled event set E_s comes from. Secondly, we will only consider computations originating in a state s_0 such that $l(s_0) \stackrel{\text{def}}{=} l(\epsilon, s_0)$ holds. Our l also has some of the properties as their partial function f_τ has. Thirdly, we have the requirements similar to theirs for an initialized computation of our temporal logic model, a sequence of states with labeled events:

$$\sigma : s_0 \stackrel{e1}{\rightarrow} s_1 \stackrel{e2}{\rightarrow} s_2 \stackrel{e3}{\rightarrow} \cdots$$

For example, *Initiality*: $l(s_0) = true$ and *state-to-state transition*: For each step $s_i \rightarrow s_{i+1}$ in σ we have $s_{i+1} = f(e_{i+1}, s_i)$. In addition, we define a finite path (or trajectory) and an infinite path (or trajectory) as follows.

Definition 3.3.2: A finite sequence $\sigma : \quad s_0 \stackrel{e1}{\rightarrow} s_1 \stackrel{e2}{\rightarrow} s_2 \cdots \stackrel{e3}{\rightarrow} s_n$ is a finite path or trajectory of a temporal logic model if $e_i \in E_{s_{i-1}}$ and $s_i = f(e_i, s_{i-1})$ for $i = 1, 2, ..., n$.

Definition 3.3.3: An infinite sequence $\sigma : \quad s_0 \stackrel{e1}{\rightarrow} s_1 \stackrel{e2}{\rightarrow} s_2 \cdots \stackrel{e3}{\rightarrow} s_n \cdots$ is an infinite path or trajectory of the system if for any n, $s_0 \stackrel{e1}{\rightarrow} s_1 \stackrel{e2}{\rightarrow} s_2 \cdots \stackrel{e3}{\rightarrow} s_n$ is a finite path.

For $\sigma : \quad s_0 s_1 s_2 \cdots s_n$, we define the length of σ by $|\sigma| = n + 1$. For

σ_1 : $s_0 s_1 \cdots s_k$ and σ_2 : $t_1 t_2 \cdots t_n$, we denote the concatenated sequence by $\sigma_1 * \sigma_2$:

$$\sigma_1 * \sigma_2 : \quad s_0 s_1 \cdots s_k t_1 t_2 \cdots t_n$$

Here $|\sigma_1 * \sigma_2| = |\sigma_1| + |\sigma_2|$. If $\sigma = \sigma_1 * \sigma_2$, we write $\sigma_1 \propto \sigma$ to represent that σ_1 is a prefix of σ (σ_1 is a head of σ if σ_1 is an element; and we write $\sigma_2 = \sigma|^{\sigma_1}$ to represent that σ_2 is obtained by deleting σ_1 from σ. In the controller design, states of the controller may be sequenced into a queue; and this queue is also a sequence of states.

For simplicity, we often write σ : $s_0 s_1 s_2 \cdots$ as a sequence of states. Denote $\mathbf{S}^*$ as the set of all (finite or infinite) sequences (or paths) over S [1]. If $\sigma \in \mathbf{S}^*$ and $n \in \mathbf{N}$ where N is the set of all positive integers, we denote $\sigma(n)$ by σ_n. Let W_s be the set of all paths that begin at s, that is $s_0 = s$. As in the theory of Markov chains [25], for any $s \in \mathbf{S}$, in a bounded stochastic model, the function p yields a probability distribution on the set W_s. This probability distribution is denoted by $\tilde{p}_s$. As mentioned in [25], it is sufficient to know that the set A of all sequences σ of $\mathbf{S}^*$, satisfying $\sigma_0 = s_0$, $\sigma_1 = s_1$,..., $\sigma_n = s_n$ ($n \geq 0$), is measurable and

$$\tilde{p}_{s_0}(A) = p(s_0, s_1) \times p(s_1, s_2) \times \cdots \times p(s_{n-1}, s_n)$$

[1] $\mathbf{S}^*$ seems non-standard and might be confusing. S^ω for infinite sequences and S^∞ for the union of finite and infinite sequences may be more standard. Since we do not use infinite sequences so much, it should be understandable.

A graphical representation of a TLM is intuitive and useful. A TLM consists of transitions (events) and states (conditions). Corresponding to these, a circle o represents a state; a directed arc for the deterministic case, or a directed tree with a probability distribution for the nondeterministic case, represents a transition. Directed arcs or trees connect the states into a graph. This graph is the state transition diagram of the TLM.

In the following, for any $\sigma \in \mathbf{S}^*$ with $\sigma = s_0 s_1 s_2 s_3 \cdots$, we denote the sequence $s_1 s_2 s_3 \cdots$ by $\sigma^{(1)}$, and the sequence $s_2 s_3 s_4 \cdots$ by $\sigma^{(2)}$, and so on. Intuitively, a state represents a snapshot of the current state of affairs and a sequence represents a possible behavior or trajectory of the system over successive instants of time.

3.4 A Generalized Temporal Logic Language

In this section, we generalize the temporal logic presented in Chapter 2 to include the modal operators about certainty adapted from [53]. After defining the syntax and semantics of the language, a proof system is outlined, and soundness and completeness are discussed.

3.4.1 The language, its syntax and semantics

As in the language given in Chapter 2, the symbols of the language include global constant symbols; local variable symbols; function letters; predicate letters; logical connectives: $\neg$ and $\vee$; temporal operators: $\bigcirc$, $\Box$, and $\mathcal{U}$; and the modal

operator: ∇ (the certainly operator, it denotes a probability one [53]).

Definition 3.4.1: The well-formed formulas (or simply, formulas) of the language are defined as those in Definition 2.2.1 and

(iv): for any formula w, (∇w) is a formula.

The formulas are evaluated with respect to the structure (M, σ) where the model $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, p, s_0)$ specifies an interpretation and σ is a trajectory, *i.e.*, a sequence of states with the initial state s_0. For the model $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, p, s_0)$ with sequence $\sigma = s_0 s_1 s_2 \cdots$, and for any integer $k \geq 0$, $M^{\sigma^{(k)}}$ represents the model $(\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, p, s_k)$ with sequence $\sigma^{(k)} = s_k s_{k+1} \cdots$, and $M^{\sigma^{(0)}} = M^\sigma$. The notations $M^\sigma[t]$, $M^\sigma[C]$, $M^\sigma[x] \in l(s_0)$, f^{M^σ}, and P^{M^σ} are used in a way similar to $M[t]$, $M[C]$, $M[x]$, f^M, and P^M in Chapter 2, respectively.

Definition 3.4.2: The satisfaction of a formula w by a model M with a sequence σ, written by $\models^\sigma_M w$, is defined for $\models^\sigma_M P(t_1, t_2, ..., t_n)$, $\models^\sigma_M (\neg w)$, $\models^\sigma_M (v \vee w)$, $\models^\sigma_M (\bigcirc w)$, $\models^\sigma_M (\Box w)$, $\models^\sigma_M (v \mathcal{U} w)$ in a way similar to $\models_M P(t_1, t_2, ..., t_n)$, $\models_M (\neg w)$, $\models_M (v \vee w)$, $\models_M (\bigcirc w)$, $\models_M (\Box w)$, $\models_M (v \mathcal{U} w)$ in Definition 2.2.2, respectively. In addition

(vii). $\models^\sigma_M (\nabla w)$ if and only if $\tilde{p}_{s_0}(\{\tau | \tau \in W_{s_0} \text{ and } \models^\tau_M w\}) = 1$ where W_{s_0} is the set of all paths of $\mathbf{S}^*$ that begin at s_0.

In addition to the abbreviations listed in Chapter 2, we define

(vi). possible operator: (Δw) by $(\neg(\nabla(\neg w)))$.

The intuitive interpretation of (∇w) has the English paraphrase, 'w will be true with probability one' while the paraphrase of (Δw) is, 'w will be possibly true'.

Definition 3.4.3: A formula w is satisfied by a model $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, p, s_0)$, written by $\models_M w$, if $\tilde{p}_{s_0}(w) \stackrel{\text{def}}{=} \tilde{p}_{s_0}(\{\tau | \tau \in W_{s_0} \text{ and } \models^{\tau}_M w\}) = 1$ where W_{s_0} is the set of all paths of $\mathbf{S}^*$ that begin at s_0.

Proposition 3.4.1: Let w be a formula and M be a model. Then

$$\models_M w \Leftrightarrow \models_M \nabla w$$

Proof: By Definition 3.4.3, $\models_M \nabla w \Leftrightarrow \tilde{p}_{s_0}(\{\tau | \tau \in W_{s_0} \text{ and } \models^{\tau}_M (\nabla w)\}) = 1$. It follows from (vii) of Definition 3.4.2 that $\models^{\tau}_M (\nabla w)$ if and only if $\tilde{p}_{s_0}(\{t | t \in W_{s_0} \text{ and } \models^{t}_M w\}) = 1$. Hence, $\models_M \nabla w \Leftrightarrow \tilde{p}_{s_0}(\{\tau | \tau \in W_{s_0} \text{ and } \models^{\tau}_M w\}) \tilde{p}_{s_0}(\{t | t \in W_{s_0} \text{ and } \models^{t}_M w\}) = [\tilde{p}_{s_0}(\{\tau | \tau \in W_{s_0} \text{ and } \models^{\tau}_M w\})]^2 = 1$, that is, $\tilde{p}_{s_0}(w) \stackrel{\text{def}}{=} \tilde{p}_{s_0}(\{\tau | \tau \in W_{s_0} \text{ and } \models^{\tau}_M w\}) = 1$. By Definition 3.4.3, this is $\models_M w$. ■

Definition 3.4.4: A formula w is said to be valid, written $\models w$, if it is satisfied by all models.

3.4.2 The proof system

The proof system is obtained by enhancing the general part of the proof system in Chapter 2 by introducing the axioms for the certainty operators; and it

is best viewed as composed of a number of levels.

The first level

The first level deals only with the formal linear-time temporal logic as given in Section 2.3 of Chapter 2.

The second level

The second level concerns general truths about certainty as follows:

Axiom schemata

(A14) $\nabla(w_1 \Rightarrow w_2) \Rightarrow (\nabla w_1 \Rightarrow \nabla w_2)$;

(A15) $\Delta \nabla w \Leftrightarrow \nabla w$;

(A16) $\nabla w \Rightarrow w$.

Generalization rule:

If $\vdash w$, then $\vdash \nabla w$.

This level amounts to the model system $LPC + S5$ in [37]. This system is well-known and well suited for the certainty notions of Proposition 3.4.1 that there is no difference between satisfaction and satisfaction with probability one.

The third level

The last level describes the interrelation between time and chance, as found in [53].

(A17) $\nabla \bigcirc w \Rightarrow \bigcirc \nabla w$

It says that the passing of time can only reduce the span of the possible.

(A18) $\Box\Diamond \triangle [\wedge_{i=0}^{k} \bigcirc^{(i)} \nabla w_i] \Rightarrow \Diamond[\wedge_{i=0}^{k} \bigcirc^{(i)} \nabla w_i]$

which is equivalent to

$$\Box\Diamond(\nabla w_0 \wedge \triangle \bigcirc (\nabla w_1 \wedge \triangle \bigcirc (\nabla w_2 \wedge \triangle \bigcirc (\cdots \wedge \triangle \bigcirc \nabla w_k)\cdots)))) \Rightarrow$$
$$\Diamond(w_0 \wedge \bigcirc w_1 \wedge \bigcirc \bigcirc w_2 \wedge \cdots \wedge \bigcirc^{(k)} w_k)$$

It expresses the fact that successive random draws are independent.

This completes the descriptions of our language and proof system.

3.4.3 Soundness and completeness

The soundness and completeness of the above-mentioned logical system are given by the following soundness theorem and completeness theorem.

Theorem 3.4.2 (Soundness Theorem): The proof system is sound and strongly sound.

Proof: By Definition 2.4.1, if the proof system is sound, then $\vdash w \Rightarrow \models w$ for all w. For any Γ and w, we have $\vdash w \Rightarrow \Gamma \vdash w$ and $p(\neg\Gamma \vee w) \geq p(w)$. By using $\models w \Leftrightarrow p(w) = 1$, we obtain that $\models w \Rightarrow (\models \neg\Gamma \vee w) \Leftrightarrow (\models \Gamma \Rightarrow w) \Leftrightarrow (\Gamma \models w)$. Hence $\Gamma \vdash w \Rightarrow \Gamma \models w$. That is, if the proof system is sound then it is strongly sound. Therefore, on the assumed derivation of w from the set of formulas Γ, it is sufficient to prove that w is valid.

It is obvious that axioms (A0)-(A13) hold for all models. For (A14), noticing that $\tilde{p}_{s_0}(w) = 1 \Leftrightarrow \tilde{p}_{s_0}(\nabla w) = 1$ for any w, we have $\tilde{p}_{s_0}(w_2) \geq \tilde{p}_{s_0}(w_1 \wedge (w_1 \Rightarrow w_2)) = 1$ since that $\tilde{p}_{s_0}(\nabla(w_1 \Rightarrow w_2)) = 1$ and $\tilde{p}_{s_0}(w_1) = 1$. Therefore, (A14) holds. (A15) holds for all models because $\tilde{p}_{s_0}(\Delta \nabla w) = 1 \Leftrightarrow \tilde{p}_{s_0}(\nabla \neg \nabla w) = 0 \Leftrightarrow \tilde{p}_{s_0}(\neg \nabla w) \neq 1 \Leftrightarrow \tilde{p}_{s_0}(\nabla w) \neq 0 \Leftrightarrow \tilde{p}_{s_0}(\nabla w) = 1$ since that $\tilde{p}_{s_0}(\nabla w)$ and $\tilde{p}_{s_0}(\Delta w)$ may only be 0 or 1. For (A16), noticing that $\tilde{p}_{s_0}(\neg \nabla w \vee w) \geq \tilde{p}_{s_0}(\neg \nabla w)$ and $\tilde{p}_{s_0}(\neg \nabla w \vee w) \geq \tilde{p}_{s_0}(w)$, we obtain $\tilde{p}_{s_0}(\nabla w \Rightarrow w) = \tilde{p}_{s_0}(\neg \nabla w \vee w) = 1$ since one of $\tilde{p}_{s_0}(\neg \nabla w)$ and $\tilde{p}_{s_0}(w)$ must be 1. For (A17), $\tilde{p}_{s_0}(\bigcirc w) = \Sigma_{s_1 \in S} p(s_0, s_1) \tilde{p}_{s_1}(w) = 1$ since $\tilde{p}_{s_0}(\nabla \bigcirc w) = 1$. It follows that $\tilde{p}_{s_1}(w) = 1$. Therefore $\tilde{p}_{s_1}(\nabla w) = 1$. Thus, (A17) holds. For (A18), let $w = \nabla w_0 \wedge \Delta \bigcirc (\nabla w_1 \wedge \Delta \bigcirc (\nabla w_2 \wedge \Delta \bigcirc (\cdots \wedge \Delta \bigcirc \nabla w_k)))$, then w is a state formula and there are states $s_0, s_1, \cdots, s_k$, such that the formula Δw_i holds at s_i for any $i \leq k$ and $p(s_i, s_{i+1}) > 0$ for $i < k$. Since M is bounded, the sequence $s_0, s_1, \cdots, s_k$ has a bounded probability $p \geq \alpha > 0$. Let $v = w_0 \wedge \bigcirc w_1 \wedge \bigcirc \bigcirc w_2 \wedge \cdots \wedge \bigcirc^{(k)} w_k$ and assume $\tilde{p}_{s_0}(\Box\Diamond w) = 1$, then $\tilde{p}_{s_0}(\{\sigma_s \mid \sigma_s \in W_{s_0}, \models_M^{\sigma_s} v\}) \geq \alpha^k > 0$ where σ_s is the trajectory that begins by $s_0, s_1, \cdots, s_k$. For any trajectory σ that does not satisfy (A18), σ must have an infinite number of states and never takes the sequence $s_0, s_1, \cdots, s_k$. Therefore,

$\tilde{p}_{s_0}(\{\sigma \mid \sigma \in W_{s_0},\ \models^{\sigma}_{M} v\}) = 0$. Hence, (A18) holds. Since the proof rules preserve validity, [72], the proof is complete. ■

Theorem 3.4.3 (Completeness Theorem): The proof system is complete but not strongly complete.

Proof: The proof system is complete as proved in [53]. However, it is not strongly complete. For example [105], assume that the formula v contains no temporal operators, then the formula $\Box v$ is satisfied by any model that satisfies the set of formulas

$$\Gamma = \{v,\ (\bigcirc v),\ (\bigcirc(\bigcirc v)),\ \cdots,\ \}.$$

But, $\Gamma \not\vdash v$ since the $\Box$ insertion rule $\frac{v}{\Box v}$ is omitted. ■

The "certainty" operators ∇ and Δ have been included in our system and in that of [53] in a slight different way. Our system is obtained by including the modal operators into the system of [105] while the system in [53] was a strict extension of the system described in [93]. The system of [105] was adapted from [72] and that in [93] was an early version of [72]. The distinction between our system and that of [53] is the same as that between the system in [105] and that in [72]. The former is not strongly complete while the latter is not strongly sound. The inclusion of the model operators makes the logic framework introduced in this chapter different from those in [105] (or [72]) and also from the system used in [53] where only propositional logic context was discussed.

As long as our proof system is sound, it suffices to show that the formal expressions of the desired properties can be deduced within our proof system. In the next chapter, this framework will be used for the specification and verification of the control problem of the nondeterministic DES with the known probability distributions.

3.5 Conclusion

In this chapter, an event structure has been defined to describe discrete event behavior of the nondeterministic DES with probability distribution known or probabilistic DES. Based on such an event structure, a temporal logic model has then been defined. After that, a generalized temporal logic, which includes the modal operators about certainty, has been developed for reasoning about the probabilistic DES. It has been noted that the deterministic model is a special case of temporal logic models, and therefore the results of this chapter are generalizations of those given by Manna and Pnueli in [72,74].

We have described a system of proof rules. Soundness and completeness of the proof system have been discussed and the system has been related to those in [53,72,105].

As the one of the most important applications of this generalized temporal logic, we will apply it in the next chapter to the specification and verification of the properties for the control problem of a class of nondeterministic DESs where

the point probability distributions are known.

Chapter 4

Specification and Verification of DESs in the Generalized Temporal Logic

In this chapter, the generalized temporal logic developed in Chapter 3 is applied to the specification and verification of the properties for the control problem of a class of nondeterministic discrete event systems where point probability distributions are known.

4.1 Introduction

The temporal logic presented in Chapter 3 includes the certainty operators to express the probabilistic properties. As an application of this generalized temporal logic, a class of nondeterministic DES, where the point probability distributions are known, will be investigated in this chapter. We show that the properties of the nondeterministic discrete event systems with known probabilities can be ver-

ified by deducing the temporal logic specifications of the desired behaviors from descriptions of the system dynamics. The verifications can then be completed qualitatively without use of the probability theory.

The organization of this chapter is as follows. In Section 4.2, the general forms of the specifications using the temporal logic are given for the plant, the closed-loop system, and the controller of a DES, respectively. Then a general procedure for verification is proposed. In Section 4.3, the verification is illustrated by an example of flexible manufacturing systems. Finally in Section 4.4, this chapter is concluded by an appraisal of the usefulness of the generalized temporal logic and some comparisons.

4.2 Specification and Verification of DESs

In this section, the generalized temporal logic is used to verify the properties of the nondeterministic DES with known point probability distributions. The problem of verifying a property involves the description of the relationship between events and states of the control system. In other words, the verification of a property requires statements about the way the variables change during the executions of events. For instance, one must sometimes assert that some condition will remain true through each execution of system events; a statement of this type is called a 'safety' assertion. An example of such an assertion is a statement of 'mutual exclusion': if several processes share a resource, then it requires that, at any time, no more than one process be using that resource. In the following,

we will give the general models of specifications for the nondeterministic DES by using the generalized temporal logic, including the plant, the closed-loop system, and the controller design.

4.2.1 Plant specifications

The plant of a DES can be represented by a TLM plant which itself is composed of a number of TLMs representing the plant processes. The global constant symbols, A, B, $C, \cdots$, are used to represent the possible state values of the system plant, and the local variable symbols x_1, x_2, $x_3, \cdots, x_N$ to represent the states of the plant (here we assume that the number of states is finite to avoid the use of quantification). In the case where some of the $x_i's$ are integers, they are assigned the values of integers instead of the global constant symbols. As described above, the global constant symbols are independent of time, and the local variable symbols vary with time.

The first specification of a plant is the set of possible events that can occur in the system. TLMs are used to describe processes by specifying the state changes of their enabled events. In each process, a state of a TLM corresponds to the execution of exactly one event. To reflect the fact that in the sequence of event occurrences executed by each process, only one event occurs at a time, we systematically add to our specifications for each process in the TLM the following:

$$\Box\{[\vee_{1\leq i\leq n} e_i] \wedge [\wedge_{1\leq i<j\leq n} \neg(e_i \wedge e_j)]\}$$

where n is the number of the enabled events for the process, and $\neg$ is the logical

negation.

The basic specifications of a plant will describe the state changes caused by occurrences of events. A general model has the following form:

$$\Box[\delta = \alpha_i \Rightarrow g(x_i, \bigcirc x_i) \wedge (\wedge_{j \neq i}(\bigcirc x_j) = x_j)]$$

which means that the occurrence of event α_i causes changes in state x_i. Here, g is a function of operators and connectives, and describes the way in which x_i changes. In the deterministic DES, there is only one next state, say $(\bigcirc x_i) = B$, of the present state $x_i = P$. Then, the function $g(x_i, \bigcirc x_i)$ is as follows:

$$x_i = P \wedge (\bigcirc x_i) = B$$

In the probabilistic DES, there is more than one next state, say $(\bigcirc x_i) = B$ with probability p or $(\bigcirc x_i) = C$ with probability $1 - p$ where $0 < p < 1$, of the present state $x_i = P$. Then, the function $g(x_i, \bigcirc x_i)$ is as follows:

$$x_i = P \wedge [(\bigcirc x_i) = B \vee (\bigcirc x_i) = C] \wedge \Delta(\bigcirc x_i) = B \wedge \Delta(\bigcirc x_i) = C$$

This formula gives a general form on how a probability propagates in the rules written in temporal logic. It describes that the state s changes from the present value P to one possible value B of the next state s' with a probability p or to another possible value C of s' with a probability $1 - p$. Similarly, we also can specify the case having more than two possible values with their probabilities. In these probabilistic cases, we assumed that the processes operate an infinite number of times. This is called impartiality [53].

In addition to these basic specifications of a plant, one should also specify all the distinct possible state values and the initial states of the system.

4.2.2 Closed-loop system specifications

The closed-loop system specifications describe the desired behavior of system dynamics. These may include 'mutual exclusion' which states that two physical objects may not occupy the same place at the same time, 'priority' which states that a particular event always be prevented from occurring under certain conditions or an event must occur under a particular condition, and 'precedence' relation which states that events will occur in some particular order. These properties usually can be precisely described in temporal logic although it may be cumbersome in the extreme. This aspect will be discussed later.

4.2.3 Controller design

As in the conventional control theory, the controller itself is a deterministic plant. In the design of a controller for a nondeterministic DES, the objective is to ensure that the requirements of the desired system behavior are met. Thus, the controller must ensure that the properties such as 'mutual exclusion', 'priority', and 'precedence' relation are satisfied.

The controller has the same set of events as the plant. The states of the controller are represented by local variable symbols and used to store the data

for the controller. In order to monitor and control the system, the states of the controller consist of queues q's and/or counts c's. Queues are assigned values that are either some finite sequence of integers between 1 and N, or the empty sequence denoted by the symbol ϕ, and counts are assigned non-negative integer values. Thus, the basic specifications of the controller have the following general form:

$$\Box[\delta = \alpha_i \Rightarrow h_1(q, \bigcirc q) \wedge h_2(c, \bigcirc c)]$$

which means that every occurrence of events causes a change of the queue contents or the values of the counts. Here, h_1 and h_2 are functions of operators and connectives, and describe the way in which q and c change, respectively. The function h_1 has the following forms:

$$(\bigcirc q) = q * i,\ i \propto q,\ (\bigcirc q) = q|^1,\ (\bigcirc q) = q, \text{ and } q = \epsilon$$

where $*$ represents the concatenation of sequences and $q * i$ means that i is pushed onto the tail of queue q; $i \propto q$ means that i is the initial element of q; $q|^1$ represents the sequence obtained by deleting the first element of q. The function h_2 has the following forms:

$$(\bigcirc c) = c,\ (\bigcirc c) = c + 1,\ (\bigcirc c) = c - 1,\ c = 0$$

Besides these basic specifications, one should also specify the initial states of the controller.

The operation of the controller for the nondeterministic DES is the modification of the queue contents and the values of the counts such that the controller can ensure the properties of the 'precedence', 'priority', and 'mutual exclusion'

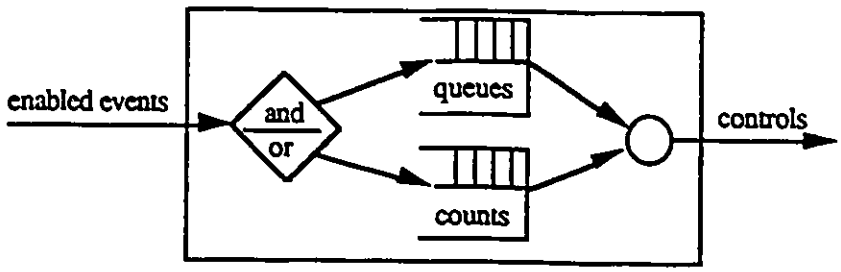


Figure 4.1: The structure of the controller.

by queues and/or counts. In this way, the controller can monitor and control the system, and ensure that the closed-loop system has the desired properties. The structure of the controller with a queue and/or a count is shown in Figure 4.1.

The controller can prevent an event from occurring under certain conditions, but can not do anything without these conditions. Specifically, the controller can stop the stochastic mechanism of the nondeterministic DES under a particular condition, but can not do anything to the stochastic mechanism if this condition does not hold. These will be made clearer by an example in the next section.

4.2.4 Verification procedure

After giving the specifications of the plant, closed-loop system, and controller, we obtain a diagram of the closed-loop system as shown in Figure 2.1. Now, we have to prove that the controller ensures the desired properties of the control system. This is equivalent to verifying the closed-loop system specifications from the

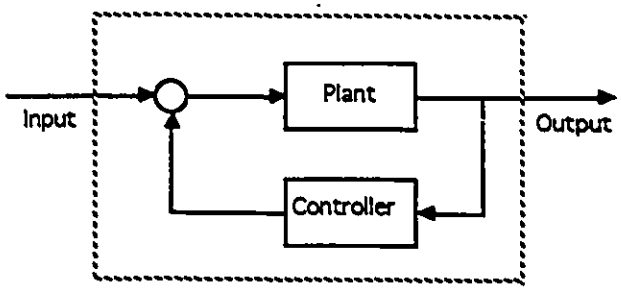


Figure 4.2: A diagram of the closed-loop system.

plant and controller specifications.

In the following, we will give a general procedure for the verification of the properties for control problems of a DES based on its plant and controller specifications.

Procedure 4.2.1:

(*i*). Specify the plant TLM of the DES. The specification is presented by the formulas of the plant dynamics and the initial conditions.

(*ii*). Describe the required behavior of the closed-loop system by temporal logic formulas which may include 'mutual exclusion', 'priority' and 'precedence' assertions.

(*iii*). Specify the controller TLM that has the same set of events as the plant and the specification is described by formulas giving the controller dynamics and initial conditions.

(*iv*). Verify that the description of the required behavior of the closed-loop

system can be deduced from the specification of the plant and the specification of the controller.

If the verification Step (*iv*) fails, one has to go to Step (*iii*) to redesign the controller and then continue to Step (*iv*) until the verification succeeds. Once the verification succeeds, it has verified that the controller ensures the desired properties of the control system.

This section has described the general forms of temporal logic formulas for the formal specification and verification of the control problems of the nondeterministic DES. As the deterministic DES in [105] is a special case of the nondeterministic DES, the above results can also be applied to the DES given in [105] where such a general description of the specifications and such a procedure for verification of properties for the control problems of DESs were not given.

4.3 An Example of Verification

As an example of verification, we consider an automated workcenter [57,61] consisting of two machines which can process parts of two different classes C_1 and C_2. These parts must be processed according to the following rules:

R1. Every part of both C_1 and C_2 has to visit both the machines, m_1 and m_2, no matter which machine can be visited first;

R2. The parts are to be processed in mutual exclusion: two parts can not be processed on the same machine at the same time. In other words, parts are processed one at a time on each machine;

R3. The parts have the following distributions to access machines: Parts of C_1 can visit machine m_1 with probability p_{11} and machine m_2 with probability p_{12}; Parts of C_2 can visit machine m_1 with probability p_{21} and machine m_2 with probability p_{22}.

In order to obtain the desired behavior of the system, our controller must ensure that these requirements are met However, the controller is not supplied with complete information about the parts. The controller checks the parts when they leave the machines. If a part leaves the machine m_i without being marked, then it will be marked and sent to machine m_j, $j \neq i$, $i, j = 1, 2$, by the controller. If a part leaves a machine with a mark on it, then it is sent outside the workcenter.

After parts arrive at the workcenter, they will be waiting for a stochastic choice made by the specific machine to process them for the first time. A part which has been processed for the first time on a machine is called a pf part. The accesses of pf parts to the second machine are controlled by the controller. One first-in-first-out queue should be maintained for each machine in order to keep track of pf parts and the order they come from the first machine. Also, counts should be set to monitor the processing of the non-pf parts on machines and to record the number of them inside the workcenter.

The controller can only reach a part which is a pf part and send it onto the tails of the queues for the other machine, but can not do anything to the part before it is a pf part. The controller can stop a machine making the stochastic choice for getting a non-pf part and can send a pf part to the machine if the queue of the machine is not empty, but can not do anything to the stochastic choices of machines if both queues are empty. This is the stochastic feature of our nondeterministic discrete event system.

This example can be viewed as a realistic model for the nondeterministic automaton G_2 of example 1.1 in [14]. It shows that nondeterministic discrete event systems are better models for flexible manufacturing systems than the deterministic ones.

In order to outline a formal proof of the fact that the controller ensures the satisfaction of the desired properties of the control problem, we first write down a temporal logic description of the plant.

4.3.1 Plant specifications

The global constant symbols O, W, R, Q, C, and D will represent the six possible states of the K parts of class C_1 and the $N-K$ parts of class C_2: those that are outside the workcenter, waiting to be processed for the first time, being processed at machine m_1, being processed at machine m_2, waiting to be sent to m_2 after being processed at m_1, and waiting to be sent to m_1 after being processed

at m_2, respectively.

The local variable symbols x_1, x_2, $\cdots$, x_N will be used to represent the states of the N parts (including the K parts of class C_1 and the $N-K$ parts of class C_2). These symbols are of the same sort as the global constant symbols O, W, R, Q, C and D.

For any i, $0 < i \leq N$, the event symbols α_i, β_i, ρ_i, γ_i, μ_i, ν_i, λ_i and ω_i represent the arrival, commencement of processing on the first machine (including the stochastic choice of machines), queue of pf parts to machine m_2, queue of pf parts to machine m_1, commencement of processing on the second machine m_2, commencement of processing on the second machine m_1, departure from machine m_1, and departure from machine m_2, respectively.

The state transition diagram of the example is shown in Figure 4.3. The descriptions of the dynamics of the plant are as follows:

$$\Box[\vee_{i=1}^{N}(\delta = \alpha_i \vee \delta = \beta_i \vee \delta = \rho_i \vee \delta = \gamma_i \vee \delta = \mu_i$$

$$\vee \delta = \nu_i \vee \delta = \lambda_i \vee \delta = \omega_i \vee \delta = \epsilon)] \qquad (4.3.P1)$$

This formula simply describes the set of possible events; it means that the only events that can occur are those listed above. The formula

$$\Box\{\wedge_{i=1}^{N}[\delta = \alpha_i \Rightarrow x_i = O \wedge (\bigcirc x_i) = W \wedge (\wedge_{i \neq j=1}^{N}(\bigcirc x_j) = x_j)]\} \qquad (4.3.P2)$$

describes the effect of the arrival of part i: its state changes from O to W, while

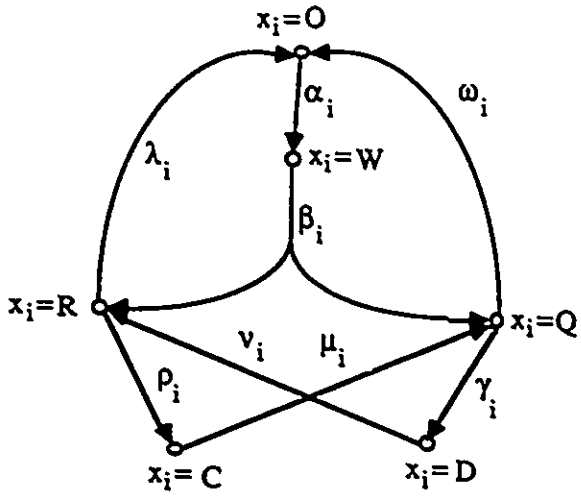


Figure 4.3: The state transition diagram of the example.

the states of other parts remain the same.

$$\Box\{\wedge_{i=1}^{N}[\delta = \beta_i \Rightarrow x_i = W \wedge [(\bigcirc x_i) = R \vee (\bigcirc x_i) = Q] \wedge$$

$$\Delta(\bigcirc x_i) = R \wedge \Delta(\bigcirc x_i) = Q \wedge (\wedge_{i \neq j=1}^{N}(\bigcirc x_j) = x_j)]\} \qquad (4.3.P3)$$

This formula describes the possible changes of states by the stochastic choices of machines; it says that, after event β_i, one of two things must occur: the next state of x_i is either R (on machine m_1) with a probability p where $0 < p < 1$, or Q (on machine m_2) with a probability $1 - p$. It is a nontrivial probabilistic statement.

$$\Box\{\wedge_{i=1}^{N}[\delta = \rho_i \Rightarrow x_i = R \wedge (\bigcirc x_i) = C \wedge (\wedge_{i \neq j=1}^{N}(\bigcirc x_j) = x_j)]\} \qquad (4.3.P4)$$

$$\Box\{\wedge_{i=1}^{N}[\delta = \gamma_i \Rightarrow x_i = Q \wedge (\bigcirc x_i) = D \wedge (\wedge_{i \neq j=1}^{N}(\bigcirc x_j) = x_j)]\} \qquad (4.3.P5)$$

$$\Box\{\wedge_{i=1}^{N}[\delta = \mu_i \Rightarrow x_i = C \wedge (\bigcirc x_i) = Q \wedge (\wedge_{i \neq j=1}^{N}(\bigcirc x_j) = x_j)]\} \qquad (4.3.P6)$$

$$\Box\{\wedge_{i=1}^{N}[\delta = \nu_i \Rightarrow x_i = D \wedge (\bigcirc x_i) = R \wedge (\wedge_{i \neq j=1}^{N}(\bigcirc x_j) = x_j)]\} \qquad (4.3.P7)$$

$$\Box\{\wedge_{i=1}^{N}[\delta = \lambda_i \Rightarrow x_i = R \wedge (\bigcirc x_i) = O \wedge (\wedge_{i\neq j=1}^{N}(\bigcirc x_j) = x_j)]\} \qquad (4.3.P8)$$

$$\Box\{\wedge_{i=1}^{N}[\delta = \omega_i \Rightarrow x_i = Q \wedge (\bigcirc x_i) = O \wedge (\wedge_{i\neq j=1}^{N}(\bigcirc x_j) = x_j)]\} \qquad (4.3.P9)$$

(4.3.P4), (4.3.P5), (4.3.P6), (4.3.P7), (4.3.P8), and (4.3.P9) are similar to (4.3.P2), describing the effects of the queue at machine m_2, queue at machine m_1, commencement of processing on the second machine m_2, commencement of processing on the second machine m_1, departure from machine m_1, and departure from machine m_2, respectively.

$$\Box[O \neq W \wedge O \neq R \wedge O \neq Q \wedge O \neq C \wedge O \neq D$$

$$\wedge W \neq R \wedge W \neq Q \wedge W \neq C \wedge W \neq D \wedge R \neq Q$$

$$\wedge R \neq C \wedge R \neq D \wedge Q \neq C \wedge Q \neq D \wedge C \neq D] \qquad (4.3.P10)$$

This means that the six possible states of a part are distinct.

$$\Box\Diamond \wedge_{i=1}^{N} [x_i = O \wedge \bigcirc x_i = W \vee x_i = W \wedge \Delta \bigcirc x_i = R \wedge \Delta \bigcirc x_i = Q \vee$$

$$x_i = R \wedge \bigcirc x_i = C \vee x_i = Q \wedge \bigcirc x_i = D \vee x_i = C \wedge \bigcirc x_i = Q \vee$$

$$x_i = D \wedge \bigcirc x_i = R \vee x_i = R \wedge \bigcirc x_i = O \vee x_i = Q \wedge \bigcirc x_i = O] \qquad (4.3.P11)$$

This formula expresses the assumption of impartiality: the processes operate an infinite number of times. This assumption is nontrivial and gives a fair condition for the probabilistic processes (those which have the possible operator Δ).

$$\wedge_{i=1}^{N} x_i = O \qquad (4.3.P12)$$

It means that initially there are no parts of either class at the workcenter.

(4.3.$P1$) – (4.3.$P12$) are the descriptions of the plant. Expressions of the desired properties of the closed-loop system are formally given below.

4.3.2 Closed-loop system specifications

$$\Box[\wedge_{i=1}^{N}(x_i = R \Rightarrow \wedge_{i\neq j=1}^{N} x_j \neq R)] \qquad (4.3.CL1)$$

$$\Box[\wedge_{i=1}^{N}(x_i = Q \Rightarrow \wedge_{i\neq j=1}^{N} x_j \neq Q)] \qquad (4.3.CL2)$$

(4.3.$CL1$) and (4.3.$CL2$) are the statements of the mutual exclusion property: two parts are not processed concurrently on the same machine.

$$\Box[\wedge_{j=1}^{N}(\vee_{i=1}^{N}(x_i = C \vee x_i = D) \Rightarrow \delta \neq \beta_j)] \qquad (4.3.CL3)$$

This means that the stochastic choices are not made if there are pf parts waiting for being processed on the second machine.

$$\Box[\wedge_{i,j=1;i\neq j}^{N}(x_i \neq C \Rightarrow (\delta = \rho_i \mathcal{P}\delta = \rho_j \Rightarrow \delta = \mu_i \mathcal{P}\delta = \mu_j))] \qquad (4.3.CL4)$$

$$\Box[\wedge_{i,j=1;i\neq j}^{N}(x_i \neq D \Rightarrow (\delta = \gamma_i \mathcal{P}\delta = \gamma_j \Rightarrow \delta = \nu_i \mathcal{P}\delta = \nu_j))] \qquad (4.3.CL5)$$

These express that pf parts are processed in the order in which they arrive: if the next arrival of part i precedes the next arrival of part j, then part i will be sent before part j, as long as part j is not already in the queues, waiting to be processed.

$$\Box[(\wedge_{i=1}^{N}(x_i = O \vee x_i = W)) \wedge \neg(\wedge_{k=1}^{N} x_k = O) \Rightarrow \vee_{j=1}^{N} \delta = \beta_j] \qquad (4.3.CL6)$$

If all parts are either outside the workcenter or waiting to be processed but not all of them outside the workcenter, then one of the β_js must occur.

4.3.3 Controller specifications

We use additional local variable symbols to represent the data stored by the controller. The symbols r and q represent the queues at machines m_2 and m_1, respectively. They are assigned values that are either some finite sequence of integers between 1 and N, or the empty sequence denoted by the symbol ϕ. The symbols a, b are assigned 1 and 0 representing if a non-pf part is being processed on the machines m_1 and m_2 respectively. The symbol c represents the number of non-pf parts in the workcenter and it is assigned a non-negative integer value.

$$\Box[\wedge_{i=1}^{N}(\delta = \rho_i \Rightarrow a = 1 \wedge (\bigcirc a) = 0 \wedge (\bigcirc r) = r * i \wedge (\bigcirc b) = b \wedge (\bigcirc c) = c)] \quad (4.3.C1)$$

$$\Box[\wedge_{i=1}^{N}(\delta = \gamma_i \Rightarrow b = 1 \wedge (\bigcirc b) = 0 \wedge (\bigcirc q) = q * i \wedge (\bigcirc a) = a \wedge (\bigcirc c) = c)] \quad (4.3.C2)$$

(4.3.$C1$) ((4.3.$C2$), respectively) says that if pf part i comes from machine m_1 (m_2, respectively), the number i is pushed to the tail of queue r (q, respectively), the value of a (b, respectively) is changed from 1 to 0, and the values of b (a, respectively) and c are not changed.

$$\Box[\wedge_{i=1}^{N}(\delta = \mu_i \Rightarrow i \propto r \wedge (\bigcirc r) = r \wedge (\bigcirc a) = a \wedge (\bigcirc b) = b \wedge (\bigcirc c) = c)] \quad (4.3.C3)$$

$$\Box[\wedge_{i=1}^{N}(\delta = \nu_i \Rightarrow i \propto q \wedge (\bigcirc q) = q \wedge (\bigcirc a) = a \wedge (\bigcirc b) = b \wedge (\bigcirc c) = c)] \quad (4.3.C4)$$

(4.3.$C3$) ((4.3.$C4$), respectively) means that a pf part is sent to machine m_2 (m_1, respectively) only if it is at the head of the queue r (q, respectively) while the values of a, b, c remain the same.

$$\Box[\wedge_{i=1}^{N}(\delta = \omega_i \Rightarrow (\bigcirc r) = r|^1 \wedge (\bigcirc a) = a \wedge (\bigcirc b) = b \wedge (\bigcirc c) = c)] \quad (4.3.C5)$$

$$\Box[\wedge_{i=1}^{N}(\delta = \lambda_i \Rightarrow (\bigcirc q) = q|^1 \wedge (\bigcirc a) = a \wedge (\bigcirc b) = b \wedge (\bigcirc c) = c)] \quad (4.3.C6)$$

(4.3.$C5$) ((4.3.$C6$), respectively) says that if a part makes a departure from machine m_2 (m_1, respectively), the queue r (q, respectively) is popped and the values of a, b, c remain the same.

$$\Box\{\wedge_{i=1}^{N}(\delta=\beta_i \Rightarrow a=0 \wedge b=0 \wedge (\bigcirc c)=c-1 \wedge [(\bigcirc x_i)=R \Rightarrow (\bigcirc a)=1 \wedge (\bigcirc b)=0]$$

$$\wedge[(\bigcirc x_i)=Q \Rightarrow (\bigcirc a)=0 \wedge (\bigcirc b)=1] \wedge q=\phi \wedge (\bigcirc q)=q \wedge r=\phi \wedge (\bigcirc r)=r)\} \quad (4.3.C7)$$

(4.3.C7) means that a stochastic choice is made only if queues r and q are empty and the queue contents are not changed; and then, a or b has its value changed from 0 to 1 and the value of c is decremented by 1.

$$\Box[\wedge_{i=1}^{N}(\delta=\alpha_i \Rightarrow (\bigcirc c)=c+1 \wedge (\bigcirc a)=a \wedge (\bigcirc b)=b \wedge (\bigcirc q)=q \wedge (\bigcirc r)=r)] \quad (4.3.C8)$$

It says that if a part arrives at a workcenter, the value of the count c is incremented by 1 while the queue contents and the values of a and b remain the same.

$$a=0 \wedge b=0 \wedge c=0 \wedge r=\phi \wedge q=\phi \quad (4.3.C9)$$

It says that initially queues r and q are empty and the values of a, b and c are zero.

