Quantitative Evaluation of Blocking-Recovery Discrete Modular Agents

FRANCISCO PALOMERA PALACIOS

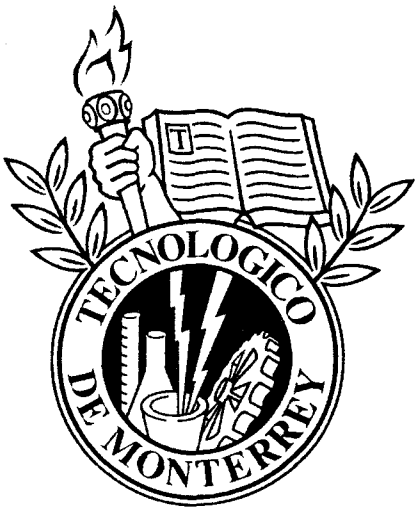
Ph.D. DISSERTATION

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

DECEMBER 2006

Quantitative Evaluation of Blocking-Recovery Discrete Modular Agents

FRANCISCO PALOMERA PALACIOS



Ph.D. DISSERTATION

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY

December 2006

Quantitative Evaluation of Blocking-Recovery Discrete Modular Agents

A Dissertation Presented by

Francisco Palomera Palacios

Submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy
in the field of
Artificial Intelligence

Thesis Committee:

Rogelio Soto Rodríguez, ITESM Campus Monterrey
Jorge Limón Robles, ITESM Campus Monterrey
Ramón Brena Pinero, ITESM Campus Monterrey
Ricardo A. Ramírez Mendoza, ITESM Campus Monterrey
Max Hering de Queiroz, UFSC-Brazil

Center for Intelligent Systems
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey
December 2006

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

Graduate Program in
Information Technology and Electronics

The committee members hereby recommend the dissertation presented by Francisco Palomera Palacios to be accepted as a partial fulfillment of requirements to be admitted to the **Degree of Doctor of Philosophy in Artificial Intelligence**.

Committee members:

Dr. Rogelio Soto Rodríguez
Advisor

Dr. Jorge Limón Robles
Co-Advisor

Dr. Ramón Brena Pinero

Dr. Ricardo A. Ramírez Mendoza

Dr. Max Hering de Queiroz

Dr. Graciano Dieck Assad.
Director of Graduate Programs in
Information Technology and Electronics

Declaration

I hereby declare that I composed this dissertation entirely by myself and that it describes my own research.

Francisco Palomera Palacios
Monterrey, N.L., México
December 2006

Quantitative Evaluation of Blocking-Recovery Discrete Modular Agents

By:
Francisco Palomera (PhD student)
Dr. Rogelio Soto (advisor) and Dr. Jorge Limón (co-advisor)

Abstract

Modern manufacturing production environments, made up of a set of independent machines, execute high production plans trying to develop the full potential of the machines, but still keeping a safety operation. The synthesis of control entities (supervisors, controllers, agents) requieres a quantitative evaluation rather than just a qualitative one in order to measure its performance. Quantitative measurements of evaluation are defined in terms of production parameters or production conditions to achieve a desired production plan. Since control entities rule the processing machines for a high production system, an undesirable condition named *blocking* (dead-lock) may occur. If so, instead of rejecting such behavior, that at first sight seems to be problematic, we should evaluate how convenient is to use a blocking recovery procedure (use of an un-blocking mechanism) and move to a free-blocking production condition in order to re-initialiaze and resume a production process.

This thesis work provides three complementary methods to deal with blocking- occurrence and evaluates, quantitatively, its blocking-recovery. To illustrate and show the proposed methods, we control a manufacturing transfer line that includes re-processing work, which feeds-back the system. The methods are developed to evaluate the performance of the transfer line in transient and steady-state analysis.

The *first method* helps to cope up with dead-lock occurrence by synthesizing a set of blocking-recovery discrete-modular agents (BRDMA) along with an un-blocking mechanism (UBM). The *second method* provides a procedure to compute the time to block, t_b, as a quantitative evaluation metric of the controlled transfer line, following two different approaches. The *third method* proposes a way to evaluate quantitatively and make a comparison of the *profit rate* between the best blocking recovery policy and its counter part of the free-blocking solution, both in steady-state. To show our experimental results, we provide state-graphs, plots and numerical results to synthesize and evaluate quantitatively the performance of the control entities named as Blocking-Recovery Discrete Modular Agents (BRDMA), under transient and steady-state analysis. Theory from discrete-event systems, agents and multiagents, discrete events and supervisory control theory, automated manufacturing systems and continuous time Markov chains provide the support of this thesis work.

Acknowledgments

I would like to express my gratitude to all members of my thesis's committee for their real commitment to lead me until the end of this work.

To: my advisor *Dr. Rogelio Soto* to give me the confidence, freedom and support to develop this thesis work, as well as his patience and motivation to face and complete the whole process, and finally achieve my own goal.

To: my co-advisor *Dr. Jorge Limón* for his unvaluable ideas and comments about CTMC and the time devoted to discuss and feedback this work.

To: *Dr. Walter Murray Wonham* for his guidence, assistance and support to introduce me into DES and SCT theory, sharing his time and knowledge, and over all, for his friendship in my stay into his Control Group at University of Toronto.

To: *Dr. Max Hering de Queiroz* to share his experiences about DES and SCT applications as well as pushing me to complete this work, and for his friendship.

To: *Dr. Ramón Brena* for his great suggestions and discussions about Multiagents (MAS) as

a very important topic in this thesis.

To: *Dr. Ricardo Ramírez* for his motivation and feedback about this work.

To: *Dr. Hugo Terashima* Coordinator of the PhD Program in Artificial Intelligence for all his support while I was enrolled in this graduate program.

To: *ITESM Campus Monterrey* for the support given in my full academic graduate program.

Dedication

To: *God*, for giving me the opportunity to acknowledge all my weaknesses and use them as strengths through this journey.

To: my wife *Gabriela*, for her motivation, encouragement, great support and for helping me to overcome the hard times we faced during this journey.

To: my children: *Gaby, Paco and Emmanuel* for their understanding to join me in this journey.

To: my parents *Maria Luisa and Ezequiel* and my grandmother *Luisa* because their words have always been a pray for me.

To: my *siblings*, thanks for their cheers and motivation through these last years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

In modern manufacturing systems like in other production systems there is still a need to synthesize *control entities* to make independent machines develop their full potential into a production plan as well as evaluate them quantitatively [28]. These control entities will help us to achieve high production levels in short periods of time, within systems with low and medium production levels. Although, there may be a price to pay, such as *blocking (dead-lock)*, however, the synthesis of reliable blocking-recovery control entities, along with the use of an un-blocking mechanism, can provide a nice performance to resolve blocking.If so, a quantitative and qualitative comparison must be done between a blocking-recovery production plan versus a nonblocking plan. Two main ideas of this work are: i) do not reject the synthesis of control entities which generate blocking occurrence without a previous quantitative evaluation, and, ii) perform a quantitative evaluation and comparison between the best blocking-recovery policy and its counter-part nonblocking solution. The synthesis of control entities named as *blocking-recovery discrete modular agents* will help us to resolve not only blocking, but also, to provide a state-based structure to evaluate the different un-blocking policies in transient and steady-sate analysis.

1.2 The Domain (state of the art)

As stated in [9, 11, 22, 30, 41, 42] the main approaches to deal with *blocking (dead-lock)* in different production systems are: *blocking-avoidance*, *blocking-recovery* and *blocking-prevention. Blocking avoidance* is the most conservative production approach that generates a low production level. On the other hand, *blocking-recovery* seems to be the most appropriate approach to maintain a high level of production. However, in spite of big efforts and great contributions provided to deal with deadlock, current approaches seem to be great solutions, but at the same time, not suitable, or too complicated, or even expensive approaches for small and medium size manufacturing systems. However, we can provide a *simple blocking-recovery* approach to deal with blocking, through the synthesis of what we will call as *blocking-recovery discrete modular agents* [28, 29]. Main ideas about how agents work, in a multiagent system, are useful to synthesize a set of control entities to resolve blocking. Some of the taken ideas are how to de-

compose a main task into a set of sub-tasks, as well as how agents take control of such sub-tasks [23, 28, 38]. These control entities will try to achieve main system's goal, as well as develop the full potential of all independent machines within a manufacturing system. All those characteristics can be brought together to be developed as discrete modular agents based on discrete events systems and supervisory control theory, both from Ramadge-Wonham framework, in a state-based modeling. Although, most approaches to evaluate automated manufacturing systems (AMS) are carried out by simple state-based model representation for a desired behavior, these models do not show or provide a realistic evolution or behavior a system may follow. However, we can perform the synthesis of discrete state-based control entities, and from their state-based structure, we can acquire a more realistic evolution and evaluation of the whole production system as a controlled system. Other issues about blocking and blocking recovery are found in [1, 10, 16, 24, 47]. A relation between agents as discrete supervisors and multiagents robots can be found in [19, 20, 35, 36].

1.3 Proposed Solution

Three complementary methods are proposed to synthesize and evaluate, quantitatively, the performance of a set of blocking-recovery discrete modular agents taking control of a manufacturing transfer line. The *first method* is concerned about how to perform the synthesis of a set of blocking-recovery discrete modular agents (BRDMA) running along with an unblocking mechanism, both developed under Ramadge-Wonham framework [28]. These discrete agents generate different un-blocking policies to be evaluated. The *second method* helps us to calculate the time to block, t_b, as a quantitative evaluation metric, from a cumulative distribution function (cdf) and from the coefficients of a statistical moment generator, both for and from different re-initialization states (un-blocking policies). This evaluation is performed from the state-based representation of the BRDMA obtained in the first method. The *third method* provide a procedure to evaluate quantitatively the *profit-rate* that a blocking-recovery policy may generate and it makes a comparison versus its nonblocking counter-part profit-rate. This evaluation is performed in steady-state analysis.

The three proposed methods rely on a state-based representation to apply continuous time Markov-chains (CTMC) to perform a transient and steady-state time analysis. We formulate a set of first-order differential equations named as *balance equations*, to perform the transient and steady-state analysis, for the absorbing (deadlock) and non-absorbing states. In order to apply the proposed methods, we use three cases of study named as TLB1B1, TLB2B1 and TLB3B1. For each case, we perform the synthesis of a set of blocking- recovery discrete modular agents (BRDMA), and we evaluate them by changing some production parameters such as processing times for the machines, the size of one buffer, and the failure probability in the test unit.

Some math-programs were developed in ©Maple to evaluate the performance of the transfer line and obtain all the plots to analyze the system's time behavior, in both, in transient and steady-state time analysis, for each case of study and under different production conditions.

1.4 Overview of the thesis

In chapter 1 we provide the framework of this work. In chapter 2 we present the fundamental theory from different fields of science that support this work. In chapter 3 we present the first method to perform the synthesis of blocking-recovery discrete modular agents (BRDMA) and its state-based structure. In Chapter 4 we present the evaluation methods to calculate the blocking time, t_b, in transient time analysis, and the evaluation of their steady-state probabilities. The experimental results are presented in Chapter 5 by developing the *profit rate evaluation method.* The conclusions and future research are presented in Chapter 6.

1.5 Thesis Contributions

We can point out that the main contributions of this thesis work are focus on proposing a set of complementary methods to face, solve and evaluate quantitatively the blocking-recovery into discrete event systems such as a manufacturing transfer line among these kind of systems. Additional contributions are: provide a state-based structure from the set of control entities (blocking recovery discrete modular agents) that show a realistic state evolution of a discrete event system; an evaluation structure where we replace the events by transient rates to perform an evaluation in transient and steady-state analysis. Also, we provide two approaches to measure and evaluate the time to block, t_b, from different re-initialization states, and finally, we present a profit rate method to evaluate quantitatively a blocking-recovery production plan and make a comparison of the profit rate of its counter-part nonblocking solutions.

Chapter 2

Theoretical Framework

To analyze and evaluate performance of modern man-made systems, such as manufacturing systems, is not an easy task, due to their complex structure and operation. To, there is a need to joint strengths from different fields of knowledge to evaluate such performance. Subjects from different fields of knowledge posses strengths that make them suitable to contribute to evaluate performances either under a theoretical or a real application framework. Modern manufacturing systems such complexities For instance, for such kind of man-made systems, it is neither easy to find a tools to perform modeling, design (synthesis), evaluation, and computation for a qualitative evaluation, nor for a quantitative evaluation.

In this section we include the theory and concepts from different areas that encompass this thesis work. Main areas are: Multiagent Systems (**MAS**), Discrete Event Systems (**DES**) and Supervisory Control Theory (**SCT**), both under Ramadge-Wonham framework, Automated Manufacturing Systems (**AMS**), Continuous Time Markov Chains (**CTMC**), and Probability Theory (**PT**).

We include concepts and theory from the above areas that support the ideas of this thesis work, without trying to provide an exhaustive review of each area.

2.1 Agents and Multiagent Systems (MASs)

The knowledge of theory and concepts from *agents* and *multiagents* have mainly been devoted to computational applications. However, some characteristics that agents posses, how they act, how they behave, how their structures are, how they interact to solve conflicts, may also be used to develop other *entities* based on software, hardware or both, with new characteristics or capabilities. So, agents and multiagents may provide source of inspiration or as reference models to build other similar tools, or they may look much alike to others current tools. From this viewpoint, solution to problems of different degree of complexity can be resolved by applying the frameworks of agents and multiagent systems, as long as they become a suitable tool to ease a task, rather than turn it in a complicated task, for specific application or problem solving.

A more exhaustive theory and concepts about agents and multiagents can be found in [15, 31, 37, 44].

Definition 2.1 *In [37] an agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors. The knowledge base contains not only what the agent knows about the environment he interacts with, but also, what to do (react) over the environment after processing input information, as in Fig. 2.1.*

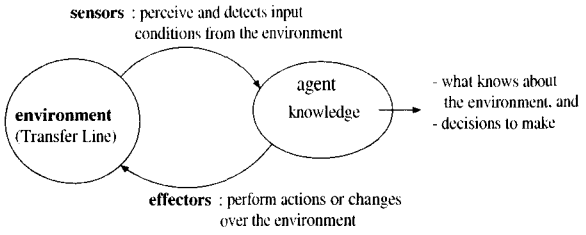


Figure 2.1: An agent basic structure

As defined in [15] an *agent* is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives. Other definition found about agents may be that, they are considered as "entities based on software, hardware, or both". However since no unique definition has been adopted, all of them provide ideas to clarify about what an agent is or it can do.

In [38] a list of desirable characteristic for an agent is provided, and a short description of them is given for each characteristic.

2.1.1 Agent-based Systems Characteristics

According to [38] the internal architecture of an agent is essentially the description of its modules and how they work together. Agent-based systems vary form the very simple (a single function control unit with an input and an output) to the very complex, even human like. Also, for an agent-based application some required agent's characteristics must be defined in order to represent them as modules, and define, how such characteristics would be implemented. It is also pointed out that, for an agent-based system, only a sub-set of such characteristics are needed. A list of some basic modules (components) for agents is also provided and their features are described, as well as remarking that, in an agent-based system, it may only be composed of a subset of such given list of modules (characteristics). A few of the given characteristics are rewritten to be related with this thesis work.

Reasoning: a reasoning module is required to make decisions according to its knowledge.

Control: this module controls the information flow mapping of the agent's goals into operational actions, and even monitoring the execution of actions.

Conflict management: enables the agent to make decisions when confronted with contradictory information derived or received from other agents. In most agent-based systems, conflict detection and resolution is done at the system level (or multiagent systems), and in some cases, with *human interventions.*

2.1.2 Agents Architecture

In [38] different agent's architectures have been described in the literature for agent-based concurrent design and manufacturing systems. Two types of classification are given: by *behavior* and by *internal organization.* Only the first one is shortly described, and the second one can be reviewed in [38].

Within a classification by *behavior*, agent's architectures may be classified into the following four categories: *deliberative, reactive, collaborative, and hybrid* architectures.

In *reactive architectures, reactive agents* respond in an event-condition-action mode. They dont́ have internal models of the world. They respond solely to external stimuli and the information available from their sensing of the environment. Also, various architectures have been described for reactive agents. The simplest one is provided by a finite number of rules which relates perceptions to actions an provides a means to describe reactions to situations. More details about other types of architectures and comparison of them can be found in [38].

2.1.3 Agents Systems Architecture

In [38] several typical architectures that have been widely used, especially for agent-based concurrent design and manufacturing systems, are described. Agent-based architectures are classified into three categories: *hierarchical, federated and autonomous* agent system architecture, where a description for each architecture is provided, too.

Complex multiagent architectures such as INTERRAP [26] and others mentioned in [38] show how complex an architecture for a multiagent system (MAS) can be, depending on the size and the matters to be dealing with by a MAS.

2.1.4 Agents Interactions

Agents' interactions are needed between agents when they want to communicate due to request or send of information, need of help, conflict resolution, negotiation, among other reasons. In complex or sophisticated tasks [18] a protocol is defined and remains within each agent to establish the structure of messages they can exchange. If resource allocation or task decomposition and allocation features need to be done, high level hierarchies need to be used. However, the way these interactions and allocations are designed and executed into a given system, can generate ideas to inspire simple mechanisms for conflict resolution.

2.2 Multiagent Systems (MAS)

In [44] two primary types of Distributed Artificial Intelligence(DAI) systems have been distinguished: *multiagent systems* and *distributed problem solving.* Initially, multiagent systems were pointed out on *behavior coordination*, whereas distributed problem solving was on *task decomposition and solution synthesis.* Also, modern concepts of multiagent systems covers both types of systems. In [15] individual intelligent agents can perform useful functions, but the possibilities start when consider teams of intelligent agents working together. Also, it

is stated that the study of artificial intelligence has been inspired by attempts to mimic the logical reasoning of a human. In the same way, a branch of AI known as *distributed artificial intelligence*(DAI) has been inspired by attempts to mimic a society of humans working together. *Multiagent systems* (MASs), sometimes called *agent – oriented* or *agent – based* systems, are one of the most important approaches to DAI; *blackboard* systems are another.

Definition 2.2 *From [15], a multiagent system (MAS) can be defined as a system in which several interacting, intelligent agents pursue a set of individually held goals or perform a set of individual tasks.*

In [44] some major characteristics of multiagent systems are identified:

a) each agent has just incomplete information and is restricted in its capabilities;

b) system control is distributed;

c) data is decentralized; and

d) computation is asynchronous.

Benefits of a multiagent system

In [15] there are some problems for which multiagent systems offer the only practical approach:

1. *Inherently complex problems* : such problems are simply too large to be solved by single hardware or software system.
2. *Inherently distributed problems*: Here the data and the information may exist in different physical locations, or at different times, or may be clustered into groups requiring different processing methods or semantics. These types of problems require a distributed solution, which can be provided by *agents running concurrently*, each with its own thread of control.

Furthermore, a multiagent system (MAS) offers the following general benefits:

1. A more natural *view of intelligence.*
2. *Speed and efficiency gains*, brought about because *agents can function concurrently* and communicate asynchronously.
3. *Robustness and reliability*: No agent is vital provided there are others that can take over its role in the event of its failure. Thus, the performance of an MAS will degrade gracefully if individual agents fail.
4. *Scalability*: DAI systems can generally be scaled-up simple by adding additional components. In the case of an MAS, additional agents can be added without adversely affecting those already present.

5. *Granularity*: Agents can be designed to operate at an appropriate level of detail. Many "fine grained" agents may be required to work on the minutiae of a problem, where each agent deals with a separate detail. At the same time, a few "coarse-grained", more sophisticated agents can concentrate on higher-level strategy.

6. *Ease of development*: encapsulation enables individual agents to be developed separately and to be re-used wherever applicable.

7. *Cost*: a system comprising many small processing agents is likely to be cheaper than a large centralized system.

2.2.1 Building a Multiagent System

In [15] a *multiagent system* is dependent on interactions between intelligent agents. There are, therefore, some key design decisions to be made, e.g., when, how and to whom should agents interact with? In *cooperative models*, several agents try to combine their efforts to accomplish as a group what the individuals cannot. In *competitive models* , each agent tries to get what only some of them can have. In either type of model, agents are generally assumed to be honest.

Multiagent systems are often designed as computer models of human function roles, such as may exist in a team-based organization. In [15] three models for managing agent interaction are mentioned and described. Those are: *contract nets*, *cooperative problem solving (CSP)*, and *shifting matrix management* (SMM). In each model communication between agents is required. Different protocols, as described in [18] [38], are used to structure the messages they exchange. In [18] a formal framework based on the theory of computing languages is presented for interaction protocols in multiagent system, as illustrated in figs. 2.2 and 2.3.

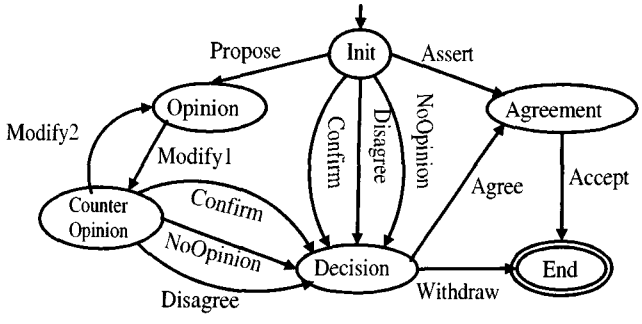


Figure 2.2: State-based agents' conversation

2.3 Discrete Event Systems (DES) and Supervisory Control Theory (SCT)

Most man-made systems show a discrete behavior in the way they work [3, 5, 6]. Manufacturing systems, network communications, and computer systems, are examples of such man-made

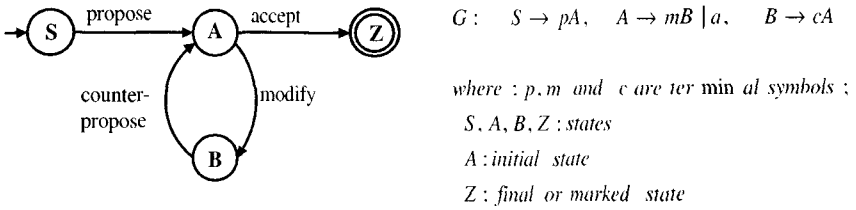


Figure 2.3: Simple agents' conversation based on a language

discrete systems. Their representation for modeling, analysis, synthesis and evaluation can be done by applying math, graph and computational tools that are able to handle such discrete behaviors. Discrete systems can be *event driven* or *time driven. Discrete Event Systems* (**DES**) are event driven systems, and they have emerged as a field where many researches from Universities and Industries areas have provided great contributions to perform the study of discrete event systems [3].

For a given discrete event system, we may represent it by different *state-based models*, considering only relevant states needed to analyze and generate desired information. In [6] [34] some of the main techniques for modeling discrete event dynamic systems (DEDS)are the followings: *Markov processes* and their *embedded Markov chains*, *Petri Nets*, *queuing networks*, *automata* and *finite − state machines*, *formal languages*, *finitely recursive process*, *min − max algebra models*, and *discrete event simulation* and *semi − Markov processes*. Also, in [6] [34] those techniques are brief and clearly explained and illustrated. Two of the most common modeling formalism for DES are based on *automata and Petri nets (PN)*, and each formalism holds its own characteristics that users show a preference for any of them as they feel more comfortable or find them more suitable to deal with a DES. In this work we only are interesting in DES models based on automata and formal languages.

Also, powerful computational tools have been developed not only to deal with continuous time systems, but also, for discrete event systems, *DES*. *Matlab* and *Maple* are trademarks computer programs to handle the math complexity required for a given example. In the specific field of DES very useful computer programs have been developed an a list of them is found in [2]. Their capabilities and friendly-user interaction are briefly explained. These computer programs can help an user to deal with the complexity of modeling, analysis and graph representation leaving the user to acquire the needed information.

In this thesis work we use the formalism of DESs, under Ramadge-Wonham *RW* frame-work, to generate a state-based model to be turned into on state-graph to be analyzed by use of Continuous Time Markov Chains (*CTMC*) with absorbing states. In this work we use the computer program CTCT55, developed by Control Group at University of Toronto, for modeling, synthesis and tests of properties for discrete modular supervisors. The numerical calculations and plots were carried out by using the computer program ©Maple, Ver. 9.0

2.3.1 Discrete Event Systems

Definition 2.3 *(Discrete Event Systems, DES), a discrete-event system is a discrete-state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time. [6, 33].*

In this work, we use *finite-state automaton* as a formal modeling tool as a graph representation. A finite-state automaton is the most explicit model among the DES representations, [34].

Finite state machines (FSM) and automata theory are some of the high level of abstraction languages and graph-representations for discrete event systems. Such representations only include relevant state information, without a detail of information in between main states, such as in Fig. 2.4.

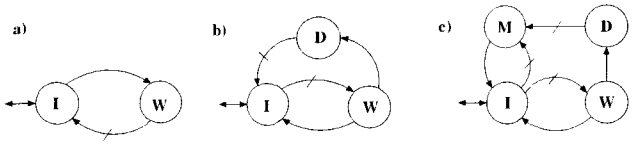


Figure 2.4: DES Modeling

2.3.2 Languages and Finite Automata

Based on [13, 14, 17] we provide the following definitions.

Definition 2.4 *(language) A language defined over an event set E is a set of finite-length strings formed from events in E.*

Example 2.1 *Let $E = \{a, b, c\}$ be the set of events. We may generate the languages L_1 and L_2 as follows.*

$$L1 = \{\varepsilon, a, ab, bc\}$$

$$L2 = \{\textit{all possible strings on length 4 starting with event } b\}$$

Definition 2.5 *(catenation) Catenation is an operation involved in building strings, and thus languages, from a set of events E.*

Example 2.2 *String bc from L_1 is formed by catenation of events b and c.*

Example 2.3 *The catenation of two strings x and y generates a string $z = xy$*

Definition 2.6 *(E^*: Kleene-closure) Let us denote E^* as the set of all finite strings from elements of E, including the empty string ε. The * operation is called The $Kleene - closure$. E^* is countable but infinite since it includes strings of arbitrary long length [5] [6] [45].*

Example 2.4 *Let $E = \{a, b\}$, then*

$E^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, bbb, ...\}$

A language over an event set Σ is therefore a subset of Σ^*. In particular, $\emptyset, \Sigma$, and Σ^* are languages.

From [5] some *concepts about strings* are expressed as follows. Let string $s = xyz$ with $x, y, z \in \Sigma$ and $s \in \Sigma^*$, then:

Definition 2.7 *(prefix) x is named a $prefix$ of s,*

Definition 2.8 *(substring) y is named a substring of s, and*

Definition 2.9 *(suffix) z is named a $suffix$ of s.*

Note that both ε and s are *prefixes* (substrings, suffixes) of s.

Among some operations on languages there is one called $prefix-closure$ which is defined as follow:

Let $L \subseteq E^*, then$

$$\bar{L} := \{s \in E^* : \exists t \in E^* \ (st \in L)\}$$

Paraphrased, it means that the $prefix\ closure$ of L is the language denoted by $\bar{L}$ and consisting of all the prefixes of all the strings in L. In general, $L \subseteq \bar{L}$.

Definition 2.10 *L is said to be $prefix-closed$ if $L = \bar{L}$. Thus, language L is $prefix-closed$ if any prefix of any string in L is an element of $\bar{L}$.*

2.3.3 Finite Automata

A *finite automaton* (FA) is a graph model of a discrete event system (DES), that is made up of a finite number of nodes (*states*) and directed-arcs (*events*). This graph-representation is called the *transition diagram* of a finite automaton. A given DES may be represented by different number of states, depending on the needed information to be obtained from that system. In Fig.2.4 some examples of possible state-based representation are shown for a manufacturing machine. In the first graph, relevant states are: **I**dle and **W**orking. In the second graph, the relevant states are: **I**dle, **W**orking and **D**own; and in the third graph, the relevant states are: **I**dle, **W**orking, **D**own and under **M**aintenance.

