A DECISION-THEORETIC CONTROL SYNTHESIS MODEL FOR DISTRIBUTED SYSTEMS

by

ARDAVAN AMINI

A Dissertation submitted to the
Graduate School-New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Industrial and Systems Engineering
written under the direction of
Dr. Mohsen A. Jafari
and approved by

____

____

____

____

____

New Brunswick, New Jersey
May, 2007

ABSTRACT OF THE DISSERTATION

A DECISION-THEORETIC CONTROL SYNTHESIS MODEL FOR DISTRIBUTED SYSTEMS

by

ARDAVAN AMINI

Dissertation Director:

Mohsen A. Jafari

This dissertation presents a distributed control model built on a decision-theoretic framework, where intelligent beings (i.e., agents with their own controllers) are capable of utilizing their resources and functions to perform tasks in a dynamically and probabilistically changing environment. The intellectual merit of this work is to model the decision making process of agents in consideration of the non-determinism caused by: (i) probabilistic agent's behavior, i.e., probabilistic outcomes for agent's actions due to failures, faults or other natural reasons; (ii) partially unknown and probabilistic environment, due to distributed nature of the system, lack of means for full observation within an agent or within the environment, lack of full knowledge about other agents' behaviors, and due to failures and faults occurring within the environment. The main novelty of this work comes from the way the decision making process of each agent, while modulated and distributed, constantly takes the benefit of what it learns from its environment and from itself. The decision making process of each agent includes synthesizing task plans, and executing these plans.

Dedication

To my parents,

for their endless love and support

Acknowledgements

First, I would like to express my sincere gratitude to my advisor, Professor Mohsen A. Jafari, for his invaluable support, encouragement, and enthusiasm throughout this research. Without his excellent guidance, patience and perseverance during the past years, I could not possibly finish this dissertation.

I am truly grateful to Prof. Thomas O. Boucher, an exceptional scientist and teacher. Since the beginning of my studies at Rutgers University, his comments and ideas were certainly guiding me through this difficult journey.

I would like to extend my thanks to Dr. Stéphane Mocanu for his great guidance and remarks, which significantly improved the outcome of this work.

I am also greatly indebted to Dr. Tayfur Altiok and Dr. Anna Buczak for being on my committee and reviewing this work. I truly appreciate their help.

My special thanks are extended to Rutgers University and to the Department of Industrial and System Engineering for providing me an environment to study and research.

Many thanks to Farhad Jafari and Jila Farsheed for their support as well as their kindness and helpfulness throughout these years.

I would like to thank my friends Leyla Safavi, Behnam Torrei and Lisa Bodnar for their encouragement during the last year of my research.

Finally, I wish to thank my parents for their strong emotional and moral support as well as their generous love throughout my life. Their precious guidance and assistance are unquestionably beyond words.

Table of Content

Table of Figures

1. Introduction

1.1 Objective

The objective is to develop a control framework for multi-layered distributed systems with autonomous agents. Agents are minimally pre-programmed to perform their tasks in a cost-effective manner by utilizing their basic functions and through interactions with the environment. Agents' interaction with their environment may be in the form of collaboration with one or more other agents for the purpose of avoiding adverse or undesirable circumstances or achieving a favorable condition. Agents may also interact with their environment by maneuvering and selfishly altering one or more conditions of mutual interest. Except for preliminary models, the underlying behavioral models of agents and their environment are assumed to be probabilistic. Agents are subject to internal failures and there are fault conditions (unknown in advance) involving one or more agents.

1.2 Motivation & Intellectual Merit

This work is inspired by the fact that automated systems of the future (in manufacturing and service applications) will require time and cost effective responses to the dynamic changes in their environment. Pre-programmed automated systems with pre-defined & fixed normative and disruptive behavior and centralized control are too-rigid and costly to change and adapt to new environments. New distributed control paradigms with cost effective, reconfigurable and scalable solutions will be required.

Our proposed distributed control model is built on a decision-theoretic framework, where intelligent beings (i.e., agents with their own controllers) are capable of utilizing their resources and functions to perform tasks in a dynamically and probabilistically changing environment. The intellectual merit of this work is to model the decision making process of agents in consideration of the non-determinism caused by: (i) probabilistic agent's behavior, i.e., probabilistic outcomes for agent's actions due to failures, faults or other natural reasons; (ii) partially unknown and probabilistic environment, due to distributed nature of the system, lack of means for full observation within an agent or within the environment, lack of full knowledge about other agents' behaviors, and due to failures and faults occurring within the environment. *The main novelty of this work comes from the way the decision making process of each agent constantly takes the benefit of what it learns from its environment and from itself, while modulated and distributed.* The decision making process of each agent includes synthesizing task plans, and executing these plans.

Agent's goal is to establish task plans and control, and cost estimation models, so that a sufficient level of profit can be maintained regardless of the environmental conditions. The prices of tasks are calculated by each agent based on a model which it learns from its past history and through a feedback mechanism that attempts to optimize task plans and reduce the risks of previously experienced faults and failures. It does so by attempting to control the underlying conditions to the extent possible. While synthesizing task plans, agents not only deal with their own internal conditions, constraints and risks (due to internal unobservable failures and faults), but also deal with environmental conditions, constraints, and non-determinism. While executing a task, an agent may often be required to avoid number of adversary conditions and/or to ensure some favorable conditions in the environment. Since the environment is initially unknown to the agent, it must synthesize its perception model of the environment including the impact of its own control actions. This model constantly changes as the agent makes additional observation of the environment. For computational efficiency, and also due to the fact the agent's decision making must satisfy some real time control constraints, cost and benefit of new information received from the environment must be analyzed by the agent prior to any changes in the existing perception model. For an agent to evaluate its risks associated with internal but unobservable fault conditions, or environmental fault conditions, it uses an adaptive and evolutionary model for the diagnosis and avoidance of these faults conditions. Both parametric and model structural changes are included in the formulation.

1.3 Problem Definition and Approach

In traditional control synthesis problems, given a fixed plant model $\mathscr{P}$ and control specification $\mathfrak{S}$, the objective is to compute a controller $\mathfrak{C}$, such that $\mathscr{P} \wedge \mathfrak{C} \rightarrow \mathfrak{S}$. While the overall problem remains the same, there are a number of major differences between our formulation and the traditional ones. While we will discuss the detail of these differences in various chapters, here we will merely focus on the overall aspects of the problem. For one, $\mathscr{P}$ which defines behavioral model of the agent and any interaction with its environment is constantly changing. $\mathscr{P}$ is initially known to the extent that it describes the normative behavior of the agent. Disruptive behavior, such as failures of agent's components, is partially known at the best, but in any case it is probabilistic. Any part of $\mathscr{P}$ which relates to the interaction of the agent with its environment is usually unknown or only partially known, but probabilistic. Furthermore, those elements (e.g., events or transitions) of $\mathscr{P}$, which involves the environment or even the internal failures, are often partially controllable and observable. While the agent will never know the true model of $\mathscr{P}$, it must make observations from the environment so that it can build its perception of the environment. Every time that an agent interacts with the environment and executes a control action either collaboratively or selfishly, the perception model becomes subject to potential changes. This is because the rewards of and response to these actions are unknown (even probabilistically) if not experienced before. On the other hand, specification $\mathfrak{S}$ depends on the task that agent is planning to perform, and thus is not necessarily known in advance. Furthermore, $\mathfrak{S}$ depends on the type of solution that agent intends to take. For some solutions, the agent may be required to interact selfishly or collaboratively with the environment, which adds additional layers of requirements to $\mathfrak{S}$. Finally, control $\mathfrak{C}$ is synthesized on the basis of minimizing the overall cost for an agent while satisfying the required specifications. Cost for an agent includes not only the operational cost but also the cost of learning about the environment and the risks and penalties involved with internal or external failures.

Generally speaking, the control synthesis problem posed here can be formulated as a Markov Decision Process provided that the underlying models were known. Thus we attempt to formulate and solve these problems using heuristic search and machine learning techniques. While in this thesis, we will work on many different aspects of the above synthesis problem in a distributed system, we will only briefly tackle the problem of fault detection and avoidance. For more results on this, we will refer the reader to a companion dissertation work [Zhao 06].

1.4 Background and Literature Review

1.4.1 Agent

Many different definitions for the word “agent” have been introduced in the literature, each hoping to explicate a specific point of view [Franklin 96]. [Maes 95] provides the following definition: *Autonomous agents are computational systems that inhabit some complex dynamic environment; sense and act autonomously in this environment and by doing so realize a set of goals or tasks for which they are designed*. Another definition is given by Russell and Norvig [Russell 03], which states that an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

The word *autonomous* usually refers to the fact that agent’s decisions are rather relying on its own perception of the environment and not on the facts given to it at the design time. A *rational agent*, also called an *intelligent agent* [Vlassis 03], is an agent that attempts to make the best decisions based on a given performance measure. On the other hand a *computational agent* is an agent that tries to perform/solve a specific task and is implemented on a computational device [Vlassis 03].

In another categorization we can find three types of agents: *reactive* or *reflex* agents, *deliberative* or *goal-oriented* agents and *collaborative* agents [Bigus 01]. Reactive agents respond only to external stimuli and the information available from their sensing of environment (present state of the environment) [Brooks 86]. With this approach the agents do not require to revise their perception of their world as it changes. They show emergent behavior, which is the result of the interactions of these simple agents. However, purely reactive systems suffer from two main limitations: First, because purely reactive agents make decisions based on local information, they cannot take into consideration non-local information or predict the effect of their decisions on global behavior [Sycara 98, Thomas 98, Huberman 88]. Secondly, the relationship between individual behaviors, environment, and overall behavior is not understood, which necessarily makes it hard to design agents to fulfill specific tasks [Sycara 98]. Goal-directed agents have domain knowledge and the planning capabilities necessary to take a sequence of actions in the hope of reaching or achieving a specific goal. Collaborative agents work together to solve big problems (cooperating agent community). Each individual agent is autonomous. These agents can solve problems by collaboration and synergy. Problems will be parsed into smaller chunks that can be solved by a modular approach. The approach is based on specialization of agent functions and domain knowledge. [Bratman 87] introduced *agents with beliefs, desires and intentions* (BDI). What an agent believes to be true is the basis for all of its reasoning, planning and actions. When an agent reasons about the state of the world (beliefs) and its desires (goals) it must decide what course of action to take (intensions). The objective of designing intelligent agents is to achieve specific tasks automatically. An intelligent agent works based on *events*, i.e. if any specific event happens the corresponding agent will react accordingly in order to satisfy a predetermined objective by taking some special *actions* under different conditions (states). When an event occurs the corresponding agent must recognize it and respond to it [Bigus 01].

1.4.2 Multiagent Systems

First we need to introduce two concepts that are widely used in multiagent systems, namely *organization* and *structure*. An *organization* provides a framework for agent interactions through the definition of roles, behavior expectations, and authority relations. A *structure* gives each agent a high level view of how the group solves problems [Sycara 98]. Multiagent Systems (MAS) are systems in which many intelligent agents interact with each other. Their interaction can be either cooperative or selfish [Durfee 89a]. This means they can either pursue a common goal like an ant colony or they can have their own individual goals like being in an economic market environment.

[Howarth 04] defines MAS by: “Agents are small software programs that communicate with each other, acting behaviorally to interact and respond, matching available resources to demand. In a multiagent system, each agent communicates with the network of agents, considering options for matching its capabilities with demand, negotiating on such constraints as quality, price and time, and then making decisions for committing resources to match demand.” MAS is usually an “*Open System,*” meaning that the structure of the system itself is capable of dynamically changing. The characteristics of such a system are that (1) its components are not known in advance; (2) can change over time (plug and play); and (3) can consist of highly heterogeneous agents implemented by different people, at different times, with different software tools and techniques [Sycara 98].

There are many fundamental aspects that characterize multiagent systems which distinguish them from single agent systems. These characteristics have different dimensions and therefore different authors categorize them differently in the literature. Vlassis, for example, introduces these characteristics in [Vlassis 03] as “Design,” “Knowledge,” “Perception,” “Environment,” “Control” and “Communication” and compares them with the single agent systems. *Design*: Software agents, also known as “softbots,” will be designed differently in a MAS, since they will be implemented by different people. Such agents are called heterogeneous in contrast to homogeneous agents. Homogeneous agents are designed in an identical way. They basically have the same capabilities. On the other hand heterogeneity can affect all the functional aspects of an agent, like its decision making process, where as in homogeneous environment this problem does not exist. *Knowledge*: In a MAS the levels of knowledge of every agent about the current state of the environment can differ. Each agent must consider the knowledge of every other agent that is involved in the decision making process. *Perception*: The collective information that reaches the sensors of the agents in a MAS is typically distributed. The agents may observe data that differ spatially, temporally or may be semantically (different interpretations). This makes the world state partially observable to each agent. *Environment*: Agents have to deal with environments that can be either static or dynamic. Most techniques in AI are designed for stationary environments, because they are easier to handle. In MAS, because of the existence of many agents, the environment appears dynamic from the point of view of each agent. *Control*: Control in MAS is distributed and not centralized. The decision making of each agent lies to a large extent within the agent itself. In a cooperative environment distributed decision making results in asynchronous computation and may be speed ups, but appropriated coordination mechanisms need to be additionally developed. *Communication*: Interaction between agents are associated with some form of communication. In MAS agents can potentially be both senders and receivers of messages. Communication can be used in coordination or negotiation. It also raises the issue of different protocols and common language.

Sycara [Sycara 98] on the other hand, describes the characteristics of MAS as: (1) each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint; (2) there is no system global control; (3) data are decentralized; and (4) computation is asynchronous.

From another point of view we can characterize MAS by (1) In an multiagent environment, the agents are usually *interoperating*. In other words they are able to coordinate with each other in peer-to-peer interactions. Such functions will require techniques based on negotiation or cooperation [Jennings 98, O'Hare 96, Bond 88].

Therefore another definition for a MAS can be a loosely coupled network of problem solvers that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver [Durfee 89a]. (2) Higher *Computational efficiency.* (3) Higher *reliability* due to the fact that agents can cover for each other. (4) The number of agents and their capabilities can be changed easily; therefore *extensibility* will be another important feature of a MAS. (5) Since appropriate information is exchanged among agents, system can tolerate uncertainty and therefore a MAS is *robust*. (6) Because of *modularity* a MAS is easier to *maintain*. (7) MASs are reusable. Specific agents with certain functionalities can be reused in different teams to solve different type of problems. (8) They are *flexible*. Agents with different abilities can adaptively organize to solve the current problem [Sycara 98, Huhns 97, O'Hare 96, Wooldridge 95, Chaib-draa 92, and Bond 88].

In order for MAS to solve common problems coherently, the agents must communicate amongst themselves, coordinate their activities. Coordination and communication are central to MAS, for without it, any benefits of interaction vanish and the group of agents quickly degenerates into a collection of individuals with a chaotic behavior [ASAP 05].

MAS researchers develop communications languages, interaction protocols, and gent architectures that facilitate the development of multiagent systems [AAAI 05]. Multiagent systems have been widely used in Negotiations, Task Allocation Problems and Modeling other agents for the sake of conflict resolution or state identification. Some of these categories will later be discussed in different chapters of this dissertation.

1.4.3 Synthesis

The control synthesis for both centralized and distributed systems have been extensively discussed and analyzed within the discrete-event system control-theoretic framework. Pinzon et al. have a very comprehensive study of different methods in [Pinzon 99]. Classical methods include techniques by Petri Nets [Petri 62, Jafari 95], supervisory control theory developed by Ramadge and Wonham [Ramadge 87a, Ramadge 87b] control synthesis via net condition/event systems [Krogh 96, Rausch 95] and time transition models (temporal logic). These works are model-based, synthesizing the complete controller from specifications provided as input. Should the rules or the underlying processes change, the whole synthesis algorithm must be run again in order to synthesize the control model. This is in contrast to our approach where we synthesize control actions only when needed. Here, we describe each of these methods briefly. A comprehensive description of these methods can be found in [Pinzon 99].

Ramadge and Wonham [Ramadge 87a, Ramadge 87b] framework uses formal languages and finite state automata in order to design the controller. First the plant model $\mathcal{P}$ is obtained by describing the process in terms of a formal language. This language is obtained from a finite automaton whose alphabet consists of a finite set of events. Controller $\mathcal{C}$ is then synthesized by identifying those controllable events that must be disabled according to the input specification $\mathcal{S}_{\mathcal{P}}$ at certain states. This formalism provides solutions for the *state-avoidance problem* and the *string-avoidance problems*.

Similar modeling frameworks have also been used by Alpan [Alpan 97] and Darabi [Darabi 00].

Timed Transition Models (TTM's) are an extension of state machines. They were introduced by Ostroff [Ostroff 89a, 89b] in describing a plant model $\mathscr{P}$ for *real time* discrete events systems. In this framework specifications are described using *Real Time Temporal Logic* (RTTL). In order to develop the controller $\mathscr{C}$, the "unsafe" states are identified and a set of sufficient conditions are obtained under which the unsafe behavior can be made safe. The process works based on a step by step backtracking of states which could bring the system to an unsafe state. To guarantee safe behavior, a control strategy is devised, which could change the enabling conditions of some of transitions in the system. This control strategy is sufficient but conservative in that some unreachable unsafe state may also be considered in the backtracking process. Pinzon [Pinzon 01] developed control synthesis solutions for sequential specifications using temporal logic.

Petri Nets were first introduced by C. A. Petri [Petri 62]. Petri Nets can describe many of the characteristics of discrete event systems. They are mathematically well defined and developed and are also very practical. They can graphically describe complex real systems. Using Petri Nets we can analyze systems both qualitatively and quantitatively. [Giua 92] developed a procedure for safety controllers using "monitors." This method is fairly similar to the method introduced in [Yamalidou 96]. In their synthesis procedure an uncontrolled plant was first constructed. Desired marking constraints were described by a set of generalized mutual exclusion constraints. A constraint is enforced by a new place called "monitor."

Net condition event systems (NCES) are extensions of Petri Nets. They are used in the controller synthesis method introduced by Hanisch and Rausch [Rausch 95]. In the general model of Condition Event Systems, modules described by their input/output behavior are interconnected by means of two signals: (1) piece-wise constant "condition signals," and (2) point-wise non-zero "event signals." Condition signals connect states from one module to the states in other modules. Hence, state transitions in other modules can be enabled or disabled by condition signals. Event signals provide information about state transitions in one module. They are used to force state transitions in other modules if these state transitions are enabled.

[Han 03] introduced a formal methodology that works on the idea of reusing of off-the-shelf software components to ensure system properties at design time (controller design), and automation of software design process (automatic generation of the controller). They proposed a hierarchical plant tree architecture and a systematic controller synthesis for manufacturing cells. The structure is partitioned into three class diagrams, namely the *static structure*, which is modeled by class diagrams of Unified Modeling Language (UML); the *dynamic structure*, which is modeled using Petri nets and the *functional logic*, which is modeled using rule sets.

1.4.4 Fault Detection and Autonomic Computing

We can view the problem of fault detection and diagnostics from two different perspectives: a control engineering perspective and a computer science perspective.

In the literature there are some classical views to the problem of fault detection in the discrete event dynamic systems. Lin [Lin 94] has formally defines the *diagnosability* and studies its properties. Examples of this classical approach can be found in [Sampath 95, Zad 99]. [Sampath 96] introduced a discrete event systems approach to the failure diagnosis problem. This approach is applicable to systems that fall naturally in the class of discrete event systems (DES), moreover, for the purpose of diagnosis, continuous-variable dynamic systems can often be viewed as DES at a higher level of abstraction. They present a methodology for modeling physical systems in a DES framework. [Sampath 01] also presents a hybrid approach to failure diagnosis that integrates the qualitative discrete event system diagnosis methodology with quantitative analysis based techniques. [Srinivasan 93] uses backtracking. In this approach the path from the current state to a faulty state is defined by solving sets of equations and inequalities obtained from the underlying discrete-event/continuous-time dynamical system.

IBM has recently introduced the idea of *autonomic and proactive computing*. Internet has fueled the growth of computing applications and in turn the complexity of their administration. The idea is to design a network of organized, "smart" computing components that give us what we need, when we need it, without a conscious mental or even physical effort [Karp 02]. We can ask ourselves what if when all systems made a mistake in the planning, they learned from that error and built the results of that learned experience into their next series of decisions. The vision is to solve some of the problems using principles of system design to overcome current limitations. These principles include the ability of systems to self-monitor, self-heal, self-configure and improve their performance. Furthermore, systems should be aware of their environment, defend against attack, communicate with use of open standards, and anticipate users actions [Want 03].

Formally, the phrase “autonomic computing” is used to describe systems that can configure, protect, optimize and heal themselves without a lot of input from the humanware that has until now been required to keep them up and running. (The term “autonomic,” comes from the autonomic nervous system found in mammals and other higher order creatures - basically those necessary body functions that we don’t have to think about in order to perform.) Autonomic computing has two simple rules, namely *discover* and *twiddle*. Our environment changes dramatically, sometimes in unexpected manner. An autonomic storage management system should perpetually scan for events, diagnose problems down to their root cause, and respond in the most efficient fashion. As the environment changes, the system should learn to anticipate the impact of those changes, and, as it gets smarter, it should learn to proactively intervene so as to preempt negative events.

This idea is more used in computer science framework and especially in network systems. But we think that it can be used in our framework too. A control engineer cannot predict all the possible states of the system beforehand. With a high likelihood the controlled system may face situations, which was not planned by the designer. The system should be capable of learn from the new experiences and avoid those situations in the future.

One of the approaches to deal with autonomic computing is the idea of Recovery Oriented Computing (ROC) [Patterson 02]. Recovery Oriented Computing (ROC) takes the perspective that hardware faults, software bugs, and operator errors are facts to be coped with, not problems to be solved. By concentrating on Mean Time to Repair (MTTR) rather than Mean Time to Failure (MTTF), ROC reduces recovery time and thus offers higher availability. Since a large portion of system administration is dealing with failures, ROC may also reduce total cost of ownership. One to two orders of magnitude reduction in cost means that the purchase price of hardware and software is now a small part of the total cost of ownership.

Another approach is the one presented by Gruschke [Gruschke 98]. The idea is to use an event management to condense events to meaningful fault reports. This severe practical need is addressed by *event correlation*, which is an area of intense research in the scientific community and the industry. He introduces an approach for event correlation that uses a dependency graph to represent correlation knowledge. It deals with the complexity, dynamics and distribution of real–life managed systems. That is why it is considered to provide integrated event management. The basic idea is to connect the event correlator to a given management system and gain a dependency graph from it to model the functional dependencies within the managed system. The event correlator searches through the dependency graph to localize managed objects whose failure would explain a large number of management events received.

1.4.5 Problem of Deadlock

A deadlock occurs, because two or more components require the same resources during a particular time period in a circular manner. Computer scientists have a different approach to deadlock problem than the one taken by control engineers. In our work we plan to integrate these approaches to a point that an efficient and practical solution can obtained.

Prevention, *detection/recovery*, and *avoidance* are three strategies that address deadlock. Prevention is the method where the designer breaks the circular wait offline (during the design time). Detection/recovery methods use a monitoring mechanism and a resolution procedure for preemption of some deadlocked resources. In avoidance methods the designer tries to prevent circular waits in a dynamic manner by an appropriate operational control.

Viswandadham, Narahari and Johnson [Viswanadham 90] consider a flexible manufacturing cell as a combination of processes and resources. They assume that parts are the processes and the machines, buffers, conveyors and other equipments are the resources. In [Viswanadham 90] they discuss deadlock prevention and deadlock avoidance. Like in the computer science literature they assume four conditions for the deadlock situation: *mutual exclusion*, *no preemption*, *hold and wait*, and *circular wait* ([Coffman 71]). “Mutual exclusion” means that a resource cannot be used by two or more processes at the same time. “No preemption” means that a resource will not be released by a process, until the process completes its job. “Hold and wait” is referred to the situation where the processes hold resources allocated to them, while waiting for additional ones, and “circular wait” is when a circular claim of tasks exist. Fanti and Zhou [Fanti 04] claim that the first three items are satisfied in the flexible manufacturing cells and the only condition to be checked is the latter one.

Deadlock *prevention* can be satisfied by falsifying one or more of these necessary conditions using static resource allocation. Viswandadham, Narahari and Johnson [Viswanadham 90] use the reachability graph of a Petri Net (PN) to arrive at resource-allocation policies that enforce deadlock prevention. These kinds of policies can become infeasible if the state space becomes very large. Therefore, in such cases they suggest deadlock *avoidance* as a preferred alternative. In deadlock avoidance they try to falsify one of the four necessary deadlock conditions in a dynamic way. They keep track of the current state and possible future conditions using a look-ahead algorithm and a generalized stochastic PN model. Deadlocks can be avoided depending on the degree of the deadlock and the degree of the look-ahead process. The assumptions in this method are: I) system’s model is known, II) the capacity of all resources is one, III) the nature of the resources is the same for all.

In order to characterize the deadlock situation in computer science literature, the state of the system can be represented by three types of graphs, namely Task-Wait-For Graph (TWFG), Task-Resource Graph (TRG), and General-Resource Graph (GRG). In TWFG, which is a digraph, the nodes are the tasks. In this representation an edge means that a task is waiting for the other. A cycle in this graph represents a deadlock. In this research we use the TWFG graph for detecting the deadlocks. They categorize deadlocks into three groups, which are the resources/communication deadlock model, the general resource system model and a hierarchy of deadlock models presented by Knapp [Knapp 87]. He has proposed a hierarchical set of deadlock models to describe the characteristics of deadlocks. Each model is restricted on the form resource requests can assume. For example, a task might need to acquire a combination of resources like (R1 and R2) or R3. This hierarchy includes: single-resource model, AND model, OR model, AND-OR model, $C(n,k)$ model and Unrestricted model. In the AND model tasks can request a set of resources. A task cannot be executed (is blocked) until all its requested resources are available. A cycle in a Wait-For graph is the necessary and sufficient condition for the existence of deadlocks in this model. And a blocked task T is said to be deadlocked if T is either in a cycle or can only reach deadlocked tasks. In the distributed systems the deadlocks can be classified into three different groups, namely *Path-Pushing* algorithms, *Probe-Based* algorithms, and *Global State Detection* algorithms. Some of these algorithms can be found in Chandy-Misra algorithm [Chandy 82], Chandy-Misra-Haas algorithm [Chandy 83], and the Obermarck algorithm [Obermarck 82].

The algorithm proposed by Mitchell and Merritt in [Mitchell 84] is the one that we are going to mainly focus on. This algorithm is a Probe-Based algorithm. Probes are going to be sent in opposite directions, in a TWFG. This algorithm will be described in the next chapter.

1.4.6 Learning

Supervised learning is learning from examples provided by a knowledgeable external supervisor. This is an important kind of learning, but alone it is not adequate for learning from interaction. In interactive problems it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act [Sutton 98].

In reinforcement learning (RL), the learner tries by direct interaction with the environment to discover the best action (by trying them), which gives the highest reward. In this method the emphasis is on how agents can improve their performance in a given task by perception and trial-and-error [Vlassis 03]. The agent has to *exploit* what it already knows in order to obtain reward, but it also has to *explore* in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best. On tasks with a stochastic outcome, each action must be tried many times to gain a reliable estimate its expected reward [Sutton 98].

Weiβ [Weiβ 95] describes the concept of multiagent RL as the following: "Multiagent learning is learning that is done by several agents and that becomes possible only because several agents are present." In multiagent RL many agents are simultaneously learning by interaction with environment and each other. While there are many results and findings in the area of single RL, the multiagent RL has been rowing quite slowly. The reason is that theoretical results for single agent RL do not directly apply to multiagent RL. Issues like, exponentially large state spaces, and the intractability of several distributed decision making algorithms make the multiagent RL much more complex [Bernstein 00]. Recent results have been showing that game-theoretic models can help in this matter [Wang 03].

In many MAS applications, agents only possess incomplete information about their environment and about other agents. Clearly complete knowledge at the time of design would help agents to properly predict possible future conflicts and/or interpret other agent's actions for more effective planning or decision making. On the other hand, if this knowledge was not given to the agent during the design time, there would be two options to gain that knowledge during the run-time. One approach is to have an advertisement structure like in RETSINA [Sycara 03]. In this approach agents advertise their capabilities and services through a middle agent to the whole society and therefore others can inform themselves about the behavior of the other agents by communicating with the middle agent. Another approach is to estimate the model of the other components by observing and interpreting their behavior. This is the avenue that we are planning to take in our research work. Our challenge is to devise a methodology that can help agents to construct a perception of their environment or other agents that they interact with. Our identification methodology will be based on Formal Language Inference, which has been studied by different authors with distinctive assumptions.

Oncina and Garcia [Oncina 92] introduced an algorithm, such that given a set of positive data (correct observations) and a set of negative data (wrong observations) of an unknown regular language, we can obtain a deterministic finite automaton consistent with the data. Carrasco and Oncina [Carrasco 94] proposed another algorithm which allows for the identification of any stochastic deterministic regular language as well as the determination of the probabilities of the strings in the language. The algorithm builds the prefix tree acceptor form the sample set and systematically merges equivalent states. Higuera [Higuera 98] presents a new learning method called learning distributions from experts. In his paper the experts are stochastic deterministic finite automata (sdfa). This method deals with situations arising when we want to learn sdfa from unrepeated examples. It is intended to model situations where data is not generated automatically, but in some probabilistic order as one would expect from a human expert.

1.5 Preliminaries

1.5.1 System

System is one of those primitive concepts whose understanding might best be left to intuition rather than an exact definition [Discrete Event Systems, Modeling and Performance Analysis, Ch. Cassandras]. Based on IEEE's standard dictionary of Electrical and Electronic Terms a *system* is a combination of components that act together to perform a function not possible with any of the individual parts. A *dynamic* system is one where the output generally depends on past values of the input. The state of a system at any time instant *t* should describe its behavior at that time instant in some measurable way. Another definition of *state* could be the collective information that is contained in the world at any time step *t*, and that is relevant for the task at hand [Vlassis 03]. The *state space* of a system is the set of all possible values that the state may take.

One way to classify systems is based on the nature of the state space selected for a model. In *continuous-state* models, the state space is a continuum consisting of all *n*-dimensional vectors of real numbers. In *discrete-state* models the state space is a discrete set. In *hybrid* systems some state variables are discrete and some others are continuous.

1.5.2 Discrete Event System

As with the term "system", we do not attempt to define what an *event* is. It is a primitive and should be thought of as occurring instantaneously and causing transitions from one discrete state value to another. In *continuous-state systems* the state generally changes as time changes. Therefore we refer to such a system *time-driven system*. In *discrete-state systems* the state changes only at certain points in time through instantaneous transitions. With each such transition we associate an event. Hence we call them *event-driven systems*.

Discrete event systems (DES) are event-driven systems. They describe the system state changes driven by the occurrence of individual events. Formally a discrete-event dynamic system (DEDS) or more broadly, a DES is a discrete-time, event-driven system, i.e. its state evolution depends entirely on the occurrence of asynchronous discrete events (no clock) over time. A discrete event occurs instantaneously and causes transitions from one discrete state to another.

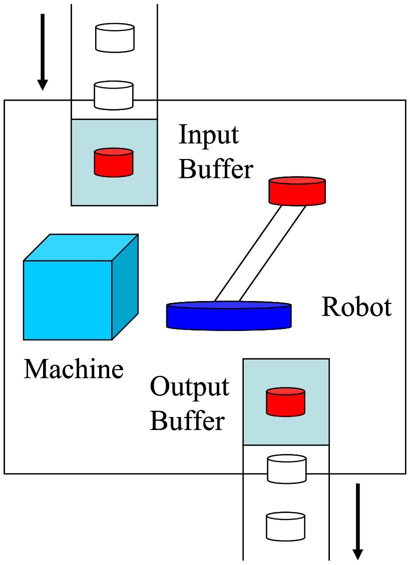

Figure 1-1: A manufacturing system

A model of a manufacturing system can be a good example of a DES. The entities (or clients) in a manufacturing system are parts. A manufacturing system (Figure 1-1) usually consists of a set of *machines* performing specific operations on the parts, a set of *material handling devices* such as robots, and a set of buffers for holding parts that are temporarily in idle state.

One possible event set for this example can be:

e_1: arrival of a part from the outside world to the input buffer

e_2: movement of the part from the input buffer to the machine, done by the robot

e_3: completion of service at the machine

e_4: movement of the part from the machine to the output buffer, done by the robot

A possible state space of the example might be:

$X = \{(x_1, x_2, x_3) : x_1 \in \{E, F\}, x_2 \in \{I, W\}, x_3 \in \{E, F\}\}$, where x_1 is the state of the input buffer: empty (*E*) or full (*F*), x_2 is the state of the machine: idle (I) or busy (B) and x_3 is the state of the output buffer: empty (*E*) or full (*F*).

Obviously there are many other alternatives to model this system, both for the set of events and the state space. How to look into the system and design the controller to achieve the desired objective depends on the control and process engineer. Modeling is always subject to personal biases.

Every system obviously exists to perform a particular function. In order for such a function to be performed, the system needs to be *controlled*. To get the desired behavior the right input will be selected by the *controller*. Design and implementation of the controller is the main issue in the literature. Since modeling is always the first step taken by the control engineer to design the controller, we introduce a method for modeling a DES in the next section, namely, state automaton and Petri net.

1.5.3 Formal Language Theory

One of the formal ways to study the behavior of a DES is based on the theory of formal languages and automata. Since in this dissertation we will be using some of the related concepts, we briefly introduce at this point the basics of the theory.

A *symbol* is an abstract entity and has no formal definition, e.g. digits – letters. A *string* (*word*) is a finite sequence of symbols, for example *a*, *b* and *c* are symbols and *abcb* is a string. The empty string will be denoted by λ. A finite set of symbols is called an *alphabet*, e.g. $\Sigma = \{0,1\}$.

A *formal language* is a set of strings of symbols from an alphabet, e.g. set of all strings over a fixed alphabet Σ, denoted by Σ^*. For example if $\Sigma = \{0,1\}$, then $\Sigma^* = \{\lambda, 0,1,00,01,10,11,000,...\}$ is a language. The formal language theory attempts to model all the admissible event sequences in a DES. A *regular expression* is a string that is used to describe or match a set of strings, according to certain syntax rules. Regular expressions provide a compact finite representation for potentially complex languages with an infinites numbers of words. Any language that can be denoted by a regular expression is a *regular language*.

1.5.4 Finite State Automata

A finite state automaton is a finite set of states and a set of transitions among the states that occur on input symbols chosen from an alphabet. Formally, *finite state automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, Q_m)$, where Q is a finite set of states *q*, Σ is a finite input alphabet, $\delta : \Sigma \times Q \rightarrow Q$ is the *transition function*, $q_0 \in Q$ is the *initial state* and $Q_m \subset Q$ is a subset of states to be called *marker states*. A *state transition graph* is associated with an finite automaton, where the vertices of the graph correspond to the states and if there is a transition from state *q* to *p* on input *a*, then there is an arc labeled *a* from state *q* to state *p*.