This completes the formal description of the controller.

4.3.4 Formal verifications

Now, we can formulate the verification of desired properties of the control problem by temporal logic, and outline a deduction of the closed-loop system

specifications (4.3.$CL1$) – (4.3.$CL6$) from the descriptions of the plant (4.3.$P1$) – (4.3.$P12$) and the controller (4.3.$C1$) – (4.3.$C9$).

To formally verify that the closed-loop system specifications (4.3.$CL1$) – (4.3.$CL6$) can be deduced from the plant specifications (4.3.$P1$) – (4.3.$P12$) and the controller specifications (4.3.$C1$) – (4.3.$C9$), we need the following lemmas.

Lemma 4.3.1: The following formulas can be deduced from (4.3.$P1$)–(4.3.$P12$) and (4.3.$C1$) – (4.3.$C9$):

$$\Box\{\wedge_{i=1}^{N}(x_i = R \Rightarrow [(i \propto q \wedge a = 0) \vee (q = \phi \wedge a = 1)])\} \qquad (4.3.L1)$$

$$\Box\{\wedge_{i=1}^{N}(x_i = Q \Rightarrow [(i \propto r \wedge b = 0) \vee (r = \phi \wedge b = 1)])\} \qquad (4.3.L2)$$

Proof of Lemma 4.3.1 (Outline):

Let w be the formula that follows $\Box$. It is obvious that w can be deduced from (4.3.$P12$) and (4.3.$C9$). Then, it suffices to show that one can deduce $\bigcirc w$ from w and $\delta = e$, for every event symbol e appearing in (4.3.$P1$). Here, we present an outline of the deduction of (4.3.$L1$) only for the case where $\delta = \beta_k$, and $\delta = \nu_k$, $0 < k \leq N$:

(1). w hypothesis

(2). $\delta = \beta_k$ hypothesis

(3). $x_k = W \wedge [(\bigcirc x_k) = R \vee (\bigcirc x_k) = Q]$
$\wedge \triangle(\bigcirc x_k) = R \wedge \triangle(\bigcirc x_k) = Q$ from (2) and (4.3.P3)

(4). $R \neq O \wedge R \neq W \wedge R \neq Q \wedge R \neq C \wedge R \neq D$ from (4.3.P10), by PR

(5). $a = 0 \wedge b = 0 \wedge [(\bigcirc x_i) = R \Rightarrow (\bigcirc a) = 1 \wedge (\bigcirc b) = 0]$
$\wedge[(\bigcirc x_i) = Q \Rightarrow (\bigcirc a) = 0 \wedge (\bigcirc b) = 1] \wedge (\bigcirc c) = c - 1$
$\wedge q = \phi \wedge (\bigcirc q) = q \wedge r = \phi \wedge (\bigcirc r) = r$ from (4.3.C7) and (2)

(6). $(\bigcirc x_i) = R \Rightarrow (\bigcirc q) = \phi \wedge (\bigcirc a) = 1$ from (3)-(5), (4.3.P11), (A17) and (A18)

(7). $\delta = \nu_k$ hypothesis

(8). $x_k = D \wedge (\bigcirc x_k) = R \wedge (\wedge_{i \neq j=1}^{N} (\bigcirc x_j) = x_j)$ from (4.3.P5) and (7)

(9). $i \propto q \wedge (\bigcirc q) = q \wedge a = 0 \wedge (\bigcirc a) = a$ from (4.3.C2), by PR

(10). $(\bigcirc x_i) = R \Rightarrow i \propto (\bigcirc q) \wedge (\bigcirc a) = 0$ from (7)-(9), (4.3.P11)

(11). $\wedge_{i=1}^{N}[\bigcirc x_i = R \Rightarrow (i \propto \bigcirc q \wedge (\bigcirc a) = 0)$
$\vee(\bigcirc q = \phi \wedge \bigcirc a = 1)]$ from (6),(10), by PR

(12). $\bigcirc w$ from (11) and by FT.

(4.3.L2) can be deduced in a similar way. ■

Lemma 4.3.2: The following formula can be deduced from (4.3.P1)−(4.3.P12) and $(C1) - (C9)$:

$$\Box[\wedge_{i=1}^{N}(x_i = O \vee x_i = W \Leftrightarrow i \notin q \wedge i \notin r)] \qquad (4.3.L3)$$

Proof of Lemma 4.3.2 (Outline):

With the same idea as that of Lemma 4.3.1, here we only give an outline of the deduction for the case where $\delta = \beta_k$, $0 < k \leq N$:

(1). $\wedge_{i=1}^{N}(x_i = O \vee x_i = W \Leftrightarrow i \notin q \wedge i \notin r)$	hypothesis
(2). $\delta = \beta_k$	hypothesis
(3). $x_k = W \wedge [(\bigcirc x_k) = R \vee (\bigcirc x_k) = Q]$ $\wedge \triangle(\bigcirc x_k) = R \wedge \triangle(\bigcirc x_k) = Q$	from (2) and (4.3.P3)
(4). $R \neq O \wedge R \neq W \wedge R \neq Q$	from (4.3.P10), by PR
(5). $a = 0 \wedge b = 0 \wedge [(\bigcirc x_i) = R \Rightarrow (\bigcirc a) = 1 \wedge (\bigcirc b) = 0]$ $\wedge[(\bigcirc x_i) = Q \Rightarrow (\bigcirc a) = 0 \wedge (\bigcirc b) = 1] \wedge (\bigcirc c) = c - 1$ $\wedge q = \phi \wedge (\bigcirc q) = q \wedge r = \phi \wedge (\bigcirc r) = r$	from (4.3.C7) and (2)
(6). $(\bigcirc x_i) = O \vee (\bigcirc x_i) = W \Leftrightarrow i \notin (\bigcirc q) \wedge i \notin (\bigcirc r)$	from (2)-(5), (4.3.P11), (A17) and (A18)
(7). $\bigcirc[\wedge_{i=1}^{N}(x_i = O \vee x_i = W \Leftrightarrow i \notin q \wedge i \notin r)]$	from (6), by PR, by FT.

In this way, the results can be deduced. ■

With the help of the above Lemmas, we can prove the following.

Theorem 4.3.3: The closed-loop system specifications (4.3.CL1) – (4.3.CL6) can be deduced from the plant specifications (4.3.P1) – (4.3.P12) and the controller specifications (4.3.C1) – (4.3.C9).

Proof of Theorem 4.3.3 (Outline):

(4.3.CL1) and (4.3.CL2) can be deduced by Lemma 4.3.1 and Generation rule. (4.3.CL1) can be deduced by Lemma 4.3.1 as follows:

(1). $x_k = R, 0 < k \leq N,$	hypothesis
(2). $\wedge_{i=1}^{N}[x_i = R \Rightarrow (i \propto q \wedge a = 0) \vee (q = \phi \wedge a = 1)]$	by Lemma 4.3.1
(3). $(k \propto q \wedge a = 0) \vee (q = \phi \wedge a = 1)$	from (1) and (2), by PR
(4). $\wedge_{k \neq j=1}^{N} x_j \neq R$	from (2),(3), by MP
(5). $x_k = R \Rightarrow \wedge_{k \neq j=1}^{N} x_j \neq R$	from(1)-(4), and by DT
(6). $\wedge_{i=1}^{N}(x_i = R \Rightarrow \wedge_{i \neq j=1}^{N} x_j \neq R)$	from (5), by PR.

Thus, (4.3.$CL1$) follows by Generation rule.

In the same way as the above, (4.3.$CL2$) can be deduced by Lemma 4.3.1.

By Lemma 4.3.2 and PR and MR, (4.3.$CL3$) can be deduced from (4.3.$P6$), (4.3.$P7$), (4.3.$P10$), (4.3.$C3$) and (4.3.$C4$). (4.3.$CL6$) follows from (4.3.$P10$) and (4.3.$C7$), by Lemma 4.3.2 and PR. (4.3.$CL4$) and (4.3.$CL5$) can be deduced by Lemmas 4.3.1 and 4.3.2 and by the $\mathcal{P}-$chain derived rule in a way similar to that of $CL3$ in [105]. ■

Remark 4.3.1: It is obvious that, from (4.3.$P3$) in the plant specifications, the discrete event system is nondeterministic with known probability distributions, and it follows from the controller specifications that the controller of the system is deterministic. This fact coincides with that in [14,95] and [57,61].

Remark 4.3.2: In the rule $R3$, the concrete probability distribution is given in the nondeterministic part of the system [57,61]. However, Lemmas 4.3.1 and

Theorem 4.3.3 are valid in the sense of probability one. Thus, our verification here does not mention probabilities explicitly except probability one. Hence, the verifications are qualitative but not quantitative [53] and have been completed without any need to use the probability theory.

4.4 Conclusion

In this chapter, the generalized temporal logic obtained in Chapter 3 has been applied to a class of nondeterministic discrete event systems where the point probability distributions are known. It has been noted that the deterministic DES is a special case of temporal logic models, and therefore the results of this chapter are generalizations of those given by Thistle and Wonham in [105].

The desired qualitative and symbolic analysis of this chapter has been given by a careful exposition of how some aspects of the nondeterministic DES, such as specifications and verification, can be studied in the generalized temporal logic framework through control concepts. We have illustrated the use of the proof system given in Chapter 3 by proving closed-loop specification formulas from plant and controller specifications. An example of the flexible manufacturing systems has been given to demonstrate the verification process using the generalized temporal logic in Chapter 3. The verifications are qualitative and have been completed without using the probability theory.

Apparently the probabilistic content is hardly seen in this chapter; and this

is because the concrete probabilities have not been written explicitly due to that our approach for the probabilistic content is qualitative but not quantitative. In fact, the probabilistic content is in the statements asserted such as Lemmas 4.3.1 and 4.3.2 and Theorem 4.3.3. Notice that the satisfaction of a formula w by a model M is defined by means of probability one (Definition 3.4.3). So is the definition of $\models w$ (valid). Particularly, formulas (4.3.P3) and (4.3.P11) are nontrivial probabilistic statements. They are used in the proofs of Lemma 4.3.1 ((1)⇒(6)) and Lemma 4.3.2. This means that Lemmas 4.3.1 and 4.3.2, and Theorem 4.3.3 are valid with probability 1 although it is not written explicitly in these results. We could put more probabilistic formulas into the system; however, that would make the system too complicated as compared with Thistle's results.

Comparing with other approaches for the nondeterministic DES, the applications of the generalized temporal logic in this chapter show that the temporal logic approach leads to easily-understandable high-level specifications and to a formal verification procedure. For example, the method of regular grammars used in automata approach [95] which has been proved useful in theoretical analysis of DESs, becomes cumbersome even for simple informal specifications. The process of finding the corresponding grammar could be difficult as mentioned in [105]. On the other hand, the translation of a natural language specification into temporal logic is more straightforward.

In the given example of the specification and verification in the temporal logic framework, we have seen the reasoning power of the generalized temporal logic is

impressive. Unfortunely, automatic proof procedures do not exist in general and they are needed to be developed for the applications in the analysis and synthesis of DESs in the temporal logic approach.

In the next chapter, we will analyze the dynamics of DESs based on temporal logic models. Following that, we will define reachability and investigate its relationship with the validity of logic formulas within a temporal logic model; and then we will develop algorithms for the reachability analysis.

Chapter 5

Reachability Analysis of DESs

In this chapter we analyze the dynamics of DESs. To see the dynamical behavior of DESs, the reachability analysis is given and related to the validity of logic formulas of the given temporal logic model.

5.1 Introduction

As stated in the previous chapters, temporal logic has been applied into many applications in DESs, particularly, the computer-related systems [28]. Currently, computational algorithms in a temporal logic approach are badly needed for applications. In other words, the motivation of this chapter is from the need to implement computational algorithms for the analysis of DESs in a temporal logic approach. In such an analysis, we need to follow the track of the computation as a trajectory progressing in time. This involves analyzing the dynamic behavior of TLMs such as reachability properties. Therefore, the main objective of the chapter is to develop procedures for the reachability analysis of DESs in a temporal

logic framework.

Perhaps reachability analysis was first mentioned for a temporal logic approach by Bochmann [9] where the definition of reachability was not given. Reachability has been defined by Ostroff [85,84] and some procedures for constructing reachability graphs have been also given for a special class of formulas in his language.

In this chapter, dynamic behavior of DESs is characterized by the mapping, driven by enabling an event, from one state to another state for the deterministic case or a set of states with their probabilities for the probabilistic case. The enable condition of events depends upon the truth of the temporal logic formulas describing the dynamics. The computation algorithm is developed for reachability analysis. One of the important results is that reachability of states is equivalent to validity of logic formulas of specifications for the system. It simplifies the system design procedures as it uses the reachability properties instead of validity properties.

This chapter is organized as follows. In Section 5.2, the dynamics of temporal logic models is discussed. In Section 5.3, the relationship between reachability of states and validity of formulas is given by the definition of reachability and the properties of TLM; and then the algorithm for computing the reachability set and constructing the reachability graph is developed. In Section 5.4, an example of data link protocols is demonstrated to illustrate the results. Finally Section 5.5

concludes this chapter with a comparison of our results with Ostroff's work.

5.2 Dynamics in Temporal Logic Models

As discussed in Chapter 3, a DES is characterized by temporal logic models which have a set of events **E**, a set of states **S**, a set of formulas **F** describing the enable relations of the events, a state mapping f, a labelling function l, a probability distribution function p, and the initial state s_0.

The dynamics of DESs is produced by the mapping f. It is convenient to extend the mapping to map a state and a sequence of events into a new state for the deterministic case or into a set of states with a probability distribution for the nondeterministic case.

Definition 5.2.1: For a set A, $f(e,A) \stackrel{\text{def}}{=} \{b \ : \ b = f(e,a), \ a \in A\}$ and the probability distribution of $f(e,A)$ is defined by that of A. Then, the state mapping is extended at state s for a sequence of events $\sigma \in E_s$ and $\forall e \in \mathbf{E}$ such that $\sigma e \in E_s$ by

$$f(\sigma e, s) \stackrel{\text{def}}{=} f(e, f(\sigma, s)) \tag{5.2.1}$$

$$f(\epsilon, s) \stackrel{\text{def}}{=} s \tag{5.2.2}$$

where ϵ is a null event; the probability distribution of A means that A has k elements s_i, $i = 1, 2, ..., k$, and a probability p_i for each element s_i (to be picked up),

respectively, and $\sum_{i=1}^{k} p_i = 1$.

For the above definition, we assume that, for $e_1 \in E_s$ and $e_2 \in \mathbf{E}$ such that $e_1 e_2 \in E_s$, we have $e_1 e_2 \in \mathbf{E}$ and $\epsilon \in \mathbf{E}$. (5.2.2) means that the state of the system is unchanged if nothing happens; and it is described in Rule (A13) in Chapter 2. For a sequence of events $e_1 e_2 \cdots e_k$, and state s, $f(e_1 e_2 \cdots e_k, s)$ is the result of enabling first e_1, then e_2, and so on, until e_k is enabled. Here, $f(\sigma e, s)$, $f(e, f(\sigma, s))$, and $f(e_1 e_2 \cdots e_k, s)$ are single state for the deterministic case, or a set of states with a probability distribution for the nondeterministic case. Generally we use this extended state mapping.

As seen in the example of Chapter 4, a TLM is described by a set of formulas. Corresponding to $f(e, s)$, the labelling function l labels every pair (e, s), $e \in E_s$, a set of formulas in $\mathbf{F}^*$. For instance, in the example of Chapter 4, in the case of $N = 2$, the labelling function l labels the pair (α_2, s_2), where $s_2 = (W, O)$, the formula (4.3.P2).

To simplify the notations in the sequel, we write the formula $l(e, s)$ which describes the dynamics of a DES in the following form:

$$\Box[s(x) \wedge e(x) \Rightarrow s'(\bigcirc x)] \tag{5.2.3}$$

for the deterministic case, or

$$\Box\{s(x) \wedge e(x) \Rightarrow [\vee_{j \in J} s_j(\bigcirc x)] \wedge [\wedge_{j \in J} \triangle\, s_j(\bigcirc x)]\} \tag{5.2.4}$$

for the probabilistic case. In both (5.2.3) and (5.2.4), x is a local variable, $s(x)$

stands for the state of x (as $x_i = W$ in (4.3.P3)), $e(x)$ for the occurrence of the event corresponding to x (as $\delta = \beta_i$ in (4.3.P3)), $s'(\bigcirc x)$ and $s_j(\bigcirc x)$, $j \in J$, for the state of x at the next time (as $(\bigcirc x_i) = R$ and $(\bigcirc x_i) = Q$ in (4.3.P3)). (4.3.P2) and (4.3.P3) can be seen as examples of (5.2.3) and (5.2.4) respectively.

The following theorem gives the enable rule for the TLM.

Theorem 5.2.1: In a TLM $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, p, s_0)$ for $s \in \mathbf{S}$, an event $e \in E_s$ (e is enabled) if and only if $l(e, s) \in \mathbf{F}^*$ and has the form of (5.2.3) for the deterministic case or (5.2.4) for the probabilistic case.

Proof: *Sufficiency.* The fact that $l(e, s) \in \mathbf{F}^*$ and has the form of (5.2.3) for the deterministic case or (5.2.4) for the probabilistic case means that e has an outcome on s. By Definition 3.3.1, e is enabled at s, *i.e.* $e \in E_s$.

Necessity. By Theorem 3.2.1, the fact of that event $e \in E_s$ (e is enabled) means that there exists $s' \in \mathbf{S}$ such that $s' = f(e, s)$ for the deterministic case and $f(e, s)$ is a set for the probabilistic case. By Definition 3.3.1, l labels (e, s) a set of formulas in $\mathbf{F}^*$ as $l(e, s)$ to describe such dynamics. Hence, $l(e, s)$ has the form of (5.2.3) for the deterministic case or (5.2.4) for the probabilistic case.■

The following result shows the transitive properties of the dynamics described by the logic formulas $l(e, s)$ in a TLM.

Theorem 5.2.2: The set of formulas (T1) and (T2)

$$\Box[s(x) \wedge e'(x) \Rightarrow s'(\bigcirc x)] \qquad (T1)$$

$$\Box[s'(x) \wedge e''(x) \Rightarrow s''(\bigcirc x)] \qquad (T2)$$

is equivalent to the formula (T3)

$$\Box[s(x) \wedge e'(x) \wedge e''(x) \Rightarrow s''(\bigcirc x)] \qquad (T3)$$

Proof: Using the assumption below Definition 5.2.1, we have $e'e'' \in E_s$. Then obviously, $(T1) \wedge (T2) \Rightarrow (T3)$. Now assume that $(T3)$ is given, then $s(x) \wedge e'(x)$ implies that $e' \in E_s$. So there exists $s' \in \mathrm{S}$ such that $(T1)$ is true. Substituting $s(x) \wedge e'(x)$ in $(T3)$ by $s'(x)$ then we obtain $(T2)$. ■

Corollary 5.2.3: The set of formulas (T4)-(T6)

$$\Box\{s(x) \wedge e(x) \Rightarrow [s_1(\bigcirc x) \vee s_2(\bigcirc x)] \wedge \Delta s_1(\bigcirc x) \wedge \Delta s_2(\bigcirc x)\} \qquad (T4)$$

$$\Box[s_1(x) \wedge e'(x) \Rightarrow s_1'(\bigcirc x)] \qquad (T5)$$

$$\Box[s_2(x) \wedge e'(x) \Rightarrow s_2'(\bigcirc x)] \qquad (T6)$$

is equivalent to the formula (T7)

$$\Box\{s(x) \wedge e(x) \wedge e'(x) \Rightarrow [s_1'(\bigcirc x) \vee s_2'(\bigcirc x)] \wedge \Delta s_1'(\bigcirc x) \wedge \Delta s_2'(\bigcirc x)\} \qquad (T7)$$

Proof: In a similar way to the proof Theorem 5.2.2, we can prove this result. ■

In the TLMs, l labels a set of formulas of $\mathbf{F}^*$ corresponding to f. Theorem 5.2.2 is the counter-part of the deterministic case while Corollary 5.2.3 is that of the probabilistic case, of (5.2.1), respectively.

5.3 Reachability Analysis of TLMs

In this section, we will define the reachability set of a DES to describe the dynamic behavior of the TLM introduced above, then find out the relationship between reachability of states and validity of formulas, and finally develop an algorithm for computing the reachability set and constructing the reachability graph.

For the deterministic case, enabling an event e in a state s results in a new state $t = f(e, s)$; and we say that t is one step reachable from s. For the nondeterministic case, the result of enabling an event e in a state s is a set $f(e, s)$ of states with their probabilities; and we say that $t \in f(e, s)$ is one step probably reachable from s. The definition is as follows.

Definition 5.3.1: For a TLM $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, p, s_0)$, a state $t \in \mathbf{S}$ is one step reachable from $s \in \mathbf{S}$ if there is an event $e \in E_s$ such that $t = f(e, s)$; and $t \in \mathbf{S}$ is one step probably reachable from $s \in \mathbf{S}$ if there is an event $e \in E_s$ such that $t \in f(e, s)$ ($f(e, s)$ is a set of states) with the probability $p(s, t)$.

By Definition 5.3.1, the set of all one step reachable states from state s is given by enabling all physically possible events in E_s for the deterministic case, and the set of all one step probably reachable states from state s is given by enabling all physically possible events in E_s with a probability distribution to reach them for the nondeterministic case. Doing this to every $s \in \mathbf{S}$, it will give all possible state transitions of the system. In the sense of seeing whether a state is possibly to be reached, the difference between one step reachable and probably reachable is not significant. Hence, to simplify the notations, we will use one step reachable for both deterministic case and the probabilistic case; and further more we will extend Definition 5.3.1 to the following definition of the reachability set $R(M, s_0)$ of M to be the set of all reachable states as follows.

Definition 5.3.2: The reachability set $R(M, s_0)$ for $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, p, s_0)$ is the smallest set defined by

- $s_0 \in R(M, s_0)$;
- If $s \in R(M, s_0)$ and $t = f(e, s)$ for some $e \in E_s$, then $t \in R(M, s_0)$.

The *smallest set* is used to ensure that all elements of the set are reachable from s_0. In the following, when we say that $s \in \mathbf{S}$ is reachable, it simply mean that $s \in R(M, s_0)$. Then reachability has the following property.

Theorem 5.3.1: The reachability relationship is transitive, *i.e.* if $s' \in \mathbf{S}$ is reachable from $s \in \mathbf{S}$ and $s'' \in \mathbf{S}$ is reachable from $s' \in \mathbf{S}$, then $s'' \in \mathbf{S}$ is reachable from $s \in \mathbf{S}$.

Proof: By Definition 5.3.1, there exist $e \in E_s$ and $e' \in E_{s'}$ such that $s' = f(e,s)$ or $s' \in f(e,s)$ and $s'' = f(e',s')$ or $s'' \in f(e,s)$. By Definitions 5.2.1 and 5.3.2, $f(e', f(e,s)) = f(ee',s)$ and $ee' \in E_s$. Hence $s'' = f(e',s') = f(ee',s)$ or $s'' \in f(ee',s)$. Therefore s'' is reachable from s. ■

Theorem 5.3.2: In a TLM $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, p, s_0)$, s' is reachable from s if and only if the dynamical formula $l(e,s) \in \mathbf{F}^*$ and has the form (5.2.3) for the deterministic case or (5.2.4) for the probabilistic case.

Proof: By Definition 5.3.1, s' is reachable from s means that there exists $e \in E_s$ such that $s' = f(e,s)$ or $s' \in f(e,s)$. By Theorem 5.2.1, this is equivalent to the dynamical formula $l(e,s) \in \mathbf{F}^*$ which has the form (5.2.3) for the deterministic case or (5.2.4) for the probabilistic case. ■

Using Theorem 5.3.2, we can establish the relationship of transition mapping f and labelling function l and it gives a relationship between reachability of states and validity of formulas in a TLM as follows.

Corollary 5.3.3: For a given TLM, the reachability of states is equivalent to validity of the corresponding dynamical formulas.

Proof: By Theorem 5.3.2, s' is reachable from s if and only if $l(e,s)$, which has the form (5.2.3) for the deterministic case or (5.2.4) for the probabilistic case,

belongs to $\mathbf{F}^*$. From Definitions 3.3.1, 3.4.2, and 3.4.3 it is equivalent to that $l(e,s)$ is valid.■

Hence we may use reachability properties of a TLM to verify the corresponding validity properties. Therefore, we may verify the specification of the plant by the reachability analysis.

The verification of the specification given by a set of temporal logic formulas in a TLM is a procedure of proving the validity of the formulas; and by Corollary 5.3.3, this can be converted to the reachability analysis of the TLM. As a computation algorithm for a TLM $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, s_0, l, p)$, the procedure for computing the reachability set and constructing the reachability graph G_M is as follows:

Algorithm 5.3.1:

Step 1: Given a temporal logic model $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, p, s_0)$. Then,

$$node(G_M) = \{s_0\}, \quad edge(G_M) = \phi$$

where ϕ is an empty set and s_0 is unmarked.

Step 2: IF there is an unmarked node s in $node(G_M)$, *THEN MARK* node s as a root and *FIND* the set E_s using the following rule. *OTHERWISE GOTO Step 5.*

For each $e \in \mathbf{E}$, *IF* $l(e,s) \in \mathbf{F}^*$, *THEN* $e \in E_s$.

Step 3: For every $e \in E_s$, *DO* $edge(G_M) := edge(G_M) \cup \{e\}$ and *FIND* $f(e,s)$. *IF* $f(e,s)$ is a single element, *THEN* $t = f(e,s)$ and $p = 1$. *OTHERWISE*, for every $t \in f(e,s)$, $p = p(s,t)$.

Step 4: IF t is unmarked, *DO* $node(G_M) := node(G_M) \cup \{t\}$. *GOTO Step 2.*

Step 5: END.

For the nondeterminstic case, the mapping f is set-valued with the probability distributions. A state which is possibly be reached is also reachable with a probability as shown in the above algorithm. Thus, the reachability set is obtained as $R(M, s_0) = node(G_M)$. It does not have duplicated elements; and this is ensured by the marking scheme used in the procedure such that no state is visited more than once. When a state is visited, it is marked as a root. Only those edges that are in the enabled event set are added to the set of edges and only those states that have not previously been marked are marked as roots and added to the reachability set.

F^* is the set of all subsets of F. Let the number of elements of F be n. Then it takes 2^n steps to get F^* by using the combination theorems. So Algorithm 5.3.1 is exponential in the size of F. At each state s, there are at most $|E_s| \leq |\mathbf{E}|$ events to be checked for successor states, where $|\mathbf{E}|$ is the number of events in $\mathbf{E}$. Hence at most $|\mathbf{E}| \cdot |R(M, s_0)|$ steps are needed to compute the reachability graph and the reachability set.

Let $G_M = (node(G_M), edge(G_M))$, and let $|node(G_M)|$ and $|edge(G_M)|$ be the number of nodes in $node(G_M)$ and the number of edges in $edge(G_M)$, respectively. An algorithm is said to be linear in the size of the graph G_M if the algorithm takes no more than $order(|node(G_M)| + |edge(G_M)|)$ steps. Since at

most $|edge(G_M)| = |\mathrm{E}| \cdot |R(M, s_0)|$ and $node(G_M) = R(M, s_0)$, $|\mathrm{E}| \cdot |R(M, s_0)| + |R(M, s_0)| = (|\mathrm{E}| + 1)|R(M, s_0)|$. This means that algorithm takes no more than $order(|node(G_M)| + |edge(G_M)|)$ steps. Hence it is linear in the size of the reachability graph.

The procedure of this algorithm is implemented in the software which will be discussed in Chapter 8.

The above reachability algorithm is suggested for use in verifying the specification given by temporal logic models. This has a great significance, particularly, when the verification is carried out by a computer which needs automatic verification procedure. According to Corollary 5.3.3, the reachability algorithm can be used as an automatic method for doing the verification.

In applications, the verification of concurrent programs, operating systems, and communication protocols requires statements about the way variables change during execution. The statements can be a *safety* (*invariance* in the terminology of [72]) assertion or a *liveness* (*eventualities* in the terminology of [72]) assertion. Obviously reachability algorithm can be used to verify a safety assertion since the reachable states of the system will remain within the reachability set.

The algorithm can also be used to verify the specification that particular events occur in a certain order. Such a specification could be either a liveness or a safety assertion, depending on whether it is to mean that the events actually will

occur or simply that, if they do occur, they will do so in the proper sequence. As an example, we will use this algorithm, in the next section, to verify the correctness of a data link protocol which specifies the order in which events take place in the communication network.

5.4 An Example of Application

As an example of a system which has a discrete event behavior, consider a data link layer protocol [31] which consists of a sender sending packets and a receiver receiving packets through a communication link. The protocol is the alternating bit protocol (ABP), for the packets 0 and 1 [107]. The state of the protocol is $s = (x, y)$ where the sender x can be in one of three states:

- $0(x)$ waiting for an acknowledgement $ACK0$, for the packet 0;
- $1(x)$ waiting for an acknowledgement $ACK1$, for the packet 1; and
- $n(x)$ no acknowledged packet.

and the receiver y can be in one of two states:

- $0(y)$ waiting for a packet numbered 0; and
- $1(y)$ waiting for a packet numbered 1.

Hence, the protocol has six states: $(n,0)$, $(0,1)$, $(n,1)$, $(1,0)$, $(0,0)$, and $(1,1)$. Let s_i, $i = 0,1,2,3,4,5$ represent these six states, respectively. The events would be as follow:

- e_1 The packet 0 is lost ($ep0lo(.)$);
- e_2 The packet 0 is received ($ep0rc(.)$);
- e_3 The packet 1 is lost ($ep1lo(.)$);
- e_4 The packet 1 is received ($ep1rc(.)$);
- e_5 The acknowledgement $ACK0$ is lost ($ea0lo(.)$);
- e_6 The acknowledgement $ACK0$ is received ($ea0rc(.)$);
- e_7 The acknowledgement $ACK1$ is lost ($ea1lo(.)$);
- e_8 The acknowledgement $ACK1$ is received ($ea1rc(.)$);
- e_9 Timeout and 0 is received or lost ($ep0to(.)$);
- e_{10} Timeout and 1 is received or lost ($ep1to(.)$);

The chart of events and their preconditions and postconditions of the example is constructed as shown in Table 5.1.

From Table 5.1, the sets of enabled events in states $s_1, s_2, ..., s_6$, respectively, are: $E_{s_4} = \{e_1, e_2\}$, $E_{s_0} = \{e_1, e_2\}$, $E_{s_1} = \{e_5, e_6, e_9\}$, $E_{s_2} = \{e_3, e_4\}$, $E_{s_5} = \{e_3, e_4\}$, and $E_{s_3} = \{e_7, e_8, e_{10}\}$. Thus, we can obtain the next state function $s' = f(e, s)$ from Table 5.1.

The state transition graph of the protocol is shown in Figure 5.1 and we want to prove the correctness of the protocol. We notice that correctness is a logic

preconditions	events	postconditions
s_4	e_1	s_4
s_4	e_2	s_1
s_0	e_1	s_4
s_0	e_2	s_1
s_1	e_5	s_1
s_1	e_9	s_1
s_1	e_6	s_2
s_2	e_3	s_5
s_2	e_4	s_3
s_5	e_3	s_5
s_5	e_4	s_3
s_3	e_7	s_3
s_3	e_{10}	s_3
s_3	e_8	s_0

Table 5.1: Events and conditions of the example.

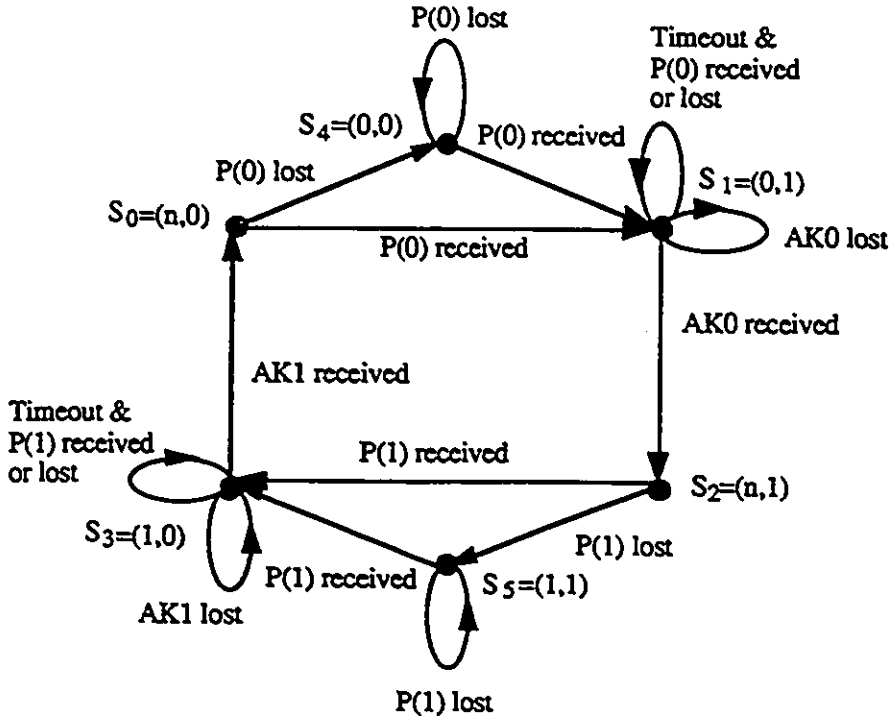


Figure 5.1: The graph of the protocol.

property: it only concerns the order in which events take place.

When there is no transmission error, the sequence of states of the protocol is

$$(n,0) \to (0,1) \to (n,1) \to (1,0) \to (n,0) \to (0,1) \to \cdots$$

These transitions represent the arrival of a packet numbered 0, the arrival of an $ACK0$, the arrival of a packet numbered 1, the arrival of an $ACK1$, and so on. Once Figure 5.1 has been verified, the correctness of ABP is proved. Indeed, Figure 5.1 proves that, even with arbitrary transmission errors, the receiver gets a packet numbered 0, then the sender gets an $ACK0$, then the receiver gets a packet

numbered 1, and so on.

In order to verify the protocol, we need to specify the protocol in the temporal logic models. Then the simulator in the software developed in Chapter 8 can take in the rules of the formulas given by the protocol model. By using Corollary 5.3.3, the reachability algorithm performs the verification. Now we first specify the temporal logic model of the protocol.

Let the local variables x and y represent the states of sender x and receiver y. Using $0(x)$, $1(x)$, $n(x)$, $0(y)$, and $1(y)$ as predicates, taking $ep0rc(.)$, $ep0lo(.)$, $ea0rc(.)$, $ea0lo(.)$, $ep0to(.)$, $ep1rc(.)$, $ep1lo(.)$, $ea1rc(.)$, $ea1lo(.)$, and $ep1to(.)$ as event symbols given in the previous section, we give the set of formulas of the specifications for the protocol as follows.

Dynamics

$$\Box[ep0rc(y) \wedge n(x) \wedge 0(y) \Rightarrow 0(\bigcirc x) \wedge 1(\bigcirc y)] \qquad (5.4.P1)$$

$$\Box[ep0lo(x) \wedge n(x) \Rightarrow 0(\bigcirc x)] \qquad (5.4.P2)$$

$$\Box[ep0rc(x) \wedge 0(x) \Rightarrow n(\bigcirc x)] \qquad (5.4.P3)$$

$$\Box[ea0lo(y) \wedge 1(y) \Rightarrow 1(\bigcirc y)] \qquad (5.4.P4)$$

$$\Box[ep0to(x) \wedge 0(x) \Rightarrow 0(\bigcirc x)] \qquad (5.4.P5)$$

$$\Box[ep0rc(y) \wedge 0(y) \Rightarrow 1(\bigcirc y)] \qquad (5.4.P6)$$

$$\Box[ep0lo(x) \wedge 0(x) \Rightarrow 0(\bigcirc x)] \qquad (5.4.P7)$$

$$\Box[ep1lo(x) \wedge n(x) \Rightarrow 1(\bigcirc x)] \qquad (5.4.P8)$$

$$\Box[ep1rc(y) \wedge n(x) \wedge 1(y) \Rightarrow 1(\bigcirc x) \wedge 0(\bigcirc y)] \qquad (5.4.P9)$$

$$\Box[ep1rc(y) \wedge 1(y) \Rightarrow 0(\bigcirc y)] \qquad (5.4.P10)$$

$$\Box[ep1lo(x) \wedge 1(x) \Rightarrow 1(\bigcirc x)] \qquad (5.4.P11)$$

$$\Box[ea1lo(y) \wedge 0(y) \Rightarrow 0(\bigcirc y)] \qquad (5.4.P12)$$

$$\Box[ep1to(x) \wedge 0(x) \Rightarrow 0(\bigcirc x)] \qquad (5.4.P13)$$

$$\Box[ea1rc(x) \wedge 1(x) \Rightarrow n(\bigcirc x)] \qquad (5.4.P14)$$

Formula (5.4.$P1$) describes the effect of that packet 0 is received: The state of the sender changes from $n(x)$ to $0(x)$, and the state of the receiver changes from $0(y)$ to $1(y)$. (5.4.$P2$) – (5.4.$P14$) are similar to (5.4.$P1$), describing the effects of the occurrences of the corresponding events, respectively.

Initial condition

$$n(x) \wedge 0(y) \qquad (5.4.P15)$$

It means that initially the sender is at $n(x)$ and the receiver is at $0(y)$.

Using the results obtained in Section 5.3, Algorithm 5.3.1 can be used to identify the states that can be reached by the specification (5.4.P1)-(5.4.P15). By Corollary 5.3.3, the states specified by (5.4.P1)-(5.4.P15) are reachable.

Applying Algorithm 5.3.1 to the ABP protocol, the reachability computation result of the protocol, with the reachability set and the enabled events, is produced

The state transitions of the system are:

```
The root:  s0=(n,0)
e1=(ep0rc 2)  —> s1=(0,1)
e2=(ep0lo 1)  —> s2=(0,0)

The root:  s1=(0,1)
e3=(ea0rc 1)  —> s3=(n,1)
e5=(ep0to 1)  —> s5=(0,1)=s1
e4=(ea0lo 2)  —> s4=(0,1)=s1

The root:  s2=(0,0)
e7=(ep0lo 1)=e2 —> s7=(0,0)=s2
e6=(ep0rc 2)=e1 —> s6=(0,1)=s1

The root:  s3=(n,1)
e8=(ep1lo 1)  —> s8=(1,1)
e9=(ep1rc 2)  —> s9=(1,0)

The root:  s8=(1,1)
e11=(ep1lo 1)=e8 —> s11=(1,1)=s8
e10=(ep1rc 2)=e9 —> s10=(1,0)=s9

The root:  s9=(1,0)
e14=(ea1rc 1)  —> s14=(n,0)=s0
e12=(ea1lo 2)  —> s12=(1,0)=s9
e13=(ep1to 2)  —> s13=(1,0)=s9
```

Figure 5.2: Reachability computation of the example.

by our software as shown in Figure 5.2. The states other than those specified by (5.4.P1)-(5.4.P15) are not reachable. This proves the correctness of the ABP protocol.

A representation for the TLM of the example has been obtained and now we can visualize all the components of the TLM of the example. The states, events, the sets of enabled events, the mapping f, and the initial state can be seen from Figure 5.2. The set $\mathbf{F}$ is the set of formulas of the specification (P1)-(P15), and the labelling function can be seen as it is labelling the events and states in Figure 5.2 to $\mathbf{F}^*$. The processes in the example are deterministic in a sense of that an event has only one outcome on a state so that $p = 1$. Nevertheless, there are still some nondeterminism, for instance, from s_0 to s_1 or to s_4, in a sense of the physical possibility rather than probability of event occurrences. The use of reachability as a means of proving the correctness of the ABP protocol gives the entire dynamic behavior of the system.

5.5 Conclusion

The dynamics of DESs has been described by the next state function with the enable conditions of events upon the truth of formulas. In the reachability analysis of the DES, it has been shown that reachability of states is equivalent to validity of the corresponding logic formulas. Hence, we may use reachability properties to verify the corresponding validity properties.

It is noticed that validity here is limited and relative to the logic specification provided by the user in applications; and therefore, it may be invalid if the user is erroneous. Nevertheless, this result theoretically simplifies the system design procedures as it is a bridge between reachability and validity. Based on this result, reachability properties may be used instead of validity properties. Hence, the reachability set has been defined and an algorithm for computing the reachability set and constructing the reachability graph has been developed.

An example of a data link protocol has been demonstrated to convey the theoretical discussions. The correctness of the protocol has been proved by the reachability algorithm. It shows that the specification of the plant can be verified which means an early verification, *i.e.* the verification of the plant before a controller is available. Our methods can do more than what Thistle's did in [105] where the full plant-controller system must be available before verification can proceed.

Comparing the related works, our work in this chapter is close to the corresponding parts of Ostroff [85]. Ostroff's paper [84] is a further development of his book [85] which verifies the fairness of arbitrary system. The paper verifies the time bounds of finite state systems. He uses a Timed Transition Model which adds a time into the fair transition system while we add probabilistic features into a simplified fair transition system. His model is complicated on the states with state assignments and state maps, and so is his reachability definition. His reachability is defined as: s is reachable if it occurs in a legal trajectory. If q is a

state map corresponding to s, then q is reachable if s is reachable. His reachability graph is the set of all state-maps reachable from the set of initial state-maps since the set of reachable states is always infinite because of time ticks but the number of the state-maps may be finite. But our definition of reachability is simple, straightforward, and including the probabilistic case. He has two reachability algorithms, RG1 for lower time bounds being zero and RG2 for lower time bounds being not zero. RG1 is less general than RG2; but the reachability graphs of RG2 are usually much larger than those of RG1. Our reachability algorithm is closer to RG1 than RG2 and the procedure of our algorithm is different from those of RG1 and RG2 because our model is different from theirs. He uses RG1 and RG2 to verify the real-time properties while we use our reachability algorithm to verify the system behavior by reachability synthesis. We can do what his RG1 can do without talking about real-times but we can also deal with probabilistic case. We do not deal with his real-time part in RG2 and he does not handle probabilistic case.