Definition 2.11 ***(Finite Automaton (FA) representation)*** *(From Fig. 2.5)*

A *finite automaton* is a 5-tuple $G = (Q, \Sigma, \delta, q_0, Q_m)$, where:

- Q : is a nonempty finite set of states; $Q = \{I, W, D, M, R\}$
  - Σ : is a nonempty finite set of symbols, named the *alphabet*; $\Sigma = \{\alpha, \beta, \lambda, \sigma, \mu, \nu, \omega\}$
  - $\delta : Q \text{ x } \Sigma \longrightarrow 2^Q$ is the transition function; $\delta(I, \alpha) = W$; $\delta(W, \sigma) = D, ...$

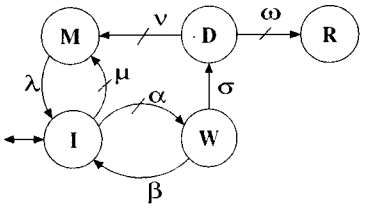


Figure 2.5: DES automaton

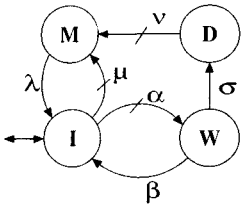


Figure 2.6: Co-Reachable Automaton

- $Qm \subseteq Q$ is a finite set of marked or final states; $Q_m = \{I\}$
- $q_0 \in Q$ is the initial state; $q_o = I$

δ is a *partial function*, where only meaningful or feasible transitions can occur, *i.e.*, for $I, W \in Q$, and $\alpha \in \Sigma$, the transition $\delta(I, \alpha) = W$, represents the transition from state I to state W by event α. In a graph mode, it is represented as $I \xrightarrow{\alpha} D$. For states $I, W, D \in Q$, *and* $\alpha, \beta, \sigma \in \Sigma$, the transition $\delta(\delta(I, \alpha), \beta) = \delta(W, \sigma) = D$. The partial transition function δ possesses a property named *catenation.*

In the sequel, for a state $q \in Q$, and $\sigma_1 \in \Sigma$, the expression $\delta(q, \sigma_1)!$ is denoted to represent *feasible and well defined transitions from state* q *by event* σ_1.

Let us denote by Σ^* the set of all finite strings of elements of Σ, including the empty string ε. Say string $s_1 \in \Sigma^*$ can be formed by a catenation of symbols $\sigma_1, \sigma_2 \in \Sigma$, thus is, $\sigma_1\sigma_2 = s_1$; and $\delta(q_1, s_1) = q_3$. Now δ is extended to a partial function $\delta : Q \text{ x } \Sigma^* \rightarrow Q$ by the rules

$$\delta(q, \varepsilon) = q$$

$$\delta(q, s\sigma) = \delta(\delta(q, s), \sigma)$$

provided that $q' = \delta(q, s)!$ and $\delta(q', s)!$.

Automaton G is named as a *deterministic automaton* when it is defined by $Q \text{ x } \Sigma \longrightarrow Q$, otherwise is named as *non-deterministic automaton.*

Note that the words *state machine* and *generator* are also often used to describe an automaton [6] [45].

From Fig.2.5 some important states *properties* are the followings.

Definition 2.12 *(reachable state): For states $q_1, q_2 \in Q$,state q_2 is called reachable if it can be reached from the initial state q_0 by successive transitions.*

Example 2.5 *$\delta(\delta(I,\alpha),\beta) = \delta(W,\sigma) = D$; state D is reachable.*

Definition 2.13 *(co-reachable state) For state $q \in Q$ and $q_m \in Qm$, where $Q_m \subseteq Q$, state q is called $co-reachable$ if a marked or final state q_m can be reached from that state.*

Example 2.6 *$\delta(W,\beta) = I$; state W is $co-reachable$. Also, states I, W, D, M, and R are reachable states.*

Definition 2.14 *(accessible automaton) An automaton is accessible if all states are reachable.*

Example 2.7 *Automaton from Fig. 2.5 is reachable, since all its states can be reached from the initial state I by successive transitions.*

Definition 2.15 *An automaton is called $co-accessible$ if all states area co-reachable.*

Example 2.8 *Automaton from Fig. 2.5 is not $co-accessible$ since state I cannot be reached from state R.*

Automaton from Fig. 2.6 is $co-accessible$ since its initial state I can be reached from all the states.

Definition 2.16 *An automaton is named trim, when it is both, accessible and co-accessible.*

Example 2.9 *Automaton from Fig. 2.6 is a trim automaton since it is both, accessible and co-accessible.*

Definition 2.17 *If automaton G is trim implies G is nonblocking, but the converse is false[45].*

Example 2.10 *From automaton of Fig. 2.6: for $q \in Q$, and $\sigma_1 \in \Sigma$, $\delta(q,\sigma_1)$!.*

Two languages generated by automaton G are of interest:

Definition 2.18 *$L(G)$ is the language generated by the uncontrolled process G, i.e., the set of all possible sequence of events which take the initial state q_0 to some reachable state in G.*

$L(G) = \{\omega | \omega \in \Sigma^*, \delta(qo,\omega) \in Q \text{ is defined}\}$

Definition 2.19 *$L_m(G)$ is the marked language generated by the uncontrolled process G, i.e., the set of all possible sequence of events which take the initial state q_0 to some final (or marked) state in G. $L_m(G) = \{\omega | \omega \in \Sigma^*,\ \delta(qo,\omega) \in Q_m\}$*

2.3.4 Basic Supervisory Control Theory

In the Supervisory Control Theory (SCT)[6, 32, 33, 45] the *plant* is a model of the uncontrolled system, i.e., the plant models everything that can be done in the (physical) system to be controlled. The *supervisor* is an entity that restricts the behavior of the plant such that given specifications are not violated. Accordingly, the *specification(s)* models how the closed-loop system should behave. Both, the plant, the supervisor, and the specifications can be modeled as finite automata.

SCT partitions the events into *controllable* or *uncontrollable*. If Σ is an event set, then $\Sigma_c \subseteq \Sigma$ denotes the subset of *controllable events*, and $\Sigma_u \subseteq \Sigma$ the subset of *uncontrollable events*. Furthermore, $\Sigma_c \dot{\cup} \Sigma_u = \Sigma$ and $\Sigma_c \cap \Sigma_u = \emptyset$.

SCT solves two main problems: the *controllability* and the *non-blocking* problems. In this paper, we examine supervisors that results in a controllable and non-blocking closed-loop system. Let G be the automaton modeling the plant, and K the automaton modeling the specification. We will assume that all languages from G are marked as accepting, i.e. $L_m(G) = L(G)$. Since the plant is a model of the uncontrolled system, all information about what is *an accepting or forbidden state* should be modeled in the specification. In general, $L_m(K) \neq L(K)$.

Given a plant G and a specification K, the *supervisor synthesis problem* is to find a supervisor S such that, assuming $\Sigma_G = \Sigma_S = \Sigma_K$,

1. $\overline{L_m(G\|S)} = L(G\|S)$,
2. $\emptyset \subset L_m(G\|S) \subseteq L_m(K)$,
3. G is complete for G.

Condition 1 implies that all closed-loop strings can be always extended to a string that is marked by both, the plant and the supervisor, i.e., the closed-loop system is non-blocking. Condition 2 states that the closed-loop system must always be able to reach a string marked by K. Finally, condition 3, states that the supervisor, S, is never allowed to disable any uncontrollable event possible in G.
A supervisor S is *complete* for plant$G$, if

$$G\|S = G_\Sigma\|_{\Sigma^c} S \tag{2.1}$$

A necessary and sufficient condition for completeness (see [3])

$$L(S)\,\Sigma_u \cap L(G) \subseteq L(S) \tag{2.2}$$

When S is *complete* for G, we say that the language $L(S)$ is *controllable* with respect to $L(G)$ Controllability can be reformulated in the following way.

Definition 2.20 *(Controllability) Let G be a plant, and let S be a supervisor. $L(S)$ is controllable with respect to $L(G)$ if the following relations hold.*

$$\begin{aligned} \text{i. } st &\in L(G) \\ \text{ii. } t &\in (\Sigma_u)^* \end{aligned}$$

$$s \in L(S) \Rightarrow st \in L(S) \tag{2.3}$$

Condition i requires st to be a string in the plant, and condition ii requires t to be a string of exclusively uncontrollable events. If the two condition are fulfilled, then, controllability implies that the continuation t of s must be possible in S if the string s is possible in S.

Let $C(P)$ be the set of all languages that are controllable with respect to P and $NB(P,K)$ be all languages such that condition i and ii above are fulfilled. Then, it is known that $C(G)$ is closed under union [32]. It is also known that $NB(G,K)$ is closed under union. The intersection of two union-closed sets of languages contains an unique supremal language, e.g. [6]. This supremal language will be denoted $NBC^{\uparrow}(G,K)$, where

$$NBC^{\uparrow}(G,K) = \{L \mid L \in C(G) \cap L \in NB(G,K)\} \tag{2.4}$$

When G is clear from the context, we will write $K^{\uparrow}$, to refer to *the supremal controllable* and non-blocking language.

2.3.5 SCT from RW framework

Without any external control, DES generates some languages that describe the uncontrolled behavior of the system. Most of the time, it is required to restrict this behavior to satisfy some *specification*. As we mentioned earlier, the events in a DES can be categorized as controllable and uncontrollable. Only the controllable events (which will be denoted by the notation ": γ" in the figures) can be enabled or disabled by an external agent. A *supervisor* is an agent that enables or disables controllable events such that the process G generates the language that conforms to the specifications. Formally, the supervisor S consists of a finite automaton S_a and an output function Ψ (i.e. the control pattern), $S = (S_a, \Psi)$ where

$$S_a = (\Sigma, X, \xi, x_0, X_m) \text{ and}$$

$$\Psi : \Sigma \times X \rightarrow (0 : \text{disable}, 1 : \text{enable}) \text{ s.t.} \Psi(\sigma, x) = 0 \text{ or } 1 \text{ if } \sigma \in \Sigma_c$$

$$\Psi : \Sigma \times X \rightarrow (0 : \text{disable}, 1 : \text{enable}) \text{s.t.} \Psi(\sigma, x) = 1 \text{ if } \sigma \in \Sigma_u.$$

Definition 2.21 *Controllability*

According to Ramadge-Wonham theory, both, the supervisor S and the process G are modeled as finite automata that work in parallel. G generates an un-controlled plant language and the finite automaton model S_a for the supervisor generates a *specification language*. The supervisor and the process are coupled to form a closed-loop system. This feedback mechanism between S and G functions as follows: Let's assume that the process is in state q_i and the supervisor is in state x_j at a given time. A subset of events $\sigma \in L(G)$ can occur in the uncontrolled process in state q_j. These subset of events are fed into the supervisor. According to state x_j (which actually derives from the specification language) only a subset of these events may be allowed. Then, the supervisor issues a *control pattern* Ψ, such that, the controllable events are enabled if they are permitted at x_j, and disabled, if they are not. The idea is illustrated by figure 2.7.

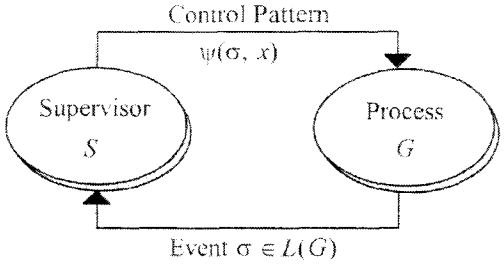


Figure 2.7: Supervisor and DES coupled for closed loop system

The closed loop system under control is denoted by S/G. The following three languages generated by S/G are of interest.

- The *closed – loop behavior* of S/G is described by the language

$$L(S/G) = L(G) \cap L(S)$$

$L(S/G)$ consists of the sequence of events of uncontrolled process language that survive under supervision.

- The *marked behavior* of S/G is described by the language

$$L_m(S/G) = L_m(G) \cap L_m(S)$$

$L_m(S/G)$ consists of the sequence of events that are marked by both G and S.

- The *controlled behavior* of S/G is described by the language

$$L_c(S/G) = L_m(G) \cap L(S/G)$$

$L_c(S/G)$ consists of the marked sequence of events of the uncontrolled process language that survive under supervision.

Example 2.11 *Suppose that the plant model is represented by G and the supervisor is given by S in figure 2.8 below. The coupled system is represented by S/G. The marked state sets for the plant and the supervisor are* $Q_m = \{q_o\}$ *and* $X_m = \{x_o\}$, *respectively.*

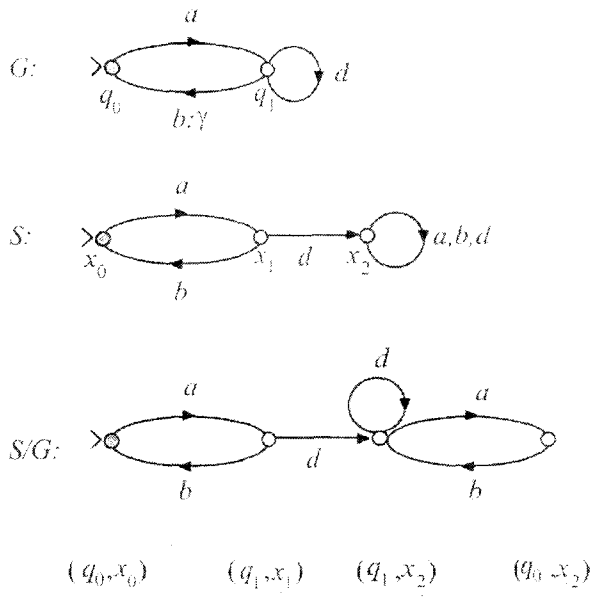


Figure 2.8: A plant G, supervisor S and the coupled system S/G.

The marked languages of the plant and the supervisor are $L_m(G) = (ad^*b)^*$ and $L_m(S) = (ab)^*$, respectively.

From figure 2.8 we can write $L(S/G)$ as

$$L(S/G) = (ab)^*(\varepsilon + (ad^*b)^*)$$

Then the controlled language $L_c(S/G)$ is given by

$$L_c(S/G) = (ab^*)(\varepsilon + (ad^*b)^*) \cap (ad^*b)^* = (ad^*b)^*$$

and the marked language is

$$L_m(S/G) = (ab)^* \cap (ad^*b)^* = (ab)^*$$

The difference between $L_m(S/G)$ and $L_c(S/G)$ can be seen easily with this example. The strings in $L_m(S/G)$ satisfy the marking conditions set by both S and G. That is, $L_m(S/G)$ contains those strings which takes the coupled system from the initial state to the combined marked states, which is (q_0, x_0) in this example. On the other hand, $L_c(S/G)$ consists of those strings that are marked by G only and that survive under supervision. For example 2.11, q_0 is the only state marked by G. And two states marked by q_0, namely (q_0, x_0) and (q_0, x_2), survive under supervision. $L_c(S/G)$ contains those strings which take the combined system from the initial state to either (q_0, x_0) or (q_0, x_2).

It is desirable that the supervisor holds some properties:

Definition 2.22 *A supervisor S is complete (with respect to G), if $\omega \in L(S/G)$, $\omega\sigma \in L(G)$ and $\Psi(\sigma, \xi(\omega, x_0)) = 1$, then, $\omega\sigma \in L(S/G)$. That is, if a string ω is included in the closed loop behavior and event σ is allowed by the supervisor, then, the' string $\omega\sigma$ must also be included in the closed loop behavior.*

Definition 2.23 *A supervisor S is proper (with respect to G) if S/G is nonblocking, i.e., $\overline{L_m}(S/G) = L(S/G)$.*

For the combined system in example 2.11 (figure 2.8), $\overline{L_m}(S/G) = \overline{(ab)^*}$ and $L(S/G) = \overline{(ab)^*(\varepsilon + (ad^*b)^*)}$. Hence $L_m(S/G) \neq L(S/G)$ and supervisor S is not proper since S/G is blocking. In this example G is trim but S is not. The resultant S/G is not trim in this case. We need to note that even if S and G are both trim S/G may not necessarily be trim. This is illustrated by the following example.

Example 2.12 *We have a trim generator G for the plant and a trim generator S for the supervisor as seen in figure 2.9 below. However the supervisor is not proper since S/G is blocking.*

As seen in figure 2.9 any string in G and S can be completed to a string that brings the generators to their respective final states, q_0 and x_0. However, no string in S/G takes the combined system back to the final state (q_0, x_0).

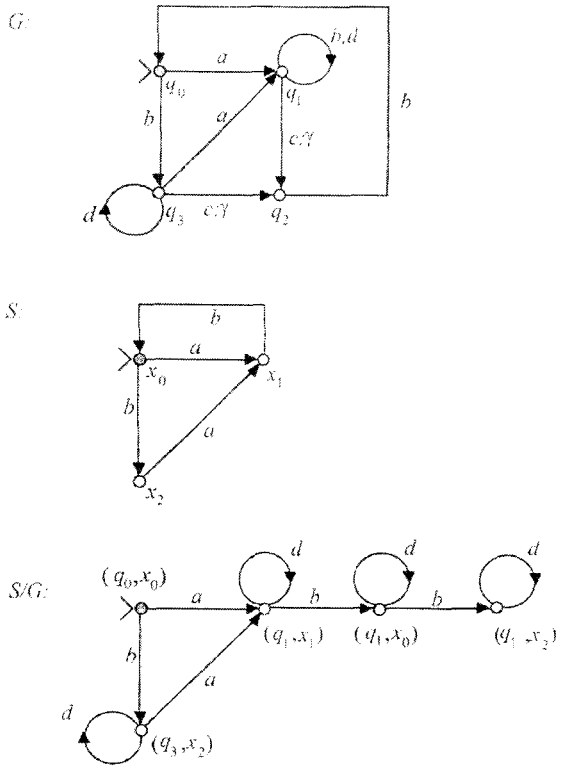


Figure 2.9: Generators G and S are trim but S/G is still blocking

It may not always be possible to construct a supervisor with respect to the given specifications. The following theorems are reported by Ramadge and Wonham [32] [46] on the existence of a supervisor. First we give some more definitions.

Definition 2.24 *Let K and L be two languages, $K \subset L$. The language K is L – closed if $K = \overline{K} \cap L$*

Definition 2.25 *A language K is controllable with respect to G if $\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$.*

This means, given a string (sequence of events) ω which is a prefix of K, we add an uncontrollable event $\sigma \in \Sigma_u$ s.t. the word $\omega\sigma$ is a sequence of events in G,i.e., $\omega\sigma \in L(G)$. If adding the event σ causes the string to exit from the prefix closure $\overline{K}$, then K is not controllable since σ is an uncontrollable event and cannot be avoided. See figure 2.10 below

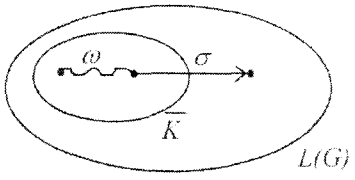


Figure 2.10: K is uncontrollable

Theorem 2.1 *Let $K \subset L(G)$ and $K \neq \emptyset$. There exists a complete supervisor S such that $L(S/G) = K$ if and only if $K = \overline{K}$ (i.e. K is prefix closed) and controllable.*

Theorem 2.2 *Let $K \subset L_m(G)$ and $K \neq \emptyset$.*

1. There exists a proper supervisor S such that $L_m(S/G) = K$ if and only if K is controllable.
2. There exists a proper supervisor S such that $L_c(S/G) = K$ if and only if K controllable and $L_m(G)$ closed.

Example 2.13 *Consider the plant model G as in the figure 2.8. The set of controllable and uncontrollable events are $\Sigma_u = \{a, d\}$ and $\Sigma_c = \{b\}$, respectively.*

1. Assume that the specification language is given as $K = (ab)^*$. K is uncontrollable with respect to G because when the system is in state q_1, the supervisor can enable event b but cannot force it to happen. Since $d \in \Sigma_u$ event d can also occur when the system is in state q_1. Hence, no complete supervisor can be constructed to satisfy the specification language by theorem 2.1.
2. Assume that the specification language is given as $K = (ab)^*d^*$. Now K is controllable with respect to G. However, K is not $L_m(G)$ closed since $L_m(G) \cap \overline{K} = (ad^*b)^* \cap \overline{(ab)^*d^*} = (ab)^* \neq K$. Hence there does not exist a proper supervisor such that $L_c(S/G) = K$.

If the specification language K is not controllable, i.e., no supervisor can be constructed such that $L(S/G) = K$, then, the specification language is modified as a controllable sublanguage K' , $K' \subset K \subset L(G)$, for which a supervisor S' exists, and $L(S'/G) = K'$. The sublanguage K' is called the *supremal language*. The term *supremal* refers to the "maximal inclusion of strings" with respect to the original specification language K. [32] provides an iterative procedure to compute the *supremal language*. The idea is illustrated by figure 2.11.

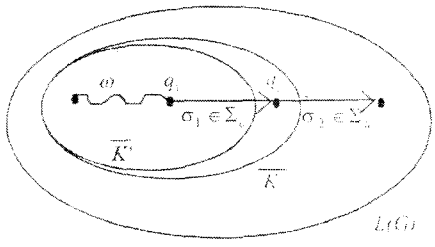


Figure 2.11: Computation of the supremal language

Initially, the specification language is given by K. However, at state q_j no external agent can avoid the system from crossing the boundaries of the specification language K since $\sigma_2 \in \Sigma_c$, implying K is uncontrollable. Hence, the idea is to prevent the system from entering in to state q_j. In figure 2.11, state q_j is only accessible from state q_i by an event $\sigma_1 \in \Sigma_c$ Since the event σ_1 is controllable, it can be disabled by a supervisor so that state q_j is no longer reachable. According to this procedure, the specification language is reduced to a sub language K' (shaded region in figure 2.11) which excludes those states (like q_j in figure 2.11) that are the cause for the uncontrollability of the specification language K.

2.4 Modular Supervisory Control

A *monolithic approach* is that G is controlled by only one supervisor S, and a problem arises with a combinatorial state-space explosion. Models of industrially interesting problems grow huge, limiting the monolithic approach to both, with regard to time and space [3].

However, typically a model of an industrial plant is built by interacting smaller sub-plants. This is,

$$G = G_1 \| G_2 \| \ldots \| G_m \tag{2.5}$$

where the sub-plants potentially have very different alphabets. Furthermore, a supervisor may also be composed of smaller sub-supervisors with potentially very different alphabets,

$$S = S_1 \| S_2 \| \ldots \| S_n \tag{2.6}$$

each supervising parts of the global plant G. Exploiting this modular structure of the plant and supervisor, we may gain significant improvement in time and space consumption of the verification and synthesis operations.

In a *modular* setting, the modular sub-plants G_i and supervisors S_j do not in general have the same alphabets. Since we want to exploit the modularity in verifying controllability and synthesizing controllable supervisors, we have to take the non-equality of the alphabets into account. In this case, the definitions of controllability above do *not* capture the necessary requirements. For modular supervisors and modeling of sub-plants see [32, 46, 45, 7, 8]

Definition 2.26 *(Controllability - Automata) Let P be a plant and S be a supervisor. No restrictions are placed on the alphabets of P and S. S is controllable with respect to P if for each string $s \in L(S\|G)$ the following relation holds,*

$$\sigma \in \Sigma(p) \cap \Sigma_u \cap \Sigma_S \Longrightarrow \sigma \in \Sigma(q) \tag{2.7}$$

where

$$p = \delta_P\left(q_{iP}, \mathrm{Pr}\,oj_{\Sigma_P}(s)\right) \tag{2.8}$$

$$s = \delta_S\left(q_{iS}, \mathrm{Pr}\,oj_{\Sigma_S}(s)\right) \tag{2.9}$$

That is, p is the state in G after observing the string s, and q is the corresponding state in S.

S is non-controllable with respect to P there exists a string s and a corresponding state $\langle q, p\rangle$ such that $\sigma \in \Sigma(p) \cap \Sigma_u \cap \Sigma_S \Longrightarrow \sigma \notin \Sigma(q)$, in this case we will say that $\langle q, p\rangle$ is an uncontrollable state.

Sometimes the term *completeness* is used to denote controllability between two automata.

The definitions of controllability in terms of languages becomes more complicated with non-equals alphabets, forcing us to use inverse projection in order to be able to intersect languages.

Definition 2.27 *(Controllability - Languages)*

The same assumptions as in Definition (number). S is controllable with respect to P if their languages fulfill the following relation.

$$L^{-1}(S)(\Sigma_u \cap \Sigma_P) \cap L^{-1}(G) \subseteq L^{-1}(S) \tag{2.10}$$

Definition 2.28 *(Configuration)*

A configuration is a finite set of automata that interact through FSC.

We will regard the sets of sub-plants and sub-supervisors, as well as respective sub-sets thereof, as configurations. Thus, it becomes meaningful to define what we mean by controllability of one configuration with respect to another.

Definition 2.29 *(Controllability - Configurations)*

Let $F_1 = \{F_1^1, \ldots, F_1^m\}$ and $F_2 = \{F_2^1, \ldots, F_2^n\}$ be two configurations. F_1 is said to be controllable with respect to F_2 if $F_1^1 \| \ldots \| F_1^m$ is controllable with respect to $F_2^1 \| \ldots \| F_2^n$.

If F_1 is empty then it is controllable by definition. To make the notation easier we will introduce the the function $Controllable\,(F_1, F_2)$, which is true if F_1 is controllable with respect to F_2, and false otherwise. We also want to consider controllability for a subset, $\Sigma' \subseteq \Sigma_u$, of uncontrollable events. This is written as $Controllable\,(F_1, F_2, \Sigma')$. To include this case 2.7 is changed to

$$\sigma \in \Sigma(p) \cap \Sigma' \cap \Sigma_S \Longrightarrow \sigma \in \Sigma(q) \tag{2.11}$$

Note, that when implementing this function only one synchronization, if more than two automata are allowed in the synchronization, is needed.

2.5 Probability and Random Variables

As in [21] random phenomena are common in natural as well as in man-made systems. The subject of *probability* provides a mathematical tool to study of random phenomena to understand their behavior, and predict (and possible control) their future. A probability model has three components:

1. sample space;
2. events; and
3. probability of events.

Definition 2.30 *A sample space, Ω, is the set of all possible outcomes of a random phenomenon. Elements of Ω, generally denoted by ω, are called the outcomes or sample points.*

Definition 2.31 *An* event *is a subset of the sample space. Since Ω is subset of Ω, it is seen that Ω is an event, called the* universal event. *Similarly, the null set, $\emptyset$, is a subset of Ω, that is called the* null event. *Thus, the universal event always takes place, while the null event never takes place.*

Events E_1, E_2,... are said *exhaustive* if at least one of these events always takes place. Mathematically, this is equivalent to saying that:

$$\bigcup_{n=1}^{\infty} E_n = \Omega.$$

Similarly, events E_1, E_2,... are said to be *mutually exclusive or disjoint* if at most one of the events can take place. This is equivalent to saying:

$$E_i \cap E_j = \emptyset, \text{ if } i \neq j.$$

Thus, when E_1, E_2,... are mutually exclusive and exhaustive, they define a *partition* of Ω, i.e., each sample point $\omega \in \Omega$ belongs to one and only one E_n.

As in [21], Table 2.1 show the correspondence between event and a set terminology.

The *third component* of a probability model is *the probability function*. It associates a numerical value to each event that describes the likelihood of observing the event.

Definition 2.32 *(Probability of event E), denoted as **P**(E), is a number representing the likelihood of occurrence of the event E. The probability function satisfy the following requirements.*

Axioms of Probability. **(1)** $0 \leq \mathbf{P}(\mathrm{E}) \leq 1$. **(2)** $\mathbf{P}(\Omega)=1$. **(3)** If E_1, E_2 ,... are disjoint events, then $\mathbf{P}(\cup_{n=1}^{\infty} E_n) = \sum_{n=1}^{\infty} P(E_n)$. In different real life situations we only know partial knowledge about a phenomenon, and we need to know probabilities of some events based on the current knowledge. The concept of conditional probability is needed to handle this situation.