It is often convenient to represent an automaton graphically through a state transition diagram, which is simply a directed graph consisting of circles (states) and arcs (events). Figure 1-2 shows a sample finite automaton. State q_0 is the initial marking, and is shown with a double circle.

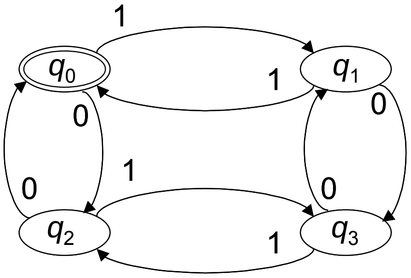

Figure 1-2: Finite automaton

In this example $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0,1\}$, $f(q_0,0) = q_2$, $f(q_0,1) = q_1$, $f(q_1,0) = q_3$, $f(q_1,1) = q_0$, $f(q_2,0) = q_0$, $f(q_2,1) = q_3$, $f(q_3,0) = q_1$, $f(q_3,1) = q_2$. The initial state is q_0.

2. Framework and Preliminary Results on Control Synthesis – The case of Deterministic Environment and Selfish Agents

2.1 Overview

In this chapter we discuss the architecture and framework of the system that we will be using in this dissertation. We also introduce some preliminary results with relaxed assumptions. Later in chapter 3 we will discuss the more general case of our methodology.

In our distributed framework, each agent is capable of performing one or more basic functions (in the literature, basic functions are sometimes called *actions*.). The objective of an agent is to achieve a certain goal(s) as part of the system's global goal(s). In this chapter we assume that the agents are selfish (non-*collaborative*) but communicate according to a pre-defined protocol. In Chapter 4 we will relax this assumption and will deal with collaborative agents. We also assume that agent's environment is deterministic, fully observable and known (either initially or through communication with other agents). An agent's environment is defined by everything which is outside of its jurisdictions, including other agents. These assumptions will be relaxed in Chapter 3 and the later chapters. Finally, in this chapter we will assume that the agent is not subject to failures, but it is possible for the agent to be engaged in a deadlock condition with other agents. Deadlock conditions are not known in advance, but once a condition is detected and a solution is identified, the agent’s control synthesis algorithm can be modified to avoid it in the future.

Next we will present our formulation of agent’s control synthesis. We will then present some results on deadlock detection and avoidance, and will also discuss how these can be embedded into the synthesis model. We will start with some preliminaries and definitions.

Our modeling framework encompasses the following major characteristics:

- Agents can be either providers and/or requesters.
- Each agent embeds in it a set of basic functions and a set of rules or control specifications.
- Agent’s actions are triggered by either internal or external events (described by flags). (4) Each such event or flag is associated with a goal (sub-goal), which must be achieved by the agent.
- Each agent locally synthesizes a set of control actions in order to reach its goal(s). The solution depends on the agent’s current conditions.
- It is possible for several agents (providers) to simultaneously respond to the same event triggered by a requester agent. In such a case, the requester would select a provider agent that provides the least-cost solution.
- *Local rules* enforce agents to execute their basic functions only when the precondition rules are satisfied in a certain state. *Postaction* rules persuade the agent to carry out some other basic functions after the main basic function is

executed (Some authors use the expression *effectors* - *action generating devices* - to describe postactions).

- There are also common rules, called global *rules hereafter* that must be satisfied by every agent in the system.

The control synthesis of agents is based on a ***search technique***. In the simplest form, an agent may just adopt the first feasible solution, which defines the set of control actions to be taken to reach the goal from the current state. In a more complex form, an agent may seek an optimal solution by taking into account its current state, including the set of its existing tasks. Yet, in a more complex scenario, the provider agent may face competition from other provider agents, thus seeking a solution which will undermine the other bidders. Finally, a solution may be sought in the face of uncertainties associated with failures or drop offs from other agents. In this chapter we will only focus on the simplest solution where the first feasible solution obtained is priced and communicated to the requester. It is of course possible that in some cases, no feasible solution is obtained due to conditions such as deadlocks. More complex cases will be studied and presented in the following chapters.

2.2 Definitions

Agent: We define an agent as a 6-tuple $\Phi = (S, F, fl, R, A, In)$, where:

$S = \{s_1, s_2, ..., s_m\}$ is a finite set that defines the state of the agent at any moment,
$F = \{f_1, f_2, ..., f_n\}$ is a set of basic functions,
$fl = \{\varphi_1, \varphi_2, ..., \varphi_r\}$ is a set of flags,
$R = \{r_1, r_2, ..., r_s\}$ is a set of rules,

$A = \{a_1, a_2, ..., a_t\}$ is a set of attributes,
$In = \{i_1, i_2, ..., i_w\}$ is a set of initiators.

Rules: These are condition-action rules, defined as part of system specification. There are two types of rules:

Global Rules: Inter-agent requirements which can be changed at the system configuration level.

Layout Specifications: Inter-agent requirements which define physical/logical accessibility or inter-connectivity between agents. Layout specifications are part of the global rules.

Local Rules: Intra-agent requirements changeable at the agent configuration level. Local rules are divided into three categories, namely "Preconditions", "Postactions" and "Poststates." "Preconditions" are those conditions that must hold before a basic function can be executed. *Preconditions* are usually a conjunction of function-free literal stating what must be true in a state before the action can be executed. *Postactions* are functions that must be executed by an agent after an action is taken.

Post-state or *Transition function* $\delta_{\Phi,f}$ is a mapping from $S \times F$ to *S*. Transition functions change the state of the agent Φ upon execution of a basic function *f*. They are also referred by *effects*.

Flags are set after a goal function is executed.

Basic Functions: These are agent's primitive capabilities or skills, pre-defined at the system design level, and cannot be changed by the designer. Every basic function has a set of inputs and outputs. Basic functions requiring physical actions are performed using agent's *actuators*. A basic function is *applicable* in any state that satisfies the preconditions; otherwise, the function cannot be applied.

State: Each agent is associated with a state defined by an n-tuple, where n is a finite number. State of an agent changes according to the basic functions executed by that agent.

Flags: A low/high signal triggered when the execution of a task (sub-goal) is finished by the agent. This flag will become low when the task is transferred to another agent.

Initiators: These are flags (events) that trigger a sequence of tasks to be executed by the agent. Each initiator is associated with one or more ordered goal functions (the goal functions must be executed in an ordered sequence). Initiators will be activated by request messages that the agent receives from its requesters.

Attributes: Define properties of agents, such as capacity of a machine.

Entity: An entity E is an object that is processed and manipulated by agents and can be shown by a tuple $E = (PP_i, j, k)$, where PP_i is the processing sequence id associated with the entity E, j is an instance id of the entity E and k is the current stage of the entity in the processing sequence.

Processing Sequence: is an ordered set of agents where PP_i = {All the agents Φ_i, which must be visited in a sequential order by an entity E}

2.3 Assumptions and Methodology

The following assumptions are made:

1. Agents communicate with their environment according to a pre-defined protocol.
2. An agent's environment is assumed to be fully *observable* and *deterministic.* Agents have full and complete perception model of each other, either initially or through communication channels. This assumption will be relaxed in Chapter 2.

3. Agents can be grouped into colonies, where a colony defines the boundary by which agents can communicate and access each other.
4. Pre- and postaction rules, also known as *condition-action rules* or *if-then rules* are used in our application. Using "forward chaining" new rules can be generated – these are postaction rules which define new functions for execution (production rules). A rule has *antecedent* clauses joined by *conjunctions* and has a consequent clause. A rule states a relationship between clauses. A rule whose antecedent clauses are all true is said to be *triggered* or *ready to fire*. Most rule systems allow boolean condition operators in addition to equality [Bigus 01].
5. A *requester* agent requiring a certain service issues a request to other agents in its colony. Every agent who receives that request is a potential *provider*. An agent can be a requester of a job and at the same time the provider of another job.
6. When a provider completes a task (reaches a sub-goal), it signals an event and raises a *flag*. Flags cause the agent to send a request to its providers for the next operation that has to be done on a given entity. It basically transfers the job (global goal) to the other agents (cooperative environment). In return to this request message, the providers set a flag that initiates a search algorithm for the execution of the task. These flags are called *initiators*.
7. The providers have a list in their active memory where they keep track of *to-do* jobs. This list includes the set of initiators that are currently in "set" position.
8. The colonies, the basic functions, the rules, and the flags of an agent are all determined by the control designer in the specifications of the agent during the design stage.

2.3.1 Methodology

In this chapter we will deal with the *problem-solving agents* which search for sequence of actions that lead to desirable states. These desirable states are called *goal states*. The process of finding out which sequence of actions will take the agent to its goal state without breaking any of the rules is called ***synthesis***. An agent with several immediate options of unknown value can decide what to do by first examining different possible sequences of actions that lead to states of known value and then choosing the best sequence. This process is called ***search*** [Russell 03].

The synthesis methodology can be summarized as follows:

1. A provider agent initiates synthesis as soon as it receives a request message from a requestor agent, which is generated by an initiator from the requestor agent,
2. Since it is possible to have more than one provider for every request, there can be a competition between different providers. In order to have an organized competition, we have assumed a market environment for the agents, where they can announce their costs/prices to the requester.
3. Every synthesis solution is associated with a cost, which depends on the basic functions to be executed, their sequence and duration of execution, and also the tasks or jobs waiting to be executed by that agent. Generally speaking, there are also non-deterministic elements to the cost function, including internal failures within the agent which could potentially make some or all of the basic functions un-executable. Furthermore, there are also environmental factors which are often un-observable by

agents, but could directly or indirectly affect the agent's performance. In chapter three we will discuss these issues in more detail. If a provider is not able to perform the tasks associated with a request, the cost value will be set at infinity.

4. The bidding policy in this chapter is *first-price sealed bid auction*. The requester will receive the estimated cost value from all of its providers and will choose the provider with the lowest cost. Many other auction protocols can be chosen as a desired policy.
5. The selected provider will be announced to all the providers involved in this request. They will basically update their flag list and remove the request from their active memory. And an assignment will be sent to the selected provider. From this point on the provider must put the assigned task into its *task (to-do) list*. At the same time the requester will remove the flag from its own list.
6. As a last step the provider physically commits the sequence of basic functions and removes the initiator from its task (to-do) list.
7. When the selected provider attempts to execute the task (reach the sub-goal), it must first acquire all the necessary resources required for the solution. One solution for the agent is to reserve these recourses and lock them. These resources will not be available for other agents until the agent unlocks them – This assumption is similar to an *enforced* control that can be considered as part of the communication protocol. We will later relax this assumption.

The above steps are illustrated in figures 2-1a, b and c:

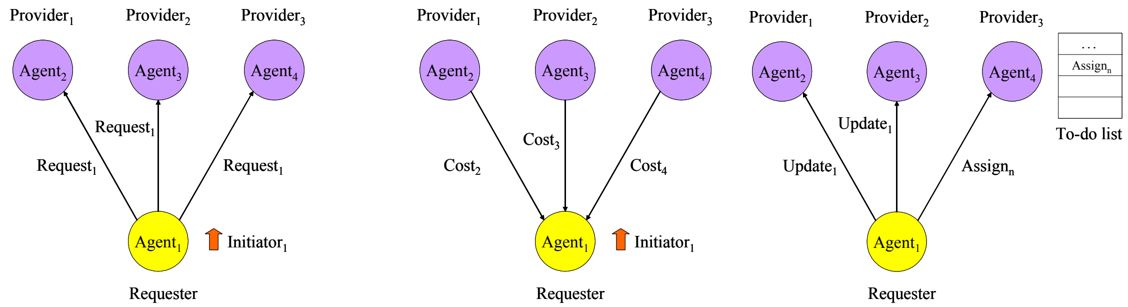

Figure 2-1a, b, c: Requester - Provider

2.3.2 Search Algorithm - Introduction

The problem of synthesis can be solved using a *search* technique. Search techniques usually use an explicit *search tree* that is generated by the initial state and the successor function that together define the state space.

Search algorithms can be categorized into two groups. *Uninformed search* methods (or *blind search*) have no additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from a non-goal state. On the other hand, *informed search* or *heuristic search* knows whether one non-goal state is "more promising" than another one [Russell 03]. In this chapter we will discuss both. We will start with un-informed search techniques.

Exponential-complexity search problems cannot be solved by uninformed methods such as *breadth first*, *uniform-cost* or *depth first* search for any but the smallest instances [Russell 03]. Since the only objective of this chapter is to familiarize the reader with the system's framework and the approach, we assume that the instances of the problems are small and consequently the uninformed search methods can find a solution. We will consider the depth-first search as our search strategy and we will build the synthesis based on this strategy (We do not consider cost optimality in this chapter. Uniform-cost search finds optimal solution as opposed to breadth first or depth first search.).

Depth first search (DFS) always expands the *deepest* node in the current fringe of the search tree and the search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the fringe, so then the search "backs up" to the next *shallowest* node that still has unexplored successors. The advantage of depth first search strategy is that it requires very modest memory. It needs only to store a single path from the root to a leaf node, along with the remaining unexpanded sibling nodes for each node on the path. For a state space with branching factor b and maximum depth of m, the space requirements for this strategy is in order of $O(bm)$ and the time cost is $O(b^m)$ (m is infinity in case of unbounded trees). The disadvantage of depth first search is that it can make a wrong choice and get stuck going down a very long or sometimes infinite path when a different choice would lead to a solution near the root of the search tree; hence it is not optimal. Another drawback of this method is that it is not complete. If the left sub tree were of unbounded depth with no solution, the search would never terminate. There are some possible improvements that are explained in section 2.3.4.

2.3.3 Search Algorithm – Depth First Search Strategy

Below we present the algorithm that can be used to synthesize control laws for agents to reach a goal state starting from a given initial condition, Algorithm *path search* – part I and II are built based on the depth first search strategy. Initiator of an agent defines the goal function(s) that must be searched from the current state of the agent. The algorithm initially starts at the root of the search tree. The total number of possible branches at the root node is equal to the total number of the basic functions that the agent possesses. Each branch is equivalent to the execution of the corresponding basic function. If all the preconditions for a specific basic function are satisfied, the algorithm generates a new node and expands the tree. Otherwise, that branch will not be part of the solution and the tree will be at a dead end and the algorithm will continue with another yet unexpanded node of the tree.

The algorithm is shown below. The search_path function will first be called after receiving the initiator and this procedure will call the DFS function recursively. In each iteration, the algorithm takes the first possible function, and stores the step with the corresponding predicted state in a linked list. If the search algorithm reaches the target, it will return the path, the related state transitions and the corresponding total cost. If it does not find any possible path it returns NULL with an infinite value for the cost. This search will be invoked when there is a service request for the agent.

Note: Throughout this dissertation we use the methodology mentioned in [Baase 90] to describe the algorithms.

Algorithm: *path search – part I*

Input: Destination list (sub-goals) of the initiator i_j – Current state S of the agent Φ

Output: Path P – Cost C

Comment: All the inputs are the elements of the agent Φ. Initiator i_j has a corresponding list of goal functions (ordered), which we call the destination list. The goal of the agent is to execute the goal functions in the list one by one, starting from the first goal function in the list. The output of this algorithm is the path *P* (if any found) and its corresponding cost *C* (This cost is not optimal). The path will be a list of basic functions that must be executed by the agent in order to achieve the goal defined by the initiator i_j. If there is no solution the returned path will be NULL. Function calculate_cost is a function that calculates the cost of the solution in the last step.

```
procedure search_path (destlst: BasicFunctionsList, s: StateOfAgent)
var
    dest: BasicFunction
    state: StateOfAgent
    p: BasicFunctionsList
    c: Integer
begin
    p is NULL
    while destlst is nonempty
        dest = first element of destlst
        remove first element from destlst
        (p, state) = DFS_path((p, s), dest)
        if p = NULL
            break
        end if
    end while
    c = calculate_cost(p)
    return (p, c)
end
```

Algorithm: *path search – part II*

Input: Basic function set *F* – Transition functions $\delta_{\Phi,i}$ – Current state *S* of the agent Φ - Agent's local and global rules *R* – Path *P* – Destination dest, as the goal function

Output: (path, state, c) which is the path to the goal function from the current state with its cumulative cost.

Comment: Variable (path, state, c) is a collection of basic functions and the corresponding states and costs (after hypothetical execution). The element "path" is a list of basic functions that shows the steps that must be executed by the agent in order to reach the goal and the element "state" is the list of states, which shows the consecutive state transitions according to the "path" and c is the cumulative cost of the execution of the "path." (path, state, c) can be implemented as a linked list. PostactionEnables are the preconditions that must be true, in order to execute a specific basic function as a postaction. Note that these conditions are different than the preconditions that are part of the specifications. "update" function updates the state of the agent based on its current state and the transfer functions (effects).

```
procedure: DFS_path ((path: BasicFunctionsList, state: StateOfAgent), dest: BasicFunction)
var
    bfunc: BasicFunctionsList/*Basic functions defined in the specifications of the agent*/
    (path, state): (BasicFunctionsList, StateList)
    last: (BasicFunction, StateOfAgent)
    expected_state: StateOfAgent/*Next hypothetical state of the agent after execution of a basic
function*/
    x: BasicFunction
begin
    last = last element of (path, state)
    if last.path has Postactions then
        for all Postactions of last.path
            if all PostactionsEnables of the Postaction are True then
                if all the preconditions of the Postaction are True then
                    expected_state = update(last.state, Postaction)
                    (path, state).add(Postaction, expected_state)
                    if last.path = dest then
                        return (path, state)
                    else
                        (path, state) = DFS_path((path, state), dest)
                        if path not equal to NULL then
                            return (path, state)
                        end if
                    end if
                end if
            end if
        next
    else
        for all x in bfunc
            if all Preconditions of x are True then
                expected_state = update(last.state, x)
                (path, state).add(x, expected_state)
                    if x = dest then
                        return (path, state)
                    else
                        (path, state) = DFS_path((path, state), dest)
                        if path not equal to NULL then
                            return (path, state)
                        end if
                    end if
                end if
            end if
        next
    end if
    ruturn NULL
end
```

As we already mentioned, the cost calculation can be based on the duration of time that resources are utilized or fixed cost on each resourse used. Since in this chapter we assume that the agents can only execute one job at a time, the cost calculation is straight-forward.

Here we assume that the total cost of a single task (from initial state to some goal) is equal to the total number of basic functions executed.

2.3.4 Possible Improvements

The problem of DFS not being complete can be alleviated by using a predetermined depth limit l which solves the infinite-path problem. Unfortunately if $l < d$ (where d is the depth of the shallowest goal node) then the shallowest goal is beyond the depth limit and another source of incompleteness will be created. This case can very much happen, because d is usually unknown. To overcome this issue we can also use *iterative deepening search* or *iterative deepening depth first search*, which finds the best depth limit. In this algorithm we start with d equal to 0 and then we increase d in each iteration by 1, until a goal is found, which obviously happens when the algorithm reaches d, i.e. the depth of the shallowest goal node. This strategy is complete when the branching factor is finite and even is optimal when the path cost is a non-decreasing function of the depth of the node. The time complexity of this algorithm is $O(b^d)$ and the memory complexity is $O(bd)$.

2.3.5 Search Algorithm – Informed (Heuristic) search – A* search

These types of search use problem specific knowledge beyond the definition of the problem itself and therefore can find solutions more efficiently than an uninformed strategy. A* search [Hart 68] is one of the informed search algorithms that fits under the category of best-first search. In best-first search algorithms the nodes are expanded based on their evaluation function $f(n)$. The evaluation function usually measures the distance to the goal from that node. Another key component of these algorithms are the heuristic functions that are usually denoted as *h*(*n*). This function estimates the cost of the cheapest path from node *n* to a goal node. A^* search evaluates nodes by combining *g*(*n*), the cost to reach the node and *h*(*n*) the cost to get from the node to the goal: $f(n) = g(n) + h(n)$ therefore *f*(*n*) will be the estimated cost of the cheapest solution through *n*. If the heuristic function *h*(*n*) satisfies certain conditions, A^* search is both complete and optimal. A^* is optimal if *h*(*n*) is an admissible heuristic, that is provided that *h*(*n*) never *overestimates* the cost to reach the goal.

Suppose that we describe the state of an agent by $S = \{s_1, s_2, ..., s_m\}$. The goal function can then be stated as $S' = \{s'_1, s'_2, ..., s'_m\}$. Assuming that each basic function of the agent can influence only one of the elements of the state set, then the agent needs at least *m* of its basic functions to reach S' from *S* if $s_1 \neq s'_1, s_2 \neq s'_2, \ldots, s_m \neq s'_m$. In other words, the heuristic function *h* is equal to the number of the elements in *S* and S' that do not match. This assumption guarantees the admissibility of the heuristic function and therefore the completeness and optimality of the algorithm.

2.3.6 Illustrative Example

We use the same example used in chapter 1 to clarify our approach. Consider a flexible manufacturing cell (FMC) as shown in figure 2-2, which consists of one input buffer, one output buffer, one robot, two machines and one part type. Parts arrive to the system and are processed according to their “process plan”. The process plan determines the sequence of machines to be visited by each part. Robots transfer parts from one station to another. Each step in a process plan is considered to be a sub-goal, while the overall goal is to complete each part successfully. In this system the buffers, the machines and the robot are agents with different capabilities and specifications, and parts are entities,

System specifications

In general, specifications are categorized into three different categories. The global rules are common between all agents. Every agent must follow the global rules. An example of a global rule can be the process plan of an entity (entity will be described shortly). All the agents have access to this information. Another category is the layout specification, which define how the agents are connected to each other and who can access whom. The last category is the set of local rules. These rules are private and embedded into each agent. Basic functions of an agent are defined in its local specification, and can be different from one agent to another. We categorize the local specifications into three different groups. Preconditions are those that have to be satisfied before a basic function can be executed. Postactions are those functions that must be executed by the agent after an action has been taken. Poststates or transition functions are the states of agents after the execution of a basic function. Initiators are also part of the local specifications. Initiators initiate a set of functions to be executed by an agent after receiving a request.

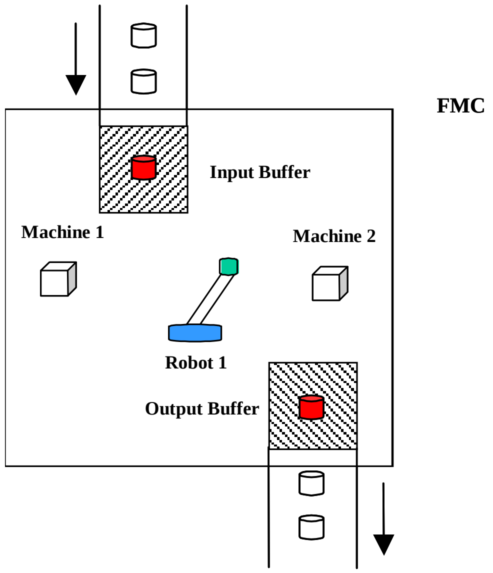

Figure 2-2: Flexible Manufacturing Cell (FMC)

Note: Each agent has a set of functions that is public to other agents in its colony. We call them the “interface functions.” Some preconditions or postactions require information about the state of the other agents. And because of the distributed nature of our framework, the agent’s are not able to directly measure the required states. The purpose of having interface functions is to be able to get information about the state of the other agents during the synthesis. These interface functions will be announced to the other agents in the colony by the “bookkeeper agent.”

<u>Input Buffer</u> $\Phi = I_i \ (i = 1)$

Description: This agent is the place where different parts arrive to the cell. When a part arrives, it is placed in the input buffer by some mechanical means. We do not consider the arrival mechanism in this example. At this point the flag of the input buffer will set to high and hence it will generate a request message.

Attributes ($I_i.A$): $I_i.A.a_1$(capacity) = 1, $I_i.A.a_2$(providers) = $\{R_1\}$, $I_i.A.a_3$(interface function) = {Out(E)} (The output of out(E) is a boolean value).

State ($I_i.S$): (s_1, s_2, s_3) = (*parts_in_system*, *part_id_in_process, availability*): (# of parts in the buffer, part_id in front of the queue, locked/unlocked)

Basic Functions ($I_i.F$): none

Flags ($I_i.fl$): $I_i.fl.\varphi_1$ is equal to one if there is a part in the buffer and equal to zero, otherwise.

Specifications ($I_i.R$):

- $I_i.R.r_1$: specifications of out(E)
  out(E) returns "true" if:
  $I_i.S.s_1 > 0$
  $I_i.S.s_2 \neq$ NULL
  $I_i.S.s_3$ = unlocked

Initiators ($I_i.In$): none

The capacity of the buffer is one according to the value of $I_i.A.a_1$. The only provider of the input buffer is R_1. The interface function out(E) will return a true value, if any agent attempts to take a part from it and if all the conditions for returning "true" are satisfied. The state of the input buffer will be determined by three values, namely, number of parts currently in the buffer, the state of the part currently in the front of the buffer (if there is no part in the buffer this value will be NULL), and the availability of the buffer, meaning whether the buffer is locked by any agent (requester) or not. The input buffer does not have any basic functions, therefore there will be no local specifications defined for it.

Since the part arrives to the cell by some mechanical means the buffer does not have initiators. But if when a part arrives to the buffer the flag $I_i.fl.\varphi_1$ ($i = 1$) is set to high.

Output Buffer $\Phi = O_i$ ($i = 1$)

Description: This agent will be the last place that parts visit in the cell. When the last process of machining is done, the part will be placed in the output buffer by the robot. We do not consider the removal mechanism of parts from the buffer.

Attributes ($O_i.A$): $O_i.A.a_1$(capacity) = ∞

States ($O_i.S$): (s_1) = Since the capacity of the buffer is infinite, the state of the output buffer is irrelevant.

Basic Functions ($O_i.F$): none

Flags ($O_i.fl$): none

Specifications ($O_i.R$): none

Initiators ($O_i.In$): none

Machine $\Phi = M_i$ ($i = 1, 2$)

Description: M_1 and M_2 are responsible for the machining of parts. We assume that the machines can only process one part at a time. Our assumption is that the local machine controllers will control the machining process.

Attributes ($M_i.A$): $M_i.A.\mathrm{a}_1$(capacity) = 1, $M_i.A.a_2$(providers) = $\{R_1, M_i\}$, $I_i.A.a_3$(interface function) = {in(E), out(E)} (The output of in(E) and out(E) is boolean).

States ($M_i.S$): (s_1, s_2, s_3, s_4) = (*stage*, *parts_in_system*, *part_id_in_process, availability*): (idle/busy/done, # of parts in the machine, part_id being worked on, locked/unlocked)

Basic Functions ($M_i.F$):

$M_i.F.f_1$ = start_man(E), where E is the entity.

Flags ($M_i.fl$):

$M_i.fl.\varphi_1$ is equal to one if a part is ready to be taken out of the machine.

Initiators ($M_i.In$): if $M_i.In.\varphi_1 = 1 \rightarrow$ start_man(E)

Specifications ($M_i.R$):

- $M_i.R.r_1$: specifications of start_man(E)

  Preconditions:
  $M_i.S.(s_1, s_2, s_3, s_4)$ = (idle, 1, ≠ Null, unlocked)
  Postactions: none
  Poststates:
  $M_i.fl.\varphi_1 = 1$
  $M_i.In.\varphi_1 = 0$
- $M_i.R.r_2$: specifications of in(E)

  in(E) returns true if:
  $M_i.S.s_1$ = idle
  $M_i.S.s_2 < M_i.A.\mathrm{a}_1$
  $M_i.S.s_4$ = unlocked
- $M_i.R.r_3$: specifications of out(E)

  out(E) returns true if:
  $M_i.S.s_1$ = done
  $M_i.S.s_2 > 0$
  $M_i.S.s_4$ = unlocked

The state vector of the machines are determined by four values. The first value signifies the operating condition of the machine which can be “idle”, “busy” and “done”. The machine will be in “idle” state when there is no part in its manufacturing table. It will be in “busy” state, if it is involved with machining of a part or it can be in “done” state if the machining process of the part is finished and the machine is waiting for the part to be removed. The next value in the state vector of the machine is the number of parts in it. In our example this number is either one or zero. The third value in the vector is the state of the part currently in the machine (this value will be NULL if there is no part in the machine). The last value is the availability of the machine, which can be either "locked" or "unlocked". Each machine has a flag. It will be activated upon arrival of a part and will be deactivated as soon as the finished part is removed from the machine. Each machine has only one basic function available. The basic function is called start_man(*E*). This function will be executed whenever there is the need for manufacturing ($M_i.In = 1$). It will be initiated upon arrival of a part into the machine.

The functions *in* and *out* are for communication purposes. Since other agents are not aware of the state of the machines (because of the distributed nature of the system), they must get the necessary information (state of the machine) through communication using these two functions. These functions return a Boolean "true" or "false" value based on the conditions of the machine. For example if the circumstances for putting any part into the machine are met (from machine's point of view) then the robot will receive a true value upon calling the *in* function.

Before the execution of start_man(*E*) some preconditions must be satisfied: Machine *i* must be in idle condition; there must be one part in the machine, therefore the entity's id will not be NULL; and finally the machine must be in unlocked state. After the job is finished there will be no other mandatory basic function to be executed. And obviously the initiator flag goes low, upon completion of the job and the flag goes high.

In order to put a part into a machine, first of all the machine must be in idle state. And obviously number of parts currently in the machine must be less than the capacity of the machine, and the machine must also be unlocked. These are all the conditions for the *in* function to return a true value. If any single one of these conditions is not satisfied, the machine will return "false." The conditions of the *out* function are similar to the *in* function.

Robot $\Phi = R_i$ $(i = 1)$

Description: R_1 is responsible for moving the parts between different stations. We assume that the robot can hold at most one part in its arm. We do not consider the technical details of the complex mechanical functions executed by the robot in order to grab a part and move it around. We consider them as granted. The robot has one initiator, namely $R_i.In.\varphi_1$. If this initiator becomes active the robot must take the part from the requester and put it in the final destination. The process plan determines the next destination.

Attributes ($R_i.A$): $R_i.A.\mathrm{a}_1$(capacity) = 1, $R_i.A.\mathrm{a}_2$(reachability) = $\{I_1, M_1, M_2, O_1\}$

States ($R_i.S$): $(s_1, s_2, s_3, s_4, s_5)$ = (*stage*, *occupancy*, *location*, *part_id_in_process, availability*): (idle/busy, empty/not empty, $\mathrm{I_i/O_i/M_i}$, E, locked/unlocked)

Basic Functions ($R_i.F$):

$R_i.F.f_1$ = move(Y, E), where $Y \in \{M_1, M_2, I_1, O_1\}$, where E is an entity

$R_i.F.f_2$ = take(Y, E) where $Y \in \{M_1, M_2, I_1\}$, where E is an entity

$R_i.F.f_3$ = put(Y, E) where $Y \in \{M_1, M_2, O_1\}$, E is an entity

Flags ($R_i.fl$): none

Initiator ($R_i.In$): if $R_i.In.\varphi_1 = 1 \rightarrow$ put($E.PP_i(k+1), E$) $X_i \in \{M_1, M_2, I_1\}$

Specifications ($R_i.R$):

- $R_i.R.r_1$: specifications of move(Y, E):
  Preconditions:
  $R_i.S.s_3 \neq Y$
  $R_i.S.s_5$ = unlocked
  $Y \in R_i.A.a_2$
  Postactions:
  if $R_i.S.s_4 \neq$ Null then put(Y, E)
  if $R_i.S.s_4$ = Null then take(Y, E) – These two postactions are not necessary but they will reduce the search space of the agent.
  Poststates: none

- $R_i.R.r_2$: specifications of take(Y, E):
  Preconditions:
  $R_i.S.s_1$ = idle
  $R_i.S.s_2$ = empty
  $R_i.S.s_3 = Y$
  $R_i.S.s_4$ = NULL
  $R_i.S.s_5$ = unlocked
  $Y \in R_i.A.a_2$
  Y.out = true
  Postactions:
  move(W, E) where W is the next station in the processing sequence after Y and $E = (PP_i, j, k + 1)$ – This postaction is not necessary but it will reduce the search space of the agent.
  Poststates: none

- $R_i.R.r_3$: specifications of put(Y, E):

  Preconditions:

  $R_i.S.s_1$ = idle
  $R_i.S.s_2$ = not empty
  $R_i.S.s_3 = Y$
  $R_i.S.s_4 \neq$ Null
  $R_i.S.s_5$ = unlocked
  $Y \in R_i.A.a_2$
  $Y.in = true$
  Postactions: none
  Poststates: $R_i.In.\varphi_1 = 0$

Robot R_1 can reach all other agents. The state of the robot is determined by five different values. The first value is the stage of the robot, which can be "idle" or "busy". The robot will be in idle state when there is no part in its arm and does not move. It will be in busy state, if it has any parts in its arm or if it is moving towards a station. The second value is the occupancy of the robot. It will be either "empty" or "not empty". The third value is robot's location. Since the robot is moving around in the cell, this value has to be tracked. The state of the part, currently hold by the robot, is the next value (this value will be NULL if there is no part in the robot). The last value is the availability of the robot, which can be either "locked" or "unlocked."

R_1 has three basic functions available: Robot executes move(Y, E) function by moving towards location Y, and performing some task on entity E. Function take(Y, E) is executed by taking a part form agent Y, and put(Y, E) is executed by putting part E in agent Y. R_1 has also an initiator, which is going to be set by a requester like M_1 or M_2.

Transition functions

The changes in the states are depending on the level we are at. The transition functions at the synthesis level are different than the transition functions at the execution level. The reason is that at the execution level agents may lock some other agents to avoid conflict. A locked agent cannot perform any synthesis. In order to avoid this, we consider two sets of transitions functions; functions at the synthesis level and at the execution level. The transition functions at the execution level are described below. As we can see in figure 7, when a basic function is executed by the agent, its state is changed to "locked." The transition functions are shown as Colored Petri Nets, and the conditions are shown as guards.

a) $R_i.S(s_1, s_2, s_3, s_4, s_5) = (stage, occupancy, location, part\_id\_in\_process, availability)$: (idle/busy, empty/not empty, $I_i/O_i/M_i$, E, locked/unlocked)
$\delta_{R_i,move}(R_i.S(s_1, s_2, s_3, s_4, s_5), \text{move}(Y, E)) = R_i(s_1, s_2, Y, E, locked)$
$\delta_{R_i,take}(R_i.S(s_1, s_2, s_3, s_4, s_5), \text{take}(Y, E)) = R_i(s_1, \text{not empty}, Y, E, locked)$
$\delta_{R_i,put}(R_i.S(s_1, s_2, s_3, s_4, s_5), \text{put}(Y, E)) = \mathrm{R}_i(s_1, \text{empty}, Y, E, unlocked)$

b) $M_i.S(s_1, s_2, s_3, s_4) = (stage, parts\_in\_system, part\_id\_in\_process, availability)$: (idle/busy/done, # of parts in the machine, part_id being worked on, locked/unlocked)
$\delta_{M_i,start\_man}(M_i.S(s_1, s_2, s_3, s_4), \text{start\_man}(E)) = M_i.S(s_1, s_2, E, locked)$

Process Plan

Now that we have defined all the system components and specifications we need to introduce the process plan of the entities in the system. An example process plan is $PP_1=\{I_1, M_1, O_1\}$, which means that part type 1 has to go to stations I_1, M_1 and O_1, respectively.