As mentioned in Section 5.2, a DES is represented by a TLM which itself may be composed of a number of TLMs. Hence, the procedure of composing these TLMs into a TLM is needed to be developed.

The reachability graph of the example shows the natural behavior of the uncontrolled system. If a required behavior is specified within the natural behavior, then a controller or supervisor is needed to be composed and synthesized into the system to achieve the required behavior. Therefore, in the next chapter, we

will study the composition of temporal logic models and then the procedures for reachability synthesis and controller design.

Chapter 6

Composition and Controller Synthesis of DESs

In this chapter we will study the composition of the temporal logic models and develop the procedures of the controller design and reachability synthesis for DESs.

6.1 Introduction

In Chapter 5, Figure 5.2 shows the natural behavior of the uncontrolled system. If the natural behavior is not satisfactory in some aspects, then a description of the required behavior of the system is formulated in temporal logic specifications. The description includes a restriction on the reachability set of the system to prevent the unsatisfactory states from being reached. A controller is then needed to be designed and synthesized to realize the specified closed-loop behavior. The controller can be represented as a TLM. Hence, procedures for the

controller design and synthesis and for the composition of a number of TLMs are needed to be developed.

In the automata and formal languages approach [95,112], algorithms have been developed for the synthesis of the designed supervisors (controllers). Reachability and controller synthesis have been studied in the Petri nets approach in [21,39]. In the temporal logic framework presented in Chapter 2, Thistle and Wonham have designed a controller for a manufacturing system and verified that the required behavior description can be deduced from the plant and controller specifications of that example; and no general procedures for the composition and synthesis have been given there. Hence, there is a need for a theoretical basis and procedures for the composition and controller synthesis in the temporal logic approach. This involves specifying the behavior of a number of closely interacting subsystems and synthesizing a controller to coordinate the system behavior to avoid problems such as conflicts during access to shared resources. Therefore, the main objective of this chapter is to develop procedures for the composition and controller synthesis of DESs in a temporal logic framework.

In this chapter, we will formulate the temporal logic models established in Chapter 4 as Σ-algebras and introduce a Σ-homomorphism of temporal logic models to the composition of DESs, and then develop the procedures for the reachability synthesis and controller design of DESs.

This chapter is organized as follows. In the following section, as an example,

a system of computer read-write processes is described in a temporal logic model and is used to propose the need for the composition and controller synthesis of DESs. By applying Hennessy's process algebra [32], in Section 6.3, a temporal logic model (TLM) is formulated as a Σ−algebra and a Σ−homomorphism of TLMs is introduced to provide a composition operation for reachability synthesis of TLMs. This will lead naturally, in Section 6.4, to a controller synthesis procedure using reachability properties to ensure that the closed-loop system has the required behavior. Finally in Section 6.5, as a conclusion of this chapter, the related works are discussed as comparing with our results.

6.2 An Example

Consider the problem of controlling the access of a number of computer processes to a shared disk [38]: each process x can be in one of five states: idle ($ID(x)$), waiting for reading ($WR(x)$), waiting for writing ($WW(x)$), reading ($R(x)$), and writing ($W(x)$). Let s_1, s_2, s_3, s_4, and s_5 represent these five states, respectively. Then the events would be as follows:

- e_1 The process requests to read ($erqr(x)$);
- e_2 The process requests to write ($erqw(x)$);
- e_3 The process starts reading ($estr(x)$);
- e_4 The process starts writing ($estw(x)$);
- e_5 The process finishes reading ($endr(x)$);

preconditions	events	postconditions
s_1	e_1	s_2
s_1	e_2	s_3
s_2	e_3	s_4
s_3	e_4	s_5
s_4	e_5	s_1
s_5	e_6	s_1

Table 6.1: Events and conditions of the example.

- e_6 The process finishes writing ($endw(x)$);

The chart of events and their preconditions and postconditions of the example is constructed as shown in Table 6.1.

From Table 6.1, the sets of enabled events in states $s_1, s_2, ..., s_5$, respectively, are: $E_{s_1} = \{e_1, e_2\}$, $E_{s_2} = \{e_3\}$, $E_{s_3} = \{e_4\}$, $E_{s_4} = \{e_5\}$, $E_{s_5} = \{e_6\}$. Thus, we can express the next state function $s' = f(e, s)$ for the given example as shown in Figure 6.1. As an illustration, Figure 6.1 is just a try to show a relation between states and events for the mapping f similar to that between the function values and its variables.

Let the local variable x represent the states of process x. Using $ID(x)$, $WW(x)$, $WR(x)$, $W(x)$, and $R(x)$ as predicates, taking $erqr(x)$, $erqw(x)$, $estr(x)$, $estw(x)$, $endr(x)$, and $endw(x)$ as event predicate symbols, we give the set of formulas of the specifications for the given example as follows.

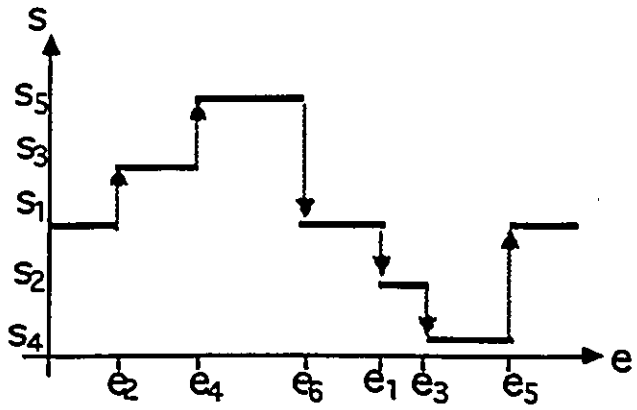


Figure 6.1: The next state function of the example.

Dynamics of the plant

$$\Box[erqr(x) \wedge ID(x) \Rightarrow WR(\bigcirc x)] \qquad (6.2.P1)$$

$$\Box[estr(x) \wedge WR(x) \Rightarrow R(\bigcirc x)] \qquad (6.2.P2)$$

$$\Box[endr(x) \wedge R(x) \Rightarrow ID(\bigcirc x)] \qquad (6.2.P3)$$

$$\Box[erqw(x) \wedge ID(x) \Rightarrow WW(\bigcirc x)] \qquad (6.2.P4)$$

$$\Box[estw(x) \wedge WW(x) \Rightarrow W(\bigcirc x)] \qquad (6.2.P5)$$

$$\Box[endw(x) \wedge W(x) \Rightarrow ID(\bigcirc x)] \qquad (6.2.P6)$$

Formula (6.2.$P1$) describes the effect of the request to read of process x: its state changes from $ID(x)$ to $WR(x)$, while the states of the other processes remain the same. (6.2.$P2$) – (6.2.$P6$) are similar to (6.2.$P1$), describing the effects of the

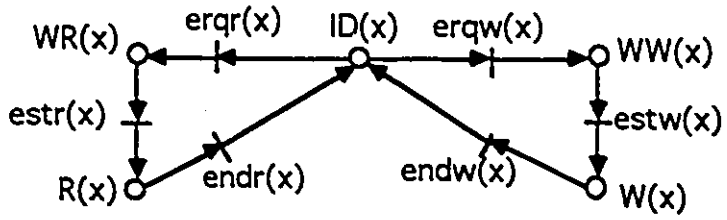


Figure 6.2: The graph of process x.

commencement of reading, the end of reading, the request to write, commencement of writing, and the end of writing, respectively, of process x.

Initial condition

$$ID(x) \tag{6.2.P7}$$

(6.2.P7) means that initially all processes are idle.

The graph of the example is shown in Figure 6.2. To see the synchronization of events clearly, in this chapter, we add a bar as the transition of an event into the graph of the temporal logic models.

For the given example in the case of $N = 2$, the simulation result given by our software is shown in Figure 6.3. The reachability graph, with the reachability set as its set of nodes and events as its edges, is constructed in Figure 6.4.

The state transitions of the unsupervised system are:

The root: s0=(ID,ID)
e1=(erqr 1) ---> s1=(WR,ID)
e3=(erqw 1) ---> s3=(WW,ID)
e2=(erqr 2) ---> s2=(ID,WR)
e4=(erqw 2) ---> s4=(ID,WW)

The root: s1=(WR,ID)
e5=(estr 1) ---> s5=(R,ID)
e6=(erqr 2)=e2 ---> s6=(WR,WR)
e7=(erqw 2)=e4 ---> s7=(WR,WW)

The root: s2=(ID,WR)
e9=(erqr 1)=e1 ---> s9=(WR,WR)=s6
e10=(erqw 1)=e3 ---> s10=(WW,WR)
e8=(estr 2) ---> s8=(ID,R)

The root: s3=(WW,ID)
e11=(estw 1) ---> s11=(W,ID)
e12=(erqr 2)=e2 ---> s12=(WW,WR)=s10
e13=(erqw 2)=e4 ---> s13=(WW,WW)

The root: s4=(ID,WW)
e15=(erqr 1)=e1 ---> s15=(WR,WW)=s7
e16=(erqw 1)=e3 ---> s16=(WW,WW)=s13
e14=(estw 2) ---> s14=(ID,W)

The root: s5=(R,ID)
e17=(endr 1) ---> s17=(ID,ID)=s0
e18=(erqr 2)=e2 ---> s18=(R,WR)
e19=(erqw 2)=e4 ---> s19=(R,WW)

The root: s6=(WR,WR)
e20=(estr 1)=e5 ---> s20=(R,WR)=s18
e21=(estr 2)=e8 ---> s21=(WR,R)

The root: s7=(WR,WW)
e22=(estr 1)=e5 ---> s22=(R,WW)=s19
e23=(estw 2)=e14 ---> s23=(WR,W)

The root: s8=(ID,R)
e24=(erqr 1)=e1 ---> s24=(WR,R)=s21
e25=(erqw 1)=e3 ---> s25=(WW,R)
e26=(endr 2) ---> s26=(ID,ID)=s0

The root: s10=(WW,WR)
e27=(estw 1)=e11 ---> s27=(W,WR)
e28=(estr 2)=e8 ---> s28=(WW,R)=s25

The root: s11=(W,ID)
e29=(endw 1) ---> s29=(ID,ID)=s0
e30=(erqr 2)=e2 ---> s30=(W,WR)=s27
e31=(erqw 2)=e4 ---> s31=(W,WW)

The root: s13=(WW,WW)
e32=(estw 1)=e11 ---> s32=(W,WW)=s31
e33=(estw 2)=e14 ---> s33=(WW,W)

The root: s14=(ID,W)
e34=(erqr 1)=e1 ---> s34=(WR,W)=s23
e35=(erqw 1)=e3 ---> s35=(WW,W)=s33
e36=(endw 2) ---> s36=(ID,ID)=s0

The root: s18=(R,WR)
e37=(endr 1)=e17 ---> s37=(ID,WR)=s2
e38=(estr 2)=e8 ---> s38=(R,R)

The root: s19=(R,WW)
e39=(endr 1)=e17 ---> s39=(ID,WW)=s4
e40=(estw 2)=e14 ---> s40=(R,W)

The root: s21=(WR,R)
e41=(estr 1)=e5 ---> s41=(R,R)=s38
e42=(endr 2)=e26 ---> s42=(WR,ID)=s1

The root: s23=(WR,W)
e43=(estr 1)=e5 ---> s43=(R,W)=s40
e44=(endw 2)=e36 ---> s44=(WR,ID)=s1

The root: s25=(WW,R)
e45=(estw 1)=e11 ---> s45=(W,R)
e46=(endr 2)=e26 ---> s46=(WW,ID)=s3

The root: s27=(W,WR)
e47=(endw 1)=e29 ---> s47=(ID,WR)=s2
e48=(estr 2)=e8 ---> s48=(W,R)=s45

The root: s31=(W,WW)
e49=(endw 1)=e29 ---> s49=(ID,WW)=s4
e50=(estw 2)=e14 ---> s50=(W,W)

The root: s33=(WW,W)
e51=(estw 1)=e11 ---> s51=(W,W)=s50
e52=(endw 2)=e36 ---> s52=(WW,ID)=s3

The root: s38=(R,R)
e53=(endr 1)=e17 ---> s53=(ID,R)=s8
e54=(endr 2)=e26 ---> s54=(R,ID)=s5

The root: s40=(R,W)
e55=(endr 1)=e17 ---> s55=(ID,W)=s14
e56=(endw 2)=e36 ---> s56=(R,ID)=s5

The root: s45=(W,R)
e57=(endw 1)=e29 ---> s57=(ID,R)=s8
e58=(endr 2)=e26 ---> s58=(W,ID)=s11

The root: s50=(W,W)
e59=(endw 1)=e29 ---> s59=(ID,W)=s14
e60=(endw 2)=e36 ---> s60=(W,ID)=s11

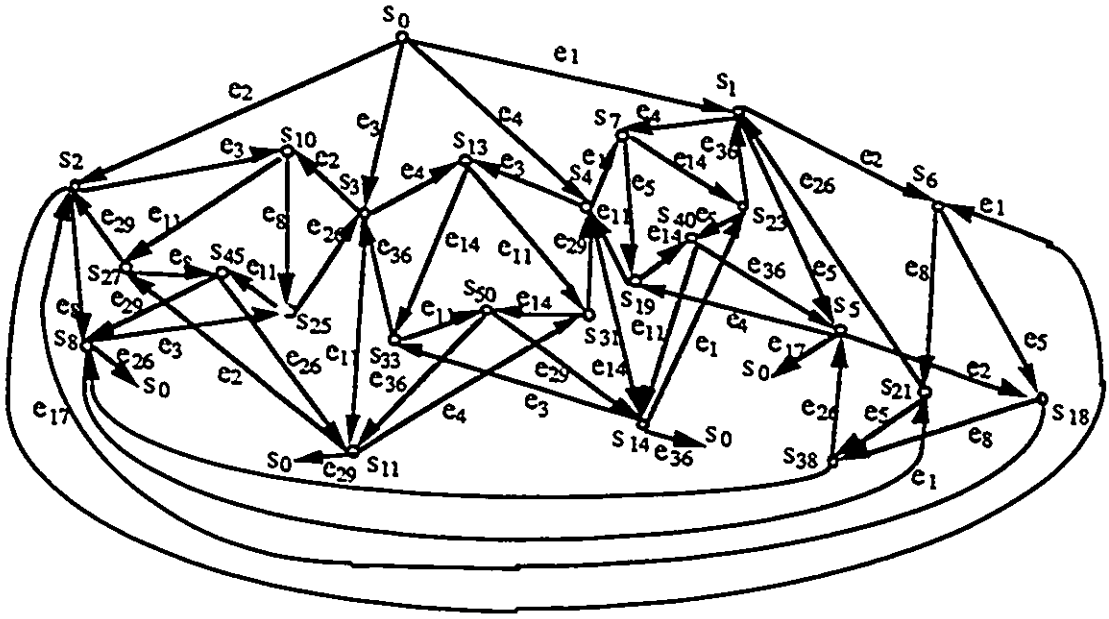


Figure 6.4: Reachability graph of the example.

As we can see from Figures 6.2 and 6.3 that there is no restriction on the number of processes which are allowed simultaneous access to the disk for either reading or writing. For instance, at $s_{31} = (W, WW)$, process 2 is waiting for writing while process 1 is writing data to the disk. If event e_{14} were to be executed at this moment, two processes would be writing to the disk at same time, *i.e.* the system is at $s_{50} = (W, W)$, then the data in the disk would be altered inconsistently and cause grave problems such as the crash of the system.

Therefore, a controller is needed to avoid such problems. For this example, a control program should be designed to tell the interrupt handler of the system that process 1 is updating the data and it should not perform non-local goto. It will allow process 2 to write to the disk after process 1 finishing its writing.

As designing a controller for the given system, it proposes a problem of composing and synthesizing a controller with the plant to achieve the required behavior of the system. This motivates us to study the composition and synthesis of TLMs. We will introduce a process algebra in the next section to compose and synthesize TLMs.

6.3 Σ-Algebra and Composition of TLMs

As mentioned in Section 5.2, a DES may be composed of a number of TLMs with each TLM representing one of the DES processes. In this section, by applying Hennessy's process algebra [32], a TLM will be formulated as a $\Sigma-$algebra

and a $\Sigma-$homomorphism of TLMs will be introduced to provide the composition operation and controller synthesis for DESs. The reason that Hennessy's book cited here is its updating usage of process algebras for computational processes. Of course, $\Sigma-$algebra is a kind of universal algebra [8]. Our purpose of applying the $\Sigma-$algebra is to provide a theoretical basis for the operational verification of DESs by the reachability synthesis which is the synchronization of two systems to achieve a desired reachability set for the composite system. In terms of Hennessy's algebra, a Σ-algebra is defined as follows.

Definition 6.3.1 [32]: Given a set of formal functional symbols Σ. A $\Sigma-$algebra is defined by a pair (A, Σ_A) in which A is a set, and Σ_A is a set of functions $\{f_A, f \in \Sigma\}$ such that, if the number of arguments of f is n, then f_A is a function from $A^n \rightarrow A$.

To show the mathematical significance of Σ-algebras, we need to introduce the $\Sigma-$homomorphism, between two Σ-algebras, which is defined in the following.

Definition 6.3.2 [32]: A $\Sigma-$homomorphism from $\Sigma-$algebra (A, Σ_A) to $\Sigma-$algebra (B, Σ_B) is defined by a function h satisfying for every f in Σ with k arguments

$$h(f_A(a_1, ..., a_k)) = f_B(h(a_1), ..., h(a_k)).$$

$\Sigma-$homomorphisms are simply functions which preserve the structures between $\Sigma-$algebras. By viewing our TLM as a $\Sigma-$algebra, this property can be

used in the composition and reachability synthesis of TLMs. For this purpose, the following two results establish TLMs as Σ−algebras and the Σ−homomorphism between them.

Theorem 6.3.1: A TLM is a Σ−algebra $(\{\mathbf{S}, \mathbf{E}, \mathbf{F}^*\}, \{f, l, p, s_0\})$ with the set $\{\mathbf{S}, \mathbf{E}, \mathbf{F}^*\}$, the binary operations $f: \mathbf{E} \times \mathbf{S} \rightarrow \mathbf{S}$, $l: \mathbf{E} \times \mathbf{S} \rightarrow \mathbf{F}^*$, and $p: \mathbf{S} \times \mathbf{S} \rightarrow [0,1]$, and the nullary operation $s_0 \in \mathbf{S}$.

Proof: By Definition 3.3.1, $\mathbf{S}$, $\mathbf{E}$, $\mathbf{F}^*$ are sets, and f, l, p, are functional symbols with $f: \mathbf{E} \times \mathbf{S} \rightarrow \mathbf{S}$, $l: \mathbf{E} \times \mathbf{S} \rightarrow \mathbf{F}^*$, and $p: \mathbf{S} \times \mathbf{S} \rightarrow [0,1]$. s_0 can be viewed as a nullary operation over $\mathbf{S}$. By Definition 6.3.1, $(\{\mathbf{S}, \mathbf{E}, \mathbf{F}^*\}, \{f, l, p, s_0\})$ is a Σ-algebra. ■

Corollary 6.3.2: Given two temporal logic models $M = (\{\mathbf{S}, \mathbf{E}, \mathbf{F}^*\}, \{f, l, p, s_0\})$ and $M' = (\{\mathbf{S}', \mathbf{E}', \mathbf{F}^{*'}\}, \{f', l', p', s_0'\})$. For any four functions: $h: \mathbf{S} \rightarrow \mathbf{S}'$, $g: \mathbf{E} \rightarrow \mathbf{E}'$, $i: [0,1] \rightarrow [0,1]$, and $k: \mathbf{F}^* \rightarrow \mathbf{F}^{*'}$, if they satisfy $h(s_0) = s_0'$, and for all $e \in E_s$, $g(e) \in E_{h(s)}$, $f'(g(e), h(s)) = h(f(e,s))$, $l'(g(e), h(s)) = k(l(e,s))$ and $p'(h(s_1), h(s_2)) = i(p(s_1, s_2))$, then $[h, g, i, k]$ is a Σ−homomorphism from M to M'.

Proof: Following Theorem 6.3.1, M and M' are Σ-algebras. Since $h(s_0) = s_0'$, and for all $e \in E_s$, $g(e) \in E_{h(s)}$, $f'(g(e), h(s)) = h(f(e,s))$, $l'(g(e), h(s)) = k(l(e,s))$ and $p'(h(s_1), h(s_2)) = i(p(s_1, s_2))$, we define $[h, g, i, k](f(e,s)) \stackrel{\text{def}}{=} h(f(e,s))$, $[h, g, i, k](l(e,s)) \stackrel{\text{def}}{=} k(l(e,s))$, $[h, g, i, k](p(s_1, s_2)) \stackrel{\text{def}}{=} i(p(s_1, s_2))$, $[h, g, i, k](s_0) \stackrel{\text{def}}{=} h(s_0)$. Then, $[h, g, i, k](f(e,s)) = f'(g(e), h(s))$, $[h, g, i, k](l(e,s)) = l'(g(e), h(s))$,

$[h,g,i,k](p(s_1,s_2)) = p'(h(s_1),h(s_2))$, $[h,g,i,k](s_0) = s_0'$. By Definition 6.3.2, $[h,g,i,k]$ is a Σ−homomorphism from M to M'.■

An important property of a Σ−homomorphism between TLMs is that it preserves the dynamic behavior and reachability of the TLMs as shown in the following theorem.

Theorem 6.3.3: Let $[h,g,i,k] : M \rightarrow M'$ be a Σ−homomorphism of TLMs. Then

(1). If $t = f(e,s)$ in M, it follows that $h(t) = f'(g(e),h(s))$ in M'.

(2). If t is a reachable state of M, then $h(t)$ is a reachable state of M'.

Proof: (1). Since $[h,g,i,k] : M \rightarrow M'$ is a Σ−homomorphism, it follows from Corollary 6.3.2 that $h(f(e,s)) = f'(g(e),h(s))$. Because of $t = f(e,s) \in M$, $h(t) = h(f(e,s)) = f'(g(e),h(s))$. Hence $h(t) \in M'$.

(2). Since t is a reachable state of M, there exist $s \in$ S and $e \in E_s$ such that $t = f(e,s) \in M$. It follows from (1) of this theorem that $h(t) = f'(g(e),h(s)) \in M'$ where $g(e) \in E_{h(s)}$ and $h(s) \in M'$. By Definitions 5.3.1 and 5.3.2, $h(t)$ is a reachable state of M'. ■

The above results provide a possible composition operation to build a TLM from its component TLMs, say two TLMs, by putting together two TLMs with synchronization between selected events for reachability synthesis. In the following, we will define the composition operation and investigate its properties in

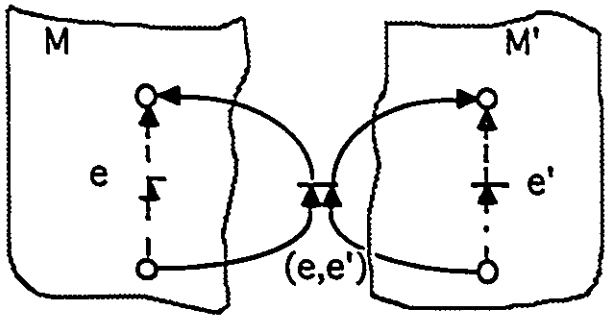


Figure 6.5: Synchronization of two events.

particular, what effect the composition has on the dynamic behavior of the TLM and on the reachability set of the system.

Definition 6.3.3: For two temporal logic models $M = (\{\mathbf{S}, \mathbf{E}, \mathbf{F}^*\}, \{f, l, p, s_0\})$ and $M' = (\{\mathbf{S}', \mathbf{E}', \mathbf{F}^{*'}\}, \{f', l', p', s_0'\})$, the composition of M and M', denoted by $\bar{M} \stackrel{\text{def}}{=} M \| M'$, is defined by the concurrent operation of M and M' in which selected events, *e.g.* $e \in \mathbf{E}$ and $e' \in \mathbf{E}'$, are synchronized and form a synchronized event $\bar{e} = (e, e')$. The event set of $\bar{M}$ is $\bar{\mathbf{E}} = \{(e, \epsilon) | e \in \mathbf{E}\} \cup \{(\epsilon, e') | e' \in \mathbf{E}'\} \cup \{(e, e') | e \in \mathbf{E}, e' \in \mathbf{E}'\}$ where (e, ϵ) and (ϵ, e') denote events in M and M' respectively which are not synchronized with an event in the other TLM. For all $s \in \mathbf{S}$ and $s' \in \mathbf{S}'$ with $\bar{s} \stackrel{\text{def}}{=} (s, s')$, $\bar{e} \in E_{\bar{s}}$ only if $e \in E_s$ and $e' \in E_{s'}$. Thus, $\bar{f}(\bar{e}, \bar{s}) \stackrel{\text{def}}{=} (f(e, s), f'(e', s'))$, $\bar{l}(\bar{e}, \bar{s}) \stackrel{\text{def}}{=} (l(e, s), l'(e', s'))$, and $\bar{p}(\bar{s_1}, \bar{s_2}) \stackrel{\text{def}}{=} (p(s_1, s_2), p'(s_1', s_2'))$.

The synchronization of two events from M and M' is illustrated graphically in Figure 6.5. Let $\bar{\mathbf{S}}$ be the set of all $\bar{s} = (s, s')$. Then the composed model

$\bar{M} = M \| M'$ has the set of states $\bar{\mathbf{S}} = \mathbf{S} \times \mathbf{S}'$, the set $\bar{\mathbf{F}}^* = \mathbf{F}^* \cup \mathbf{F}'^*$, and the set of events $\bar{\mathbf{E}}$, the functions $\bar{f}$, $\bar{l}$, $\bar{p}$, and the initial state $\bar{s}_0 = (s_0, s'_0)$. Particularly, $\bar{f}$ works with (e, ϵ) in this way: $\bar{f}((e,\epsilon),(s,s')) = (f(e,s), f(\epsilon,s')) = (f(e,s),s')$.

For the synchronization of events, the set $\mathbf{E}$ can be defined more generally as $\bar{\mathbf{E}} = \{(e,*)|e \in \mathbf{E}\} \cup \{(*,e')|e' \in \mathbf{E}'\} \cup \{(e,e')|e \in \mathbf{E}, e' \in \mathbf{E}'\}$ where $(e,*)$ and $(*,e')$ denote unsynchronized events in M and M' respectively. However, we have the initial state $\bar{s}_0 = (s_0, s'_0)$ for $\bar{M}$ such that ϵ can be included for null synchronization to make those unsynchronized events as the events in the composite system. Thus, the dynamic behavior of the composed TLM is now given in terms of the projection mappings from $\bar{M}$ into M and M'.

Theorem 6.3.4: Let $[h, g, i, k]$ and $[h', g', i', k']$ be the projections from $\bar{M} = M \| M'$ to M and M' respectively, *i.e.* $h: \bar{\mathbf{S}} \to \mathbf{S}$, $g: \bar{\mathbf{E}} \to \mathbf{E}$, $i: [0,1] \to [0,1]$, $k: \bar{\mathbf{F}}^* \to \mathbf{F}^*$, $h': \bar{\mathbf{S}} \to \mathbf{S}'$, $g': \bar{\mathbf{E}} \to \mathbf{E}'$, $i': [0,1] \to [0,1]$, and $k': \bar{\mathbf{F}}^* \to \mathbf{F}'^*$. Then

(1): The projection mappings are $\Sigma-$homomorphisms;

(2): If $\bar{t} = \bar{f}(\bar{e}, \bar{s})$ is in $\bar{M}$, then both $h(\bar{t}) = f(g(\bar{e}), h(\bar{s}))$ is in M and $h'(\bar{t}) = f'(g'(\bar{e}), h'(\bar{s}))$ is in M'; and

(3): If $\bar{t}$ is reachable in $\bar{M} = M \| M'$, then both $h(\bar{t})$ is reachable in M and $h'(\bar{t})$ is reachable in M'.

Proof: (1) By the definitions of projection $[h, g, i, k]$, we have $h(\bar{s}) = s$, $g(\bar{e}) = e$, $k(\bar{l}(\bar{e}, \bar{s})) = l(e,s)$, $i(\bar{p}(\bar{s_1}, \bar{s_2})) = p(s_1, s_2)$ and $h(\bar{f}(\bar{e}, \bar{s})) = f(e,s)$. Hence we

have $l(g(\bar{e}), h(\bar{s})) = l(e,s) = k(\bar{l}(\bar{e},\bar{s}))$, $p(h(\bar{s_1}), h(\bar{s_2})) = p(s_1,s_2) = i(\bar{p}(\bar{s_1},\bar{s_2}))$, $f(g(\bar{e}), h(\bar{s})) = f(e,s) = h(\bar{f}(\bar{e},\bar{s}))$, and $h(\bar{s_0}) = s_0$. By Corollary 6.3.2, $[h,g,i,k]$ is $\Sigma-$homomorphism. The same results can be obtained for $[h',g',i',k']$.

(2): By (1) of this theorem, $[h,g,i,k]$ and $[h',g',i',k']$ are $\Sigma-$homomorphisms. For $\bar{t} = \bar{f}(\bar{e},\bar{s})$ in $\bar{M}$, it follows from (1) of Theorem 6.3.3 that $h(\bar{t}) = f(g(\bar{e}), h(\bar{s}))$ is in M and $h'(\bar{t}) = f'(g'(\bar{e}), h'(\bar{s}))$ is in M'.

(3): By (1) of this theorem, $[h,g,i,k]$ and $[h',g',i',k']$ are $\Sigma-$homomorphisms. Because $\bar{t}$ in $\bar{M} = M \| M'$ is reachable, it follows from (2) of Theorem 6.3.3 that both $h(\bar{t})$ is reachable in M and $h'(\bar{t})$ is reachable in M'. ■

We use the given example to illustrate our composition procedure. Further consideration of this model shows that it is formed by the composition of two TLMs, one for processes and one for the disk. Events in each TLM are synchronized with one another in relation to the way in which the two subsystems interact such that these two TLMs are composed. For example, when a process goes from waiting for reading to reading, the disk goes from idle to being read. We can then combine the two pairs of the synchronized events into a single event, and then the two states, reading for the process model and being read for the disk model, into a single state.

For the disk model in the given example, we give the following description. $BR(d)$, $BW(d)$, and $ID(d)$ will represent the three possible states of the disk d, those of being read, being written, and idle. The event symbols $esbr(d)$ and $enbr(d)$ represent the commencement and end of being read, and $esbw(d)$ and

```
The state transitions of the diskfile are:

The root: s0=(ID,ID)
e1=(esbr 1)  ---> s1=(BR,ID)
e3=(esbw 1)  ---> s3=(BW,ID)
e2=(esbr 2)  ---> s2=(ID,BR)
e4=(esbw 2)  ---> s4=(ID,BW)

The root: s1=(BR,ID)
e5=(enbr 1)  ---> s5=(ID,ID)=s0
e6=(esbr 2)=e2 ---> s6=(BR,BR)
e7=(esbw 2)=e4 ---> s7=(BR,BW)

The root: s2=(ID,BR)
e9=(esbr 1)=e1 ---> s9=(BR,BR)=s6
e10=(esbw 1)=e3 ---> s10=(BW,BR)
e8=(enbr 2)  ---> s8=(ID,ID)=s0

The root: s3=(BW,ID)
e11=(enbw 1)  ---> s11=(ID,ID)=s0
e12=(esbr 2)=e2 ---> s12=(BW,BR)=s10
e13=(esbw 2)=e4 ---> s13=(BW,BW)

The root: s4=(ID,BW)
e15=(esbr 1)=e1 ---> s15=(BR,BW)=s7
e16=(esbw 1)=e3 ---> s16=(BW,BW)=s13
e14=(enbw 2)  ---> s14=(ID,ID)=s0
```

Figure 6.6: Reachability computation result of the disk model.

$enbw(d)$ represent the commencement and end of being written, respectively, of the disk. Then, the specifications of the disk are as follows.

$$\Box[esbr(d) \wedge ID(d) \Rightarrow BR(\bigcirc d)] \qquad (6.3.M1)$$

$$\Box[enbr(d) \wedge BR(d) \Rightarrow ID(\bigcirc d)] \qquad (6.3.M2)$$

$$\Box[esbw(d) \wedge ID(d) \Rightarrow BW(\bigcirc d)] \qquad (6.3.M3)$$

$$\Box[enbw(d) \wedge BW(d) \Rightarrow ID(\bigcirc d)] \qquad (6.3.M4)$$

The reachability graph routines together with the enabled event sets of the disk model is given in Figure 6.6. Now we can visualize all the components of the

h	g	i	k
$s_0 \rightarrow s'_0$	$e_1 e_5 \rightarrow e'_1$	$1 \rightarrow 1$	$(6.2.P1) \wedge (6.2.P2) \rightarrow (6.3.M1)$
$s_5 \rightarrow s'_1$	$e_2 e_8 \rightarrow e'_2$		$(6.2.P3) \rightarrow (6.3.M2)$
$s_8 \rightarrow s'_2$	$e_3 e_{11} \rightarrow e'_3$		$(6.2.P4) \wedge (6.2.P5) \rightarrow (6.3.M3)$
$s_{11} \rightarrow s'_3$	$e_4 e_{14} \rightarrow e'_4$		$(6.2.P6) \rightarrow (6.3.M4)$
$s_{14} \rightarrow s'_4$	$e_{17} \rightarrow e'_5$		
$s_{38} \rightarrow s'_6$	$e_{26} \rightarrow e'_8$		
$s_{40} \rightarrow s'_7$	$e_{29} \rightarrow e'_{11}$		
$s_{45} \rightarrow s'_{10}$	$e_{36} \rightarrow e'_{14}$		
$s_{50} \rightarrow s'_{13}$			

Table 6.2: The homomorphism from the process model to the disk model.

Σ-homomorphism (h, g, i, k) from the two-processor model M to the disk model M' ($'$ is added to e and s of Figure 6.6) as shown in Table 6.2.

The composition of the disk and process models is shown graphically in Figure 6.7. This composition shows us that the composition operation could be used to build the closed-loop system from the plant and the controller; and it can be made by placing the controller in the place of the disk in the given example. This result provides us a basis for a reachability synthesis procedure for constructing a controller which, when composed with the plant, will impose the required behavior on the composite system.

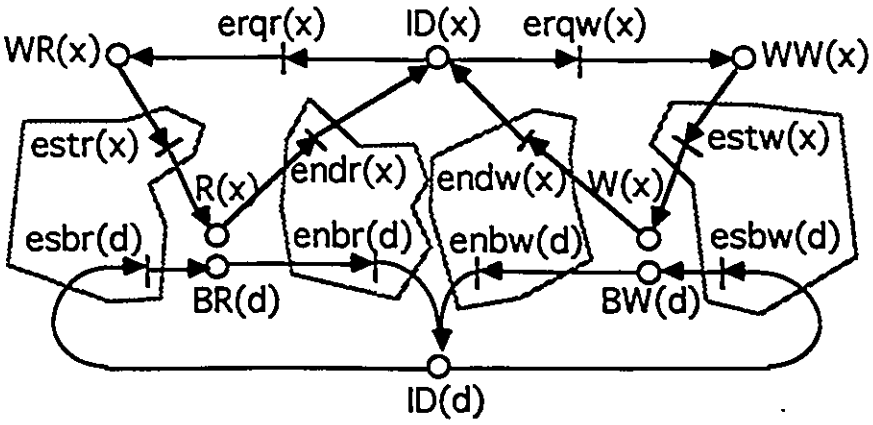


Figure 6.7: Composition of the disk and process models.

6.4 Controller Design and Synthesis Procedure

The behavior of a DES plant (uncontrolled system) is usually unsatisfactory in some aspects, a description of the required plant behavior is then imposed to prevent from the unsatisfactory behavior. The description can be formulated in temporal logic formula specifications; and it usually is a restriction on the reachability set of the system such that the unsatisfactory states may not be reached. It is the task of the controller to ensure that unsatisfactory behavior in the plant is eliminated.

As in [95], the event set can be partitioned into the disjoint sets as the set of controllable events and the set of uncontrollable events, and then define that controllable events are those which an external agent may enable or disable while uncontrollable events are those which cannot be prevented from occurring. This is

an important issue; but it will complicate our work too much. For the simplicity, we assume the existence of a controller for the given system. This simply means that the controller only enable or disable those events which it may enable or disable. We simply do not talk about the uncontrollable events and do not assume that all events are subject to disablement. In this section, the controller is designed according to the required behavior of the closed-loop system. Our controller synthesis here is to synthesize the controller designed by a set of rules of temporal logic formulas such that the closed-loop system has the desired reachability set. This may be called reachability synthesis. The synchronization of events between the system and its controller is made by synchronizing the events in the controller with those in the system.

Let M denote the TLM of the plant and M_c denote the TLM of the controller, as yet to be constructed according to the specifications of the required behavior. We propose that the two systems operate concurrently and that control action is achieved by the synchronization of the events in the plant with the events in the controller. Hence the behavior of the initially uncontrolled plant is restricted since certain selected events are prevented from occurring unless the controller is also in the state which allows the corresponding synchronized event to occur.

Let $M = (\{\mathbf{S}, \mathbf{E}, \mathbf{F}^*\}, \{f, l, p, s_0\})$ and let $M_c = (\{\mathbf{Q}, \mathbf{E}, \mathbf{F}_c^*\}, \{f_c, l_c, p_c, q_0\})$. Then the closed-loop system of the plant M and the controller M_c is defined as follows.

Definition 6.4.1: The closed-loop system of M and M_c, written as $\overline{M} = M\|M_c$, is defined by replacing M' in Definition 6.3.3 by M_c; and in addition, for all $s \in \mathbf{S}$ and $q \in \mathbf{Q}$ with $\bar{s} \stackrel{\text{def}}{=} (s,q)$, $e \in E_{\bar{s}}$ if and only if $e \in E_s$ and $e \in E_q$.

Then the closed-loop system $\overline{M} = M\|M_c$ has the set of states $\overline{\mathbf{S}} = \mathbf{S} \times \mathbf{Q}$, the set of events $\overline{\mathbf{E}} = \mathbf{E}$, the set $\mathbf{F}^* \cup \mathbf{F}_c^*$, the transition function $\bar{f}$, the labelling function is $\bar{l} = (l, l_c)$, and the initial state (s_0, q_0). As shown in Chapter 3, the controller is deterministic; and therefore, we have $\bar{p} = p$. Hence, we obtain $\overline{M} = (\mathbf{S} \times \mathbf{Q}, \mathbf{E}, \mathbf{F}^* \cup \mathbf{F}_c^*, \bar{f}, \bar{l}, p, (s_0, q_0))$; and the following result is obtained by replacing M' in Theorem 6.3.4 as M_c.

Corollary 6.4.1: Let $\overline{M} = M\|M_c$ as stated above. Then

(1): $(s', q') = \bar{f}(e, (s, q))$ is in $\overline{M}$ if and only if both $s' = f(e, s)$ is in M and $q' = f_c(e, q)$ is in M_c; and

(2): $\bar{s}' = (s', q')$ is reachable in $\overline{M} = M\|M_c$ if and only if both s' is reachable in M and q' is reachable in M_c.

Proof: (1): *Necessity.* It follows from (2) of Theorem 6.3.4.

Sufficiency. Since both $s' = f(e, s)$ is in M and $q' = f_c(e, q)$ is in M_c, it follows from Definition 6.4.1 that $e \in E_{(s,q)}$. Hence $(s', q') = \bar{f}(e, (s, q))$ is in $\overline{M}$.

(2). *Necessity.* It follows from (3) of Theorem 6.3.4.

Sufficiency. Since both s' is reachable in M and q' is reachable in M_c, by Definitions 5.3.1 and 5.3.2, there exist $s \in \mathbf{S}$, $e \in E_s$, and $q \in \mathbf{Q}$, $e' \in E_q$ such that $s' = f(e, s)$ and $q' = f_c(e', q)$. By the projections from $\overline{M}$ to M and

M_c, $\bar{s}' = (s', q')$. Using Definition 6.4.1, we have $\bar{e} = (e, e') \in E_{(s,q)}$. Hence, $\bar{f}(\bar{e}, (s, q)) = (f(e, s), f_c(e', q)) = (s', q') = \bar{s}' \in \overline{M}$ and $\bar{e} \in E_s$. By Definitions 5.3.1 and 5.3.2, we obtain that $\bar{s}'$ is reachable in $\overline{M} = M \| M_c$. ■

By Definition 5.3.2, the reachability set $R(\overline{M}, (s_0, q_0))$ is the smallest set defined by

- $(s_0, q_0) \in R(\overline{M}, (s_0, q_0))$;
- If $(s, q) \in R(\overline{M}, (s_0, q_0))$ and $(s', q') = \bar{f}(e, (s, q))$ for some $e \in E_s \cap E_q$, then $(s', q') \in R(\overline{M}, (s_0, q_0))$.

It follows from Corollary 6.4.1 that the reachability set of $\overline{M}$ is the direct product of the reachable set of M and that of M_c. Here the states in $\overline{M}$ are composed of the states of M and M_C corresponding to the same events.

Using the obtained results, we develop a procedure in the following, for a given TLM of a DES plant M, for designing a controller M_c and reachability synthesis of M_c with M to achieve the required behavior of the closed loop system $\overline{M}$.

Procedure 6.4.1:

(*i*): Give the specification of the plant M and compute the reachability set $R(M, s_0)$;

(*ii*): Specify the required behavior of the closed-loop system $\overline{M}$ by temporal logic formulas;

(*iii*): Find out the set of states which should not be reached by the closed-

loop system according to (ii). Denote the set by R_u and then list the events which lead to these states;

(iv): Construct the controller M_c to prevent those events given in (iii) from occurring such that R_u may not be reached, and then compute the reachability set $R(M_c, q_0)$ of the controller;

(v) Synchronize the events in the controller M_c with the same events in the plant M; and compose M and M_c into the closed-loop system $\overline{M}$, and then compute the reachability set $R(\overline{M}, (s_0, q_0))$.

From the computational analysis of Algorithm 5.3.1, at most $|\mathbf{E}| \cdot |R(M, s_0)|$, $|\mathbf{E}| \cdot |R(M_c, q_0)|$, and $|\mathbf{E}| \cdot |R(\overline{M}, (s_0, q_0))|$ steps are needed to compute the reachability graphs for the plant, the controller, and the closed-loop system, respectively. It follows from Procedure 6.4.1 that the reachability graphs for the controller and the closed-loop system are smaller than the reachability graph of the plant. Hence, at most $3|\mathbf{E}| \cdot |R(M, s_0)|$ steps are needed to complete the computations in Procedure 6.4.1. Using the same analysis as that after Algorithm 5.3.1, we can obtain that Procedure 6.4.1 is linear in the size of the reachability graph $R(M, s_0)$. The complexity of the system is getting more complicated as its size gets larger. However, practically we always decompose the system into a number of smaller subsystems that are tractable.