Table 2.1: Correspondence between event and set-theoretic terminology

Event description	Set-theoretic Notation
E_1 *or* E_2	$E_1 \cup E_2$
E_1 *and* E_2	$E_1 \cap E_2$ or E_1E_2
Not E	E^c or $\overline{E}$
At least one E_1, $E_2, ...$	$\bigcup_{n=1}^{\infty} E_n$
All of $E1$, $E2, ...$	$\bigcap_{n=1}^{\infty} E_n$

Definition 2.33 *The conditional Probability of an event E, given that event H has taken placed, is denoted by* $\mathbf{P}(E|H)$ *and is given by*

$$\mathbf{P}(E \mid H) = \frac{\mathbf{P}(\mathrm{EH})}{\mathbf{P}(\mathrm{H})} \tag{2.12}$$

assuming $\mathbf{P}(\mathrm{H}) > 0$. As it is pointed out in [21] about the above definition, some remarks are the followings.

1. (a) i. $\mathbf{P}(\mathrm{F}) = 0$, then $\mathbf{P}(\mathrm{EH})$ is also 0.
   ii. $\mathbf{P}(E|H)$ does not imply that event E takes place after event H. There is no chronological order implied..
   iii. The probability of event E, given that event H is going to take palace in the future, is also given by (2.12)
   iv. Equation 2.12 can be written as: $\mathbf{P}(\mathrm{EH}) = \mathbf{P}(\mathrm{E|H})\ \mathbf{P}(\mathrm{H})$. This equation is valid even if $\mathrm{P}(\mathrm{H}) = 0$.

2.5.1 Independence

Two events are independent if information about one does not affect the likelihood of the occurrence of the other.

Definition 2.34 *Events E and H are said to be independent of each other if:*

$$\mathbf{P}(EH) = \mathbf{P}(E)\mathbf{P}(H)$$

If events E and H are independent, and P(H) > 0, then we have

$$\mathbf{P}(E|H) = \frac{\mathbf{P}(EH)}{\mathbf{P}(H)} = \frac{\mathbf{P}(E)\mathbf{P}(H)}{\mathbf{P}(H)} = \mathbf{P}(E)$$

Definition 2.35 *Mutual Independence; Events $E_1, E_2, ..., E_n$ are said to be mutually independent if for any subset $S \subseteq \{1, 2, ..., n\}$*

2.5.2 Cumulative Distribution Function(cdf)

In [21] when we are interested in the probability that a random variable takes values in a given set, these sets are intervals in the real line (-∞,∞). Thus, we want to know the probability of $\{X \in (a, b)\}$.

Definition 2.36 *(Probability function of X). Let E be a subset of the real line. Then*

$\mathbf{P}(\{X \in E\}) = P(\{\omega \in \Omega : X(\omega) \in (a, b)\})$. If $E = (-\infty, x]$ we write:
$\mathbf{P}(X \leq x)$ instead of $\mathbf{P}(\{X \in E\})$.Similar notations are given as follow.
$\mathbf{P}(\{X \in (a, b)\}) = P(a < X < b)$,
$\mathbf{P}(\{X \in (a, b]\}) = P(a < X \leq b)$,
$\mathbf{P}(\{X \in (x, \infty)\}) = P(X > x)$,
$\mathbf{P}(\{X \in \{x\}\}) = P(X = x)$

Definition 2.37 *(Cumulative Distribution Function, cdf). The function*

$$F(x) = \mathbf{P}(X \leq x), \quad x \in (-\infty, \infty)$$

is called the *cumulative distribution function, cdf*, of the random variable X. Then, for the following expression:

$$\mathbf{P}(X > x) = 1 - F(X)$$

Without any proof, the following theorems show the four main properties of the *cdf*:

Theorem 2.3 *(Properties of the cdf)*

1. $F(.)$ is a nondecreasing function, i.e., $x \leq y \Longrightarrow F(x) \leq F(y)$.
2. $F(.)$ is *right continuous*, i.e., $\underset{\varepsilon \downarrow 0}{Lim} F(x + \epsilon) = F(x)$
3. $Lim_{s \to \infty} F(x) = F(-\infty) = 0$.
4. $Lim_{s \to \infty} F(x) = F(\infty) = 1$.

2.5.3 Moments

From [39] let X be a random variable. The first moment, E[X], is the ordinary expectation or mean value of X. To center the origin of measurement, it is convenient to work with powers of $(X - E[X])$. We define the k-th central moment, μ_k, of the random variable X by $\mu_k = E[(X - E[X])^k]$. Of special interest is the quantity $\mu_2 = E[(X - E[X])^2]$ known as the variance of X, Var[X], often denoted by σ^2.

Definition 2.38 *(Zero Moment) for a random variable X is equal to 1.*

Definition 2.39 *(First Moment): for a random variable X, is represented by E[X] and it corresponds to the expected or mean value of X.*

Definition 2.40 *(Second Moment): The variance of a random variable X is represented by* σ^2*, or by* $E[(X - E[X])^2]$.

Definition 2.41 *The square root of the variance is called standard deviation,* σ_X *or* σ.

Definition 2.42 *(Third Moment): Skewness is defined to be the third moment about the sample mean, divided by the third power of the standard deviation, and it measures the degree of symmetry of the data set. A perfectly symmetric data set has a skewness of zero. If the data set has some extremely small values, then the skewness will be negative. If the data set has some extremely large values, then the skewness will be positive.*

Definition 2.43 *(Fourth Moment): Kurtosis is defined to be the fourth moment about the sample mean divided by the fourth power of the standard deviation, and it measures the degree of flatness or peakedness of a data set. For the normal distribution, the kurtosis is 3. If the distribution has a flatter top, the kurtosis is less than 3. If the distribution has a high peak, the kurtosis is greater than 3.*

2.5.4 Transform Methods

As pointed out in [39] in many probability problems, the form of the density function (for the continuous case) may be so complex so as to render computations difficult, if not impossible. A transform can provide a compact description of a distribution, and it is relatively easy to compute: the mean, the variance, and other moments directly from a transform, rather than resorting to a tedious sum (discrete case) or an equally tedious integral (for the continuous case). So, the Laplace-Stieltjes transform can become an useful tool to solve the differential equations that may be generated.

2.5.5 Continuous Random Variables

Continuous random variables take values continuously in an interval in the real line.

Definition 2.44 *(Continuous Random Variables). A random variable with cdf $F(.)$ is said to be continuous if there exists a function $f(.)$ such that*

$$F(x) = \int_{-\infty}^{\infty} f(u)\, du \tag{2.13}$$

for all $x \in (-\infty, \infty)$. In particular, if F(.) is a differentiable function, then it is a cdf of a continuous random variable an the function f(.) is given by

$$f(x) = \frac{d}{dx}F(x) = F'(x). \tag{2.14}$$

Eqs. 2.13 and 2.14 show a way to describe a continuous random variable.

Definition 2.45 *Probability Density Function, pdf. The function f(.) of eq. 2.13 is called the probability density function of X.*

2.5.6 The Exponential Continuous Random Variable: $Exp(\lambda)$

Consider a random variable with parameter $\lambda > 0$ taking values in [0, ∞) with the following pdf:

$$\begin{aligned} f(x) &= \{0, \quad if x < 0, \\ f(x) &= \lambda e^{-\lambda x}, \ if x \geq 0. \end{aligned} \tag{2.15}$$

The corresponding *cdf* can be computed using (2.13) as

$$\begin{aligned} F(x) &= 0, \qquad\qquad if\ x < 0, \\ F(x) &= 1 - e^{-\lambda x}, \qquad if\ x \geq 0. \end{aligned} \tag{2.16}$$

For the exponential function:

$$\mathbf{P}(X > x) = 1 - P(X \leq x) = 1 - [1 - e^{-\lambda x}] = e^{-\lambda x}. \tag{2.17}$$

2.5.7 Moment Generation Function

In [21, 39] for a random variable X, C is another random variable. Th expectation $E[e^{X\theta}]$ will be a function of θ.

Define the moment generation function (MGF) $M_X(\theta)$, abbreviated as $M(\theta)$, of the random variable X by

$$M(\theta) = E[e^{X\theta}] \tag{2.18}$$

so that

$$M(\theta) = \int_{-\infty}^{\infty} e^{x\theta} f(x)dx$$

for the continuous case.

If X is a nonnegative continuous random variable, then we define the (one-sided) Laplace-Stieltjes transform (LST) of X:

$$L_X(s) = L(s) = \int_{-0}^{\infty} e^{-sx} dF(x)dx = \int_{-0}^{\infty} e^{-sx} f(x)dx \tag{2.19}$$

From [39] the moment generation property of the MGF, $e^{X\theta}$, can be expanded in into a power series:

$$e^{X\theta} = 1 + X\theta + X^2\frac{\theta^2}{2!} + ... + X^k\frac{\theta^k}{k!} + ... \tag{2.20}$$

Taking expectations on both sides of (2.20) (assuming all expectations exist) yields

$$M(\theta) = E[e^{X\theta}] = 1 + E[X]\theta + ... + \frac{E[X^k]\theta^K}{k!} + ... = \int_{k=0}^{\infty} \frac{E[X^k]\theta^K}{k!} \tag{2.21}$$

Therefore, the coefficient of $\frac{\theta^k}{k!}$ in the power series-expansion of its MGF yields the k-th moment $E[X^k]$ of the random variable X. Alternatively

$$E[X^k] = \frac{d^k M}{d\theta^k}|_{\theta=0}, \quad k = 1, 2, ... \tag{2.22}$$

For the case of $Laplace - Stieltjes$ transform

$$E[X^k] = (-1)^k \frac{d^k L(s)}{ds^k}|_{s=0}, \ k = 1, 2, ... \tag{2.23}$$

(From [39]) Let X be exponentially distributed with parameter λ. Then

Example 2.14 $f_X(x) = \lambda e^{-\lambda x}$, *for* $x > 0$, *and*

$L_X(s) = \int_{-0}^{\infty} e^{-sx}\ f(x)dx$

$L_X(s) = \int_{-0}^{\infty} e^{-sx}\ \lambda e^{-\lambda x}dx$

$L_X(s) = \frac{\lambda}{s+\lambda}\int_{-0}^{\infty}(s+\lambda)e^{-(s+\lambda)x}$

$L_X(s) = \frac{\lambda}{s+\lambda}$.

2.6 Markov Chains and Continuous Time Markov Chains

In real life system's operation show a certain degree of *uncertainty* due to fail of machines, human being interaction, to mention a few of them. This uncertainty is included by considering the underlying transfer line (TL) as a Markov process, which constitute [12] the most important subclass of stochastic processes, which can be considered as a generalization of the concept of random variables. A stochastic process provides a relation between the elements of a possibly infinite family of random variables. In this work we consider the case of Continuous Time Markov Chains (CTMC) with finite state space. The following definitions are taken from [12] and [39].

Definition 2.46 *A stochastic process is defined as a family of random variables* $\{X_t : t \in T\}$ *where each random variable* X_t *is indexed by parameter* $t \in T$*, which is named as the time parameter if* $T \subseteq \mathbb{R}_+ = [0, \infty)$.

The set of all possible values of X_t *(for each* $t \in T$*) is known as the state space* S *of the stochastic process. The state space of a stochastic process may also be either continuous or discrete.*

Definition 2.47 *A stochastic process* $\{X_t \in T\}$ *constitute a Markov process if for all* $0 = t_0 < t_1 < ... < t_n < t_{n+1}$ *and for all* $s_i \in S$*, the conditional CDF of* $X_{t_{n+1}}$ *depends only on the last previous value* X_{t_n} *and not on the earlier values* $X_{t_0}, X_{t_1}, ..., X_{t_{n-1}}$:

$$P(X_{t_{n+1}} \leq s_{n+1} | X_{t_n} = s_n, X_{t_{n-1}} = s_{n-1}, ..., X_{t_0} = s_0) = P(X_{t_{n+1}} \leq s_{n+1} \leq | X_{t_n} = s_n) \tag{2.24}$$

This equation possesses what it called as a *Markov memoryless property*. Informally, it can be interpreted as if a system is found in its current state X_n, its next state transition only depends on the current state and neither on the history (previous states) that led to that state nor on the time already spent in the current state.

2.6.1 Continuous Time Markov Chains (CTMC)

As mentioned in [12] CTMC and DTMC provide different but related modeling paradigms, each of them having their own domain of applications. In CTMC the state transitions may occur at any arbitrary instants of time.

Definition 2.48 *A given stochastic process* $\{X_t : t \in T\}$ *constitutes a CTMC if for arbitrary* $t_i \in \mathbb{R}_0^+$*, with* $0 = t_0 < t_1 < ...t_n < t_{n+1}$, $\forall n \in \mathbb{N}$, *and* $\forall s_i \in S = \mathbb{N}_0$, *for the conditional pmf, the following relation holds:*

$$P(X_{t_{n+1}} \leq s_{n+1}|X_{t_n} = s_n, X_{t_{n-1}} = s_{n-1}, ..., X_{t_0} = s_0) = P(X_{t_{n+1}} \leq s_{n+1}|X_{t_n} = s_n). \quad (2.25)$$

Now, the *exponential distribution* is the only continuous-time distribution that provides the *memoryless* property, and the state sojourn time of a CTMC which is necessarily exponential distributed. The right hand side of *Eq.*(2.25)is referred as the *transition probability* $p_{ij}(u, v)$ of the CTMC to travel from state i to state j during the period of time $[u, v)$, with $u, v \in t$,and $u \leq v$:

$$p_{ij}(u, v) = P(X_v = j|X_u = i) \quad (2.26)$$

If the state transition probabilities $p_{ij}(u, v)$ depend only on the time difference $t = v - u$, and not in the actual values of u and v, the time homogeneous CTCM is expressed as follows.

$$p_{ij}(t) = p_{ij}(0, t) = P(X_{u+t} = j|X_u = i) = P(X_t = j|X_0 = i), \forall u \in T. \quad (2.27)$$

Now, given the transition probabilities $p_{ij}(u, v)$ and the probabilities $\pi_i(u)$ of the CTMC at time u, the unconditional state probabilities $\pi_j(v)$, $j \in S$ of the process at time v can be represented as follows.

$$\pi_j(v) = \sum_{i \in S} p_{ij}(u, v)\pi_i(u), \quad \forall u, v \in T(u \leq v) \quad (2.28)$$

In the time homogeneous case, Eq. (2.28) is reduced to:

$$\pi_j(t) = \sum_{i \in S} p_{ij}(t)\pi_i(0) = \sum_{i \in S} p_{ij}(0, t)\pi_i(0) \quad (2.29)$$

The Chapman-Kolmolgorov for the transition probabilities of a CTMC can be expressed as follows.

$$p_{ij}(u, v) = \sum_{k \in S} p_{ik}(u, w)p_{kj}(w, v), \quad 0 \leq u \leq w \leq v. \quad (2.30)$$

As mentioned in [12, 39, 5], Eq. (2.30) cannot be solved easily. So, it has to be transformed into instantaneous transition rates $q_{ij}(t)$ $(i \neq j)$ of the CTMC traveling from state i to state j.

$$\frac{d\pi_j(v)}{dt} = \sum_{k \in S} q_{kj}(v)\pi_k(v) \quad (2.31)$$

Expressed Eq. (2.31) as in the time-homogeneous case:

$$\frac{d\pi_j(t)}{dt} = \sum_{k \in S} q_{kj}(t)\pi_k(t), \quad \forall j \in S \tag{2.32}$$

and Eq. (2.32) expressed in vector-matrix form,

$$\dot{\pi}(t) = \frac{d\pi(t)}{dt} = \pi(t)Q \tag{2.33}$$

Where, $\mathbf{Q} = [q_{ij}], \forall i, j \in S$,and the q_{ii} elements from the main diagonal of Q are defined as: $q_{ii} = -\sum_{j,j \neq i} q_{ij}$.
Fig.2.12 shows a rate balance for a given state.

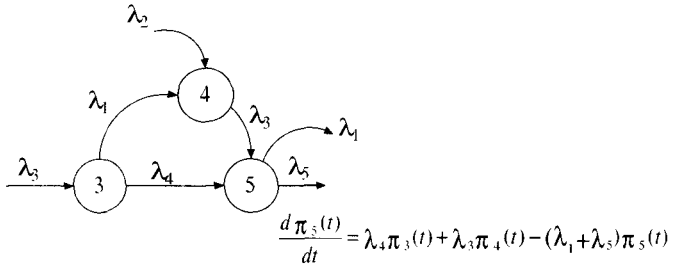


Figure 2.12: Rate balance representation for some given states

To acquire a time-based representation of the TL in its transient time evolution, the TL is analyzed as finite state and continuous time representation. The technique to be used is Continuous Time Markov Chain (CTMC). As mentioned in [5, 12, 39] to analyze a system under this framework, it is needed as information: a transition state matrix and an initial state. The kind of states a system may include corresponds to *transient* and *absorbing* states (one or more). If only transient states made up the system, it transits until reaching a steady state behavior. On the other hand, if one or more absorbing states exist, the system won't reach its steady state, but it will reach an absorbing state. When one absorbing state is present we know that sooner or later, that state will be reached by the system. If so, it may be interesting to analyze or observe the transient behavior or characteristics of the system in this short term. When more than one absorbing state exist, we may be interested on which state will be reached first. In [12] the sojourn time of a CTMC must have a memoryless property. Since the exponential distribution is the only continuous distribution with this property, the random variables denoting the sojourn times or holding times, must be exponentially distributed. Note that the same is true for the *residual state holding time*, that is, the time remaining until the next state change occurs. Furthermore, the means of the two random variables (holding and residual times) are equal to $1/(-q_{ii})$. Let the random variable R_i denotes, either, the sojourn time or the residual time in state i, then the cdf is given by:

$$F_{R_i}(r) = 1 - e^{q_{ii}r}, \quad r \geq 0 \tag{2.34}$$

The mean value of R_i, the mean sojourn time or mean residual time, is given by:

$$E[R_i] = -\frac{1}{q_{ii}} \tag{2.35}$$

where q_{ii} *is interpreted* as the total rate at which state i is exited (to another state) at time t, and $q_{ij}(t)$, $(i \neq j)$, denotes the rate at which the CTMC leaves state i in order to transit to state j at time t.

2.7 Transient Solution of Markov Chains

A steady state solution for CTMC for AMS becomes a suitable analysis and representation, when we want to check the long run behavior for an underlying system. But, if a system is prone to fail due to a machine failure, blocking, and other different conflicts, *transient analysis* for CTMC is more appropriated for modeling and analysis of a system in a shorter-term behavior. *Transient analysis* as mentioned in [12, 27, 39], is gaining interest and particular importance in dependability modeling. Also, it has been lately important, [40, 43] in computation of quantities related to transient probabilities such as *cumulative measures*. To model and analyze a system under Markov chain approach, for the discrete time case, we need the transition probability matrix ($\mathbf{P}$) and an initial state ($\pi(\mathbf{0})$).

In the case of CTMC, the computation of the transient state probability vector $\pi(t)$, requires the solution of a system of linear differential equations, and two information pieces, such as the infinitesimal generator matrix, $\mathbf{Q}$, and the initial probability vector, $\pi(0)$ are needed.

$$\frac{d\pi(t)}{dt} = \pi(t)\mathbf{Q}; \quad \pi(0) = (\pi_0(0), \pi_1(0), ...) \tag{2.36}$$

In [4, 12, 27, 43] measures that can be immediately derived from transient state probabilities are often referred to as *instantaneous measures*. Relevant information as measures based on cumulative accomplishments, such as time to block, can be obtained during a given period of time $[0, t)$. Let

$$L(t) = \int_o^t \pi(u)du \tag{2.37}$$

Cumulative measures can be directly computed from the transient solution of Eq. (2.35).

2.8 Automated Manufacturing Systems (AMS)

A main characteristic of manufacturing processes is that they exhibit a discrete behavior. This is not only because they generate countable parts, but also, in the logic order of steps to follow to complete a finished product. The size and flexibility to process manufacturing products, among others factors, generate different arrangement of machines (layouts), where the flow of raw parts follows a predefined order and the source allocation to different machines must be controlled. Such is the case of a *transfer line*, where parts are processed in a tandem order

by different machines until they become a finished part. In automated manufacturing systems (AMS) one well known conflict named *deadlock* occurs on the shop floor. *Deadlock* [41, 30] is considered as a situation where each of a set of two or more parts keep waiting indefinitely for the other parts in the set to release resources, or is a logical condition arising in discrete event dynamic systems (DEDS). It is caused by an inappropriate allocation of *resources* to concurrent executing processes.

In AMS, *resources* refers to: *machines, buffers, AGV, robots, fixtures, tools, etc.* Deadlocks lead to degraded performance and ultimately zero throughput, or at least the productivity remains in a halt state. [42] In an improperly designed AMS, the only *remedy* for deadlock may be *manual clearing* of machines or AGV or buffers, and then, restart of the system from an initial condition that is known to produce deadlock-free operation under nominal operation conditions. The loss of production and the labor costs in resetting the system in this way, can be avoided by proper design. Of course, some particular approaches dealing with blocking may includes policies of no-preemption actions to avoid waste of materials. Often, deadlocks may result because of simple errors in software specifications of the controllers (synthesis). Four necessary conditions have been identified for the occurrence of deadlocks [41, 30].

1. Mutual Exclusion: a resource can not be used by two or more processes simultaneously.
2. No preemption: when a resource is being used, it is not released unless the process using it finishes with it.
3. Hold and Wait: there must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.
4. Circular Wait: there must exist a set of waiting processes $[p_1, p_2, \ldots, p_n]$ such that p_1 is waiting for a resource held by p_2; p_2 is waiting for a resource held by $p_3, \ldots, p_{n-1}$ is waiting for a resource held by p_n; and p_n is waiting for a resource held by p_1.

Different approaches have been used for modeling and analyzing blocking [30]. These modeling approaches are based on: Automata, Petri Nets, Markov Chains, and Digraphs. Although, each approach has its own strengths and weakness, but not superiority is obtained from any of those. As it is also mentioned, their use is a matter of preference by the users, as they feel more comfortable with each approach. The three main strategies addressing deadlock issues in different contexts are:

1. Prevention
2. Detection/Recovery, and
3. Avoidance.

Definition 2.49 Deadlock prevention: *Refers to static resource allocation policies to eliminate completely the possibility of deadlock.*

The static allocation is made so as to break one or more of the necessary conditions. It is known that deadlock prevention policies lead to reduction in resource utilization and system throughput. Prevention strategies use information about process to specify all required

resource requirements and demand each process to specify all required resources before transactions begin. The simplest means of preventing deadlock is to outlaw *concurrency* (no more than two process run simultaneously); but such a method is overly conservative, generally leading to significantly low utilization of system resources. Its main advantage is that it does not require the knowledge of the system state to realize the control, leading to a simple control law.

Deadlock-detection/recovery: Deadlock detection and recovery go together. It uses a monitoring mechanism for detecting the deadlock occurrence and a resolution procedure for preempting some deadlock resources.

Definition 2.50 Recovery: *refers to restart of the system from an initial condition that is known to produce deadlock-free operation. It can be expensive since it may involve removal of semi-finished parts from the system or resource preemption.*

This strategy in general allows higher resource utilization than prevention does. It should be used when deadlock is rare and detection and recovery are not expensive and much faster than the deadlock generation. *This type of strategy plays a very important role in this thesis work.*

Definition 2.51 Deadlock-avoidance: *We let the system try to use and develop the full potential of the resources and avoid a deadlock just before it occurs by breaking one of the necessary conditions. Thus, deadlock avoidance involves dynamic resource allocation, using look-ahead policies, in order to know the acquisition and release of resources. This is a dynamic approach that can utilize the knowledge of the current allocation of resources and the future behavior of processes to control the release and acquisition of resources. When a deadlock avoidance algorithm is implemented, it is necessary to consider and avoid not only deadlock states, but also, unsafe states, i.e., states that lead inevitably the system to reach deadlock. Deadlock avoidance techniques are considered the most efficient strategies to avoid deadlock in AMS.*

Chapter 3

Method I: Synthesis of Blocking-Recovery Discrete Modular Agents (BRDMA)

3.1 Introduction

In this chapter we present the first method related to synthesize the set of blocking recovery discrete modular agents, named as BRDMA, which will be dealing with blocking to control a manufacturing transfer line. To execute the three proposed methods, we use as a main example, a manufacturing transfer line taken from [45], an it is described in section 3.1.1. In different literature about controlling manufacturing systems, we find *two important issues* which normally are dealt with independently. The *first issue* refers to design or synthesis of control entities named as controllers or supervisors, but not evaluation of their performance is provided. On the other hand, the *second issue* considers that a set of given specifications are satisfied and a performance evaluation is provided, but nothing is shown about the synthesis of control entities. To show a more realistic behavior of a manufacturing system both issues should be included, to analyze the whole performance, of both, the control entities and the controlled system. From the control entities we expect not only to execute a production plan, but also, we want to obtain the full potential of the machines involved in a production system. A high expectation may cause undesirables behaviors such as deadlock (blocking).

We need to keep in mind that the performance of controlling any transfer line is measured in terms of *throughputs* and *utilization*, and other ones, but such success is not acomplished by the independent machines, but by the way we control the flow of raw-parts from their input to the system, until they become finished products. In this chapter we concentrate on the synthesis of control entities devoted to control a manufacturing transfer line executing blocking recovery. An engineer some times executes the synthesis of such controllers following a common sense procedure, or some times, he follows a given procedure as mentioned in [3, 45, 6].

We will develop the synthesis of control entities based on ideas from *agents* (Artificial Intelligence), *multiagents* (Distributed Artificial Intelligence), discrete *modular supervisors* from Ramadge-Wonham framework, and Automated Manufacturing Systems (AMS). These ideas enrich the structure of the control entities to be developed and they will be named as

blocking-recovery discrete modular agents,(BRDMA), which will posses some of the main characteristics for an agent or a multiagent system mentioned in [15] [38] [44] and from modular supervisors as mentioned in [45] [6]. The formal approach of supervisory control theory (SCT) from Ramadge-Wonham is used to synthesize such BRDMA and the computer program CTCT (from University of Toronto) is used to execute such synthesis, in a based-state representation.

3.1.1 The Transfer Line (TL) as the uncontrolled plant

The example of manufacturing transfer line (TL) is taken from [45] and it is made up of two processing machines, M_1 and M_2 , followed by a Test Unit, TU. Two buffers, B_1 and B_2, are placed in between these machines. The transfer line executes a full processing when a raw-part visits the machines in the following order: M_1 -> B_1 -> M_2 -> B_2 ->TU -> *Success*: leaves the system; OR, *Fail* : backs to B_1. The need of reprocessing defective parts feeds back the system to place a defective part into buffer B_1 and it may overload it, in case of not empty place. Also, the processing time of each machine may cause in both buffers *starving* or *overloading* into buffers B_1 and B_2. Fig 3.1 shows the TL and the route a part follows either for success or fail in a run test at TU; besides that, a list of labels (numbers) is given to represent the main events generated in the TL.

The *challenge*: synthesize high level control entities that take control of M_1, M_2 and TU to avoid *starving* or *overloading* at buffers B_1 and B_2, but with a high production level.

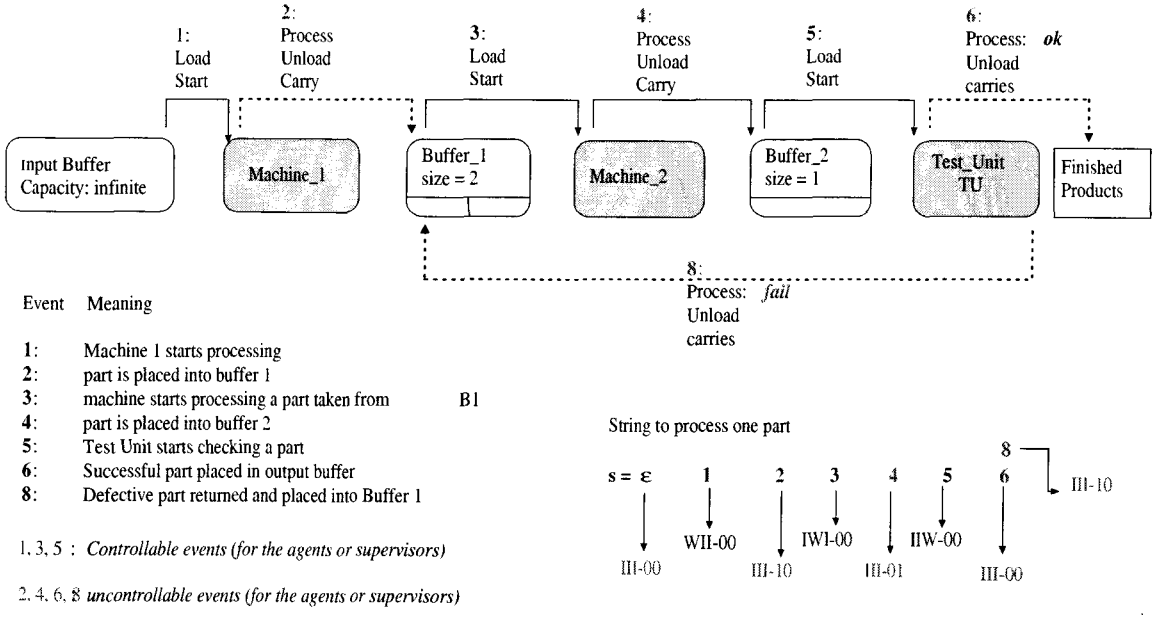


Figure 3.1: Block diagram for the transfer line and its related events and environment.