Scenario

Suppose that the initial state of R_1 is $R_1.S$ = (idle, empty, M_1, NULL, unlocked), and the initial state of M_1 and M_2 are $M_1.S = M_2.S$ = (idle, 0, NULL, unlocked). Also suppose that part $E(PP_1, 1, 1)$ arrives in input buffer I_1. PP_1 means that the arriving part is of type 1. The first "1" in the above tuple means that this part is the first instance of this entity type, arriving into the cell. The second "1" means that the part is currently in the first station of the process plan (I_1).

$I_1.f.\varphi_1$ becomes equal to one. I_1 will send an initiator request message to R_1, since R_1 is in the list of the providers' of I_1. R_1 receives the request message that entity E is ready to be picked up (Figure 2-3) and therefore its initiator $R_1.In.\varphi_1$ will change its state to "1".

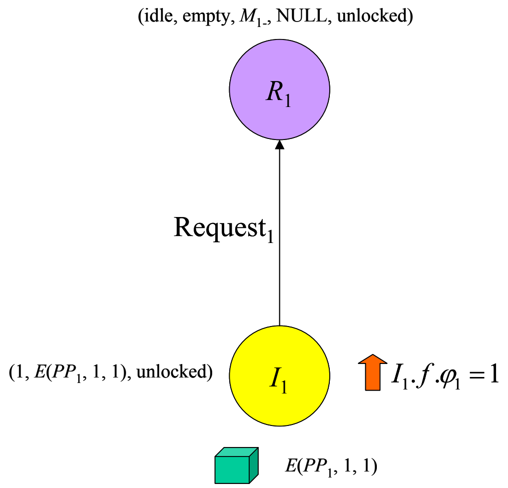
(idle, empty, $M_{1.}$, NULL, unlocked)
R_1
Request$_1$
(1, $E(PP_1, 1, 1)$, unlocked)
I_1
$I_1.f.\varphi_1 = 1$
$E(PP_1, 1, 1)$

Figure 2-3: Request from the input buffer

According to R_1's initiator $R_1.In.\varphi_1 = 1 \rightarrow$ put($E.PP_1(2), E$), R_1 must search for a path that takes the part from I_1 and puts it into $E.PP_1(2) = M_1$. At this point the robot will initialize the search algorithm by calling function search_path(put(M_1, E), (idle, empty, M_1, NULL, unlocked)). As we can see in Figure 2-4, a search tree will be generated, where the root is the current state of R_1 and the branches of the root are the possible basic functions of R_1. Since the location of R_1 is currently next to M_1, it cannot take the part from the buffer (One of the preconditions of basic function take(I_1, E) is $R_1.S.s_3 = I_1$, which is currently not true - $R_1.S.s_3$ is equal to M_1). So the left most branch will reach a dead end. The second branch will also reach a dead end because for executing put(I_1, E), $R_1.S.s_2$ must be "not empty," which is not the case. The only possible action that the robot can take will be move(I_1, E) (All the preconditions for this action are satisfied.) Now the search algorithm will hypothetically assume that R_1 takes this action; therefore based on the given transition function $\delta_{R_i,move}$ it will predict the next state of the robot which is $R_1.S.$(busy, empty, I_1, NULL, unlocked). Note that $R_1.S.s_5$ remains equal to "unlocked," because we are still in the synthesis level and not real physical execution.

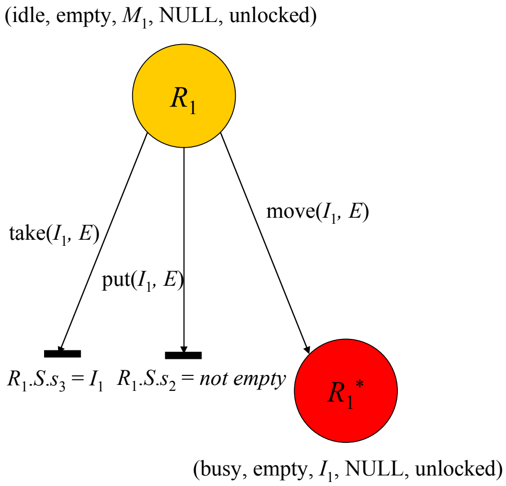

Figure 2-4: Search tree

Now the search algorithm continues the same procedure to find that all the preconditions for take(I_1, E) are satisfied. This means that the part can be taken from the input buffer. The next state prediction is going to be $R_1.S.$(busy, not empty, I_1, E(PP_1, 1, 1), unlocked). This step can be seen in Figure 2-5. By taking this action half of the goal functions is satisfied and the search will continue until it reaches the last basic function in its destination list, which is put(M_1, E). Since the "take" function has a postaction, the robot will do the postaction first. This action is going to be move(M_1, E(PP_1, 1, 2)). Since all the preconditions are satisfied, the robot will take this action (without doing the search) and therefore its next state will be $R_1.S.$(busy, not empty, M_1, E(PP_1, 1, 2), unlocked).

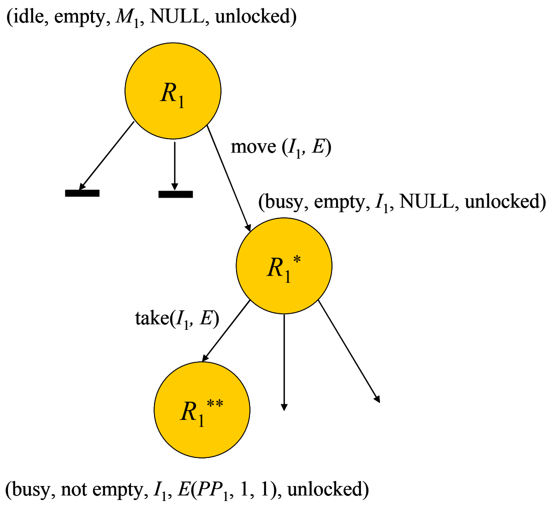
(idle, empty, M_1, NULL, unlocked)
R_1
move (I_1, E)
(busy, empty, I_1, NULL, unlocked)
R_1^*
take(I_1, E)
R_1^{**}
(busy, not empty, I_1, $E(PP_1$, 1, 1), unlocked)

Figure 2-5: Search tree – second level expansion

The search continues to find out that the next possible expansion of the nodes can be move(M_1, E). This node is shown in Figure 2-6 as R_1^{***}. The second branch of this node brings the robot to the goal function. One of the preconditions of put function is Y.in = true. In this case Y is M_1. The state of M_1 is currently $M_1.S$.(idle, 0, NULL, unlocked). The robot will call this public function (*in*) of the machine. Since all the preconditions of this function are satisfied, it will return “true”, so the robot can now put the part into the machine. This successfully completes the search (Figure 2-6). Now that the control synthesis is performed successfully, that agent must announce the cost of this operation. Note that according to the process plan this task (done by the robot) is a sub-goal of the main goal.

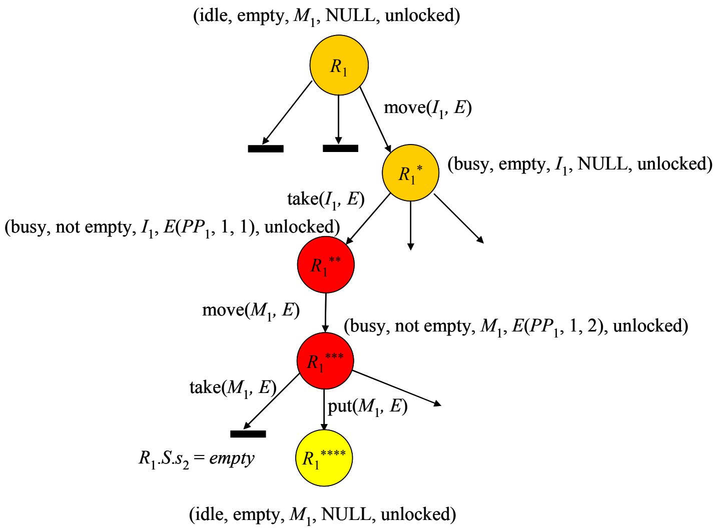
(idle, empty, M_1, NULL, unlocked)
R_1
move(I_1, E)
R_1^*
(busy, empty, I_1, NULL, unlocked)
take(I_1, E)
(busy, not empty, I_1, $E(PP_1, 1, 1)$, unlocked)
R_1^{**}
move(M_1, E)
(busy, not empty, M_1, $E(PP_1, 1, 2)$, unlocked)
R_1^{***}
take(M_1, E)
put(M_1, E)
$R_1.S.s_2 = empty$
R_1^{****}
(idle, empty, M_1, NULL, unlocked)

Figure 2-6: Search tree - expansion

Next the search algorithm calculates the cost of execution of this task, which is assumed here to be proportional to the total number of basic functions used, or is the depth of the generated path in the search tree. In this example the agent R_1 uses the "move" function twice, the "take" and put functions each once, therefore the cost of this operation is 4. R_1 will announce this cost to its requester I_1 (Figure 2-7). Since R_1 is the only provider of I_1, I_1 will accept this offer, it will assign the task to R_1 and it will clear its flag (Figure 2-8). R_1 can now execute this task.

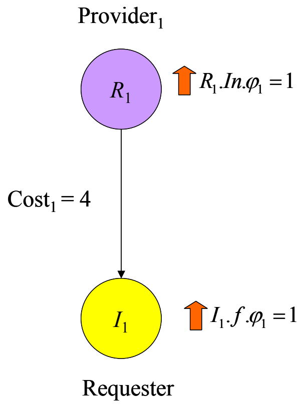

Figure 2-7: Response from the Provider R_1

All the other steps in the process plan goes through the same set of synthesis procedure until the part exits the cell. Figure 2-9 shows the whole solution state space, when a part is processed by the agents in the example manufacturing cell.

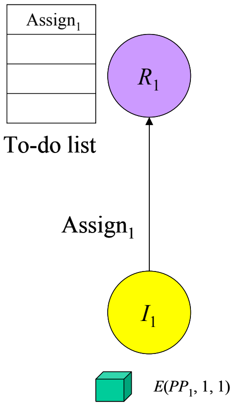

Figure 2-8: Task assignment

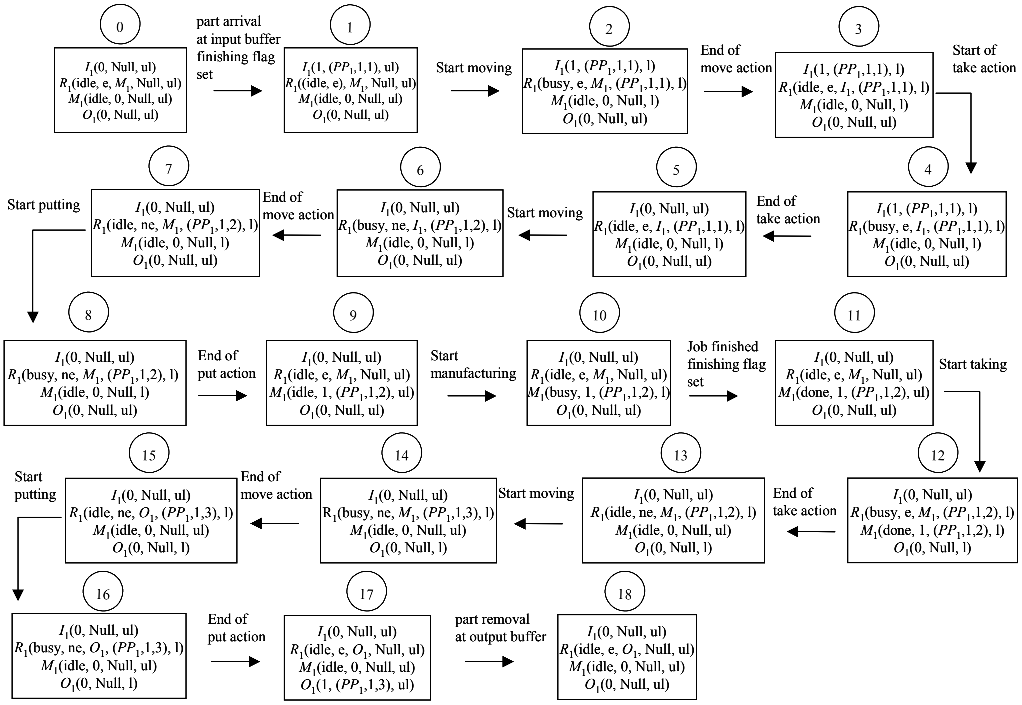

Figure 2-9: Solution state space

2.4 Deadlock

2.4.1 Overview

As we mentioned earlier in chapter one, a deadlock occurs because two or more agents require same resources in a circular manner during a particular time period. Obviously, in a complex distributed system such as the one used in this chapter, we can face many different kinds of deadlocks, which must be detected and possibly avoided. Below we explain different deadlock levels using illustrative examples.

Consider the situation where Machine M_1 and robot R_1 each have capacity of one. Machine M_1 is occupied with a part of part type PP_1, and suppose that this part has to be removed by R_1 (after the job is done by the machine) from M_1. Let us also suppose that R_1 has taken a part from another location to put into machine M_1, according to the given process plan (Figure 2-10). Since M_1 is occupied by another part and since that part must be removed by R_1 and R_1 is already busy with another part in its hand that has to go to M_1, we are in a deadlock situation. This is the simplest case of a deadlock.

We can prevent the system to go to these kinds of deadlocks by applying proper local rules. In the previous example of this chapter, the "take" action cannot happen, unless the entire path can be created by the search algorithm (The machine, which is the target destination of the entity, must be free). Hence we see that the local specifications will prevent this sort of deadlock in the system.

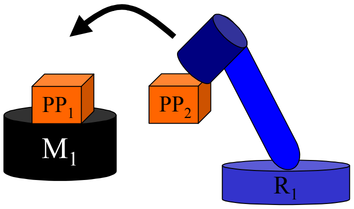

Figure 2-10: Deadlock

Now assume the following configuration: Machine M_1 is occupied with a part type PP_1 and machine M_2 is occupied with a part type PP_2. According to the process plans of PP_1 and PP_2, the part in M_1 must be transferred by R_1 to M_2 and the part in M_2 must be transferred by R_1 to M_1 (Figure 2-11). Obviously this is a deadlock situation, since we assume that R_1 is the only service provider agent for M_1 and M_2.

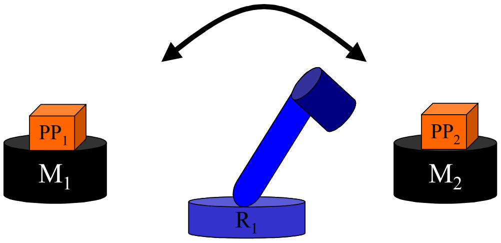

Figure 2-11: First Level Deadlock

Now consider another scenario: Machine M_1 is occupied with a part of type PP_1 and machine M_3 is occupied with a part of type PP_2. R_1 is the provider of machine M_1 and R_2 is the provider of machine M_3. M_2 is currently empty. According to the process plans, the part in M_1 must be transferred to M_2 and then to M_3 and the part in M_3 must be transferred to M_2 and then to M_1 (Figure 2-12). Since M_2 is currently free, we are not in a deadlock situation, but as soon as any of the robots moves the part to M_2, the system is deadlocked. This kind of deadlock is called *Impending Part Flow* deadlock (or *Second Level* deadlock), i.e. the deadlock will happen in the next two steps.

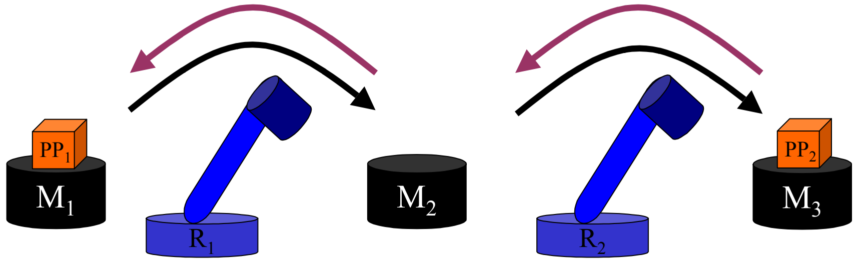

Figure 2-12: Impending Part Flow Deadlock (Second Level Deadlock)

This example can be extended to higher level deadlocks (Figure 2-13). In figure 2-13 we have a situation, where we have *n* machines and *n*-1 robots (service providers of the machines). With the same reasoning as in the previous case we can conclude that a (*n*-1)th level of deadlock can occur in this case.

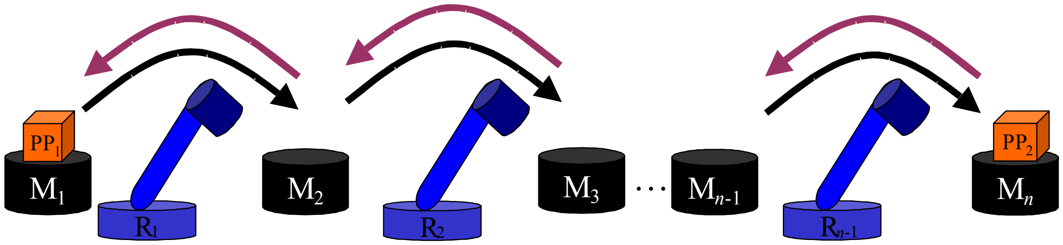

Figure 2-13: Impending Part Flow Deadlock ((n-1)th Level Deadlock)

A completely different type of deadlock can happen at execution level. Suppose that R_1 has found a path from M_1 to M_2 for a part of type PP_1. At the same time, R_2 finds a path from M_3 to M_2 for transferring a part of type PP_2 (Figure 2-14). Since M_2 is free, both R_1

and R_2 are able to generate a path from their respective sources to the single destination (M_2). The problem occurs when the two robots attempt to execute their planned tasks. Since the capacity of M_2 is only one, the robot working faster will seize the destination resource (M_2) and the other robot will go to a deadlock state. This type of deadlock depends on the timing of the events. The solution to this problem is the process of "locking the resources," which we already saw in the previous example - as soon as a resource is locked by an agent, it cannot be used by other agents. At the execution level the first step is to check the availability of all the required resources as planned by a synthesized solution. If all the resources are available, they will be locked by the agent. In this example R_2 will lock machine M_2 and therefore R_1 will not take its part from M_1 anymore, until M_2 becomes available again.

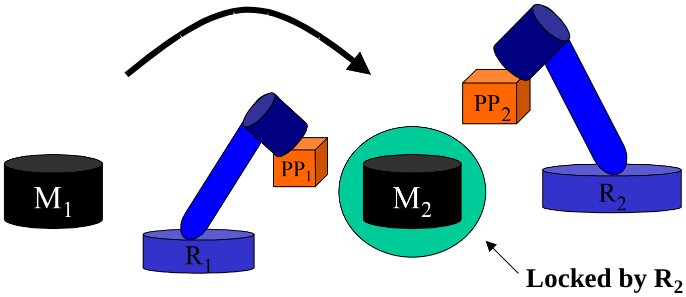

Figure 2-14: Timing deadlock

2.4.2 Approach

In a distributed environment, there is not a single centralized controller that can manage the issue of deadlock. Therefore we must present a framework through which agents can communicate and solve the deadlock problem collaboratively. When the cost estimates from all the providers for a request are infinity, there is a potential for a deadlock condition. Therefore the system must explore the situation and eventually look for the existence of a deadlock. This process includes three steps - The first step is to determine resources that are causing an agent not to find any path from a source to a destination. Having the list of unavailable resources, we can then start the process of detecting the deadlock (if there exist any), which will be the second step. The last step is to configure the system in a way that it does not go to the deadlock state again in the future. Figure 2-15 shows the three steps:

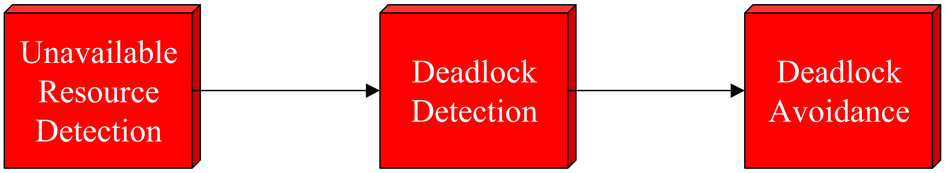

Figure 2-15: Deadlock Detection and Avoidance Steps

Before we describe our methodology, we characterize agents as *active* and *passive* agents. *Active agents* are those that are explicitly involved in a deadlock, and cause the deadlock. Examples of active agents can be machines and buffers. *Passive agents* are those that are not explicitly involved in deadlocks. A robot is an example of a passive agent. To show the difference between active and passive agents we refer to figure 2-11 again. We recall that deadlock occurred here because machine M_1 and machine M_2 cannot take any action after they are finished with their current tasks. But robot R_1 is not explicitly involved in this deadlock situation. Suppose that R_1 is also a provider for other two machines, M_3 and M_4. R_1 can easily transfer a part from M_3 to M_4 even though M_1 and M_2 are in deadlock situation.

In manufacturing applications it is reasonably straight-forward to distinguish between active and passive agents. Generally speaking, however, it is extremely difficult to distinguish between passive and active sources.

2.4.3 Deadlock Detection

Two separate algorithms will be presented for detection and avoidance of deadlocks. The deadlock detection itself includes two separate sub-algorithms. The first sub algorithm detects all the resources that cannot be used during the path-generation at synthesis level. The second sub-algorithm detects the deadlock. If the returned path obtained from synthesis is NULL, we then need to determine those resources that cause this. This will be accomplished by our first sub-algorithm, hereafter referred to by RDT (Resource detection algorithm). After finding the resources that are responsible for NULL solution, the deadlock algorithm will be called from these resources.

2.4.4 Resource Detection Algorithm (RDT)

This algorithm works based on a Breadth First Search strategy (section 2.3.2). If the search algorithm of the agent (usually the robot) fails to find a path from a source to destination, RDT is called. The objective is to find the list of resources that causing a NULL path solution. The algorithm works as follows:

I) All the agents in the reachability attribute of the agent *A* are listed as $\Phi_1, \Phi_2, ..., \Phi_n$. In other words from agent *A* point of view $\Phi_1, \Phi_2, ..., \Phi_n$ are known and reachable.

II) A breadth first search tree is created with the following nodes: $\{\Phi_1\}, \{\Phi_2\}, ..., \{\Phi_n\}, \{\Phi_1, \Phi_2\}, ..., \{\Phi_{n-1}, \Phi_n\}, ..., \{\Phi_2, ..., \Phi_n\}, \{\Phi_1, \Phi_2, ..., \Phi_n\}$

III) Every single node is examined, based on the breadth first search, and according to the following:
   a) For all the basic functions: all the preconditions corresponding to the resource or resources in the node is set to "true."
   b) The search algorithm is called. If the path is found, the cost of the generated path and the node is returned. The members of the nodes are the resources that cause the cost to be infinity.

IV) If there is no path, the cost of infinity is returned.

RDT will return the list of the resources that are currently not available. Based on this list we can initialize the deadlock detection algorithm, which is based on the Mitchell-Merritt algorithm.

2.4.5 Mitchell-Merritt algorithm

Mitchell and Merritt [Mitchell 84] create in their algorithm a TWFG, where the vertices are the tasks and each vertex has one *private label* and one *public label*. The private label (indicated by an index in the lower half of the node) is a unique variable and non-decreasing over time to the vertex. The public label (indicated by an index in the upper half of the node) is available to every task and is not necessarily unique. Initially the public and private labels are the same. The edges in the TWFG mean that one task is waiting for another. Four types of state transitions are used in this algorithm:

- When a process starts to wait for another resource it becomes a *blocked task (process)*. The labels of the blocked task become a value larger than their original value, and also greater than the public label of the process being waited on, which is also unique to the node. In the TWFG an edge is going to be drawn from the blocked task to the needed resource.
- A task becomes *active* when it gets the necessary resource. The edge will be removed from the waiting vertex.
- A *transmit* step occurs when a blocked task discovers that its public label is smaller than the public label of the process it is waiting on. In that case the waiting process will replace its public label with the one just read. This causes the larger labels to migrate along the edges of the TWFG.
- A (deadlock-) *detect* step occurs when a task receives its own public label back.

Example I

Consider the situation in figure 2-11 again. M_1 wants its part to be transferred to M_2 by R_1. It has to wait for M_2 because M_2 is occupied by a part. Therefore this task will be blocked. By the above algorithm two nodes will be created having labels 1 and 2. Based on blocked task transition step, both will get value 2 for label. On the other hand, M_2 wants its part to be transferred to M_1 by R_1. It has to wait for M_1, because it is occupied by a part. Therefore this task will be blocked too. When we update the graph, node M_1 will detect the same public label (2), hence a deadlock condition. Figure 2-16 shows these steps.

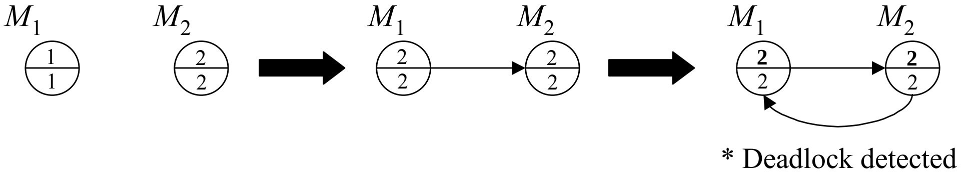

Figure 2-16: Example a

Example II

Consider figure 2-17, where we have the layout of a flexible manufacturing cell. M_1 needs M_2, M_2 needs M_3 and M_3 needs M_1. Since M_2 is occupied with part of type PP_2, the TWFG will create two nodes for M_1 and M_2. The graph will be expanded because M_2 has to wait for M_3. If we initially assign the public label 3 to node M_3, then the public label of M_1 and M_2 will change according to the steps in figure 2-18. Eventually M_3 will wait for M_1, which generated the cycle. Since both public labels are the same for M_1 and M_3, a deadlock will be detected.

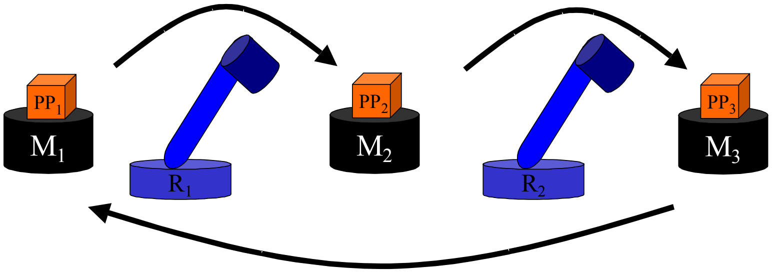

Figure 2-17: Example b – Layout

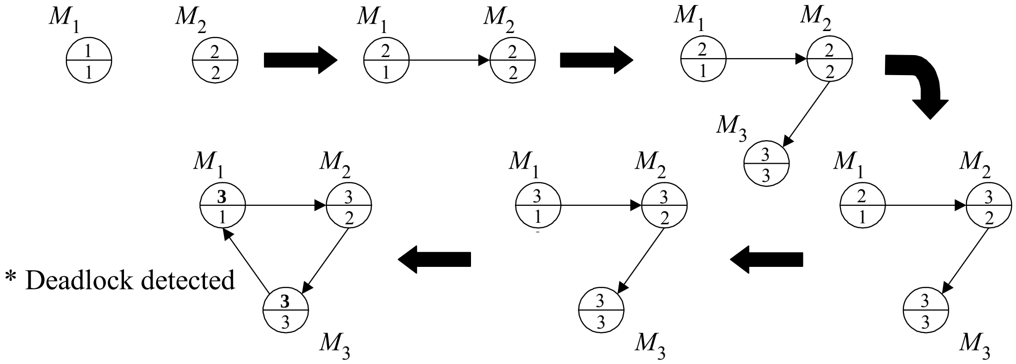

Figure 2-18: Example b - TWFG

2.4.6 Deadlock Avoidance

The last step in our methodology is to avoid the deadlock. In order to implement our methodology we assign a list of "bad" states to all the active agents. We call the list "To-Avoid-List" (Figure 2-19). Each element of this list is a list itself. Every such sub-list includes the state of all the active agents involved in a deadlock case, and have been detected in the previous steps.

Whenever there is any interaction between the agents, the active agent will check the To-Avoid-List to make sure the system does not make a transition to a bad state. When a request is sent to an active agent, it will predict its next state. Having the predicted state, the active agent will check its To-Avoid-List, to find whether the forecasted state is in one of the sub-lists. If the predicted state is found, a query will be sent to other active agents in the sub-list. If the current states of all the mentioned agents match the states in the sub-list, the system will eventually go to a deadlock state. Therefore the active agent will not allow the requester to execute the task, thus will avoid the deadlock.

Obviously, this method will only avoid the deadlocks of first level and second level types. An illustrative example will clarify this methodology.

$\Phi_1.S, \Phi_j.S, \ldots, \Phi_k.S$
$\Phi_1.S, \Phi_r.S, \ldots, \Phi_t.S$
$\Phi_1.S, \Phi_j.S, \ldots, \Phi_k.S$
• • •

Figure 2-19: To-Avoid-List of agent Φ_1

Example

Suppose that the situation in figure 2-17 has been detected as a deadlock condition. Upon detection, the system will update the To-Avoid-List of all the active agents involved in this matter (M_1, M_2 and M_3). These lists can be seen in figure 2-20. Some remarks in order here:

a) State of every machine also includes the state of the entity being worked on. The second item of entity's state is its instance number. Since there is only one instance of each entity type, we will never come back to the exact same state again, because the other parts will have a different instance number. Hence we will ignore the instance number in the To-Avoid-List and we will depict it with an "*" (don't care).

b) Stage and the availability of the machines do not affect the deadlock situation; therefore we consider them also as don't care ("*").

c) We assume the current stage numbers of the parts to be r, s and t.

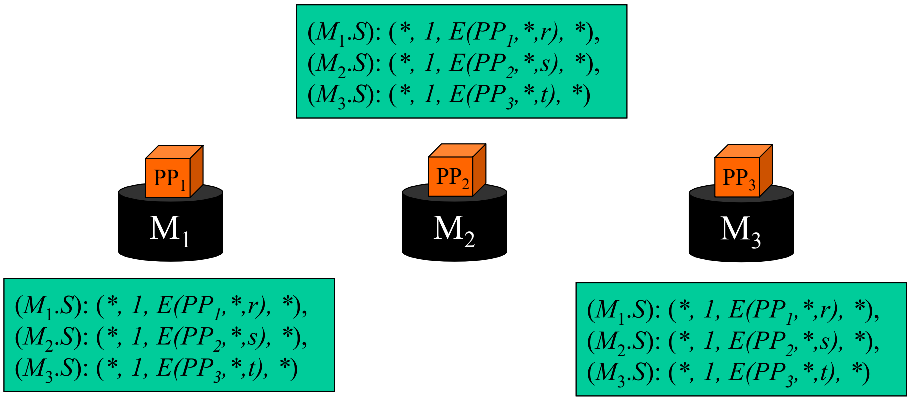
$(M_1.S)$: $(*, 1, E(PP_1, *, r), *)$,
$(M_2.S)$: $(*, 1, E(PP_2, *, s), *)$,
$(M_3.S)$: $(*, 1, E(PP_3, *, t), *)$
PP_1
PP_2
PP_3
M_1
M_2
M_3
$(M_1.S)$: $(*, 1, E(PP_1, *, r), *)$,
$(M_2.S)$: $(*, 1, E(PP_2, *, s), *)$,
$(M_3.S)$: $(*, 1, E(PP_3, *, t), *)$
$(M_1.S)$: $(*, 1, E(PP_1, *, r), *)$,
$(M_2.S)$: $(*, 1, E(PP_2, *, s), *)$,
$(M_3.S)$: $(*, 1, E(PP_3, *, t), *)$

Figure 2-20: To-Avoid-Lists

After the recovery process, which can be through some manual operations (like system reset), the system starts to work again. Now suppose that the system is in the state shown in figure 2-21. Assume that the state of the part of type PP_1 is $E(PP_1, *, r)$, the state of the part of type PP_2 is $E(PP_2, *, s\text{-}1)$ and the state of the part of type PP_3 is $E(PP_3, *, t)$. Let us assume that PP_2 has to be taken from its current location (such as M_4) and be put into M_2 (according to a given process plan). R_1 will send a request to M_2 and will wait for its response. M_2 will predict its next state which is $(M_2.S)$: $(*, 1, E(PP_2, *, s), *)$. It will check its To-Avoid-List with the predicted state. Since they match, M_2 will send queries to M_1

and M_3. M_1 and M_3 will respond to these queries by sending their current states to M_2. Since the current states of M_1 and M_3 match with the states in the To-Avoid-List of M_2, M_2 will conclude that the system eventually goes to a deadlocked state. Therefore it will not accept the request of R_1. And hence the robot will not take the part from M_4 until M_1 or M_3 change their states.

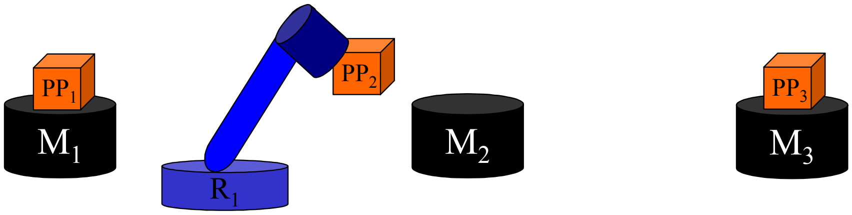

Figure 2-21: A state before the deadlock

2.5 Summary

In this chapter we presented our preliminary results on control synthesis algorithm for agents which are selfish. We assumed that the agent's environment is deterministic and fully observable. The agent is not subject to failures, but it is possible to reach to fault conditions, specifically deadlocks. Two search algorithms were presented and it was also shown how to embed deadlock detection and avoidance in this algorithms.