Theorem 6.4.2: Procedure 6.4.1 can be used to do the controller synthesis of DESs; and at the same time, it does the verification as Procedure 4.2.1 does in Chapter 4.

Proof: Procedure 6.4.1 is developed based on the results obtained in the last chapter and this chapter so that it can be used to do the controller synthesis of DESs. As Procedure 6.4.1 does the synthesis, by Corollary 6.4.1, $R(\overline{M}, (s_0, q_0)) = R(M, s_0) \times R(M_c, q_0)$ in a sense that a state of $\overline{M}$ is composed of states from M and M_c corresponding to the same event. It follows from Steps (*iii*) and (*iv*) that

$$R(M, s_0) \times R(M_c, q_0) \rightleftharpoons R(M, s_0) - R_u$$

Hence $R(\overline{M}, (s_0, q_0)) \rightleftharpoons R(M, s_0) - R_u$ where $\rightleftharpoons$ stands for the isomorphic relation. This proves that the composite system model $\overline{M}$ will have the required behavior given by Step (*ii*). By Corollary 5.3.1, it is equivalent to verify that the required behavior of the closed-loop system $\overline{M}$ given by Step (*ii*) can be deduced from the specification of M given in Step (*i*) and the specification of the controller given in Step (*iv*). Therefore Procedure 6.4.1 also does the verification as Procedure 4.2.1 does in Chapter 4. In other words, it is equivalent to Procedure 4.2.1. ■

Now Procedure 6.4.1 can be used to the controller synthesis of DESs in the temporal logic approach; and at the same time it verifies that the controller ensures the required behavior of the system. Procedure 6.4.1 is implemented in the software developed in Chapter 8. It gives the reachability computations of the plant, the controller, and the closed-loop system. According to the specifications of the required behavior, its monitor will produce a list of warning on which states should not be reached and suggestions on which events should be disabled. Then it synthesizes the controller into the system.

To illustrate the use of Procedure 6.4.1, we return to the given example. We need to impose some control on the system so that access is regulated according the following requirements:

- Any number of processes can read from the disk simultaneously;
- Only one process can write to the disk at any time; and
- If a process is writing to the disk then no process may simultaneously read from the disk.

Let $EQ(x, y)$ represent that x is identical to y. Following Procedure 6.4.1, we first express the required behavior of the system as follows:

$$\Box[W(x) \wedge \neg EQ(x, y) \Rightarrow \neg W(y) \wedge \neg R(y)] \qquad (6.4.CL1)$$

$$\Box[R(x) \wedge \neg EQ(x, y) \Rightarrow \neg W(y))] \qquad (6.4.CL2)$$

where $\neg$ is the negation.

According to the specifications, the monitor found that $R_u = \{s_{40}, s_{45}, s_{50}\}$ and that which events such that these states are unreachable. These are shown in Figure 6.8. The next step in our procedure is to construct a controller to disable those events such that states s_{40}, s_{45}, and s_{50} will not be reached in the closed-loop system. Following the warnings and suggestions given by the monitor, we design a controller to prevent the events that lead to the states in R_u from occurring.

The events to be controlled are those which result in a process going from the waiting for reading state to the reading state and from the waiting for writing

```
According to the specifications of the required behavior

s40 should not been reached! So event e40 should be disabled at
state s19 !
s43=(R,W)=s40 So event e43 should be disabled at state s23 !
s45 should not been reached! So event e45 should be disabled at
state s25 !
s48=(W,R)=s45 So event e48 should be disabled at state s27 !
s50 should not been reached! So event e50 should be disabled at
state s31 !
s51=(W,W)=s50 So event e51 should be disabled at state s33 !
s40 may not be a root!
s45 may not be a root!
s50 may not be a root!

Then the set of states which should not be reached are:
s40=(R,W) s45=(W,R) s50=(W,W)
```

Figure 6.8: List of warnings and suggestions given by the monitor.

state to the writing state. In order for the controller to know when to permit a process to go from a waiting for reading state to a reading state, for example, it must also monitor the transition of processes out of the writing state and out of the reading state. Hence, the events to be selected and synchronized are those which result in processes entering and leaving the reading and writing states.

We use additional local variable symbols u and v to represent the data stored by the controller. u is assigned values 1 and 0 representing if a process is writing to the disk or not; and v is assigned non-negative integer values representing the number of processes which are reading from the disk. Let $ADD1(v)$ stand for that v is increased by 1 and $MIN1(v)$ for that v is decreased by 1. Then, the specifications of the controller are as follows.

Dynamics of the Controller

$$\Box[estr(x) \wedge EQ(u,0) \Rightarrow ADD1(\bigcirc v) \wedge EQ(\bigcirc u,0)] \qquad (6.4.C1)$$

$$\Box[endr(x) \wedge EQ(u,0) \wedge \neg EQ(v,0) \Rightarrow MIN1(\bigcirc v)] \qquad (6.4.C2)$$

$$\Box[estw(x) \wedge EQ(u,0) \wedge EQ(v,0) \Rightarrow EQ(\bigcirc u,1) \wedge EQ(\bigcirc v,0)] \qquad (6.4.C3)$$

$$\Box[endw(x) \wedge EQ(u,1) \wedge EQ(v,0) \Rightarrow EQ(\bigcirc u,0)] \qquad (6.4.C4)$$

Initial Condition of the Controller

$$EQ(u,0) \wedge EQ(v,0) \qquad (6.4.C5)$$

```
The transitions in the controller are:

The root:   C0 =(0,0)
e1=(estr 1)   --->    C1 =(0,1)
e2=(estr 2)   --->    C2 =(0,1) = C1
e3=(estw 1)   --->    C3 =(1,0)
e4=(estw 2)   --->    C4 =(1,0) = C3
The root:   C1 =(0,1)
e5=(endr 1)   --->    C5 =(0,0) = C0
e6=(endr 2)   --->    C6 =(0,0) = C0
e7=(estr 1)   --->    C7 =(0,2)
e8=(estr 2)   --->    C8 =(0,2) = C7
The root:   C3 =(1,0)
e9=(endw 1)   --->    C9 =(0,0) = C0
e10=(endw 2)   --->    C10 =(0,0) = C0
The root:   C7 =(0,2)
e11=(endr 1)   --->    C11 =(0,1) = C1
e12=(endr 2)   --->    C12 =(0,1) = C1
```

Figure 6.9: The state transitions of the controller.

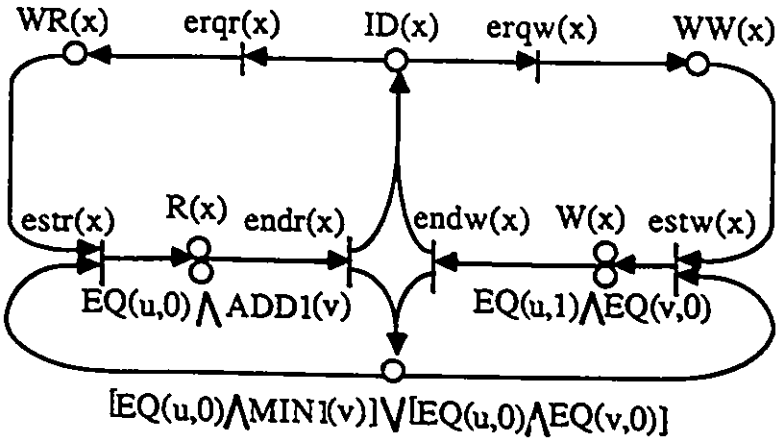


Figure 6.10: Graph of the composite system of the example.

The state transitions of the controller are given in Figure 6.9 and the controller synthesis procedure is illustrated graphically in Figure 6.10. The closed-loop system is composed of the plant and the controller. Its dynamic behavior is simulated by the software developed in Chapter 8 as shown in Figure 6.11 and its reachability graph is constructed in Figure 6.12.

By using Procedure 6.4.1, according to the required behavior of the system, we have designed and synthesized a controller, and we have obtained the behavior of the closed-loop system. From Figures 6.3, 6.8, and 6.11, we can see that $R(\overline{M},(s_0,q_0)) \rightleftharpoons R(M,s_0) - R_u$. This proves that the resulting composite system has the required behavior. By Procedure 6.4.1 and Theorem 6.4.2, we have also verified that the required behavior of the closed-loop system given by (6.4.$CL1$) – (6.4.$CL2$) can be deduced from the specification of the two processor given in (6.2.$P1$) – (6.2.$P7$) and the specification of the controller given in

(6.4.$C1$) – (6.4.$C5$). This example is a deterministic case; and a similar procedure can be carried out for the probabilistic DES such as the example in Chapter 4.

As we can see from this example, the controller synthesis procedure works like a formal verification. As it takes the specification of the plant and synthesizes a controller according to the required behavior description, it proves that the required behavior description can be deduced from the specification of the plant and the controller design by using the reachability algorithm. The reachability properties verify the validity of the temporal logic formulas such as the form (5.2.3) for the deterministic case and the form (5.2.4) for the nondeterministic case.

6.5 Discussion

By applying Hennessy's process algebra, TLMs have been viewed as Σ–algebras together with the Σ–homomorphisms between them, and a composition operator has been then defined to provide a basis for the composition of a number of TLMs. Control action has been achieved by the synthesis of a controller with the plant. The controller has been designed through the reachability properties and imposed by synchronization of events in the controller with the selected events in the plant. The synthesis procedure guarantees the safety assertions such as the reachability properties.

An example of a system of computer read-write processes has been demonstrated to propose the need for the composition and synthesis of a controller into

Now the state transitions of the controlled system are:

The root: S0 =(ID,ID,0,0)
e1=(erqr 1) —> S1 =(WR,ID,0,0)
e3=(erqw 1) —> S3 =(WW,ID,0,0)
e2=(erqr 2) —> S2 =(ID,WR,0,0)
e4=(erqw 2) —> S4 =(ID,WW,0,0)

The root: S1 =(WR,ID,0,0)
e5=(estr 1) —> S5 =(R,ID,0,1)
e6=(erqr 2)=e2 —> S6 =(WR,WR,0,0)
e7=(erqw 2)=e4 —> S7 =(WR,WW,0,0)

The root: S2 =(ID,WR,0,0)
e9=(erqr 1)=e1 —> S9 =(WR,WR,0,0)=S6
e10=(erqw 1)=e3 —> S10 =(WW,WR,0,0)
e8=(estr 2) —> S8 =(ID,R,0,1)

The root: S3 =(WW,ID,0,0)
e11=(estw 1) —> S11 =(W,ID,1,0)
e12=(erqr 2)=e2 —> S12 =(WW,WR,0,0)=S10
e13=(erqw 2)=e4 —> S13 =(WW,WW,0,0)

The root: S4 =(ID,WW,0,0)
e15=(erqr 1)=e1 —> S15 =(WR,WW,0,0)=S7
e16=(erqw 1)=e3 —> S16 =(WW,WW,0,0)=S13
e14=(estw 2) —> S14 =(ID,W,1,0)

The root: S5 =(R,ID,0,1)
e17=(endr 1) —> S17 =(ID,ID,0,0)=S0
e18=(erqr 2)=e2 —> S18 =(R,WR,0,1)
e19=(erqw 2)=e4 —> S19 =(R,WW,0,1)

The root: S6 =(WR,WR,0,0)
e20=(estr 1)=e5 —> S20 =(R,WR,0,1)=S18
e21=(estr 2)=e8 —> S21 =(WR,R,0,1)

The root: S7 =(WR,WW,0,0)
e22=(estr 1)=e5 —> S22 =(R,WW,0,1)=S19
e23=(estw 2)=e14 —> S23 =(WR,W,1,0)

The root: S8 =(ID,R,0,1)
e24=(erqr 1)=e1 —> S24 =(WR,R,0,1)=S21
e25=(erqw 1)=e3 —> S25 =(WW,R,0,1)
e26=(endr 2) —> S26 =(ID,ID,0,0)=S0

The root: S10 =(WW,WR,0,0)
e27=(estw 1)=e11 —> S27 =(W,WR,1,0)
e28=(estr 2)=e8 —> S28 =(WW,R,0,1)=S25

The root: S11 =(W,ID,1,0)
e29=(endw 1) —> S29 =(ID,ID,0,0)=S0
e30=(erqr 2)=e2 —> S30 =(W,WR,1,0)=S27
e31=(erqw 2)=e4 —> S31 =(W,WW,1,0)

The root: S13 =(WW,WW,0,0)
e32=(estw 1)=e11 —> S32 =(W,WW,1,0)=S31
e33=(estw 2)=e14 —> S33 =(WW,W,1,0)

The root: S14 =(ID,W,1,0)
e34=(erqr 1)=e1 —> S34 =(WR,W,1,0)=S23
e35=(erqw 1)=e3 —> S35 =(WW,W,1,0)=S33
e36=(endw 2) —> S36 =(ID,ID,0,0)=S0

The root: S18 =(R,WR,0,1)
e37=(endr 1)=e17 —> S37 =(ID,WR,0,0)=S2
e38=(estr 2)=e8 —> S38 =(R,R,0,2)

The root: S19 =(R,WW,0,1)
e39=(endr 1)=e17 —> S39 =(ID,WW,0,0)=S4

The root: S21 =(WR,R,0,1)
e41=(estr 1)=e5 —> S41 =(R,R,0,2)=S38
e42=(endr 2)=e26 —> S42 =(WR,ID,0,0)=S1

The root: S23 =(WR,W,1,0)
e44=(endw 2)=e36 —> S44 =(WR,ID,0,0)=S1

The root: S25 =(WW,R,0,1)
e46=(endr 2)=e26 —> S46 =(WW,ID,0,0)=S3

The root: S27 =(W,WR,1,0)
e47=(endw 1)=e29 —> S47 =(ID,WR,0,0)=S2

The root: S31 =(W,WW,1,0)
e49=(endw 1)=e29 —> S49 =(ID,WW,0,0)=S4

The root: S33 =(WW,W,1,0)
e52=(endw 2)=e36 —> S52 =(WW,ID,0,0)=S3

The root: S38 =(R,R,0,2)
e53=(endr 1)=e17 —> S53 =(ID,R,0,1)=S8
e54=(endr 2)=e26 —> S54 =(R,ID,0,1)=S5

The reachability Set:
S0 =(ID,ID,0,0) S1 =(WR,ID,0,0) S2 =(ID,WR,0,0)
S3 =(WW,ID,0,0) S4 =(ID,WW,0,0) S5 =(R,ID,0,1)
S6 =(WR,WR,0,0) S7 =(WR,WW,0,0) S8 =(ID,R,0,1)
S10 =(WW,WR,0,0)S11=(W,ID,1,0)S13=(WW,WW,0,0)
S14 =(ID,W,1,0) S18 =(R,WR,0,1) S19 =(R,WW,0,1)
S21 =(WR,R,0,1) S23 =(WR,W,1,0) S25 =(WW,R,0,1)
S27 =(W,WR,1,0) S31 =(W,WW,1,0) S33=(WW,W,1,0)
S38 =(R,R,0,2)

Figure 6.11: The simulation results of the composite system.

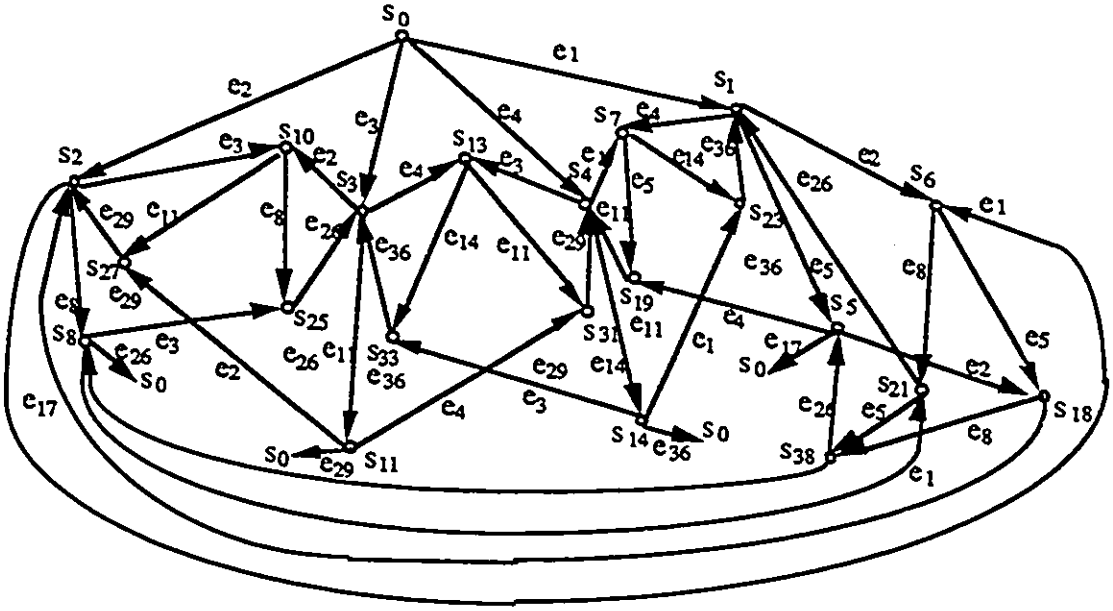


Figure 6.12: The reachability graph of the composite system.

the system. Results of simulating this example have been given. The simulation was performed using a simulation tool which will be described in Chapter 8.

As a comparison, our work in this chapter is close to Denham's work [21], among the related work. He uses the two-sorted algebra to do the composition of DESs in the Petri nets approach while we use Σ-algebra in the temporal logic approach. He uses invariants of the Petri nets to specify the system behavior while we use reachability of TLMs. The disadvantage of his approach is that an additional language is needed to express the required behavior over a sequence of states. Our temporal logic language allows formal statements to be made which are interpreted over the sequence of states that constitute the dynamic behavior of the system. These can be seen by comparing the results of the read-write example in this chapter with the results of the same example in [21].

Another work to be compared is that by Manna and Wolper [75]. They applied a propositional temporal logic to the specification and synthesis of the synchronization part of communicating processes. They abstracted concurrent computations to sequences of 'events' and described these sequences using propositional temporal logic, and then synthesized the processes using a tableau decision procedure. Apparently both our method and theirs do the synthesis from a temporal logic description of the required behavior by applying a mechanical procedure. However, the methods differ substantially in terms of the models, orientations and the techniques used to realize the concepts. They synthesize communicating sequential process (CSP) [35] programs such that their model is based on mes-

sage passing primitives in a distributed computing environment whereas ours is oriented toward a plant-controller discrete event system in temporal logic models. They use a tableau-like satisfiability algorithm to construct a model of the given specifications while we use our reachability algorithm to synthesize a closed-loop system for the given description of the required behavior.

Our work here uses Σ-algebra cited from Hennessy [32] which is related to the work of the basic theorems about general *algebras* given by Birkhoff and Lipson such as those given in [8]. Our composition of temporal logic models has the ideas similar to their direct products and powers of algebras in [8]. Their purpose was to show that some algebras can be studied both as homogeneous algebras and as heterogeneous algebras while our objective is to build a controller synthesis procedure for DESs.

As we can see from the reachability graphs in the example, the controller designed is preventive in the sense that it prevents the system from reaching the bad-state. Hence, there might be more than one trajectory from the given initial state to the given final state. There may be one, among these trajectories, along which the system can achieve an optimal cost index in some sense. Being motivated by this, in the next chapter, we will investigate the problem of optimization of the controller design for DESs in a temporal logic framework.

Chapter 7

Optimization for DESs

In this chapter, we investigate the optimization problems of discrete event systems. The temporal logic model defined in Chapter 3 is enhanced with a cost function and a semi-metric space is also introduced to measure the cost function. Then the optimization problem is solved using a heuristic search method.

7.1 Introduction

Figures 6.11 and 6.12 show that there might be more than one trajectory from the given initial state to the given final state within the reachability set. Among these trajectories, there might be one along which the cost of executing events would be the smallest. This proposes the need for a procedure of optimization for DESs. In this chapter, we will investigate the optimization problem of DESs in the temporal logic framework.

As mentioned in Chapter 1, the optimization problems have been studied for

DESs. Ramadge and Wonham [95] have considered the optimality of the supremal controllable sublanguage of a given language. The graph-theoretical formulation of the optimal control problem for a class of DESs has been given by Sengupta and Lafortune [100], and the supervisory optimal control has been studied in [55]. Passino and Antsaklis [89] have associated a cost with every state transition $x \rightarrow x'$ of a logical DES and examined optimality with respect to an event cost function $\chi(x, x')$. However, among the growing literature that discusses the optimization for DESs, little work has been done in a temporal logic approach.

This chapter trends towards an optimization for DESs in a temporal logic framework. A controlled DES in a temporal logic framework consists of a plant and a controller. Our motivation is to design a controller to generate a sequence of events to drive the system from the given initial state to the given final state and minimize a given cost function index. Thus, the objective of this chapter is to develop a procedure of optimization for DESs in a temporal logic framework.

This chapter is organized as follows. In the next section, the issues of active controller and forcible events are discussed to propose the active control for forcing the forcible events to occur in order to obtain the optimization. In Section 7.3, the temporal logic model is enhanced by including the cost function and a semi-metric space is defined. In Section 7.4, the optimization problem of DESs is formulated; and then the optimization procedure is proposed through a heuristic search method in Section 7.5. In Section 7.6, an example of applications is shown. Finally in Section 7.7, as a conclusion of this chapter, our results are compared

with the related works and topics for further research are suggested.

7.2 Active Control and Forcible Events

The controllers designed in the previous chapter are used to prevent the events that lead to bad states from occurring so that the safety properties are ensured. These controllers are passive in the sense that they do not force events to occur but merely approve or disapprove event occurrences.

For a passive controller, at a given state s, there is a possibility of occurrence of one among several events in the enabled event set E_s. In general, the controller merely restricts the occurrence of the next event to a subset of E_s; and the precise choice of the next event among the enabled events is not determined by the controller.

For instance, the controller in the example of read-write processes in Chapter 6, which is designed by Procedure 6.4.1, is passive. It has been designed to prevent events e_{40}, e_{43}, e_{45}, e_{48}, e_{50}, e_{51} from occurring because they will lead to states s_{40}, s_{45}, s_{50} which do not satisfy the required behavior of the closed-loop system. The controller disapprove those events such that these states are not reachable by the closed-loop system to ensure the safety assertions.

On the other hand, the controller approves all events e_1, e_2, e_3, e_4 to occur at s_0. As a consequence, there are a number of paths from s_0 to s_{18}; and all these

paths stay within the reachability set.

If a controller can be designed not only to prevent events from occurring but also to force them to occur, then the controller is said to be active (*forceful* in terminology of [79]).

For an active controller, among the events in the enabled event set E_s at state s, the next event to occur is uniquely determined by the controller. In this sense, an active controller may be used to drive the system along a prescribed trajectory or path.

From a practical point of view, there might be a path along which the cost of executing events is smallest among those paths that stay within the reachability set of the system. If these events can be forced to occur (*forcible* in terminology of [30]), then we may design an active controller to force them to occur.

When we say an event is forcible, we mean that the event can be forced to occur if necessary. On the other hand, for an event e in the enabled event set E_s of state s, if all the events other than e in E_s can be prevented from occurring, then e will be forced to occur since one and only one of the events in E_s has to occur at state s. In real world problems, one may start from the system and/or controller design to make the prescribed event forcible. This might be an interesting topic for the further research.

In order to do the optimization, in this chapter, we assume that all events in the system can be forced to occur, or we only force those events that can be forced to occur, so that an active controller can be designed. We also assume that the execution of an event will result in exactly one state, that is, the given DES is deterministic so that in the temporal logic model (TLM) $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, p, s_0)$ i.e. $J = \{j\}$ and $p_j = 1$. Hence, in this chapter, we omit p from TLMs and write $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, s_0)$.

7.3 Temporal Logic Model in a Semi-Metric Space

In order to measure the cost of driving the system from one state to another by executing an event and the cost of maintaining the system at a state, we need a measurement which is similar to the distance function and the metric space defined in the functional analysis [48]. Because of the symmetry of the distance function, we can not use it and the metric space associated with it to measure the above costs. For example, the cost of driving the system from s_1 to s_2 may be different from that from s_2 to s_1. In this case, we want to chose the heuristic function to be non-symmetric so that the values of the heuristic function can be close to the values of the cost function which is non-symmetric.

A precedence relation $\preceq$ on a set X is defined as that for all $x, y, z \in X$ there hold $x \preceq x$, and $(x \preceq y,\ y \preceq z) \Rightarrow x \preceq z$. Obviously, the precedence relation $\preceq$ is reflexive and transitive. Then we define a semi-metric space associated with a semi-metric function as follows.

Definition 7.3.1: A semi-metric space is defined by a pair (X, m) where X is an arbitrary non-empty set with $\preceq$ as a precedence relation on X, and $m : X \times X \to R^+$ is a semi-metric function on X, that is, for all $x, y, z \in X$, m has the following properties:

(i). $m(x, y) \geq 0$ for $x \preceq y$;

(ii). $m(x, x) = 0$;

(iii). $m(x, z) \leq m(x, y) + m(y, z)$ for $x \preceq y \preceq z$.

It is noticed that this definition is different from the *semimetric space* associated with the *semimetric function* defined by Wilansky in [108]. The difference is that the distance function $d(a, b)$ Wilansky defined there is symmetric while our $m(x, y)$ is not necessarily symmetric and we have a precedence relation $\preceq$ on the elements of X. Hence, our semi-metric space is more general than the *semimetric space* defined in [108] which is more general than the metric space defined in [48]. Therefore, any metric space is a *semimetric space* which is a semi-metric space. So, our definition of the semi-metric space is a generalization of that of the metric space.

Here is an example of semi-metric spaces:

For any non-empty set X having countable elements, we can count its elements and assign a positive integer to each element. This produces a sequence $x_1 x_2 x_3 \cdots$ and gives a precedence relation $\preceq$ on X. For any integer i, j with $j \geq i + 1$

$$m(x_i, x_j) \stackrel{\text{def}}{=} j - i \tag{7.3.1}$$

then $\{X; m\}$ is a semi-metric space.

Definition 7.3.2: A cost function $\theta : \mathbf{E} \times \mathbf{S} \rightarrow [0, \infty)$ is defined by $\theta_{ij} \stackrel{\text{def}}{=} \theta(e_{ij}, s_i)$ for $e_{ij} \in E_{s_i}$ such that $s_j = f(e_{ij}, s_i)$. The cost function is defined iff the event $e_{ij} \in E_{s_i}$.

The cost function $\theta(e_{ij}, s_i)$ includes the cost of maintaining the system at s_i and the cost of executing the event e_{ij} at s_i. Obviously, $\theta(\epsilon, s)$ for any $s \in \mathbf{S}$ is the cost of maintaining the system at s where ϵ is a null event. Thus an enhanced temporal logic model $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, \theta, s_0)$ has been obtained.

Theorem 7.3.1: Given a TLM $M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, \theta, s_0)$, for a trajectory $s_0 \stackrel{e1}{\rightarrow} s_1 \stackrel{e2}{\rightarrow} s_2 \cdots \stackrel{en}{\rightarrow} s_n$, let $e_{1n} = e_1 e_2 \cdots e_n$. Then

$$\theta(e_{1n}, s_0) \leq \theta(e_1, s_0) + \theta(e_2, s_1) + \cdots + \theta(e_n, s_{n-1}) = \sum_{i=0}^{n-1} \theta(e_{i+1}, s_i) \qquad (7.3.2)$$

Proof: For the given trajectory of the TLM, it follows from Definition 3.2.1 that $f(e_1, s_0)$, $f(e_2, s_1)$, $\cdots$, $f(e_n, s_{n-1})$ exist. By Definition 7.3.2, the cost functions $\theta(e_1, s_0)$, $\theta(e_2, s_1)$, $\cdots$, $\theta(e_n, s_{n-1})$ is defined. By Definition 5.2.1, $e_{1k} = e_1 e_2 \cdots e_k \in E_{s_0}$; and therefore $\theta(e_{1n}, s_0)$ is defined. (7.3.2) follows from Definitions 5.2.1 and 7.3.2. ■

Since executing an enabled event represents an action performed, $\theta_{ij} = \theta(e_{ij})$ is a measure of the cost to process the state s_i into s_j through executing e_{ij}. The

execution of an enhanced TLM results in three sequences: a sequence of states, a sequence of events, and a sequence of the costs. It can be represented explicitly by a Θ-graph:

- $\mathbf{S} = \{s_0 s_1 s_3 \cdots\}$ is the non-empty set of nodes;
- $\mathbf{E} = \{e_{ij}\} = \{(s_i, s_j);\ s_i, s_j \in \mathbf{S}\}$ is the non-empty set of directed arcs with e_{ij} pointing from s_i to s_j;
- $\Theta = \{\theta_{ij}\} = \{\theta_{ij} = \theta(e_{ij}, s_i);\ e_{ij} \in E_{s_i},\ s_i \in \mathbf{S}\}$ is the non-empty set of costs associated with each arc e_{ij} and node s_i, and $\theta_{ij} \geq 0$ for all $\theta_{ij} \in \Theta$.

This Θ-graph is an extension of the graphical representation of a TLM given in Chapter 3. The enhanced TLM is used for the representation as a model of the DES plant. It allows for the specification of a cost to execute an event via the specification of the cost function. Such cost could, for example, represent a measure of the resources consumed in performing the actions associated with executing an event.

7.4 Optimization Problem of DESs

In system and control theory, optimization has been an important part in the controller synthesis and optimal control [1,111]. This section will establish the first step towards developing the foundations of the optimization for DESs in a temporal logic approach.

7.4.1 Synthesis of Plant and Controller

In the previous section, temporal logic model has been enhanced by defining a cost function index in terms of the costs for executing events and maintaining the system at the corresponding states. In addition, for a given final state s_g, a DES plant is modelled by

$$M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, l, \theta, s_0, s_g)$$

where $\mathbf{S}$ is the set of states; $\mathbf{E}$ is the set of events; $f : \mathbf{E} \times \mathbf{S} \rightarrow \mathbf{S}$ is the mapping; s_0 is the initial state; $l : \mathbf{E} \times \mathbf{S} \rightarrow \mathbf{F}^*$ is a labelling function; $\mathbf{F}^*$ is the set of all subsets of the set of formulas $\mathbf{F}$; $\theta : \mathbf{E} \times \mathbf{S} \rightarrow [0, \infty)$ is the cost function; s_g is the final state which is reachable from s_0.

We assume the existence of a controller for the above system; and we are going to design a controller to generate a sequence of events which drives the system along a trajectory that has the minimum of the cost. As discussed in Chapter 6, the controller is modeled by

$$M_c = (\mathbf{Q}, \mathbf{E}_c, \mathbf{F}_c^*, f_c, l_c, q_0)$$

where $\mathbf{Q}$ is the set of controller states; $\mathbf{E}_c \subset \mathbf{E}$; $f_c : \mathbf{E}_c \times \mathbf{Q} \rightarrow \mathbf{Q}$ is the mapping; q_0 is the initial controller state; $l_c : \mathbf{E}_c \times Q \rightarrow \mathbf{F}_c^*$ is a labelling function; and $\mathbf{F}_c^*$ is the set of all subsets of the set of formulas $\mathbf{F}_c$.

As shown in Chapter 6, the closed-loop system $\overline{M} = M \| M_c$ has the set $\mathbf{F}^* \cup \mathbf{F}_c^*$, the set of states $\overline{\mathbf{S}} = \mathbf{S} \times \mathbf{Q}$, and the set of events $\overline{\mathbf{E}} = \mathbf{E}$. The mapping in $\overline{M}$ is $\bar{f}$, and $e \in E_t$ if both $e \in E_s$ and $e \in E_q$ where $t = (s, q)$. (s_0, q_0) is the

initial state of $\overline{M}$. The labelling function is $\bar{l} = (l, l_c)$ for $\mathbf{F}^* \cup \mathbf{F}_c^*$ and the cost function is θ. Thus, we have $\overline{M} = (\mathbf{S} \times \mathbf{Q}, \mathbf{E}, \mathbf{F}^* \cup \mathbf{F}_c^*, \bar{f}, \bar{l}, \theta, (s_0, q_0))$.

In addition to the specifications of the required behavior of the system, a cost function index which will be defined in the following subsection is given to calculate the cost. So the controller is constructed in such a way to generate a sequence of events which drives the plant from the initial state s_0 to the final state s_g along the desirable path or trajectory. In the case that there are more than one such trajectories, the optimization of the cost function index is desired so that a sequence of events can be generated to drive the system from s_0 to s_g along the optimal trajectory and minimize the cost function index.

7.4.2 Optimal cost function index

When there are more than one path from s_0 to s_g, we take a subsequence of $\{0, 1, 2, \cdots, g\}$ as subscripts of the path $s_0 s_{k_1} s_{k_2} \cdots s_{k_{g-1}} s_g$. Then the cost function index is defined by

$$J(s_0, s_g) = \sum_{i=0}^{g-1} \theta(e_{k_{i+1}}, s_{k_i}) \tag{7.4.1}$$

for all trajectories starting from s_0 and ending at s_g. For simplicity, we write (7.4.1) as

$$J(s_0, s_g) = \sum_{i=0}^{g-1} \theta(e_{i+1}, s_i) \tag{7.4.2}$$

Obviously, $J(s_0, s_g) = 0$ if $s_g = s_0$.

Particularly, we are interested in the case of trajectories starting from s_0 and

ending in s_g. Let $M(s_0, s_g)$ denote the set of all finite trajectories $\sigma = s_0 s_1 \cdots s_g$ of M beginning with s_0 and ending with s_g. Our objective is to find a sequence of events that will produce a trajectory σ^* and minimize the cost function index. That is

$$h(s_0, s_g) \stackrel{\text{def}}{=} \inf_{\sigma \in M(s_0, s_g)} J(s_0, s_g)$$

where $J(s_0, s_g)$ is defined in (7.4.2).

For finite state DESs, there are only a finite number of trajectories of finite length. Hence we have the optimal index

$$h(s_0, s_g) \stackrel{\text{def}}{=} \min_{\sigma \in M(s_0, s_g)} J(s_0, s_g) \tag{7.4.3}$$

The optimization problem is to generate a sequence of events that drives the system M along the trajectory $\sigma^* \in M(s_0, s_g)$ such that (7.4.3) holds. Such a trajectory (path) σ^* is called the optimal trajectory (path). Once such σ^* and its corresponding sequence of events are found, a controller can be designed to generate this sequence of events to drive M along σ^*.

The cost function index (7.4.2) and the formulation of the optimization problem of DESs is similar to those in the conventional (continuous or discrete) systems. However, there are differences between them. For instance, we use events instead of the control variables in (7.4.2). This makes our statement of the optimization problem slightly different; and that is why we call it the optimization problem

instead of the optimal control problem. It has the advantage of adopting easily the heuristic searching methods, which will be presented in the next section, to solve the optimization problem.

7.5 Solution of the Optimization Problem

The purpose of the optimization is to produce a tentative procedure or guide for accomplishing some task satisfying the prescribed cost index. The optimization problem of DESs in a temporal logic framework usually involves searching a state space or its subset, for example, the reachability set, to find a path and the corresponding sequence of events with the minimum of the cost index. Here we use the theory of heuristic search using the A^* algorithm to solve the problem.

The Θ-graph of the enhanced TLM defined above can be used to the heuristic graph-search procedure [81]. An implicit representation of the Θ-graph G is given by a set of initial nodes and a successor operator $Z : \mathrm{S} \rightarrow 2^{\mathrm{S}\times\Theta}$ defined by $Z(s_i) \stackrel{\text{def}}{=} \{(s_{i+1}, \theta_{i,i+1}) \mid s_{i+1} = f(e_{i,i+1}, s_i),\ \theta_{i,i+1} = \theta(e_{i,i+1}, s_i),\ e_{i,i+1} \in E_{s_i}\}$. When Z is applied to a node s, the graph is expanded. A subgraph G_s from any $s \in \mathrm{S}$ can be obtained by applying Z to the initial node s. Each node in G_s is reachable from s. There exists a path from s_i to s_j iff s_j is reachable from s_i. Each path has a cost which is obtained by adding the costs $\theta_{i,i+1} \in \Theta$. An optimal path from s_i to s_j is a path having the smallest cost over the set of all paths from s_i to s_j. Denote this cost by $h(s_i, s_j)$ as in (7.4.3) and an estimate of it by $\hat{h}(s_i, s_j)$.

The objective is to find the optimal path from the initial state s_0 to the final state s_g. Certainly it can be solved by the exhaustive search methods such as breadth-first or depth-first. It has been shown [S1] that the search using these methods expands too many nodes before a path is found. So they are inefficient. To help guide the search, an evaluation function $v : \mathbf{S} \dashrightarrow R^+ \cup \{0\}$ is defined so that a node s with the smallest value of $v(s)$ is chosen for expansion. The A^* algorithm is used to perform heuristic search here.

Let $h(s) \stackrel{\text{def}}{=} h(s_0, s) + h(s, s_g)$ be the cost of an optimal path constrained to go through s from s_0 to s_g where $h(s_0, s)$ is the cost of an optimal path from s_0 to s and $h(s, s_g)$ is the cost of an optimal path from s to s_g as defined in (7.4.3). Since $h(s)$, $h(s_0, s)$ and $h(s, s_g)$ are not known, the estimates $\hat{h}(s)$, $\hat{h}(s_0, s)$, and $\hat{h}(s, s_g)$ are used. Hence,

$$v(s) = \hat{h}(s) = \hat{h}(s_0, s) + \hat{h}(s, s_g) \tag{7.5.1}$$

is chosen. The function $\hat{h}(s, s_g)$ is called the heuristic component of the evaluation function, the heuristic function, for short. If it satisfies certain conditions then the A^* algorithm performs the search well.

Definition 7.5.1: The A^* algorithm is said to be admissible if s_g is reachable from s_0 and $0 \leq \hat{h}(s, s_g) \leq h(s, s_g)$ for all $s \in \mathbf{S}$.

Definition 7.5.2: The heuristic $\hat{h}(s, s_g)$ is said to be consistent if $\hat{h}(s_i, s_g) - \hat{h}(s_j, s_g) \leq h(s_i, s_j)$ for all $s_i, s_j \in \mathbf{S}$.

Note that when $\hat{h}(s, s_g) \equiv 0$ in (7.5.1), the algorithm is identical to breadth-first search which is guaranteed to find an optimal solution but it is inefficient [S1]. When $\hat{h}(s, s_g) \not\equiv 0$ in the A^* algorithm, the optimization problem can be solved by using the heuristic function to guide the search.

The following theorem says that if the heuristic function $\hat{h}$ is a semi-metric function and is a low bound on the cost function θ for the given optimization problem of DESs, then we can find an admissible and consistent A^* algorithm that will generate a sequence of events to drive the system from the initial state to the final state with the minimum of cost.

Theorem 7.5.1: For the optimization problem given by

$$\begin{cases} M = (\mathbf{S}, \mathbf{E}, \mathbf{F}^*, f, s_0, l, \theta, s_g) \\ \min_{\sigma \in M(s_0, s_g)} \{\theta(\epsilon, s_g) + \sum_{i=0}^{g-1} \theta(e_{i+1}, s_i)\} \end{cases}$$

Suppose that the final state s_g is reachable from the initial state s_0. If the heuristic function $\hat{h}$ is a semi-metric function and satisfies $\hat{h}(s_i, s_j) \leq \theta(e_{ij}, s_i)$ for all $s_i, s_j \in \mathbf{S}$ such that $s_j = f(e_{ij}, s_i)$, then there exists an admissible and consistent A^* algorithm that can select a sequence of events to drive the system from the initial state s_0 to the final state s_g with the minimal cost.

Proof: The enhanced TLM has a Θ-graph representation. From the initial state s_0, the edges and the costs of the Θ-graph are generated by $Z(s_i) = \{(s_{i+1}, \theta_{i,i+1}) \mid s_{i+1} = f(e_{i,i+1}, s_i),\ e_{i,i+1} \in E_{s_i},\ \theta_{i,i+1} = \theta(e_{i,i+1}, s_i)\}$. Let $\hat{h}(s_i, s_j)$ be any semi-metric function which satisfies $\hat{h}(s_i, s_j) \leq \theta(e_{ij}, s_i)$ for all $s_i, s_j \in \mathbf{S}$ such

that $s_j = f(e_{ij}, s_i)$. Using $\hat{h}$ as the heuristic function, we need to prove that the A^* algorithm is admissible and consistent. By assumption that s_g is reachable from s_0, there exists an integer n such that $s_n = s_g$. Let $s_0 s_1 \cdots s_n$ be any path generated by the A^* algorithm. For all i, $0 \leq i \leq n-1$, $0 \leq \hat{h}(s_i, s_n) \leq \hat{h}(s_i, s_{i+1}) + \hat{h}(s_{i+1}, s_n)$ since $\hat{h}$ is a semi-metric function. Therefore

$$\hat{h}(s_i, s_n) \leq \sum_{k=i}^{n-1} \hat{h}(s_k, s_{k+1}) \leq \sum_{k=i}^{n-1} \theta(e_{k,k+1}, s_k) \leq \sum_{k=i}^{n-1} \theta(e_{k,k+1}, s_k) = J(s_i, s_n)$$

where $J(s_i, s_n)$ is the actual cost of the path from s_i to s_n as defined in (7.4.2), but not necessarily the optimal one. Since this is is true for any path, it is true for the optimal path. Hence for all $s_i \in S$,

$$\hat{h}(s_i, s_n) \leq h(s_i, s_n) \tag{7.5.2}$$

Thus, the A^* algorithm is admissible. By the triangle inequality, $\hat{h}(s_i, s_n) \leq \hat{h}(s_i, s_j) + \hat{h}(s_j, s_n)$. Noticing that (7.5.2) also holds for any j, $i \leq j \leq n$, since s_j is also reachable from s_0. Now we have $\hat{h}(s_i, s_j) \leq h(s_i, s_j)$. It follows that $\hat{h}(s_i, s_n) \leq h(s_i, s_j) + \hat{h}(s_j, s_n)$. That is, $\hat{h}(s_i, s_n) - \hat{h}(s_j, s_n) \leq h(s_i, s_j)$. Hence the A^* algorithm is consistent. Therefore, the A^* algorithm finds an optimal path from s_0 to s_g. ■

Theorem 7.5.1 is a further development to Theorem 1 in [89] by Passino and Antsaklis. They used $h(s) = inf\{\rho(s, s_g); s_g \in X_{af}\}$ with ρ being a metric and $\chi(s, s')$ as cost function satisfying $\rho(s, s') \leq \chi(s, s')$ and some other restriction, and proved that $h(s)$ is admissible and monotone. We define a semi-metric and further develop it to solve the optimization problem. It gives the existence of solutions to the optimization problem of DESs by using A^* algorithm. The search

algorithm utilizes the "principle of optimality" of dynamic programming [7] and the advantages of branch and bound algorithms [81]. We will give a comparison of A^* with the dynamic programming later in this chapter. Using this theorem, we implemented the A^* algorithm into the simulator in the software package which is developed in the next chapter.