3.1.2 The three cases of study for the transfer line

We generate three cases of study for the model of transfer line we will be dealing with in this thesis work. Each case of study is different from the other in the size of the first buffer. So, we

will refer to such cases of study as TLB1B1, TLB2B1 and TLB3B1, indicating that the size of the first buffer changes but the size of the second buffer is left un-changed.

3.1.3 Assumptions for the transfer line as a discrete event system

The following assumptions are considered.

1. All events generated by the plant are *observable*. That is, the information from the plant is generated, acquired and available to be used by whom it may need it.
2. Only relevant or high level events drive the system to make it evolve.
3. Deadlock may be a price to pay as we want to synthesize control entities that try to develop the full potential of machines within a production system.

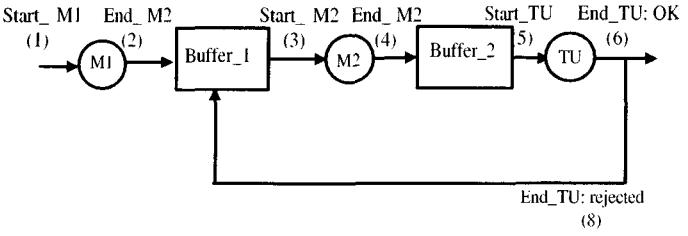
3.2 Elaborating a Multiagent environment into the Transfer Line.

The transfer line, made up of three machines and two buffers in between them, as in Fig. 3.2(a), is the main environment to be dealing with. As a main task is to control overflow and under-flow (starving) in both buffers. We can decompose such main task into a sub-set of tasks, where control of overflow and under-flow for each buffer as the two new sub-tasks, as in Fig. 3.2(b). Now, we want that *independent* control entities, agents or supervisors, take control of each subtask, as in Fig. 3.2(c), and they may *satisfy* not only their own goal, but also, now as a main task, they may develop the full potential of the machines for a high performance of a production system. An undesirable behavior may occur such as *deadlock* (blocking), and the idea is not reject its occurrence, but face it by use of an *un-blocking* mechanism. The set of agents, as in Fig. 3.2(d), can lead the system to any production state, including blocking, as well as they may lead the system from a blocking state to any other free-blocking state, executing an unblocking procedure, as in a conflict resolution problem.

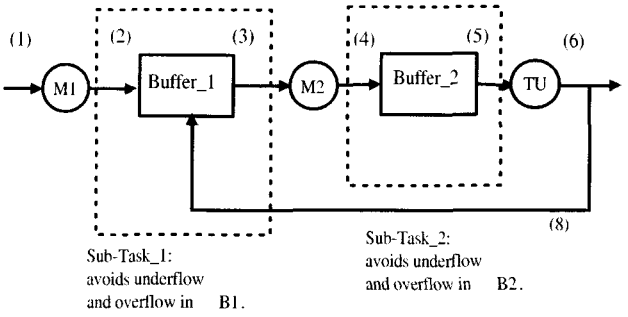
3.3 Synthesis of Blocking-Recovery Discrete Modular Agents, (BRDMA)

The given name of *blocking-recovery discrete modular agents* (BRDMA) suggests that a set of modular agents will be dealing with blocking, within the plant they control, and they will allow the execution of external events to re-initialize the plant to work each time the system blocks. So, first of all, let see how we synthesize a set of discrete modular agents, and after that, how we synthesize their counter-part of *blocking-recovery discrete modular agents.*

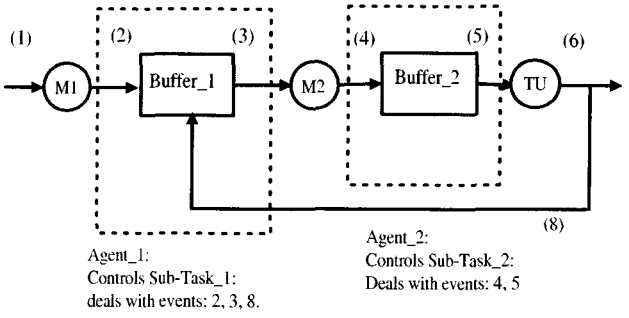
For the three cases of study, we will synthesize the set of BRDMA, rembering that since we perform the synthesis of independent modular agents, we only redo the synthesis for the one



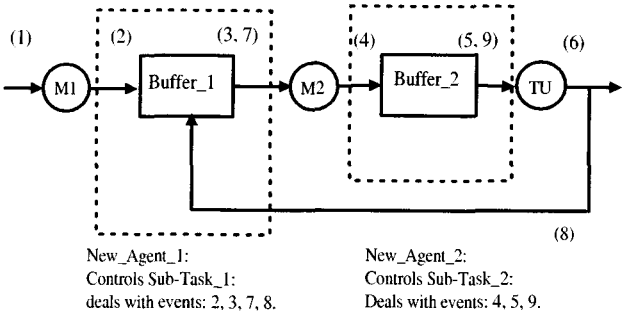
(a) Original transfer line, TL



(b) Sub-task decomposition into the transfer line



(c) Assigning control of sub-tasks to agents



(d) Blocking recovery agents using external events.

Figure 3.2: Generating a multiagent environment to deal with blocking occurrence.

in charge to control the first buffer, but we left unchanged the discrete modular agent for the second buffer.

3.3.1 Synthesis of BRDMA.

Once we have defined an environment for agents into a multiagent system, we model the plant and the specifications, and synthesize the set of modular agents, under Ramadge-Wonham framework, [32, 46, 33]. The main idea to use this state-based framework is to verify formal tests that the modular agents must satisfy to become a feasible solution to deal with blocking recovery. The commands to synthesize the set of discrete modular supervisors are based on the ones included in the computer program CTCT55, from University of Toronto, and as in [45]. We only perform the full development for the case of study TLB1B1, and we mention some changes that apply for the other two cases of study, since we only modify the model of the specifications (buffer-size) for the first buffer, and the synthesis of its own modular agent.

1. The transfer line (TL) as uncontrolled plant (G) is represented by, $G{=}TL$, composed by a set of sub-plants M_1, M_2 and TU as represented in Fig.3.3.
   $TL = synch(synch(M1, M2), TU)$
2. The given set of specifications (sub-tasks) are represented by, $E_i = B_iSP$. In our example, the main task or specification is decomposed in two sub-tasks or a set of two specifications, as $E_1 = B_1SP$ and $E_2 = B_2SP$, for buffers B_1 and B_2, respectively, as represented in Fig. 3.4
3. Perform the synthesis of each independent modular-agent $ModS_i$, (for $i = 1, 2, 3$). In Figures 3.5 and 3.6 we show the state-based representation for AG_1 and AG_2, respectively.

ModS1=*Supcon*(TL, B1SP);

ModS2=*Supcon*(TL, B2SP);

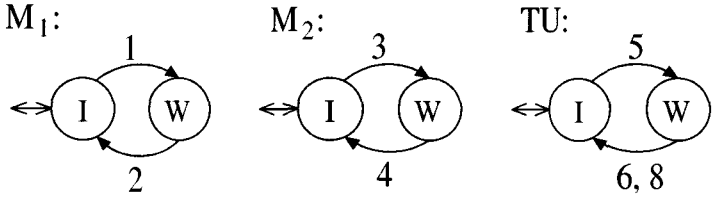


Figure 3.3: DES modeling for the set of sub-plants of the Transfer Line (TL), as a main plant.

Once we performed the synthesis of each modular-agent, it is important to run some tests to verify if they hold two main properties: *controllability* and *non-blocking*, as well as we perform some other tests to obtain their jointed-structure, (like meet operation).

1. Test of *proper- agent* (only disablements of controllable events);
   condat(TL, $ModS_i$; $_{i=1,2}$) We check the list of the disabled events in each state of each agent-supervisor.

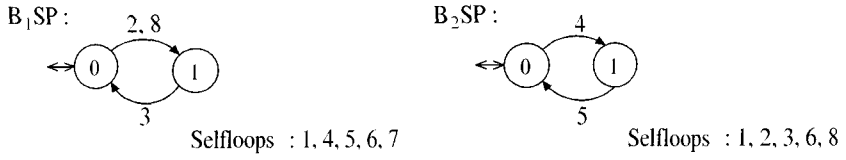


Figure 3.4: Buffers' specifications for the case of B_1=1 and B_2=1.

2. Test of *nonblocking* with respect to the plant they control (no deadlock state is generated);
   nonblocking(TL,AG_i; $_{i=1,2}$). Each independent test may o may not provide a $false$ (blocking) result.

3. Compose both modular-agents, say as $MODA_1A_2$. Their equivalent state based representation is shown in Fig. 3.7 $Meet$(ModS_1,ModS_2)=ModS_1S_2; generate a jointed based-structure for both modular supervisors..

4. Test of *nonblocking*(TL,ModS_1S_2); if the result is $false$, then *they block*, (*acceptable* for our case).

5. Test of proper supervisors. $Condat$(TL,ModS_1S_2): generates the *list* of events that both modular-agents disable from the states of their jointed structure.

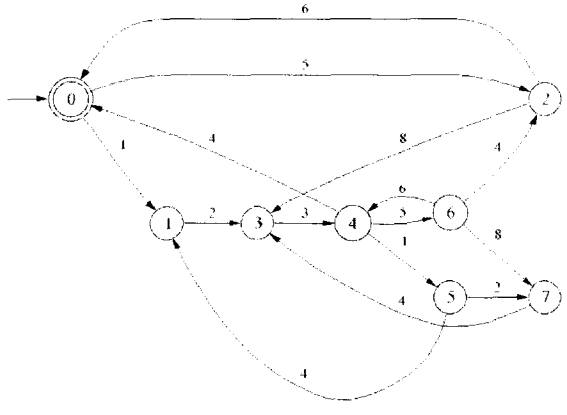


Figure 3.5: Modular Supervisor $ModS1$ to control overflow and underflow at buffer B_1.

In Fig. 3.7, we can see the jointed structure of both modular supervisors, starting their evolution from state 0 as they transit through other states until they reach the blocking-state (9). From state (9), no further transition can be executed. Then, we have reached the main condition, deadlock (blocking) to synthesize the set of BRDMA.

Example 3.1 *A string* $s \in \Sigma_E^*$, *made up of events* s = 123412 *is a possible string to ocurr for the transfer line under control of the current supervisors, and such string will lead the system to reach its deadlock-state. Its state evolution is presented in the Table 3.1, and based on Fig.3.7.*

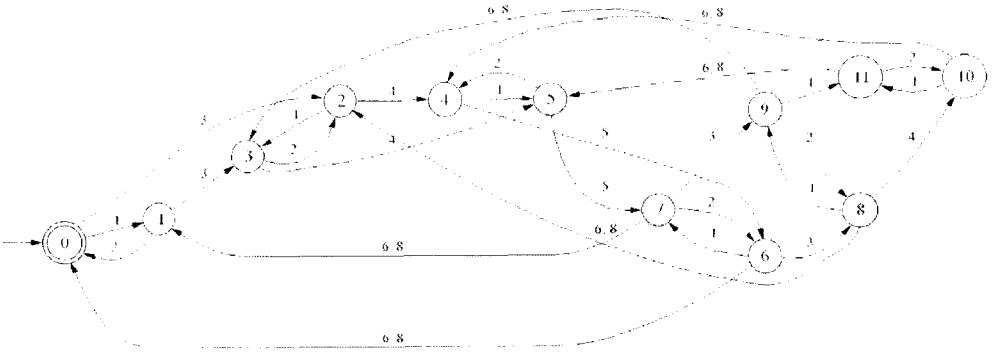


Figure 3.6: Modular Supervisor $ModS2$ to control overflow and underflow at buffer B_2.

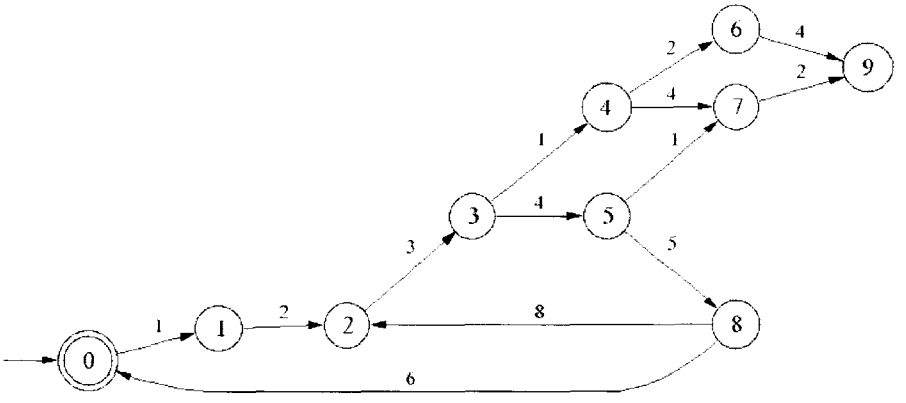


Figure 3.7: Jointed-structure (meet) of both modular supervisors for the case of B_1=1 and B_2=1.

Table 3.1: State Evolution

String	ε	1	2	3	4	1	2
State	0	1	2	3	5	7	9

We will see later, that as we develop the synthesis of our *blocking-recovery discrete modular agents*, BRDMA, the same string of events won't cause blocking.

For the case of TLB2B1, Fig. 3.8 shows its specification represented as a state model. Fig. 3.9 shows the jointed-state structure of both modular agents.

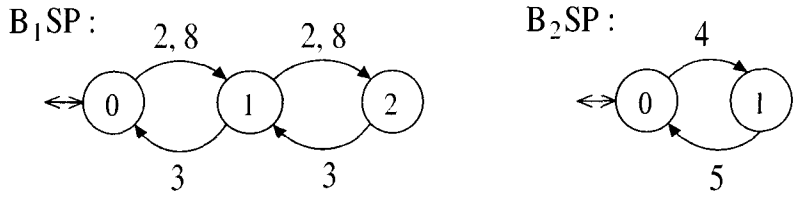


Figure 3.8: Buffers' specifications for the case of B_1=2, and B_2=1.

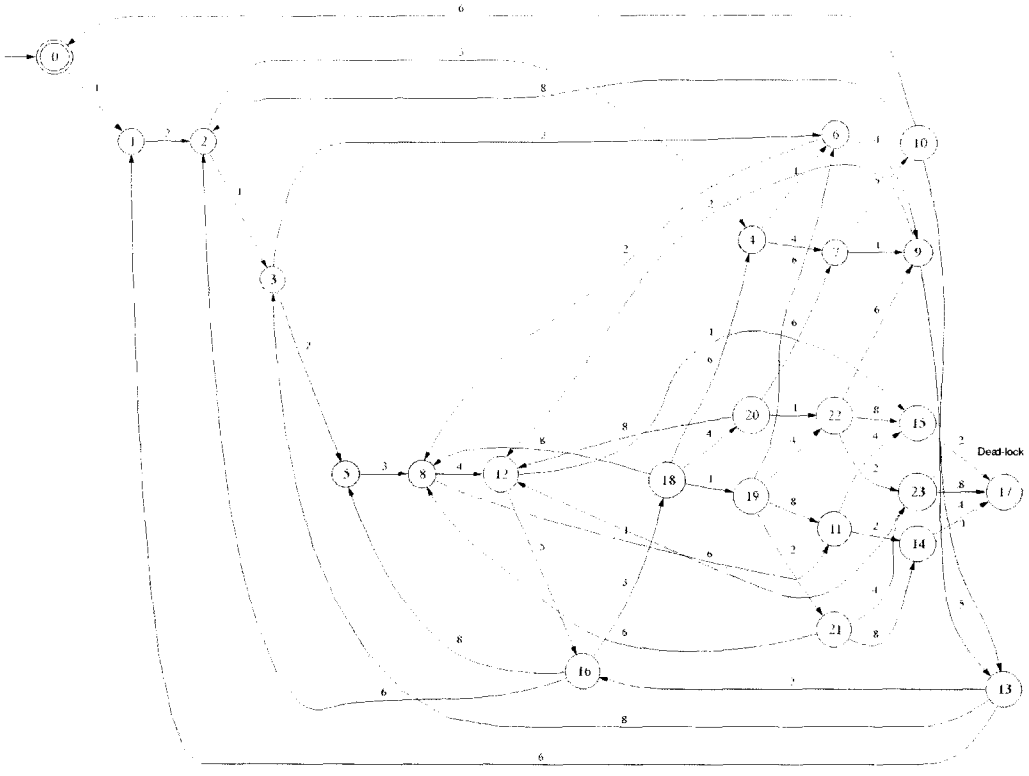


Figure 3.9: *jointed-structure* of both discrete modular supervisors, for the case of study TLB2B1

For the case of TLB3B1, Fig. 3.10 shows its specification represented as a DES model.

3.4 The Unblocking Mechanism (UBM)

Now, let's show how we plan to achieve a blocking recovery of the controlled transfer line of Fig. 3.7. The status the systems adopts as described in [28] is represented in Fig. 3.11, where now,

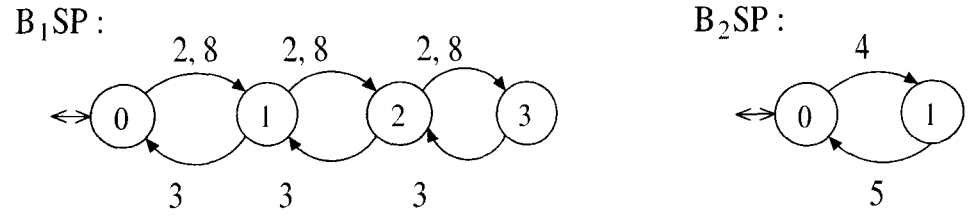


Figure 3.10: Buffers' specifications for the case of B_1=3, and B_2=1.

each state holds a status described as follows: the status the three machine hold followed by the number of parts into buffer B_1 and buffer B_2. This is, for the three machines $M_1M_2TUB_1B_2$, where M_1=M_2=TU={Iddle, Working) can adopt the status of Idle or Working. For instance, a status *IWI10* within state **6** represents that machine M_1 is in its idle state, machine M_2 is working, and test unit TU is idle; meanwhile, buffer B_1 holds one part (full for our example), and buffer B_2 is empty. By an intuitive analysis, when machine M_2 releases the part it is processing (event 4), that part will go to buffer B_2 (an empty place available), but, B_2 will be full. Then the system reaches its blocking state (state 9) leaving the system with a status III11, which means: neither machine M_1 or M_2 is allowed to load a part to be processes, nor TU is allowed to test a part from B_2. So, the system reaches its blocking state. But, how do we know what is allowed or not at state 9? That answer we can be checked from the *condat* test for the composition of both modular supervisors, where we generate a list of events to be disabled by both agents in their jointed based-structure of $ModS_1S_2$. This part is carried out at above step 5.

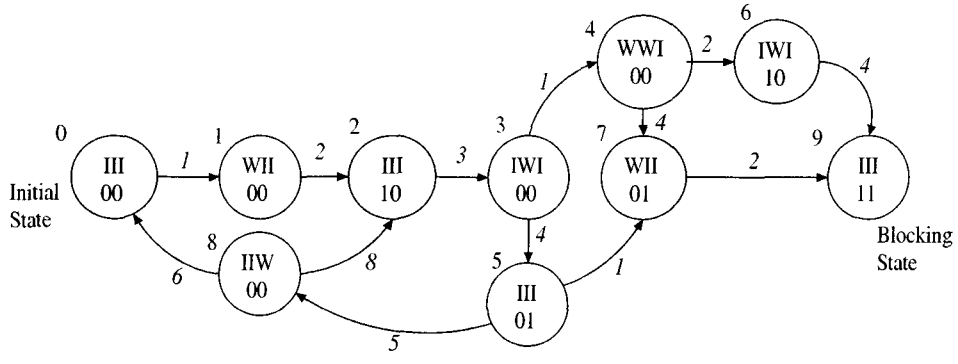


Figure 3.11: Status the transfer line adopts as it evolves under control of both modular supervisors, for the case of TLB1B1.

Definition 3.1 *(un-blocking mechanism) represented by the automaton* $UBM = Q_{ext}, \Sigma_{ext}, q_{eo}, \eta, Q_{me}$, *represents the sequence or execution of external events carried out by an user or a machine, to remove parts from buffers* B_1 *and* B_2, *to release available spaces. The order to start removing parts from one or both buffers, as well as the number of parts to be removed, can be defined by an user or chosen by an* evaluation function *or a quantitative evaluation. The proposed un-blocking mechanism (UBM) is shown in Fig. 3.12.*

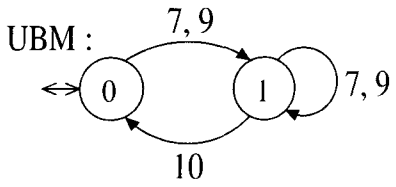


Figure 3.12: DES modeling of the Unblocking Mechanism (UBM)

We can consider $\Sigma_{ext} = \Sigma_{ext\_c} \dot{\cup} \Sigma_{ext\_uc}$ as an alphabet of events generated outside from the TL, where Σ_{ext_c} corresponds to the set of *controllable-external events*, and Σ_{ext_uc} corresponds to the set of *un-controllable external events.*

Definition 3.2 *(External events) are those events that represents external actions executed by a user or technology (arm-manipulator, for instance) to remove, one by one, a part from buffers B_1 or B_2, and a done-event, which complete the end of the process removal.*

The unblocking mechanism provides a set of external events, $\Sigma_{ext} = \{t_1, t_2, t_3\}$, that affect the plant and the specifications. The meaning for those external events are as follows.

- t_1: removal of one part from buffer B_1. (controllable event).
- t_2: removal of one part from buffer B_2; (controllable event).
- t_3: end of the part removal process; (uncontrolled event).

Definition 3.3 *(External alphabet: Σ_{ext}) includes the external events, $\Sigma_{ext} = \{t_1, t_2, t_3\}$, that will help to perform the blocking recovery process into the transfer line.*

$$\Sigma_{ext} = \Sigma_{ext\_c} \dot{\cup} \Sigma_{ext\_uc}$$

$$\Sigma_{ext\_c} = \{t_1, t_2\}$$

$$\Sigma_{ext\_uc} = \{t_3\}$$

It is important to point out that the *main roles* of the *external events* are:

- not only to help us to unblock the system, but also
- lead the system from its blocking state to a free-blocking feasible state, previously chosen.

Definition 3.4 *(equivalent external events) are those events now included in a DES representation. Their equivalent representation as discrete events to be processed into CTCT-program, are: $t_1 = 7$, $t_2 = 9$, $t_3 = 10$, as in Fig. 3.12.*

Definition 3.5 *(Extended Alphabet Σ_E) is the new alphabet made up by disjoint-union of the current events $\sigma \in \Sigma$,and the external events $t \in \Sigma_{ext}$.Thus is,*

$$\Sigma_E = \Sigma \dot{\cup} \Sigma_{ext}$$

$$\Sigma_E = \Sigma_{E_c} \dot{\cup} \Sigma_{E_{uc}}$$

$$\Sigma_{E_c} = \{1, 3, 5, 7, 9\}$$

$$\Sigma_{E_{uc}} = \{2, 4, 6, 8, 10\}$$

For our given DES system, **TL**, if for any $s_1 \in \Sigma^*$ and $q \in Q$, the transition $\delta(s_1, q)$ *is not defined*, we can say that q is a *blocking state* since no further transition can be executed from that state.

If some external events, say t_1, $t_2 \in \sum_{ext}$, and $\sum_{ext} \cap \Sigma = \phi$, but for any t_1, $t_2 \in \sum_{ext}$ and $q \in Q$, $\delta(t_1, q)!$, *isdefined*, we say that $q \in Q$ is not anymore a *blocking state*, due to the external event t_i, for $i = 1, 2$. We can consider $\Sigma_{ext} = \Sigma_{ext\_c} \overset{\bullet}{\cup} \Sigma_{ext\_uc}$ as an alphabet of external events generated outside from the TL, where Σ_{ext_c} corresponds to the set of *controllable−external events*, and Σ_{ext_uc} corresponds to the set of *un−controllable external events*. Now, if we extend the current alphabet Σ, now named and defined as $\Sigma_{extended} = (\Sigma_{ext\_c} \overset{\bullet}{\cup} \Sigma_c) \overset{\bullet}{\cup} (\Sigma_{uc} \overset{\bullet}{\cup} \Sigma_{ext\_uc})$, the new external events will help us not only to unblock the system, but also, to lead the system from the blocking state to a free-blocking feasible state.

3.5 Synthesis of Blocking Recovery Discrete Modular-Agents (BRDMA)

As described in [29][28] we perform *the synthesis of BRDMA* in two steps.

1. Include the effect of the external event to generate: the new-plant (NewTL), the new-specifications(New-B_iSP, for $i = 1, 2$); and synthesize the new-modular supervisors (NewMod$_i$, $i = 1, 2$).
2. Run the set of BRDMA along with the unblocking mechanism, UBM.

This two steps are developed in the two following parts.

3.5.1 Part I:

- **Step 1**: Include the external events, $\Sigma_{ext} = \Sigma_{ext\_c} \overset{\bullet}{\cup} \Sigma_{ext\_uc}$, into the Plant as a DES model, inserting them as self loops, to generate the $New - Plant$ as $DES\ model$, as in Fig. 3.13

- **Step 2**: Include the external events, $\Sigma_{ext} = \Sigma_{ext\_c} \overset{\bullet}{\cup} \Sigma_{ext\_uc}$, into the *specifications*, in the way they affect the desired behavior. The rest of external events include them as self loops. This generates the *new − specifications* as DES models, as in Fig. 3.14 and Fig. 3.15.

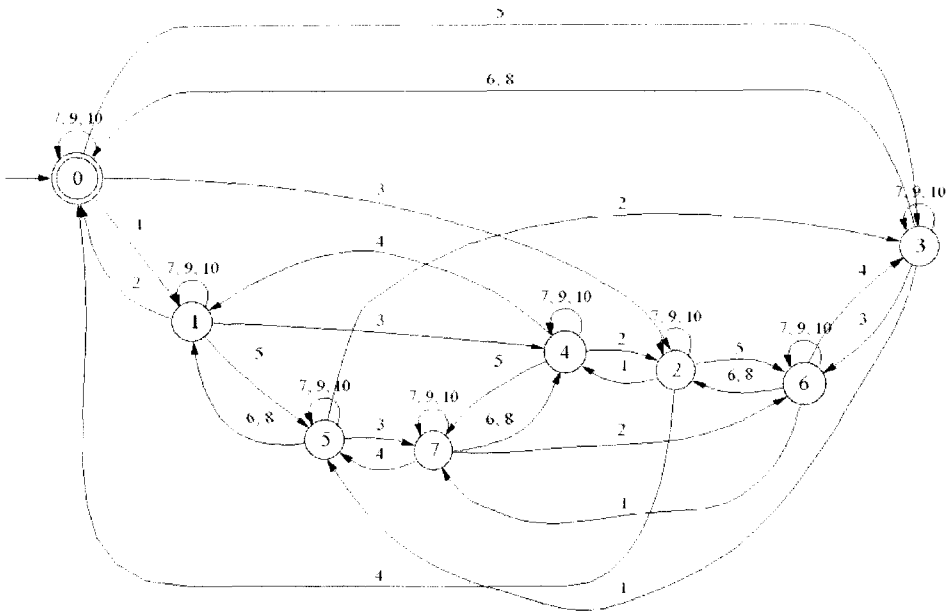


Figure 3.13: State based representation of the whole uncontrolled new-plant: NewTL, including the external events.