3. An Agent and Its Environment – Control Synthesis

3.1 Introduction

In the previous chapter we assumed that the environment is fully observable, thus the agent is aware of environment's state at any given time. We also assumed the environment to be deterministic, that is, the outcome of agent's interaction with the environment is fully predictable. By *environment* we mean anything which falls outside of the agent's boundary. We understand that this is a loose definition, and depends on the level of control and functionality provided by the agent. In real life applications, the environment is not always deterministic, and therefore, the outcome of actions taken by the agent and their impact on the environment are not fully predictable. Furthermore, at any point in time, an agent only has a partial knowledge on state of its environment. The imperfect knowledge on the state of environment and non-deterministic nature of interactions between agent and its environment pose a major challenge on our synthesis solution methodology. In this chapter we will focus our attention on selfish agents which interact with a probabilistic and partially known environment. The agent has its own set of tasks to carry out, but to synthesize and obtain a cost effective task plan, it must interact with its environment and manipulate it, if necessary. We note that the agent can be part of a larger distributed system, but for the sake of our analysis here, they are all considered as part of the environment. Since our agent is selfish and has no intention of collaborating with its environment, it could decide to take control actions without seeking any consensus from the environment. It only hopes that the environment will respond positively to its course of actions. We recall that due to the probabilistic nature of these interactions, the outcomes can only be measured on some probabilistic scale. Our agent manipulates the environment either for the purpose of avoiding some adverse and disruptive conditions, or for the purpose of achieving some desirable or favorable goals. In this chapter we will discuss both and show that under some circumstances these two problems can be dealt with in a similar way.

3.2 Problem Formulation

In traditional control synthesis problems, given a fixed plant model $\mathscr{P}$ and control specification $\mathfrak{I}$, the objective is to compute a controller $\mathfrak{C}$, such that $\mathscr{P} \wedge \mathfrak{C} \rightarrow \mathfrak{I}$. While the overall problem remains the same, there are a number of major differences between our formulation and the traditional ones [Ramadge 87a, b]: (1) Our agent intends to synthesize a controller for the purpose of manipulating its environment. To that end, plant model $\mathscr{P}$ describes the perception of the agent from its environment. The true $\mathscr{P}$, while unknown, can only be approximated by making series of observations and model identifications. (2) The plant model is probabilistic here. (3) Since the approximated model remains incomplete, there is no guarantee that the model will not change when the agent reinforces an action against its environment. Furthermore, these changes are completely unknown to our agent, and can only be learned through explore and exploit mechanisms. (4) Specification $\mathfrak{I}$ can only be defined on a probabilistic scale. It is defined to ensure that disruptive state(s) are avoided in the environment; favorable conditions are reached in the environment; or a mixture of the two. (5) Since our agent only observes events from the environment (and that is only partially), the control specification $\mathfrak{I}$ is initially defined based on these observable events. To solve the problem at hand, we will need to convert this specification to one defined according to states (and not events).

3.2.1 Background and Preliminaries

In all scenarios that we cover in this chapter we assume that the agent has a set of *sensors* to observe its own states and the inputs from the environment. We also assume that the agent has basic functions some of which can be used to alter state of the environment. Our solution approach to this problem will utilize *reinforcement learning*. The objective in reinforcement learning is to learn a *control strategy*, or *policy*, for choosing actions that achieve a specific goal. In reinforcement learning the actions of the agents are directly related to a reward function that assigns a numerical value to each distinct action the agent may take from each distinct state. This reward function can be built into the agent and is a feedback that the agent receives from its environment when an action is performed. This is illustrated in Figure 3-1.

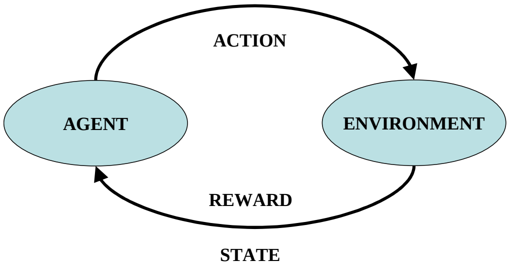

Figure 3-1: Reinforcement Learning

Generally speaking, an *agent builds a sequence of actions, observes the consequences and learns a control policy* $\pi : S \rightarrow A$, which defines appropriate actions *a* from set *A*, given the current state $s \in S$. The best control policy is the one that maximizes the accumulated reward over time by the agent.

To solve this problem, *Markov Decision Processes* (MDP) has been used in the literature [Mitchell 97]. In a Markov Decision Process (MDP) the agent has a set of actions *A* that it can perform in a set of states *S*. At each discrete step *t*, the agent senses the current state, s_t, selects and performs an action, a_t (From now on we may use *a* for an action and *f* for a basic function interchangeably). The environment responds to this action by rewarding the agent, valued at $r_t = r(s_t)$, and generating a succeeding state $s_{t+1} = \delta(s_t, a_t)$. Note that δ and *r* are part of the environment and not necessarily known to the agent (They can be even nondeterministic, as we will see later in this chapter). We assume that *S* and *A* are finite. The agent is seeking a policy that produces the *greatest possible expected reward over time, also known as discounted cumulative reward*, *defined by*

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \qquad (3\text{-}1)$$

In this equation γ is the *discount factor* which has a value between 0 and 1 and it determines the relative value of delayed versus immediate rewards. In case of probabilistic outcomes, we use expected outcome as follows:

$$V^{\pi}(s_t) \equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right] \qquad (3\text{-}2)$$

To simplify the notation, we show s_t as *s* from now on. Given a fixed policy π, its value function V^{π} satisfies the **Bellman** equations:

$$V^{\pi}(s) = r(s) + \gamma \sum_{s' \in S} P_{sa}(s') V^{\pi}(s') \qquad (3\text{-}3)$$

In these equations P_{sa} is the state transition probabilities. P_{sa} gives the distribution over all possible states that can be reached from state *s* by taking action *a*. Note that in case of deterministic action-transition functions, P_{sa} is either 0 or 1 for each set of s and a. The above equation states that the expected sum of discounted rewards $V^{\pi}(s)$ in state s is the sum of immediate reward $r(s)$ (sometimes shown as $r(s, a)$) and the expected sum of future discounted rewards. These equations can be used to efficiently solve for V^{π}. A version of Bellman's equations for the optimal value function is as follows:

$$V^*(s) = r(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s')V^*(s') \qquad (3\text{-}4)$$

The action that attains the maximum in equation (3-4) is:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s')V^*(s') \qquad (3\text{-}5)$$

The objective of the agent is to find a policy that maximizes V^{π}, that is,

$$\pi^* \equiv \arg\max_{\pi} V^{\pi}(s) \quad \forall s \qquad (3\text{-}6)$$

Once the states, actions, immediate rewards and the discount factor are defined, the agent can start calculating the optimal policy by learning V^*(its value function) and using equations (3-4) and (3-5). The agent can acquire the optimal policy by learning V^*, assuming that it has perfect knowledge about immediate reward function r and the state transition function δ. Two algorithms called *value-iteration* and *policy-iteration* can solve the Bellman equations and find the best possible policy or control actions for each state [Sutton 98].

In terms of agent's knowledge of its environment and also the nature of outcomes from the environment, there are a number of scenarios as listed below:

I) Known and fully observable and deterministic environment – Agent's actions and outcomes are deterministic

II) Known and fully observable but probabilistic environment – Agent's actions and rewards are probabilistic – An optimal policy can be learned using Bellman equations – equations (3-4) and (3-5).

III) Unknown environment but deterministic – Agent's actions and rewards are deterministic – Agent has to learn the environment once and forever.

IV) Unknown and probabilistic environment – Agent's actions and rewards are probabilistic – Agent must explore and exploit the environment.

We did discuss scenario I comprehensively in the second chapter. Now let us discuss the more complicated scenarios where the agent cannot predict next states (outcome of actions) nor it can predict the possible rewards. This problem can be solved using one of the reinforcement learning algorithms, the *Q*-Learning algorithm. In case of deterministic environments the value of *Q* is the reward received immediately upon applying action *a* as the first action from state *s*, plus the value of the following optimal policy thereafter. That is,

$$Q(s,a) = r(s) + \gamma V^*(\delta(s,a)) \qquad (3\text{-}7)$$

This value is maximized in equation (3-5), to choose the optimal action *a* in state *s*. Hence:

$$\pi^*(s) = \arg\max_a Q(s,a) \qquad (3\text{-}8)$$

This rewrite is important, because it shows that if the agent learns the *Q* function instead of V^* function, it will be able to select optimal actions even when it has no knowledge of the functions *r* and *δ*. In each state the agent needs to consider each available action *a* in its current state *s* and choose the one that maximizes *Q*(*s*,*a*).

Q-Learning [Mitchell 97]

The key problem is to find a reliable approach to estimate training values for Q, given only a sequence of immediate rewards r spread out over time. This can be accomplished through *iterative approximation*. There is a close relationship between Q and V^*:

$$V^*(s) = \max_{a'} Q(s, a') \qquad (3\text{-}9)$$

Equation (3-9) allows rewriting equation (3-7) as:

$$Q(s,a) = r(s) + \gamma \max_{a'} Q(\delta(s,a), a') \quad (3\text{-}10)$$

This recursive definition of Q provides the basis for algorithms that iteratively approximate Q. $\hat{Q}$ is the learner's estimate of the actual Q. In this algorithm the leaner represents its estimate by a large table with a separate entry for each state-action pair. That is, the table entry for the pair $\langle s,a \rangle$ stores the value for $Q'(s,a)$, which will be learner's current hypothesis about the actual but unknown $Q(s,a)$. This table will be initially filled with random values. The agent repeatedly observes its current state s, chooses some action a, executes this action and then observes the resulting reward $r = r(s,a)$ and the new state $s' = \delta(s,a)$. It then updates the table entry for $\hat{Q}(s,a)$ following each such transition, according to the rule:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} Q'(s', a') \qquad (3\text{-}11)$$

This training rule uses the agent's current $\hat{Q}$ values for the new state s' to refine its estimate of $\hat{Q}(s,a)$ for the previous state s. With this method the agent does not need to know the general functions $\delta(s,a)$ and $r(s)$. Instead it executes the action in its environment and then observes the resulting new state s' and reward r.

Q-Learning Algorithm

Algorithm: *Q-Learning*

Input: Table entry $\hat{Q}(s,a)$ for each State s, Action a combination

Output: Table $\hat{Q}(s,a)$

```
procedure Q_Learning ($\hat{Q}(s,a)$ : 2-dimensional array)
var
   r: Real
   $s$, $s'$: State
   $a$: Action
begin
   For each $s$, $a$ initialize the table entry $\hat{Q}(s,a)$ to zero
   Observe the current state $s$
   Do forever:
      Select an action $a$ and execute it
      Receive immediate reward $r$
      Observe the new state $s'$
      Update the table entry for $\hat{Q}(s,a)$ as (3-11)
      $s \leftarrow s'$
   end Do
end
```

For case of probabilistic reward and actions (non-deterministic outcomes) the functions $r(s)$ and $\delta(s,a)$ can be viewed as first producing a probability distribution over outcomes based on s and a and then drawing an outcome at random according to this distribution. With this assumption the Q learning algorithm for deterministic case can be extended to handle nondeterministic MDPs. The value V^{π} of a policy π is defined as in equation (3-2). We now generalize the definition of Q, again by taking its expected value:

$$
\begin{aligned}
Q(s,a) &\equiv E\left[r(s) + \gamma V^{*}(\delta(s,a))\right] \\
&= E[r(s)] + \gamma \sum_{s'} P_{sa}(s')V^{*}(s')
\end{aligned}
\qquad (3\text{-}12)
$$

where $P_{sa}(s')$ is the probability that taking action a in state s will produce the next state s'. So we can re-express Q recursively:

$$
Q(s,a) = E[r(s)] + \gamma \sum_{s'} P_{sa}(s') \max_{a'} Q(s',a') \qquad (3\text{-}13)
$$

To solve these equations we need a new training rule. Our earlier rule will fail in case of nondeterministic environments, because it will fail to converge. This difficulty can be overcome by modifying the training rule so that it takes a decaying weighted average of the current $\hat{Q}$ value and the revised estimate. Writing $\hat{Q}_n$ to denote the agent's estimate on the nth iteration of the algorithm, the following revised training rule is sufficient to assure convergence of $\hat{Q}$ to Q:

$$
\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n\left[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')\right] \text{ where } \alpha_n = \frac{1}{1+visits_n(s,a)} \qquad (3\text{-}14)
$$

Where s and a here are the state and action updated during the nth iteration and where $visists_n(s,a)$ is the total number of times this state-action pair has been visited up to and including the nth iteration. By reducing α at an appropriate rate during training, we can achieve convergence to the correct Q function.

Scenario IV above is the one which resembles our case the best, except for the fact that in the above scenarios, the agent and environment states are combined into a single state, whereas in our case, these states are clearly separate from each other. Furthermore, in our case, the reward function is defined by the agent, but its value, though probabilistic, is evaluated based on the direct or indirect response of the environment to the agent's *actions. A word of caution is in place here – since the agent and environment are each acting selfishly, there may not be any response (negative or positive) from the environment against an action taken by the agent. In such a case, the reward of this action, as compared to taking no action, is nil. On the other hand, due to its own dynamics, the environment may respond positively or negatively to the action taken by the agent. This can be seen as a change in transition probabilities between its states, additional states or transitions, etc.*

3.2.2 Control Synthesis for the Selfish Agent – Manipulation of Environment

Let us now move into our own problem. We will start by stating some assumptions:

1) The environment is initially unknown, probabilistic, and only partially observable.
2) The environment is considered as an uncontrolled process, but can possibly be manipulated by the agent using its basic functions.
3) The agent does not have any state/transition model of the environment. It only receives some input signals from the environment, and it interprets them as events.
4) Environment's model, as seen by the agent, is assumed to be a Deterministic Stochastic Finite Automaton (DSFA).

To apply any synthesis solution methodology, we need to first identify the initial perception model of the environment. Having done that we can then apply the *Q*-learning algorithm described above to reach the goal. In section 3-3 you can find the identification methodology using regular language theory. To better appreciate our issues here, we will give an analogy from continuous control theory.

In continuous control theory we define the process of determination of the equations of a model's dynamics as “model identification.” This can be done off-line: for example, executing a series of measures from which to calculate an approximate mathematical model, typically its transfer function or transfer matrix. Such identification from the output, however, cannot take into account the underlying unobservable dynamics. Even assuming that a "complete" model is used, all the parameters included in these equations (called "nominal parameters") are never known with absolute precision: therefore the control system will have to behave correctly even in absence of their true values. Some advanced control techniques include an "on-line" identification process (“Adaptive Control”). The parameters of the model are calculated ("identified") while the controller itself is running: in this way, if a drastic variation of the parameters ensues (for example, if the robot's arm releases a weight), the controller will adjust itself consequently in order to ensure the correct performance (See figure 3-2). In this chapter we will use a similar concept for the identification of the underlying model, but within context of the discrete event control theory. The method of identification is based on the regular language theory and proper sampling.

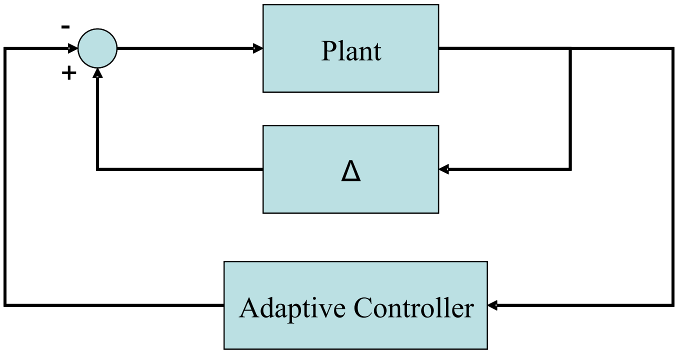

Figure 3-2: Adaptive Control Loop

Before we start with the details of our solution approach we will provide an example.

3.2.3 Example I

The following example will clarify our approach to solve the problems that need identification and cost estimation. The solution to this example requires understanding of three major concepts that are being addressed in this thesis. The first one is the concept of "identification." The agent has to identify the unknown or partially known environment. The identification includes recognition of the state/transition model of the environment. Obviously this model is only for the transitions that are observable to the agent. The second concept is the "risk analysis" and "fault avoidance" concept. And the third concept is the concept of "global search with learning" in partially known and probabilistic environments.

Figure 3-3 shows the example. There are 5 machines and two robots. Robot R_1 is currently located at M_3. Part P_1 arrives at M_1, which has to first visit either M_2 or M_5 and then move to M_4 according to the process plan. Based on the specifications of the agent if it chooses the path from M_1 to M_2, it has to visit the black box that is located in the path between M_1 and M_2 and perform a process control that is defined in its specifications (For example the robot has to prevent the black box of transitioning to a bad state). The agent does not have any information about neither the model of the process running in the black box, nor about how it can influence the behavior of that process – the only known information is that the bad event occurring in the black box must be avoided. On the other hand if the robot chooses the path from M_1 to M_5, there is the risk of going to a known global bad state (Deadlock happens if R_1 takes P_1 and waits for M_5 to become empty, while R_2 is doing the same) since R_2 is bringing in part P_3 into M_5, even though M_5 is occupied with P_2. For the agent (R_1) to compute the most cost effective path, it may need to explore the box, and possibly manipulate it (if possible) as well as calculate the risk and may be do some collaboration with R_2. In this chapter we will only discuss how the agent can identify and control the black box. Risk analysis and collaboration will be done in chapter four. The cost effectiveness of this solution compared to other paths and synthesizing task plans for the robot will be discussed in chapter five.

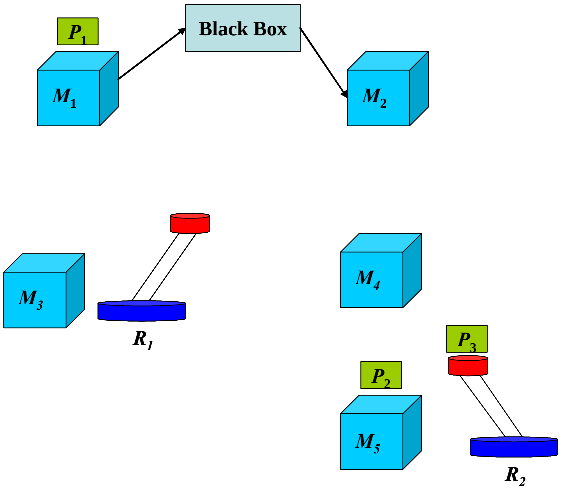

Figure 3-3: A manufacturing system – Example

3.3 Identification – Perception Model Building

In this section we discuss how an agent can create a perception model of its environment. Perception provides agents with information about the world they inhabit and is initiated by sensors [Sutton 98].

We make the following assumptions:

I) States of the environment have initially <u>no</u> meaning to the agent. What the agent knows about its environment is through the events that it can observe. And it cannot observe all the events taking place in its environment.

II) Using its sensors, the agent has the capability to catch certain events (alphabets in the stochastic regular language), that are predefined in it. These events are equivalent to transitions in the environment's model.

III) Control objective related to the environment is initially defined to the agent in the context of *events*. After the uncontrolled environment is identified, the control objective is redefined using the environment's states, which have been created in the initial perception model.

IV) Environment will be modeled as a deterministic stochastic finite automaton (DSFA) which always starts from a certain known initial state.

3.3.1 Preliminaries

Let $₤$ be a finite alphabet, $₤^*$ the set of all strings on $₤$ and λ the empty string such that for every symbol e in $₤$ satisfies $e\lambda = \lambda e = e$. A stochastic finite automaton (SFA), $A =$

($\mathfrak{E}$, Q, P, q_1) consists of an alphabet $\mathfrak{E}$, a finite set of nodes $Q = \{q_1, q_2, \dots q_n\}$, with q_1 the initial node and a set of probability matrices $p_{ij}(e)$ giving the probability of event $e \in \mathfrak{E}$ such that the model makes a transition from node q_i to node q_j led by the symbol e in the alphabet. If we assume that p_{if} is the probability that the string ends at node q_i, the probability $p(w)$ for the string w to be generated by A is defined by:

$$p(w) = \sum_{q_j \in Q} p_{1j}(w) p_{jf} \qquad \text{(3-15)}$$

$$p_{ij}(w) = \sum_{q_k \in Q} \sum_{e \in E} p_{ik}(we^{-1}) p_{kj}(e) \text{ where } w = xe \text{ and } x = we^{-1} \qquad \text{(3-16)}$$

The language generated by the automaton A is defined as $L = \{w \in \mathfrak{E}^* : p(w) \neq 0\}$. The language that is generated by means of a stochastic finite automaton is called *stochastic regular language* (SRL).

Two SRL are said to be equivalent if they provide identical probability distributions over $\mathfrak{E}^*$. It is not enough that two languages L_1 and L_2 include the same strings for them to be equivalent, also the probability of every string must be equal:

$$L_1 \equiv L_2 \Leftrightarrow p_1(w) = p_2(w) \forall w \in \mathfrak{E}^* \qquad \text{(3-17)}$$

In this work we limit ourselves to *deterministic stochastic finite automata (DSFA)*. This means that for every node $q_i \in Q$ and symbol $e \in \mathfrak{E}$, there exists at most one node such that $p_{ij}(e) \neq 0$. In such cases, a transition function $k = \delta(i, a)$ can be defined. This function gives the final node q_k for the transition starting at q_i and driven by symbol a. The probability of this single transition will be denoted by $p_i(e)$ [Carrasco 94].

3.3.2 Identifying Stochastic Regular Languages

A complete sample consists of two subsets: S_+ with strings in L (positive examples) and S_- with stings not in L (negative examples). If only S_+ is presented, then S is a positive sample. The algorithm *identifies in the limit* L if adding new examples to S may only produce a finite number of changes of hypothesis. Negative examples may cause the algorithm to reject language L' whose only difference with L lays on $L' - L$. The concept of identification in the limit was first introduced by [Gold 67]. In that model, an algorithm $\mathscr{A}$ reads strings of a presentation and for each string it conjectures a DFA as the solution. We say that algorithm $\mathscr{A}$ identifies the DFA A in the limit if and only if for any presentation of $L(A)$ the infinite sequence of automata conjectured by $\mathscr{A}$ converges to a DFA that accepts the same language as the original A. The convergence point depends on the presentation.

Samples of SRL consist only of positive examples which appear repeatedly. Nevertheless, the statistical regularity is able to compensate the lack of negative data. Using ideas from [Oncina 92], [Carrasco 94] introduced in their work an algorithm which identifies, in the limit, stochastic regular languages. This algorithm does not grow exponentially with the size of S. They use the fact that the probability of appearance of every string follows a well defined distribution.

The algorithm first builds the *prefix tree* T from S and evaluates at every node the relative frequencies of the outgoing arcs, incorporating this information in T. We will write as n_i the number of strings arriving at node q_i, $f_i(e)$ the number of strings following arc $\delta_i(e)$, and $f_{\mathrm{i}}(\#)$ the number of strings ending at node q_i. The quotients $f_{\mathrm{i}}(e)/n_{\mathrm{i}}$ and $f_{\mathrm{i}}(\#)/n_{\mathrm{i}}$ estimate the probabilities $p_{\mathrm{i}}(e)$ and p_{if} respectively. The algorithm compares couples of nodes (q_i, q_j), varying j from 2 to t and i from 1 to j-1. *Equivalent* nodes are shown as ($q_i \equiv q_j$). As T is built, equivalent nodes have equal outgoing transition probabilities for every symbol $e \in \mathcal{E}$ and the destination nodes must be equivalent too; or mathematically we can say:

$$q_i \equiv q_j \Rightarrow \forall e \in \mathcal{E} \begin{cases} p_i(e) = p_j(e) \\ \delta_i(e) \equiv \delta_j(e) \end{cases} \qquad (3\text{-}18)$$

This provides a criterion in order to reject equivalence of nodes. However, experimental data are subjected to statistical fluctuations and equivalence must be accepted within a confidence range. In such case, the nodes will be called *compatible*. A confidence range for a Bernoulli variable with probability p and observed frequency f out of n tries is given by the Hoeffding bound [Hoeffding 63]:

$$\left| p - \frac{f}{n} \right| \prec \sqrt{\frac{1}{2n} \ln \frac{2}{\alpha}} \text{ with probability larger than } (1\text{-}\alpha) \qquad (3\text{-}19)$$

A recursive algorithm called ALERGIA has been designed [Carrasco 94] to identify DSFA. The algorithm is given in Appendix.

There are two types of errors where the algorithm could fail when looking for a solution:

1. (type α) rejection of compatibility between two equivalent nodes,
2. (type β) merge of two non-equivalent nodes.

[Carrasco 94] shows that type α error is bounded by $2\alpha\left(|\mathcal{A}|+1\right)t$, where α is the confidence level, t is the size of the prefix tree and $|\mathcal{A}|$ is the size of the alphabet set. Using the

Chebychev's inequality [Feller 50], [Carrasco 94] also shows that the upper bound of the type error β is B, where:

$$B = \begin{cases} \left(|\delta p| - \varepsilon\right)^{-2} Var(\delta f) & \text{if } \varepsilon \prec |\delta p| \text{ and } Var(\delta f) \prec \left(\delta p - \varepsilon\right)^2 \\ 1 & \text{otherwise} \end{cases} \qquad (3\text{-}20)$$

In these equations, we have: $\varepsilon = \sqrt{\frac{1}{2}\ln\frac{2}{\alpha}}\left(\frac{1}{\sqrt{n}} + \frac{1}{\sqrt{n'}}\right)$, $|\delta p| = |p_1 - p_2| = \left|\frac{f_1}{n_1} - \frac{f_2}{n_2}\right|$ and $Var(\delta f) = \frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}$. We can see that B vanishes with s (sample size), because $Var(\delta f)$ tends to zero and so does ε.

Experimentally, it can be shown that ALERGIA needs a very short time and comparatively small samples in order to identify a regular set. Evan for large samples, only a linear time is needed [Carrasco 94].

3.3.3 An Illustrative Example (Example II)

Let us consider the following chemical process in which temperature and pressure are changing. A heater is working next to this plant which may cause the temperature to rise. Think of an agent which needs to somewhat interact with and manipulate this process (it is like the black box in our previous example). This agent cannot observe the heater, but can only sense the effects of the heater (temperature/pressure going up or down). The agent can also turn on or off a fan in order to regulate the temperature within the process. The objective is to identify the uncontrolled process model of this plant using a positive collection of samples. For the agent to interact with this process, it must make sure that it does not reach to any disruptive or adverse condition (such as explosion when both temp.

and pressure are too high). At this point we only identify the uncontrolled process and leave the control part for later. Figure 3-4 shows the state/transition model of this plant represented as a deterministic stochastic finite automaton (DSFA) which should be recognized by the agent. We assume that the agent can use the "reset" function to reset the system to its initial position after visiting the only disruptive state (marked as x in the figure below).

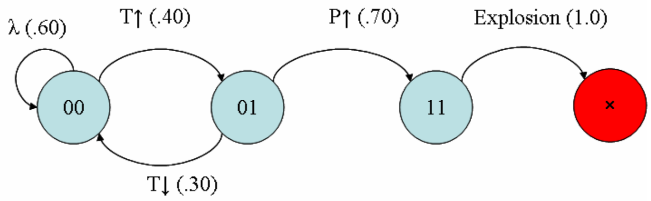

Figure 3-4: Chemical Process Model

In this model T↑, T↓ and P↑ are events showing that the temperature and pressure have passed certain thresholds. Event "Explosion" shows that the chamber has exploded. The agent can sense these events using special sensors (switches). λ indicates an empty string. The numbers in front of each event indicates the probability of making that transition in that state. λ is the empty string. Now suppose that the sample set shown in figure 3-5 is collected by the agent.

λ	15	T↑T↓	2
T↑	12	T↑T↓T↑	2
T↑P↑ex	20	T↑T↓T↑P↑ex	4

Figure 3-5: Sample Set *S*

With the choice of α = 0.8, one has:

$$\gamma := \sqrt{0.5\ln\frac{2}{\alpha}} \approx 0.67$$

As shown in figure 3-6, the algorithm starts by building the PTA. Each node is labeled with a number corresponding to its lexicographic order. In brackets, the number of strings arriving and terminating at the node are plotted. Every arc has a label with the symbol (Available events) including the transitions, and in brackets appears the number of strings using that arc. Next, the algorithm checks if nodes 2 and 1 are equivalent (compatible). Since there is no common event going out of these nodes, they cannot be equivalent; therefore the algorithm jumps to the next comparison between nodes 1 and 3. The termination probabilities shown below are similar:

$$|p-p'| = \left|\frac{15}{55}-\frac{2}{8}\right| \approx 0.0227 \prec \gamma\left(\frac{1}{\sqrt{n}}-\frac{1}{\sqrt{n'}}\right) = 0.67\left(\frac{1}{\sqrt{55}}+\frac{1}{\sqrt{8}}\right) \approx 0.3272$$

Also, the outgoing transitions have similar probabilities:

$$|p-p'| = \left|\frac{40}{55}-\frac{6}{8}\right| = 0.0273 \prec \gamma\left(\frac{1}{\sqrt{n}}-\frac{1}{\sqrt{n'}}\right) \approx 0.3272$$

Here, we observe that nodes 1 and 3 have the potential of being compatible, but first we have to see whether $\delta_1(T\uparrow) \equiv \delta_3(T\uparrow)$ is true or not. In order to verify this condition, the algorithm uses its recursive property and checks first the children of these nodes.

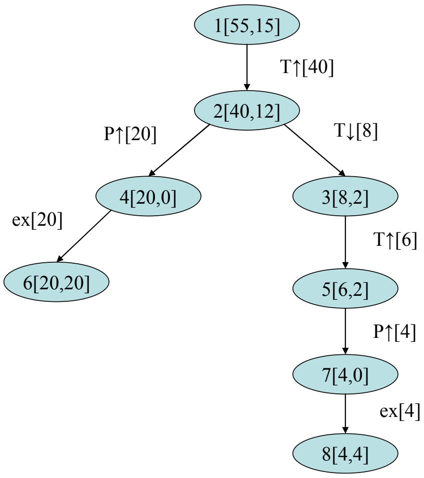

Figure 3-6: Prefix Tree Acceptor (PTA)

By executing ALERGIA we find out that nodes 4 and 7 as well as nodes 6 and 8 are compatible and can be merged. The resulting prefix tree is shown in figure 3-7. By continuing the algorithm, we discover that nodes 2 and 5 as well as nodes 1 and 3 are also compatible and can be merged. The result is shown in figure 3-8. Figure 3-9 shows the final uncontrolled DSFA as hypothesis. We observe that the structure of this finite automaton is the same but the probabilities have been only roughly estimated, due to the small size of the sample. Obviously we need larger samples in order to find more accurate probabilities and in order to choose a reasonable confidence level.

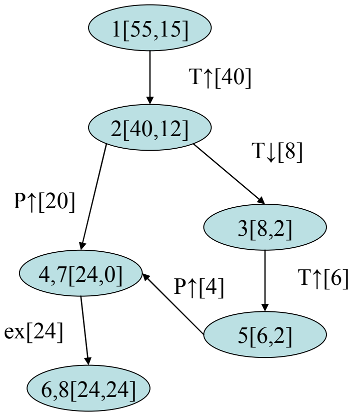

Figure 3-7: Prefix Tree Acceptor after merging states 6-8 and 4-7

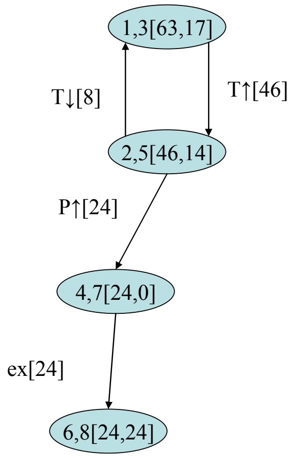

Figure 3-8: Prefix Tree Acceptor after merging states 2-5 and 1-3

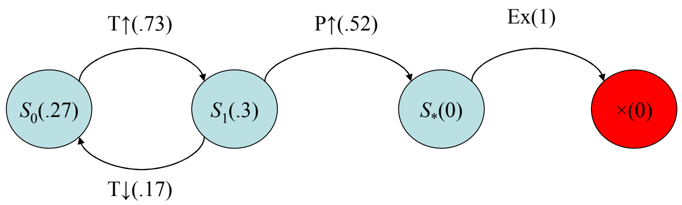

Figure 3-9: Identified model

3.4 Control Synthesis

As mentioned earlier in this chapter, our agent manipulates its environment for one of the two reasons; either avoid some adverse or disruptive condition(s), or reach a favorable condition or state. Here we will discuss the former goal. The latter one will be discussed later.

Theoretically speaking, this problem can be formulated as a MDP, and can be solved by a *Q*-Learning approach, assuming that the underlying environment state/transition model exists. While we can always start from the initial perception model identified above, we note that with every control action of the agent, there is a possibility that this model can change not only parametrically but also structurally. Since we are interested in reducing the overall probability of reaching bad state(s), the calculations should be done on the perception model identified after a control action is taken. Also, due to changing underlying models (after each control action), the computation of the future cumulative reward becomes rather very challenging. Thus, here we present an approximate methodology to synthesize a controller for the agent.

3.4.1 The Problem of Avoiding Bad State(s)

To formulate this problem we note that the agent can only see the “events” and has no knowledge about the “states” of its environment. Therefore, any objective for manipulating the environment must be stated in terms of these events. From these events and also the perception model built, the agent then recognizes the state or condition which must be avoided. The objective must be specified in some probabilistic terms, since the model of the environment is stochastic. As we can see in figure 3-10, the agent must cut off all the possible paths to the bad state. In case of probabilistic systems, this can be done by reducing the overall transition probability to the bad state. To formulate the criterion for satisfying this goal, we assume that the underlying perception model of the environment is a homogeneous Markov chain with a single absorbing state (we consider the bad state as the absorbing state). Then we calculate the probability of absorption using a phase-type distribution, assuming that the chain starts in a transient state.

As for the solution, the agent tries to take those actions which reduce the probability of absorption (reaching the adverse state). It stops when this probability reaches a certain specified threshold. Hence, $\Im$ can be defined such that the overall probability of going to absorbing state from transient states is less than a given threshold. The tendency in this approach is to reduce probability of accessing the bad state from its close by neighboring state, so that the maximally permissive property of the model can still hold. Next we present the details. We start with some basics of Markov chains and phase-type distribution.

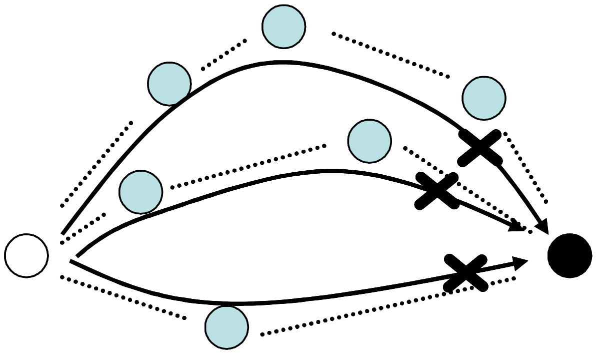

Figure 3-10: Cutting off the possible paths to the bad state

3.4.2 Markov Chains

To specify a Markov chain model we only need to identify:

1. A state space
2. An initial state probability $p_0(q) = P[X_0=q]$ for all $q \in Q$
3. Transition probabilities $p(i, j)$, where q_i is the current state and q_j is the next state.