Since we use A^* algorithm to do searching, the cost of search is the computational complexity of A^* algorithm. In a worst case analysis of A^*, we assume the expansion of a state as a basic operation. At most r states, where r is the number of reachable states, will be expanded at termination. If the heuristic function is only admissible (and not consistent), then for each state expanded every other state that is expanded by termination could also be expanded [76]. Therefore it is possible that A^* expanded $order(2^r)$ states. This is exponential in the number of reachable states. A way to solve this problem is to use a consistent heuristic function. If the heuristic function is consistent and also admissible, then each state is only expanded once so A^* runs in $order(r)$ steps [76] which is linear in the number of reachable states.

7.6 An Example of Application

As an application, we consider an example of the call processing of the packet layer protocol in the packet-switched data communication network. In this system, two packet layer protocol entities, for instance, transport end-points, call each other to implement the user services by exchanging packet protocol data units [31].

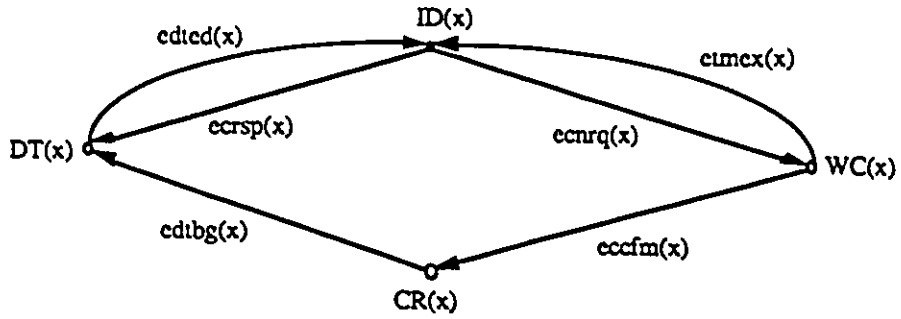


Figure 7.1: The diagram of the protocol.

The call processing can be described by temporal logic models that specify how events cause changes of states and which event can occur at a particular state, as shown in Figure 7.1.

Let the local variable x represent the states of the protocol entities. The system has four states: $ID(x)$ which means that there is no connection established (idle); $WC(x)$ which stands for waiting for a call connected packet; $CR(x)$ which means connection established and ready for data transfers; $DT(x)$ which stands for data transmission.

There are six events in the system. At state $ID(x)$, there are two events which can occur: $ecnrq(x)$, the connection request; and $ecrsp(x)$, the call response. At state $WC(x)$, there are two events which can occur: $eccfm(x)$, the connection confirmed; and $etmex(x)$, the call connected timer expires. At state $CR(x)$, one event can occur: $edtbg(x)$, the data transmission begins. At state $DT(x)$, one event can occur: $edted(x)$, the data transmission ends.

The set of formulas of the specifications for the system are given as follows.

Dynamics

$$\Box[ecnrq(x) \wedge ID(x) \Rightarrow WC(\bigcirc x)] \qquad (7.6.P1)$$

$$\Box[ecrsp(x) \wedge ID(x) \Rightarrow DT(\bigcirc x)] \qquad (7.6.P2)$$

$$\Box[eccfm(x) \wedge WC(x) \Rightarrow CR(\bigcirc x)] \qquad (7.6.P3)$$

$$\Box[etmex(x) \wedge WC(x) \Rightarrow ID(\bigcirc x)] \qquad (7.6.P4)$$

$$\Box[edtbg(x) \wedge CR(x) \Rightarrow DT(\bigcirc x)] \qquad (7.6.P5)$$

$$\Box[edted(x) \wedge DT(x) \Rightarrow ID(\bigcirc x)] \qquad (7.6.P6)$$

Initial condition

$$ID(x) \qquad (7.6.P7)$$

The simulation result of the uncontrolled system is shown in Figure 7.2 and its reachability graph is given in Figure 7.3.

Let $EQ(x, y)$ represent that x is identical to y. The required behavior of the two entities system is given as follows:

$$\Box[CR(x) \wedge \neg EQ(x, y) \Rightarrow \neg CR(y) \wedge \neg WC(y)] \qquad (7.6.CL1)$$

$$\Box[DT(x) \wedge \neg EQ(x, y) \Rightarrow \neg WC(y)] \qquad (7.6.CL2)$$

$$\Box[WC(x) \wedge \neg EQ(x, y) \Rightarrow \neg CR(y) \wedge \neg DT(y)] \qquad (7.6.CL3)$$

The state transitions of the uncontrolled system are:

```
The root:  s0=(ID,ID)
e1=(ecnrq 1)   --> s1=(WC,ID)
e3=(ecrsp 1)   --> s3=(DT,ID)
e2=(ecnrq 2)   --> s2=(ID,WC)
e4=(ecrsp 2)   --> s4=(ID,DT)

The root:  s1=(WC,ID)
e5=(eccfm 1)   --> s5=(CR,ID)
e6=(etmex 1)   --> s6=(ID,ID)=s0
e7=(ecnrq 2)=e2 --> s7=(WC,WC)
e8=(ecrsp 2)=e4 --> s8=(WC,DT)

The root:  s2=(ID,WC)
e10=(ecnrq 1)=e1 --> s10=(WC,WC)=s7
e9=(ecrsp 1)=e3 --> s9=(DT,WC)
e11=(eccfm 2)  --> s11=(ID,CR)
e12=(etmex 2)  --> s12=(ID,ID)=s0

The root:  s3=(DT,ID)
e13=(edted 1)  --> s13=(ID,ID)=s0
e15=(ecnrq 2)=e2 --> s15=(DT,WC)=s9
e14=(ecrsp 2)=e4 --> s14=(DT,DT)

The root:  s4=(ID,DT)
e17=(ecnrq 1)=e1 --> s17=(WC,DT)=s8
e16=(ecrsp 1)=e3 --> s16=(DT,DT)=s14
e18=(edted 2)  --> s18=(ID,ID)=s0

The root:  s5=(CR,ID)
e19=(edtbg 1)  --> s19=(DT,ID)=s3
e21=(ecnrq 2)=e2 --> s21=(CR,WC)
e20=(ecrsp 2)=e4 --> s20=(CR,DT)

The root:  s7=(WC,WC)
e22=(eccfm 1)=e5 --> s22=(CR,WC)=s21
e23=(etmex 1)=e6 --> s23=(ID,WC)=s2
e24=(eccfm 2)=e11 --> s24=(WC,CR)
e25=(etmex 2)=e12 --> s25=(WC,ID)=s1

The root:  s8=(WC,DT)
e27=(eccfm 1)=e5 --> s27=(CR,DT)=s20
e28=(etmex 1)=e6 --> s28=(ID,DT)=s4
e26=(edted 2)=e18 --> s26=(WC,ID)=s1

The root:  s9=(DT,WC)
e31=(edted 1)=e13 --> s31=(ID,WC)=s2
e29=(eccfm 2)=e11 --> s29=(DT,CR)
e30=(etmex 2)=e12 --> s30=(DT,ID)=s3

The root:  s11=(ID,CR)
e33=(ecnrq 1)=e1 --> s33=(WC,CR)=s24
e32=(ecrsp 1)=e3 --> s32=(DT,CR)=s29
e34=(edtbg 2)  --> s34=(ID,DT)=s4

The root:  s14=(DT,DT)
e35=(edted 1)=e13 --> s35=(ID,DT)=s4
e36=(edted 2)=e18 --> s36=(DT,ID)=s3

The root:  s20=(CR,DT)
e37=(edtbg 1)=e19 --> s37=(DT,DT)=s14
e38=(edted 2)=e18 --> s38=(CR,ID)=s5

The root:  s21=(CR,WC)
e41=(edtbg 1)=e19 --> s41=(DT,WC)=s9
e39=(eccfm 2)=e11 --> s39=(CR,CR)
e40=(etmex 2)=e12 --> s40=(CR,ID)=s5

The root:  s24=(WC,CR)
e43=(eccfm 1)=e5 --> s43=(CR,CR)=s39
e44=(etmex 1)=e6 --> s44=(ID,CR)=s11
e42=(edtbg 2)=e34 --> s42=(WC,DT)=s8

The root:  s29=(DT,CR)
e46=(edted 1)=e13 --> s46=(ID,CR)=s11
e45=(edtbg 2)=e34 --> s45=(DT,DT)=s14

The root:  s39=(CR,CR)
e47=(edtbg 1)=e19 --> s47=(DT,CR)=s29
e48=(edtbg 2)=e34 --> s48=(CR,DT)=s20
```

Figure 7.2: Simulation result of the uncontrolled system.

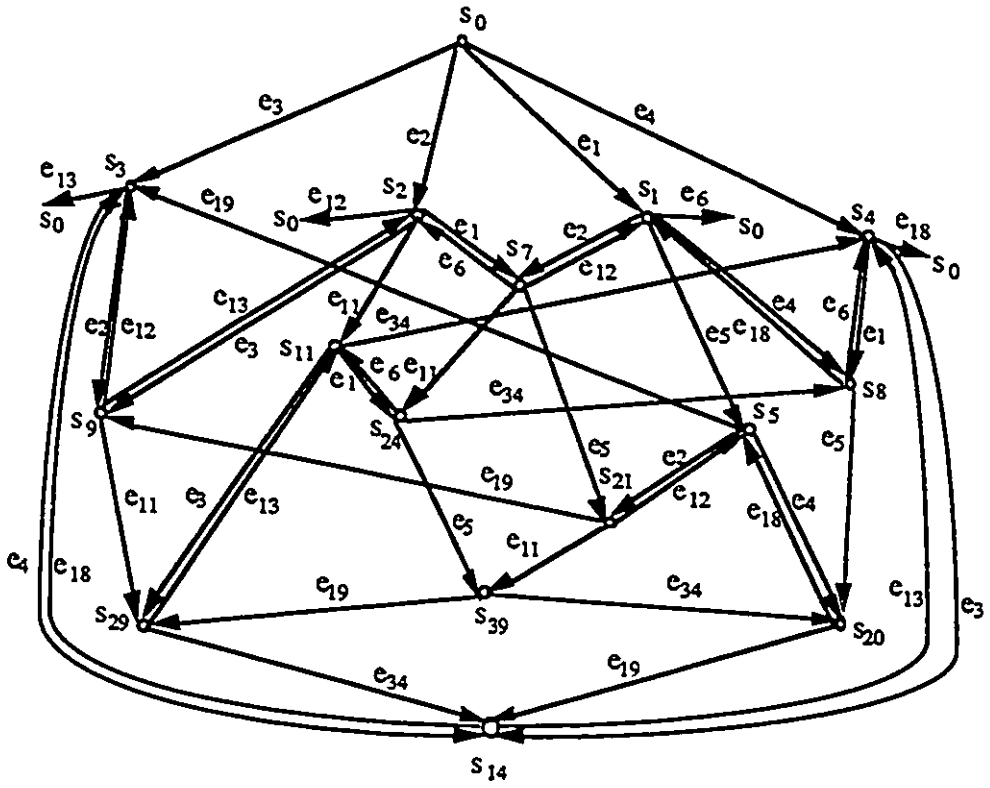


Figure 7.3: Reachability graph of the uncontrolled system.

where $\neg$ is the negation.

First of all, we have to design a controller using Procedure 6.4.1 such that the closed-loop system has the required behavior as specified above. Then we can find an optimal trajectory within such required behavior.

According to these requirements, the monitor in our software package which will be discussed in the next chapter, gives the warnings of those states that should not be reached and suggestions on those events that should be disabled. These warnings and suggestions are shown in Figure 7.4. Using Procedure 6.4.1 given in Chapter 6 and following the suggestions given by the monitor, we design the controller as follows.

We use local variable symbols p, q, and r to represent the data stored by the controller. q is assigned values 1 and 0 representing if an entity is in state CR or not; p and r are assigned non-negative integer values representing the number of entities which are in state WC or state DT. Let $ADD1(n)$ stand for that n is increased by 1 and $MIN1(n)$ for that n is decreased by 1. Then, the specifications of the controller are as follows.

Dynamics of the Controller

$$\Box[ecnrq(x) \wedge EQ(q,0) \Rightarrow ADD1(\bigcirc p) \wedge EQ(\bigcirc q,0)] \qquad (7.6.C1)$$

$$\Box[etmex(x) \wedge \neg EQ(p,0) \Rightarrow MIN1(\bigcirc p)] \qquad (7.6.C2)$$

$$\Box[eccfm(x) \wedge EQ(p,1) \wedge EQ(q,0) \Rightarrow EQ(\bigcirc p,0) \wedge EQ(\bigcirc q,1)] \qquad (7.6.C3)$$

According to the specifications of the required behavior

s8 should not been reached!
s9 should not been reached! So event e9 should be disabled at state s2 !
s15=(DT,WC)=s9 So event e15 should be disabled at state s2 !
s17=(WC,DT)=s8 So event e17 should be disabled at state s4 !
s21 should not been reached! So event e21 should be disabled at state s4 !
s22=(CR,WC)=s21 So event e22 should be disabled at state s7 !
s24 should not been reached! So event e24 should be disabled at state s7 !
s33=(WC,CR)=s24 So event e33 should be disabled at state s11 !
s41=(DT,WC)=s9 So event e41 should be disabled at state s11 !
s39 should not been reached! So event e39 should be disabled at state s21 !
s43=(CR,CR)=s39 So event e43 should be disabled at state s24 !
s42=(WC,DT)=s8 So event e42 should be disabled at state s24 !
s8 may not be a root!
s9 may not be a root!
s21 may not be a root!
s24 may not be a root!
s39 may not be a root!

Then the set of states which should not be reached are:

s8=(WC,DT) s9=(DT,WC) s21=(CR,WC) s24=(WC,CR) s39=(CR,CR)

Figure 7.4: Warnings and suggestions given by the monitor.

$$\Box[ecrsp(x) \wedge EQ(q,1) \wedge EQ(r,0) \Rightarrow EQ(\bigcirc q,1) \wedge EQ(\bigcirc r,1)] \qquad (7.6.C4)$$

$$\Box[edtbg(x) \wedge EQ(q,1) \wedge EQ(r,1) \Rightarrow EQ(\bigcirc q,0) \wedge EQ(\bigcirc r,2)] \qquad (7.6.C5)$$

$$\Box[edtcd(x) \wedge \neg EQ(r,0) \Rightarrow MIN1(\bigcirc r)] \qquad (7.6.C6)$$

Initial Condition of the Controller

$$EQ(p,0) \wedge EQ(q,0) \wedge EQ(r,0) \qquad (7.6.C7)$$

The controller designed in (7.6.C1)-(7.6.C6) ensures that the closed-loop system has the required behavior. The simulation result of the closed-loop system is given in Figure 7.5 and its reachability graph is shown in Figure 7.6. It shows that there are two paths from state S_0 to state S_7 and from state S_0 to state S_{37}, respectively.

Now we can do the optimization. In this example, the costs of maintaining the system at states are assumed as 0 and the costs of events are given in Table 7.1. We want to find the path, starting from state S_0 and ending at state S_{37}, to have the minimal cost; and we also want to generate the corresponding sequence of events to drive the system along the path.

In order to use the A^* algorithm to solve the optimization problem, some heuristic index will be helpful. For instance, the preference indexes of the two-entity system are given in Table 7.2.

```
Now the state transitions of the controlled system are:

The root:  S0 =(ID,ID,0,0,0)
e1=(ecnrq 1)   --->  S1 =(WC,ID,1,0,0)
e2=(ecnrq 2)   --->  S2 =(ID,WC,1,0,0)

The root:  S1 =(WC,ID,1,0,0)
e5=(eccfm 1)   --->  S5 =(CR,ID,0,1,0)
e7=(ecnrq 2)=e2 --->  S7 =(WC,WC,2,0,0)

The root:  S2 =(ID,WC,1,0,0)
e10=(ecnrq 1)=e1 --->  S10 =(WC,WC,2,0,0)=S7
e11=(eccfm 2)   --->  S11 =(ID,CR,0,1,0)

The root:  S5 =(CR,ID,0,1,0)
e20=(ecrsp 2)=e4 --->  S20 =(CR,DT,0,1,1)

The root:  S7 =(WC,WC,2,0,0)
e23=(etmex 1)=e6 --->  S23 =(ID,WC,1,0,0)=S2
e25=(etmex 2)=e12 --->  S25 =(WC,ID,1,0,0)=S1

The root:  S11 =(ID,CR,0,1,0)
e32=(ecrsp 1)=e3 --->  S32 =(DT,CR,0,1,1)

The root:  S20 =(CR,DT,0,1,1)
e37=(edtbg 1)=e19 --->  S37 =(DT,DT,0,0,2)

The root:  S32 =(DT,CR,0,1,1)
e45=(edtbg 2)=e34 --->  S47 =(DT,DT,0,0,2)=S37

The root:  S37 =(DT,DT,0,0,2)
e35=(edted 1)=e13 --->  S48 =(ID,DT,0,0,1)
e36=(edted 2)=e18 --->  S49 =(DT,ID,0,0,1)

The root:  S48 =(ID,DT,0,0,1)
e18=(edted 2)=e18 --->  S50 =(ID,ID,0,0,0)=S0

The root:  S49 =(DT,ID,0,0,1)
e13=(edted 1)=e13 --->  S51 =(ID,ID,0,0,0)=S0

The reachability Set:
 S0 =(ID,ID,0,0,0) S1 =(WC,ID,1,0,0) S2 =(ID,WC,1,0,0)
S5 =(CR,ID,0,1,0) S7 =(WC,WC,2,0,0) S11 =(ID,CR,0,1,0)
S20 =(CR,DT,0,1,1) S32 =(DT,CR,0,1,1) S37 =(DT,DT,0,0,2)
S48 =(ID,DT,0,0,1) S49 =(DT,ID,0,0,1)
```

Figure 7.5: The simulation result of the controlled system.

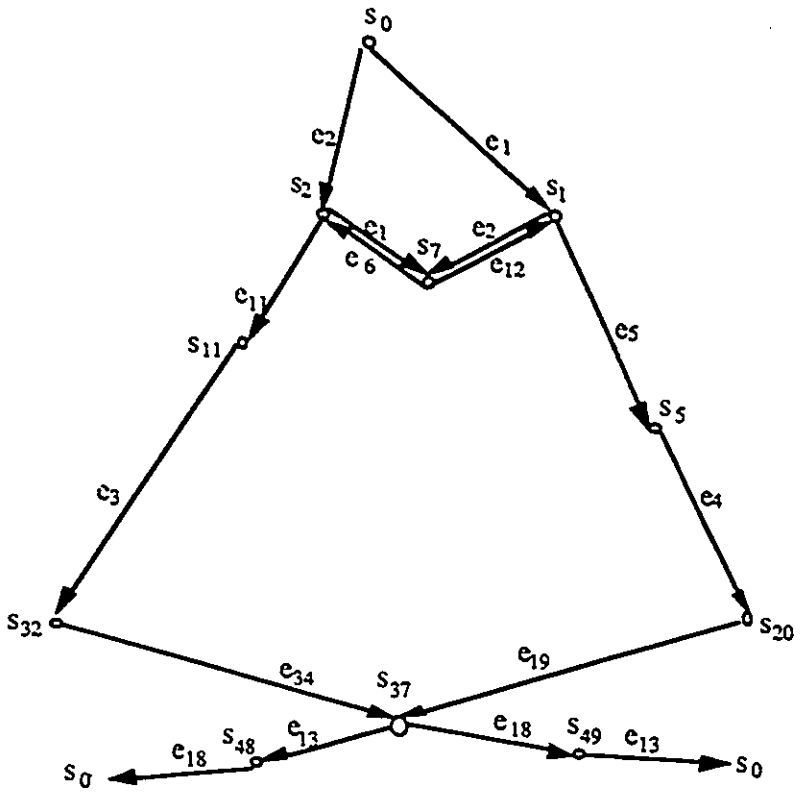


Figure 7.6: The reachability graph of the controlled system.

The preference index is a kind of trouble index of the states. We use the preference indexes as heuristic indexes, that is, we chose heuristic function of the example as the preference index of the states. The numbers in Table 7.2 have a background from the example given in Figure 7.8. The smaller number means the traffic load is less in that state; therefore it is more preferred.

Let the heuristic function

$$\hat{h}(s_i, s_j) \stackrel{\text{def}}{=} |index(s_i) - index(s_j)|$$

Then $\hat{h}(s_i, s_j)$ is a semi-metric function and there holds

$$\hat{h}(s_i, s_j) = |\sum_{k=i}^{j-1}[h(s_k) - h(s_{k+1})]|$$
$$\leq \sum_{k=i}^{j-1} |h(s_k) - h(s_{k+1})| = \sum_{k=i}^{j-1} \theta(e_{k,k+1}, s_k) = \theta(e_{ij}, s_i)$$

By using Theorem 7.5.1 and A^* algorithm, our simulator of the software package can find the optimal paths starting from the given initial state ending at the given final state. Figure 7.7 gives the optimal trajectory and corresponding optimal sequence of events for the initial state S_0 and the final state S_{37}.

As we can see from Figures 7.6 and 7.7, the optimization procedure goes one step further than the controller synthesis given in Chapter 6. In Figure 7.5, the optimal path and the corresponding sequence of events are found using Theorem 7.5.1 of this chapter, while the required behavior is realized by the controller synthesis procedure in Chapter 6. According to the optimal cost function index, Theorem 7.5.1 specifies a trajectory within the required behavior given by Proce-

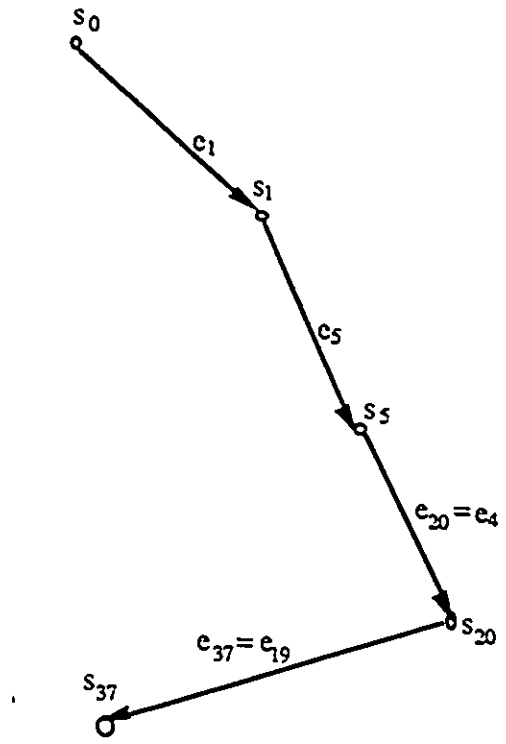


Figure 7.7: The optimal path and corresponding events

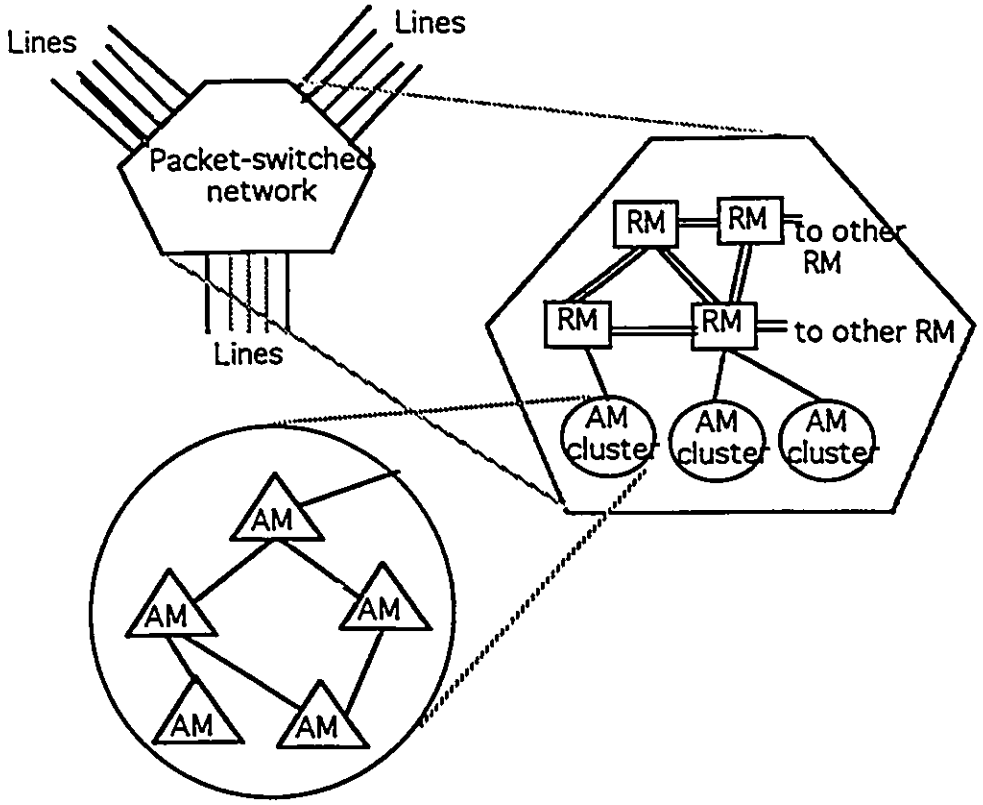


Figure 7.8: Partial configuration of a packet-switched network.

dure 6.4.1.

Informally speaking, the safety assertion says that bad things should not happen while the liveness says that good things will eventually happen. In this sense, we can say that Procedure 6.4.1 guarantees the safety assertions and the optimization procedure realizes the liveness assertions.

This is an example of the minimum event cost control problem of DESs. By

using the heuristic function, the A^* algorithm gives a solution to the optimization procedure; and it is dependent on the selection of the heuristic function. In this example we use the preference index of states as the heuristic function. This has a practical signification in the real world problem. For instance, a packet-switched data network composing of access modules (AMs) and resource modules (RMs), as shown in Figure 7.8, has an optimizer to optimize the data routines [22]. The optimizer should have the preference index of each AM or RM and the costs of operation to perform the optimization. The preference indexes can be used to form a heuristic function as we did in the example.

7.7 Conclusion

In this chapter, we have proposed an enhanced temporal logic model with the cost function in a semi-metric space and formulated the optimization problem of DESs. To solve the optimization problem, we have shown that there exists an admissible and consistent A^* algorithm that generates a sequence of events driving the system from the initial state to the final state with the least cost if the heuristic function is a semi-metric function and is a low bound of the cost function. As an application, we have demonstrated an example of the packet-switched communication networks to illustrate our results.

A comparison of the A^* algorithm with the dynamic programming (DP) is given as follows. For the given initial state s_0 and the final state s_f, we assume that $f(s_0, s_f) = g(s_0, s) + h(s, s_f)$ is the cost of a path from s_0 to s_f through s such

that $g(s_0, s)$ is the cost of optimal path from s_0 to s and $h(s, s_f)$ is the heuristic function which is an estimate of the optimal cost from s to s_f. In the case of DP, the principle of optimality says that $f(s_0, s_f)$ is optimal implies $h(s, s_f)$ is optimal. The optimal condition of DP is a necessary condition but not sufficient. Our Theorem 7.5.1 is a sufficient condition. In fact, the A^* algorithm was developed based on the optimal principle of DP for graph searching. When $h = 0$, the A^* algorithm is equivalent to the uniform algorithm, which is a combination of branch and bound algorithms; and therefore it combines the principle of DP with the advantages of the uniform search techniques that can eliminate certain candidate solutions from consideration by using the information from the system. Hence, the A^* algorithm has the advantages of the uniform search techniques over DP. It is good for the graph searching of large systems.

Among related works, our work here is close to that of Passino and Antsaklis [89]. They also use the A^* algorithm to solve the optimal control problem of DESs. The main differences between their approach and ours are that ours is special for DESs in a temporal logic framework using the temporal logic model while theirs is general for any abstract logical DES and that our emphasis is on the optimization procedure while theirs is the heuristic function. In addition, the problem formulations are slightly different. Furthermore, the semi-metric space we defined is more general than the metric space they used since that the cost function θ_{ij} may be different from θ_{ji} in the real world problems. To illustrate this point, we use $f(s) = g(s_0, s) + h(s, s_f)$ as above mentioned. In order to achieve $f(s)$ to be the optimal cost quickly, we have to make $h(s, s_f)$ as close as possible to the optimal

cost which is the sum of $\theta(e_{ij}, s_i)$, from s to s_f for all s. In the case of $\theta(e_{12}, s_1)$ is not equal to $\theta(e_{21}, s_2)$ for $h(s_1, s_2) \leq \theta(e_{12}, s_1)$ and $h(s_2, s_1) \leq \theta(e_{21}, s_2)$, if h is symmetric, $h(s_1, s_2) = h(s_2, s_1)$ can only be chosen to close to the smaller one of $\theta(e_{12}, s_1)$ and $\theta(e_{21}, s_2)$ but can not close to the larger one, then the optimization will be slow. However, if h is non-symmetric, then we can make $h(s_1, s_2)$ and $h(s_2, s_1)$ close to $\theta(e_{12}, s_1)$ and $\theta(e_{21}, s_2)$, respectively. A simple case of $\theta(e_{12}, s_1)$ is not equal to $\theta(e_{21}, s_2)$ is a robot moving blocks. The cost of picking up a block is larger than the cost of putting it down because of gravity.

The suggestions for further research include:

1. The uniqueness of solutions to the optimization problem of DESs.
2. A set of final states instead of only one final state is also an important case of optimization problems.
3. The computational complexity of the A^* algorithm.
4. How to select a heuristic function.

event e	e_1	e_2	e_3	e_4	e_5	e_6	e_{11}	e_{12}	e_{13}	e_{18}	e_{19}	e_{34}
cost θ	1	2	3	4	1	2	3	4	5	6	1	2

Table 7.1: The cost function.

state S_i	S_0	S_1	S_2	S_5	S_7	S_{11}	S_{20}	S_{32}	S_{37}	S_{48}	S_{49}
$index(s_i)$	0	1	2	2	2	3	3	4	4	4	5

Table 7.2: The index function.

Chapter 8

A Software for Reasoning about and Simulating DESs

In this chapter a software package aimed towards reasoning about and simulating DESs is designed and implemented. The software is to be used to make logic reasoning and simulate the dynamical behavior of a DES. This involves the implementation of three types of actions: the simulation action, the monitor action, and the synthesis action of the software. The simulation action is the action of the software as a simulator to simulate the given system for the modeling and measurements. The monitor action is the action of the software as a monitor to monitor the given system for the controller design. The synthesis action is the action of the software as a coordinator to synthesize a number of subsystems or to synthesize a controller for the system configuration.

8.1 Introduction

As DESs are finding more and more applications to the analysis and design of complex manufacturing, communication, and computer systems, a simulation tool becomes more and more important to build an application. Therefore, a software package is needed for simulating a broad range of DESs. For the temporal logic framework presented in previous chapters, it is useful to design and implement a simulation environment where the temporal relations of the system states can be reasoned and the dynamic behavior such as reachability properties can be demonstrated for a wide range of DESs. On the other hand, these features of the simulation tool can be implemented immediately by models which can give the users the forbidden states and therefore recommendations for design.

There has been a great quantity of research into discrete event simulations [98]. Recently, the object-oriented approach is receiving an increasing consensus owing to reusability and information hiding issues. As mentioned in Chapter 1, DES simulation languages have played an important role in the development of the object-oriented programming languages. For discrete event system computation and simulation, O'Young and Wonham [88], using the object-oriented methodology, created a class of DES objects with the generic class name DES to encapsulate the transition structure of finite state machines at various levels of architectural abstraction in the modeling of a large-scale DES. For developing software for DESs, Baldassari and Bruno presented in [5] PROTOB, an object-oriented language and methodology based on PROT nets which integrate extended data-flows and Petri nets into an object-oriented formalism. Little work has been done for the design

and implementation of a software package for logic evaluation, reasoning, and simulating DESs.

In this chapter, a software package is developed for proving the correctness of the models and theories built in a temporal logic framework established in the previous chapters. Our motivation is to provide tools and components necessary to produce efficiently a broad range of DESs for their temporal pattern and dynamical behavior. With this motivation, the software is designed as a tool for temporal reasoning to perform the logic evaluation and inference. It can also carry out the reachability analysis in Chapter 5, the design and synthesis of a controller of a DES in Chapter 6, and the optimization of finding the minimum-cost path in Chapter 7. In addition, the software is designed in an object-oriented approach which has the advantages of reusability, inheritance, dynamic binding, etc. The implementation language is Objective-C.

This chapter is organized as follows. In the next section, the requirements of the software package are described according to its different usages. In Section 8.3, we present the reason why Objective-C is chosen as the programming environment for the software package; and we also highlight the advantages of Objective-C and its module structure. In Section 8.4, the software architecture is constructed in an object-oriented paradigm; and each class is described. In Section 8.5, the implementation of the software is discussed and demonstrated. Finally, a discussion is given in Section 8.6.

8.2 Requirements of the Software Package

In this section we outline the requirements for the software package and then describe its tools corresponding to each requirement.

To prove the correctness of the models and theories obtained in the previous chapters, the following requirements are proposed for the software package:

- The software should be able to evaluate the truth of a temporal logic statement expressed by predicates combined with logic connectives and temporal operators based on the given facts
- The software should be able to deduce conclusions from the given rules, facts and initial conditions
- The software should be able to simulate the dynamic behavior for the given DES
  - It should perform the reachability analysis and find the enabled event set for each state
  - It should give suggestions for designing a controller according to the given description of the required behavior of the system and synthesize a controller with the system into the closed-loop system
  - It should produce an optimal path and the corresponding sequence of events for the given costs and heuristic index

According to the above requirements, the software package is composed of three separated but strongly interconnected tools described as follows.

1. *Evaluator* – Given a group of facts and statements, it evaluates the statements to see if they are true or false according to the given facts. A fact is given by that a predicate is true or false at some time period. For example, the fact that `time 1 5 friend tom,cathy 1` represents predicate $friend(tom, cathy)$ is true from time 1 to time 5. The statements are expressed by predicates combined with logic connectives and temporal operators as well as times. `time 2 6 (and (friend tom,cathy) (next (friend tom,smith)))` is an example of a statement.

2. *Logician* – Given a group of facts, a group of rules, and the initial conditions, it deduces conclusions. An example of a group of rules is `IF ((loaded 1)(enfir 1)) THEN (loaded (next 1))` and `IF ((loaded 2)(efire 2)) THEN (noise (next 2))`. Also `(enfire 1), (efire 2)` is an example of a group of facts (events or conditions) while an example of the initial condition is `(loaded 1)`. Then it will deduce that `(noise 3)`.

3. *Simulator* – It simulates the dynamic behavior for the given DES

   - For a system given in temporal logic models, it performs the reachability analysis by giving the reachability set, the enabled event set for each reachable state, and the reachability graph routines as shown in Chapter 5

   - For the given description of the required behavior of the system, it gives a list of warnings on which states should not be reached and a list of suggestions on which events should be disabled to guide designing a controller to eliminate the undesired behavior, and then synthesizes

the controller with the system into the closed-loop system as shown in Chapter 6

- For the given costs of events and heuristic indexes of states, it produces an optimal path and corresponding sequence of events which drive the system along the optimal path from the given initial state to the given final state with the minimum of the cost as shown in Chapter 7.

8.3 Programming Environment

The complexity of simulating DESs requires adequate programming environment capable of capturing a reality consisting of elements which operate in parallel and asynchronously. Elements interact with each other through synchronization and information passing mechanisms. They respond to events with timing constraints according to their states, which depends on time and on their interactions with other components. According to the requirements in the previous section, an object-oriented programming environment was chosen such that the knowledge may be formalized in building blocks and reused. Previously defined blocks that have been validated may be composed to specify new systems. Therefore, in this section we briefly introduce Objective-C, an object-oriented language, as our programming environment.

Objective-C is a superset of C. It has the advantages that object-oriented design principles can be used as needed and that the program can be written in C when efficiency is required.

Having such advantages, Objective-C programming differs most dramatically from traditional programming which separated data from the operators that manipulate the data. It uses encapsulation to encapsulate the data with operators into a unit called *object* which may be thought of as physical or conceptual entity in the problem domain [92]. In fact, an object can most easily be visualized as an abstraction of a physical device. For example, events and states of our temporal logic model are instances of objects.

To manipulate the data inside an object, a *message* is sent to the object; and it invokes a *method* which is a list of detailed instructions that define how an object responds to a particular message. Thus, every message has a corresponding method. Each object is instantiated from a *class* which is a template for creating objects. A class includes in its description a name for the kind of object, a list of instance variables, and a list of messages with corresponding methods to which an object of the class can respond. As a superset of C, Objective-C contains global variables and C-functions in its module structures [92]. The function definitions and global variables can be accessed directly by methods in the same or other class. Methods define the behavior of an object. Each method has a unique name called its selector; and it may use other methods of either the same or other class.

An Objective-C interface file defines the instance variables for all objects of the class as well as the message selectors for all methods of the class. In addition, it specifies a parent class. This proposes another concept in the object-oriented programming, *inheritance* – another advantage of using Objective-C. The Objective-

C Stepstones ICpak101 library [92] is used to implement the software package. From bottom up, it is the *IS-A* hierarchy. For example, *IdArray* is a *Array* which is an *Object*. On the other hand, from top down, it is a structure of the problem decomposition with generic classes at the roots and very specialized classes at the leaves. For example, *Array* class is decomposed into more specialized classes *IdArray* and *IntArray*.

All instance variables of ancestor classes are inherited by and can not be redefined within a subclass. These instance variables are directly accessible within the implementation file but inaccessible outside of the implementation file. All methods of ancestor classes are inherited but can be redefined. Often, super-classes are constructed abstractly such that instance variables and message selectors are grouped to be used by a cluster of subclasses. Many of the methods in such a superclass are not defined; and their definitions are given in subclasses. This leads to one of the advantages of using Objective-C (or any other object-oriented language) known within the object-oriented paradigm as *dynamic binding*.

8.4 The General Architecture of the Software

The software package we are designing is an object-oriented application. The fundamental component of the application is the object. Thus the first step in the design is to define the objects. Objects are characterized by their attributes and the messages to which they respond; and this leads to the second step —defining the attributes and messages for each object. The objects are related to each other

by their dependences; therefore the third step is to build a structure that accurately describes these relationships. The implementation of these three steps is to define and develop a hierarchy of classes.

Figure 8.1 diagrams the general architecture of the software. (The data flow in the architecture is given in Figure 8.2 later.) It shows that the building blocks of the software are objects. An object needs not to be an abstraction of a real physical device, but in the architecture of the software designed here, the object-oriented approach yields a natural mapping of abstract objects to physical devices such as *Fact, Rule, Statement, Reasoner, Operator, Event, State, Process, Controller, Monitor, ClosedLoop, Optimizer*, of twelve classes. These classes are generated by the user interface *dedsMain*, the main program which will be introduced below.

8.4.1 The main program, dedsMain

The main program, *dedsMain.m*, manages the software. As a user interface, it links the user to one of the three tools as the user needs. It provides a process control to start and to stop a simulation process. Generally, it carries out the following tasks:

- Create *aFact* and *aStatement* as instances of the **Fact** and **Statement** classes to input the facts and statements from the text files called *FactFile* and *StatementFile*, respectively. Then it creates *aReasoner*, the only instance of the **Reasoner** class to evaluate the statements.

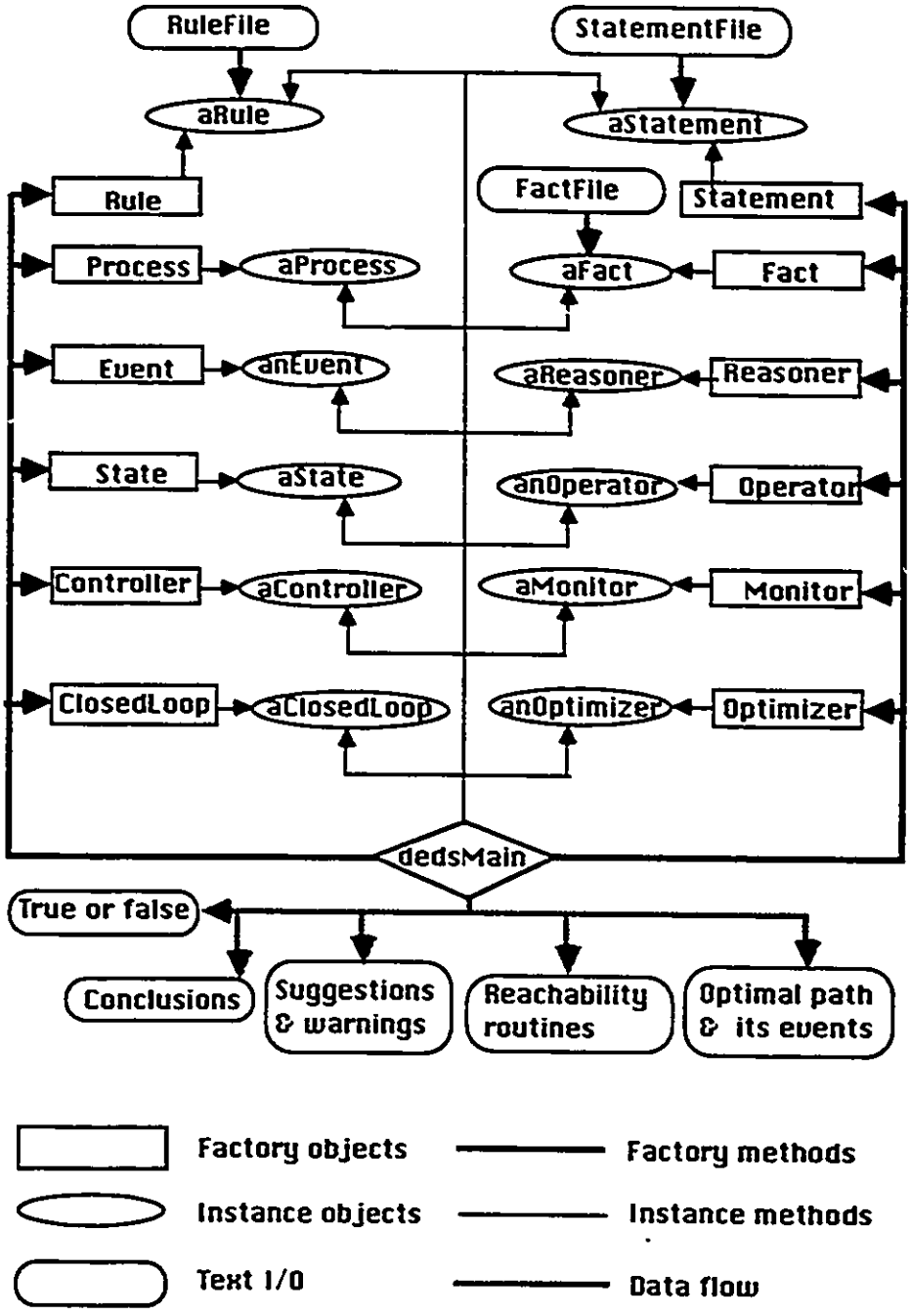


Figure 8.1: The general architecture of the software.