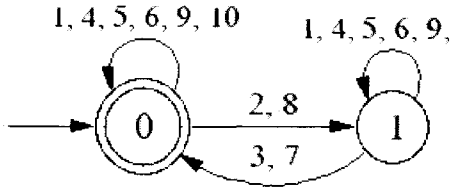


Figure 3.14: New buffer specification, $NewB_1SP$, including the external events 7, 9, and 10.

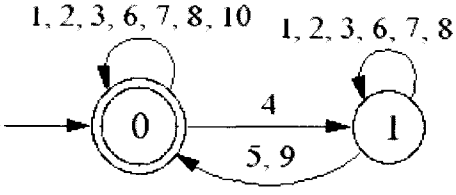


Figure 3.15: New buffer specification, $NewB_2SP$, including the external events 7, 9, and 10.

- **Step 3**: Perform the synthesis of the *new-BRDMA*, as follow.

NewMod1=$Supcon$(NewTL,NewB1SP)

NewMod2=$Supcon$(NewTL,NewB2SP)

The state-based models for the new independent discrete modular supervisors are shown in 3.16 and 3.17, for the NewMod1 and NewMod2, respectively.

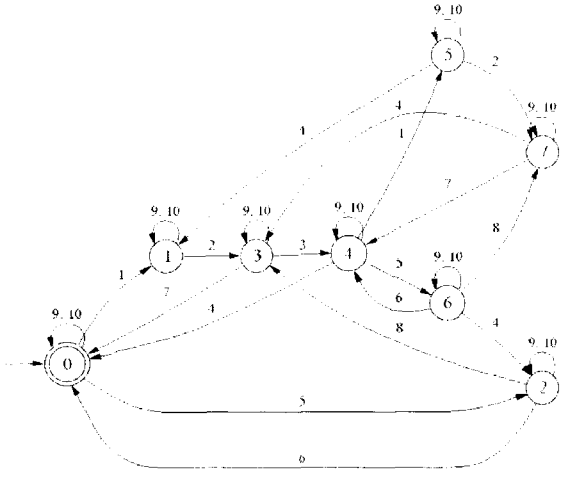


Figure 3.16: State-based representation for the synthesis of the first BRDMA,named NewMod1.

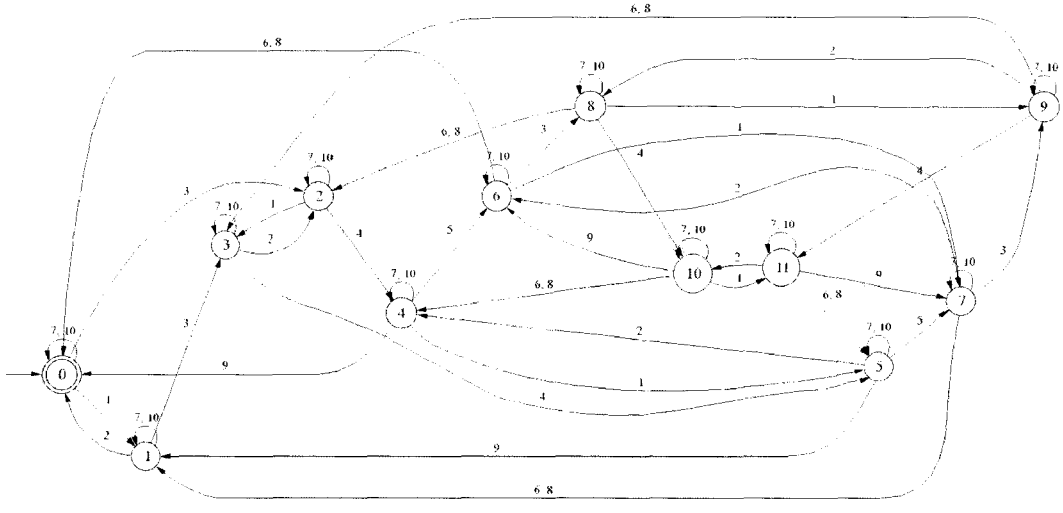


Figure 3.17: State-based representation for the synthesis for the second BRDMA,named NewMod2.

- **Step 4**: Carry out for each BRMDMA the tests of: *controllability* and *nonblocking* with respect to the plant they control. Note: it may be the case of one of this test fails, but continue with step 5.

Test: *nonblocking*(NewTL,NewMod1) = *true* : it means no blocking condition occurs.

Test: *nonblocking*(NewTL,NewMod2) = *true* : it means nonblocking condition occurs.

- **Step 5**: Perform *meet* operation between all BRDMA to compose a jointed-based structure, and perform the same tests as in step 4. If the *nonblocking-test* is *true*, and the *controllability-test* (*condat* feature into CTCT program) does not disable uncontrollable events, we have done our *new-BRDMA*.

NewS1S2=*meet*(NewMod1,NewMod2)

Test: *nonblocking*(NewTL,NewS1S2)=*true*; it means that no blocking condition occurs.

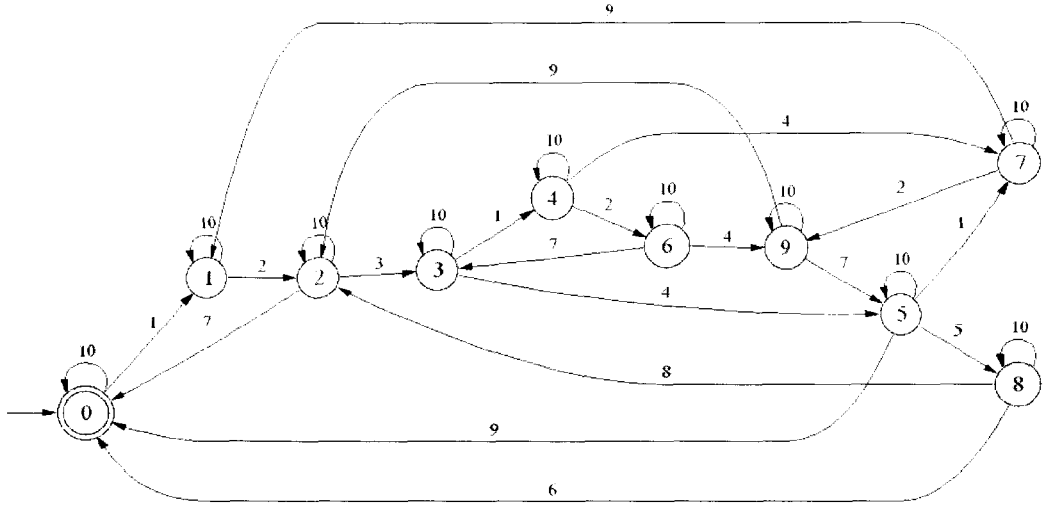


Figure 3.18: Composed state-based representation for the two BRDMA, without a deadlock state within it.

So far, from Fig. 3.18 it can be understood that:

1. No deadlock state is present, since state 9 used to be the deadlock state.
2. Now, external event 9 *partially* recovers the system from blocking as an expected outcome.
3. The state based structure of both BRDMA are ready to accept the external events to execute the unblocking process.

The final blocking-recovery process is presented in the following section 3.5.2, as part II.

3.5.2 Part II

- **Step 6:** Generate the UBM as DES model, as represented in Fig. 3.12.
- **Step 7**: Generate the jointed-base structure of the two BRDMA along with the unblocking mechanism, UBM, as shown in Fig. 3.19.

Perform: $sync$(New-BRDMA,UBM)=$UNewTLB1B2$

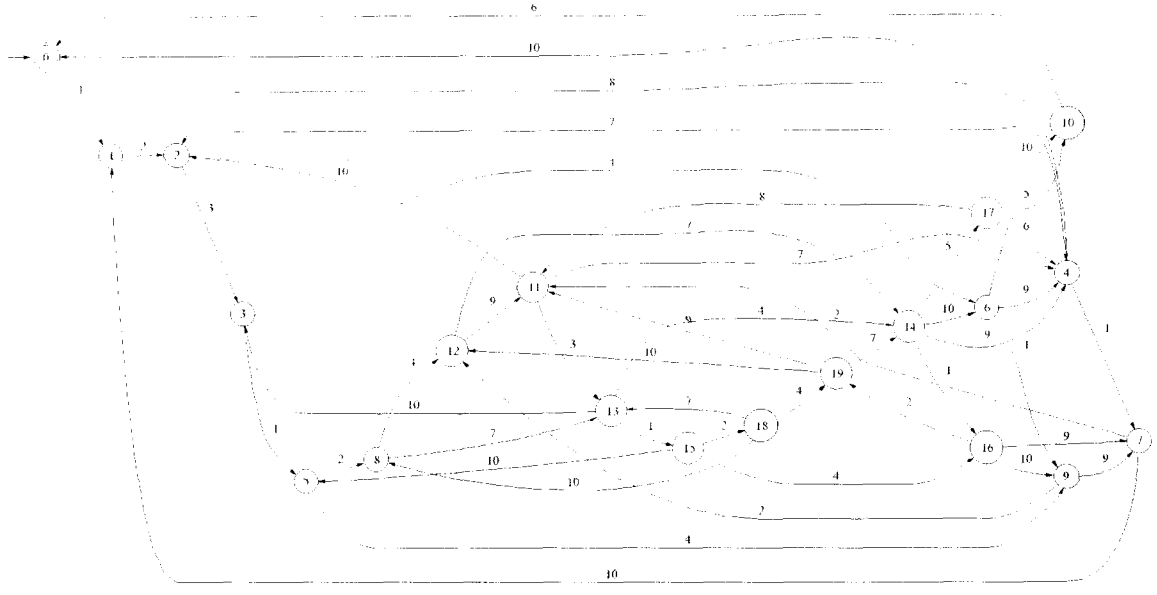


Figure 3.19: Blocking-recovery state-based representation for the case of B_1=1 and B_2=1

As a brief review, we can identify from Fig. 3.7 that state **9** is the deadlock-state before the external events were introduced in this new synthesis. But from 3.18, we can identify that from state **9** the two external events 7 and 9 produce transitions to non-absorbing or free-blocking states, recovering the system from its deadlock-state. But so far, this is partially true. This is, the BRDMA do not avoid blocking occcurrence, however, they execute a blocking recovery through the use of the external events, as shown in Fig. 3.19.
An intuitive analysis and interpretation about blocking recovery can be done from Fig. 3.20, where we can follow the state evolution of the controlled transfer line, and the status each state holds, before and after a blocking and un-blocking evolution takes place within that state representation.

So, one of our expected outcomes *was done.* The next and final expected outcome is to assure that the BRDMA lead the system from the blocking state to any other free-blocking re-initialization state, which holds a similar status for the machines, as the blocking state does, (III11), but, of course, with a different status of both buffers. The feasible and desired states are the followings: *state 1* (III00), *state 6* (III10), and *state 3* (III01). Those are desirable states that can be achieved by running the set of BRDMA along with the unblocking mechanism.

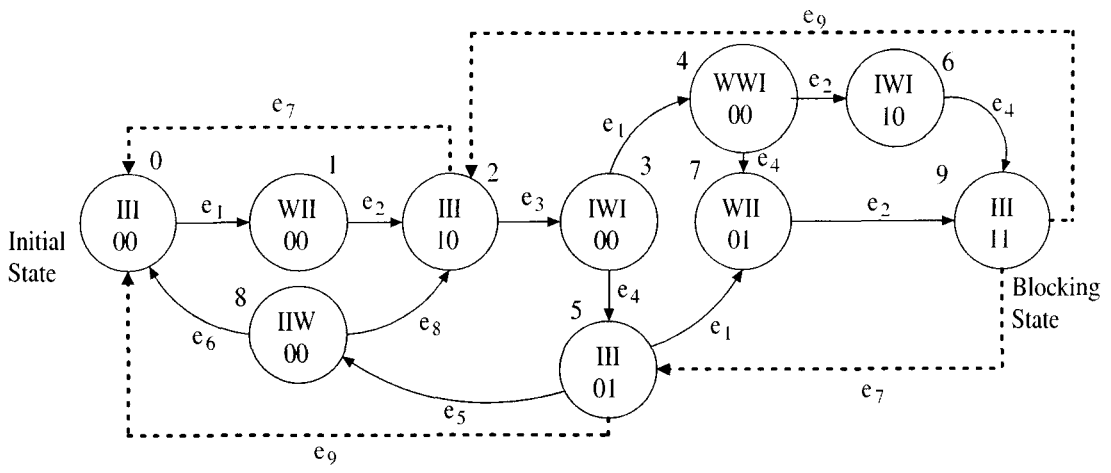


Figure 3.20: Blocking recovery for the case TLB1B1 showing possible re-initialization free-blocking states.

3.5.3 The controlled blocking-recovery transfer line as a nonblocking system

As mentioned in chapter 2.4, a system is *nonblocking* if it is both, reachable or co-reachable. Reviewing the graph from Fig. 3.18 we can check if the two properties are satisfied to consider the controlled transfer line as a non-blocking system. If so, we have completely achieved our expected outcomes. This is, to achieve the synthesis of blocking-recovery discrete modular agents, BRDMA.

- The system from Fig. 3.18 is a reachable system, because

q_0, $q \in Q_{ub}$ and $s \in \Sigma_E^*$, as well as the transition $\eta(qo, s)!$, satisfy the *reachability* condition.

Example 3.2 *For the initial state* $\mathbf{0} \in Q_{ub}$, *and string* s_1=*1231247, then* $\eta(\mathbf{0}, s_1) = \mathbf{5}$.

- The system from Fig. 3.18 is co-reachable.

By definition of co-reachability, if the marker state $q_9 \in Q_{ub}$ can be reached from any state $q \in Q_{ub}$, by a string $s \in \Sigma_E^*$, i.e., $\eta(q, s) = q_9$; then, the system is *co-reachable*. From Fig. 3.18 we can see that the initial state is also the marker (final) state.

Example 3.3 *For the state* $\mathbf{9} \in Q_{ub}$, *and* $s = 79$, $\eta(\mathbf{9}, 79) = \eta(\eta(\mathbf{9}, 7), 9) = \eta(\mathbf{5}, 9) = \mathbf{0}$.

Also, if the generated language of the automaton BRDMA, $L(BRDMA)$, is equal to its marked language, $\overline{L(BRDMA)}$, we say that the set of BRDMA represented by automaton of Fig. 3.18 is a *nonblocking system*, i.e., $L(BRDMA) = \overline{L(BRDMA)}$.

Example 3.4 *Let string* $s_2 \in \Sigma_E^*$, *where* s_2=*123124; its prefix-closure of* s_2 *is made up of:*

$$\bar{s_2} = \{\emptyset, 1, 2, 3, 4, 12, 123, 1231, 12312, 123124\}$$

So, if the controlled transfer line satisfies both properties of being *reachable* and *co-reachable*, then, we can say that the state-based representation of the set of BRDMA, *is a non-blocking system*, as an *expected outcome*.

3.5.4 Structure for Blocking Recovery

As mentioned in [30, 42, 25], to execute a blocking-recovery procedure, it is important to detect such blocking condition. In Fig. 3.21 we present the structure to execute a blocking-recovery process through the set of BRDMA and the unblocking-mechanism, UBM.

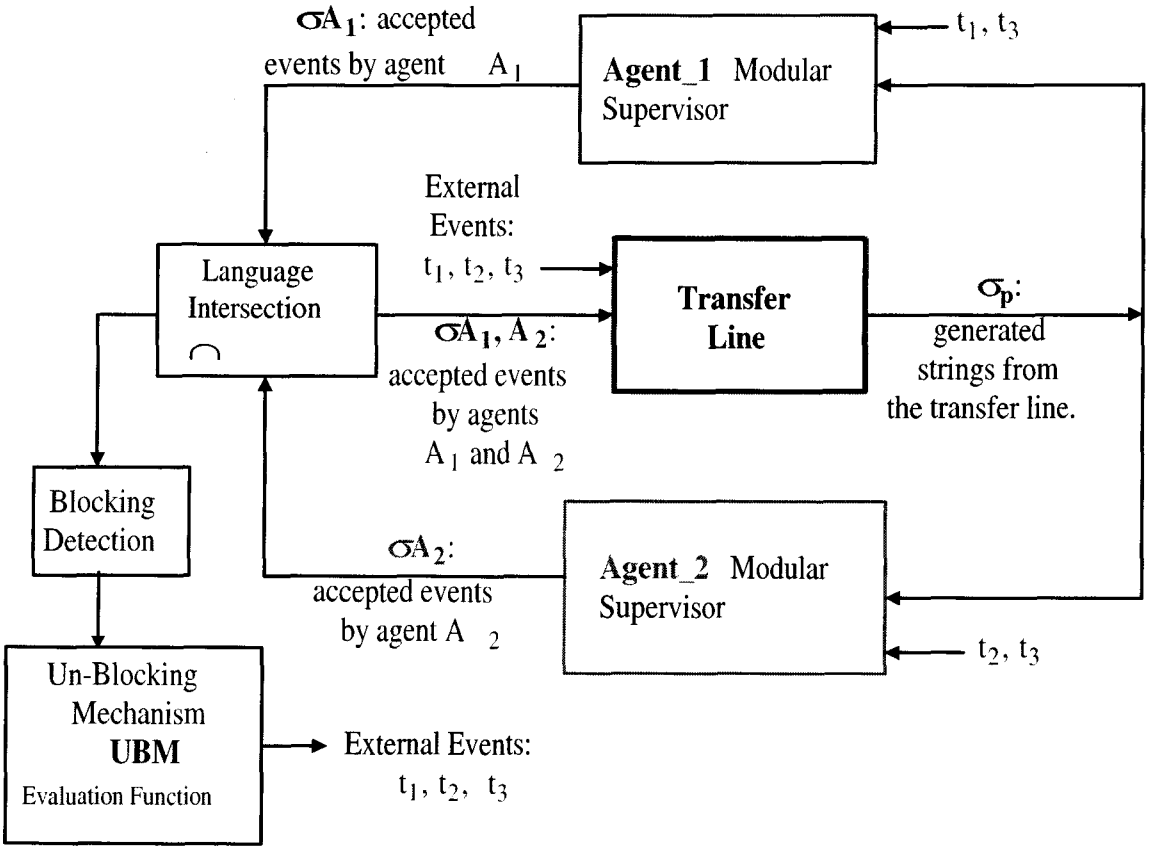


Figure 3.21: Structure of Blocking Recovery.

The structure represented in Fig. 3.21, shows that the plant only generates the events $\sigma \in \Sigma_E$, which are sent to both modular agents, Ag-1 and Ag-2. Each supervisor enables or disables such event σ, according to its own function. The intersection of the languages generated by both agents, defines the events which are accepted by both agents. If so, such event is executed into the tranfer line (plant). As each agent executes its own evolution, a combination of states

can be reached and it is said that the system has reached its blocking states. After that, a blocking recovery procedure needs to be defined. i.e., we need to select to which state we want to move from the current blocking state, by evaluating: time to block again from that state, the cost to remove unfinished parts, cost of unblocking (idle system). The time to block from different re-initializing states is computed in the proposed methods described in the next chapter.

3.5.5 Case for B_1=2 and B_2=1

A great deal of effort is saved to synthesize in a modular way one of the set of BRDM when the size of buffer B_1 is increased with an additional space, B_1=2. The only change to be done, with respect to the case of B_1=B_2=1, previously described, is into the DES model of specification for B_1SP, as shown in Fig. 3.7. Because the size of buffer B_2 is left unchanged, so, it is for its agent supervisor. Then, we only need to make a few changes to synthesize the modular-agent to satisfy the specification for buffer B_2 . Once we do this, we follow the steps according to part I and part II, as before.

Figure 3.22: State-based representation for the first modular-agent for the case of B_1=2.

Fig. 3.22 shows the state-based representation for the modular-agent to satisfy the specifications for buffer B_1 with a new size B_1=2.

Without a need to show all the steps recommended in parts I and II, we present the state-based structure of the jointed-structure for both modular-agents, for the case TLB2B1, as shown in

Fig. 3.23, where once again, a deadlock occurrence can be detected, and it corresponds to *state*17.

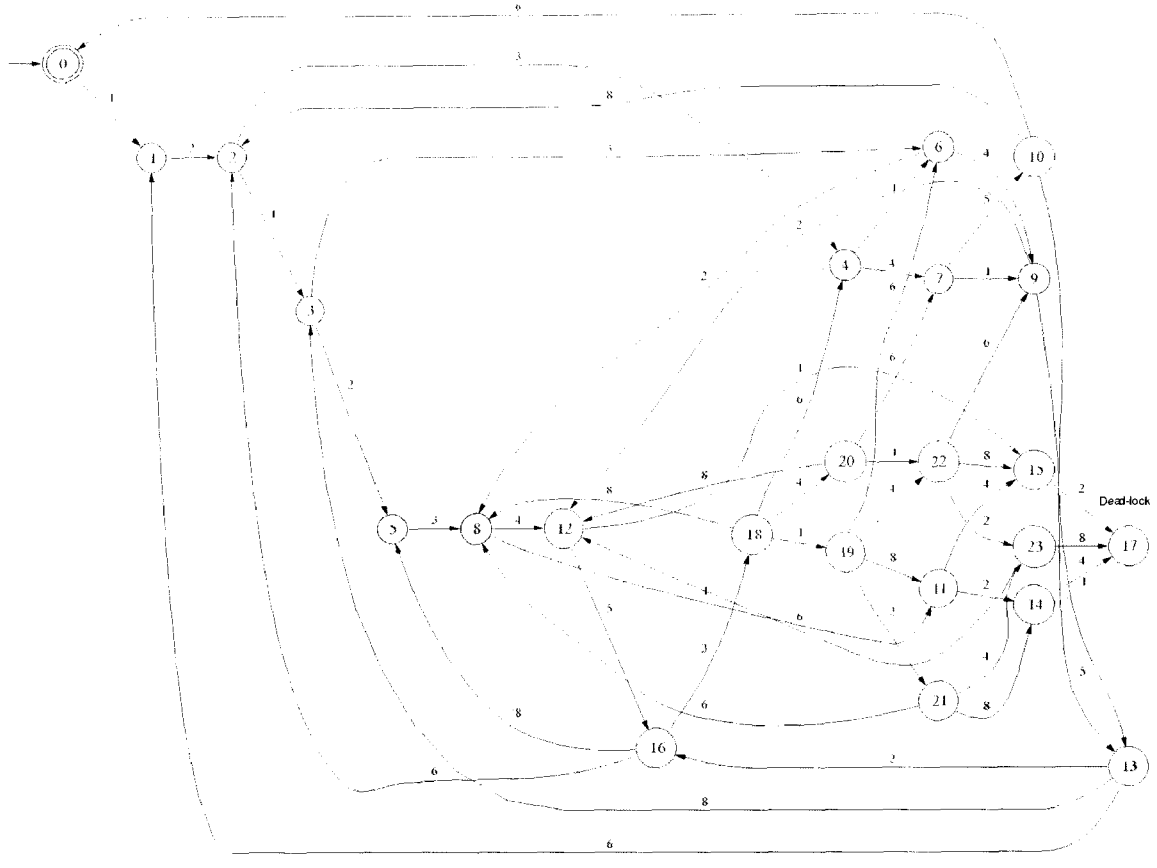


Figure 3.23: Deadlock-occurrence for the case of B_1=2 and B_2=1.

Example 3.5 *Within the case TLB2B1, a string* $s \in \Sigma_E^*$, *made up of events* $s = 12123412$ *is a possible string to occurr for the transfer line under control of the current set of modular supervisors, and such string will lead the system to reach its deadlock-state. Its state evolution is presented in the table 3.2.*

Table 3.2: State Evolution

String	ε	1	2	1	2	3	4	1	2
State	**0**	**1**	**2**	**3**	**5**	**8**	**12**	**15**	**17**(B)

Now, we can keep track the evolution of the controlled system as we execute the given string from Table 3.2 as shown in Fig. 3.23 to verify how the system reaches its blocking state.

For the case of $B_1 = 3$ *and* $B_2 = 1$, we only modify the DES model of B_1SP, to add a new empty space, and synthesize the new modular-agent to satisfy the new specification. As in the

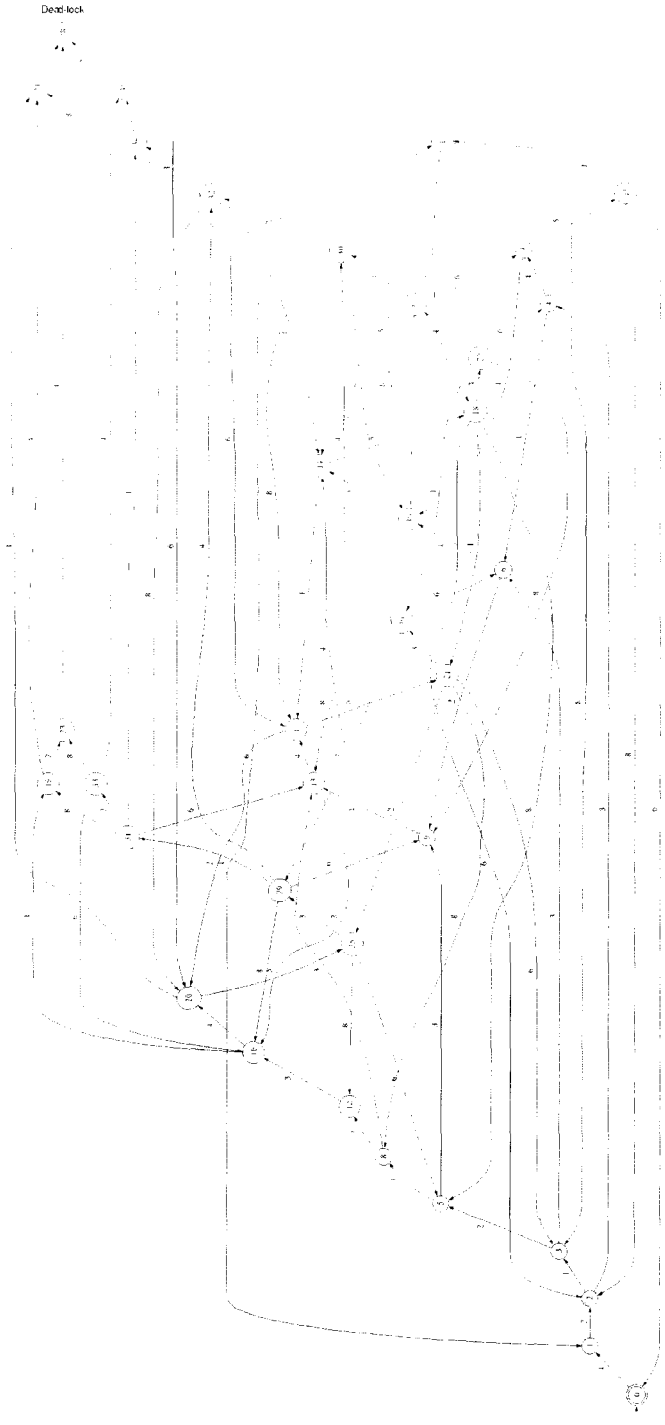


Figure 3.24: State-based representation of the jointed structure for both modular-agents for the case of B_1=3 and B_2=1.

two previous cases, we only follow the same steps as pointed out in parts I and II, as before. Fig. 3.24 shows the jointed state-base structure for the two discrete modular-agents.

From Fig. 3.24 we can detect, once again, the occurrence of a deadlock-state, which corresponds to state 28 (III31).