In DSFA, A = (Ē, Q, P, q_1), state transitions are driven by events belonging to the alphabet set Ē. And the transition probabilities are expressed as $p_{ij}(e)$, where e belongs to Ē. In Markov chains, however, we will only be concerned with the total probability $p(i, j)$ of making a transition from q_i to q_j, regardless of which event actually causes the transition. By applying the rule of total probability we get:

$$p(i, j) = \sum_{e \in Ē} p_{ij}(e) p_i(e) \qquad \text{(3-21)}$$

where $p_i(e)$ is the probability that event e occurs at state q_i. If the transition probabilities do not change over time (the probability of going from state q_i to state q_j is the same as it will be at any other time in the future), then we call the Markov chain a *Homogeneous Markov Chain*, which will be the case throughout our discussion.

3.4.3 Phase-type Distribution

A *phase-type distribution* [Neuts 81] consists of a "general" mixture of exponentials and is characterized by a *finite* and *absorbing* Markov chain. The number *n* of phases in the PH distribution is equal to the number of transient states in the associated (underlying) Markov chain. A PH distribution represents random variables that are measured by the time *T* that the underlying Markov chain spends in its transient portion till absorption. Each of the states of the Markov Chain represent one of the phases. Some of the probability distributions that can be considered as special cases of a phase-type distribution are:

- Exponential distribution - 1 phase.
- Erlang distribution - 2 or more identical phases in sequence.
- Deterministic distribution (or constant) - The limiting case of an Erlang distribution, as the number of phases become infinite, while the time in each state becomes zero.
- Coxian distribution - 2 or more (not necessarily identical) phases in sequence, with a probability of transitioning to the terminating/absorbing state after each phase.
- Hyperexponential distribution - 2 or more non-identical phases, that each has a probability of occurring in a mutually exclusive, or parallel, manner. (Note: The exponential distribution is the degenerate situation when all the parallel phases are identical.)

In the discrete time case we have the following definitions and theorems [Neuts 81] [Mocanu 06]: Assume that for $m \geq 1$ the matrix $\boldsymbol{P}$ is the transition matrix of a $m+1$ state discrete Markov Chain with one absorbing state (From this point on we illustrate matrices and vectors by bold face letters.). Hence, arranging the matrix to have the $m+1$ th state as the absorbing one, we will have

$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{T} & \boldsymbol{T}_0 \\ \boldsymbol{0} & 1 \end{bmatrix}$, and since $\boldsymbol{T}$ is a stochastic matrix we will have the following condition:

$\boldsymbol{T}\ \boldsymbol{1} + \boldsymbol{T}_0 = \boldsymbol{1}$, where **1** is a column vector of ones. Hence we can have:

$$\boldsymbol{T}_0 = (\boldsymbol{I} - \boldsymbol{T})\boldsymbol{1} \qquad (3\text{-}22)$$

The time to absorption of a discrete Markov Chain represented by the matrix $\boldsymbol{P}$ is denoted by $PH(\boldsymbol{\alpha}, \boldsymbol{T})$, where $\boldsymbol{\alpha}$ is the initialization probability vector. The probability density function of this random variable is equal to:

$$f(k) = \boldsymbol{\alpha T}^{k\text{-}1}\boldsymbol{T_0} \text{ where } k \succ 0 \qquad (3\text{-}23)$$

and we also have

$$\begin{aligned} F(X \leq k) &= \sum_{i=1}^{k} f(i) = \sum_{i=1}^{k} \boldsymbol{\alpha T}^{i\text{-}1}\boldsymbol{T_0} = \boldsymbol{\alpha}\sum_{i=1}^{k} \boldsymbol{T}^{i\text{-}1}\boldsymbol{T_0} = \boldsymbol{\alpha}(\boldsymbol{I} - \boldsymbol{T}^k)(\boldsymbol{I} - \boldsymbol{T})^{-1}(\boldsymbol{I} - \boldsymbol{T})\boldsymbol{1} = \boldsymbol{\alpha}(\boldsymbol{I} - \boldsymbol{T}^k)\boldsymbol{1} \quad (3\text{-}24) \\ &= \boldsymbol{\alpha 1} - \boldsymbol{\alpha T}^k\boldsymbol{1} = 1 - \boldsymbol{\alpha T}^k\boldsymbol{1} \end{aligned}$$

3.4.3 Approximate Control Synthesis Methodology

As we mentioned earlier the objective of the agent is to use its basic functions (actions) to avoid a bad event or rather a bad state in the environment. As we can see in figure 3-11, the agent starts with an initial perception model of the environment, but incrementally expands its knowledge about the environment during control synthesis. The eventual goal of the agent is to learn more about the environment as much as possible and to control the process in a way to achieve the desired goal.

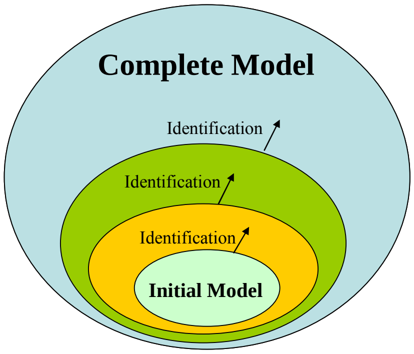

Figure 3-11: Evolution of the perception model

The basic approach is as follows: (1) Identify the uncontrolled perception model of the environment based on the collected samples. (2) Search for all the possible direct paths (cycles are excluded) from the initial state to the bad state by creating the "forward-path structure," which will be explained more in detail later in the chapter. (3) To deactivate (disable) those paths to the bad state by searching through the basic functions and selecting and applying the appropriate ones. This will be done by a search process that considers nodes of the forward-path structure from bad state to the initial state (backward-search) level by level.

It is known that in a "maximally permissive" context the agent must first disable the events that are in the direct neighborhood of the bad state. If that is not possible through the direct neighbors then the second closest neighbors (nodes) will be examined and so on. Hence in this methodology the agent starts the search from a level that is the closest to the bad state. The agent then applies its basic functions in every node in the level and identifies the newly generated finite automata. Every finite automaton outputs a reward to the agent which is a function of the probability of the time to absorption. If any of the received rewards is satisfying the objective (probability is less than a certain threshold) then the corresponding finite automaton is the solution. If the received rewards in that level do not satisfy the objective but at least one of the rewards is greater than the latest reward from the previous level then the new finite automaton with the best action-node combination will be selected. The selected finite automaton will be the base finite automaton for the next level and the agent moves to the next level in its search.

In order to proceed we first need to slightly change the definition of the DSFA, in order to be able to embed the actions of the agent in the mode. Initially we defined DSFA as $A = (\mathcal{E}, Q, P, q_1)$ consisting of an alphabet $\mathcal{E}$, a finite set of nodes Q with q_1 the initial node and a set of probabilities $p_{ij}(e)$ of transitioning from node q_i to node q_j led by symbol e in the alphabet. With transitions affected by the basic functions of the agent, we redefine the elements of P as $p_{ija}(e)$, where $p_{ija}(e)$ is giving the probability of a transition from q_i to node q_j led by symbol e when action a is applied in node q_i. Figure 3-12 illustrates this concept.

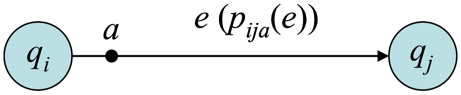

Figure 3-12: Basic function *a* applied in node q_i

Agent's goal is to avoid a state q_x with maximum likelihood. To do so, we first generate a node-arc structure, called "forward-path structure." This tree basically indicates all the paths from the initial state to the bad state by eliminating all the cycles.

Our approach here is similar to what has been commonly used to solve the state avoidance problem in supervisory control literature. Starting from a designated bad state, the immediate states with direct access to the bad state are examined first. If the events or transitions between theses states and the bad state are controllable, then the problem is solved by disabling the respective events. If the event leading to the bad state from a neighboring state is not controllable, then this state is designated as a pre-bad state, and its neighboring states are examined. The process continues until all paths leading to the bad state are disabled, or no controller is found. We basically take a similar approach except for the fact that there the underlying models were assumed to be deterministic, and a complete model of the underlying process or plant model existed. For us, perception model is non-deterministic and that the complete model of the environment is unknown. Having said this, we will also travel backward in state space, but as soon as a control action by the agent satisfies our avoidance criterion, we will stop. For this approximation to work properly, we must assume that, except possibly for the bad state, all the other states are accessible from each other. We will return to this discussion later on, after we present all the necessary formulations. For now we will present the algorithm for creating the forward path structure:

Algorithm: *forward* path search

Input: Finite Automaton $A = (\mathcal{E}, Q, P, q_1)$ where elements of P are defined as $p_{ija}(e)$

Output: Node *n*

```
procedure forward_path_structure (fa: FiniteAutomaton)
var
    m, n, tmp: Node
    q: Queue
begin
    n = $q_1$
    q.enqueue(n)
    while q is nonempty
        Node m = q.dequeue()
        for each tmp in neighbors[m]
                if not qualified(m, tmp)
                        continue
                end if
                tmp.data = c.data
                tmp.parent = m
                m.children.add(tmp)
                q.enqueue(tmp)
        end for
    end while

    return n
end
```

Algorithm: *qualified*

Input: Nodes m and tmp

Output: Boolean

Comment: This algorithm examines a temporary node as a leaf in the current forward path structure and decides whether it is qualified to be added to the structure or not.

```
procedure qualified (m: Node, tmp: Node)
var
    m, tmp: Node
begin
    if m.data = tmp.data
        return false
    end if
    while not (m.parent is nothing)
        if m.parent.data = tmp.data
                return false
        end if
    end while

    return true
end
```

The example in figure 3-13 clarifies the method of creating the forward-path structure. The root of the tree is the initial state (level 0). The next level of the tree (level 1) includes all the neighbor nodes of the initial state, namely q_2 and q_4. Both nodes are qualified to be added to the structure. The second level of the tree (level 2) includes all the direct neighbors of q_2 and q_4. The neighbors of q_2 are q_1, q_3 and q_x. Since q_1 is the initial state therefore we dismiss it (cycle is eliminated). Node q_3 had not been seen before therefore we add it to the forward path structure. q_x is the bad state and therefore is being considered as the leaf of the tree. On the other hand q_4 has one neighbor namely q_x. It will be considered as another leaf of the tree. According to figure 3-13 there are 3 paths from starting to the terminating state without having any cycles. They are ($q_1 \rightarrow q_2 \rightarrow q_x$), ($q_1 \rightarrow q_4 \rightarrow q_x$) and ($q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_x$).

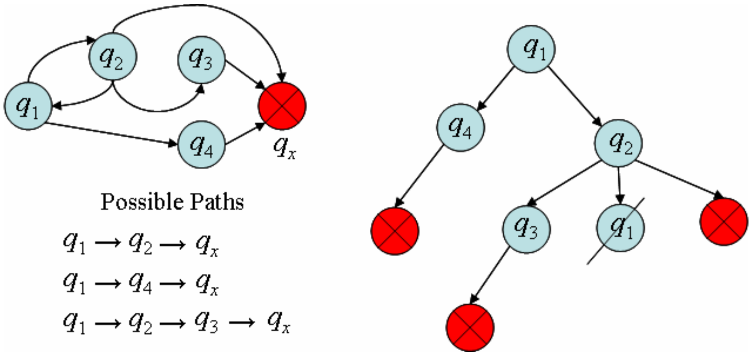
q_1
q_2
q_3
q_4
q_x
Possible Paths
$q_1 \rightarrow q_2 \rightarrow q_x$
$q_1 \rightarrow q_4 \rightarrow q_x$
$q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_x$
q_1
q_4
q_2
q_3
q_1

Figure 3-13: A finite automaton and its corresponding forward-path structure

The last step is to arrange all the batches using the output of the above algorithm. We define a batch of nodes *i* as the set of nodes which are *i* nodes apart from the terminating node, inclusive. We also make note of those nodes that have been appeared in higher level batches at least once. This is because that an agent cannot apply two or more different actions at one single node. Continuing with the example of figure 3-12, we will have the batches depicted in figure 3-15.

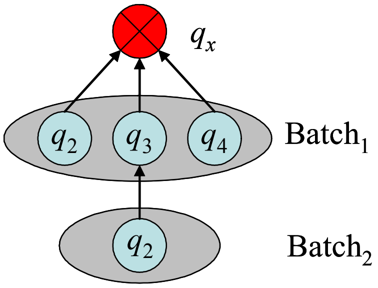

Figure 3-15: Restructuring the forward-path structure

Now that we have found all the paths from the starting state leading to the bad state(s), the objective of the agent will be to reduce the probability of entering to the bad state to a given threshold through all of these paths. Notice that these paths are set of nodes that start from any leaf of this structure and end up into the bad state.

3.4.3.1 Reward Criterion and Control Policy

We assume that agent's clock (measured in discrete time ticks) is independent of the environment's clock and we also assume that the agent must interact with its environment for *k* ticks given the initial state (*k* is assumed to be known to the agent). We consider environment's perception model as a homogeneous Markov chain with a single absorbing state. *The objective is to reduce the probability of absorption within time period* $\leq k$ *, to below (* $0 \leq \varepsilon \leq 1$). From equation (3-24) we know that $P(X \leq k) = 1 - \boldsymbol{\alpha T}^k \mathbf{1}$ where X is the random variable representing the time to absorption. *We want to make sure that regardless of what state the agent finds its environment in, and as long as their* *interaction takes $\le k$ time ticks, the overall probability of reaching the bad state is less than a pre-specified threshold.* That is, we have for $\Im$,

$$D_{FA} = P(X \le k) = 1 - \boldsymbol{\alpha T}^{k}\mathbf{1} \le \varepsilon \qquad (3\text{-}25)$$

The agent starts from batch one of the forward-path structure (there are t nodes). We denote the set of nodes in batch j by

$$an_j = \left\{ q_{s_i} \middle| q_{s_i} \in batch_j \text{ and (if } q_{s_i} \in batch_t \text{ then } f \text{ applied at } q_{s_i} \text{ is } no\_action \text{ for } 1 \le t < j) \right\} (3\text{-}26)$$

Starting from any node in an_j, the agent applies each of its m basic functions to the first node (if the preconditions of the functions are met and the function is available) followed by the identification of the new controlled model of the environment. As such, at most m new finite automaton are created. With the first finite automaton satisfying (3-25), the alghorithm stops, and a solution is obtained. If no such solution exists, but there are D_{FA}s with positive rewards, the one with the highest reward is selected. If none of the D_{FA}s creates a positive reward, the node is ignored and the next node in the same level is examined. When all the nodes in an_j are examined (t nodes), then the algorithm moves to the next j, investigating the finite automaton with the highest reward in the previous batch. The same process is applied to the nodes in batch j + 1 until the solution is found or some termination criterion is reached. The criterion for choosing a finite automaton can be set according to the rules of reinforcement learning. We can choose the one that gives the agent the highest reward. Or to conduct "exploration" we can decide to choose a path which does not necessarily give the highest reward.

The reward function is dependent on D_{FA} defined in (3-25) where FA indicates the corresponding finite automaton. The action-node pair that reduces the value of D_{FA} the most will get the highest reward and can therefore be selected for the next step. This is not necessarily true if the agent decides in the next step to explore rather than to exploit. Equation (3-27) can then be used for each node for a given level of the forward-path structure to select the control policy with the highest immediate reward:

$$policy_{FA,q_m} = \arg\max_{f \in F}\left(D_{FA} - D_{FA_{f,q_m}}\right) \rightarrow FA_{new} \qquad (3\text{-}27)$$

In equation (3-27) $policy_{FA,q_m}$ is the policy selected for node q_m in current finite automaton *FA*, *F* is the set of basic functions, FA_{f,q_m} is a new finite automaton when basic function *f* is applied in q_m and FA_{new} is the newly generated finite automaton by selecting the desired policy. In this equation $D_{FA} - D_{FA_{f,q_m}}$ is the *reward*. Equation (3-27) shows that with $policy_{FA,q_m}$ the agent moves towards a direction which brings it closer to the desired goal.

When the agent moves to the next batch, it must also carry all the information about the new model to the next level in order to make correct decisions. Figure 3-16 shows the structure that will be used for the storage of the finite automata during the search in order to achieve the mentioned goal. Each level of this linked list structure corresponds to a batch. And in each level the linked list includes at most *n* elements if we assume that *n* is the number of nodes in that batch. Each best node-basic function combination generates a new FA and therefore a new element in this structure. The last best FA in each batch will be moved to the next level as the starting FA. Finite automata are indicated as FA_j^i, where *i* is the level and *j* is the index of the corresponding node in that level (batch). $(q_m, f)_m^i$ on top of each element in the linked list shows that node q_m has been effected with the basic function *f* in level *i*.

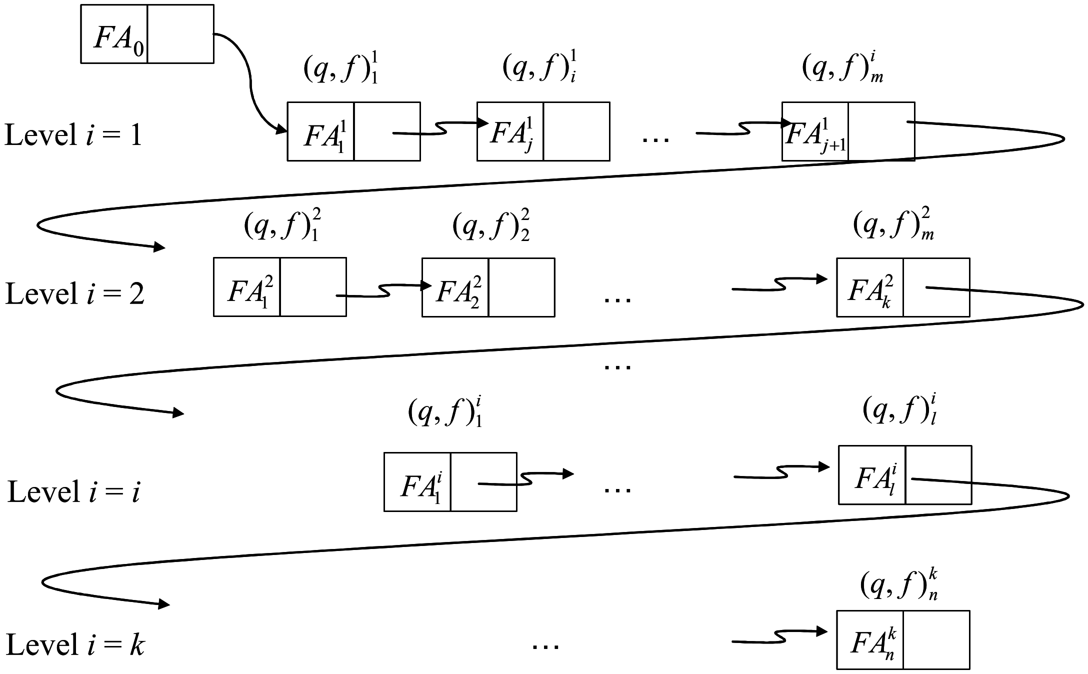

Figure 3-16: Finite Automata Linked List Structure

3.4.3.2 Special Cases and Drawbacks

I. If a node can make two or more transitions to the upper level in the forward-path structure, then only one common basic function can be applied to that node. Otherwise there will be a conflict between two actions in that node.

II. If a new FA generates a new node (state) that has a smaller distance to the bad state than the current active node (which is being evaluated in the current level), then we will ignore this option. The reason is that the upper levels in the tree have already been checked in the algorithm and we cannot go back to the previous levels in the search.

III. We conjecture that the algorithm presented above will work better for some topologies of the DSFA and worse for some others. Especially, we believe that for strongly connected DSFAs the algorithm will terminate faster and more accurately than others.

3.4.3.3 Algorithms

Algorithm: *Control Policy*

Input: FiniteAutomaton FA_0, Integer k

Output: (FiniteAutomaton FA, LinkdedList faLinkedListStructure)

Comment: The uncontrolled finite automaton will be identified by the agent and will be indicated by FA_0 and inputted to this algorithm. It will be the first element of the output finite automaton linked list structure. The forward-path structure will be built for FA_0. Set an_i will be generated based on the forward-path structure. For all nodes in an_i, the agent applies all m possible basic functions $F = \{f_1, f_2, ..., f_m\}$ to each node and creates new FA for each node-basic function combination. It is obvious that the preconditions and other rules $R = \{r_1, r_2, ..., r_s\}$ regarding basic function f_j, must hold for the agent in order to be able to apply f_j at any node. Also two different basic functions cannot be applied to the same node at once. These finite automata will be stored in a list called Candidate_FA. Out of the list in the candidate FAs, one with the highest reward (exploitation) or any other one based on the exploration criteria will be chosen and added to the FA linked list structure. If the reward is zero or negative then the agent moves to the next batch. The selected candidate will be the base for the next level calculations (active FA). This procedure will continue for each and every node in each level until a solution is found or until all nodes are examined and the agent cannot find any solution.

```
procedure control_policy ($FA_0$: FiniteAutomaton, k: Integer)

var
    candidate_FA, batch: List
    activeFPS, $q_n$: Node
    f: BasicFunction
    $D_{activeFA}$, $D_{tempFA}$: Double
    F: BasicFunctionsList
    faLinkedListStructure: LinkedList
    activeFA , tempFA: FiniteAutomaton
begin
    activeFA = $FA_0$
    faLinkedListStructure.add($FA_0$)
    if $D_{activeFA} \leq \varepsilon$
        return (activeFA, faLinkedListStructure)
    end if
    activeFPS = forward_path_structure(activeFA)
    do forever
        for each $batch_j$ in activeFPS
            create $an_j$
            for each node $q_n$ in $an_j$
                for each $f$ in $F$ select $f$ /*selecting can be based on strategic experimentation to add the exploration factors to this algorithm*/
                    if $f$ can be executed based on the rules and if there is no other function already applied in $q_n$
                        candidate_FA.add(identify($FA$, $q_n$, $f$))
                    end if
                end for
                tempFA = FAnew based on equation (3-24) /*or any other exploration criterion
                if FA_qualified(activeFA, tempFA, batch, j)
                    if $D_{tempFA} \leq \varepsilon$
                        faLinkedListStructure.add(tempFA)
                        return (tempFA, faLinkedListStructure)
                    end if
                    activeFA = tempFA
                    activeFPS = forward_path_structure(activeFA)
                    candidate_FA.clear
                    update $an_j$
                end if
            end for
        end for
        if maxCost has been reached then return (Nothing, faLinkedListStructure)  /*the agent has a upper bound cost that it does not want to surpass
    end do
end
```

Algorithm: *Finite Automaton Qualification*

Input: FiniteAutomaton FA_0, List k

Output: Boolean

Comment: This algorithm examines a temporary finite automaton in the Control Policy algorithm, compares it to the current active finite automaton and decides whether it is qualified to be added to the finite automata linked list structure.

```
procedure FA_qualified (activeFPS: Node, $FA_{temp}$: FiniteAutomaton, batch: List, j: Integer)

var
    tempFPS: Node
    i: Integer
begin
    tempFPS = forward_path_structure($FA_{temp}$)
    for each $batch_i$ in tempFPS and each $batch_j$ in activeFPS where i < j
        if there is a new node in $batch_i$ that is not in $batch_j$ then return false
        end for
    end for
    return true
end
```

3.4.4 Example III

Let us consider the chemical process of section 3.3.3 again to show this approach in more details. As you can see in figure 3-17, we have a furnace that is filled with air. There is a heater that turns on and off by some means that are unobservable and unknown to the agent. Its basic functions include FAN_ON, FAN_OFF and NO_ACTION. FAN_ON function turns on the fan in the furnace to reduce the temperature, FAN_OFF turns the fan off. The effects of these basic functions are "unknown" to the agent. Each function has its own *preconditions*, *postactions* and *poststate* (see chapter two). The current state of the agent is: fan = off. The objective of the agent is to interact with the environment (the furnace) for $k = 10$ clock ticks and keep the probability of transitioning to the explosion state less than $\varepsilon = 0.7$. The set of events that the agent can sense from the environment are: λ, T↑, T↓, P↑, P↓ and Explosion. λ indicates an empty string. T↑, T↓, P↑, P↓ are events showing that the temperature and pressure have passed certain thresholds. Event "Explosion" shows that the both temperature and pressure have passed certain threshold and the chamber has exploded. The agent can sense these events using special sensors (switches). We assume that the environment always starts from the same starting state.

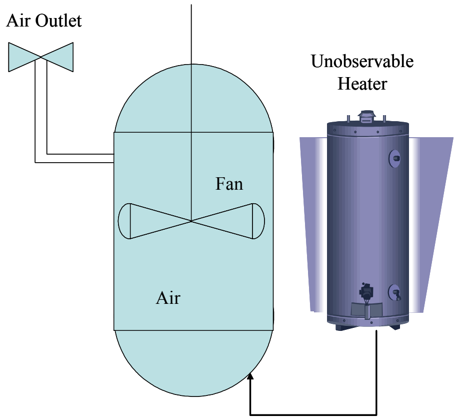

Figure 3-17: Chemical Process

Solution: Using the method described in section 3.3.3, the agent identifies the model shown in figure 3-9. It then triggers the Control Policy algorithm taking the identified model as its input. The first step is to test the uncontrolled finite automaton (FA_0) for its time to absorption probability. If is satisfy the required threshold then there will be no need for control policy search. Based on the structure of the model (Figure 3-9) we have:

$$\boldsymbol{P}_{\boldsymbol{FA_0}} = \begin{bmatrix} .27 & .73 & 0 & 0 \\ .17 & .3 & .52 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

, and we know that $\boldsymbol{\alpha} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ and $\mathbf{1} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}'$. Having k = 10, we will have

$$D_{FA_0} = P(X \le k) = 1 - \boldsymbol{\alpha T}^k \mathbf{1} = 1 - \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} .27 & .73 & 0 \\ .17 & .3 & .52 \\ 0 & 0 & 0 \end{bmatrix}^{10} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0.9738,$$

which is not less than $\varepsilon = 0.7$. Hence the algorithm continues.

Figure 3-18 illustrates Forward_path_structure and the batches. In the first batch we have only S_*. There is only one function available for the agent to execute, namely FAN_ON.

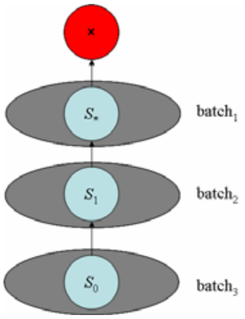

Figure 3-18: Batches of FA_0

After applying basic function FAN_ON in S_* the agent identifies the new generated FA. The sample set obtained is shown in figure 3-19. The output model is depicted in figure 3-20. As you can see, there is one new transition edge added from S_* to S_1, which results in lesser probability of absorption while $k = 10$. In other words the perception model of the environment has been changed by manipulating system dynamics using external means.

λ	10	T↑P↑P↓	2
T↑	10	T↑P↑P↓T↓	1
T↑T↓	2	T↑T↓T↑P↑P↓	1
T↑T↓T↑	2	T↑T↓T↑P↑ex	1
T↑P↑ex	14		

Figure 3-19: Sample set S when applying fan_on in S_*

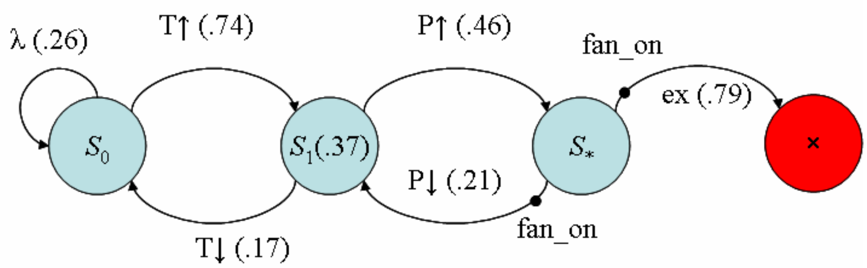

Figure 3-20: Identified model FA_1^1

This new model has the same number of states and the same forward path structure. The new transition creates only a cycle which is eliminated in the forward path structure. The agent will now calculate the time to absorption for the new model. Based on the structure of the model shown in figure 3-20 we have: $\boldsymbol{P}_{FA_1^I} = \begin{bmatrix} .26 & .74 & 0 & 0 \\ .17 & .37 & .46 & 0 \\ 0 & .21 & 0 & .79 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and

hence $D_{FA_1^1} = P(X \le k) = 1 - \boldsymbol{\alpha T}^k \mathbf{1} = 1 - \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} .26 & .74 & 0 \\ .17 & .37 & .46 \\ 0 & .21 & 0 \end{bmatrix}^{10} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0.8977$.

As we can see, the probability of time to absorption has been reduced to 0.8977 but it is still not enough to stop the search. The agent adds model of figure 3-20 to its finite automata linked list structure and moves to the next step. We recall that by applying FAN_ON in S_*, the agent can now use only FAN_OFF as its basic function. The next batch to look at is batch$_2$ which includes node S_1. By applying FAN_OFF in S_1, the agent will increase the probability of time to absorption therefore the environment's reward will be negative and therefore it will be rejected. Hence the agent will move to the last batch. The same scenario repeats since the probability increases again. At this point, depending on the computational costs, the agent may stop the algorithm without finding any control solution or continue the search by going back to FA_0.

By visiting the first batch again the agent comes back to the S_* state and now we suppose that in this iteration the agent will choose NO_ACTION instead of FAN_ON, even though FAN_ON has a higher reward as opposed to reward zero, associated with

NO_ACTION. By doing so, the agent will move to the next batch and this time it will apply FAN_ON in S_1.

λ	9	T↑P↑P↓	1
T↑	1	T↑P↑P↓T↓	2
T↑T↓	8	T↑T↓T↑T↓	3
T↑T↓T↑	2	T↑T↓T↑P↑ex	1
T↑P↑ex	2		

Figure 3-21: Sample set *S* when applying FAN_ON in *S*$_1$

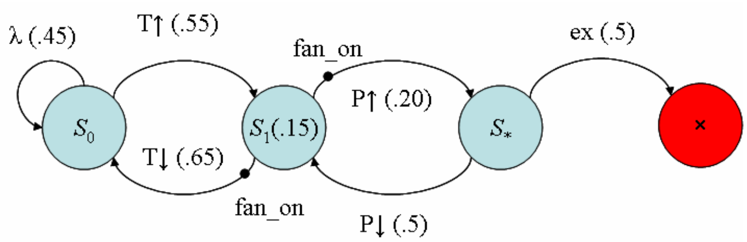

Figure 3-22: Identified model FA_1^2

Based on the structure of the model shown in figure 3-22 we have:

$$\boldsymbol{P} = \begin{bmatrix} .45 & .55 & 0 & 0 \\ .65 & .15 & .20 & 0 \\ 0 & .5 & 0 & .5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and hence $D_{FA_1^1} = P(X \le 10) = 1 - \boldsymbol{\alpha T}^k \mathbf{1} = 1 - \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} .45 & .55 & 0 \\ .65 & .15 & .20 \\ 0 & .5 & 0 \end{bmatrix}^{10} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0.2964$.

As you can see the probability of time to absorption has been drastically reduced and therefore the agent has found a control solution.

3.4.5 Example IV

Let us consider the chemical process again but this time we assume that the agent has access to a valve which it can open or close to reduce the pressure. Its additional basic functions now include VALVE_OPEN to open the valve to reduce the pressure and VALVE_CLOSE to close the valve. The current state of the agent is: fan = off and valve = close. The objective of the agent is to interact with the environment (the furnace) for *k* = 10 clock ticks and to keep probability of absorption within a period $\leq k$, to less than ε = 0.7.

Solution: Using the same approach described previously, the agent identifies the model shown in figure 3-9. We know that the structure of figure 3-9 does not satisfy the control requirements. Having triggered the search algorithm, our agent applies two of its basic function in node S* of the first batch. Suppose that by applying FAN_ON in S_* the structure in figure 3-20 is identified, with the probability of time to absorption equal to 0.8977. By applying VALVE_OPEN in S_* the agent obtains a better probability of time to absorption, i.e. 0.8774, with the structure shown in figure 3-23. Therefore this structure will be added to the finite automata linked list structure.

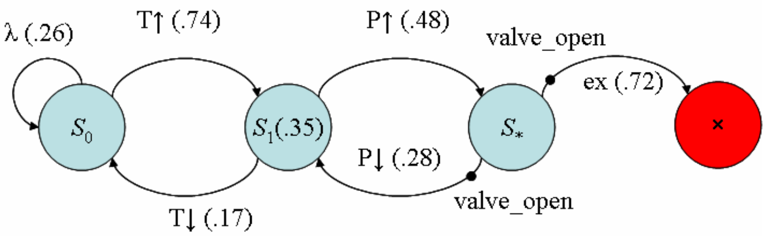

Figure 3-23: Identified model when applying VALVE_OPEN in S_*

Now the agent moves to the next batch. In this batch we have node S_1. The available functions will be VALVE_CLOSE and FAN_ON. VALVE_CLOSE will increase the probability of time to absorption; therefore it cannot be an option. On the other hand, applying FAN_ON in S_1 will generate the sample set shown if figure 3-24 with the corresponding model shown in figure 3-25. This function thus, changes the perception model of the agent from its environment.

λ	24	T↑P↑ex	5
T↑	5	T↑P↑T↓P↓	3
T↑T↓	8	T↑T↓T↑T↓	1
T↑T↓T↑	2	T↑T↓T↑P↑ex	1
T↑P↑T↓	5	T↑P↑T↓T↑ex	1
T↑P↑T↓P↓T↑T↓	1		

Figure 3-24: Sample set *S* when applying FAN_ON in S_1

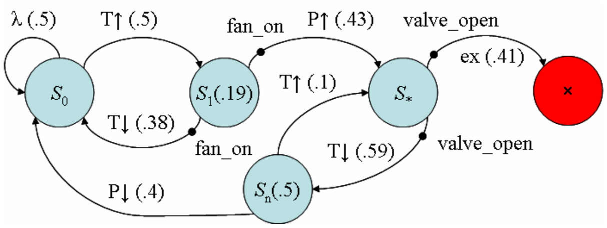

Figure 3-25: Identified model when applying FAN_ON in S_1

As we can see in figure 3-25, the new model has an additional state and three new transitions. This shows that the agent can learn a more complete model by applying some new basic functions. By calculating the new time to absorption for the new model, we obtain:

$$\boldsymbol{P} = \begin{bmatrix} .5 & .5 & 0 & 0 & 0 \\ .38 & .19 & .43 & 0 & 0 \\ 0 & 0 & 0 & .59 & .41 \\ .4 & 0 & .1 & .5 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \text{and}$$

$$D_{FA_1^2} = P(X \leq 10) = 1 - \boldsymbol{\alpha T}^k \mathbf{1} = 1 - \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} .5 & .5 & 0 & 0 \\ .38 & .19 & .43 & 0 \\ 0 & 0 & 0 & .59 \\ .4 & 0 & .1 & .5 \end{bmatrix}^{10} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = 0.4106 \leq \varepsilon\,,$$

which shows that the combination of two basic functions can also be used to reduce the probability of absorption within a period $\leq k$.