- Create *aFact* and *aRule* as instances of the **Fact** and **Rule** classes to input facts and rules from the text files called *FactFile* and *RuleFile*, respectively. Then it creates *aForw*, the instance of the **Forw** class which is a subclass of the **Rule** class, to infer the conclusions.

- Create instances of the **Process, Event, State, Monitor, Controller, ClosedLoop** classes as the user requires to perform the simulation of the dynamic behavior of the DES. If necessary, it then creates *anOptimizer*, the only instance of the **Optimizer** class to find the optimal path and the corresponding sequence of events.

It displays on screen the outcomes of the simulation: the truth values of the statements; the conclusions of the logic reasoning; the reachability set, the enabled event set, the lists of warnings and suggestions, the reachability graph routines, the optimal paths and corresponding sequences of events.

It generates objects and links all classes in the software. In the sequel of this section, we will introduce each of these classes.

8.4.2 The Fact Class

The **Fact** class is used to implement the database of facts in the simulation. It reads in the data of facts and allocates in arrays for facts by using the **IdArray** class which is in the Objective-C library. The facts are stored in the fact database. For evaluating a statement, facts are matched against the statement to see if the statement is true or not. For reasoning of logic conclusion, the facts are used to

update the conclusions of the rules in the rule base implemented by the **Rule** class which will be introduced below.

8.4.3 The Rule Class

The **Rule** class is used to implement the database of rules in the simulation; and these rules are used to infer the conclusions by the given facts in the fact base as mention above. It reads in the data of rules and add the rules in the rule database **RuleDataBase** to be used for inferring. It has the methods to find the given rules and to delete them when necessary.

The **Rule** class has a subclass, the **Forw** class. It inherits the instance variables and methods of the **Rule** class. The **Forw** class has a method of forward chaining which reasons from the facts in the fact database and from the rules in the rule database and then produces the conclusions.

8.4.4 The Statement Class

The **Statement** class is used to implement the database of statements in the simulation. Similarly to the **Fact** class, it reads in the statements from the text file and allocates in arrays for them by using the **IdArray** class. The statements are added into the statement database and they are only used to make the logical evaluation.

8.4.5 The Reasoner Class

The main function of the **Reasoner** class is to evaluate the truth of the logical expressions. It first creates and initializes an`Operator`, the instance of the **Operator** class which has three subclasses, **And, Not, Or**; and then evaluate the statement. When it encounters the time points or temporal operators such as *next*, it checks the times in the fact database for each predicate. When it meets the logical operators: *and, not, or*, it uses a dynamical binding to call the descendants of the **Operator** class which is described below.

8.4.6 The Operator Class

The **Operator** class is a superclass and it has three subclasses: **And, Not, Or**. It defines the common methods for all three descendant classes and leaves the key reasoning method to its subclasses for the particular operations. In fact, it is called by the **Reasoner** class when the reasoning encounters with any of its descendants: **And, Not, Or**; and then, it goes directly to the descendant through the dynamical binding.

8.4.7 The Monitor Class

The **Monitor** class is used to monitor the behavior of the system. It first checks the results from reasoning and recognizes the events and the processes of the system. Then it sends the signals to the **Event** class and the **Process** class to generate their instance objects.

The **Monitor** class takes in the specifications of the required system behavior. Following the specifications, it checks the states of the system plant; it determines which state should not be reached (*e.g.* deadlock) and finds out the events that should not be enabled. Then it lists suggestions for the states which should not be reached and which event should be disabled and sends them to the controller as reference information.

8.4.8 The Event and Process Classes

The **Event** class deals with the discrete events occurring in the system and implements the event set and the enabled event sets of states. The **Event** class has the methods to check if an event is equal to a given event, and to print it out, and so on.

The **Process** class is used to implement states of the systems for occurred events. A process is a component of the state vector of the system. Processes are added into the database of processes to be used as a component of the state vector.

8.4.9 The State, Controller, ClosedLoop Class

The **State, Controller, ClosedLoop** classes are used to implement the plant, the controller, and the closed-loop of the system. The three classes have similar structures.

The states are generated by the **Process** class. The enabled event set can be found for a given current state. For each event in the enabled event set, the next state of the current state will then be generated. The dynamical behavior of the three systems: plant, controller, and the closed-loop can be shown by the reachability set and the reachability graph of the systems. The differences of the **Controller, ClosedLoop** classes from the **State** class is described below.

The **Controller** class is used to implement a controller of the system according to the reference information that comes from the monitor. It takes in the specifications of the controller design. Using the suggestions provided by the monitor, the controller can be constructed efficiently.

The **ClosedLoop** class includes the methods that allow it to synthesize the controller and the plant. Each of its states contains the state of the plant and the control provided by the controller. So the dimension of the closed-loop system is the sum of the dimension of the plant and that of the controller.

8.4.10 The Optimizer Class

The **Optimizer** class is used to implement the A^* search algorithm: to find a minimum-cost path using an evaluation function which is the sum of the cost function and the heuristic function to make the search efficient. It finds the optimal path which starts from the given initial state and ends at the given final

state. It has methods to select the best from a set and to redirect the search by using the heuristics of the evaluation function.

8.5 Implementation of the Software

In this section, we briefly discuss the implementation of the software by a short introduction to the implementation of dedsMain and each class outlined in the previous section.

8.5.1 The main program, dedsMain

The implementation of the most important tasks carried out by the *dedsMain* in the software is shown in Figure 8.2 and the details of the implementation is presented below:

1. It creates instances (1) and (2) to implement the databases of facts and statements, respectively

```
(1):  aFact=[Fact new: numFact];
(2):  aStatement=[Statement new: numState];
```

Then it creates the instance `aReasoner`

```
(3):  aReasoner=[Reasoner new];
```

At this point, it goes into the **Reasoner** class to create the instances of the **Operator** classes and its subclasses **And, Not, Or** to evaluate the truth of the

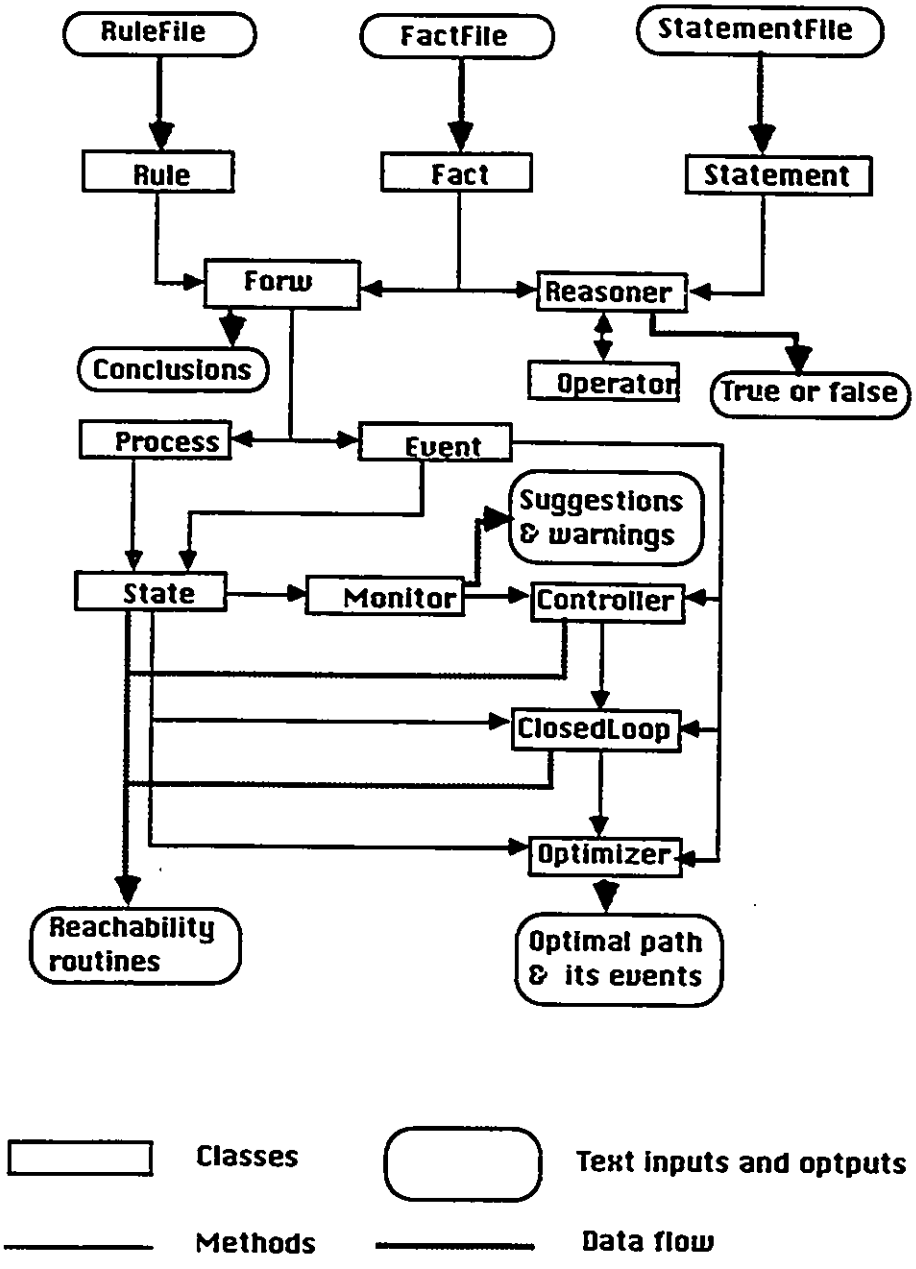


Figure 8.2: Implementation of dedsMain.

statements.

2. It creates instances (4) and (5) to input facts and rules

```
(4): aRule=[[Rule create] read_rule:fp];
(5): aFact=[[Fact create] read_fact:fp];
```

Then it creates the instance (6) to infer the conclusions from the facts and by firing rules to update the base of the conclusions.

```
(6): aForw=[[Forw create] takev: RuleDataBase];
```

where RuleDataBase is a global variable used to implement the database of the rules.

3. It creates the instances (7)-(10) to simulate a DES

```
(7): anEvent=[[Event create] read_event:sp];
(8): aProcess=[[Process create] read_process:sp];
(9): aState=[[State create] readFirstState:pp];
(10): aMonitor=[Monitor new];
```

The instances of the **Controller** and **ClosedLoop** can be created in a similar way to that of the **State** class. As the user requires, it creates the instance (11)

```
(11): anOptimizer=[Optimizer create];
```

to find an optimal trajectory and its sequence of events.

As a user interface, it prompts the requests for the user to make a choice to link to one of the three tools of the software or one of the functions in a tool. The following is an example of some requests.

```
Would you like to use the logic evaluator?(y/n)
Would you like to get conclusions and then stop?(y/n)
Would you like to make optimization? (y/n n --> end)
```

8.5.2 The Fact Class

The Fact class is implemented in the files: `Fact.h` and `Fact.m`. To be used for the logic evaluation, it is instantiated as in (1) and allocated by sending message `allocate:` to the object aFact. The source code of the method is given below.

```
-allocate:(unsigned)aNum
{ id row;
  int exPointer = 0;
  for(exPointer=0;exPointer<numFact;exPointer++)
    { row=[IdArray new:aNum];
      [self at:exPointer put:row];
    }
  return self;
}
```

where the factory message `new:` is sent to the IdArray class which is a class in Objective-C library and it creates an instance `row` to allocate in arrays for facts.

For inferring the logic conclusions, the **Fact** class is instantiated as in (5) to be used for updating the database of the conclusions of the rules implemented by the **Rule** class which is introduced below.

8.5.3 The Rule Class

The **Rule** class is implemented in the files: `Rule.h` and `Rule.m`. It is instantiated as in (4). The **Rule** class receives the factory message `create` and creates its instance object: `aRule`. Then `aRule` receives the instance message: `read_rule` and reads the rules from the data source `fp`. To find the rule for the given rule name, the message `find_rule:` is sent to an instance of the **Rule** class, for example, `next` in the following source code of the method `find_rule:`.

```
-find_rule:(char*)s
{ if(!strcmp(s,name))
     return self;
  else
     if(next==NULL)
        return NULL;
     else
        return [next find_rule:s];
}
```

The **Forw** class, the subclass of the **Rule** class, is implemented in the files: `Forw.h` and `Forw.m`. It is instantiated as in (6). To make the forward chaining,

the message forward_chain is sent to the instance aForw to update the database of the conclusions of the rules.

8.5.4 The Statement Class

The **Statement** class is implemented in the files: `Statement.h` and `Statement.m`. It is instantiated as in (2). The statements are added into the database of statements by sending the message `addAt:` to the instance `aStatement`. The source code of the method `addAt:` is shown as follows:

```
-addAt:(int)aNum
{ char aString[51];
  int stateCol;
  stateCol = 0;
  for(stateCol=0;stateCol<lenState;stateCol++)
    { scanf("%s",aString);
      [[self at:aNum] at:stateCol put:[String str:aString]];
    }
}
```

where **String** is a class in Objective-C library.

8.5.5 The Reasoner Class

The **Reasoner** class is implemented in the files: `Reasoner.h` and `Reasoner.m`. It is instantiated as in (3). To evaluate an logic expression, it sends the message

new to the **Operator** class to create the instance:

```
(12): anOperator=[Operator new];
```

After that, it initializes the logic operators and then it evaluate the statement by the message `evaluate`. In the instance method `evaluate`, it receives the messages: `reasonIn: And`, `reasonIn: Not`, `reasonIn: Or`, when it encounters the logical operators **And, Not, Or**, respectively. All these messages go to its method: `reasonIn: Operator` which does the dynamical binding. At the compile time, the compiler does not know which logical operator it is, and it couples messages to methods at run time. At run time, it checks if that is `And class`, or `Not class`, or `Or class`. The Objective-C run time system couples the message `reasonIn:` to the appropriate method (either in class **And**, or **Not**, or **Or**) dynamically. The benefit of dynamical binding is realized here because the reasoning methods are different in the three subclasses of the **Operator** class. Here, the user is unburdened from having to test which classes the receiving object is in order to determine the precise action to take. The dynamical binding takes care of this. This is one of the cornerstones of object-oriented programming.

8.5.6 The Operator Class

The **Operator** class is implemented in the files: `Operator.h` and `Operator.m`. It is a superclass and is the top of a hierarchy of three subclasses **And, Not, Or.** The protocol in the superclass is shared by subclasses. The hierarchy is shown in Figure 8.3.

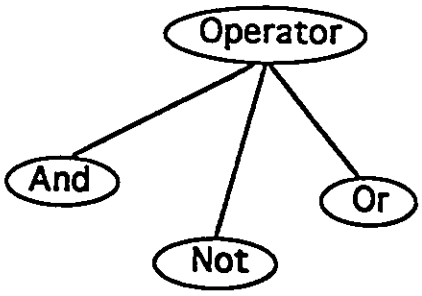


Figure 8.3: The hierarchy of the logical operators.

In the **Operator** class, the key methods `reasoning` and `reasoningFrom:to:` are implemented as `subclassResponsibility`. The details of these methods are implemented in the three subclasses. They are different and dependent on the properties of the logical operators.

8.5.7 The Monitor Class

The **Monitor** class is implemented in the files: `Monitor.h` and `Monitor.m`. It is instantiated as in (10). As a monitor, it checks if a predicate is an event or a process. The `check:` method is defined as follows:

```
-(int)check:(id) pred
{ char *t;
  id pd = pred;
  t=[[pd car] cars];
  if(strchr(t,'e'))
     return EVENT;
```

```
  else
     return PROCESS;
}
```

The **Monitor** class follows the specifications of the required behavior of the system, and uses the `checkBad:` method to check whether a state is bad or not. If the state does not satisfy the specifications, then it is a bad state. A warning is given by the message: `printf(" %s should not been reached!",[bad name])`. Meanwhile, it finds the root state and the event which leads to the current state. Then it sends suggestions by the message: `printf(" So, event %s may not occur at state %s !", [et name],[st name])` or `printf(" So event %s should be disabled at state %s !",[et name],[rt name])` and the message: `printf(" ! %s may not be a root!",[bad name])`.

8.5.8 The Event and Process Classes

The **Event** class is implemented in the files: `Event.h` and `Event.m`. It is instantiated as in (7). To check if `anEvent` equals to a given event `ep`, the message `eventEqual:` is sent to `anEvent`; and this method is presented below.

```
-(int)eventEqual:(id)ep
{
  return ([value equal:[ep value]] && [number equal:[ep number]]);
}
```

where the method `equal` is defined in the superclass **Object**.

The **Process** class is implemented in the files: `Process.h` and `Process.m`; and it is instantiated as in (8). The processes are added by the following code:

```
(13): SkbList=[aProcess add_process:SkbList];
```

where `SkbList` is a global variable for the database of processes.

8.5.9 The State, Controller, ClosedLoop Classes

The **State, Controller, ClosedLoop** classes are implemented in the files: `State.h` and `State.m`, `Control.h` and `Control.m`, `ClosedLoop.h` and `ClosedLoop.m`, respectively.

The most important tasks implemented by the **State** class is to create the `rootSet` which contains only the root states that are not repeated and to create the `StateBase` which contains all the possible states of the system. For the first state

```
(14): states=[[State create] readFirstState:pp];
(15): StateBase=[states add_state:StateBase];
(16): aState=[[State create] readFirstState:pp];
(17): rootSet=[aState add_state:rootSet];
```

Then find the enabled event set of the current state aState

```
(18): [aState findEventSet:[rootSet name]:eLeve with:m];
```

For each event in the enabled event set

```
(19): ep=[aState eventSet];
```

find the next state of aState

```
(20): states=[[State create] read_state:aRoot from:pp at:[ep level]];
(21): StateBase=[states add_state:StateBase];
(22): aState=[[State create] read_root:StateBase at:k];
rootSet=[aState add_state:rootSet];
```

Then repeat the procedures from (18) to (23). Finally, the `rootSet` gives the reachability set, and the `StateBase` together with the enabled event set produces the reachability graph routines.

The similar procedures are implemented for the **Controller** and **Closed-Loop** classes. In the **ClosedLoop** class, the methods `readFirstState:` in (14)) and (16), `findEventSet::` `with:` in (18), and `readState:` `from:` `at:` in (20) are modified to synthesize the states from the **State** class and the controls from the **Controller** class.

8.5.10 The Optimizer Class

The **Optimizer** class is implemented in the files: `AStar.h` and `AStar.m`; and it is instantiated as in (11). It manipulates two sets: `openS` and `closeS`. To find the minimum-cost path, it uses the `select_best:` method which is defined as follows:

```
-select_best:(id)gl
```

```
{ id lt,stat;
  lt=openS;
  if(strcmp([lt name],[gl name]))
     [self better:lt:[lt next]:gl];
  else
    { stat=[self find_state:[lt name]];
      return stat;
    }
}
```

The message better::: is sent to self to compare the costs of two states which are not the goal state and to find a better one. Then the searching is directed by the smaller evaluation function using the smaller: method whose source code is presented below.

```
-(int)smaller:(int)k
{ int i,t=0;
  id cs,os;
  cs=closeS;
  for(i=0;i<k;i++)
     { if(!strcmp([cs name],[self name]))
         { if([cs fVal] > [self fVal])
             { t=1;
               i=k;
             }
         }
```

```
        else cs=[cs next];
      }
  return t;
}
```

8.6 Discussion

In this chapter, a software package has been designed to make logic evaluation and reasoning, to simulate the dynamical behavior of DES by its reachability set and reachability graph, to design and synthesize a controller to achieve the given requirements, and to do the optimization. The advantages of using an object-oriented language (Objective-C) as the programming environment have been highlighted. The software has been designed in an object-oriented paradigm with a hierarchy structure of classes; and the design of the software has been presented by outlining the design of each classes and their relationship. The implementation of the software has made use of the advantages of the object-oriented programming such as inheritance and dynamical binding. It provides an environment for proving the correctness of the models and theories established in this thesis.

As the related work to be compared, our work here is close to the work of O'Young and Wonham [88] and the work of Baldassari and Bruno [5]. The following gives the comparisons.

As an object-oriented computation and simulation of large-scale discrete event

systems [88], O'Young and Wonham have created a generic class DES based on Wonham's automata theory. This generic class contains a kernel set of class methods which enables an instance of its class or subclass to recognize a string in the closed and marked languages generated by its encapsulated transition structure. They designed methods in their product structures without any details of their implementation. Neither language nor software was presented in [88]. They did not have the logic evaluation or reasoning mechanism, nor methods for optimization.

In their paper, Baldassari and Bruno [5] presented an object-oriented language and methodology, PROTOB, and the CASE environment that supports it. The CASE environment consists of several tools supporting specification, modelling and prototypical activities using Petri nets. It can automatically generate the distributed and object-oriented implementation code in Ada or C. While they had a method for a similar kind of optimization they did not have the controller synthesis. Also they did not have the logic evaluation or reasoning mechanism.

There is a large room for improving the software presented in this chapter. As a user interface, *dedsMain*, the main program of the software, could be further implemented into a graphical user interface. In the *Evaluator*, we have implemented the logic connectives *and, not, or* and the temporal operator *next*; and this is only the very first step in building a TL verifier. Further implementation could include the quantifiers *all, some* and the temporal operators *until, eventually*, and so on. These further implementations will be continued by the members of our research

group.

The simulation results of using this software package will be presented in the next chapter.

Chapter 9

Simulating DESs with the Software Package

In this chapter, we demonstrate three examples of simulation using the software designed in the previous chapter. In Section 9.1, we show an example of logic evaluation. In Section 9.2, we present a gun shooting example to show the logic inferring with the software, and compare it with other implementation. In Section 9.3, we give an example of a flexible manufacturing system, a typical DES, to demonstrate the reachability analysis, the controller design, and the optimization. Comparisons with related works are made to show the advantages of the software. Finally, we give the conclusion of this chapter in Section 9.4.

9.1 An Example of Logic Evaluation

We are given a group of facts. The facts are given in the following form:

time i j predicateName argList value

It means that from time i to time j, the predicate *predicateName* with the argument list *argList* has the truth value *value*. A value of 0 and 1 is interpreted as false and true, respectively. This establishes the domain of the predicate as the time interval $[i, j]$ where i and j are integers. For example, *time 1 5 friend tom,cathy 1* implies that the predicate $friend(tom, cathy)$ is true from time 1 to time 5. *time 4 9 friend tom,smith 0* means that the predicate $friend(tom, smith)$ is false from time 4 to time 9; and it is the negation of *time 4 9 friend tom,smith 1*.

Based on a group of such facts, the software can evaluate the truth of a statement. All statements have to be enclosed with brackets '( )'. The following logic operators are supported: *and*, *not*, and *or*. They have the following formats:

$$(and\ (\ *\ )\ (\ *\ )\ \cdots\ (\ *\ )\ )$$

$$(not\ (\ *\ )\ )$$

$$(or\ (\ *\ )\ (\ *\ )\ \cdots\ (\ *\ )\ )$$

where $*$ stands for an expression which may include again *and*, *not*, and *or*. There is no limit on the number of expression items inside of these operators.

The temporal expression is of the format

$$time\quad i\quad j\quad temporalOp$$

For example, `(time i j (next (predicateName argList)))` means that the predicate *predicateName* with the argument list *argList* will be evaluated from $i + 1$.

```
time 1 5 friend tom,cathy 1
time 4 9 friend tom,smith 0
time 2 9 has tom,car 1
time 4 8 great x,y 1
time 1 8 less y,z 1
time 1 9 equal y,10 1
```

Figure 9.1: Facts in the evaluation example.

In the simulation, we first input a group of facts; and then evaluate some statements. An example of a group of facts is shown in Figure 9.1 and the evaluation of some statements is given in Figure 9.2 which shows that the operators *and*, *not*, and *or* can be inside of each other. The complete screen display for the outcome of this logic evaluation example can be found in Appendix A.

9.2 A Gun Shooting Example

Consider a simple example which involves reasoning about firing a loaded gun as shown in Figure 9.3. Suppose that the gun is loaded at time 1 and that the trigger of the gun is pulled at time 5. We want to be able to conclude that a loud noise will take place at time 6.

This example is taken from [102] where it was used for reasoning causation. Informally, causation is the relationship between reason and result; and the causal rule is used to infer the result from the reason. In our terminology, the rule which

```
Enter the  statement.
( and ( time 2 5 ( friend tom,cathy ) )
      ( time 4 6 ( next ( has tom,car ) ) ) )
Statement is True.

Enter the  statement.
( and ( time 2 5 ( friend tom,cathy ) )
      ( not ( time 4 6 ( has tom,car ) ) ) )
Statement is False.

Enter the  statement.
( or ( time 5 8 ( friend tom,smith ) )
     ( time 1 3 ( next ( friend tom,cathy ) ) ) )
Statement is True.

Enter the  statement.
( and ( or ( time 5 7 ( great x,y ) ) ( time 2 6 ( equal z,9 ) ) )
      ( and ( time 1 3 ( less y,z ) ) ( time 1 4 ( equal y,10 ) ) )
)
Statement is True.

Enter the  statement.
( or ( and ( time 5 7 ( great x,y ) ) ( time 2 6 ( equal z,9 ) ) )
     ( or ( time 1 3 ( less y,z ) ) ( time 1 4 ( equal y,10 ) ) ) )
Statement is True.
```

Figure 9.2: The evaluation of statements.

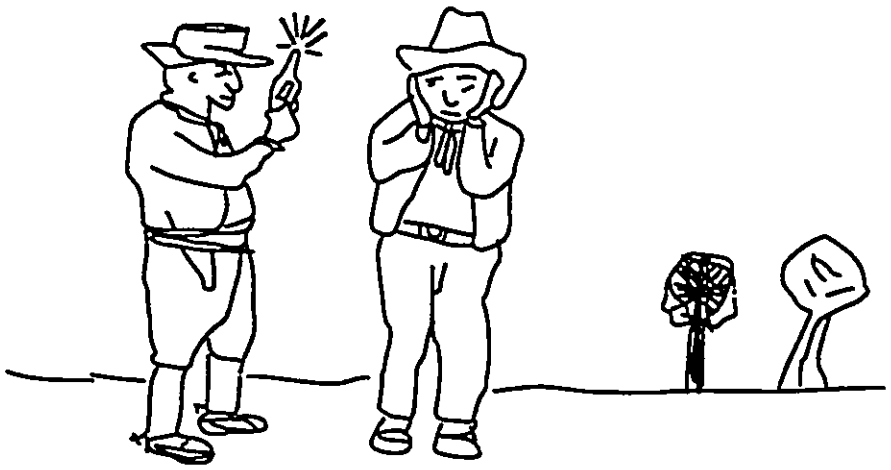


Figure 9.3: The gun shooting example.

says what conclusion comes from what condition is used to infer the conclusion by matching the condition against the given facts. This example seems to be able to serve this purpose; and therefore we take it as an example of reasoning the conclusions from the given facts and rules. The reason why this example is chosen also can be seen from the note given in [102] as follows:

> One feels the need to apologize for such violent examples (even though no one is about to get hurt). By way of such an apology let me point out that the prototypical examples of causation center either around an application of force which results in movement, or around an action which results in some bodily harm. The current discussion will turn out to be intimately connected to the notion of causation.

Here we assume that the gun shooting follows the following rules:

- If the gun is loaded and if the trigger of the gun is not pulled, then the gun

will be loaded.

- If the gun is loaded, if it is not in vacuum, and if the trigger of the gun is pulled, then a loud noise will take place.

To simplify the discussion, the statement that *it is not in vacuum* is representing the conjunction of *(it is not in vacuum)* and *(it is not emptied manually)* and ..., all those mundane facts that are, strictly speaking, needed in order to make a sound reasoning.

The symbol x is used to represent the state variable of the gun. The predicate $loaded(x)$ represents that the gun is loaded; $noise(x)$ represents that the loud noise takes place; $air(x)$ represents that the gun is not in vacuum. The event that the trigger of the gun is pulled is written as $efire(x)$. Then the rules are given as follows:

$$\Box[loaded(x) \wedge \neg efire(x) \Rightarrow loaded(\bigcirc x)] \tag{9.2.1}$$

$$\Box[loaded(x) \wedge air(x) \wedge efire(x) \Rightarrow noise(\bigcirc x)] \tag{9.2.2}$$

Based on this temporal logic model of the gun shooting example, the software gives the results of the reasoning as shown in Figure 9.4 where $enfire$ stands for $\neg efire$.

In Figure 9.4, Process 1 says that initially the gun is loaded at time 1. Events $e1$, $e2$, $e3$, $e4$ are that the trigger of the gun is not pulled at times 1, 2, 3, 4, respectively. Correspondingly, Processes 2, 3, 4, 5 are saying that the gun is loaded

```
Events:
 e1=(enfir 1)
 e2=(enfir 2)
 e3=(enfir 3)
 e4=(enfir 4)
 e5=(efire 5)

Conclusions:
 Process 1=(loadEd 1)
 Process 2=(loadEd 2)
 Process 3=(loadEd 3)
 Process 4=(loadEd 4)
 Process 5=(loadEd 5)
 Process 6=(noisE 6)
```

Figure 9.4: Reasoning of the gun shooting example.

at times 2, 3, 4, 5, respectively. Event $e5$ is that the trigger of the gun is pulled at time 5. Then process 6 is saying that a loud noise takes place at time 6. This finishes the reasoning. The complete outcome of this simulation can be found in Section 2 of Appendix A.

Comparing the reasoning of this example in [102] where Shoham used it to demonstrate his causation theories on reasoning about change, we used it as an example of reasoning in our temporal logic framework. He did it only as a theoretical reasoning while we performed its reasoning by our software as a simulation.

9.3 An Example of Simulating DESs

Consider an automated machine-shop in a factory [105] which can process parts of two different classes: custom and stock. These parts must be processed according to the following rules:

R1. Custom and stock parts are to be processed in mutual exclusion; *i.e.* custom parts must not be processed at the same time as stock parts;

R2. Custom parts have priority: once a custom part arrives at the machine-shop, no more stock parts are to be accepted until there are no longer any custom parts waiting for processing or be processed;

R3. Custom parts require individual attention and so are to be processed once at a time, and in the order in which they arrive;

R4. Any number of stock parts may be processed concurrently and, unless there are custom parts at the machine-shop, all stock parts that arrive are to be accepted for processing.

In this section, we use the simulator, one of the tools in the software package, to simulate this discrete event system. In the next subsection, we simulate the natural behavior of the uncontrolled system, the plant. In Subsection 9.3.2, we present the monitor of the software corresponding to the given rules of the required behavior of the system. In Subsection 9.3.3, we simulate the behavior of the controller given in [105]. In Subsection 9.3.4, we demonstrate the behavior of the closed-loop system and verify the results given in [105]. Comparisons with [105], [88], and [5] are given when appropriate.

9.3.1 The System Model and Its Behavior

The example is specified in the temporal logic framework [105]. For the plant, the following notations are used. The symbols O, W, and P represent the three possible states of a custom part: those of being outside the machine-shop, waiting to be processed, and being processed, respectively. The symbol n represents the number of stock parts being processed. The symbols $x_1, x_2, \cdots, x_N$ are used as the local variables of the N custom parts. For any i, $0 < i \leq N$, the event symbols α_i, β_i, and γ_i represent the arrival, commencement of processing, and departure, respectively. The event symbol λ represents the arrival and immediate rejection of a stock part while μ stands for the arrival of a stock part and the immediate commencement of its processing. The symbol ν is used to represent the departure

of a stock part and the symbol ϵ stands for the "null" event. The specifications of the plant are as follows:

$$\Box[\vee_{i=1}^{N}(\delta=\alpha_i \vee \delta=\beta_i \vee \delta=\gamma_i) \vee \delta=\mu \vee \delta=\nu \vee \delta=\lambda \vee \delta=\epsilon] \quad (9.3.P1)$$

$$\Box\{\wedge_{i=1}^{N}[\delta=\alpha_i \Rightarrow x_i=O \wedge (\bigcirc x_i)=W \wedge (\wedge_{i\neq j=1}^{N}(\bigcirc x_j)=x_j) \wedge (\bigcirc n)=n]\} \quad (9.3.P2)$$

$$\Box\{\wedge_{i=1}^{N}[\delta=\beta_i \Rightarrow x_i=W \wedge (\bigcirc x_i)=P \wedge (\wedge_{i\neq j=1}^{N}(\bigcirc x_j)=x_j) \wedge (\bigcirc n)=n]\} \quad (9.3.P3)$$

$$\Box\{\wedge_{i=1}^{N}[\delta=\gamma_i \Rightarrow x_i=P \wedge (\bigcirc x_i)=O \wedge (\wedge_{i\neq j=1}^{N}(\bigcirc x_j)=x_j) \wedge (\bigcirc n)=n]\} \quad (9.3.P4)$$

$$\Box[\delta=\mu \Rightarrow (\bigcirc n)=n+1 \wedge (\wedge_{j=1}^{N}(\bigcirc x_j)=x_j)] \quad (9.3.P5)$$

$$\Box[\delta=\nu \Rightarrow n>0 \wedge (\bigcirc n)=n-1 \wedge (\wedge_{j=1}^{N}(\bigcirc x_j)=x_j)] \quad (9.3.P6)$$

$$\Box[\delta=\lambda \Rightarrow (\bigcirc n)=n \wedge (\wedge_{j=1}^{N}(\bigcirc x_j)=x_j)] \quad (9.3.P7)$$

$$\Box[O \neq W \wedge O \neq P \wedge P \neq W] \quad (9.3.P8)$$

$$\wedge_{i=1}^{N} x_i = O \wedge n=0 \quad (9.3.P9)$$

These specifications have been given in [105] and are similar to those in Chapter 4; and therefore their explanations are omitted here. By using the simulator in the software package, we simulate this example for the simple case where there are two custom parts and two stock parts. A part of the simulation result of the plant system is shown in Figure 9.5 and its complete version can be found in Section 3 of Appendix A. The reachability graph of the plant is given in Figure 9.6.

The state transitions of the uncontrolled system are:

The root: s0=(0,0,0)
e1=(ecav 1) —> s1=(W,0,0)
e2=(ecav 2) —> s2=(0,W,0)
e3=(earr 3) —> s3=(0,0,0)=s0
e4=(eara 3) —> s4=(0,0,1)

The root: s1=(W,0,0)
e5=(ecsp 1) —> s5=(P,0,0)
e6=(ecav 2)=e2 —> s6=(W,W,0)
e7=(earr 3)=e3 —> s7=(W,0,0)=s1
e8=(eara 3)=e4 —> s8=(W,0,1)

The root: s2=(0,W,0)
e9=(ecav 1)=e1 —> s9=(W,W,0)=s6
e10=(ecsp 2) —> s10=(0,P,0)
e11=(earr 3)=e3 —> s11=(0,W,0)=s2
e12=(eara 3)=e4 —> s12=(0,W,1)

The root: s4=(0,0,1)
e13=(ecav 1)=e1 —> s13=(W,0,1)=s8
e14=(ecav 2)=e2 —> s14=(0,W,1)=s12
e15=(earr 3)=e3 —> s15=(0,0,1)=s4
e16=(eara 3)=e4 —> s16=(0,0,2)
e17=(eand 3) —> s17=(0,0,0)=s0

The root: s5=(P,0,0)
e18=(ecnp 1) —> s18=(0,0,0)=s0
e19=(ecav 2)=e2 —> s19=(P,W,0)
e20=(earr 3)=e3 —> s20=(P,0,0)=s5
e21=(eara 3)=e4 —> s21=(P,0,1)

The root: s6=(W,W,0)
e22=(ecsp 1)=e5 —> s22=(P,W,0)=s19
e23=(ecsp 2)=e10 —> s23=(W,P,0)
e24=(earr 3)=e3 —> s24=(W,W,0)=s6
e25=(eara 3)=e4 —> s25=(W,W,1)

The root: s8=(W,0,1)
e26=(ecsp 1)=e5 —> s26=(P,0,1)=s21
e27=(ecav 2)=e2 —> s27=(W,W,1)=s25
e28=(earr 3)=e3 —> s28=(W,0,1)=s8
e29=(eara 3)=e4 —> s29=(W,0,2)
e30=(eand 3)=e17 —> s30=(W,0,0)=s1

The root: s10=(0,P,0)
e31=(ecav 1)=e1 —> s31=(W,P,0)=s23
e32=(ecnp 2) —> s32=(0,0,0)=s0
e33=(earr 3)=e3 —> s33=(0,P,0)=s10
e34=(eara 3)=e4 —> s34=(0,P,1)

The root: s12=(0,W,1)
e35=(ecav 1)=e1 —> s35=(W,W,1)=s25
e36=(ecsp 2)=e10 —> s36=(0,P,1)=s34
e37=(earr 3)=e3 —> s37=(0,W,1)=s12
e38=(eara 3)=e4 —> s38=(0,W,2)
e39=(eand 3)=e17 —> s39=(0,W,0)=s2

The root: s16=(0,0,2)
e40=(ecav 1)=e1 —> s40=(W,0,2)=s29
e41=(ecav 2)=e2 —> s41=(0,W,2)=s38
e42=(earr 3)=e3 —> s42=(0,0,2)=s16
e43=(eand 3)=e17 —> s43=(0,0,1)=s4

The root: s19=(P,W,0)
e44=(ecnp 1)=e18 —> s44=(0,W,0)=s2
e45=(ecsp 2)=e10 —> s45=(P,P,0)
e46=(earr 3)=e3 —> s46=(P,W,0)=s19
e47=(eara 3)=e4 —> s47=(P,W,1)

The root: s21=(P,0,1)
e48=(ecnp 1)=e18 —> s48=(0,0,1)=s4
e49=(ecav 2)=e2 —> s49=(P,W,1)=s47
e50=(earr 3)=e3 —> s50=(P,0,1)=s21
e51=(eara 3)=e4 —> s51=(P,0,2)
e52=(eand 3)=e17 —> s52=(P,0,0)=s5

The root: s23=(W,P,0)
e53=(ecsp 1)=e5 —> s53=(P,P,0)=s45
e54=(ecnp 2)=e32 —> s54=(W,0,0)=s1
e55=(earr 3)=e3 —> s55=(W,P,0)=s23
e56=(eara 3)=e4 —> s56=(W,P,1)

.
.
.

Figure 9.5: A part of the simulation result of the plant.

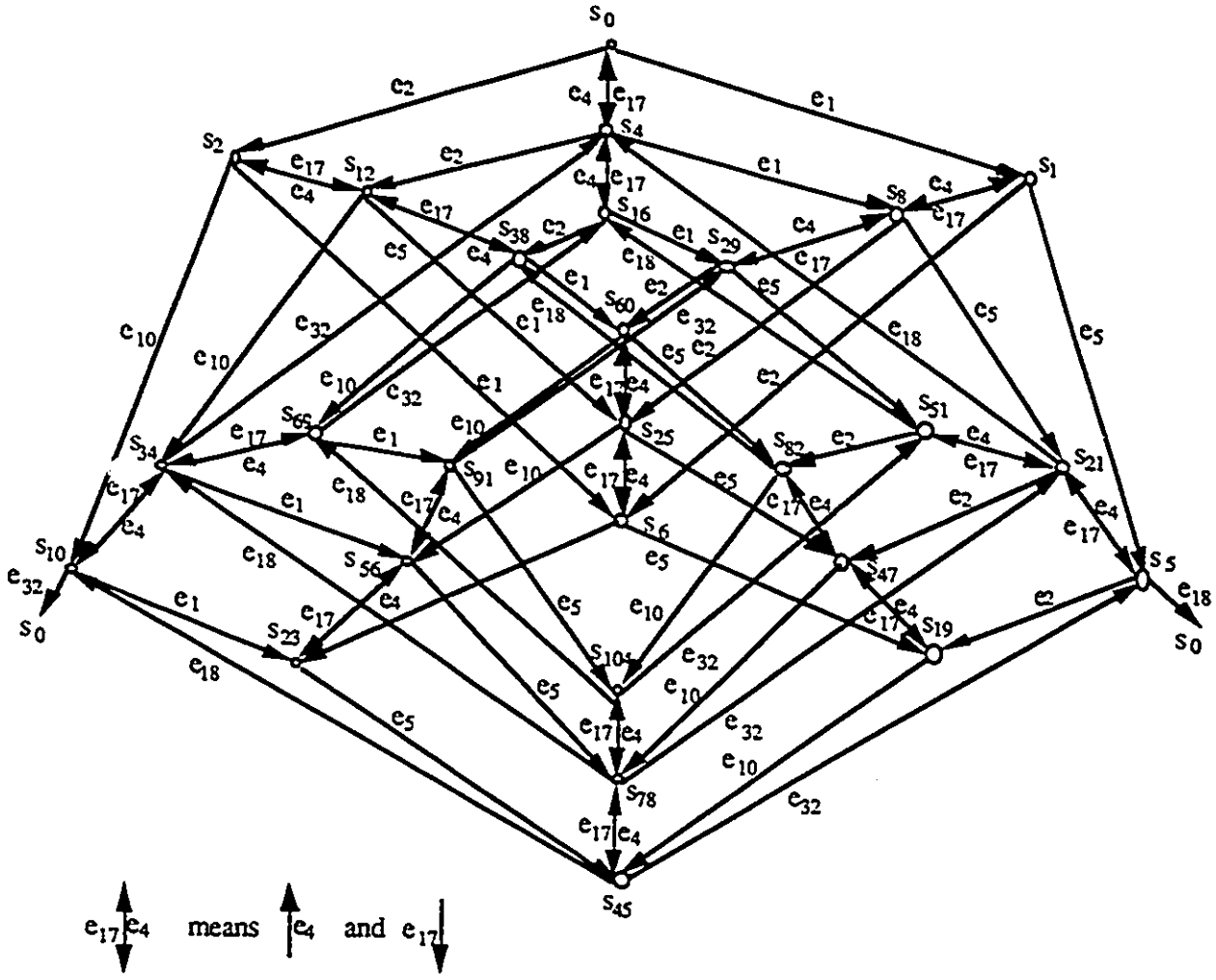


Figure 9.6: Reachability graph of the plant.