Example 3.6 *Within the case TLB3B1, a string* $s \in \Sigma_E^*$, *made up of events* $s = 1212123412$ *is a possible string to ocurr for the transfer line under control of the current supervisors, and such string will lead the system to reach its deadlock-state. Its state evolution is presented in the table 3.3.*

Table 3.3: State Evolution

String	ε	1	2	1	2	1	2	3	4	1	2
State	**0**	**1**	**2**	**3**	**5**	**8**	**12**	**16**	**20**	**24**	**28**(B)

Remark 3.1 *It is important to point out that, even though the transfer line works under control of the set of BRDMA, the blocking state will be reached. However, the transfer line is ready to execute its blocking-recovery process, as soon as the external events be generated and executed by the unblocking mechanism, and a blocking-recovery policy must be chosen among the feasible ones.*

Next chapter presents the method to compute the time to block from different re-initialization states, and formulate different blocking-recovery policies, to decide which state would be an acceptable option to re-initialize the blocking recovery process.

Chapter 4

Methods II and III: Quantitative Evaluation of the BRDMA in Transient and Steady-State Analysis

The next two proposed methods deal with a quantitative evaluation of two different issues of the controlled transfer line. Method II, provides a quantitative evaluation of the time to block, t_b, by the use of the statistical coefficients of the moment generator function (MGF) as we evaluate the cumulative distribution function, cdf, of the absorbing state. This evaluation is obtained in transient state analysis. Method III is a procedure to evaluate *the profit rate* for each unblocking policy, in the steady-state analysis, and make a comparison against its counter-part nonblocking solution. This last evaluation is crucial to make up our mind if our proposed unblocking policy is a good solution. It is important to point out that the proposed evaluation methods can be applied to different discrete event systems, as it may be the case of any manufacturing system with or without blocking-occurrence. However, in this case, we will concentrate only on our controlled transfer line with one absorbing state.

Methods I and II evaluate the time to block, t_b, by applying two different approaches. Both methods start considering blocking occurrence as illustrated in Fig. 4.1 for the case of study TLB1B1. As explained in previous chapter, method I allows us to obtain t_b from the graphs of the pdf of the blocking state, its median, $X_{t_b} = 0.5$. On the other hand, method II will provide the first n-th statistical moments for the time to block, t_b. Method III will evaluate the *profit rate* by first computing the the steady-state probabilities within the blocking-recovery condition.

4.1 Stochastic Framework Analysis for Method II

The proposed methods II and III are based on a stochastic process framework. So, first of all, let's define our transfer line under a stochastic framework, based on the case of study TLB1B1.

1. Let define the random variable X_t to represent the state of the transfer line at time t. Then, $\mathbf{X} = \{X_t; t \in \mathbb{R}\}$ is a continuous time stochastic process, with discrete state-space

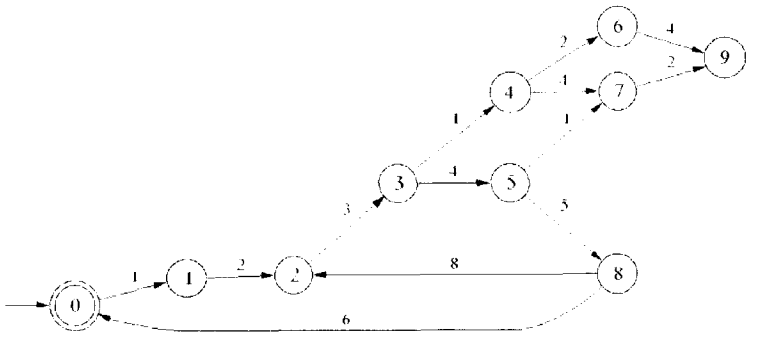


Figure 4.1: State based evolution of the transfer line for the case of TLB1B1.

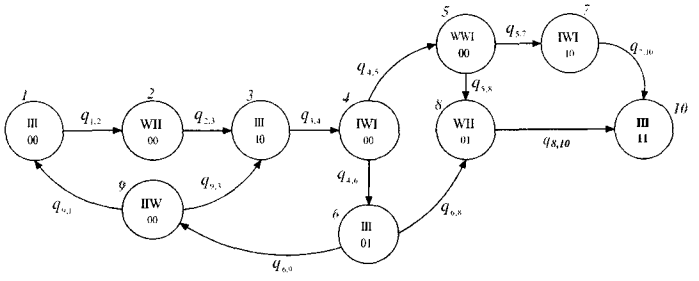


Figure 4.2: Rate based-evolution, q_{ij}, for the case of TLB1B1.

$\mathbf{E} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

2. Now, let define t_{ij} as the transition time from the current state i to the next state $j \in E$. Assume t_{ij} is an independent variable with exponential distribution with mean μ_{ij} for all states $i, j \in E$.

3. Then $\mathbf{X}$ is a Continuous Time Markov Chain (CTMC) with the state transition diagram as in Fig. 4.2.

4. Let define q_{ij} as the instant transition rate defined as follow:

5. $q_{ij} = \begin{cases} \frac{1}{t_{ij}} & \text{for all defined transitions from current state } i \text{ to state } j \in E \\ 0 & \text{other case} \end{cases}$

6. Let's call S_b for the *absorbing state*, $S_b = 9$, (for our example), and *non-absorbings* for the rest of the states.

7. Let $\mathbf{Q} = [q_{ij}]$ for all $i, j \in E$, be the *infinitesimal generator matrix.*

8. Let's define also, $\pi_i(t) = \mathbf{P}\{X_t = i\}$ for all t; and we call $\pi(t) = [\pi_1(t)\ \pi_2(t)\ ...]$ the *transient state probability vector.*

Now, if we leave the same state-based structure as in Fig. 4.1, but replacing the events σ_{ij} between states by evolutions rates between states, q_{ij}, as in Fig. 4.2, we obtain our desired model representation to apply our proposed evaluation methods.

Now, we need to obtain the magnitude of time to block, t_b, between different solutions of BRDMA, for the three cases of study, as the system reaches the absorbing state. However, t_b is a random variable, and therefore, we are interested in the probability distribution of t_b as expressed by

$F(t) = \mathbf{P}\{t_b \leq t\}$.

Proposition 4.1

Consider a CTMC $\mathbf{X} = \{Xt; t \in \mathbb{R}\}$ *with only one absorbing state Sb. Let* t_b *be the time to reach the absorbing state from an initial state. Then,*

$$\mathbf{P}\{t_b \leq t\} = \pi_{S_b}(t)\}$$

Proof. Given that S_b is the only absorbing state, then the following events are equivalents

$\{\omega | t_b(\omega) \leq t\} = \{\omega | X_t(\omega) = S_b\}$

Therefore,

$\mathbf{P}\{t_b \leq t\} = P\{X_t(\omega) = S_b\}$

but from definition of $\pi(t)$:

$$\mathbf{P}\{t_b \leq t\} = \pi_{S_b}(t) \tag{4.1}$$

From proposition 4.1, in order to obtain the desired distribution, we require, $\pi(t)$. ■

In [12, 39], $\pi(t)$ can be obtained by solving the following set of differential equations.

$$\dot{\pi}(t) = \frac{d\pi(t)}{dt} = \pi(t)\mathbf{Q} \tag{4.2}$$

where: $\pi(t)$ is the *transient-state probability vector* of the CTMC, and

$\pi(0) = \pi_0$; is the *initial state vector.* For our case, it can be the *first initial state* or any other *re-initialization state.*

Obtaining $\pi(t)$ and $\mathbf{P}\{t_b \leq t\}$

We can apply the Laplace-transform to Eq. 4.2 to obtain:

$$s\mathbf{\Pi}(s) - \mathbf{\Pi}_0 = \mathbf{\Pi}(s)\mathbf{Q}$$

$$\mathbf{\Pi}(s)[s\mathbf{I} - \mathbf{Q}] = \mathbf{\Pi}_0$$

$$\mathbf{\Pi}(s) = \mathbf{\Pi}_0[s\mathbf{I} - \mathbf{Q}]^{-1} \tag{4.3}$$

Applying inverse-Laplace transform to Eq. 4.3

$$\pi(t) = L^{-1}\{\mathbf{\Pi}_0[s\mathbf{I} - \mathbf{Q}]^{-1}\} \tag{4.4}$$

From Eq. 4.4 we obtain the transient probability of being at each state i at time t.

Now, we are going to model the case TLB1B1 based on Fig. 4.2.

$$\begin{aligned}
\dot{\pi_1} &= q_{9,1}\,\pi_9 - q_{1,2}\,\pi_1 \\
\dot{\pi_2} &= q_{1,2}\,\pi_1 - q_{2,3}\,\pi_2 \\
\dot{\pi_3} &= q_{2,3}\,\pi_2 - q_{3,4}\,\pi_3 + q_{9,3}\,\pi_9 \\
\dot{\pi_4} &= q_{3,4}\,\pi_3 - (q_{4,5} + q_{4,6})\pi_4 \\
\dot{\pi_5} &= q_{4,5}\,\pi_4 - (q_{5,7} + q_{5,8})\pi_5 \\
\dot{\pi_6} &= q_{4,6}\,\pi_4 - (q_{6,8} + q_{6,9})\pi_6 \\
\dot{\pi_7} &= q_{5,7}\,\pi_5 - q_{7,10}\,\pi_7 \\
\dot{\pi_8} &= q_{5,8}\,\pi_5 + q_{6,8}\,\pi_6 - q_{8,10}\,\pi_8 \\
\dot{\pi_9} &= q_{6,9}\,\pi_6 - (q_{9,1} + q_{9,3})\pi_9 \\
\dot{\pi_{10}} &= q_{7,10}\,\pi_7 + q_{8,10}\,\pi_8
\end{aligned}$$

And the Q instant matrix is expressed as follows.

$$Q = \begin{pmatrix}
-q_{1,2} & q_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -q_{2,3} & q_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -q_{3,4} & q_{3,4} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -(q_{4,5} + q_{4,6}) & q_{4,5} & q_{4,6} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -(q_{5,7} + q_{5,8}) & 0 & q_{5,7} & q_{5,8} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -(q_{6,8} + q_{6,9}) & 0 & q_{6,8} & q_{6,9} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -q_{7,10} & 0 & 0 & q_{7,10} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -q_{8,10} & 0 & q_{8,10} \\
q_{9,1} & 0 & q_{9,3} & 0 & 0 & 0 & 0 & 0 & -(q_{9,1} + q_{9,3}) & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -(q_{7,10} + q_{8,10})
\end{pmatrix}$$

From proposition 4.1, only for the case of one *absorbing state*, it is also the *cdf* of the *blocking time*, i.e., $\mathbf{P}\{t_b \leq t\}$. For the rest of the non-absorbing states, it corresponds to their *pdf*, i.e., $P(t_b) > t$.

Now, we obtain the statistical moments for t_b, as in [12] [39] and expressed in Chapter 2.7.

We know that, if t_b has a *cdf*, $\pi_{S_b}(t)$, then, its Laplace transform corresponds to:

$$\mathbf{\Pi}_{S_l}(s) = E[e^{-t_b s}] \tag{4.5}$$

Expanding $\Pi_{S_b}(s)$ in Taylor's series around $s = 0$, as in chapter 2.7 [39]:

$$\mathbf{\Pi}_{S_b}(s) = 1 - t_b\ s + \frac{(t_b)^2}{2!}s^2 - \frac{(t_b)^3}{3!}s^3 + ... + \frac{(t_b)^n}{n!}s^n + ... \tag{4.6}$$

Applying the expected value to each term of Eq. 4.6

$$\mathbf{\Pi}_{S_b}(s) = E[1] - sE[t_b] + s^2E[\frac{(t_b)^2}{2!}] - s^3E[\frac{(t_b)^3}{3!}] + ... + s^nE[\frac{(t_b)^n}{n!}] + ... \tag{4.7}$$

$$\mathbf{\Pi}_{S_b}(s) = a_o + a_1 s + a_2 s^2 + a_3 s^3 + ... + a_n s^n + ... \tag{4.8}$$

In general, the coefficients a_n, are represented by:

$$\begin{aligned} a_n &= \frac{(-1)^n}{n!}E[(t_b)^n] \\ &\quad and \\ E([t_b]^n) &= (-1)^n a_n\ \ (n!),\ for\ n \geq 0 \end{aligned} \tag{4.9}$$

This moment representation is little different as in [39]. The coefficients a_n for the power of s^n, for $n \geq 0$, correspond to the *n-th statistical moments.*

Example 4.1 *The* zero moment *for t_b, corresponds to $n = 0$; from Eq. 4.9:*

$$E[(t_b)^0] = (-1)^0 a_0\ \ (0!) = a_0 = 1$$

Example 4.2 *The* first moment *for t_b, corresponds to $n = 1$:*

$E[t_b] = (-1)^1 a_1\ \ (1!) = a_1$, which corresponds to the *expected or mean* value of t_b.

Example 4.3 *The second moment for t_b, corresponds to $n = 2$:*

$$E([t_b]^2) = (-1)^2 a_2\ \ (2!) = E([t_b]^2) = 2a_2$$

But, the *second moment* corresponds to the *variance,* σ^2, which can be calculated as follow [21]:

$$Var[t_b] = E[(t_b)^2] - (E[t_b])^2 = 2a_2 - (-a_1)^2 = 2a_2 - a_1^2$$

4.2 Computing t_b from different re-initialization states.

NOTE: Important points to remark to build the **Q** instant transient matrix and resolve for $\pi(t)$ from the set of first order differential equations.

1. The synthesis of each set of BRDMA, for different values of buffer B_1={1, 2, 3}, generates different number of re-initialization states.

2. The analysis for each underlying synthesis needs a particular information of: **Q** and π_0.

3. Based on a given synthesis of BRDMA, the computation of the time to block from the different re-initialization states, can be calculated following the same procedure, but providing a new initial vector state, leaving unchanged the infinitesimal transient matrix **Q**. For instance, for the case of B_1=1 and B_2=1, the set of BRDMA, generates 3 re-initialization states (1, 3, 6), as in Fig. 4.3. Don't forget that the set of *feasible re-initialization states*, are those states holding the same machine-status (IIIxy), as left in the blocking state (III11), where, both, x and y can take values of 0 and 1 , and the *feasible re-initialization states* are: *state 1 (III00)*; *state 3:(III10)*, and *state 6 (III01)*.

   The initial state vectors for the re-initialization states 1, 3 and 6 are the followings:

   π_1 = [1 0 0 0 0 0 0 0 0 0 0], to start or reinitialize at state 1 [III00].

   π_3 = [0 0 1 0 0 0 0 0 0 0 0], to re-initialize from state 3 [III10].

   π_6 = [0 0 0 0 0 1 0 0 0 0 0], to re-initialize from state 6 [III01].

4. For the other cases of the BRDMA, B_1={2, 3}, we leave unchanged B_2=1, and build the new **Q** instant transient matrix from their respective state-graph, obtained from the meet of the new BRDMA.

The procedure to resolve $\pi(t)$, and calculate the different statistical moments, still holds, as explained. To calculate the blocking time t_b, for and from the different re-initialization states, we provide the initial state vector, π_0 and a transient matrix Q, as mentioned in the previous remark.

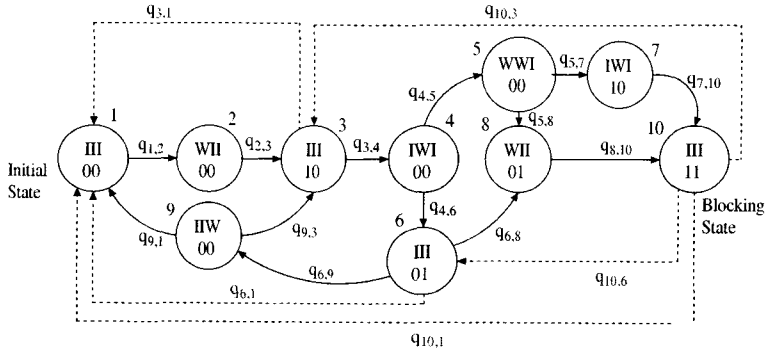


Figure 4.3: Re-initialization states for the case TLB1B1.

A procedure to follow to obtain the time to block t_b and its statistical moments, for and from different re-initialization states, is illustrated in Fig. 4.4.

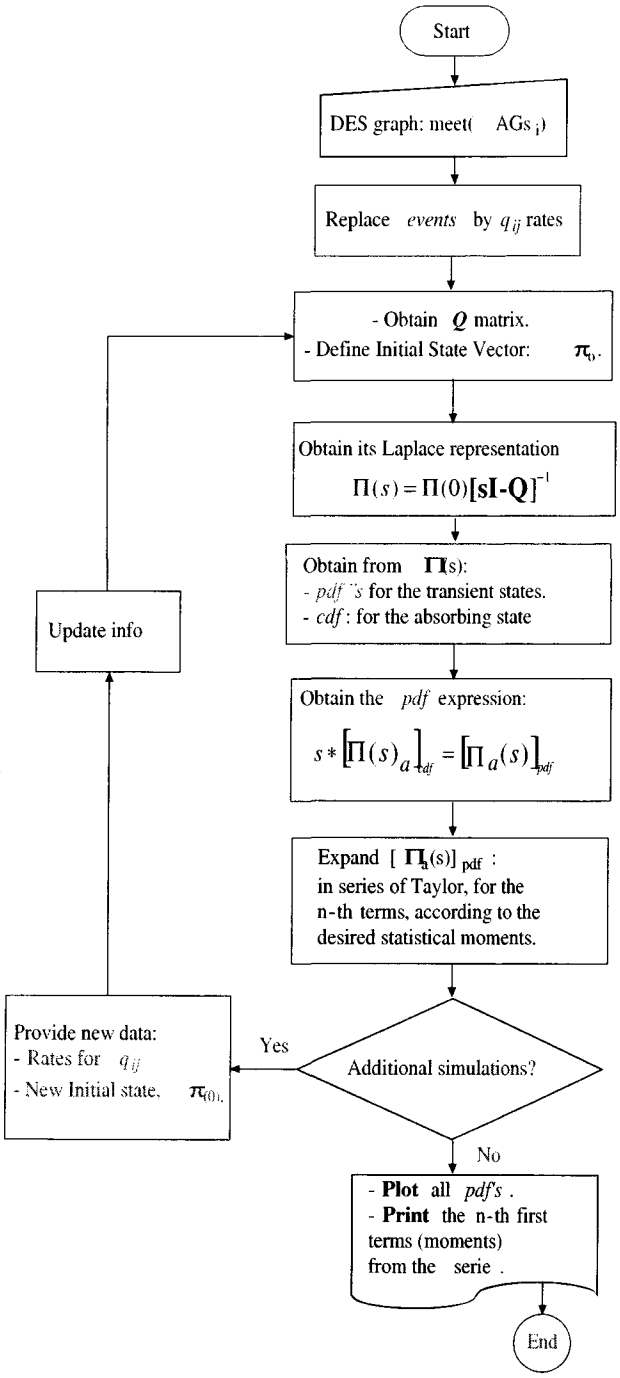


Figure 4.4: Flow diagram to calculate t_b and its statistical moments.

To analyze how the processing time of each machine affects the time to block, we provide different values for the q_{ij} rates of each machine. Also, from the same CTMC representation, we decide from which non-absorbing state we want to re-initialize the operation of the TL, i.e., we choose a new initial state condition, and we can calculate the time to block for that given resuming state.

Performing independent evaluations for different buffer sizes, but the same processing time for each machine, we can obtain a comparison of how one buffer size affects the measure of performance for the TL.

Also, a degree of *uncertainty* is included providing a *fail-probability* value c under a test into TU machine, and its corresponding *succeed probability* of $(1 - c)$.

We consider the following assumptions within a CTMC:

1. Each machine and the whole system, working under control of a set of BRDMA, behaves as a continuous time system following an exponential probability function, with multiple parameter λi.
2. The total number of states is i, where $i - 1$ corresponds to *non-absorbing states* and only one *absorbing state.*
3. The transition evolution for a given independent non-absorbing state i, evolves by independent incoming and exiting rates, with its own independent incoming flow of rates, $\sum_k q_{ki}$, toward state i; and, similarly, its own exiting flow rate, $\sum_i q_{ik}$, where k represents the *permissible states* to enter or leave a given non-absorbing state i. For the absorbing state, there are only incoming rates, but no exiting rates. For all non-absorbing states $i - 1$, except for the absorbing state, their resident time (sojourn time) follow an exponential probability function.
4. We call to q_{ij} the transition rate from state i to state j, and we generate the Instant Rate Matrix Q for our given system.
5. So, for a given instant matrix Q and a given initial state $\pi(\mathbf{0})$, we can model the whole system as a CTMC, ruled by the set of first order linear differential equations, and expressed as a vector: $\dot{\pi}(t) = \pi(t)\mathbf{Q}$, and $\pi(0) = \pi_0$

Claim 4.1 *Our task is to obtain a math-expression, named $\pi_a(t)$, that represents the behavior of the time transient probability for the absorbing (deadlock) state, which generates a cdf. We need to calculate the time to block for the absorbing state.*

Now, we can model our controlled transfer line (TL) as a CTMC, ruled by the set of first order linear differential equations, expressed as a vector-matrix form, as in [12] [39].

We focus our attention on the absorbing state, $\Pi_a(s)$ or $\pi_a(t)$, expressed in its Laplace and time domain, respectively, to obtain two main things:

1. Calculate the blocking time for the X_t random variable, and,
2. Evaluate quantitatively the performance of the TL by calculating the mean or expected t_b value from the moment generator function evaluating the expression $\Pi_a(s)$.

4.3 Method III: Profit Rate in Steady-State Analysis.

Method III is developed for the cases of study TLB1B1 and TLB2B1, and without loss of generality, it can also be applied for the case TLB3B1. For any of the three cases of study of blocking recovery, we consider that in steady-state operation, both, blocking and unblocking occur as part of the normal evolution of the controlled transfer line. Also, the blocking time and the blocking recovery time, may depend on which state we want to start or move, respectively, as re-initialization state. A feasible re-initialization state is any of those holding a status IIIxy, (where: x=0, 1; and, y=0, 1). The un-blocking process demands to choose the re-initialization state in order to apply any of the un-blocking policies as illustrated in Table 3.2. We start setting the vector representation of the balance equations, first order differential equations, for all the non-absorbing and absorbing states from the state-based representation for the of case of study TLB1B1.

Method III is a steady-state based evaluation of the profit rate for a given blocking recovery policy , based on the steady-state probabilities applying a nonblocking-state representation as in Fig. 4.3.

- step 1: Identify the different feasible unblocking options within a case of study. Each option constitutes an *unblocking policy*, LP_i.

- step 2: Define the instant transient rate matrix Q including the equivalent unblocking rates (rates from the blocking state to the re-initialization state), according to the policy selected in the previous step. Apply the state-based structure of the desired system.

- step 3: Compute the steady-state probabilities for the unblocking policy previously chosen.

- step 4: evaluate the *profit rate* of the chosen policies, by setting the unitary costs of part removal from each buffer as well as the cost of the re-initialization time.

- step 5: make a comparison between the *profit rate* for the best blocking recovery policy and its counter part of a nonblocking solution. Make up your mind, Is the blocking-recovery policy better than a nonblocking solution? If so, it's done. If not, we discard the blocking-recovery solution.

Chapter 5

Experimental Results

In this chapter we present the experimental results to evaluate quantitatively a blocking recovery behavior of a controlled transfer line, from the state-based structure of a set of blocking recovery discrete modular agents (BRDMA), which take control of such transfer line. Before we start presenting the results of this work, it is important to mention briefly, what was proposed and the expected outcomes of this research work. To achieve all of this, we developed the following main points.

1. We develop a classic synthesis of a set of discrete control entities named, discrete-modular agents (DMA), under Ramadge-Wonham framework, and fundamental ideas from multiagents. We want to achieve the following main system's goals: i) Avoid under-flow and over-flow (set of sub-tasks or specifications) into the two different buffers, and, ii) try to develop the full potential of the involved machines within the transfer line to generate a high production plan. In case of blocking occurrence, do not reject such synthesis of discrete-modular agents. Instead, we may evaluate how convenient is to deal with blocking by designing an unblocking mechanism (UBM).

2. We propose an approach to perform the synthesis of a set of control entities, playing the role of discrete modular agents, to deal with blocking occurrence in our transfer line. Those control entities are named as *blocking-recovery discrete modular agents, BRDMA*. An *un-blocking mechanism, UBM* is designed to be interpreted and executed by the modular agents. To formally verify the feasibility of the set of BRDMA to overcome blocking, we perform the formal tests of *non-blocking* and *controllability* with respect to the plant they control, under Ramadge-Wonham framework. Both, the synthesis and the tests of the set of BRDMA are performed by using the program CTCT.

3. From the state-based structure of the set of DMA (point 1), as a discrete-event system, we generate a state-based graph to detect and verify the blocking occurrence, as well as identify the *blocking-state*. From this state-graph, we can analyze how the set of DMA control the evolution of the transfer line until the system reaches the blocking state, as well as a more realistic evolution of the transfer line rather than a state-graph showing only a desired behavior to obtain.

4. From the state-based structure of the set of BRDMA, point 2, we generate a state-based graph to verify how the transfer line is led to transit through the different states, including

the blocking-state. From this state-graph we can analyze how the set of BRDMA control the evolution of the transfer line, when it transits from one state to another, rather than just modeling a state-graph representing a desired behavior to obtain.

5. We use the state-graph, from point 3, and replace the events between states, by an equivalent transition rates (sec^{-1}) following an exponential probability function for each leaving or entering rate at each state. Although, we keep the same state-based structure of the transfer line, however, the new graph represents a suitable state-based modeling to be analyzed as a finite-state and continuous time system. Continuous time Markov chains (CTMC) becomes as an important tool to be applied in our contest. Now, our state-graph is made up of one *absorbing* state and a set of non-absorbing states. To analyze a time behavior of a system by CTMC representation, we need three information data: i) a finite state-based representation, ii) an instant transient rate matrix, $\mathbf{Q}$, and, iii) an initial state probability.

6. We propose a methodology to calculate the time to block in a transient time analysis. From a vector representation for the set of states, we define a set of first-order linear differential equations or *balance equations*. Then, we transform the vector representation into its equivalent Laplace-transform. Remember that, the rate function for the absorbing state and those rates functions for the set of non-absorbing states, represent the time- probability to reach their steady state. As we know, sooner or later the system reaches its blocking state, but it is not its steady state. Then, for the absorbing state, the probability to reach blocking is equal to one. This is, $X_{t_b} = 1$, and it corresponds to its cdf (cumulative distribution function). However, for the rest of the non-absorbing states, their probability to reach blocking is equal zero, and their time behavior correspond to their pdf (probability distribution function). We solve and plot the vector representation in the time-domain of each state, obtaining the cdf for the absorbing state, and the pdf for the non-absorbing states, as expected outcomes.

7. Furthermore, we propose to evaluate quantitatively the performance of the transfer line through the use of the statistical moment generator function from the Laplace-representation of the cdf (rate function) of the absorbing state. We apply the Taylor expansion around s=0, and the $n-th$ first coefficient we obtain, represent the $n-th$ statistical moments for the time to block. So, for some given conditions to operate the transfer line, we calculate the time to block, t_b.

8. We propose an approach to calculate the steady-state probability for the transfer line working under control of the BRDMA, considering that blocking occurrence is like any other transition state, and we can execute an unblocking policy, from a set of feasible unblocking policies represented by $LP1$, $LP2$ and $LP3$.

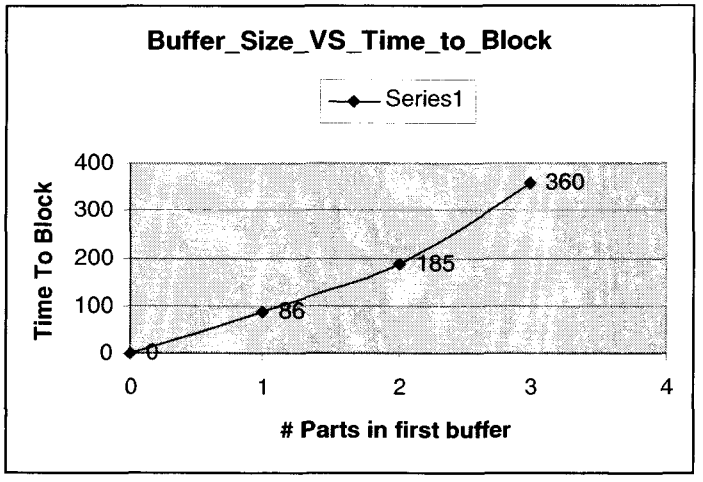
As it was mentioned in chapter 2, in this work we considered three cases of study, TLB1B1, TLB2B1 and TLB3B1 for the transfer line, where each case is different from the other, by the size of first buffer, since the second is left unchanged. Then, for each case of study, we only performed the synthesis of one BRDMA to control load/unload within the first buffer, and the second one remains unchanged, as one of the benefits to deal with a modular approach. Within each case of study, the structure of both BRDMA remains fixed and we only change the value of the production parameters, to evaluate quantitatively, how they affect the time to block t_b in transient and steady-state analysis, and the steady-state probability for each state

within the controlled transfer line. The production parameters to be changed, in each case of study, are the followings: Machines' capacities, and fail probability, p_{fail} in the test machine (TU), since the first buffer's size affects directly to t_b. The values of t_b are obtained from the plots of the *cdf* of the absorbing state, as well as from the coefficients of the statistical moment generator for such *cdf*, from its Laplace representation.