3.4.7 Complexity

What was presented above is a machine learning problem. In an ideal case the agent must visit all possible states indefinitely many times to have the best identification results. To make an exact calculation we can consider the following:

The search complexity is dependent on the number of states and number of available basic functions. Suppose that the number of states in the complete model of the environment is n and that the number of available basic functions is m. In the worst case the agent must check m^n different combinations, which is exponentially complex in time.

In a complete search, every node in each level of the forward-path tree must be examined. The order in which the basic functions are applied is important. If basic function f_1 is applied to node A first and then basic function f_2 is applied to node B, the resulting *FA* might have a different structure if the order was changed. If we consider n nodes in each level (worst case) of the forward-path structure that can have k levels and if we also consider m basic functions for the agent, then there will be $m^n \times n! \times k$ different combinations.

Despite the worst case non-polynomial complexity of our approach, we believe that the computational efforts can be drastically reduced by introducing cost factors. We recall that our agent is seeking a solution here only for the purpose of estimating a cost for various task plans that it must make for its own purposes. Therefore, if the estimated cost of obtaining a solution here (which includes the cost of learning amongst other things) exceeds some threshold that the agent calculates independently, such a solution will not be beneficial any more. We will discuss this more rigorously in section 3.4.8 and as well as in chapter 5 of this dissertation.

3.4.8 Improving the Search Algorithm

I) As you can see the algorithm *Control Policy* does not specify how basic functions are chosen by the agent. As mentioned before one obvious strategy would be for the agent in state S to select the basic function that maximizes the immediate reward, there by exploiting its current model approximation. However, with this strategy the agent runs the risk that it will overcommit to combination of functions that are found during the early training, while failing to explore other combination of functions that have even higher rewards. For this reason, like in classical Q learning algorithm, it is very common to use a probabilistic approach to selecting functions. Actions with higher rewards are assigned higher probabilities, but every "qualified" action is assigned a non-zero probability.

II) Even though a solution is found for the first time, the agent should not rely on the solution forever. It should continue to search for other solutions in the next instances, because of the probabilistic nature of the problem. If the agent comes always to the same conclusion then at some point with a certain confidence interval it can deduce that the current solution is the best possible solution.

III) During iterations the agent does not have to start always from FA_0. After the first iteration when the solution is not found the agent can start the main loop in any of the levels of the finite automata linked list structure (Figure 3-16).

IV) Cost: in order to reduce the total cost of the computation we can assign the following cost factors to the search steps:

a) cost of each basic function (c_f): Some of the basic functions can have a negative cost and some of them can have a positive cost (for instance FAN_OFF can save energy for the agent, and therefore can be assigned a negative cost.).

b) cost of the time that a basic function (c_t) is being used during the control process: This can be easily quantified by measuring the distance of the active node from the bad state.

c) cost of learning (c_l): The number of iterations that an agent goes through the training iterations.

Hence:

$$C_{total} = c_f + c_t + c_l \qquad (3\text{-}28)$$

Obviously, these costs must be normalized to the same scale. The sum of all these costs can be deducted from the reward received from the environment:

$$D_{FA} = t \times P(X \leq k) - l \times C_{total} \qquad (3\text{-}29)$$

In equation (3-26) *t* and *l* are normalizing constants.

3.4.9 The Problem of Reaching a Favorable State

Here the agent strives to guide the environment towards a state where a certain event has a higher chance of occurrence. As soon as the desired event is located in the environment's perception model, the problem becomes the problem of reaching a favorable state probabilistically.

There are two cases to consider depending on if the desired event occurs somewhere in FA_0. We present only a solution for the first case where the desired event can be located directly in the initial model. The methodology for solving this kind of problems is quite similar to the problem of avoiding a bad state, except that the criterion for entering to the desired state changes from (3-25) to (3-29).

$$D_{FA} = P(X \leq k) = 1 - \boldsymbol{\alpha T}^k \mathbf{1} \geq \varepsilon \qquad (3\text{-}30)$$

The forward-path structure will be constructed in the same way and a similar methodology as in the case of avoiding a bad state will be applied to this problem for satisfying inequality (3-27).

3.4.10 Conclusion

In this chapter we introduced an algorithm that utilizes the ideas from reinforcement learning to exert control on an unknown environment. This algorithm identifies a perception model of the environment at each level using regular language theory and adapts its search to the newly identified models. The search algorithm is goal oriented. The agent examines its basic functions around the bad state to save the maximally permissive property of the process and to reduce the computational cost. In this method we showed that even a process that is continuous in time can be controlled using a discrete event control theoretic framework.

3.5 Implementation Issues

We understand that in real life examples, the issue of observing the environment after each and every action and taking samples for the purpose of building a perception model could be either infeasible or in many instances too costly to work. Therefore, for our paradigm to be applicable for real life examples, we must think of simulations which capture the dynamic behavior of the environment.

Going back to our illustrative example, it is quite reasonable that we have access to a simulation model of the chemical process which also computes the various disturbances of its environment. In such a case, our control synthesis model can be used and learning procedures can be applied using the simulation.

4. Risk Analysis and Collaborative Fault Avoidance

4.1 Introduction

Our main focus in this chapter is on how the agents perfrom risk analysis during the synthesis and how they can overcome a potential faulty state using collaboration concepts. We recall that the environment model is probabilistic and unknown. In chapter three we showed how agents must prevent a fault condition while they are interacting with some environmental components. We showed a control synthesis strategy that our agent can take into affect the state of this environmental component for the purpose of avoiding adverse conditions, or creating some desirable effects. Here, we assume that our agent could reach to a state which in conjunction with other agents could result in a global bad condition. We will assume that the nature of such a global state is known a priori. But, the likelihood of reaching this state must be calculated by the agent at different times. Should such likelihood be too high (compared to some threshold limit) the agent must then ask for collaboration from other agents. In this chapter, we will address both risks calculation and collaboration scheme.

The collaborative measures discussed in here are based on game theory concepts. The first step to avoid the global faulty state uses pure communication. The agents may announce their local state to each other. If this step does not help, they may decide to move to the second level of collaboration where they execute some sort of game theory algorithm to find the best possible solution by changing their schedule. The specifics of scheduling changes are outside of the scope of this dissertation.

4.2 Problem Definition

The risk of reaching a globally defined fault state depends on and is defined by the agent's perception of its own behavior and that of other agents. Agents create their own perception models. They can also create models of their perception from others or they can receive the models from other agents through communication. In this sequel we will assume that any model of the environment is probabilistic and originally unknown by any agent.

Since a fault state is defined by a collection of states of several agents visited during a specific period of time, we need to include "time" into our modeling paradigm. We will assume that agents share a common time origin. Our new paradigm requires us to switch from Markov chains we used in chapter three to continuous time Markov processes.

Let S_F be a *global fault state* and $\mathscr{F}$ define the set of all agents which are involved in this fault state. That is

$$\mathscr{F} = \left\{ (\Phi_i, S_F^i) \middle| Agent\ \Phi_i \text{, at its local state } S_F^i \text{, is involved in the global fault state } S_F \right\} \quad (4\text{-}1)$$

The fault condition associated with S_F is then defined by

$$FC = \left\{ \left(\Phi_1, S_F^1, \left(t_1^1, t_2^1\right)\right), \left(\Phi_2, S_F^2, \left(t_1^2, t_2^2\right)\right), \ldots, \left(\Phi_m, S_F^m, \left(t_1^m, t_2^m\right)\right) \middle| \Phi_1, \Phi_2, \ldots, \Phi_m \in \mathscr{F}, \bigcap_i (t_1^i, t_2^i) \neq \varnothing \right\} \quad (4\text{-}2)$$

where the tuple $\left(\Phi_i, S_F^i, \left(t_1^i, t_2^i\right)\right)$ shows that agent Φ_i is in state S_F^i during time period $\left(t_1^i, t_2^i\right)$. Note that *FC* is a minimal set, that is, removal of one or more agents from it will result in the elimination of the fault condition. If no non-null *FC* set is obtained for any given system, then we say that system experiences no faults.

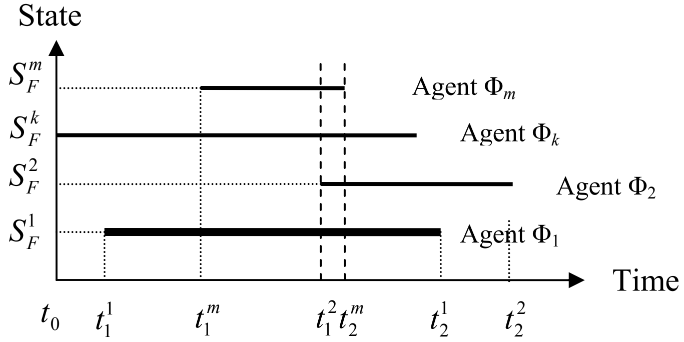

Figure 4-1: Timelines

Figure 4-1 illustrates a fault condition. Clearly $\left(t_1^2, t_2^m\right)$ is the time period where the fault condition holds.

4.2.1 Formulation of Risk

We define the risk of agent Φ_1 entering into global fault state S_F by:

$$Risk_{\Phi_1}\left(S_F^1, (t_1^1, t_2^1)\right) = \prod_{i \in \mathscr{F} - \{\Phi_1, \Phi_m\}} P\begin{pmatrix} \Phi_i \text{ visits state } S_F^i \text{ during } \left(t_1^1, t_2^1\right) \text{and overlaps in time with all the other} \\ \text{agents, members of } \mathscr{F} \end{pmatrix} \quad (4\text{-}3)$$

That is, the risk for agent Φ_1 is the probability that its visits to state S_F^1 during $\left(t_1^1, t_2^1\right)$ overlaps with the overall condition *FC* becoming true.

4.2.2 Assumptions

We make the following assumptions:

I) Any global fault state is known to all agents.

II) Agents are initially modeled as DSFA – The identification methodology was fully described in chapter 3. These models are either given to the agents, or they learn about these models gradually when system evolves. We will treat these models as continuous time Morkov models for the purpose of risk calculations.

III) The agent, who initially initiates its risk calculation, knows its own schedule. In other words, if agent Φ_1 is initiating the risk calculation then the assumption is that the time period $\left(t_1^1, t_2^1\right)$ is known and will be communicated to others.

IV) Agent's action does not affect the perception model of the other agents. In other words the agents are independent of each other.

4.3 Methodology Overview

Agent Φ_1 calculates its risk $Risk_{\Phi_1}\left(S_F^1, (t_1^1, t_2^1)\right)$, using its own model and its perception models of the other agents. We assume that the perception model of other agents is stationary over time period leading to $\left(t_1^1, t_2^1\right)$. Without loss of generality we will assume that risk calculation is initiated by agent Φ_1.

We will present two approaches to calculate the above risk. First we show a shuffling approach that agent Φ_1 uses to compute its risk. By shuffling the perception models of all the agents involved in the fault condition, Φ_1 builds a global continuous time Markov chain. From this model, Φ_1 then estimates the probability of absorption to global bad state during $\left(t_1^1, t_2^1\right)$.

The second approach avoids shuffling; thus avoiding state space explosion problem. While it is only an approximation and is less accurate, it has some advantages over the shuffling approach. We will discuss these later.

4.3.1 Time to Absorption

In order to calculate the risk, an agent needs to know when it will make a transition to its potential local bad state. Like in the previous chapter we will use the time to absorption concept to calculate this parameter. It is known that the probability density function of the time to absorption in a homogeneous, single absorbing state Markov process is given by

$$f(t) = \boldsymbol{\alpha} e^{\boldsymbol{T}t}(-\boldsymbol{T}\mathbf{1}) \qquad (4\text{-}4)$$

where $\boldsymbol{\alpha}$ is the initial probability vector. $\mathbf{1}$ is vector of ones and $\boldsymbol{T}$ is the matrix of the transient states. Using distribution function (4-4) and the assumption of the potential local bad state S_F^1 being a temporary absorbing state, we can calculate the probability of going to S_F^1.

4.3.2 Shuffle

In the shuffle approach we consider the full generator of all the involved Markov processes. Then we construct the shuffled Markov process which shows the full behaviour. The transition matrix of the shuffled model shows the transitions from any composite state to any composite state. The global fault state can now be easy identified using set $\mathscr{F}$. By arranging the global transition matrix to an appropriate shape, we will now be able to calculate the probability distribution of the time to absorption by using equation (4-4) and the cumulative probability using following equation:

$$F(t) = 1 - \boldsymbol{\alpha} e^{\boldsymbol{G}t}\mathbf{1} \qquad (4\text{-}5)$$

where $\boldsymbol{\alpha} = \boldsymbol{\alpha}_{\Phi_1}^1 \otimes \boldsymbol{\alpha}_{\Phi_2}^1 \otimes \boldsymbol{\alpha}_{\Phi_3}^1$ is the shuffled initial probability and $\boldsymbol{G}$ is the shuffled transition matrix. Then the risk in the time interval $\left(t_1^1, t_2^1\right)$ is computed as:

$$Risk_{\Phi_1}\left(S_F^1, (t_1^1, t_2^1)\right) = P(t_1^1 < T < t_2^1) = P(T < t_2^1) - P(T < t_1^1) = F(t_2^1) - F(t_1^1) \qquad (4\text{-}6)$$

4.3.3 Approximation Method

We assume that during $\left(t_1^1, t_2^1\right)$ only one visit to S_F^i is possible by agent Φ_i. Note that an agent Φ_i may enter and exit S_F^i during $\left(t_1^1, t_2^1\right)$ many times, but we assume that the probability of such an event is too small and can be neglected.

We categorize agents by their time when they enter their corresponding fault states S_F^i. Two categories are defined:

- Agent Φ_i is already in S_F^i at t_0. We say $\Phi_i \in \mathscr{F}_0$.
- Agent Φ_i enters into S_F^i after t_0. We say $\Phi_i \notin \mathscr{F}_0$.

These two sets are depicted in figure 4-2.

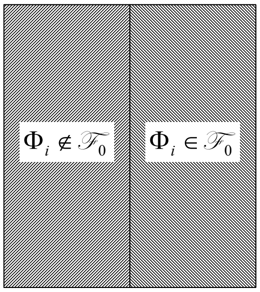

Figure 4-2: Two Categories

Each of these categories can further be subdivided into different subcategories. As shown in figure 4-3, agents that belong to $\Phi_i \in \mathscr{F}_0$ are divided into two groups, namely those that leave their fault states earlier than t_1^1 and those that leave these states after t_1^1. Obviously, agents in the first subcategory cannot contribute to the fault condition. Therefore we only consider the second subcategory, denoted by *B***.** In figure 4-3 T_{end_i} is the time when agent Φ_i leaves its bad state S_F^i.

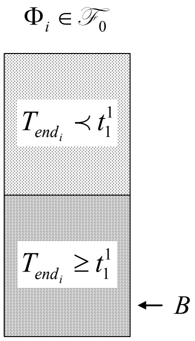

Figure 4-3: Set *B*

Agents that belong to $\Phi_i \notin \mathscr{F}_0$ can be subdivided into two groups depending on when they reach their corresponding fault states (after or before t_2^1), as shown in figure 4-4. Clearly, the ones with this time greater than t_2^1, are not of interest to us here. In figure 4-4 T_{begin_i} indicates the time when the agent makes a transition to state S_F^i.

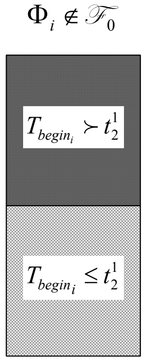

Figure 4-4: Subcategories of $\Phi_i \notin \mathscr{F}_0$

Agents with time to absorption T_{begin_i} less than t_2^1 can further be divided into two subcategories, as it can be seen in figure 4-5. The first category includes those agents that leave their fault state earlier than t_1^1. The second category includes those agents that leave their fault state after t_1^1. The first subcategory will not contribute to any fault, thus will be discarded. The second category on the other hand is of interest to us here and is denoted by A. Figure 4-6 shows all the categories and subcategories.

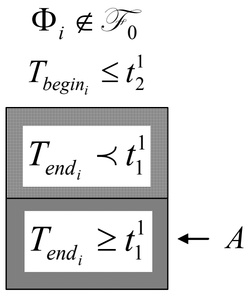

Figure 4-5: Subcategory *A*

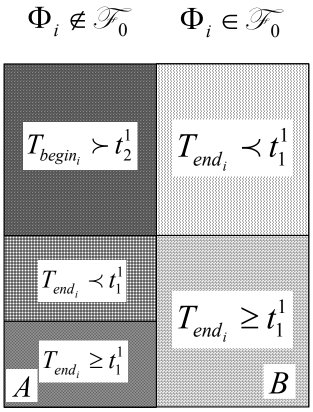

Figure 4-6: Categories and Subcategories

From this point on we may show T_{begin_i} by T_i (time to absorption) and T_{end_i} by $T_i + \tau_i$ (time to absorption plus corresponding Sojourn time). These notations may be used throughout this chapter interchangeably.

Now we discuss how any two agents are positioned with respect to each other in our time line of fault condition. The first case is where $\Phi_i \in A$ and $\Phi_j \in A$, as shown in figure 4-7.

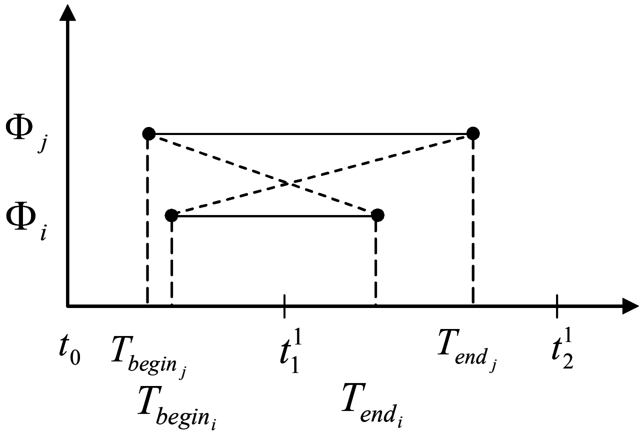

Figure 4-7: $\Phi_i \in A$ **and** $\Phi_j \in A$

This case translates into the following inequalities:

$$\begin{aligned} T_i &\prec T_j + \tau_j \\ T_j &\prec T_i + \tau_i \end{aligned} \qquad (4\text{-}7)$$

The second case is when $\Phi_i \in A$ and $\Phi_j \in B$, as shown in figure 4-8. We notice that in order to guarantee the overlap the time where agent Φ_j comes out of the absorbing state must be greater than the time where agent Φ_i goes out of the absorbing state.

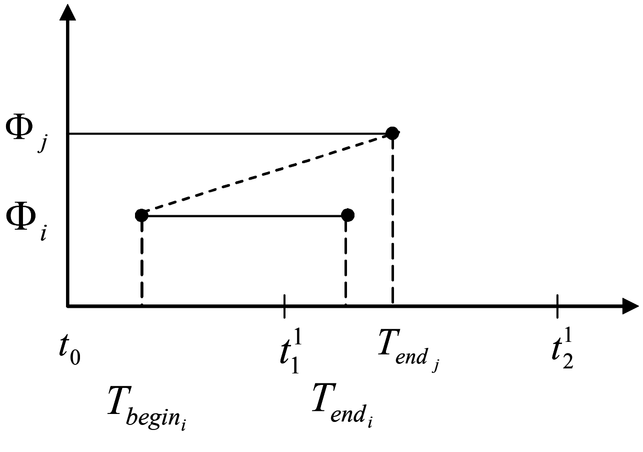

Figure 4-8: $\Phi_i \in A$ **and** $\Phi_j \in B$

This case translates into the following inequality:

$$T_i \prec \tau_j + t_0 \qquad (4\text{-}8)$$

The last case is where $\Phi_i \in B$ and $\Phi_j \in B$, as shown in figure 4-9. We notice that, in order to guarantee the overlap, the times where the agents leave their fault states must be greater than t_1^1. As long as this condition holds there is no mutual constraint between Φ_i and Φ_j. This case translates into the following inequalities:

$$\begin{aligned} \tau_i + t_0 \succ t_1^1 \\ \tau_j + t_0 \succ t_1^1 \end{aligned} \qquad (4\text{-}9)$$

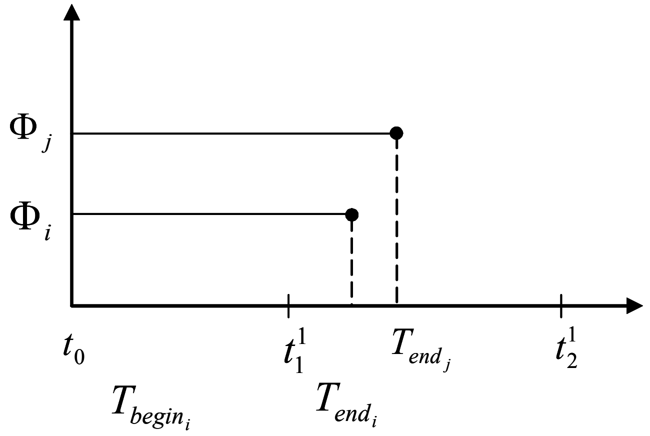

Figure 4-9: $\Phi_i \in B$ **and** $\Phi_j \in B$

Having defined the above categories we can calculate the risk as follows:

$$Risk_{\Phi_1}\left(S_F^1,(t_1^1,t_2^1)\right) = \prod_{\Phi_i \in \mathscr{F} - \{\Phi_1,\Phi_m\}} P\begin{pmatrix} \Phi_i \text{ visits state } S_F^i \text{ during } \left(t_1^1,t_2^1\right) \text{and overlaps in time with all the other} \\ \text{agents member of } \mathscr{F} \end{pmatrix}$$

$$Risk_{\Phi_1}\left(S_F^1,(t_1^1,t_2^1)\right) = \prod_{\Phi_i \in \mathscr{F} - \{\Phi_1,\Phi_m\}} P\left\{\left(\Phi_i, S_F^i, \left(t_1^i,t_2^i\right)\right), \forall \Phi_i \in \mathscr{F} \wedge \Phi_i \neq \Phi_1 \wedge \left(\bigcap_{i \neq 1} (t_1^i,t_2^i)\right) \cap \left(t_1^1,t_2^1\right) \neq \varnothing \right\}$$

$$Risk_{\Phi_1}\left(S_F^1,(t_1^1,t_2^1)\right)=\prod_{\Phi_i\in\mathscr{F}-\{\Phi_1,\Phi_m\}}\left[\begin{array}{l}P(\Phi_i\in A)\cdot P\begin{pmatrix}\text{Sojourn Time such that }\mathit{\Phi}_i\\ \text{guarantees the overlap period}\\ \text{given that }\Phi_i\in A\end{pmatrix}\\ +P(\Phi_i\in B)\cdot P\begin{pmatrix}\text{Sojourn Time such that }\mathit{\Phi}_i\\ \text{guarantees the overlap period}\\ \text{given that }\Phi_i\in B\end{pmatrix}\end{array}\right] \tag{4-10}$$

For now we consider the Sojourn Times exponentially distributed. In case of $\Phi_i \in \mathscr{F}_0$, τ_i is the *Residual Sojourn Time*.

Interpretation of equation (4-10):

For each *i* we are only interested in cases where it is likely to overlap with other agents, and that happens when Φ_i is either in *A* or in *B*. We multiply the probability of being in *A* or in *B* by the probability of having a Sojourn time such that the overlap with other agents are guaranteed, given that the agent is in *A* or is in *B*. Calculating these probabilities for all the agents and multiplying them with each other gives us the risk.

Note: In equation (4-10) the main product is taken over $\Phi_i \in \mathscr{F} - \{\Phi_1, \Phi_m\}$, because it is given that Φ_1 is in state S_F^1 during (t_1^1, t_2^1). Agent Φ_m is not part of this product because agents are lexicographically ordered and therefore every agent will compare itself only with the ones that have a greater index. The last comparison will take place for *i* = *m* -1 between Φ_{m-1} and Φ_m and then the calculation will stop.

For equation (4-10) to work, we need to calculate the following two terms, namely: $P\begin{pmatrix}\text{Sojourn Time such that } \mathit{\Phi}_i \\ \text{guarantees the overlap period} \\ \text{given that } \Phi_i \in A\end{pmatrix}$ and $P\begin{pmatrix}\text{Sojourn Time such that } \mathit{\Phi}_i \\ \text{guarantees the overlap period} \\ \text{given that } \Phi_i \in B\end{pmatrix}$.

We already know that $\Phi_i \in A$. We just need to calculate the overlap period of Φ_i with the other agents. There are actually two different types of agents who can create overlap with agent Φ_i; agents who are in set *A* and agents who are in set *B*. Let Φ_j to be an arbitrary agent, then $\Phi_j \in A$ or $\Phi_j \in B$. Using conditional probabilities, figures 4-7 and 4-8 and inequalities (4-5) and (4-6) we can rewrite the first term, as:

$$\begin{aligned}
&P\begin{pmatrix}\text{Sojourn Time such that } \mathit{\Phi}_i \\ \text{guarantees the overlap period} \\ \text{given that } \Phi_i \in A\end{pmatrix} \\
&= P\left(\underset{j,j\succ i}{\wedge}\left(\left(\Phi_j \in A \wedge \Phi_j \text{ ovelaps with } \Phi_i\right) \vee \left(\Phi_j \in B \wedge \Phi_j \text{ ovelaps with } \Phi_i\right)\right) \middle| \Phi_i \in A\right) \\
&= \prod_{j,j\succ i}\left(P\left(\Phi_j \text{ ovelaps with } \Phi_i \middle| \Phi_j \in A, \Phi_i \in A\right)\cdot P\left(\Phi_j \in A\right) + P\left(\Phi_j \text{ ovelaps with } \Phi_i \middle| \Phi_j \in B, \Phi_i \in A\right)\cdot P\left(\Phi_j \in B\right)\right) \\
&= \prod_{j,j\succ i}\begin{pmatrix} P\left(T_i \prec T_j + \tau_j, T_j \prec T_i + \tau_i \middle| t_0 \le T_i \le t_2^1, t_0 \le T_j \le t_2^1, T_i + \tau_i \ge t_1^1, T_j + \tau_j \ge t_1^1\right)\cdot P\left(\Phi_j \in A\right) \\ + P\left(T_i \prec \tau_j + t_0 \middle| t_0 \le T_i \le t_2^1, T_i + \tau_i \ge t_1^1, \tau_j + t_0 \ge t_1^1\right)\cdot P\left(\Phi_j \in B\right)\end{pmatrix}
\end{aligned} \tag{4-11}$$

Using a similar approach we can rewrite the second term:

$$\begin{aligned}
&P\begin{pmatrix}\text{Sojourn Time such that } \mathit{\Phi}_i \\ \text{guarantees the overlap period} \\ \text{given that } \Phi_i \in B\end{pmatrix} \\
&= P\left(\underset{j,j\succ i}{\wedge}\left(\left(\Phi_j \in A \wedge \Phi_j \text{ ovelaps with } \Phi_i\right) \vee \left(\Phi_j \in B \wedge \Phi_j \text{ ovelaps with } \Phi_i\right)\right) \middle| \Phi_i \in B\right) \\
&= \prod_{j,j\succ i}\left(P\left(\Phi_j \text{ ovelaps with } \Phi_i \middle| \Phi_j \in A, \Phi_i \in B\right)\cdot P\left(\Phi_j \in A\right) + P\left(\Phi_j \text{ ovelaps with } \Phi_i \middle| \Phi_j \in B, \Phi_i \in B\right)\cdot P\left(\Phi_j \in B\right)\right) \\
&= \prod_{j,j\succ i}\left(P\left(\Phi_j \text{ ovelaps with } \Phi_i \middle| \Phi_j \in A, \Phi_i \in B\right)\cdot P\left(\Phi_j \in A\right) + P\left(\Phi_j \in B\right)\right) \\
&= \prod_{j,j\succ i}\left(P\left(T_j \prec \tau_i + t_0 \middle| t_0 \le T_j \le t_2^1, T_j + \tau_j \ge t_1^1, \tau_i + t_0 \ge t_1^1\right)\cdot P\left(\Phi_j \in A\right) + P\left(\Phi_j \in B\right)\right)
\end{aligned} \tag{4-12}$$

Note: $P\left(\Phi_j \text{ ovelaps with } \Phi_i \middle| \Phi_j \in B, \Phi_i \in B\right)$ is always one, as long as both agents are in *B* (See figure 4-9).

The next step is to calculate the different terms in equations (4-11) and (4-12). These terms are:

- $P(\Phi_i \in A)$
- $P(\Phi_i \in B)$
- $P\left(T_i \prec T_j + \tau_j, T_j \prec T_i + \tau_i \middle| t_0 \le T_i \le t_2^1, t_0 \le T_j \le t_2^1, T_i + \tau_i \ge t_1^1, T_j + \tau_j \ge t_1^1\right)$
- $P\left(T_i \prec \tau_j + t_0 \middle| t_0 \le T_i \le t_2^1, T_i + \tau_i \ge t_1^1, \tau_j + t_0 \ge t_1^1\right)$
- $P\left(T_j \prec \tau_i + t_0 \middle| t_0 \le T_j \le t_2^1, T_j + \tau_j \ge t_1^1, \tau_i + t_0 \ge t_1^1\right)$

$\underline{P(\Phi_i \in A)}$

$$
\begin{aligned}
&P(\Phi_i \in A) = P(\Phi_i \notin F_0 \wedge T_i \le t_2^1 \wedge T_{end_i} \ge t_1^1) \\
&= P(T_i \ge t_0 \wedge T_i \le t_2^1 \wedge T_{end_i} \ge t_1^1) = P(t_0 \le T_i \le t_2^1 \wedge T_{end_i} \ge t_1^1) \\
&= P(t_0 \le T_i \le t_2^1 \wedge t_1^1 \le T_i + \tau_i) = P\left(t_1^1 \le T_i + \tau_i \middle| t_0 \le T_i \le t_2^1\right) \cdot P\left(t_0 \le T_i \le t_2^1\right) \\
&= P\left(t_1^1 \le T_i + \tau_i \middle| t_0 \le T_i \le t_2^1\right) \cdot \left(F_{T_i}(t_2^1) - F_{T_i}(t_0)\right) \\
&= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i \middle| T_i = t \wedge t_0 \le t \le t_2^1\right) f_{T_i | t_0 \le T_i \le t_2^1}(t)dt\left(F_{T_i}(t_2^1) - F_{T_i}(t_0)\right) \\
&+ \int_{t_1^1}^{t_2^1} P\left(t_1^1 - t \le \tau_i \middle| T_i = t \wedge t_0 \le t \le t_2^1\right) f_{T_i | t_0 \le T_i \le t_2^1}(t)dt\left(F_{T_i}(t_2^1) - F_{T_i}(t_0)\right) \\
&= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) f_{T_i | t_0 \le T_i \le t_2^1}(t)dt\left(F_{T_i}(t_2^1) - F_{T_i}(t_0)\right) + \int_{t_1^1}^{t_2^1} P\left(t_1^1 - t \le \tau_i\right) f_{T_i | t_0 \le T_i \le t_2^1}(t)dt\left(F_{T_i}(t_2^1) - F_{T_i}(t_0)\right) \\
&= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) f_{T_i | t_0 \le T_i \le t_2^1}(t)dt\left(F_{T_i}(t_2^1) - F_{T_i}(t_0)\right) + \int_{t_1^1}^{t_2^1} f_{T_i | t_0 \le T_i \le t_2^1}(t)dt\left(F_{T_i}(t_2^1) - F_{T_i}(t_0)\right) \\
&= \frac{\int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) f_{T_i}(t)dt}{F_{T_i}(t_2^1) - F_{T_i}(t_0)}\left(F_{T_i}(t_2^1) - F_{T_i}(t_0)\right) + \frac{\int_{t_1^1}^{t_2^1} f_{T_i}(t)dt}{F_{T_i}(t_2^1) - F_{T_i}(t_0)}\left(F_{T_i}(t_2^1) - F_{T_i}(t_0)\right) \qquad (4\text{-}13) \\
&= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) f_{T_i}(t)dt + \int_{t_1^1}^{t_2^1} f_{T_i}(t)dt
\end{aligned}
$$

In summary:

$$P(\Phi_i \in A) = \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) f_{T_i}(t)dt + \int_{t_1^1}^{t_2^1} f_{T_i}(t)dt \qquad (4\text{-}14)$$

Note: In order to calculate $\int_{t_0}^{t_2^1} P\left(t_1^1 - t \prec \tau_i\right) f_{T_i \mid t_0 \le T_i \le t_2^1}(t)dt$ in equation (4-13) we can use equation (4-4) and the fact that: if $B \subseteq S$ with $P(X \in B) = \int_B f(x)dx > 0$, then the *conditional density function* of X given $X \in B$ is $f\left(x \mid X \in B\right) = \dfrac{f(x)}{P(X \in B)}$ for $x \in B$. In this equation X is a continuous random variable taking values in set *S*. Hence:

$$f_{T_i \mid t_0 \le T_i \le t_2^1}(t) = \begin{cases} \dfrac{f_{T_i}(t)}{F_{T_i}(t_2^1) - F_{T_i}(t_0)} & \text{for } t_0 \le t \le t_2^1 \\ 0 & \text{otherwise} \end{cases} \qquad (4\text{-}15)$$

Therefore using equation (4-8) we can write:

$$\int_{t_0}^{t_2^1} P\left(t_1^1 - t \prec \tau_i\right) f_{T_i \mid t_0 \le T_i \le t_2^1}(t)dt = \int_{t_0}^{t_2^1} P\left(t_1^1 - t \prec \tau_i\right) \frac{\boldsymbol{\alpha}_{\Phi_i}^1 e^{\boldsymbol{T}_{\Phi_i}^1 t}(-\boldsymbol{T}_{\Phi_i}^1 \mathbf{1})}{\boldsymbol{\alpha}_{\Phi_i}^1 \left(e^{\boldsymbol{T}_{\Phi_i}^1 t_0} - e^{\boldsymbol{T}_{\Phi_i}^1 t_2^1}\right)\mathbf{1}} dt \qquad (4\text{-}16)$$

We note that in the above equation, all the parameters belonging to agent Φ_i are defined with respect to the perception model that Φ_1 has from Φ_i. In that respect, $\boldsymbol{T}_{\Phi_i}^1$ is the matrix of the transient states, $\boldsymbol{\alpha}_{\Phi_i}^1$ is the initial probability vector and $\lambda_{\Phi_1}^1$ is the exit rate from the fault state S_F^i.