For N custom parts, in general, the plant has $n+1$ dimensions. Here $N=2$, so the plant is 3−dimensional. As a uncontrolled system at this point, it has its natural dynamic behavior. First, the explanation of notions in the graph is given here. $s12=(O,W,1)$ means that at state $s12$, one custom part is outside and other is waiting to be processed while one stock part is being processed. For $i=1,2$, $(ecav\ i)=\alpha_i$, $(ecsp\ i)=\beta_i$, $(ecnp\ i)=\gamma_i$, $(earr\ 3)=\lambda$, $(eara\ 3)=\mu$, and $(eand\ 3)=\nu$. The reachability graph is constructed by starting from a root through an edge of an event to reach the next state.

The dynamic behavior given by the reachability graph routines together with the enabled event sets are more intuitive than the production structure in [88] and the PROT nets in [5].

9.3.2 The Required Behavior and the Monitor

According rules $R1$-$R4$, expressions of the desired properties of the closed-loop system are formally given below.

$$\Box[\wedge_{i=1}^{N}(x_i=P\Rightarrow\wedge_{i\neq j=1}^{N}x_j\neq P)] \qquad (9.3.CL1)$$

$$\Box[\vee_{i=1}^{N}(x_i=P)\Rightarrow n=0] \qquad (9.3.CL2)$$

$$\Box[\vee_{i=1}^{N}(x_i=W)\Rightarrow\delta\neq\mu)] \qquad (9.3.CL3)$$

$$\Box[\wedge_{i,j=1;i\neq j}^{N}(x_i\neq W\Rightarrow(\delta=\alpha_i\mathcal{P}\delta=\alpha_j\Rightarrow\delta=\beta_i\mathcal{P}\delta=\beta_j))] \qquad (9.3.CL4)$$

$$\Box[(\wedge_{j=1}^{N}(x_i=O)\Rightarrow\delta\neq\lambda] \qquad (9.3.CL5)$$

```
According to the specifications of the required behavior
 Event e3 should be disabled at state s0 !
 s21 should not be reached! Event e21 should be disabled at state
s5 !
 s26=(P,0,1)=s21 So event e26 should be disabled at state s8 !
 s21 may not be a root!
```

Figure 9.7: Examples of suggestions from the monitor

According to the specifications of the required behavior, there are many undesirable states in the uncontrolled system. For instance, at state $s21 = (P, O, 1)$, custom part 1 is being processed while there is one stock part being processed. This violates the specification (9.3.$CL2$).

A list of suggestions is given by the monitor in the simulator. Following the specifications of the required behavior, the monitor finds two kinds of violations of the specifications. The first kind is that which reaches states that should not be reached. For instance, states violate the mutual exclusion such as (9.3.$CL1$) and (9.3.$CL4$). Once the monitor detects states that should not be reached, it finds out the enabled events for those states and suggests to disable those events. The second kind is that the events should be disabled at certain states such as those stated in (9.3.$CL3$) and (9.3.$CL5$). The monitor also remarks that the states which should not be reached may not be a root for a new branch.

Examples of suggestions from the monitor for the given required behavior is

shown in Figure 9.7 and the complete list can be found in Section 3 of Appendix A. The event disablements here are chosen according to the required behavior; and therefore the controller is designed by Procedure 6.4.1 to enable those events which lead to the required system behavior. They are NOT for getting our results to match those of Thistle [103]. After showing that our approach can do more than Thistle's approach, we now want to show that our approach also works for Thistle's example. This is why we take the example from Thistle's work. Since we are simulating the system from Thistle's work with the same required behavior of the closed-loop system, we ought to have the same control law to achieve the same required behavior.

Comparing with the work of O'Young and Wonham [88] and the software given by Baldassari and Bruno in [5], both of them did not have the facility to find the undesirable states and events.

9.3.3 The Controller Model and Its Behavior

In order to obtain the desired behavior of the system, our controller must ensure that these requirements are met. The controller should have a first-in-first-out queue to keep track of custom parts and the order in which they arrive. Also, a count should be set to monitor the number of stock parts being processed. Therefore the specifications for the controller are given as follows.

The symbol q represents the queue and it is assigned a value that is either

some finite sequence of integers between 1 and N, or the empty sequence denoted by the symbol ϕ. The symbol c represents the count and it is assigned a non-negative integer value.

$$\Box[\wedge_{i=1}^{N}(\delta = \alpha_i \Rightarrow (\bigcirc q) = q * i \wedge (\bigcirc c) = c)] \qquad (9.3.C1)$$

$$\Box[\wedge_{i=1}^{N}(\delta = \beta_i \Rightarrow i \propto q \wedge c = 0 \wedge (\bigcirc q) = q)] \qquad (9.3.C2)$$

$$\Box[(\vee_{i=1}^{N}(\delta = \gamma_i) \vee \delta = \nu) \Rightarrow (c = 0 \Rightarrow (\bigcirc q) = q|^{\mathrm{I}}$$

$$\wedge(\bigcirc c) = c) \wedge (c \neq 0 \Rightarrow (\bigcirc q) = q \wedge (\bigcirc c) = c - 1)] \qquad (9.3.C3)$$

$$\Box[\delta = \lambda \Rightarrow q \neq \phi \wedge (\bigcirc q) = q \wedge (\bigcirc c) = c] \qquad (9.3.C4)$$

$$\Box[\delta = \mu \Rightarrow q = \phi \wedge (\bigcirc q) = q \wedge (\bigcirc c) = c + 1)] \qquad (9.3.C5)$$

$$c = 0 \wedge q = \phi \qquad (9.3.C6)$$

The monitor does not check the order in which custom parts arrive and it leaves to controller to make the ordering. The state transitions of the controller is shown in Figure 9.8.

As specified in the controller specifications, the controller is 2−dimensional. It disables all the undesirable events as suggested by the monitor and orders the custom parts with a principle of "first-in, first-out". As shown in Figure 9.8, the first component of the controller state is the queue and the second is the count. Correspondingly, (E,*), (a,*), (ab,*), (A,*), (Ab,*), (b, *), (B,*) are saying that the queue is empty, the first custom part is waiting in the queue, the first and the second are waiting in the queue, the first is being processed, the first is being processed while the second is waiting, the second is waiting, the second is

The transitions in the controller are:

```
The root:  C0 =(E,0)
e1=(ecav 1)  -->  C1 =(a,0)
e2=(ecav 2)  -->  C2 =(a,0) = C1
e3=(eara 3)  -->  C3 =(E,1)
The root:  C1 =(a,0)
e4=(ecav 1)  -->  C4 =(ab,0)
e5=(ecav 2)  -->  C5 =(ab,0) = C4
e6=(ecsp 1)  -->  C6 =(A,0)
e7=(ecsp 2)  -->  C7 =(A,0) = C6
e8=(earr 3)  -->  C8 =(a,0) = C1
The root:  C3 =(E,1)
e9=(eand 3)  -->  C9 =(E,0) = C0
e10=(eara 3)  -->  C10 =(E,2)
The root:  C4 =(ab,0)
e11=(earr 3)  -->  C11 =(ab,0) = C4
e12=(ecsp 1)  -->  C12 =(Ab,0)
e13=(ecsp 2)  -->  C13 =(Ab,0) = C12
The root:  C6 =(A,0)
e14=(earr 3)  -->  C14 =(A,0) = C6
e15=(ecnp 1)  -->  C15 =(E,0) = C0
e16=(ecnp 2)  -->  C16 =(E,0) = C0
e17=(ecav 1)  -->  C17 =(Ab,0) = C12
e18=(ecav 2)  -->  C18 =(Ab,0) = C12
The root:  C10 =(E,2)
e19=(eand 3)  -->  C19 =(E,1) = C3
The root:  C12 =(Ab,0)
e20=(earr 3)  -->  C20 =(Ab,0) = C12
e21=(ecnp 1)  -->  C21 =(b,0)
e22=(ecnp 2)  -->  C22 =(b,0) = C21
The root:  C21 =(b,0)
e23=(ecsp 1)  -->  C23 =(B,0)
e24=(ecsp 2)  -->  C24 =(B,0) = C23
e25=(earr 3)  -->  C25 =(b,0) = C21
The root:  C23 =(B,0)
e26=(ecnp 1)  -->  C26 =(E,0) = C0
e27=(ecnp 2)  -->  C27 =(E,0) = C0
e28=(earr 3)  -->  C28 =(B,0) = C23
```

Figure 9.8: State transitions of the controller.

being processed, respectively. Similarly, (*,0), (*,1), (*,2) are saying that no stock part is being processed, one stock part is being processed, two stock parts are being processed, respectively.

In both [88] and [5], there have been the ideas of the controller or supervisor synthesis; but neither explicit procedures nor simulation results were given for the synthesis of a controller or supervisor.

9.3.4 The Behavior of the Closed-Loop System

With such a controller being synthesized onto the plant, the closed-loop system is composed by the plant and the controller such that it is 5−dimensional. Using the simulator, the reachability set and the dynamic behavior of the closed-loop system is shown in Figure 9.9 and its reachability graph is diagramed in Figure 9.10. The complete outcomes of the simulation is given in Appendix A.

As shown in Figure 9.9, the state of the closed-loop system is composed of the state of the plant and the state of the controller. For instances, S19=(P,W,0,Ab,0) means that the first custom part is being processed while the second is waiting; and S16=(0,0,2,E,2) says that two stock parts are being processed while no any custom part is in.

We can see from Figure 9.9 that the states in the reachability set satisfy the the specification of the required behavior of the system. It follows from Figures

Now the state transitions of the controlled system are:

```
The root:  S0 =(O,O,0,E,0)
e1=(ecav 1)   --> S1 =(W,O,0,a,0)
e2=(ecav 2)   --> S2 =(O,W,0,a,0)
e4=(eara 3)   --> S4 =(O,O,1,E,1)

The root:  S1 =(W,O,0,a,0)
e5=(ecsp 1)   --> S5 =(P,O,0,A,0)
e6=(ecav 2)=e2 --> S6 =(W,W,0,ab,0)
e7=(earr 3)=e3 --> S7 =(W,O,0,a,0)=S1

The root:  S2 =(O,W,0,a,0)
e9=(ecav 1)=e1 --> S9 =(W,W,0,ab,0)=S6
e10=(ecsp 2)   --> S10 =(O,P,0,A,0)
e11=(earr 3)=e3 --> S11 =(O,W,0,a,0)=S2

The root:  S4 =(O,O,1,E,1)
e16=(eara 3)=e4 --> S16 =(O,O,2,E,2)
e17=(eand 3)   --> S17 =(O,O,0,E,0)=S0

The root:  S5 =(P,O,0,A,0)
e18=(ecnp 1)   --> S18 =(O,O,0,E,0)=S0
e19=(ecav 2)=e2 --> S19 =(P,W,0,Ab,0)
e20=(earr 3)=e3 --> S20 =(P,O,0,A,0)=S5

The root:  S6 =(W,W,0,ab,0)
e22=(ecsp 1)=e5 --> S22 =(P,W,0,Ab,0)=S19
e23=(ecsp 2)=e10 --> S23 =(W,P,0,Ab,0)
e24=(earr 3)=e3 --> S24 =(W,W,0,ab,0)=S6

The root:  S10 =(O,P,0,A,0)
e31=(ecav 1)=e1 --> S31 =(W,P,0,Ab,0)=S23
e32=(ecnp 2)   --> S32 =(O,O,0,E,0)=S0
e33=(earr 3)=e3 --> S33 =(O,P,0,A,0)=S10

The root:  S16 =(O,O,2,E,2)
e43=(eand 3)=e17 --> S43 =(O,O,1,E,1)=S4

The root:  S19 =(P,W,0,Ab,0)
e44=(ecnp 1)=e18 --> S44 =(O,W,0,b,0)
e46=(earr 3)=e3 --> S46 =(P,W,0,Ab,0)=S19

The root:  S23 =(W,P,0,Ab,0)
e54=(ecnp 2)=e32 --> S54 =(W,O,0,b,0)
e55=(earr 3)=e3 --> S55 =(W,P,0,Ab,0)=S23

The root:  S44 =(O,W,0,b,0)
e10=(ecsp 2)=e10 --> S75 =(O,P,0,B,0)
e11=(earr 3)=e3 --> S76 =(O,W,0,b,0)=S44

The root:  S54 =(W,O,0,b,0)
e5=(ecsp 1)=e5 --> S88 =(P,O,0,B,0)
e7=(earr 3)=e3 --> S89 =(W,O,0,b,0)=S54

The root:  S75 =(O,P,0,B,0)
e32=(ecnp 2)=e32 --> S101 =(O,O,0,E,0)=S0
e33=(earr 3)=e3 --> S102 =(O,P,0,B,0)=S75

The root:  S88 =(P,O,0,B,0)
e18=(ecnp 1)=e18 --> S110 =(O,O,0,E,0)=S0
e20=(earr 3)=e3 --> S111 =(P,O,0,B,0)=S88

The reachability Set:
 S0 =(O,O,0,E,0) S1 =(W,O,0,a,0) S2 =(O,W,0,a,0)
S4 =(O,O,1,E,1) S5 =(P,O,0,A,0) S6 =(W,W,0,ab,0)
S10 =(O,P,0,A,0) S16 =(O,O,2,E,2) S19=(P,W,0,Ab,0)
S23 =(W,P,0,Ab,0) S44 =(O,W,0,b,0)S54=(W,O,0,b,0)
S75 =(O,P,0,B,0) S88 =(P,O,0,B,0)
```

Figure 9.9: A part of the simulation result of the closed-loop system.

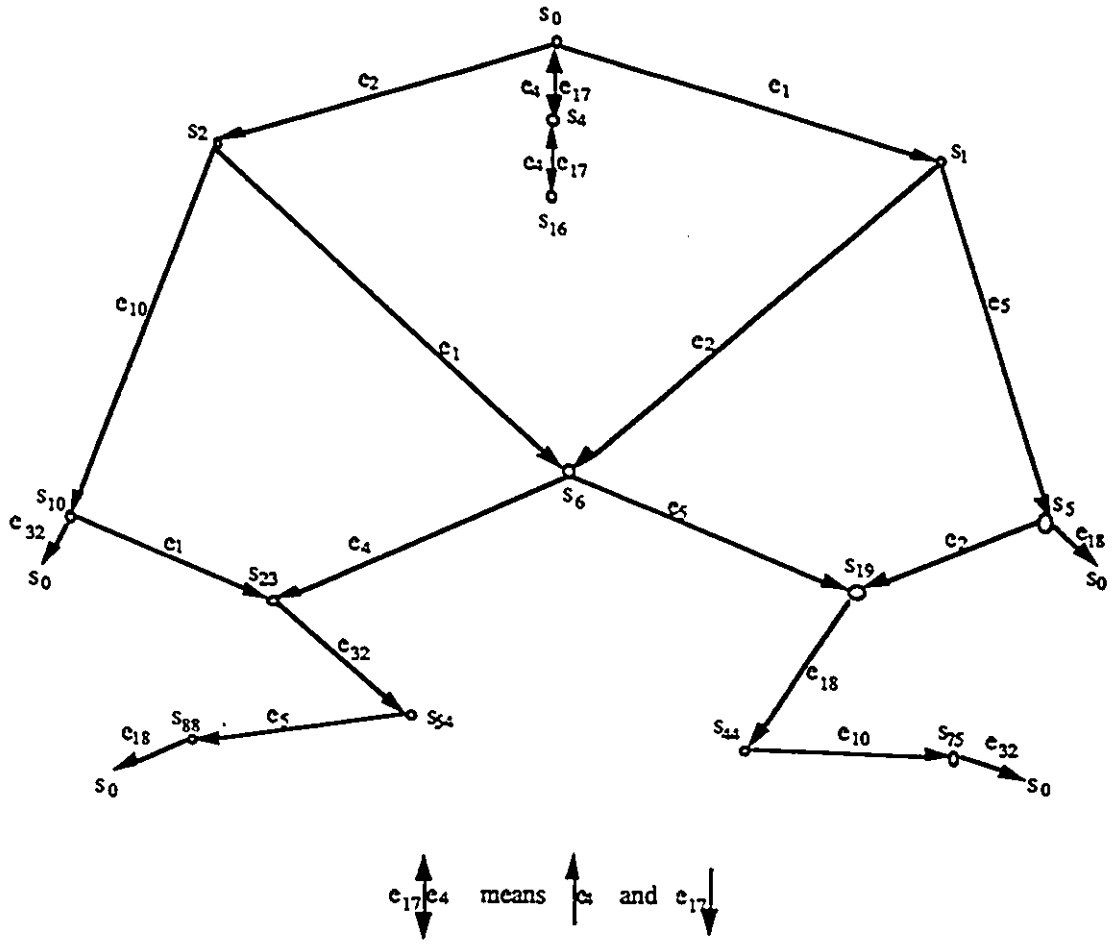


Figure 9.10: The reachability graph of the closed-loop system.

9.6, 9.7, and 9.10 that $R(\overline{M}, (s_0, q_0)) \rightleftharpoons R(M, s_0) - R_u$. By Corollary 5.3.1, and Procedure 6.4.1, this proves that the closed-loop system has the required dynamic behavior. By Theorem 6.4.2 we have proved the results given in [105].

In [105], Thistle and Wonham have verified that the required behavior of the closed-loop system given by (9.3.CL1) – (9.3.CL5) can be deduced from the specification given in (9.3.P1) – (9.3.P9) and the specification of the controller given in (9.3.C1) – (9.3.C6). In this chapter, we have done a verification by using Algorithm 5.3.1 in Chapter 5 and Procedure 6.4.1 in Chapter 6. It turns out that the simulation results are same as Thistle's results; and it proves the correctness of our approach and Thistle's work. If the required behavior is chosen to be different from Thistle's, then the controller designed by Procedure 6.4.1 will be different from Thistle's. In this thesis, we can get not only the same results as those of Thistle's work but also the probabilistic feature which is more than what Thistle's method can do. In practical problems, Thistle's axiomatic inductive proof may not be feasible but our simulation may still work it out. Moreover, in this section we simulated this example while there was no such simulation in [105].

9.3.5 The Optimization

Figure 9.10 shows that there are more than one path from one state to another state, for instances, from $S0$ to $S19$ and to $S54$, from $S10$ to $S6$, from $S16$ to $S19$, from $S23$ to $S88$, and so on. For this manufacturing system, it is obvious that the costs might be different through different paths from the same initial

state to the same final state. Therefore, it is of great significance to find out the path which has the minimum of the costs and then to generate the corresponding sequence of events that drives the system along that path.

Now the costs of events for this example are given according to the costs of operations in the machine-shop, as listed in Table 9.1. To use the A^* algorithm to solve the optimization problem, the heuristic index will be needed. The state preference indexes of the manufacturing system are given in Table 9.2.

events e	e_1	e_2	e_3	e_4	e_5	e_{10}	e_{17}	e_{18}	e_{32}
costs θ	1	2	1	2	1	1	2	5	5

Table 9.1: Costs of events.

states s_i	S_0	S_1	S_2	S_4	S_5	S_6	S_{10}	S_{16}	S_{19}	S_{23}	S_{44}	S_{54}	S_{75}	S_{88}
$index(s_i)$	0	1	2	2	1	2	3	3	3	3	4	4	5	5

Table 9.2: Indexes of states.

Let the heuristic function

$$\hat{h}(s_i, s_j) \stackrel{\text{def}}{=} |index(s_i) - index(s_j)|$$

Then it is a semi-metric function and there holds

```
Please enter the start node: S0
Please enter the end node: S54
 S0 =(0,0,0,E,0) S54 =(W,0,0,b,0)
 The optimal path (in backward order):
S54 f=13  g=9 e54
S23 f=7  g=4 e23
S6 f=5  g=3 e6
S1 f=2  g=1 e1
S0 f=0  g=0 (null)
  The end of optimization!
```

Figure 9.11: An example of optimization

$$\hat{h}(s_i, s_j) = |\sum_{k=i}^{j-1}[h(s_k) - h(s_{k+1})]|$$

$$\leq \sum_{k=i}^{j-1}|h(s_k) - h(s_{k+1})| = \sum_{k=i}^{j-1}\theta(e_{k,k+1}, s_k) = \theta(e_{ij}, s_i)$$

We choose some states from the reachability set of the closed-loop system as the initial states and the final states. An example of the simulation of finding the minimum-cost paths by the simulator is given in Figure 9.11 and the complete simulation results are shown in Section 3 of Appendix A.

Figure 9.12 gives the optimal trajectories and the corresponding sequences of events for the initial state S_0 and the final state S_{19}, and for the initial state S_0 and the final state S_{54}, respectively.

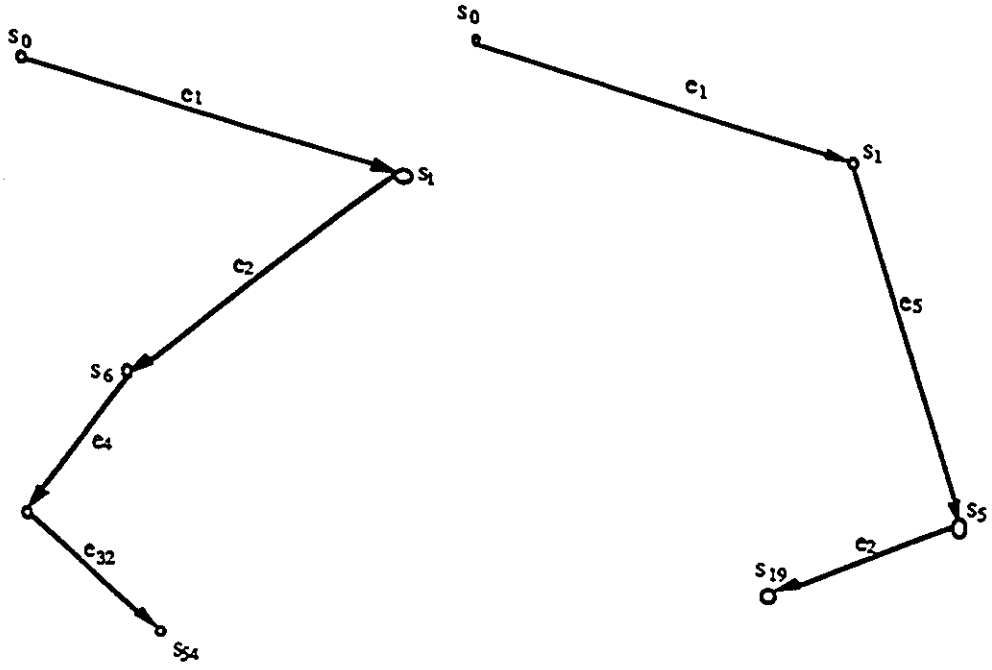


Figure 9.12: Optimal paths.

In the production control system of [5], the routine of parts is controlled by a dispatcher in a kind of optimal way to the scheduled production plans and the available resources. No explicit optimization procedure was presented there. In [88], there was no discussion of any optimization.

9.4 Conclusion

The simulation results of three examples have been presented to demonstrate that our software package has the functions: the logic evaluation and reasoning, the reachability analysis, the controller design and synthesis, and the optimization. These results have matched the theoretical results developed in Chapters 3 – 7; and therefore we have verified that they are correct. Comparisons with the related works are made through out the chapter to show the advantages of our software package.

Chapter 10

Summary, Discussions, and Suggestions

As the final chapter of this thesis, we first briefly summarize the contributions we made in both theoretical results and applications. We then compare our results with related works to show the advantages of our approach. Finally, we give suggestions for further research.

10.1 Contributions of the Thesis

Discrete event systems are receiving attention for their applications in computer science and control systems. As a part of the continuing research work in this area, the analysis and synthesis of discrete event systems have been addressed in this thesis by a temporal logic approach. The contributions of this thesis are as follows.

- Based on theories and methods established both in computer science and in control system theory, a temporal logic framework has been developed to provide a language for the specifications and verification, a facility for the modeling and analysis, and a procedure for the controller design and synthesis.

- A temporal logic model has been defined including the probability distributions, and a generalized temporal logic language has been formulated with adding the certainty operators to handle the probabilistic features of the DESs.

- With a system of proof rules, the generalized temporal logic framework has been applied to the specifications and verification of properties for the control problem of DESs.

- The dynamics of DESs has been analyzed, the relationship between reachability of states and validity of formulas has been established, and an algorithm has been developed for computing the reachability set and constructing the reachability graph.

- By applying a process algebra, the composition and synthesis of processes have been investigated through the process homomorphism; and a procedure has been proposed for the controller design and system configuration.

- A semi-metric space has been defined, the optimization problem of discrete event systems has been discussed and solved by the A^* algorithm via a heuristic search.

- Within such a framework, a software package has been developed for the temporal logic evaluation, reasoning about and simulating discrete event systems. The software has been designed in an object-oriented approach and implemented in Objective-C. The simulation results have been reported in terms of the logic evaluation, temporal logic reasoning, and discrete event system simulation.

- The examples of applications such as flexible manufacturing systems, data link protocols, computer read-write processes, and packet-switched communications networks, have been also given to convey and motivate the theoretical discussions.

Comparisons with related works have been made through out the thesis to illustrate the advantages of our results. In the next section, we will do more comparisons and also highlight some of them that we did in the previous chapters.

10.2 Comparisons with Related Work

In this section, we discuss the comparison of our results with some of related works. Firstly, we talk about the qualitative reasoning approaches which are neither for DESs nor in temporal logic but related to our reasoning. Secondly, we discuss some of the approaches of DESs other than temporal logic. Thirdly, we compare our results with other proposals in the temporal logic approach. After all, our results are seen in many cases to outperform these approaches.

10.2.1 Qualitative reasoning

In Chapter 3, we have defined a temporal logic model to the DES with the probability distributions known, and extended a temporal logic language to include certainty operators. With a system of proof rules, the verifications have been qualitative in proving closed-loop specification formulas from the plant and the controller. We have shown a qualitative analysis for a class of probabilistic systems, and that this analysis may be completed without any need to use the sophisticated probability theory. In the following, we compare our work with some other qualitative reasoning approaches.

In [50,51], B. Kuipers presented a qualitative reasoning method for predicting the behavior of systems characterized by continuous time-varying parameters. The structure of a system is described in terms of a set of parameters and constraints that hold among them: essentially a *qualitative differential equation* (QDE). The qualitative behavior description consists of a discrete set of time points, at which the values of the parameters are described in terms of ordinal relations and directions of change. The behavioral description is derived by two sets of rules: propagation rules which elaborate the description of the current time point, and prediction rules which determine what is known about the next qualitative distinct state of the system. This derivation process has been embodied in a constraint-filtering algorithm called QSIM which generates the tree of the possible behaviors following from a given initial state and a QDE.

With the *quasi-equilibrium assumption* that the system is always in, or in-

finitely close to the equilibrium, the effect of changes to such a system can be deduced simply by solving the equilibrium equations for the new state. This is essentially the inference method used by de Kleer and Brown [18].

As a model building process for qualitative reasoning, qualitative process theory by Forbus [26] has two fundamental types of description: *individual views*, which represent objects or sets of objects viewed in a particular way, and *processes* , which represent active changes taking place. All changes are considered to emanate from processes. The model building process can be summarized as follows.

1). Each individual view and process checks various conditions about the world to determine whether it should have one or more active instances.

2). The active view and process instances are grouped into sets of mutually consistent elements, called view-process structures.

3). Each view or process in a view-process structure contributes fragments, called direct or indirect influences to the constraint model.

4). The Closed World Assumption is applied with in each view-process structure. This makes it possible to determine the set of all influences, direct or indirect, applying to any given quantity, so the influences can be translated to constraints.

5). Then the resulting constraint model can be simulated.

The qualitative reasoning approaches mentioned above would not on their own have been sufficient for setting up a framework for modeling and control of their system. The qualitative process theory emphasizes on model building while QSIM emphasizes simulating. Comparing these approaches with ours, the crucial

```
Now the state transitions of the U-tube system are:

The root:   S0 =(AMAX,0)
e1=(ebegin 1)     --->  S1 =(Ax,By)

The root:   S1 =(Ax,By)
e2=(end 1)     --->  S2 =(AE1,BE)
e3=(end 2)     --->  S3 =(AE2,BMAX)
```

Figure 10.1: The behavior of the U-tube system.

control issue is not addressed and no formal proof systems are supplied in their approaches.

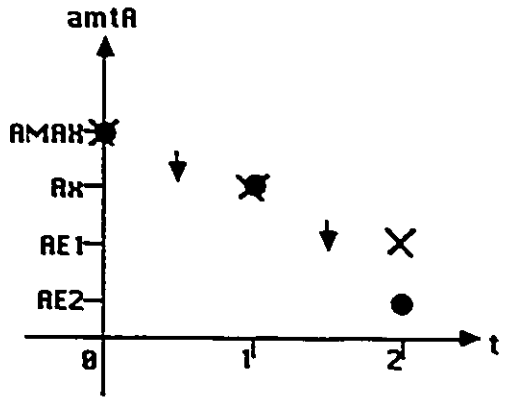
Our software is able to do the simulation that QSIM can do. For example, the U-tube in [50,51] can be simulated in our software and the qualitative behavior of the U-tube system is given by Figure 10.1.

Here Ax, $AE1$, $AE2$, $AMAX$ (By, BE, $BMAX$) stand for the values of amount of water in tank A (B, respectively) with $AE1$, BE for those when the U-tube reaches equilibrium for the case that B is not full and with $AE2$, $BMAX$ for the case B is full or overflows; and there hold

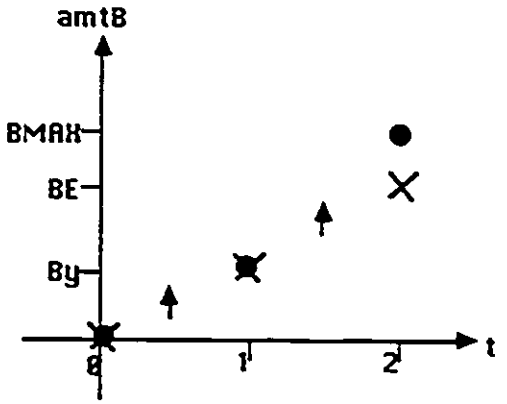
$$0 < AE2 \leq AE1 < Ax < AMAX < +\infty$$

$$0 < By < BE \leq BMAX < +\infty$$

If we decompose the above reachability graph, then the behavioral graphs of tanks



(a). Behavior of tank A.



(b). Behavior of tank B.

● The case that tank B is full or overflows

× The case that tank B is not full

Figure 10.2: The behavioral graphs of tanks A and B.

A and B are as shown in Figure 10.2. These are the exactly same as the behavioral graphs given by QSIM.

10.2.2 Other approaches of DESs

Comparing with other approaches for the modelling of DESs, the examples we used so far have shown that the proposed temporal logic approach leads to easily understandable high-level specifications and to a formal verification procedure. Although temporal logic might be cumbersome for extremely large systems, for relatively small systems (practically we always decompose the large system into a number of smaller systems), translating a natural language specifications into temporal logic is more straightforward than the process of finding the corresponding grammar in the formal language and automata approach [95] which has been proved useful in theoretical analysis of DESs. Another approach to be compared is Petri nets. Although Petri nets have the advantage of a graphical representation, it lacks satisfactory verification methods for liveness properties as shown in the examples in [21] where an additional language is needed to express the required behavior over a sequence of states.

Among related works on the optimal control problem of DESs, the closest to ours is that of Passino and Antsaklis [89]. We use the temporal logic model while they use the abstract logic model. We emphasize on the optimization procedure while they on the heuristic function. Besides, the semi-metric space we defined is more general than their metric space to handle the case that the event cost function $\theta(e_{ij})$ may be different from $\theta(e_{ji})$. By using a non-symmetric semi-metric,

we can chose a heuristic function to close the non-symmetric cost function. The standard dynamic programming may also solve the optimization problem if it can find a solution since its optimal condition is only a necessary condition. We use the A^* algorithm as a sufficient condition for optimality which has the advantages of uniform searching techniques over the standard dynamic programming.

10.2.3 Other proposals in temporal logic

The Interval Temporal Logic (ITL) of Moszkowski [80] extends ordinary temporal logic with additional operators such as *chop*, which is used to divide the time interval into two adjacent intervals. ITL is more expressive than ours in some aspects but it does not have a system of proof rules. Furthermore, since intervals are finite sequences of states, the *infinitely often* property of temporal logic can not easily be expressed.

For real-time DESs, Ostroff introduced a Real-Time Temporal Logic (RTTL) in [85] with an extended state machine models. RTTL is more expressive than ours in real-time aspects whereas ours is more expressive than RTTL in dealing with probabilistic features of DESs as shown in this thesis. The difference between these two systems is that RTTL is not strongly sound and ours is not strongly complete. This is resulted from adopting the Insertion Rule in RTTL but not in ours. As a result, state formulas are interpreted with respect to the initial state of a sequence in ours, and to all states of a sequence in RTTL.

From a control perspective, Thistle and Wonham used a linear-time temporal logic framework in [105] for the specifications and verification of control systems. In Thistle's work, a notion is set to a sort of temporal logic for system events. This allows for an easy reference to events when specifying the properties of the system. A disadvantage of Thistle's approach is that the full plant-controller must be available before the verification can proceed.

Comparing our work with Thistle's, we have extended Thistle's temporal logic to a generalized temporal logic which includes the modal operators about certainty to handle the case where the point probability distributions are known. Our method of introducing certainty operators is both simple and at same time expressive enough to assert the probabilistic properties. Our result on the relationship between reachability of states and validity of the corresponding logic formulas allows for an early verification: we may verify the specification of a DES's plant by the reachability algorithm before a controller is available. Furthermore, our composition and synthesis theory provides a procedure for designing and synthesizing a controller into the system. The optimization of DESs generates a sequence of events which produces an optimal path with the minimum cost. Finally, our software tools simulate and verify our results.

10.3 Suggestions for Further Research

This thesis provides a temporal logic approach to the analysis and synthesis of discrete event systems. There are still many interesting open problems in this

approach. Some suggestions for further research work are discussed below.

1. Applications of our temporal logic framework to the AI problem solving. Although many of the basic issues in AI problem solving are well understood [81], they have not been adequately formally described in a mathematical logic framework. Hence there is a need to develop a foundation of a temporal logic framework for modeling, analysis and design of AI problem solving. Our temporal logic framework can be used as such a fundamental logic framework. In [65], we have tried to compare our temporal logic model with a production system, and to develop a reasoning mechanism based on a forward chaining procedure. In [64], we have used A^* algorithm for searching a space of configurations in a way similar to that we did in Chapter 7. These initial attempts have been encouraging, and further steps are needed to be taken to turn the temporal logic framework into a formal method useful on the AI problem solving. Examples of open problems for further research are the computational complexity of the A^* algorithm and the selection of a heuristic function. In addition, there is a need for temporal logic reasoning in AI applications such as planning systems and expert systems [44].

2. The analysis of the computational complexity of the verification process for the generalized temporal logic framework is an open problem. Due to the complexity of the verification process, the literature has avoided approaches that can handle nondeterminism. For example, the verification of the behavioral equivalence of two nondeterministic finite automata is *P*space-complete. Our conjecture is that the computational complexity of the verification process is *P*space-hard.

3. The existence of a controller, controllability, and optimal control problem. The relationship between controllability and reachability defined in this thesis should also be investigated. Moreover, the open problems for the optimal control problem of DESs include: The uniqueness of solutions to the optimal control problem of DESs; and the case of a set of final states instead of only one final state.

4. Extension of our temporal logic framework to describe the real-time features. There are two possible extensions of our temporal logic to deal with real-time aspects of DESs. The first is to add a global clock as an explicit variable to which the specification may refer as Ostroff did [85]. To do this, we need adapting the Insertion Rule into our proof system then adding universal quantifier to formulas containing free global variables. The syntax, semantics, and proof system of the language need to be modified. It is also possible to extend Ostroff's work to deal with the probabilistic features. The second way is to add metric operators with qualitative versions as Koymans did [46]. In this case, the syntax, semantics, and proof system of the language may have to be reconstructed.

5. Use of the internal events and external events to describe the system behavior. Some of events do not have any effect on the outcome of the system. These events can be named as internal events. On the contrary, external events can be used to name those events that come from outside as inputs to the system and those that do have effects on the outcome of the system. When we analyze within the system, both internal and external events are important. On the other hand,

when we investigate the communication between systems, it is a good idea to hide the internal events and deal only with the external events. Further classification of events could also be useful. For example, controlled events, spontaneous events, shared events, local events, simple events, component events, and so on.

The above suggestions for the future research also imply that a lot of work needed to be done for DESs in a temporal logic approach. We cannot hope for significant and durable advances in this approach without resting our theories on some firm foundations. This dissertation is offered as a step towards attaining such a steady basis, so that our theories may be well-defined and evaluable. Undoubtedly, further investigations in this approach will suggest advances in the framework developed in this dissertation. In the spirit of the dissertation, we look forward to seeing these advances in the near future.

Bibliography

[1] N.U. Ahmed, *Elements of Finite Dimensional System and Control Theory*, Pitman Monographs and Surveys in Pure and Applied Math., vol. 37, Longman Scientific and Technical, U.K. John Wiley, New York, 1988.

[2] J.F. Allen, "Towards a general theory of action and time," *Artificial Intelligence*, vol.23, pp.123-154, 1984.

[3] R. Aveyard, "A boolean model for a class of discrete event systems," *IEEE Trans. Syst. Man and Cyb.*, vol. SMC-4, pp.249-258, 1974.

[4] F. Baccelli and A.M. Makowski, "Queueing models for systems with synchronization constraints," *Proceedings of IEEE*, vol.77, no.1, pp.138-161, 1989.

[5] M. Baldassari and G. Bruno, "PROTOB: An object oriented methodology for developing discrete event dynamic systems", *Computer languages*, Vol.16, no.1, pp.39-63, 1991.

[6] J. Beauquier and M. Nivat, "Application of formal language theory to problems of security and synchronization", *Formal Laguage Theory*, ed. R.V. Book. Academic Press, New York. 1980. pp.407-454.

[7] R. Bellman, *Dynamic Programming*, Princeton Univ. Press, NJ, 1957.

[8] G. Birkhoff and J.D. Lipson, "Heterogeneous algebras", *Journal of Combinatorial Theory*, vol.8, pp.115-133, 1970.

[9] G.V. Bochmann, "Hardware specification with temporal logic: an example," *IEEE Trans. Computers*, vol.C-31, no.3, pp.223-231, 1982.

[10] M.C. Browne, E.M. Clarke, D.L. Dill, and B. Mishra, "Automatic verification of sequential circuits using temporal logic," *IEEE Trans. Computers*, vol.C-35, no.12, pp.1035-1044, 1986.

[11] B.C. Bruce, "A model for temporal references and its application in a question-answering program", *Artificial Intelligence*, vol.3, pp.1-25, 1972.

[12] B.A. Carre, "An algebra for network routing problems," *J. Inst. Math. Appl.*, vol.7, pp.273-294, 1971.

[13] C.G. Cassandras and S.G. Strickland, "On-line sensitivity analysis of Markov chains," *IEEE Trans. Automat. Contr.*, vol.AC-34, no.1, pp.76-86, 1989.

[14] R. Cieslak, C. Desclaux, A.S. Fawaz, and P. Varaiya, "Supervisory control of discrete-event processes with partial observations," *IEEE Trans. Automat. Contr.*, vol.33, no.3, pp.249-260, 1988.

[15] E.M. Clarke, E.A. Emerson, and A.P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. Programming Languages and Systems*, vol. 8, no.2, pp.244-263, 1986.

[16] G. Cohen, D. Dubois, J.P. Quadrat, and M. Viot, "A linear- system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing," *IEEE Trans. Automat. Contr.*, vol. AC-30, no.3, pp.210-220, 1985.

[17] R.A. Cuninghane Green, *Minimax Algebra*. Springer-Verlag, New York, 1979.

[18] J. De Kleer and J.S. Brown, "A qualitative physics based on confluences", *Artificial Intelligence*, vol.24, pp.7-83, 1984.

[19] T.L. Dean, "Using temporal logic hierarchies to efficiently maintain large temporal data base", *Journal of the Association for Computing Machinery*, vol.36, pp.687-718, 1989.

[20] T.L. Dean and D. McDermott, "Temporal data base management", *Artificial Intelligence*, vol.32, pp.1-55, 1987.

[21] M.J. Denham, "A Petri-net approach to the control of discrete-event systems," *Advanced Computing Concepts and Techniques in Control Engineering*, NATO ASI Series, Vol. F47, ed. M.J. Denham and A.J. Laub, Springer-Verly, Berlin, 1988, pp.191-214.

[22] DPN-100 Reference Handbook, Northern Telecom, 1992.

[23] H.A. Enderton, *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.

[24] G.W. Evans, G.F. Wallance, and G.L. Sutherland, *Simulation Using Digital Computers*, Prentice-Hall, Englewood Cliffs, 1967.

[25] W. Feller, *An Introduction to Probability Theory and Its Applications*, John Wiley, New York, 1957.

[26] K.D. Forbus, "Qualitative process theory", *Artificial Intelligence*, vol.24, pp.7-83, 1984.

[27] A. Fusaoka, H. Seki, and K. Takahashi, "A description and reasoning of plant controllers in temporal logic," *Proc. 8th Int. Joint Conf. on Artificial Intelligence*, pp.405-408, 1983.

[28] A. Galton, *Temporal Logics and Their Applications*, Academic Press, London, 1987.

[29] P.W. Glynn, "A GSMP formalism for discrete event systems", *Proceedings of the EEE*, vol.77, no.1, pp.14-23, 1989.

[30] C. Golaszewski and P.J. Ramadge, "Control of discrete event process with forced events", *Proc. 26th IEEE Conf. Decision & Control*, Los Angeles, California, Dec.9-11, 1987, pp.247-251.

[31] F. Halsall, *Data Communications, Computer Networks and OSI*, Addison-Wesley, New York, 1988.

[32] M. Hennessy, *Algebraic Theory of Processes*, The MIT press, Massachusetts, 1988.

[33] Y.C. Ho, "Performance evaluation and perturbation analysis of discrete event dynamical systems," *IEEE Trans. Automat. Contr.*, vol.AC-32, no.7, pp.563-572, 1987.

[34] Y.C. Ho, X.R. Cao, and C.G. Cassandras, "Infinitesimal and finite perturbation analysis for queueing networks" *Automatica*, vol.19, pp.439-445, 1983.

[35] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, New York, 1985.

[36] L. E. Hollowway and B.H. Krogh, "Efficient synthesis of control logic for a class of discrete event systems," *Proc. 1989 American Control Conf.*, pp.2672-2677, 1989.

[37] G.E. Hughes and M.J. Cresswell, *An Introduction to Modal Logic*, Methuen & Co, London, 1973.

[38] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*. McGraw-Hill, New York, 1984.

[39] A. Ichikawa and K. Hiraishi, "Analysis and control of discrete event systems represented by Petri nets," *Discrete Event Systems: Models and Applications*. ed. P. Varaiya and A.B. Kurzhanski. Springer-Verlag, New York, pp.115-134, 1987.

[40] K. Inan and P. Varaiya, "Finitely recursive process models for discrete event systems," *IEEE Trans. Automat. Contr.*, vol.33, no.7, pp.626-639, 1988.