5.1 Effect of buffer size B_1 over time to block, t_b.

As we increase the size of first buffer, we also increase the number of states for the first discrete modular agent, and so, the total number of states for the composed-structure for the discrete modular agents. The more states we generate the longer time the controlled TL will spent to transit between non-absorbing states, until it reaches its absorbing (blocking) state. From the Fig. 5.1 we show the direct effect of increasing the first buffer size over time to block t_b, as we left un-changed the rates for the three cases of study. The rates used were the followings.

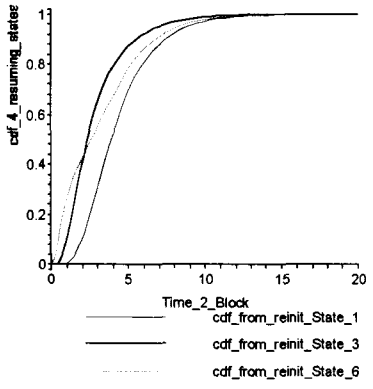
L_1= 2, L_2= 1, L_3= 2, L_4= 1, L_5= 2, L_6= 2, L_8= 2, $c = p_{fail}$= 0.01



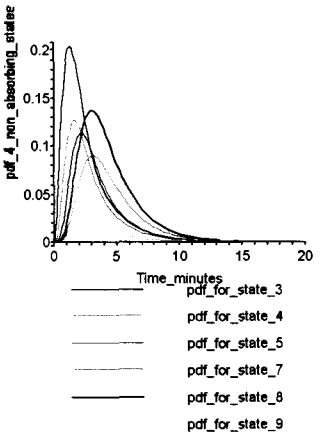
Buffer_Size	Time_2_Block (min)
0	0
1	86
2	185
3	360

Figure 5.1: Relationship between Time to block VS first buffer size.

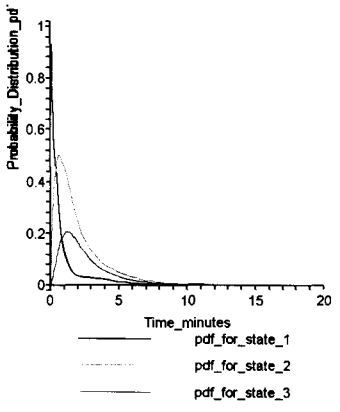
Just for the case TLB1B1, leaving un-changed the previous rates values, we show in Fig. 5.2(a) the plots of cdf's for the three possible re-initialization states, 1 (III00), 3 (III01) and 6 (III10), where state 3 reaches first its blocking state, after state 6, and finally, for state 1, it takes the longest time to reach its blocking state. Then, state 1 becomes the most desirable re-initialization state. If we calculate the statistic median, $X_{t_b} = 0.5$, from the cdf's as in 5.2(a), state 3 and 6 hold a very close median value, but state 3 reaches first its blocking state. The rest of figures, Figs. 5.2(b), 5.2(c), 5.2(d) correspond to the pdf's for all the states in TLB1B1, but initializing in state 1, showing how their probability for $t_b > t$. In Fig. 5.2(c), we observe how the pdf value for state 1 diminishes as $t_b > t$, as expected. Due to small values of some pdf's, we present them in a set of plots with different scale values. Also, from Table 5.1, we show the first five values of the coefficients from the Moment Generator Functions (MGF)



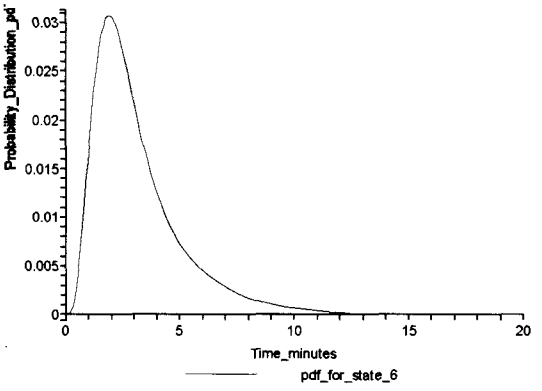
(a) cdf's from re-initialization states: 1, 3, 6



(b) pdf's for some non-absorbing states.



(c) pdf's from some non-absorbing states.



(d) pdf for non-absorbing state 6

Figure 5.2: *cdf* and *pdf's* for absorbing and non-absorbing states for the case TLB1B1.

for the three re-initialization states. Coefficient 1 gives the expected or mean value (first moment) for t_b, where state 1 holds the biggest mean value for t_b, following state 6 and state 3 in descendant order. So, state 6, becomes the second choice as re-initialization state, leaving state 3 as the last choice as re-initialization state.

Table 5.1: Coefficients from the statistics moment generator function for the case TLB1B1

TLB1B1	Coef_0	Coef_1	Coef_2	Coef_3	Coef_4
From_1	.9997561570	26.69725198	473.0516044	7034.096576	95076.76492
From_3	.9997561570	16.19981233	197.7292393	2200.540989	23584.00885
From_6	.9948792977	19.70105780	325.2166535	4738.910438	63641.74183

Table 5.2: Coefficients from the statistics moment generator function for the case TLB2B1

Re_init	Coef_0	Coef_1	Coef_2	Coef_3
From_1 (III00)	1.000000000	50.01057327	1736.370387	53468.25308
From_3 (III10)	1.000000000	39.51057328	1216.259368	35486.41680
From_6 (III20)	1.000000000	19.60121558	324.8141999	6095.863958
From_8 (III01)	1.000000000	49.70640798	1724.143447	53076.22790
From_13 (III11)	1.000000000	25.34202928	678.4552601	18854.81002

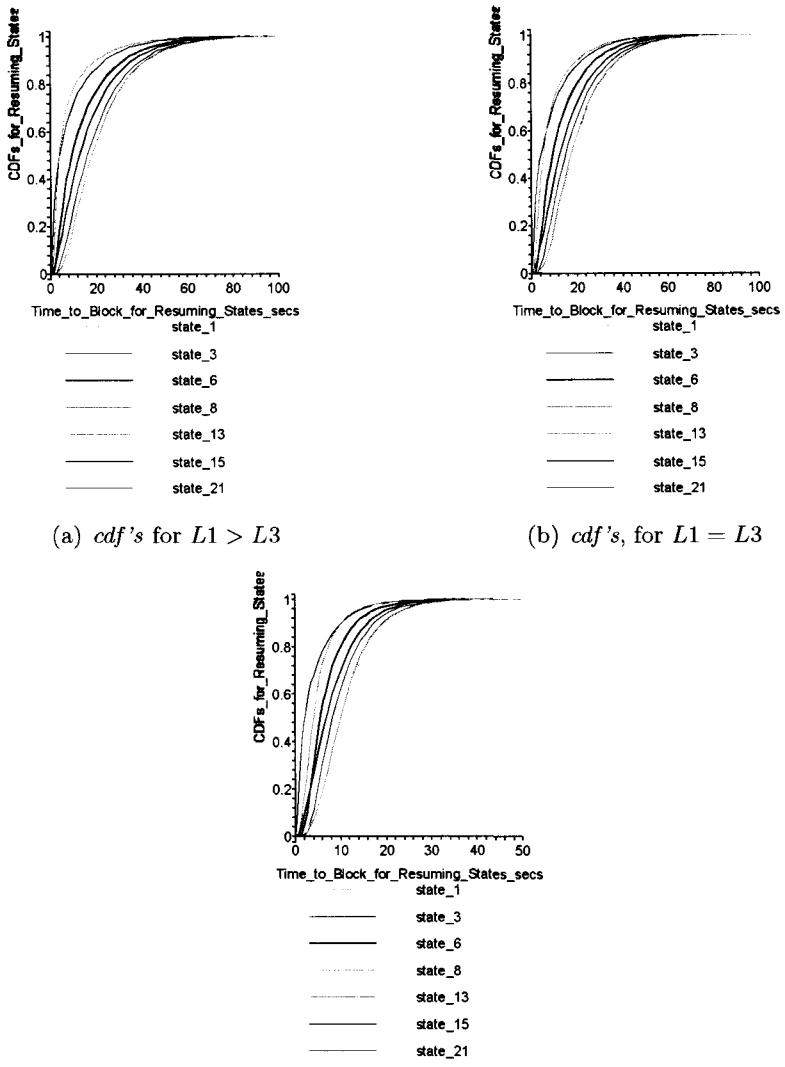
From Table5.2 we observe the coefficients from the MGF for the case TLB2B1, from the different re-initialization states. State 1 (III00) holds the biggest mean or expected value for $t_b = 50.010$, following in descendant order, state 8 (III01), state 3 (III10), state 13 (III11), and finally, state 6 (III20). So, state 1 still is considered as the most desirable re-initialization state, and state 6 with the shortest mean or expected value for t_b.

5.2 Quantitative evaluation of t_b changing the rates of the machines.

Now we use the case TLB3B1 to be evaluated, as we change the rates of machines M1 and M2, to show how they affect the value of t_b as a quantitative evaluation. As before, we use the cdf's plots and the coefficients of the MGF for all the re-initialization states. The initial rates values are the followings.

$$L_1 = 2;\ L_3 = L_5 = 2;\ L_2 = L_4 = 1;\ L_6 = L_8 = 2;\ c = p_{fail} = 0.01$$

In Fig. 5.3(a) we show the cdf's for all re-initialization states, as machine M1 has double rate of machine M2. We can observe for state 1 (III00), that it takes the longest t_b value to reach



(a) *cdf's* for $L1 > L3$

(b) *cdf's*, for $L1 = L3$

(c) *cdf's*, for $L1 < L3$

Figure 5.3: *cdf's* from re-initialization states for a change of rates of M1 and M2

blocking, and state 3 (III10) takes the shortest t_b value to reach its blocking state, among all re-initialization states, respectively. Now, under the condition of same rates for machines M1 and M2, we observe from Fig. 5.3(b) how state 1 holds the longest value for t_b, and state 13 (III11) holds the shortest t_b value, among all re-initialization states, respectively. Also, we can observe the t_b value for the rest of the re-initialization states. Also, as we consider that M2 holds double rate of machine M1, we can observe from Fig. 5.3(c) that state I (III00) still holds the longest t_b values as a re-initialization states, and state 21 (III21) holds the shortest t_b value among all the re-initialization states, respectively. Also, in Table 5.3 we show the expected or mean value of t_b, first coefficient value, only as we start from state I and for the different rate values between machines M1 and M2. From the mean or expected value of t_b, we can infer that as machines M1 holds a greater rate value than machine M2, we are going to overload very fast the first buffer, causing the controlled transfer line reaches it blocking state very soon. Following the same reasoning, we can understand the behavior of t_b as we change the rate values for the two machines M1 and M2.

Table 5.3: Coefficients from the statistics moment generator function for the case TLB3B1

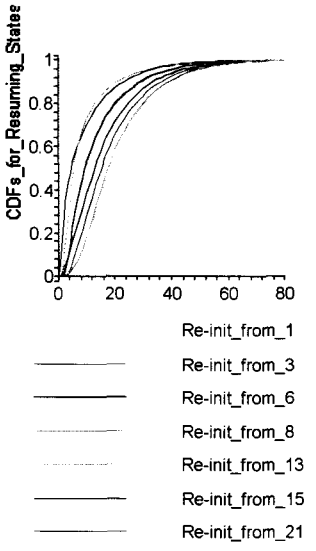
Case TLB3B1	*Median*	$coef_0$	$coef_1$	$coef_2$
$L1 < L3$	20.00	1.000000000	-23.57674912	398.1431894
$L1 = L3$	18.5	1.000000000	-10.90388593	82.88677192
$L1 > L3$	10.625	1.000000000	-11.85211027	89.00692192

5.3 Quantitative evaluation for t_b as we change the failure probability

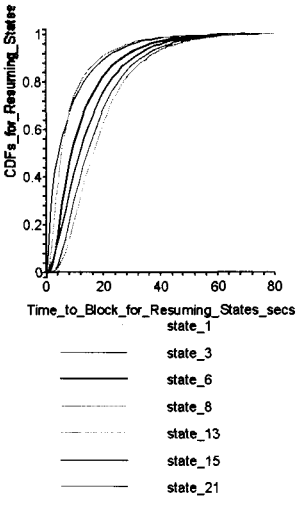
In this section we evaluate the case TLB3B1 to show how the value of t_b is affected as the machine TU (Testing Unit) rejects parts from previous machines M1 and M2, when either one or both of them, become faulty machines, causing the system a high rate of reprocessing work. The quantitative evaluation is provided from the cdf's and the coefficient of the MGF for the re-initializing states, as we use three values for the failure-probability at TU. To show how the failure probability parameter affects into the time to block, we plot the *cdf's* and the *pdf's* for the case TLB3B1. The value of the initial rates are the followings.

L_1=0.5, L_3=0.5, L_5=0.5, L_2=1, L_4=1, c=$fail_{probab}$=0.01, $L_6 = L_8 = 0.5$

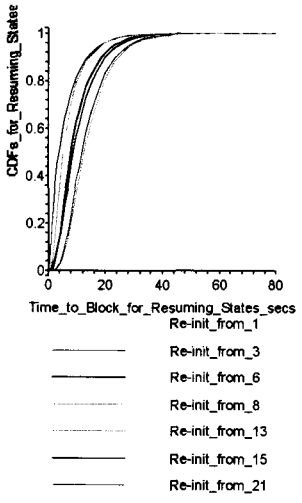
From Fig. 5.4(a) we can observe how for a failure-probability, $c = 0.01$ at machine TU, state 1 (III00) holds the longest t_b value, and state 13 (III30) holds the smallest t_b value, among all re-initialization states. Now, as we increase the failure-probability value, $c = 0.1$, we can observe in Fig. 5.4(b), that all t_b values for all re-initialization states take shorter values with respect to $c = 0.01$. But still, state I (III00) holds the longest t_b values, and state 13 III(30) holds the smallest t_b value. As an additional comment, states 13 and 21 (III21) keep a very



(a) cdf's, for $p_{fail} = 0.01$



(b) cdf's, for $p_{fail} = 0.1$



(c) cdf's, for p_{fail}= 0.55

Figure 5.4: cdf's from re-initialization states as we change p_{fail} for the case of study TLB3B1

close t_b value. For a the case that either one or both machines M1 and M2 become very faulty machines, for instance,due to a low maintenance program, $c = 0.55$, we increase the reprocessing work, causing a high feedback of parts to buffer B1, and consequently, shortening the t_b value for the re-initialization states, as shown in Fig. 5.4(c). State 1 (III00) still holds the longest t_b value, but now, state 21 (III21) holds the shortest t_b value among all the re-initialization states. It can be understood that, for a high probability-value, we feedback the system frequently, sending more parts to buffer B1, and it is shown how state 21 holds a shorter t_b value than state 13.

Table 5.4: Coefficients from the statistics moment generator function for the case TLB3B1

Case TLB3B1	*Median*	$coef_0$	$coef_1$	$coef_2$
c: fail-probab=0.01	18.812	1.000000000	-21.90210352	334.6374418
c: fail-probab=0.1	17.25	1.000000000	-20.68519560	291.4154724
c: fail-probab=0.55	15.000	1.000000000	-16.59343789	170.2645118

5.4 Evaluation of the Steady-State Probabilities for the Case of Study TLB1B1

To perform a steady-state analysis for the controlled transfer line, we need to use the BRDMA state-based structure, considering rate-transitions values between non-absorbing and absorbing states. We use the case of study TLB1B1 to illustrate method III in steady-state analysis, i.e., how we can compute the steady-state probability for each state within the controlled transfer line. Table 5.5 shows the three equivalent transition rates for the three unblocking policies, named as LP1, LP2 and LP3 from the three re-initialization states: 1, 3 and 6 based on 5.5. Each un-blocking policy requires a different rate-value, because the removal time of parts from each buffer is done at different rate. The rates represented by the L7, L9 and L10, are the equivalent external rates to unblock the system, and they made up the un-blocking policies. From the state-graph of Fig. 5.5, we easily identify the blocking state with status III11, and the equivalent un-blocking rates for the given un-blocking policies. Now, we want to evaluate the steady-state probability, $\pi_i(t)$, for each state as we apply the three different unblocking policies, leaving the same rates values for the three unblocking policies. A user can evaluate some performances such as throughput, machine-utilization, number of re-initialization for a shift, over a period of time (hours), etc. based on the steady-state probabilities. In case we want to show the effect that any single production parameter (rate) causes to any of the steady-state probability, as well as starting from different re-initialization states, we only update the rates into the instant transient matrix Q. The initial state probability vector $\pi(0)$ will always be given a probability value of one to the first initial state because the analysis is performed in steady-state.

For the three examples within case TLB1B1, we left the same production conditions as follows:

$L1$= 0.5; $L3$ =2; $L7$ =$L9$= 1; $L8$= 0.1; $L5$ = 4; $L2$= 1;
$L4$ = 4; $L6$= 0.9; $L10$= 10,and, p_{fail}= 0.10; $LP1$= $LP2$ = 0.909; $LP3$= 0.976

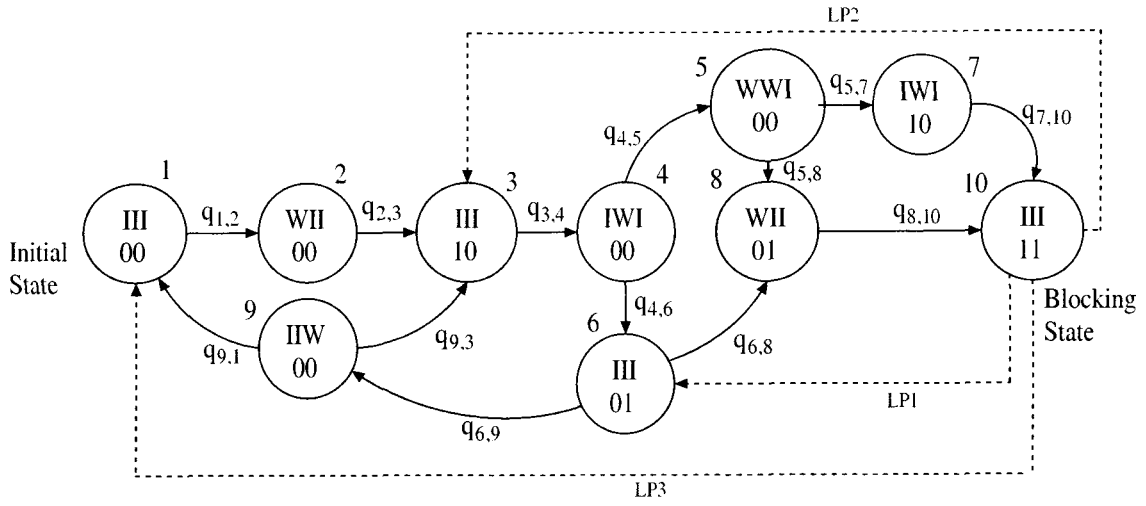


Figure 5.5: Re-initialization Policies for the case TLB1B1

Table 5.5: Unblocking policies for the case of study TLB1B1

Blocking state	Re-Initialization state	Unblocking Policy
III 11 (10)	III 00 (1)	P_3
III 11 (10)	III 10 (3)	P_2
III 11 (10)	III 01 (6)	P_1

5.4.1 Steady-state probabilities from different re-initialization states.

The proposed method III is implemented in a computer program using ©maple, since we generate and evaluate the steady-state probability matrix-representation in Laplace domain, applying the final-value theorem, as we re-initializate the system from a different state. The steady-state probability values for the three examples, are summarized in Table 5.6, and they correspond to the steady-state probabilities values for the set of plots shown in Figs. 5.6(a),5.6(b),and 5.6(c). From Table 5.6 we can observe that the lowest steady-state probability value to reach the blocking state corresponds to un-block the system from the initial state 1 (III00), and the highest steady-state probability value corresponds to un-block the system from state 3 (III10). Also, as we re-initialize the system from state 1, state 2 (WII00) holds the biggest steady-state probability value, and states 6(III01) and 9(IIW) hold the lowest steady-state probability. The total probability values is provided to verify that sum of the whole steady-state probabilities values must be equal one. Similar analysis can be obtained by the shape of the time behavior of each transient and steady-state probability for each state within TLB1B1.

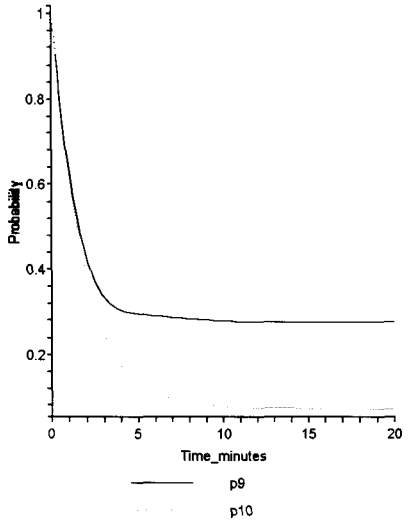
For the cases of TLB2B1 and TLB3B1 the evaluation procedure to compute their steady-state probability remains the same as for the case TLB1B1.

Table 5.6: Steady-state probabilities from re-initialization states for the case TLB1B1

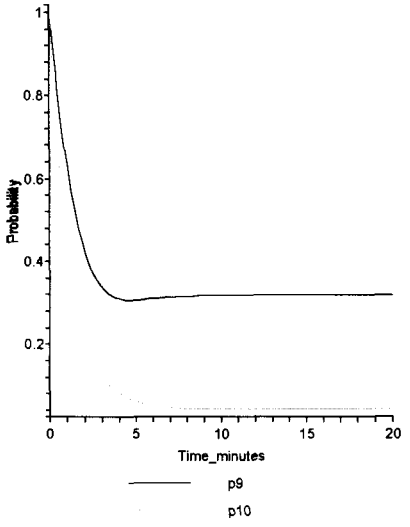
Steady-state Probab	From state 1 (III00)	From state 3 (III10)	From state 6 (III01)
State 1	0.3148948562	0.02861402880	0.2972383530
State 2	0.1574474279	0.1430701439	0.1486191764
State 3	0.08009955177	0.09254849945	0.07596091237
State 4	0.03559980070	0.04113266641	0.03376040547
State 5	0.03559980073	0.004113266641	0.003376040546
State 6	0.03164466729	0.03656237015	0.03798045615
State 7	0.0008899950186	0.001028316659	0.0008440101379
State 8	0.03006205393	0.03473425162	0.03249439025
State 9	0.2751675419	0.3179336536	0.3302648366
State 10	0.07063452520	0.04273654377	0.03946142011
Total Probability	1.000000000	1.000000000	0.9999999999

5.5 Quantitative evaluation for the case of study TLB1B1

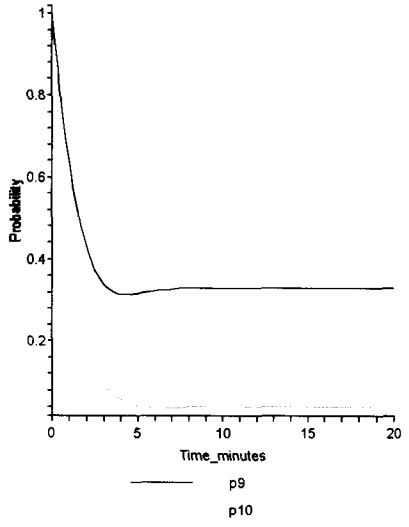
In this section we perfom two quantitative evaluation for the case of study TLB1B1. The first evaluation is to choose the best unblocking policy among the three unblocking policies. The second quantitative evaluation is a *profit rate comparison* between the best unblocking policy and a nonblocking policy.



(a) ss-probabilities from re-initialization state 1, (III00)



(b) ss-probabilities from re-initialization state 3 (III10)



(c) ss-probabilities from re-initialization state 6 (III01)

Figure 5.6: steady-state probabilities from re-initialization states for the case TLB1B1

5.5.1 Choosing the most profitable unblocking policy

In order to make a decision to choose the best unblocking policy (blocking-recovery), among the three possible ones for the case of study TLB1B1, we developed a quantitative evaluation in terms of a *profit rate*. It is based on the arithmetic difference between the gain of the throughput rate and the cost of the unblocking rate for the three unblocking policies, P1, P2 and P3.

From the Table 5.5 and Table 5.6 we show the unblocking policies and the steady-state probabilities, respectively, for the case of study TLB1B1. Two states are relevant to perform our quantitative evaluation. The first one is related to *state 9* which is the only productive state, and the second one is related to *state 10*, which is the blocking state. Table 5.6 shows that the probability of being blocked (state 10) is 7.0%, if we re-initialize the system from state 1, against 4.2% from state 3 and 3.9 % from state 6. At first glance, we could be tempted to conclude directly that the third policy is the best choice for the TLB1B1. However, we need to take into account the costs and benefits of each unblocking policy in order to make such decision and conclusion.

Now, we provide an *approach* to evaluate and compare the best profit rate among the three blocking recovery policies Pi, for $i = 1, 2and3$ (which re-initialize from state 6, 3 and 1, respectively). To perform such evaluations we define and consider the following terms: The benefit (gain) of producing a part from state 9 (named B), the unitary cost of unblocking the system from state 10 (named C_i) depending on the number of parts to be removed from any of the two buffers. There is a throughput, $T9_i$, at state 9, and a removal cost $T10_i$ at state 10. If we want to evaluate which policy is the most profitable, we compute an index named as *Profit Rate* ($\$/min$) represented by PR_i. It is defined as the difference between the gain or *Benefit Rate* (BR_i), value of the production rate, and the *Cost Rate*, CR_i, generated by the execution of each policy P_i. This is,

$PR_i = BR_i - CR_i$
The Benefit Rate, BR_i, can be obtained by multiplying the throughput rate of finished parts at $T9_i$ times the benefit of producing a finished part B, what is formulated in the following equation.

$BR_i = (B)(T9_i.)$

The throughput rate $T9_i$ can be computed by multiplying the steady-state probability of being in state 9, (π_{9-ss}), times the transition rate for event 6 (L6). This is,

$T9_i = \pi_{9-ss}(L6).$

In the same way, *the Cost Rate* to execute an unblocking policy can be given by

$CRi = CiT10_i,$

while the throughput of the unblocking action is defined as.

$T10_i = \pi_{10-ss} * LPi.$

Therefore, we have

$PRi = (B * \pi_{9-ss} * L6) - (Ci * \pi_{10-ss} * LPi)$.

For example, if we assume the unitary costs: $B = 5\$/part$ and $C3 = 1\$/unblock$, we have for policy P3.

$$\begin{aligned} PR3 &= (5\$/part)(27.5\%)(0.9parts/\min) - (1\$/unblock)(7.0\%)(0.476unblock/\min) \\ PR3 &= (1.2375\$/\min) - (0.03332\$/\min) \\ PR3 &= 1.20418\$/\min \end{aligned} \tag{5.1}$$

Table 5.7: Evaluation table for the three unblocking policies for the case TLB1B1

	P1	P2	P3
B	5 $/part	5 $/part	5 $/part
π_{9-ss}	33.00 %	31.70 %	27.5 %
L6	0.9 part/min	0.9 part/min	0.9 part/min
$T9_i$	0.297 part/min	0.297 part/min	0.297 part/min
BR_i	1.485 $/$min$	1.4265 $/$min$	1.2375 $/$min$
C_i	0.6$/$unblock$	0.6$/$unblock$	0.6$/$unblock$
π_{10-ss}	3.9 %	4.2 %	7.0 %
LP_i	0.909 unblock/min	0.909 unblock/min	0.476 unblock/min
$T10_i$	0.035451 unblock/min	0.038178 unblock/min	0.03332 unblock/min
CR_i	0.0212706 $/min	0.0229068 $/min	0.03332 $/min
PR_i	1.46 $/min	1.40 $/min	1.20 $/min

In table 5.7 we compare the Profit Rates for the three unblocking policies. From this analysis we can conclude that the policy P1 is the most profitable for the given conditions, in other words, for the case of study TLB1B1, the least expensive policy requires removing only one part from buffer 1.