$\underline{P\left(\Phi_i \in B\right)}$

$$\begin{aligned} P(\Phi_i \in B) &= P(\Phi_i \in F_0 \wedge T_{end_i} \ge t_1^1) \\ &= P(T_i \prec t_0 \wedge T_{end_i} \ge t_1^1) = P(T_i \prec t_0 \wedge t_1^1 \le \tau_i + t_0) = P\left(T_i \prec t_0\right) P\left(t_1^1 - t_0 \le \tau_i\right) \\ &= F_{T_i}(t_0) P\left(t_1^1 - t_0 \le \tau_i\right) \end{aligned} \qquad (4\text{-}17)$$

In summary:

$$P(\Phi_i \in B) = F_{T_i}(t_0) P\left(t_1^1 - t_0 \le \tau_i\right) \tag{4-18}$$

$$\underline{P\left(T_i \prec T_j + \tau_j, T_j \prec T_i + \tau_i \middle| t_0 \le T_i \le t_2^1, t_0 \le T_j \le t_2^1, T_i + \tau_i \ge t_1^1, T_j + \tau_j \ge t_1^1\right)}$$

$$\begin{aligned} &P\left(T_i \prec T_j + \tau_j, T_j \prec T_i + \tau_i \middle| t_0 \le T_i \le t_2^1, t_0 \le T_j \le t_2^1, T_i + \tau_i \ge t_1^1, T_j + \tau_j \ge t_1^1\right) \\ &= \frac{P\left(T_i \prec T_j + \tau_j, T_j \prec T_i + \tau_i, t_1^1 \le T_i + \tau_i, t_1^1 \le T_j + \tau_j \middle| t_0 \le T_i \le t_2^1, t_0 \le T_j \le t_2^1\right)}{P\left(t_1^1 \le T_i + \tau_i, t_1^1 \le T_j + \tau_j \middle| t_0 \le T_i \le t_2^1, t_0 \le T_j \le t_2^1\right)} = \frac{A}{B} \end{aligned} \tag{4-19}$$

Below, we calculate terms *A* and *B*. For *A* we have:

$$\begin{aligned} A &= \int_{t_0}^{t_1^1} P\left(t \prec T_j + \tau_j, T_j \prec t + \tau_i, t_1^1 - t \le \tau_i, t_1^1 \le T_j + \tau_j \middle| T_i = t, t_0 \le T_j \le t_2^1\right) f_{T_i|t_0 \le T_i \le t_2^1} dt \\ &+ \int_{t_1^1}^{t_2^1} P\left(t \prec T_j + \tau_j, T_j \prec t + \tau_i, t_1^1 \le T_j + \tau_j \middle| T_i = t, t_0 \le T_j \le t_2^1\right) f_{T_i|t_0 \le T_i \le t_2^1} dt = A_1 + A_2 \end{aligned} \tag{4-20}$$

Now, we calculate A_1 and A_2 separately:

$$\begin{aligned} A_1 &= \int_{t_0}^{t_1^1}\int_{t_0}^{t} P\left(t - t' \prec \tau_j, t' - t \prec \tau_i, t_1^1 - t \le \tau_i, t_1^1 - t' \le \tau_j \middle| T_j = t'\right) f_{T_j|t_0 \le T_j \le t_2^1} dt' f_{T_i|t_0 \le T_i \le t_2^1} dt \\ &+ \int_{t_0}^{t_1^1}\int_{t}^{t_1^1} P\left(t - t' \prec \tau_j, t' - t \prec \tau_i, t_1^1 - t \le \tau_i, t_1^1 - t' \le \tau_j \middle| T_j = t'\right) f_{T_j|t_0 \le T_j \le t_2^1} dt' f_{T_i|t_0 \le T_i \le t_2^1} dt \\ &= \int_{t_0}^{t_1^1}\int_{t_0}^{t} P\left(t - t' \prec \tau_j, t_1^1 - t \le \tau_i, t_1^1 - t' \le \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1} dt' f_{T_i|t_0 \le T_i \le t_2^1} dt \\ &+ \int_{t_0}^{t_1^1}\int_{t}^{t_1^1} P\left(t' - t \prec \tau_i, t_1^1 - t \le \tau_i, t_1^1 - t' \le \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1} dt' f_{T_i|t_0 \le T_i \le t_2^1} dt \\ &= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) \int_{t_0}^{t} P\left(t - t' \prec \tau_j, t_1^1 - t' \le \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1} dt' f_{T_i|t_0 \le T_i \le t_2^1} dt \\ &+ \int_{t_0}^{t_1^1}\int_{t}^{t_1^1} P\left(t_1^1 - t \le \tau_i, t_1^1 - t' \le \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1} dt' f_{T_i|t_0 \le T_i \le t_2^1} dt \\ &= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) \int_{t_0}^{t} P\left(t_1^1 - t' \le \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1} dt' f_{T_i|t_0 \le T_i \le t_2^1} dt \\ &+ \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) \int_{t}^{t_1^1} P\left(t_1^1 - t' \le \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1} dt' f_{T_i|t_0 \le T_i \le t_2^1} dt \\ &= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) \int_{t_0}^{t_1^1} P\left(t_1^1 - t' \le \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1} dt' f_{T_i|t_0 \le T_i \le t_2^1} dt \end{aligned} \tag{4-21}$$

$$
\begin{aligned}
A_2 &= \int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t} P\left(t-t' \prec \tau_j, t'-t \prec \tau_i, t_1^1 - t' \le \tau_j \middle| T_j = t'\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt \\
&+ \int_{t_1^1}^{t_2^1}\int_{t}^{t_2^1} P\left(t-t' \prec \tau_j, t'-t \prec \tau_i, t_1^1 - t' \le \tau_j \middle| T_j = t'\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt \\
&= \int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t} P\left(t-t' \prec \tau_j, t'-t \prec \tau_i\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt \\
&+ \int_{t_1^1}^{t_2^1}\int_{t}^{t_2^1} P\left(t-t' \prec \tau_j, t'-t \prec \tau_i\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt \\
&= \int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t} P\left(t-t' \prec \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt + \int_{t_1^1}^{t_2^1}\int_{t}^{t_2^1} P\left(t'-t \prec \tau_i\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt
\end{aligned}
\tag{4-22}
$$

Substituting (4-21) and (4-22) in (4-20), results:

$$
\begin{aligned}
A &= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) \int_{t_0}^{t_1^1} P\left(t_1^1 - t' \le \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt \\
&+ \int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t} P\left(t-t' \prec \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt + \int_{t_1^1}^{t_2^1}\int_{t}^{t_2^1} P\left(t'-t \prec \tau_i\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt
\end{aligned}
\tag{4-23}
$$

Now we calculate *B*:

$$
\begin{aligned}
B &= P\left(t_1^1 \le T_i + \tau_i, t_1^1 \le T_j + \tau_j \middle| t_0 \le T_i \le t_2^1, t_0 \le T_j \le t_2^1\right) \\
&= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i, t_1^1 \le T_j + \tau_j \middle| T_i = t, t_0 \le T_j \le t_2^1\right) f_{T_i|t_0 \le T_i \le t_2^1}\,dt + \int_{t_1^1}^{t_2^1} P\left(t_1^1 \le T_j + \tau_j \middle| T_i = t, t_0 \le T_j \le t_2^1\right) f_{T_i|t_0 \le T_i \le t_2^1}\,dt \\
&= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i, t_1^1 \le T_j + \tau_j \middle| t_0 \le T_j \le t_2^1\right) f_{T_i|t_0 \le T_i \le t_2^1}\,dt + \int_{t_1^1}^{t_2^1} P\left(t_1^1 \le T_j + \tau_j \middle| t_0 \le T_j \le t_2^1\right) f_{T_i|t_0 \le T_i \le t_2^1}\,dt \\
&= \int_{t_0}^{t_1^1}\int_{t_0}^{t} P\left(t_1^1 - t \le \tau_i, t_1^1 \le t' + \tau_j \middle| T_j = t'\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt + \int_{t_0}^{t_1^1}\int_{t}^{t_1^1} P\left(t_1^1 - t \le \tau_i, t_1^1 \le t' + \tau_j \middle| T_j = t'\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt \\
&+ \int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t} P\left(t_1^1 \le t' + \tau_j \middle| T_j = t'\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt + \int_{t_1^1}^{t_2^1}\int_{t}^{t_2^1} P\left(t_1^1 \le t' + \tau_j \middle| T_j = t'\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt \\
&= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) \int_{t_0}^{t_1^1} P\left(t_1^1 - t' \le \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt \\
&+ \int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t} f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt + \int_{t_1^1}^{t_2^1}\int_{t}^{t_2^1} f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt \\
&= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) \int_{t_0}^{t_1^1} P\left(t_1^1 - t' \le \tau_j\right) f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt + \int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t_2^1} f_{T_j|t_0 \le T_j \le t_2^1}\,dt' f_{T_i|t_0 \le T_i \le t_2^1}\,dt
\end{aligned}
\tag{4-24}
$$

Hence by substituting *A* and *B* in (4-19) we will get:

$$P\left(T_i \prec T_j + \tau_j , T_j \prec T_i + \tau_i \middle| t_0 \le T_i \le t_2^1 , t_0 \le T_j \le t_2^1 , T_i + \tau_i \ge t_1^1 , T_j + \tau_j \ge t_1^1\right)$$

$$= \frac{\left(\begin{array}{l} \int\limits_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) \int\limits_{t_0}^{t_1^1} P\left(t_1^1 - t' \le \tau_j\right) f_{T_j | t_0 \le T_j \le t_2^1} dt' f_{T_i | t_0 \le T_i \le t_2^1} dt \\ + \int\limits_{t_1^1}^{t_2^1} \int\limits_{t_1^1}^{t} P\left(t - t' \prec \tau_j\right) f_{T_j | t_0 \le T_j \le t_2^1} dt' f_{T_i | t_0 \le T_i \le t_2^1} dt + \int\limits_{t_1^1}^{t_2^1} \int\limits_{t}^{t_2^1} P\left(t' - t \prec \tau_i\right) f_{T_j | t_0 \le T_j \le t_2^1} dt' f_{T_i | t_0 \le T_i \le t_2^1} dt \end{array} \right)}{\left(\int\limits_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) \int\limits_{t_0}^{t_1^1} P\left(t_1^1 - t' \le \tau_j\right) f_{T_j | t_0 \le T_j \le t_2^1} dt' f_{T_i | t_0 \le T_i \le t_2^1} dt + \int\limits_{t_1^1}^{t_2^1} \int\limits_{t_1^1}^{t_2^1} f_{T_j | t_0 \le T_j \le t_2^1} dt' f_{T_i | t_0 \le T_i \le t_2^1} dt \right)} \tag{4-25}$$

$$\frac{P\left(T_i \prec \tau_j + t_0 \middle| t_0 \le T_i \le t_2^1 , T_i + \tau_i \ge t_1^1 , \tau_j + t_0 \ge t_1^1\right)}{}$$

$$P\left(T_i \prec \tau_j + t_0 \middle| t_0 \le T_i \le t_2^1 , T_i + \tau_i \ge t_1^1 , \tau_j + t_0 \ge t_1^1\right)$$
$$= \frac{P\left(T_i \prec \tau_j + t_0 , t_1^1 - \tau_i \le T_i , \tau_j + t_0 \ge t_1^1 \middle| t_0 \le T_i \le t_2^1\right)}{P\left(t_1^1 - \tau_i \le T_i , \tau_j + t_0 \ge t_1^1 \middle| t_0 \le T_i \le t_2^1\right)} = \frac{C}{D} \tag{4-26}$$

Below, we calculate terms *C* and *D*. For *C* we have:

$$\begin{aligned} C &= \int\limits_{t_0}^{t_1^1} P\left(t - t_0 \prec \tau_j , t_1^1 - t \le \tau_i , t_1^1 - t_0 \le \tau_j \middle| T_i = t\right) f_{T_i | t_0 \le T_i \le t_2^1} dt \\ &+ \int\limits_{t_1^1}^{t_2^1} P\left(t - t_0 \prec \tau_j , t_1^1 - t \le \tau_i , t_1^1 - t_0 \le \tau_j \middle| T_i = t\right) f_{T_i | t_0 \le T_i \le t_2^1} dt \\ &= \int\limits_{t_0}^{t_1^1} P\left(t - t_0 \prec \tau_j , t_1^1 - t \le \tau_i , t_1^1 - t_0 \le \tau_j\right) f_{T_i | t_0 \le T_i \le t_2^1} dt + \int\limits_{t_1^1}^{t_2^1} P\left(t - t_0 \prec \tau_j , t_1^1 - t_0 \le \tau_j\right) f_{T_i | t_0 \le T_i \le t_2^1} dt \\ &= \int\limits_{t_0}^{t_1^1} P\left(t - t_0 \prec \tau_j , t_1^1 - t_0 \le \tau_j\right) P\left(t_1^1 - t \le \tau_i\right) f_{T_i | t_0 \le T_i \le t_2^1} dt + \int\limits_{t_1^1}^{t_2^1} P\left(t - t_0 \prec \tau_j , t_1^1 - t_0 \le \tau_j\right) f_{T_i | t_0 \le T_i \le t_2^1} dt \\ &= \int\limits_{t_0}^{t_1^1} P\left(t_1^1 - t_0 \prec \tau_j\right) P\left(t_1^1 - t \le \tau_i\right) f_{T_i | t_0 \le T_i \le t_2^1} dt + \int\limits_{t_1^1}^{t_2^1} P\left(t - t_0 \le \tau_j\right) f_{T_i | t_0 \le T_i \le t_2^1} dt \\ &= P\left(t_1^1 - t_0 \prec \tau_j\right) \int\limits_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) f_{T_i | t_0 \le T_i \le t_2^1} dt + \int\limits_{t_1^1}^{t_2^1} P\left(t - t_0 \le \tau_j\right) f_{T_i | t_0 \le T_i \le t_2^1} dt \end{aligned} \tag{4-27}$$

And for *D* we have:

$$
\begin{aligned}
D &= P\left(t_1^1 - \tau_i \le T_i, \tau_j + t_0 \ge t_1^1 \middle| t_0 \le T_i \le t_2^1\right) \\
&= \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i, t_1^1 - t_0 \le \tau_j \middle| T_i = t\right) f_{T_i | t_0 \le T_i \le t_2^1} dt + \int_{t_1^1}^{t_2^1} P\left(t_1^1 - t_0 \le \tau_j \middle| T_i = t\right) f_{T_i | t_0 \le T_i \le t_2^1} dt \\
&= \int_{t_0}^{t_1^1} P\left(t_1^1 - t_0 \le \tau_j\right) P\left(t_1^1 - t \le \tau_i\right) f_{T_i | t_0 \le T_i \le t_2^1} dt + \int_{t_1^1}^{t_2^1} P\left(t_1^1 - t_0 \le \tau_j\right) f_{T_i | t_0 \le T_i \le t_2^1} dt \\
&= P\left(t_1^1 - t_0 \le \tau_j\right) \left(\int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) f_{T_i | t_0 \le T_i \le t_2^1} dt + \int_{t_1^1}^{t_2^1} f_{T_i | t_0 \le T_i \le t_2^1} dt \right)
\end{aligned}
\tag{4-28}
$$

Hence, by substituting *C* and *D* in (4-26), we obtain:

$$
\begin{aligned}
&P\left(T_i \prec \tau_j + t_0 \middle| t_0 \le T_i \le t_2^1, T_i + \tau_i \ge t_1^1, \tau_j + t_0 \ge t_1^1\right) \\
&= \frac{\left(P\left(t_1^1 - t_0 \prec \tau_j\right) \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) f_{T_i | t_0 \le T_i \le t_2^1} dt + \int_{t_1^1}^{t_2^1} P\left(t - t_0 \le \tau_j\right) f_{T_i | t_0 \le T_i \le t_2^1} dt \right)}{\left(P\left(t_1^1 - t_0 \le \tau_j\right) \left(\int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_i\right) f_{T_i | t_0 \le T_i \le t_2^1} dt + \int_{t_1^1}^{t_2^1} f_{T_i | t_0 \le T_i \le t_2^1} dt \right) \right)}
\end{aligned}
\tag{4-29}
$$

Similarly:

$$
\underline{P\left(T_j \prec \tau_i + t_0 \middle| t_0 \le T_j \le t_2^1, T_j + \tau_j \ge t_1^1, \tau_i + t_0 \ge t_1^1\right)}
$$

$$
\begin{aligned}
&P\left(T_j \prec \tau_i + t_0 \middle| t_0 \le T_j \le t_2^1, T_j + \tau_j \ge t_1^1, \tau_i + t_0 \ge t_1^1\right) \\
&= \frac{\left(P\left(t_1^1 - t_0 \prec \tau_i\right) \int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_j\right) f_{T_j | t_0 \le T_j \le t_2^1} dt + \int_{t_1^1}^{t_2^1} P\left(t - t_0 \le \tau_i\right) f_{T_j | t_0 \le T_j \le t_2^1} dt \right)}{\left(P\left(t_1^1 - t_0 \le \tau_i\right) \left(\int_{t_0}^{t_1^1} P\left(t_1^1 - t \le \tau_j\right) f_{T_j | t_0 \le T_j \le t_2^1} dt + \int_{t_1^1}^{t_2^1} f_{T_j | t_0 \le T_j \le t_2^1} dt \right) \right)}
\end{aligned}
\tag{4-30}
$$

Using equations (4-14), (4-18), (4-25), (4-29) and (4-30) in (4-11) and (4-12) and then substituting (4-11) and (4-12) in (4-10) we can now calculate risk $Risk_{\Phi_1}\left(S_F^1, (t_1^1, t_2^1)\right)$ for agent Φ_1.

4.3.4 Comparison Between Shuffled and Approximation Method

We provided two different approaches to calculate the risk in sections 4.3.3 and 4.3.4. In this section we compare these two methods.

In the shuffle method we design a model in which we include the behavior of all of the agents who are member of (4-2) at one place. Then we identify the global fault state, which is the combination of all the potential local bad states. And finally we calculate the probability of transitioning to global fault state using time to absorption distribution.

In the approximation method we look at the agents individually. We then calculate the probability of all agents being in their potential local bad state any time during a specific period of time.

We have the following observations:

I) In the shuffle method, we need to shuffle all the models together. In cases where the number of agents is high and the perception models are complex with too many number of states, the shuffled model will be too large. The approximation method avoids this problem by looking at the individual models separately.

II) The shuffled model has only one fault state, but in the approximation model we consider that *each* agent has an absorbing state.

III) Shuffle model only deals with Markovian models and exponential distributions, while approximation model could include those as well as general distributions.

IV) Approximation model fits appropriately with communication and collaboration framework which we will discuss shortly. In approximation model, it is possible to include fixed probabilities (including 0 and 1) for some agents. This is very important because some agents may decide to communicate their numbers directly to agent Φ_1.

V) Approximation method ignores the correlation of events $\{(S_F^1,(t_1^1,t_2^1))\}$ between agents. This can be a major source of error. Our observation is that while it is possible to include their correlations, the formulations will be quite difficult. This will be a future research topic.

4.4 Example

Consider three agents Φ_1, Φ_2, Φ_3. Models of agents Φ_2 and Φ_3 have been depicted in figure 4-10 and 4-11. These models are from agent Φ_1 point of view. The model of agent Φ_1 is not shown in here, because its information is on hand deterministically. In our notation of S_j^i, *i* is agent's index and *j* is the index of its state. The shaded states are the potential fault states.

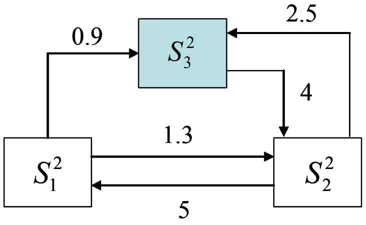

Figure 4-10: Agent Φ_2

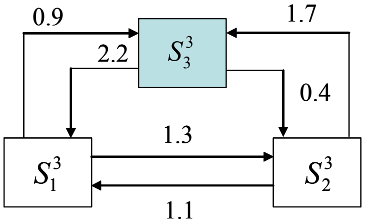

Figure 4-11: Agent Φ_3

The infinitesimal generators of the above agents can be written as:

$$\boldsymbol{G}_{\Phi_2}^1 = \begin{bmatrix} -2.2 & 1.3 & .9 \\ 5 & -7.5 & 2.5 \\ 0 & 4 & -4 \end{bmatrix}, \boldsymbol{G}_{\Phi_3}^1 = \begin{bmatrix} -2.2 & 1.3 & .9 \\ 1.1 & -2.8 & 1.7 \\ 2.2 & .4 & -2.6 \end{bmatrix} \qquad (4\text{-}31)$$

For the approximation method we consider the dark states as terminating states, and backward transitions from the fault state will be ignored, therefore the modified infinitesimal generators will be:

$$\boldsymbol{Q}_{\Phi_2}^1 = \begin{bmatrix} -2.2 & 1.3 & .9 \\ 5 & -7.5 & 2.5 \\ 0 & 0 & 0 \end{bmatrix}, \boldsymbol{Q}_{\Phi_3}^1 = \begin{bmatrix} -2.2 & 1.3 & .9 \\ 1.1 & -2.8 & 1.7 \\ 0 & 0 & 0 \end{bmatrix} \qquad (4\text{-}32)$$

We must consider the fact that the dark states are not inherently absorbing states, but for the risk calculations using the approximation method, they are considered absorbing states so that the risk of transitioning to a global fault state can be calculated.

The initial probabilities are considered to be:

$$\boldsymbol{\alpha}_{\Phi_2}^1 = \boldsymbol{\alpha}_{\Phi_3}^1 = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad (4\text{-}33)$$

The Sojourn Time rates for the agents Φ_2, Φ_3 are $\lambda_{\Phi_2}^1 = 4, \lambda_{\Phi_3}^1 = 2.6$, respectively. We also assume that $t_0 = 2, t_1^1 = 3$ and $t_2^1 = 4$.

4.4.1 Solution – Shuffle Method

First we construct the shuffled automaton of Φ_2 and Φ_3 which shows their full behaviour (see figure 4-13).

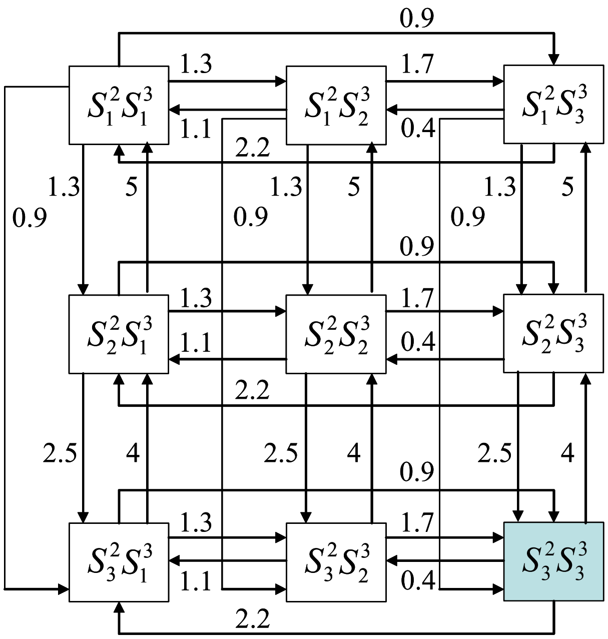

Figure 4-13: Shuffled automata Φ_2 and Φ_3

The generator of the shuffled automaton Φ_2 and Φ_3 is obtained by Kronecker sum $\oplus$ of $\boldsymbol{G}^1_{\Phi_2}$ and $\boldsymbol{G}^1_{\Phi_3}$. Then $\boldsymbol{G}^1_{\Phi_2\Phi_3} = \boldsymbol{G}^1_{\Phi_2} \oplus \boldsymbol{G}^1_{\Phi_3}$ is the following 9 by 9 matrix:

$$\boldsymbol{G} = \boldsymbol{G}^1_{\Phi_2\Phi_3} = \begin{bmatrix} -4.4 & 1.3 & .9 & 1.3 & 0 & 0 & .9 & 0 & 0 \\ 1.1 & -5 & 1.7 & 0 & 1.3 & 0 & 0 & .9 & 0 \\ 2.2 & .4 & -4.8 & 0 & 0 & 1.3 & 0 & 0 & .9 \\ 5 & 0 & 0 & -9.7 & 1.3 & .9 & 2.5 & 0 & 0 \\ 0 & 5 & 0 & 1.1 & -10.3 & 1.7 & 0 & 2.5 & 0 \\ 0 & 0 & 5 & 2.2 & .4 & -10.1 & 0 & 0 & 2.5 \\ 0 & 0 & 0 & 4 & 0 & 0 & -6.2 & 1.3 & .9 \\ 0 & 0 & 0 & 0 & 4 & 0 & 1.1 & -6.8 & 1.7 \\ 0 & 0 & 0 & 0 & 0 & 4 & 2.2 & .4 & -6.6 \end{bmatrix} \quad (4\text{-}34)$$

The lines in matrix $\boldsymbol{G}$ correspond to the transitions from composite states. Note that the last line of $\boldsymbol{G}^1_{\Phi_2\Phi_3}$ corresponds to the transitions out of the composite bad state $S^2_3S^3_3$. In order to take in account that the system never goes out from the bad state we equal to zero all the entries on the last line of the matrix $\boldsymbol{G}$. We obtain the generator $\boldsymbol{Q}$ (in which the state $S^2_3S^3_3$ is absorbing).

$$\boldsymbol{Q} = \begin{bmatrix} -4.4 & 1.3 & .9 & 1.3 & 0 & 0 & .9 & 0 & 0 \\ 1.1 & -5 & 1.7 & 0 & 1.3 & 0 & 0 & .9 & 0 \\ 2.2 & .4 & -4.8 & 0 & 0 & 1.3 & 0 & 0 & .9 \\ 5 & 0 & 0 & -9.7 & 1.3 & .9 & 2.5 & 0 & 0 \\ 0 & 5 & 0 & 1.1 & -10.3 & 1.7 & 0 & 2.5 & 0 \\ 0 & 0 & 5 & 2.2 & .4 & -10.1 & 0 & 0 & 2.5 \\ 0 & 0 & 0 & 4 & 0 & 0 & -6.2 & 1.3 & .9 \\ 0 & 0 & 0 & 0 & 4 & 0 & 1.1 & -6.8 & 1.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4-35}$$

At this point the probability distribution of the time to absorption is computed using (4-4) and the cumulative probability using (4-8). As the initial state is considered to be $S^2_1S^3_1$, the vector $\boldsymbol{\alpha}$ has a 1 in the first position and 0 elsewhere. Then the risk in the time interval (t^1_1, t^1_2) is computed using (4-9). By substituting the given numbers in the example, we get figures 4-14 through 4-16. *F(t)* and $Risk_{\Phi_1}\left(S^1_F, (t^1_1, t^1_1 + 1)\right)$ (which covers the numerical case considered in this example when $t_0 = 2, t^1_1 = 3$ and $t^1_2 = 4$) are depicted in the figures. In these figures increment is 0.01. For example, the time value of 2 corresponds to an abscissa of 200. The risk on the time interval (3, 4) is the value at point 300. For instance, the risk in time interval $(t^1_1, t^1_2) = (3,4)$ is 0.1014.

Note: In this method the value of t_0 is irrelevant. Equation (4-9) is true regardless of value of t_0. This is due to the following simple calculus:

$$\boldsymbol{\alpha}(t_0) = \boldsymbol{\alpha} e^{\boldsymbol{G}t_0} \text{ then } F_{\boldsymbol{\alpha}(t_0)}(t - t_0) = 1 - \boldsymbol{\alpha}(t_0) e^{\boldsymbol{G}(t-t_0)}\mathbf{1} = 1 - \boldsymbol{\alpha} e^{\boldsymbol{G}t_0} e^{\boldsymbol{G}(t-t_0)}\mathbf{1} = 1 - \boldsymbol{\alpha} e^{\boldsymbol{G}t}\mathbf{1} = F(t)$$

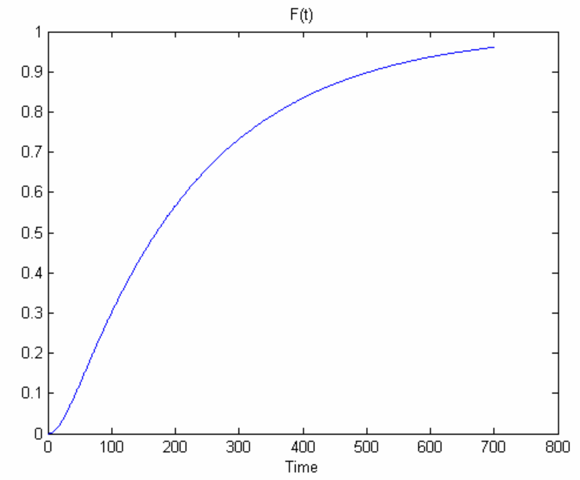

Figure 4-14: *F*(*t*) – Cumulative Distribution of Time to Enter to Global Fault State

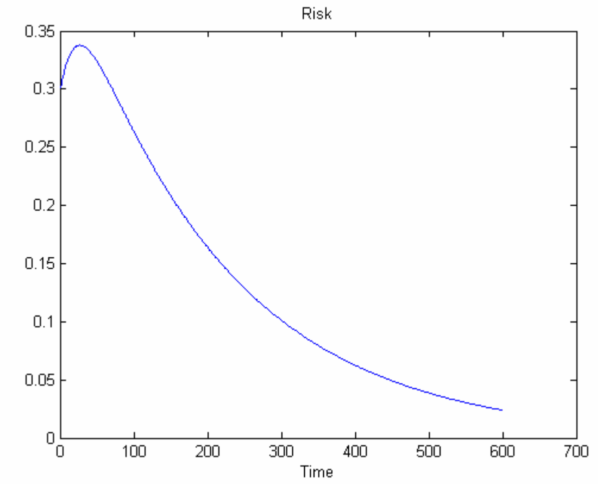

Figure 4-15: $Risk_{\Phi_1}\left(S_F^1, (t_1^1, t_1^1 + 1)\right)$

We used the same approach for calculating the Risk for another example for the same model for $\boldsymbol{G}_{\Phi_2}^1$ but a new model called $\boldsymbol{G}_{\Phi_4}^1$ with the rates below:

$$\boldsymbol{G}^1_{\Phi_4} = \begin{bmatrix} -7.3 & .4 & .6 & 1.7 & 2.3 & 0 & 1.2 & 1.1 & 0 \\ .8 & -5.15 & 2.4 & .6 & 1.25 & 0 & .1 & 0 & 0 \\ 0 & .3 & -2.82 & 0 & 0 & .32 & .5 & 1.5 & .2 \\ 0 & 2.3 & 0 & -6 & 2.1 & .65 & .95 & 0 & 0 \\ 0 & .75 & .7 & 0 & -3.25 & .3 & 1.5 & 0 & 0 \\ 0 & 1.2 & .45 & 1.35 & .6 & -4.75 & 0 & 1 & .15 \\ .75 & 0 & .4 & 0 & 0 & 0 & -2.1 & 0 & .95 \\ 0 & .5 & 0 & 0 & 2.1 & 0 & 1.7 & -5.15 & .85 \\ 0 & 0 & 0 & 0 & .4 & .3 & 0 & .7 & -1.4 \end{bmatrix}$$

As we can see in figure 4-16, the risk absolute value has decreased and the peak of the curve has shifted to the right.