[41] K. Inan and P. Varaiya, "Algebras of discrete event models," *Proceedings of IEEE*, vol.77, no.1, pp.24-38, 1989.

[42] D. Ionescu and J.-Y. Lin, "A supervisor design procedure for discrete event systems in a temporal logic framework with applications", to be presented at *IEEE Symposium on New Directions in Control Theory & Applications*, Crete, Greece, June 21-23,1993.

[43] D. Ionescu, J.-Y. Lin, and H.S. Hwang, "Specification of intelligent controllers for discrete event systems in a temporal logic framework", *Proceedings of IFAC Algorithms and Architectures for Real-Time Control*, Seoul, Korea, 1992, pp.237-242.

[44] P. Jackson, H. Reichgelt, and F. van Harmelen, *Logic-Based Knowledge Representation*, The MIT Press, Massachusetts, 1989.

[45] K. Kahn and G.A. Gorry, "Mechanizing temporal knowledge", *Artificial Intelligence*, vol.9, pp.87-108, 1977.

[46] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol.2, pp.255-299, 1990.

[47] R. Koymans, R.K. Shyamasundar, W.P. de Roever, R. Gerth, and S. Arun-Kumar, *Compositional Semantics for Real-time Distributed Computing*, LNCS 193, Springer-Verlag, June 1985.

[48] E. Kreyzig, *Introductory Functional Analysis with Applications*, John Wiley, New York, 1978.

[49] F. Kroger, *Temporal Logic of Programs*, Springer-Verlag, New York, 1987.

[50] B. Kuipers, "Qualitative simulation", *Artificial Intelligence*, vol.29, pp.289-338, 1986.

[51] B. Kuipers, "Qualitative reasoning modeling and simulation with incomplete knowledge", *Automatica*, vol.25, pp.571-585, 1989.

[52] L. Lamport, "The temporal logic of action", unpublished paper, 1991.

[53] D. Lehmann and S. Shelah, "Reasoning with time and chance," *Information and Control*, vol.53, pp.165-198, 1982.

[54] Y. Li and W.M. Wonham, "Controllability and observability in the state-feedback control of discrete-event systems," *Proc. 25th IEEE Conf. Decision & Control*, pp.203-208, 1988.

[55] F. Lin, "A note on optimal supervisory control", *Proc. 1991 IEEE Int. Sym. on Intelligent Control*, Arlington, pp.227-232, 1991

[56] F. Lin and W.M. Wonham, "On observability of discrete event systems", *Information Sciences*, vol.44, pp.173-198, 1988.

[57] J.-Y. Lin and D. Ionescu, "Output feedback control for a class of nondeterministic discrete event systems," *Proceedings of the 1990 American Control Conference*, San Diego, California, May 23-25, 1990, Vol. 1, pp.20-25.

[58] J.-Y. Lin and D. Ionescu, "Asymptotic behavior of a class of nondeterministic discrete event systems," *Proceedings of the 5th IEEE International Symposium on Intelligent Control*, Philadelphia, Pennsylvania, Sept.5-7, 1990, vol. 2, pp.993-998.

[59] J.-Y. Lin and D. Ionescu, "A temporal logic model for a class of nondeterministic discrete event systems", *Proceedings of the 1990 Canadian Conference on Electrical and Computer Engineering*, Ottawa, Ontario, September 3-6, 1990, Vol.1, pp. 2.4.1-2.4.4.

[60] J.-Y. Lin and D. Ionescu, "A generalized temporal logic approach for control problems of a class of nondeterministic discrete event systems," *Proceedings of the 29th IEEE Conference on Decision and Control*, Honolulu, Hawaii, Dec.5-7, 1990, pp.3440-3445.

[61] J.-Y. Lin and D. Ionescu, "Asymptotic behavior of output feedback for a class of nondeterministic discrete event systems," *International Journal of Control*, vol.44, no.4, pp.903-920, 1991.

[62] J.-Y. Lin and D. Ionescu, "A temporal logic approach to the control of discrete event systems", *Proceedings of the 1991 American Control Conference*, Boston, Massachusetts, June 26-28, 1991, Vol.3, pp.2934-2935.

[63] J.-Y. Lin and D. Ionescu, "Reasoning of discrete event systems in temporal logic," *Preprints of IFAC Int. Symposium on Distributed Intelligence Systems*, Arlington, Virginia, August 13-15, 1991, pp.43-48.

[64] J.-Y. Lin and D. Ionescu, "Artificial intelligence planning problems in a temporal logic framework", *Proceedings of 1991 Canadian Conference on Electrical and Computer Engineering*, Quebec, P.Q., Sept.25-27, 1991, pp. 41.3.1-41.3.4.

[65] J.-Y. Lin and D. Ionescu, "A controller approach for reasoning in a temporal logic framework," in *Proceedings of The World Congress on Expert Systems*, Orlando, Florida, Dec.16-19, 1991, Vol.3, pp.1609-1616.

[66] J.-Y. Lin and D. Ionescu, "Verifying a class of nondeterministic discrete event systems in a generalized temporal logic," *IEEE Transactions on Systems, Man and Cybernetics*, vol.22, no.6, pp.1461-1469, 1992.

[67] J.-Y. Lin and D. Ionescu, "Optimization of controller design for discrete event systems in a temporal logic framework," *Proceedings of the 1992 American Control Conference*, Chicago, Illinois, June 24-26, 1992, pp.2819-2823.

[68] J.-Y. Lin and D. Ionescu, "Analysis and synthesis procedures of discrete event systems in a temporal logic framework", *Proceedings of the 1992 IEEE International Symposium on Intelligent Control.*

[69] J.-Y. Lin and D. Ionescu, "A simulation language for discrete event systems in a temporal logic framework", *Proceedings of the 1992 Canadian Conference on Electrical and Computer Engineering*, Toronto, Sept.13-16, 1992, pp. MM6.3.1-MM6.3.4.

[70] J.-Y. Lin and D. Ionescu, "A temporal logic approach to computer communication process," *Workbook of the 4th International IEEE Workshop on Computer-Aided Modelling, Analysis and Design of Communication Links and Networks*, Montebello, Quebec, Sept. 29-Oct. 2, 1992.

[71] J.-Y. Lin and D. Ionescu, "A reachability synthesis procedure for discrete event systems in a temporal logic framework", submitted to *IEEE Transactions on Systems, Man, and Cybernetics.* (submitted on Sept.22, 92).

[72] Z. Manna and A. Pnueli, "Verification of concurrent programs: A temporal proof system," *Foundations of Computer Science IV*, Mathematical Centre Tracts 159, Mathematish Centrum, Amsterdam, pp.163-255, 1983.

[73] Z. Manna and A. Pnueli, "How to cook a temporal proof system for your pet language," *Proc. 10th Annual ACM Symp. on Principles of Programming Languages*, pp.141-154, 1983.

[74] Z. Manna and A. Pnueli, "Proving precedence properties: the temporal way", *Automata, Languages and Programming*, Lecture Notes in Computer Science 154, pp.491-512, 1983.

[75] Z. Manna, and P. Wolper, "Synthesis of communicating processes from temporal logic specifications," *ACM Trans. Programming Languages & Systems*, vol.6, no1, pp.68-93, 1984.

[76] A. Martelli, "On the search complexity of admissible search algorithms," *Artificial Intelligence*, vol.8, pp.1-13, 1977.

[77] D. McDermott, "A temporal logic for reasoning about processes and plans", *Cognitive Science*, vol.6, pp.101-155, 1982.

[78] G. Milne and R. Milner, "Concurrent processes and their syntax," *J. Assoc. Comput. Mach.*, vol.26, pp.302-321, 1979.

[79] H. Mortazavian and F. Lin, "Foundations of a logical theory of modeling and control of discrete event system," *Preprints of IFAC Int. Symposium on Distributed Intelligence Systems*, Arlington, Virginia, August 13-15, 1991, pp.19-24.

[80] B.C. Moszkowski, *Executing Temporal Logic Program*, Cambridge University Press, London, 1986.

[81] N.J. Nilsson, *Principles of Artificial Intelligence*. Palo Alto, California, 1982.

[82] G.J. Olsder, J.A.C. Resing, R.E. de Vries, M.S. Keance, and G. Hooghiemstra, "Discrete event systems with stochastic processing times." *Proc. 27th IEEE Conf. Decision & Control*, Austin, Texas, pp.214-219, 1988.

[83] G.J. Olsder and R.E. de Vries, "On an analogy of minimal realization in conventional and discrete-event dynamic systems," *Discrete Event Systems: Models and Applications*. ed. P. Varaiya and A.B. Kurzhanski. Springer-Verlag, New York, pp.149-161, 1987.

[84] J.S. Ostroff, "Deciding properties of timed transition models," *IEEE Trans. Parallel and Distributed Systems*, vol.1, pp.170-183, 1990.

[85] J.S. Ostroff, *Temporal Logic for Real-Time Systems*, John Wiley, New York, 1989.

[86] J.S. Ostroff, *Real-Time Computer Control of Discrete Event Systems Modelled by Extended State Machines: A Temporal Logic Approach*, Ph D. thesis, Dept. Electrical Engineering, University of Toronto, 1987.

[87] J.S. Ostroff and W.M. Wonham, "A framework for real-time discrete event control," *IEEE Trans. Automat. Control*, vol.35, pp.386-397, 1990.

[88] S.D. O'Young and W.M. Wonham, "Object-oriented computation and simulation of large-scale discrete-event systems", *Proceedings of the 27th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, Illinois, Sept.27-29, 1989, pp.945-954.

[89] K.M. Passino and P.J. Antsaklis, "On the optimal control of discrete event systems," *Proc. 28th IEEE Conf. Decision and Control*, pp.2713-2718, 1989.

[90] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, 1981.

[91] C.A. Petri, "Communication with automata", *Ph. D. dissertation*, Univ. Bonn, 1962, translated by C.F. Green, Applied Data Research Inc., Tech Report RADC-TR-65-377, 1965.

[92] L.J. Pinson and R.S. Wiener, *Objective-C: Object-Oriented Programming Techniques*. Addison-Wesley, New York, 1991.

[93] A. Pnueli, "The temporal semantics of concurrent programs," *Theoret. Comput. Sci.*, vol.13, pp.45-60, 1981.

[94] A. Pnueli and E. Harel, "Applications of temporal logic to the specification of real-time systems," *Proceedings of a Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, ed. M. Joseph. Lecture Notes in Computer Science, vol.331. Springer, Berlin. pp.84-98, 1988.

[95] P.J. Ramadge and W.M. Wonham, "Supervisory control of a class of discrete event process," *SIAM J. Contr. Optimiz.*, vol.25, no.1, pp.206-230, 1987.

[96] P.J. Ramadge and W.M Wonham, "Modular feedback logic for discrete event systems," *SIAM J. Contr. Optimiz.*, vol.25, no.5, pp.1202-1218, 1987.

[97] N. Rescher and A. Urquhart, *Temporal Logic*. Springer-Verlag, 1971.

[98] R. Righter and J.C. Walrand, "Distributed simulation of discrete event systems," *Proceedings of the IEEE*, vol.77, no.1, pp.99-113, 1989.

[99] R.L. Schwartz and P.M. Melliar-Smith, "From state machines to temporal logic: specification methods for protocol standards", *IEEE Transactions on Communications*, vol.COM-30, pp.2486-2496, 1982.

[100] R. Sengupta and S. Lafortune, "Optimal control of a class of discrete event systems", *Preprints of IFAC Int. Symposium on Distributed Intelligence Systems*, Arlington, Virginia, August 13-15, 1991, pp.25-30.

[101] A.C. Shaw, "Software descriptions with flow expressions", *IEEE Trans. Software Engineering*, vol. SE-4, pp.242-254, 1978.

[102] Y. Shoham, *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence*, The MIT Press, Massachusetts, 1988.

[103] Special Issue on Dynamics of Discrete Event Systems, *Proceedings of IEEE*, vol.77, no.1, 1989.

[104] D. Teichroew and J.F. Lubin, "Computer simulation: discussion of the technique and comparison of languages", *Commun. Ass. Comput. Mach.*, vol.IX, pp.723-741, 1966.

[105] J.G. Thistle and W.M. Wonham, "Control problems in a temporal logic framework," *Int. J. Control*, vol.44, pp.943-976, 1986.

[106] USA National Science Foundation, "Challenges to Control, A Collective View" (Rep. of the Workshop at Santa Clara Univ., Sept. 18-19, 1986), *IEEE Trans. Automat. Contr.* vol.AC-32, no.4, pp.275-285, 1987.

[107] J. Walrand, *Communication Network: A First Course*. IRWIN, Boston, 1991.

[108] A. Wilansky, *Functional Analysis*, Blaisdell Publishing Company, New York, 1964.

[109] G. Winskel, "Event structures," *Petri Nets: Applications and Relationships to Other Models of Concurrency*, ed. W. Brauger, W. Reisig and G. Rozenberg. Springer-Verly, Berlin, pp.325-392, 1987.

[110] W.M. Wonham, "A control theory for discrete-event systems," *Advanced Computing Concepts and Techniques in Control Engineering*, NATO ASI Series, Vol. F47, ed. M.J. Denham and A.J. Laub, Springer-Verly, Berlin, 1988, pp.129-169.

[111] W.M. Wonham, *Linear Multivariable Control: A Geometric Approach*, Springer-Verlag, New York, 1979.

[112] W.M. Wonham and P.J. Ramadge, "On the supremal controllable sublanguage of a given language", *SIAM J. on Control and Optimization*, vol.25, pp.637-659, 1987.

Appendix A

Outcomes of Simulation

In this appendix, we print out the outcomes of the three examples of simulation performed in Chapter 9: an example of logic evaluation, a gun shooting example to show the logic reasoning, and an example of a flexible manufacturing system to demonstrate the reachability analysis, the controller design, and the optimization.

A.1 An Example of Logic Evaluation

The following is the outcome on screen display of the simulation of the logic evaluation.

```
Would you like to use the logic evaluator?(y/n)y


Enter the Maxnum of facts:6


The factNum is 0
```

```
Do you want to enter a fact? (y/n)y

Enter a fact:   time 1 5 friend tom,cathy 1
factNum is 0

Do you want to enter a fact?(y/n)y

Enter a fact:   time 4 9 friend tom,smith 0
factNum is 1

Do you want to enter a fact?(y/n)y

Enter a fact:   time 2 9 has tom,car 1
factNum is 2

Do you want to enter a fact?(y/n)y

Enter a fact:   time 4 8 great x,y 1
factNum is 3

Do you want to enter a fact?(y/n)y

Enter a fact:   time 1 8 less y,z 1
factNum is 4
```

```
Do you want to enter a fact?(y/n)y

Enter a fact:   time 1 9 equal y,10 1
factNum is 5

Do you want to enter a fact?(y/n)n

Start to evaluate?(y/n)y

Enter the max-length of the statements.24

Do you want to evaluate a statement?(y/n)y

Enter the  statement.
( and ( time 2 5 ( friend tom,cathy ) )
      ( time 4 6 ( next ( has tom,car ) ) ) )


Statement is True.

To evaluate a statement without change Max-Len?(y/n)y

Enter the  statement.
```

```
( and ( time 2 5 ( friend tom,cathy ) )
      ( not ( time 4 6 ( has tom,car ) ) ) )


Statement is False.

To evaluate a statement without change Max-Len?(y/n)y

Enter the  statement.
( or ( time 5 8 ( friend tom,smith ) )
     ( time 1 3 ( next ( friend tom,cathy ) ) ) )


Statement is True.

To evaluate a statement without change Max-Len?(y/n)n

To evaluate with another Max-Len?(y/n)y

Enter the max-length of the statements.45

Do you want to evaluate a statement?(y/n)y

Enter the  statement.
```

```
( and ( or ( time 5 7 ( great x,y ) ) ( time 2 6 ( equal z,9 ) ) )
      ( and ( time 1 3 ( less y,z ) ) ( time 1 4 ( equal y,10 ) ) ) )


Statement is True.


To evaluate a statement without change Max-Len?(y/n)y


Enter the  statement.
( or ( and ( time 5 7 ( great x,y ) ) ( time 2 6 ( equal z,9 ) ) )
     ( or ( time 1 3 ( less y,z ) ) ( time 1 4 ( equal y,10 ) ) ) )


Statement is True.


To evaluate a statement without change Max-Len?(y/n)n


To evaluate with another Max-Len?(y/n)n


To make another logic evaluation?(y/n)n
```

A.2 A Gun Shooting Example

The following is the outcome on screen display of the simulation of the gun shooting example.

```
Would you like to use the logic evaluator?(y/n)n

Would you like to enter the rule file? (y/n n --> end)y

Please enter the file name gunrule.dat

Please enter the number of the rules5

Would you like to change to another file?(y/n n - continue)n

Please enter the number of the facts 6

Would you like to enter the fact file? (y/n n --> end)y

Please enter the file name gunfact.dat

Would you like to change to another file?(y/n n - continue)n

Would you like to get results and then stop?  y

Events:

 e1=(enfir 1)
 e2=(enfir 2)
```

```
e3=(enfir 3)

e4=(enfir 4)

e5=(efire 5)

Conclusions:

Process 1=(loadEd 1)

Process 2=(loadEd 2)

Process 3=(loadEd 3)

Process 4=(loadEd 4)

Process 5=(loadEd 5)

Process 6=(noisE 6)
```

A.3 An Example of Simulation

The following is the outcome on screen display of the simulation of the manufacturing system.

```
Would you like to use the logic evaluator?(y/n)n

Welcome to simulation of dynamics of systems

Please enter the specifications of the system
as the rules and the facts to two  files
```

following the instructions given in example.sho

Then enter dimension of the system (2 or)3

Would you like to enter the rule file? (y/n n --> end)y

Please enter the file name data2.dat

Please enter the number of the rules117

Would you like to change to another file?(y/n n - continue)n

Please enter the number of the facts120

Would you like to enter the fact file? (y/n n --> end)y

Please enter the file name fact2.dat

Would you like to change to another file?(y/n n - continue)n

dim=3

Please give specifications of the required behavior by rules

```
Would you like to enter the rule file? (y/n n --> end)y

Please enter the file name DesiRule.dat

Please enter the number of the rules6

Would you like to change to another file?(y/n n - continue)n

Would you like to get results and then stop?  n


 The state transitions of the uncontrolled system are:

 The root:   s0=(0,0,0)
 e1=(ecav 1)     --->  s1=(W,0,0)
 e2=(ecav 2)     --->  s2=(0,W,0)
 e3=(earr 3)     --->  s3=(0,0,0)=s0
 e4=(eara 3)     --->  s4=(0,0,1)

 The root:   s1=(W,0,0)
 e5=(ecsp 1)     --->  s5=(P,0,0)
 e6=(ecav 2)=e2 --->  s6=(W,W,0)
 e7=(earr 3)=e3 --->  s7=(W,0,0)=s1
```

```
e8=(eara 3)=e4 --->  s8=(W,0,1)

The root:   s2=(0,W,0)
e9=(ecav 1)=e1 --->  s9=(W,W,0)=s6
e10=(ecsp 2)    --->  s10=(0,P,0)
e11=(earr 3)=e3 --->  s11=(0,W,0)=s2
e12=(eara 3)=e4 --->  s12=(0,W,1)

The root:   s4=(0,0,1)
e13=(ecav 1)=e1 --->  s13=(W,0,1)=s8
e14=(ecav 2)=e2 --->  s14=(0,W,1)=s12
e15=(earr 3)=e3 --->  s15=(0,0,1)=s4
e16=(eara 3)=e4 --->  s16=(0,0,2)
e17=(eand 3)    --->  s17=(0,0,0)=s0

The root:   s5=(P,0,0)
e18=(ecnp 1)    --->  s18=(0,0,0)=s0
e19=(ecav 2)=e2 --->  s19=(P,W,0)
e20=(earr 3)=e3 --->  s20=(P,0,0)=s5
e21=(eara 3)=e4 --->  s21=(P,0,1)

The root:   s6=(W,W,0)
e22=(ecsp 1)=e5 --->  s22=(P,W,0)=s19
e23=(ecsp 2)=e10 --->  s23=(W,P,0)
```

```
e24=(earr 3)=e3 --->  s24=(W,W,0)=s6

e25=(eara 3)=e4 --->  s25=(W,W,1)


The root:   s8=(W,0,1)

e26=(ecsp 1)=e5 --->  s26=(P,0,1)=s21

e27=(ecav 2)=e2 --->  s27=(W,W,1)=s25

e28=(earr 3)=e3 --->  s28=(W,0,1)=s8

e29=(eara 3)=e4 --->  s29=(W,0,2)

e30=(eand 3)=e17 --->  s30=(W,0,0)=s1


The root:   s10=(0,P,0)

e31=(ecav 1)=e1 --->  s31=(W,P,0)=s23

e32=(ecnp 2)    --->  s32=(0,0,0)=s0

e33=(earr 3)=e3 --->  s33=(0,P,0)=s10

e34=(eara 3)=e4 --->  s34=(0,P,1)


The root:   s12=(0,W,1)

e35=(ecav 1)=e1 --->  s35=(W,W,1)=s25

e36=(ecsp 2)=e10 --->  s36=(0,P,1)=s34

e37=(earr 3)=e3 --->  s37=(0,W,1)=s12

e38=(eara 3)=e4 --->  s38=(0,W,2)

e39=(eand 3)=e17 --->  s39=(0,W,0)=s2


The root:   s16=(0,0,2)
```

e40=(ecav 1)=e1 ---> s40=(W,0,2)=s29

e41=(ecav 2)=e2 ---> s41=(0,W,2)=s38

e42=(earr 3)=e3 ---> s42=(0,0,2)=s16

e43=(eand 3)=e17 ---> s43=(0,0,1)=s4

The root: s19=(P,W,0)

e44=(ecnp 1)=e18 ---> s44=(0,W,0)=s2

e45=(ecsp 2)=e10 ---> s45=(P,P,0)

e46=(earr 3)=e3 ---> s46=(P,W,0)=s19

e47=(eara 3)=e4 ---> s47=(P,W,1)

The root: s21=(P,0,1)

e48=(ecnp 1)=e18 ---> s48=(0,0,1)=s4

e49=(ecav 2)=e2 ---> s49=(P,W,1)=s47

e50=(earr 3)=e3 ---> s50=(P,0,1)=s21

e51=(eara 3)=e4 ---> s51=(P,0,2)

e52=(eand 3)=e17 ---> s52=(P,0,0)=s5

The root: s23=(W,P,0)

e53=(ecsp 1)=e5 ---> s53=(P,P,0)=s45

e54=(ecnp 2)=e32 ---> s54=(W,0,0)=s1

e55=(earr 3)=e3 ---> s55=(W,P,0)=s23

e56=(eara 3)=e4 ---> s56=(W,P,1)

```
The root:   s25=(W,W,1)
e57=(ecsp 1)=e5 --->  s57=(P,W,1)=s47
e58=(ecsp 2)=e10 --->  s58=(W,P,1)=s56
e59=(earr 3)=e3 --->  s59=(W,W,1)=s25
e60=(eara 3)=e4 --->  s60=(W,W,2)
e61=(eand 3)=e17 --->  s61=(W,W,0)=s6

The root:   s29=(W,0,2)
e62=(ecsp 1)=e5 --->  s62=(P,0,2)=s51
e63=(ecav 2)=e2 --->  s63=(W,W,2)=s60
e64=(earr 3)=e3 --->  s64=(W,0,2)=s29
e65=(eand 3)=e17 --->  s65=(W,0,1)=s8

The root:   s34=(0,P,1)
e66=(ecav 1)=e1 --->  s66=(W,P,1)=s56
e67=(ecnp 2)=e32 --->  s67=(0,0,1)=s4
e68=(earr 3)=e3 --->  s68=(0,P,1)=s34
e69=(eara 3)=e4 --->  s69=(0,P,2)
e70=(eand 3)=e17 --->  s70=(0,P,0)=s10

The root:   s36=(0,W,2)
e71=(ecav 1)=e1 --->  s71=(W,W,2)=s60
e72=(ecsp 2)=e10 --->  s72=(0,P,2)=s69
e73=(earr 3)=e3 --->  s73=(0,W,2)=s38
```

```
e74=(eand 3)=e17 --->  s74=(0,W,1)=s12

The root:   s45=(P,P,0)
e75=(ecnp 1)=e18 --->  s75=(0,P,0)=s10
e76=(ecnp 2)=e32 --->  s76=(P,0,0)=s5
e77=(earr 3)=e3 --->  s77=(P,P,0)=s45
e78=(eara 3)=e4 --->  s78=(P,P,1)

The root:   s47=(P,W,1)
e79=(ecnp 1)=e18 --->  s79=(0,W,1)=s12
e80=(ecsp 2)=e10 --->  s80=(P,P,1)=s78
e81=(earr 3)=e3 --->  s81=(P,W,1)=s47
e82=(eara 3)=e4 --->  s82=(P,W,2)
e83=(eand 3)=e17 --->  s83=(P,W,0)=s19

The root:   s51=(P,0,2)
e84=(ecnp 1)=e18 --->  s84=(0,0,2)=s16
e85=(ecav 2)=e2 --->  s85=(P,W,2)=s82
e86=(earr 3)=e3 --->  s86=(P,0,2)=s51
e87=(eand 3)=e17 --->  s87=(P,0,1)=s21

The root:   s56=(W,P,1)
e88=(ecsp 1)=e5 --->  s88=(P,P,1)=s78
e89=(ecnp 2)=e32 --->  s89=(W,0,1)=s8
```

```
e90=(earr 3)=e3 --->  s90=(W,P,1)=s56

e91=(eara 3)=e4 --->  s91=(W,P,2)

e92=(eand 3)=e17 --->  s92=(W,P,0)=s23


The root:   s60=(W,W,2)

e93=(ecsp 1)=e5 --->  s93=(P,W,2)=s82

e94=(ecsp 2)=e10 --->  s94=(W,P,2)=s91

e95=(earr 3)=e3 --->  s95=(W,W,2)=s60

e96=(eand 3)=e17 --->  s96=(W,W,1)=s25


The root:   s69=(0,P,2)

e97=(ecav 1)=e1 --->  s97=(W,P,2)=s91

e98=(ecnp 2)=e32 --->  s98=(0,0,2)=s16

e99=(earr 3)=e3 --->  s99=(0,P,2)=s69

e100=(eand 3)=e17 --->  s100=(0,P,1)=s34


The root:   s78=(P,P,1)

e101=(ecnp 1)=e18 --->  s101=(0,P,1)=s34

e102=(ecnp 2)=e32 --->  s102=(P,0,1)=s21

e103=(earr 3)=e3 --->  s103=(P,P,1)=s78

e104=(eara 3)=e4 --->  s104=(P,P,2)

e105=(eand 3)=e17 --->  s105=(P,P,0)=s45


The root:   s82=(P,W,2)
```

```
e106=(ecnp 1)=e18 --->  s106=(0,W,2)=s38

e107=(ecsp 2)=e10 --->  s107=(P,P,2)=s104

e108=(earr 3)=e3 --->  s108=(P,W,2)=s82

e109=(eand 3)=e17 --->  s109=(P,W,1)=s47


The root:   s91=(W,P,2)

e110=(ecsp 1)=e5 --->  s110=(P,P,2)=s104

e111=(ecnp 2)=e32 --->  s111=(W,0,2)=s29

e112=(earr 3)=e3 --->  s112=(W,P,2)=s91

e113=(eand 3)=e17 --->  s113=(W,P,1)=s56


The root:   s104=(P,P,2)

e114=(ecnp 1)=e18 --->  s114=(0,P,2)=s69

e115=(ecnp 2)=e32 --->  s115=(P,0,2)=s51

e116=(earr 3)=e3 --->  s116=(P,P,2)=s104

e117=(eand 3)=e17 --->  s117=(P,P,1)=s78

According to the specifications of the required behavior


 Event e3 should be disabled at state s0 !

 Event e8 should be disabled at state s1 !

 Event e12 should be disabled at state s2 !

 Event e13 should be disabled at state s4 !

 Event e14 should be disabled at state s4 !

 Event e15 should be disabled at state s4 !
```

s21 should not be reached! Event e21 should be disabled at state s5 !

Event e25 should be disabled at state s6 !

s26=(P,O,1)=s21 So event e26 should be disabled at state s8 !

Event e27 should be disabled at state s8 !

Event e28 should be disabled at state s8 !

Event e29 should be disabled at state s8 !

Event e30 should be disabled at state s8 !

s34 should not be reached! Event e34 should be disabled at state s10 !

Event e35 should be disabled at state 12 !

s36=(O,P,1)=s34 So event e36 should be disabled at state s12 !

Event e37 should be disabled at state 12 !

Event e38 should be disabled at state 12 !

Event e39 should be disabled at state 12 !

Event e40 should be disabled at state 16 !

Event e41 should be disabled at state 16 !

Event e42 should be disabled at state 16 !

s45 should not be reached! Event e45 should be disabled at state s19 !

s47 should not be reached! Event e47 should be disabled at state s19 !

Event e48 should be disabled at state 21 !

s49=(P,W,1)=s47 So event e49 should be disabled at state s21 !

s50=(P,0,1)=s21 So event e50 should be disabled at state s21 !

s51 should not be reached! Event e51 should be disabled at state s21 !

Event e52 should be disabled at state 21 !

s53=(P,P,0)=s45 So event e53 should be disabled at state s23 !

s56 should not be reached! Event e56 should be disabled at state s23 !

s57=(P,W,1)=s47 So event e57 should be disabled at state s25 !

s58=(W,P,1)=s56 So event e58 should be disabled at state s25 !

Event e59 should be disabled at state 25 !

Event e60 should be disabled at state 25 !

Event e61 should be disabled at state 25 !

s62=(P,0,2)=s51 So event e62 should be disabled at state s29 !

Event e63 should be disabled at state 29 !

Event e64 should be disabled at state 29 !

Event e65 should be disabled at state 29 !

s66=(W,P,1)=s56 So event e66 should be disabled at state s34 !

Event e67 should be disabled at state 34 !

s68=(0,P,1)=s34 So event e68 should be disabled at state s34 !

s69 should not be reached! Event e69 should be disabled at state s34 !

Event e70 should be disabled at state 34 !

Event e71 should be disabled at state 38 !

s72=(0,P,2)=s69 So event e72 should be disabled at state s38 !

Event e73 should be disabled at state 38 !

Event e74 should be disabled at state 38 !

Event e75 should be disabled at state 45 !

Event e76 should be disabled at state 45 !

s77=(P,P,0)=s45 So event e77 should be disabled at state s45 !

s78 should not be reached! Event e78 should be disabled at state s45 !

Event e79 should be disabled at state 47 !

s80=(P,P,1)=s78 So event e80 should be disabled at state s47 !

s81=(P,W,1)=s47 So event e81 should be disabled at state s47 !

s82 should not be reached! Event e82 should be disabled at state s47 !

Event e83 should be disabled at state 47 !

Event e84 should be disabled at state 51 !

s85=(P,W,2)=s82 So event e85 should be disabled at state s51 !

s86=(P,0,2)=s51 So event e86 should be disabled at state s51 !

s87=(P,0,1)=s21 So event e87 should be disabled at state s51 !

s88=(P,P,1)=s78 So event e88 should be disabled at state s56 !

Event e89 should be disabled at state 56 !

s90=(W,P,1)=s56 So event e90 should be disabled at state s56 !

s91 should not be reached! Event e91 should be disabled at state s56 !

Event e92 should be disabled at state 56 !

s93=(P,W,2)=s82 So event e93 should be disabled at state s60 !

s94=(W,P,2)=s91 So event e94 should be disabled at state s60 !

Event e95 should be disabled at state 60 !

Event e96 should be disabled at state 60 !

s97=(W,P,2)=s91 So event e97 should be disabled at state s69 !

Event e98 should be disabled at state 69 !

s99=(O,P,2)=s69 So event e99 should be disabled at state s69 !

s100=(O,P,1)=s34 So event e100 should be disabled at state s69 !

s101=(O,P,1)=s34 So event e101 should be disabled at state s78 !

s102=(P,O,1)=s21 So event e102 should be disabled at state s78 !

s103 should not be reached! Event e103 should be disabled at state s78 !

s104=(P,P,2)=s103 So event e104 should be disabled at state s78 !

s105=(P,P,1)=s78 So event e105 should be disabled at state s78 !

Event e106 should be disabled at state 82 !

s107=(P,P,2)=s103 So event e107 should be disabled at state s82 !

s108=(P,W,2)=s82 So event e108 should be disabled at state s82 !

s109=(P,W,1)=s47 So event e109 should be disabled at state s82 !

Event e110 should be disabled at state 91 !

s111=(W,P,1)=s56 So event e111 should be disabled at state s91 !

Event e112 should be disabled at state 91 !

s112=(O,P,2)=s69 So event e112 should be disabled at state s103 !

s113=(P,P,2)=s103 So event e113 should be disabled at state s103 !

s114=(P,P,0)=s45 So event e114 should be disabled at state s103 !

s8 may not be a root!

```
s12 may not be a root!
s21 may not be a root!
s25 may not be a root!
s29 may not be a root!
s34 may not be a root!
s38 may not be a root!
s45 may not be a root!
s47 may not be a root!
s51 may not be a root!
s56 may not be a root!
s60 may not be a root!
s69 may not be a root!
s78 may not be a root!
s82 may not be a root!
s91 may not be a root!
s103 may not be a root!

Then the set of states which should not be reached are:

s21=(P,O,1) s34=(O,P,1) s45=(P,P,0) s47=(P,W,1) s51=(P,O,2) s56=(W,P,1)
s69=(O,P,2) s78=(P,P,1) s82=(P,W,2) s91=(W,P,2) s103=(P,P,2)

Now design a supervisor by giving rules and facts
following the instruction in example.sho
```

```
Then enter dimension of the supervisor (2 or  )2

Would you like to enter the rule file? (y/n n --> end)y

Please enter the file name manRule.dat

Please enter the number of the rules28

Would you like to change to another file?(y/n n - continue)n

Please enter the number of facts30

Would you like to enter the fact file? (y/n n --> end)y

Please enter the file name manFact.dat

Would you like to change to another file?(y/n n - continue)n

 cDim=2

Enter the max number of events in the event's sets 5
```

The transitions in the controller are:

The root: C0 =(E,0)

e1=(ecav 1) ---> C1 =(a,0)

e2=(ecav 2) ---> C2 =(a,0) = C1

e3=(eara 3) ---> C3 =(E,1)

The root: C1 =(a,0)

e4=(ecav 1) ---> C4 =(ab,0)

e5=(ecav 2) ---> C5 =(ab,0) = C4

e6=(ecsp 1) ---> C6 =(A,0)

e7=(ecsp 2) ---> C7 =(A,0) = C6

e8=(earr 3) ---> C8 =(a,0) = C1

The root: C3 =(E,1)

e9=(eand 3) ---> C9 =(E,0) = C0

e10=(eara 3) ---> C10 =(E,2)

The root: C4 =(ab,0)

e11=(earr 3) ---> C11 =(ab,0) = C4

e12=(ecsp 1) ---> C12 =(Ab,0)

e13=(ecsp 2) ---> C13 =(Ab,0) = C12

The root: C6 =(A,0)

e14=(earr 3) ---> C14 =(A,0) = C6

e15=(ecnp 1) ---> C15 =(E,0) = C0

e16=(ecnp 2) ---> C16 =(E,0) = C0

```
e17=(ecav 1)  --->   C17 =(Ab,0) = C12

e18=(ecav 2)  --->   C18 =(Ab,0) = C12

The root:   C10 =(E,2)

e19=(eand 3)  --->   C19 =(E,1) = C3

The root:   C12 =(Ab,0)

e20=(earr 3)  --->   C20 =(Ab,0) = C12

e21=(ecnp 1)  --->   C21 =(b,0)

e22=(ecnp 2)  --->   C22 =(b,0) = C21

The root:   C21 =(b,0)

e23=(ecsp 1)  --->   C23 =(B,0)

e24=(ecsp 2)  --->   C24 =(B,0) = C23

e25=(earr 3)  --->   C25 =(b,0) = C21

The root:   C23 =(B,0)

e26=(ecnp 1)  --->   C26 =(E,0) = C0

e27=(ecnp 2)  --->   C27 =(E,0) = C0

e28=(earr 3)  --->   C28 =(B,0) = C23
```

Now the state transitions of the closed-loop system are:

The dimension of the closedloop system is 5

```
The root:   S0 =(0,0,0,E,0)

e1=(ecav 1)     --->  S1 =(W,0,0,a,0)

e2=(ecav 2)     --->  S2 =(0,W,0,a,0)
```

```
e4=(eara 3)     --->  S4 =(0,0,1,E,1)

The root:   S1 =(W,0,0,a,0)
e5=(ecsp 1)     --->  S5 =(P,0,0,A,0)
e6=(ecav 2)=e2 --->  S6 =(W,W,0,ab,0)
e7=(earr 3)=e3 --->  S7 =(W,0,0,a,0)=S1

The root:   S2 =(0,W,0,a,0)
e9=(ecav 1)=e1 --->  S9 =(W,W,0,ab,0)=S6
e10=(ecsp 2)    --->  S10 =(0,P,0,A,0)
e11=(earr 3)=e3 --->  S11 =(0,W,0,a,0)=S2

The root:   S4 =(0,0,1,E,1)
e16=(eara 3)=e4 --->  S16 =(0,0,2,E,2)
e17=(eand 3)    --->  S17 =(0,0,0,E,0)=S0

The root:   S5 =(P,0,0,A,0)
e18=(ecnp 1)    --->  S18 =(0,0,0,E,0)=S0
e19=(ecav 2)=e2 --->  S19 =(P,W,0,Ab,0)
e20=(earr 3)=e3 --->  S20 =(P,0,0,A,0)=S5

The root:   S6 =(W,W,0,ab,0)
e22=(ecsp 1)=e5 --->  S22 =(P,W,0,Ab,0)=S19
e23=(ecsp 2)=e10 --->  S23 =(W,P,0,Ab,0)
```

```
e24=(earr 3)=e3 --->  S24 =(W,W,0,ab,0)=S6

The root:   S10 =(0,P,0,A,0)
e31=(ecav 1)=e1 --->  S31 =(W,P,0,Ab,0)=S23
e32=(ecnp 2)    --->  S32 =(0,0,0,E,0)=S0
e33=(earr 3)=e3 --->  S33 =(0,P,0,A,0)=S10

The root:   S16 =(0,0,2,E,2)
e43=(eand 3)=e17 --->  S43 =(0,0,1,E,1)=S4

The root:   S19 =(P,W,0,Ab,0)
e44=(ecnp 1)=e18 --->  S44 =(0,W,0,b,0)
e46=(earr 3)=e3 --->  S46 =(P,W,0,Ab,0)=S19

The root:   S23 =(W,P,0,Ab,0)
e54=(ecnp 2)=e32 --->  S54 =(W,0,0,b,0)
e55=(earr 3)=e3 --->  S55 =(W,P,0,Ab,0)=S23

The root:   S44 =(0,W,0,b,0)
e10=(ecsp 2)=e10 --->  S75 =(0,P,0,B,0)
e11=(earr 3)=e3 --->  S76 =(0,W,0,b,0)=S44

The root:   S54 =(W,0,0,b,0)
e5=(ecsp 1)=e5 --->  S88 =(P,0,0,B,0)
```

```
e7=(earr 3)=e3 --->  S89 =(W,0,0,b,0)=S54

The root:   S75 =(0,P,0,B,0)
e32=(ecnp 2)=e32 --->  S101 =(0,0,0,E,0)=S0
e33=(earr 3)=e3 --->  S102 =(0,P,0,B,0)=S75

The root:   S88 =(P,0,0,B,0)
e18=(ecnp 1)=e18 --->  S110 =(0,0,0,E,0)=S0
e20=(earr 3)=e3 --->  S111 =(P,0,0,B,0)=S88

The Reachability Set:
S0 =(0,0,0,E,0) S1 =(W,0,0,a,0) S2 =(0,W,0,a,0) S4 =(0,0,1,E,1)
S5 =(P,0,0,A,0) S6 =(W,W,0,ab,0) S10 =(0,P,0,A,0) S16 =(0,0,2,E,2)
S19 =(P,W,0,Ab,0) S23 =(W,P,0,Ab,0) S44 =(0,W,0,b,0) S54 =(W,0,0,b,0)
S75 =(0,P,0,B,0) S88 =(P,0,0,B,0)


Please enter the number of the costs 9

Would you like to enter the fact file? (y/n n --> end)y

Please enter the file name manCost.dat

Would you like to change to another file?(y/n n - continue)n
```

```
Please enter the number of the hints 14

Would you like to enter the fact file? (y/n n --> end)y

Please enter the file name manHint.dat

Would you like to change to another file?(y/n n - continue)n

Would you like to make optimization? (y/n n --> end)y

Please enter the start node: S0

Please enter the end node: S19
 S0 =(O,O,0,E,0) S19 =(P,W,0,Ab,0)
 The optimal path

 (in backward order):

S19 f=7  g=4 e19

S5 f=3  g=2 e5

S1 f=2  g=1 e1
```

```
S0 f=0  g=0 (null)

End of optimization!

Would you like to make another path? (y/n n --> end)y

Please enter the start node: S10

Please enter the end node: S6
 S10 =(0,P,0,A,0) S6 =(W,W,0,ab,0)
 The optimal path

 (in backward order):

S6 f=10  g=8 e6

S1 f=7  g=6 e1

S0 f=5  g=5 e32

S10 f=3  g=0 (null)

End of optimization!
```

```
Would you like to make another path? (y/n n --> end)y

Please enter the start node: S0

Please enter the end node: S54
 S0 =(0,0,0,E,0) S54 =(W,0,0,b,0)
 The optimal path

 (in backward order):

S54 f=13  g=9 e54

S23 f=7  g=4 e23

S6 f=5  g=3 e6

S1 f=2  g=1 e1

S0 f=0  g=0 (null)

End of optimization!

Would you like to make another path? (y/n n --> end)y
```

```
Please enter the start node: S16

Please enter the end node: S19
 S16 =(0,0,2,E,2) S19 =(P,W,0,Ab,0)
 The optimal path

 (in backward order):

S19 f=11  g=8 e19

S5 f=7  g=6 e5

S1 f=6  g=5 e1

S0 f=4  g=4 e17

S4 f=4  g=2 e43

S16 f=3  g=0 (null)

End of optimization!

Would you like to make another path? (y/n n --> end)y
```

```
Please enter the start node: S23

Please enter the end node: S88
 S23 =(W,P,0,Ab,0) S88 =(P,0,0,B,0)
 The optimal path

 (in backward order):

S88 f=11  g=6 e88

S54 f=9  g=5 e54

S23 f=3  g=0 (null)

End of optimization!

Would you like to make another path? (y/n n --> end)n

 Good Bye!!
```