Performing a *qualitative analysis* for the case of study TLB1B1, we can make a comparison between the states generated by a nonblocking solution (blocking prevention)and for the blocking-recovery, as shown in Figures 5.7 and 5.8, respectively. The blocking recovery solution generates more intermediate states than the nonblocking solution, and both structures include only one productive state. So, no benefits are achieved applying the blocking-recovery solution for this case of study. As a *conclusion* for this case of study, the nonblocking solution avoids wasting money since neither removal of parts nor unblocking cost is included.

Similarly, this is done for policies P1 and P2 within the case TLB1B1.

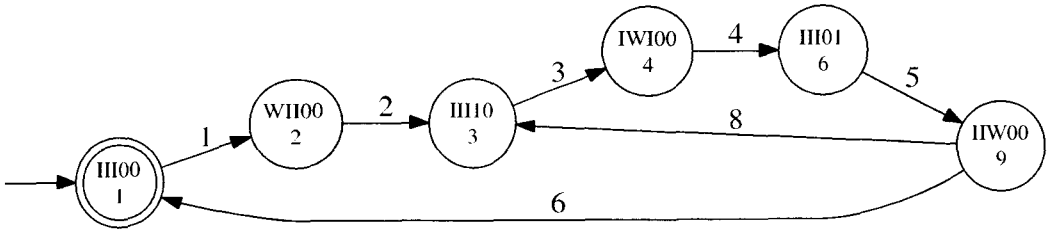


Figure 5.7: State-based structure for a nonblocking (blocking prevention) solution.

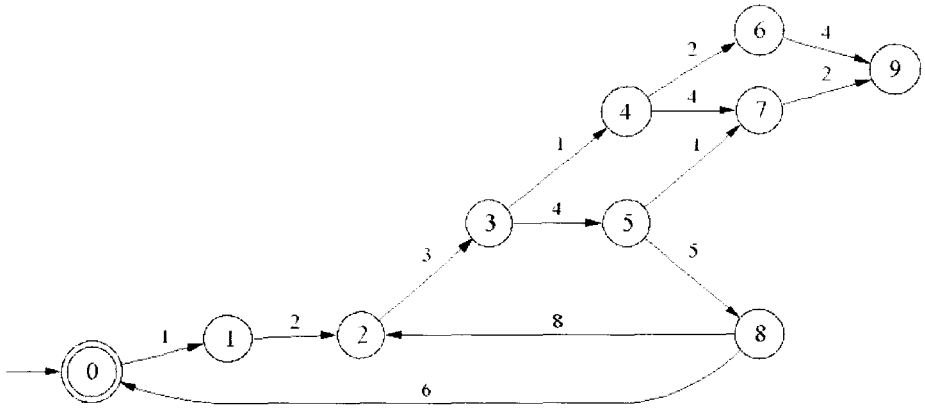


Figure 5.8: State-based structure for a blocking-recovery solution.

5.5.2 Analysis for the case of study TLB2B1

A similar evaluation of *Profit Rate* can be done for the case of study TLB2B1, for any of its five feasible unblocking policies, and its counter part nonblocking solution. The case of study TLB2B1 generates a set of 24 states. From Figure 5.9 we can observe how the whole set of states are separated in three group of states. The first group corresponds to *safe states* which generate a nonblocking behavior. The second group corresponds to *unsafe states* (19, 21, 22, 23), which are states that can lead us, under certain evolution or transition rates, to reach either the blocking state (17) or other states; and the third group corresponds to the *blocking states* (11, 14, 15, 17) which are states that unavoidable lead us to reach the blocking state (17). Also, we can verify that five unblocking policies are feasible to be executed from the blocking state (17) to the *re-initialization states* (0, 2, 5, 7 and 12). Also, this case of study generates a set of nine productive states (10, 13, 16, 18, 19, 20, 21, 22, 23). However, five of them (10, 13, 16, 18 and 20) are within the nonblocking set of states. The rest, four of them (19, 21, 22 and 23), are within the unsafe states. So, to make a quantitative evaluation of the blocking recovery for this case of study, we do the same as it was done for the case TLB1B1. We are going to select one of the unblocking policies, let say, policy LP2, as shown in Table 5.8, which means: remove two parts from buffer B_1 and execute the unblocking procedure from state 17 (III21) to the re-initialization state (III01). We need to compute the steady-state probabilities for this unblocking policy, and evalaute its *profit rate*. At first sight, we can observe that the blocking recovery policy within this case of study TLB2B1 provides four additional productive states in comparison to the nonblocking production plan. So, we need to evaluate if these four

additional productive states contribute to generate a higher profit rate in comparison to its counter part of the nonblocking solution. Now, we compute the steady-state probabilities for the the blocking recovery policies, LP2, and its counter part nonblocking solution, as shown in Table 5.9

Status	Policy	From	To	Transition Rates
III00	LP1	17	0	L_7, L_7, L_9, L_{10}
III01	LP2	17	7	L_7, L_7, L_{10}
III10	LP3	17	2	L_7, L_9, L_{10}
III11	LP4	17	12	L_7, L_{10}
III20	LP5	17	5	L_9, L_{10}

Table 5.8: Unblocking Policies

Productive State	ss-probability Blocking Recovery	ss-probability Nonblocking
(10)(IIW00)	0.0186	0.1207
(13)(WIW00)	0.0076	0.0331
(16)(IIW10)	0.0689	0.0126
(18)(IWW00)	0.0231	0.0046
(19)(WWW00)	0.0018	0.0
(20)(IIW01)	0.0472	0.6976
(21)(IWW10)	0.0003	0.0
(22)(WIW01)	0.0125	0.0
(23)(IIW11)	0.6944	0.0

Table 5.9: Steady-state probabilities for blocking recovery and unblocking within TLB2B1

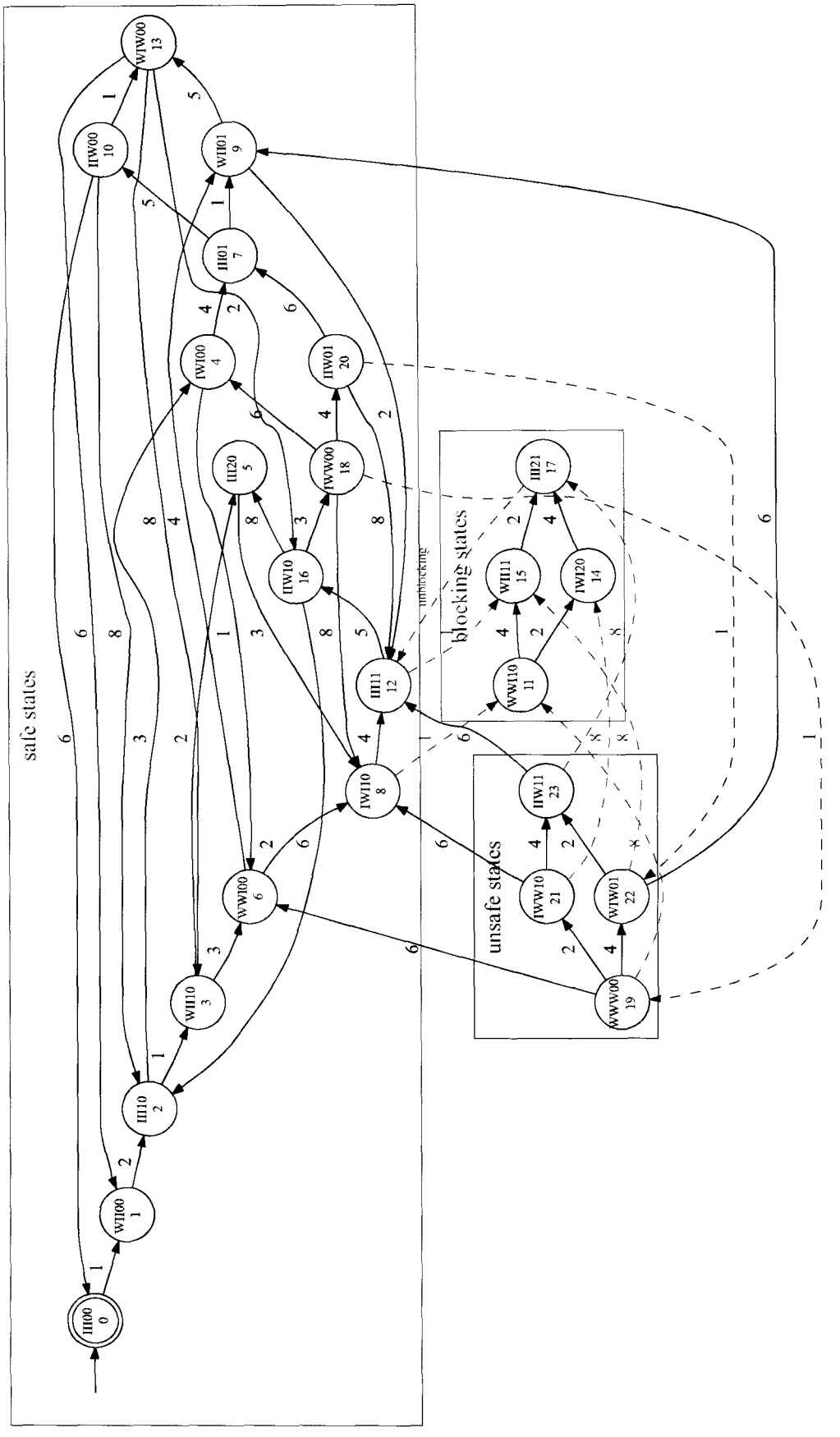


Figure 5.9: State-based structure for a blocking-recovery solution.

Profit Rate = Income Rate − Cost of unblocking rate

$$\begin{aligned}\text{Income Rate} &= 5(\$/\text{part})\text{L6}(\text{part/min}) * \sum \text{probability of productive states;} \\ &= 5(\$/part)(0.5\text{part/min})(\text{P}_{10} + \text{P}_{13} + \text{P}_{16} + \text{P}_{18} + \text{P}_{19} + \text{P}_{20} + \text{P}_{21} + \text{P}_{22} + \text{P}_{23} + \text{P}_{24}) \\ &= 5(\$/part)(0.5\text{part/min})(0.8747) \\ &= 2.186(\$/\text{min})\end{aligned}$$

Now, evaluating the cost of unblocking :

$$\begin{aligned}\text{Cost of unblocking rate} &= 0.5(\$/\text{part})(2)[\text{LP2}(\text{part/min})]\,\text{P}_{\text{blocking}} \\ &= 0.5(\$/\text{part})(2)(0.47619\text{part/min})(0.0146) \\ &= 0.006952(\$/\text{min});\end{aligned}$$

Profit Rate of Blocking Recovery = 2.179($/min);

Now, we perform the profit rate for the nonblocking counter part solution of this case of study.

$$\begin{aligned}\textbf{Profit Rate Nonblocking} &= 5(\$/\text{part})\text{L6}(\text{part/min}) \sum \text{probability of productive states;} \\ &= 5(\$/part)(0.5\text{part/min})(\text{P}_{10} + \text{P}_{12} + \text{P}_{13} + \text{P}_{14} + \text{P}_{15} + \text{P}_{16}) \\ &= 5(\$/part)(0.5\text{part/min})(0.8686) \\ &= 2.1715(\$/\text{min})\end{aligned}$$

So, as conclusion for the case of study TLB2B1 and applying the unblocking Policy LP2, its *profit rate* performs a little better than the nonblocking solution. We can evaluate the rest of the unblocking policies, LP1, LP3, LP4 and LP5, in the same way as it was done for the unblocking policy LP2.

Chapter 6

Conclusions and Future Research

Most quantitative approaches evaluate free-blocking or blocking manufacturing system from a simple state-based modeling of a desirable behavior. However, in this research we focus on performing a quantitative evaluation of a blocking-recovery manufacturing transfer line from its state-based representation, which is generated directly from the synthesis of its discrete control entities. This state-based representation shows the real state evolution the transfer line follows working under control of a set of control entities named as blocking recovery discrete modular agents (BRDMA). The set of BRDMA controls and leads the transfer line until it reaches its blocking (deadlock) state, as well as they resume the transfer line working after an unblocking policy has been defined.

The idea behind this work is that blocking (dead-lock) may occur when we want the involved machines into the transfer line develop their full potential. Although, blocking may occur in any production system, as it is the case of our transfer line, it is only an undesirable state or the worst scenario the system may reach. However, it is important to evaluate quantitatively whether or not, an unblocking recovery procedure for a given system, may generate a less conservative production plan than a free-blocking solution. This idea intends to avoid discarding blocking, at first glance, as a production plan without a previous analysis and evaluation. The approaches we presented can also be used to evaluate discrete event systems such as communication systems or computer systems.

From the state-based representation of the set of BRDMA, we only *replace* the *events* by *transient rates,* $q_{ij}(sec^{-1})$ generating a suitable modeling representation to use continuous time Markov chains (CTMC) as the performance evaluation tool. Now, we generate a matrix vector representation for the set of first order differential equation (one equation for each state) by the balance equations. The vector representation allows us to execute the quantitative evaluation in either, transient or steady state analysis.

To develop this research we support our ideas from: multiagent Systems (MAS), Discrete Event systems (DES) and Discrete Modular Supervisory theory (DMST) both under Ramadge-Wonham framework, and continuous time Markov chains (CTMC).

In this chapter we provide general conclusions about the synthesis of blocking recovery discrete multi-agents, how to compute the time to block , t_b, from different re-initialization states in transient analysis, as well as making a *profit rate evaluation* and comparison between a free-blocking and a blocking recovery production plan, both performed and evaluated in steady-

state for the transfer line. Through the three cases of study of the transfer line, we modify only the size of one of the buffers, as well as we change some of the production parameters for the processing machines to analyze and evaluate quantitatively, their effects over the performance in both, transient and steady state operation.

Further research opportunities to continue this current work are also presented.

6.1 Conclusions

General conclusions

Through the use of the three case of study within a transfer line, named as TLB1B1, TLB2B1 and TLB3B1, we have proposed and presented three complementary methods. The first one is related to the synthesize of a set of blocking-recovery discrete modular agents, BRDMA, to control and achieve a desired production behavior. The two other methods are related to perform the quantitative evaluation of the blocking recovery operation of the transfer line working under control of the set of BRDMA.

For the first method:

1. Blocking is an undesirable behavior in a production system, but it is only the worst scenario that should be explored or evaluated, but not discarded at first glance.

2. The synthesis of blocking-recovery modular-discrete-agents (BRDMA) can be a feasible solution, not a problematic solution, for some manufacturing systems that face blocking.

3. The automaton graph generated from the synthesis of modular RW supervisors can be converted directly to a CTMC representation replacing the transition events σ_{ij} by the transient rates q_{ij}. This state-based structure represents a realistic behavior of the transfer line, since its shows the main evolutions within the underlying system.

4. Modularity plays an important role to execute the synthesize of only one discrete-agent every time we changed its buffer size, allowing us to leave unchanged the second discrete-agent.

5. The proposed unblocking mechanism was crucial to make feasible the synthesis of the set of blocking-recovery discrete multiagents.

6. DES and SCT theory, under Ramadge-Wonham framework, are very helpful design tools to verify the feasibility and test of properties that the modular discrete-agents must satisfy, and the feasibility to execute the blocking-recovery within the transfer line.

For the second method:

7. Since our manufacturing system possesses an absorbing state, we know that sooner or later the system reaches this state. So, we calculate how soon the system blocks, and we obtain the time to block, t_b, as a key parameter for our quantitative evaluation.

8. We compute the time to block from different re-initialization states as well as we generate the plots of the *cdf* and the *pdf's* for the absorbing and non-absorbing states, respectively, to evaluate the mean and median t_b.

9. To calculate the time to block, t_b, we apply the Taylor expansion to the expression $\Pi_a(s)$, for the absorbing state, around $s = 0$, and select the first $n-th$ $(n = 1, 2, 3, 4, 5, ...)$ terms. Then, we obtain a polynomial representation such as: $F_a(s) = 1 + a_1 s + a_2 s^2 + a_3 s^3 + ...$ expressing the *statistical moment generator* for the absorbing state. The first $n - th$ coefficient values of $F_a(s)$ corresponds to the first $n - th$ statistical moments for the rates into the system.

10. For $a_0 = 1$ it represents the zero statistical moment. But if a_0 is not very close to one, say: $\varepsilon = 10^{-3}$, it means there is a modeling problem in the transient matrix $\mathbf{Q}$. This is, we need to review the input and output rates for each state $[q_{ij}\,] \in Q$.

11. The value for the second coefficient of $F_a(s)$ corresponds to the *mean time to block*, $a_1 = t_b$, which corresponds to a very important quantitative evaluation parameter.

12. Also, we can calculate the median time , $X_{t_b} = 0.5$, from the cdf-graphs for the absorbing state for each re-initialization state within each case of study.

13. We evaluate quantitatively how the time to block, t_b, is affected as we change the first buffer size, the machine's capacities and the test-machine failure probability within each case of study. This t_b value was obtained from the statistical moment generator function and from the median and mean value graph for the *cdf* for the absorbing-state within each case of study.

14. As expected,in the three cases of study, it takes the system a longer time to block as we re-initialize from state 1 (III00), but it does not mean to be the most profitabel unblocking policy.

For the third method:

15. The *profit rate* as a quantitative evaluation method can help us to choose not only the best unblocking policy among the feasible ones, but also, it helps us to make a quantitative evaluation and comparison between the best profitable unblocking policy and its counter-part of a free-blocking solution. This *quantitative evaluation* defines if the chosen blocking recovery policy is more profitable than a nonblocking solution.

16. For the case of study TLB1B1 the profit rate for the best blocking-recovery policy was not a better solution than its counter-part of free-blocking solution. But for the cases of study TLB2B1 and TLB3B1 it may not be the case.

17. The balance equations $\frac{d\pi(t)}{dt} = \pi(t)Q$, for transient analysis, $\frac{d\pi(t)}{dt} = 0 = \pi(t)Q$ for steady-state analysis, play a fundamental role as a math tool to generate the modeling and perform the analysis, for the blocking and blocking-recovery analysis, respectively.

18. To calculate the steady-state probability for each state, $\pi_i(t)$ for the blocking-recovery analysis, we need to provide equivalent transition rates, depending on which unblocking policy, LP_i will be executed, and we solve the vector representation of the balance equations $\frac{d\pi(t)}{dt} = \pi(t)Q$, since in steady-state $\frac{d\pi(t)}{dt} = 0$, $0 = \pi(\infty)Q$.

6.2 Future Research

The development of this thesis work required to bring and joint topics from different application areas, generating some important matters to keep moving ahead. Some of the topics to be continued in each of the different areas are the following.

1. Add different status to the machines, besides to the current Idle-Working, such as under preventive maintenance (PM), failure (F), down (D), slow and high speed (SL, HG), intermitent failure (IF), etc. in order to cover different issues as well as open new possibilities of performance analysis.

2. We can provide more intelligence to the set of agents to make decisions when the environment changes, instead of only acting as reactive agents. This is, generate intelligent agents.

3. Define new specs to add new set of modular supervisors to deal with new issues or measures.

4. Use of a discrete supervisor to act as a coordinator or moderator to resolve conflicting matters.

5. Include a new role for an agent to act as coordinator or moderator to resolve conflicting actions between agents.

6. Provide a more dynamic role to machines to act as agents.

7. Use of dynamic policies for allocation tasks to keep working all the machines in a production systems.

8. Create more complicated or cooperative production environments where some agents (machines, robots, conveyors, AGV,...) request the help of other agents (machines, robots, conveyors, AGV,...) to complete a task, by using a bid and confirmation procedure.

9. Evaluate other performances such as reliability, availability,... for a production system.

Bibliography

[1] S. Abdelwahed and W. M Wonham. Blocking detection in discrete event systems. In *Proceeding of the American Control Conference*, Denver, CO, June 2003.

[2] Knut Åkesson. *Methods and Tools in Supervisory Control Theory: Operator aspects, computational efficiency, and applications.* PhD thesis, Chalmers University of Technology, Göteborg, Sweden, 2002.

[3] Balemi and et el. Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control*, 38(7), July 1993.

[4] Amiya Bhattacharya, R. Ramesh Rao, and Ting-Ting Y. Lin. Performability analysis of non-repairable multicomponent systems using order statistics. *Parallel and Distributed Processing IEEE Conference*, pages 646–653, October 1994.

[5] G. Christos Cassandras. *Discrete Event Systems Modeling and Performance Analysis.* Richard D. Irwin, Inc. and Aksen Associates, Inc., 1993.

[6] G. Christos Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems.* IEEE Press, 1999.

[7] Max H. de Queiroz and José E. R. Cury. Modular supervisory control of large scale discrete event systems. *Workshop on Discrete Event Systems, Ghent, Belgium, WODES00*, August 2000.

[8] Max H. de Queiroz and José E. R. Cury. Synthesis and implementation of local modular supervisory control for a manufacturing cell. *6th Int. Workshop on Discrete Event Systems, Zaragoza, Spain, WODES02*, October 2002.

[9] Alan A. Desrochers and Robert Y. Al-Jaar. *Applications of Petri Nets in Manufacturing Systems, Modeling, control, and Performance Analysis.* Institute of Electrical & Electronics Engineers Press, 1995.

[10] M. Fabian and R. Kumar. Mutually nonblocking supervisory control of discrete event systems. *Automatica*, 36:1863–1869, 2000.

[11] Pia Maria et el Fanti. Event-based feedback control for deadlock avoidance in flexible production systems, June 1994.

[12] Bolch Gunter. *Queuing Networks and Markov Chains, Modeling and Performance Evaluation with Computer Science Applications.* John Wiley & Sons, 1988.

[13] John E. Hopcroft and et el. *Introducción a la Teoría de Autómatas, Lenguages y Computación.* Addison Wesley, 2002.

[14] John E. Hopcroft and D.Ullman Jeffrey. *Introducción a la Teoría de Autómatas, Lenguajes y Computación.* Prentice Hall, 1993.

[15] Adrian A. Hopgood. *Intelligent Systems for Engineers and Scientists.* CRC Press, 2000.

[16] Jiangying Hu, William Turin, and K. Michael Brown. Language modeling with stochastic automata. In *ICSLP 96 Proceedings, Fourth International Conference on Spoken Languages*, volume 1, pages 406–409, Philadelphia, PA, 1996.

[17] Bakhadyr Khoussainov, Anil Nerode, and Rajeev Motwani. *Automata Theory and its Applications*, volume 21 of *Progress in Computer Science and Applied Logic (PCS).* Birkhuser, 2001.

[18] Jean-Luc Koning. Automata for interaction protocols in multiagent systems. In *IEEE International Conference on Systems, Man, and Cybernetics, 2000*, volume 1, pages 630–635, 2000.

[19] Jana Kosecka. *PhD Thesis.* PhD thesis, University of Pennsylvania; Philadelphia, PA, 1996.

[20] Jana Kosecka and Ruzena Bajcsy. Cooperative behaviors - discrete event system based approach. *In IJACI'93, Chambery, Workshop on dynamically interacting robots*, August 1993.

[21] V. G. Kulkarni. *Modeling, Analysis, Design, and Control of Stochastic Systems.* Springer-Verlag, 1999.

[22] T. K. Kumaran, H. Cho Chang, and R. A. Wysk. A structured approach to deadlock detection, avoidance and resolution in flexible manufacturing systems, 1994.

[23] Cho Kwang-Hyun and Lim Jong-Tae. Multiagent supervisory control for antifault propagation in serial production systems. *IEEE Transactions on Industrial Electronics*, 48(2):460 –466, April 2001.

[24] M. Lawford and Walter Murray Wonham. Supervisory control of probabilistic discrete event systems. In *Proceedings of the 36 th Midwest Symposium on Circuits and Systems*, volume 1, pages 327–331, August 1993.

[25] Mark A. Lawley and Spyros A. Reveliotis. Deadlock avoidance for sequential resource allocation systems: Hard and easy cases. In *International Journal of Flexible Manufacturing Systems*, volume 13, pages 384–405, 2001.

[26] J. Müller and M. Pischel. *An architecture for dynamically interacting agents.*, volume 3. International Journal of Intelligent and Cooperative Information Systems (IJICIS), 1994.

[27] Y. Narahari and N. Viswanadham. Transient analysis of manufacturing systems performance. *IEEE Transactions on Robotics and Automation*, 10(2):230–244, April 1994.

[28] Francisco Palomera and Rogelio Soto. Synthesis of multiagents as modular supervisors in a manufacturing application. *World Automation Congress, WAC04, Seville, Spain*, June 2004.

[29] Francisco Palomera and W. M. Wonham. Internal report about dealing with blocking in modular supervisors. January, 2003.

[30] Fanti Maria Pia and MengChu Zhou. Deadlock control methods in automated manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics- Part A: Systems and Humans*, 34(1), January 2004.

[31] David Poole, Alan Mackworth, and Randy Goebel. *Computational Intelligence, a Logical Approach.* Oxford University Press, 1998.

[32] P.J. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25(1), January 1987.

[33] P.J. Ramadge and W. Murray Wonham. The control of discrete event systems. In *Invited paper Proceedings of IEEE*, number 1, page 8198, January 1989.

[34] Xi Ren and Yu Chi Ho. Models of discrete event dynamic systems. In *IEEE Control Systems Magazine*, volume 10, 1990.

[35] S. Laurie Ricker and Karen Rudie. Know means no: Incorporating knowledge into decentralized discrete-event control. In *Proceeding of the American Control Conference*, Albuquerque, New Mexico, June 1997.

[36] S. Laurie Ricker and Karen Rudie. Know means no: Incorporating knowledge into discrete-event control systems. *IEEE Transactions on Automatic Control*, 45(9), September 2000.

[37] Stuart J. Russell and Peter. Norvig. *Artificial Intelligence, A Modern Approach.* Prentice-Hall, 1995.

[38] Weiming Shen, H. Douglas Norrie, and Jean-Paul Barthés A. *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing.* Taylor & Francis, 2001.

[39] Kishor S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications.* John Wiley & Sons, 2002.

[40] K.S. Trivedi, M. Malhotra, and R.M. Fricks. Markov reward approach to performability and reliability analysis. *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1994., MASCOTS '94*, pages 7–11, January 30th–February 2nd 1994.

[41] N. Viswanadham and Y. Narahari. *Performance Modeling of Automated Manufacturing Systems.* Prentice-Hall, 1992.

[42] N. Viswanadham, Y. Narahari, and T. L. Johnson. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using petri net models. *IEEE Transactions on Robotics and Automation*, 6(6):713–723, November 1990.

[43] N. Viswanadham, R. Krishna Pattipati, and V. Gopalakrisnha. Performability studies of automated manufacturing systems with multiple part types. *Transactions on Robotics and Automation*, 11(5):692–709, October 1995.

[44] Gerhard Weiss. *MultiAgent Systems, A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.

[45] W. M. Wonham. Notes on control of discrete-event systems. Technical report, Systems Control Group Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, 2001. Revised 2001.06.08.

[46] W. Murray Wonham and P. J. Ramadge. Modular supervisory control of discrete event systems. *Math. Control Signals Systems*, 1:13–30, 1988.

[47] Li Yonghua, Feng Lin, and Hui Ling Zheng. Supervisory control of probabilistic discrete-event systems with recovery. *IEEE Transactions on Automatic Control*, 44(10):1971–1975, October 1999.