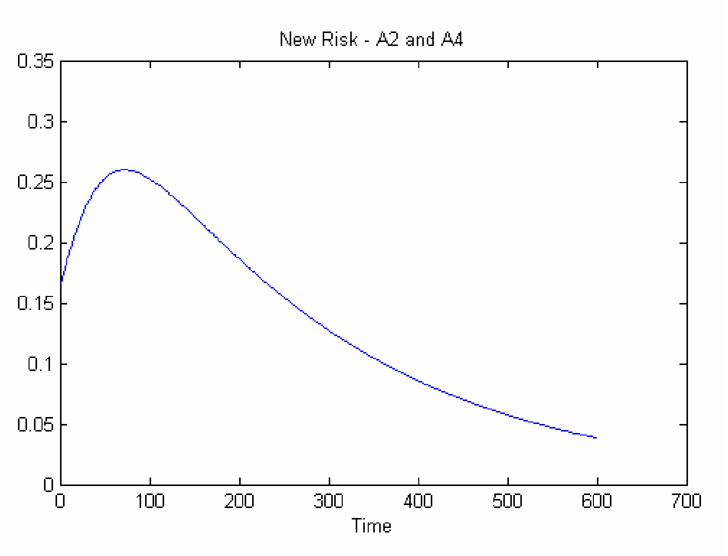

Figure 4-16: $Risk_{\Phi_1}\left(S^1_F,(t^1_1,t^1_1+1)\right)$ **with** $\boldsymbol{G}^1_{\Phi_2}$ **and** $\boldsymbol{G}^1_{\Phi_4}$

4.4.2 Solution – Approximation Method

We now solve the same problem using the approximation method. Using equation (4-12) we can write the following:

$$\begin{aligned}
&Risk_{\Phi_1}\left(S_F^1,(t_1^1,t_2^1)\right)\\
&=P(\Phi_2\in A)\prod_{j=3}\begin{pmatrix}P\left(T_2\prec T_3+\tau_3,T_3\prec T_2+\tau_2\middle|t_0\le T_2\le t_2^1,t_0\le T_3\le t_2^1,T_2+\tau_2\ge t_1^1,T_3+\tau_3\ge t_1^1\right)\cdot P(\Phi_3\in A)\\+P\left(T_2\prec\tau_3+t_0\middle|t_0\le T_2\le t_2^1,T_2+\tau_2\ge t_1^1,\tau_3+t_0\ge t_1^1\right)\cdot P(\Phi_3\in B)\end{pmatrix}\\
&+P(\Phi_2\in B)\prod_{j=3}\left(P\left(T_3\prec\tau_2+t_0\middle|t_0\le T_3\le t_2^1,T_3+\tau_3\ge t_1^1,\tau_2+t_0\ge t_1^1\right)\cdot P(\Phi_3\in A)+P(\Phi_3\in B)\right)
\end{aligned}\tag{4-36}$$

Therefore:

$$\begin{aligned}
&Risk_{\Phi_1}\left(S_F^1,(t_1^1,t_2^1)\right)\\
&=\left(\int_{t_0}^{t_1^1}P(t_1^1-t\le\tau_2)f_{T_2}(t)dt+\int_{t_1^1}^{t_2^1}f_{T_2}(t)dt\right)\left(\begin{aligned}&\frac{\begin{pmatrix}\int_{t_0}^{t_1^1}P(t_1^1-t\le\tau_2)\int_{t_0}^{t_1^1}P(t_1^1-t'\le\tau_3)f_{T_3|t_0\le T_3\le t_2^1}dt'f_{T_2|t_0\le T_2\le t_2^1}dt\\+\int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t}P(t-t'\prec\tau_3)f_{T_3|t_0\le T_3\le t_2^1}dt'f_{T_2|t_0\le T_2\le t_2^1}dt+\int_{t_1^1}^{t_2^1}\int_{t}^{t_2^1}P(t'-t\prec\tau_2)f_{T_3|t_0\le T_3\le t_2^1}dt'f_{T_2|t_0\le T_2\le t_2^1}dt\end{pmatrix}}{\left(\int_{t_0}^{t_1^1}P(t_1^1-t\le\tau_2)\int_{t_0}^{t_1^1}P(t_1^1-t'\le\tau_3)f_{T_3|t_0\le T_3\le t_2^1}dt'f_{T_2|t_0\le T_2\le t_2^1}dt+\int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t_2^1}f_{T_3|t_0\le T_3\le t_2^1}dt'f_{T_2|t_0\le T_2\le t_2^1}dt\right)}\cdot\left(\int_{t_0}^{t_1^1}P(t_1^1-t\le\tau_3)f_{T_3}(t)dt+\int_{t_1^1}^{t_2^1}f_{T_3}(t)dt\right)\\
&+\frac{\left(P(t_1^1-t_0\prec\tau_3)\int_{t_0}^{t_1^1}P(t_1^1-t\le\tau_2)f_{T_2|t_0\le T_2\le t_2^1}dt+\int_{t_1^1}^{t_2^1}P(t-t_0\le\tau_3)f_{T_2|t_0\le T_2\le t_2^1}dt\right)}{\left(P(t_1^1-t_0\le\tau_3)\left(\int_{t_0}^{t_1^1}P(t_1^1-t\le\tau_2)f_{T_2|t_0\le T_2\le t_2^1}dt+\int_{t_1^1}^{t_2^1}f_{T_2|t_0\le T_2\le t_2^1}dt\right)\right)}\cdot P(T_3\prec t_0)P(t_1^1\le\tau_3+t_0)\end{aligned}\right)\\
&+P(T_2\prec t_0)P(t_1^1\le\tau_2+t_0)\left(\frac{\left(P(t_1^1-t_0\prec\tau_2)\int_{t_0}^{t_1^1}P(t_1^1-t\le\tau_3)f_{T_3|t_0\le T_3\le t_2^1}dt+\int_{t_1^1}^{t_2^1}P(t-t_0\le\tau_2)f_{T_3|t_0\le T_3\le t_2^1}dt\right)}{\left(P(t_1^1-t_0\le\tau_2)\left(\int_{t_0}^{t_1^1}P(t_1^1-t\le\tau_3)f_{T_3|t_0\le T_3\le t_2^1}dt+\int_{t_1^1}^{t_2^1}f_{T_3|t_0\le T_3\le t_2^1}dt\right)\right)}\cdot\left(\int_{t_0}^{t_1^1}P(t_1^1-t\le\tau_3)f_{T_3}(t)dt+\int_{t_1^1}^{t_2^1}f_{T_3}(t)dt\right)+P(T_3\prec t_0)P(t_1^1\le\tau_3+t_0)\right)
\end{aligned}\tag{4-37}$$

With the assumption that the sojourn times are exponential and also equation (4-16), we can extend equation (4-37) as follows. We know that:

$$\int_{t_0}^{t_1^1}P\left(t_1^1-t\prec\tau_i\right)f_{T_i|t_0\le T_i\le t_2^1}(t)dt=\int_{t_0}^{t_1^1}\left(1-\int_0^{t_1^1-t}\lambda_{\Phi_2}^1e^{-\lambda_{\Phi_2}^1x}dx\right)\frac{\boldsymbol{\alpha}_{\Phi_i}^1e^{\boldsymbol{T}_{\Phi_i}^1t}(-\boldsymbol{T}_{\Phi_i}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_i}^1\left(e^{\boldsymbol{T}_{\Phi_i}^1t_0}-e^{\boldsymbol{T}_{\Phi_i}^1t_2^1}\right)\mathbf{1}}dt\tag{4-38}$$

For simplification, we assume:

$$Risk_{\Phi_1}\left(S_F^1,(t_1^1,t_2^1)\right)=A\left(\frac{B}{C}\cdot D+\frac{E}{F}\cdot G\right)+H\left(\frac{I}{J}\cdot D+G\right)\tag{4-39}$$

Now we calculate *A*, *B*, *C*, *D*, *E*, *F*, *G*, *H*, *I* and *J* respectively:

$$A = \int_{t_0}^{t_1^1}\left(1-\int_0^{t_1^1-t}\lambda^1_{\Phi_2}e^{-\lambda^1_{\Phi_2}x}dx\right)\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})dt+\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})dt$$

$$=\int_{t_0}^{t_1^1}e^{-\lambda^1_{\Phi_2}(t_1^1-t)}\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})dt+\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})dt$$

$$B = \int_{t_0}^{t_1^1}\left(1-\int_0^{t_1^1-t}\lambda^1_{\Phi_2}e^{-\lambda^1_{\Phi_2}x}dx\right)\int_{t_0}^{t_1^1}\left(1-\int_0^{t_1^1-t'}\lambda^1_{\Phi_3}e^{-\lambda^1_{\Phi_3}x}dx\right)\frac{\boldsymbol{\alpha}^1_{\Phi_3}e^{\boldsymbol{T}^1_{\Phi_3}t'}(-\boldsymbol{T}^1_{\Phi_3}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_3}\left(e^{\boldsymbol{T}^1_{\Phi_3}t_0}-e^{\boldsymbol{T}^1_{\Phi_3}t_2^1}\right)\mathbf{1}}dt'\,\frac{\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_2}\left(e^{\boldsymbol{T}^1_{\Phi_2}t_0}-e^{\boldsymbol{T}^1_{\Phi_2}t_2^1}\right)\mathbf{1}}dt$$

$$+\int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t}\left(1-\int_0^{t-t'}\lambda^1_{\Phi_3}e^{-\lambda^1_{\Phi_3}x}dx\right)\frac{\boldsymbol{\alpha}^1_{\Phi_3}e^{\boldsymbol{T}^1_{\Phi_3}t'}(-\boldsymbol{T}^1_{\Phi_3}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_3}\left(e^{\boldsymbol{T}^1_{\Phi_3}t_0}-e^{\boldsymbol{T}^1_{\Phi_3}t_2^1}\right)\mathbf{1}}dt'\,\frac{\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_2}\left(e^{\boldsymbol{T}^1_{\Phi_2}t_0}-e^{\boldsymbol{T}^1_{\Phi_2}t_2^1}\right)\mathbf{1}}dt$$

$$+\int_{t_1^1}^{t_2^1}\int_{t}^{t_2^1}\left(1-\int_0^{t'-t}\lambda^1_{\Phi_2}e^{-\lambda^1_{\Phi_2}x}dx\right)\frac{\boldsymbol{\alpha}^1_{\Phi_3}e^{\boldsymbol{T}^1_{\Phi_3}t'}(-\boldsymbol{T}^1_{\Phi_3}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_3}\left(e^{\boldsymbol{T}^1_{\Phi_3}t_0}-e^{\boldsymbol{T}^1_{\Phi_3}t_2^1}\right)\mathbf{1}}dt'\,\frac{\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_2}\left(e^{\boldsymbol{T}^1_{\Phi_2}t_0}-e^{\boldsymbol{T}^1_{\Phi_2}t_2^1}\right)\mathbf{1}}dt$$

$$=\int_{t_0}^{t_1^1}e^{-\lambda^1_{\Phi_2}\left(t_1^1-t\right)}\int_{t_0}^{t_1^1}e^{-\lambda^1_{\Phi_3}\left(t_1^1-t'\right)}\frac{\boldsymbol{\alpha}^1_{\Phi_3}e^{\boldsymbol{T}^1_{\Phi_3}t'}(-\boldsymbol{T}^1_{\Phi_3}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_3}\left(e^{\boldsymbol{T}^1_{\Phi_3}t_0}-e^{\boldsymbol{T}^1_{\Phi_3}t_2^1}\right)\mathbf{1}}dt'\,\frac{\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_2}\left(e^{\boldsymbol{T}^1_{\Phi_2}t_0}-e^{\boldsymbol{T}^1_{\Phi_2}t_2^1}\right)\mathbf{1}}dt$$

$$+\int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t}e^{-\lambda^1_{\Phi_3}(t-t')}\frac{\boldsymbol{\alpha}^1_{\Phi_3}e^{\boldsymbol{T}^1_{\Phi_3}t'}(-\boldsymbol{T}^1_{\Phi_3}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_3}\left(e^{\boldsymbol{T}^1_{\Phi_3}t_0}-e^{\boldsymbol{T}^1_{\Phi_3}t_2^1}\right)\mathbf{1}}dt'\,\frac{\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_2}\left(e^{\boldsymbol{T}^1_{\Phi_2}t_0}-e^{\boldsymbol{T}^1_{\Phi_2}t_2^1}\right)\mathbf{1}}dt$$

$$+\int_{t_1^1}^{t_2^1}\int_{t}^{t_2^1}e^{-\lambda^1_{\Phi_2}(t'-t)}\frac{\boldsymbol{\alpha}^1_{\Phi_3}e^{\boldsymbol{T}^1_{\Phi_3}t'}(-\boldsymbol{T}^1_{\Phi_3}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_3}\left(e^{\boldsymbol{T}^1_{\Phi_3}t_0}-e^{\boldsymbol{T}^1_{\Phi_3}t_2^1}\right)\mathbf{1}}dt'\,\frac{\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_2}\left(e^{\boldsymbol{T}^1_{\Phi_2}t_0}-e^{\boldsymbol{T}^1_{\Phi_2}t_2^1}\right)\mathbf{1}}dt$$

$$C = \int_{t_0}^{t_1^1}\left(1-\int_0^{t_1^1-t}\lambda^1_{\Phi_2}e^{-\lambda^1_{\Phi_2}x}dx\right)\int_{t_0}^{t_1^1}\left(\int_0^{t_1^1-t'}\lambda^1_{\Phi_3}e^{-\lambda^1_{\Phi_3}x}dx\right)\frac{\boldsymbol{\alpha}^1_{\Phi_3}e^{\boldsymbol{T}^1_{\Phi_3}t'}(-\boldsymbol{T}^1_{\Phi_3}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_3}\left(e^{\boldsymbol{T}^1_{\Phi_3}t_0}-e^{\boldsymbol{T}^1_{\Phi_3}t_2^1}\right)\mathbf{1}}dt'\,\frac{\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_2}\left(e^{\boldsymbol{T}^1_{\Phi_2}t_0}-e^{\boldsymbol{T}^1_{\Phi_2}t_2^1}\right)\mathbf{1}}dt$$

$$+\int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t_2^1}\frac{\boldsymbol{\alpha}^1_{\Phi_3}e^{\boldsymbol{T}^1_{\Phi_3}t'}(-\boldsymbol{T}^1_{\Phi_3}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_3}\left(e^{\boldsymbol{T}^1_{\Phi_3}t_0}-e^{\boldsymbol{T}^1_{\Phi_3}t_2^1}\right)\mathbf{1}}dt'\,\frac{\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_2}\left(e^{\boldsymbol{T}^1_{\Phi_2}t_0}-e^{\boldsymbol{T}^1_{\Phi_2}t_2^1}\right)\mathbf{1}}dt$$

$$=\int_{t_0}^{t_1^1}e^{-\lambda^1_{\Phi_2}\left(t_1^1-t\right)}\int_{t_0}^{t_1^1}e^{-\lambda^1_{\Phi_3}\left(t_1^1-t'\right)}\frac{\boldsymbol{\alpha}^1_{\Phi_3}e^{\boldsymbol{T}^1_{\Phi_3}t'}(-\boldsymbol{T}^1_{\Phi_3}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_3}\left(e^{\boldsymbol{T}^1_{\Phi_3}t_0}-e^{\boldsymbol{T}^1_{\Phi_3}t_2^1}\right)\mathbf{1}}dt'\,\frac{\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_2}\left(e^{\boldsymbol{T}^1_{\Phi_2}t_0}-e^{\boldsymbol{T}^1_{\Phi_2}t_2^1}\right)\mathbf{1}}dt$$

$$+\int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t_2^1}\frac{\boldsymbol{\alpha}^1_{\Phi_3}e^{\boldsymbol{T}^1_{\Phi_3}t'}(-\boldsymbol{T}^1_{\Phi_3}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_3}\left(e^{\boldsymbol{T}^1_{\Phi_3}t_0}-e^{\boldsymbol{T}^1_{\Phi_3}t_2^1}\right)\mathbf{1}}dt'\,\frac{\boldsymbol{\alpha}^1_{\Phi_2}e^{\boldsymbol{T}^1_{\Phi_2}t}(-\boldsymbol{T}^1_{\Phi_2}\mathbf{1})}{\boldsymbol{\alpha}^1_{\Phi_2}\left(e^{\boldsymbol{T}^1_{\Phi_2}t_0}-e^{\boldsymbol{T}^1_{\Phi_2}t_2^1}\right)\mathbf{1}}dt$$

$$\frac{B}{C}=\frac{\begin{pmatrix}\displaystyle\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_2}^1\left(t_1^1-t\right)}\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_3}^1\left(t_1^1-t'\right)}\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t'}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})dt'\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})dt\\ \displaystyle+\int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t} e^{-\lambda_{\Phi_3}^1\left(t-t'\right)}\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t'}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})dt'\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})dt\\ \displaystyle+\int_{t_1^1}^{t_2^1}\int_{t}^{t_2^1} e^{-\lambda_{\Phi_2}^1\left(t'-t\right)}\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t'}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})dt'\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})dt\end{pmatrix}}{\begin{pmatrix}\displaystyle\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_2}^1\left(t_1^1-t\right)}\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_3}^1\left(t_1^1-t'\right)}\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t'}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})dt'\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})dt\\ \displaystyle+\int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t'}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})dt'\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})dt\end{pmatrix}}$$

$$D=\int_{t_0}^{t_1^1}\left(1-\int_0^{t_1^1-t}\lambda_{\Phi_3}^1 e^{-\lambda_{\Phi_3}^1 x}dx\right)\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})dt+\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})dt$$

$$=\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_3}^1\left(t_1^1-t\right)}\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})dt+\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})dt$$

$$E=\left(1-\int_0^{t_1^1-t_0}\lambda_{\Phi_3}^1 e^{-\lambda_{\Phi_3}^1 x}dx\right)\int_{t_0}^{t_1^1}\left(1-\int_0^{t_1^1-t}\lambda_{\Phi_2}^1 e^{-\lambda_{\Phi_3}^1 x}dx\right)\frac{\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_2}^1\left(e^{\boldsymbol{T}_{\Phi_2}^1 t_0}-e^{\boldsymbol{T}_{\Phi_2}^1 t_2^1}\right)\mathbf{1}}dt$$

$$+\int_{t_1^1}^{t_2^1}\left(1-\int_0^{t-t_0}\lambda_{\Phi_3}^1 e^{-\lambda_{\Phi_2}^1 x}dx\right)\frac{\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_2}^1\left(e^{\boldsymbol{T}_{\Phi_2}^1 t_0}-e^{\boldsymbol{T}_{\Phi_2}^1 t_2^1}\right)\mathbf{1}}dt$$

$$=e^{-\lambda_{\Phi_3}^1\left(t_1^1-t_0\right)}\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_2}^1\left(t_1^1-t\right)}\frac{\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_2}^1\left(e^{\boldsymbol{T}_{\Phi_2}^1 t_0}-e^{\boldsymbol{T}_{\Phi_2}^1 t_2^1}\right)\mathbf{1}}dt+\int_{t_1^1}^{t_2^1} e^{-\lambda_{\Phi_3}^1\left(t-t_0\right)}\frac{\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_2}^1\left(e^{\boldsymbol{T}_{\Phi_2}^1 t_0}-e^{\boldsymbol{T}_{\Phi_2}^1 t_2^1}\right)\mathbf{1}}dt$$

$$F=\left(1-\int_0^{t_1^1-t_0}\lambda_{\Phi_3}^1 e^{-\lambda_{\Phi_3}^1 x}dx\right)\left(\int_{t_0}^{t_1^1}\left(1-\int_0^{t_1^1-t}\lambda_{\Phi_2}^1 e^{-\lambda_{\Phi_2}^1 x}dx\right)\frac{\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_2}^1\left(e^{\boldsymbol{T}_{\Phi_2}^1 t_0}-e^{\boldsymbol{T}_{\Phi_2}^1 t_2^1}\right)\mathbf{1}}dt+\int_{t_1^1}^{t_2^1}\frac{\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_2}^1\left(e^{\boldsymbol{T}_{\Phi_2}^1 t_0}-e^{\boldsymbol{T}_{\Phi_2}^1 t_2^1}\right)\mathbf{1}}dt\right)$$

$$=e^{-\lambda_{\Phi_3}^1\left(t_1^1-t_0\right)}\left(\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_2}^1\left(t_1^1-t\right)}\frac{\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_2}^1\left(e^{\boldsymbol{T}_{\Phi_2}^1 t_0}-e^{\boldsymbol{T}_{\Phi_2}^1 t_2^1}\right)\mathbf{1}}dt+\int_{t_1^1}^{t_2^1}\frac{\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_2}^1\left(e^{\boldsymbol{T}_{\Phi_2}^1 t_0}-e^{\boldsymbol{T}_{\Phi_2}^1 t_2^1}\right)\mathbf{1}}dt\right)$$

$$\frac{E}{F}=\frac{\left(e^{-\lambda_{\Phi_3}^1\left(t_1^1-t_0\right)}\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_2}^1\left(t_1^1-t\right)}\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})+\int_{t_1^1}^{t_2^1} e^{-\lambda_{\Phi_3}^1\left(t-t_0\right)}\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})dt\right)}{e^{-\lambda_{\Phi_3}^1\left(t_1^1-t_0\right)}\left(\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_2}^1\left(t_1^1-t\right)}\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})dt+\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})dt\right)}$$

$$G=P\left(T_3\prec t_0\right)P\left(t_1^1-t_0\le\tau_3\right)=\int_0^{t_0}\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})dt\left(1-\int_0^{t_1^1-t_0}\lambda_{\Phi_3}^1 e^{-\lambda_{\Phi_3}^1 x}dx\right)$$

$$=e^{-\lambda_{\Phi_3}^1\left(t_1^1-t_0\right)}\int_0^{t_0}\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})dt$$

$$H=P\left(T_2\prec t_0\right)P\left(t_1^1-t_0\le\tau_2\right)=\int_0^{t_0}\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})dt\left(1-\int_0^{t_1^1-t_0}\lambda_{\Phi_2}^1 e^{-\lambda_{\Phi_2}^1 x}dx\right)$$

$$=e^{-\lambda_{\Phi_2}^1\left(t_1^1-t_0\right)}\int_0^{t_0}\boldsymbol{\alpha}_{\Phi_2}^1 e^{\boldsymbol{T}_{\Phi_2}^1 t}(-\boldsymbol{T}_{\Phi_2}^1\mathbf{1})dt$$

$$I=\left(1-\int_0^{t_1^1-t_0}\lambda_{\Phi_2}^1 e^{-\lambda_{\Phi_2}^1 x}dx\right)\int_{t_0}^{t_1^1}\left(1-\int_0^{t_1^1-t}\lambda_{\Phi_3}^1 e^{-\lambda_{\Phi_3}^1 x}dx\right)\frac{\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_3}^1\left(e^{\boldsymbol{T}_{\Phi_3}^1 t_0}-e^{\boldsymbol{T}_{\Phi_3}^1 t_2^1}\right)\mathbf{1}}dt$$

$$+\int_{t_1^1}^{t_2^1}\left(1-\int_0^{t-t_0}\lambda_{\Phi_2}^1 e^{-\lambda_{\Phi_2}^1 x}dx\right)\frac{\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_3}^1\left(e^{\boldsymbol{T}_{\Phi_3}^1 t_0}-e^{\boldsymbol{T}_{\Phi_3}^1 t_2^1}\right)\mathbf{1}}dt$$

$$=e^{-\lambda_{\Phi_2}^1\left(t_1^1-t_0\right)}\int_{t_0}^{t_1^1}e^{-\lambda_{\Phi_3}^1\left(t_1^1-t\right)}\frac{\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_3}^1\left(e^{\boldsymbol{T}_{\Phi_3}^1 t_0}-e^{\boldsymbol{T}_{\Phi_3}^1 t_2^1}\right)\mathbf{1}}dt+\int_{t_1^1}^{t_2^1}e^{-\lambda_{\Phi_2}^1\left(t-t_0\right)}\frac{\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_3}^1\left(e^{\boldsymbol{T}_{\Phi_3}^1 t_0}-e^{\boldsymbol{T}_{\Phi_3}^1 t_2^1}\right)\mathbf{1}}dt$$

$$J=\left(1-\int_0^{t_1^1-t_0}\lambda_{\Phi_2}^1 e^{-\lambda_{\Phi_2}^1 x}dx\right)\left(\int_{t_0}^{t_1^1}\left(1-\int_0^{t_1^1-t}\lambda_{\Phi_3}^1 e^{-\lambda_{\Phi_3}^1 x}dx\right)\frac{\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_3}^1\left(e^{\boldsymbol{T}_{\Phi_3}^1 t_0}-e^{\boldsymbol{T}_{\Phi_3}^1 t_2^1}\right)\mathbf{1}}dt+\int_{t_1^1}^{t_2^1}\frac{\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_3}^1\left(e^{\boldsymbol{T}_{\Phi_3}^1 t_0}-e^{\boldsymbol{T}_{\Phi_3}^1 t_2^1}\right)\mathbf{1}}dt\right)$$

$$=e^{-\lambda_{\Phi_2}^1\left(t_1^1-t_0\right)}\left(\int_{t_0}^{t_1^1}e^{-\lambda_{\Phi_3}^1\left(t_1^1-t\right)}\frac{\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_3}^1\left(e^{\boldsymbol{T}_{\Phi_3}^1 t_0}-e^{\boldsymbol{T}_{\Phi_3}^1 t_2^1}\right)\mathbf{1}}dt+\int_{t_1^1}^{t_2^1}\frac{\boldsymbol{\alpha}_{\Phi_3}^1 e^{\boldsymbol{T}_{\Phi_3}^1 t}(-\boldsymbol{T}_{\Phi_3}^1\mathbf{1})}{\boldsymbol{\alpha}_{\Phi_3}^1\left(e^{\boldsymbol{T}_{\Phi_3}^1 t_0}-e^{\boldsymbol{T}_{\Phi_3}^1 t_2^1}\right)\mathbf{1}}dt\right)$$

$$\frac{I}{J}=\frac{e^{-\lambda_{\Phi_2}^{1}\left(t_1^1-t_0\right)}\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt+\int_{t_1^1}^{t_2^1} e^{-\lambda_{\Phi_2}^{1}\left(t-t_0\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt}{e^{-\lambda_{\Phi_2}^{1}\left(t_1^1-t_0\right)}\left(\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt+\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt\right)}$$

By substituting *A*, *B*, *C*, *D*, *E*, *F*, *G*, *H*, *I* and *J* into equation (4-39), we will get:

$$\begin{aligned}
&Risk_{\Phi_1}\left(S_F^1,(t_1^1,t_2^1)\right)=A\left(\frac{B}{C}\cdot D+\frac{E}{F}\cdot G\right)+H\left(\frac{I}{J}\cdot D+G\right)\\
&=\left(\begin{aligned}&\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_2}^{1}(t_1^1-t)}\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})dt\\&+\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})dt\end{aligned}\right)
\left(\begin{aligned}
&\frac{\left(\begin{aligned}&\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_2}^{1}\left(t_1^1-t\right)}\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t'\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t'}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt'\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})dt\\
&+\int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t} e^{-\lambda_{\Phi_3}^{1}\left(t-t'\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t'}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt'\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})dt\\
&+\int_{t_1^1}^{t_2^1}\int_{t}^{t_2^1} e^{-\lambda_{\Phi_2}^{1}\left(t'-t\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t'}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt'\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})dt\end{aligned}\right)}{\left(\begin{aligned}&\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_2}^{1}\left(t_1^1-t\right)}\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t'\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t'}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt'\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})dt\\
&+\int_{t_1^1}^{t_2^1}\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t'}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt'\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})dt\end{aligned}\right)}\\
&\cdot\left(\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt+\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt\right)\\
&+\frac{\left(e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t_0\right)}\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_2}^{1}\left(t_1^1-t\right)}\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})+\int_{t_1^1}^{t_2^1} e^{-\lambda_{\Phi_3}^{1}\left(t-t_0\right)}\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})dt\right)}{e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t_0\right)}\left(\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_2}^{1}\left(t_1^1-t\right)}\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})dt+\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})dt\right)}\\
&\cdot e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t_0\right)}\int_{0}^{t_0}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt
\end{aligned}\right)\\
&+e^{-\lambda_{\Phi_2}^{1}\left(t_1^1-t_0\right)}\int_{0}^{t_0}\boldsymbol{\alpha}_{\Phi_2}^{1} e^{\boldsymbol{T}_{\Phi_2}^{1} t}(-\boldsymbol{T}_{\Phi_2}^{1}\mathbf{1})dt\\
&\cdot\left(\begin{aligned}&\frac{e^{-\lambda_{\Phi_2}^{1}\left(t_1^1-t_0\right)}\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt+\int_{t_1^1}^{t_2^1} e^{-\lambda_{\Phi_2}^{1}\left(t-t_0\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt}{e^{-\lambda_{\Phi_2}^{1}\left(t_1^1-t_0\right)}\left(\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt+\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt\right)}\\
&\cdot\left(\int_{t_0}^{t_1^1} e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t\right)}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt+\int_{t_1^1}^{t_2^1}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt\right)+e^{-\lambda_{\Phi_3}^{1}\left(t_1^1-t_0\right)}\int_{0}^{t_0}\boldsymbol{\alpha}_{\Phi_3}^{1} e^{\boldsymbol{T}_{\Phi_3}^{1} t}(-\boldsymbol{T}_{\Phi_3}^{1}\mathbf{1})dt\end{aligned}\right)
\end{aligned}\tag{4-40}$$

Using Maple we obtain:

$Risk_{\Phi_1}\left(S_F^1,(t_1^1,t_2^1)\right)=0.0037$

Figures 4-17 through 4-20 show the risk as a function of t_0, t_1^1, t_2^1 and $t_1^1 - t_0$ respectively.

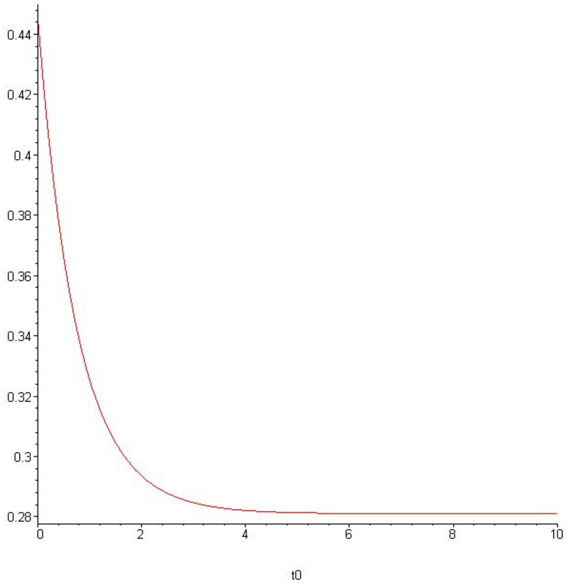

Figure 4-17: Risk as a function of t_0

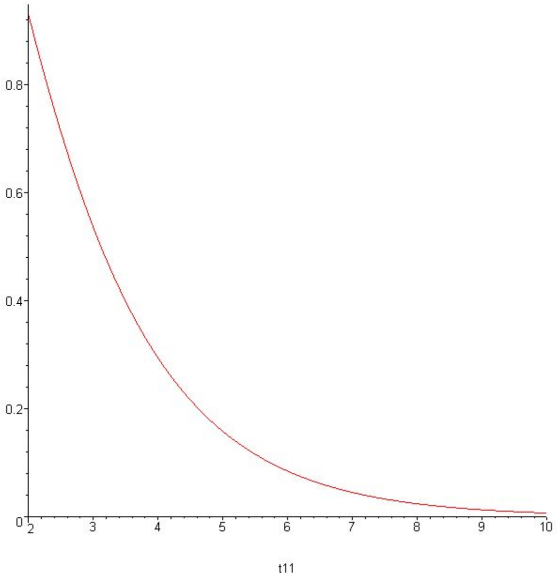

Figure 4-18: Risk as a function of t_1^1

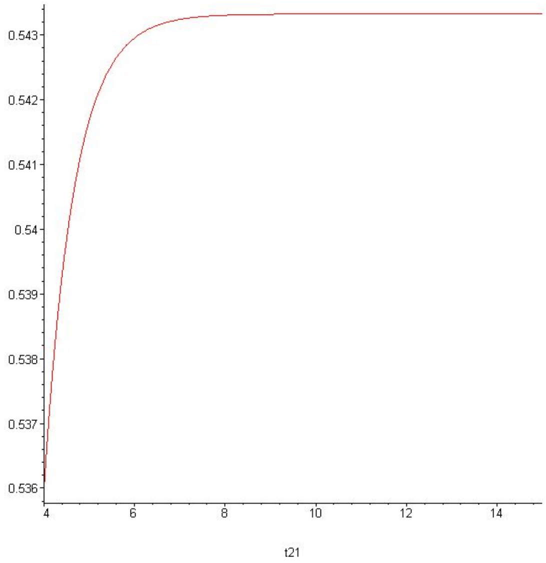

Figure 4-19: Risk as a function of t_2^1

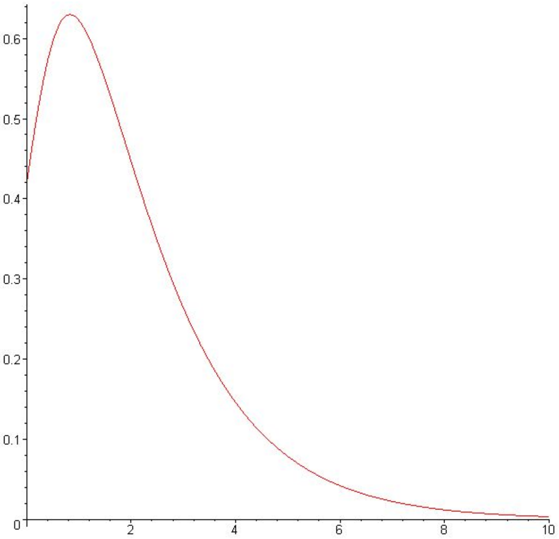

Figure 4-20: Risk as a function of $t_1^1 - t_0$

If we change the Sojourn Time rates of the agents to $\lambda_{\Phi_2}^1 = 0.25, \lambda_{\Phi_3}^1 = 0.3846$ (reciprocals of the initial values), respectively then the following result is obtained:

$Risk_{\Phi_1}\left(S_F^1,(t_1^1,t_2^1)\right)=0.5359$.

This is intuitively obvious, since by increasing the mean time spending in the potential bad states the agents will be more likely prone to transition to a global fault state and therefore the risk in increasing.

4.4.3 Interpretation of Figures

Using the results obtained from the shuffle and approximate models, we note the following – When the agents are significantly homogeneous in terms of their states and transitions, the likelihood of them reaching a conflict state is more at the beginning than later. When these agents are more heterogeneous, then this likelihood becomes more spread across the timeline, i.e. we obtain a wider distribution (higher variance). This information is important for one agent to predict the behavior of similar agents without extensive requirements on communication and collaborations.

4.5 Communication and Collaboration

The agents strive to keep their risk within an accepted range. If the risk of execution of an action is determined to be high, agents attempt to cooperate with each other. Cooperation includes two major factors; the first one is *communication*. Communication is the intentional exchange of information brought about by the production and perception of signs drawn from a shared system of conventional signs [Russell 03]. The exchange of structured messages can be enabled using a mechanism called “Language.” We assume that the agents use a common language to exchange information. The second factor is *collaboration*. If communication factor is not enough to reduce the risk then the agents attempt to collaborate. Collaboration, which requires communication as well, helps to reduce the amount of risk by rescheduling the local task plans. Communication and collaboration will be discussed in this section in more detail. We will assume that agents, if decide to communicate and collaborate, are honest and reliable. Furthermore, they are not selfish.

4.5.1 Communication Framework

In terms of communication the agents can provide the requesting agent Φ_i with two types of major information. The requester agent in this context is the one who initializes the risk calculation. We assume that any agent Φ_j located in the community of agent Φ_i will provide the following information to agent Φ_i upon request:

- Its schedule with respect to time period $\left(t_1^1, t_2^1\right)$ - it can be deterministic or probabilistic. If agent Φ_j communicates to Φ_i the exact time period as to when it will be transitioning in (and out) to (and from) its potential local fault state S_F^j, then Φ_i will be able to make more accurate risk calculations.
- Its current state at time t_0 – it is usually deterministic. Generally, this piece of information will be deterministic, because it will be about the current situation of agent Φ_j. Knowing this fact will help agent Φ_i to set $P\left(\Phi_j \in \mathscr{F}_0\right)$ to a value between 0 and 1 in its calculations.

Note: We can extend the amount of information that will be exchanged between the agents by assuming that the agents communicate their fully extended models. Evidently, an agent has a more complete model of itself compared to the requester agents. If it provides this model to the public community (requesters), then the requesters can make better decisions based on a more complete model. But on the other hand, every agent is prone to privacy and does not necessarily like to fully open up to other agents.

If these pieces of information are enough to reduce the risk factor sufficiently, then Φ_i will not go to the next step, otherwise it will trigger the collaboration methodology to reduce the risk factor.

4.5.2 Collaboration Framework

If the estimated risk (calculated by the requester agent Φ_i) is too high (compared to a given threshold), then the agent will try to reduce its risk (cost), using collaboration. Agent Φ_1 announces its high risk situation together with time period $\left(t_1^1, t_2^1\right)$ to the involved parties (see set FC - equation (4-2)).

Involved parties will investigate their schedule to see whether there is a potential overlap existing in the set *FC*. If any one agent disagrees with the overlap time, with 100% certainty – calculated from its own perception model and schedule – then no further collaboration will be required. Agent Φ_i assumes no risk. If all the parties agree on a possible overlap during $\left(t_1^1, t_2^1\right)$ then the collaboration process starts.

The main idea of collaboration in this work is to reduce the risk by moving away from the overlap period as much as needed and by keeping the cost of the movement (rescheduling) as low as possible.

As we see from figure 4-21, there exists an overlap between time t_1^1 and time t_2^2 (It's depicted between dashed lines), which makes the collaboration necessary to reduce the overall risk and cost.

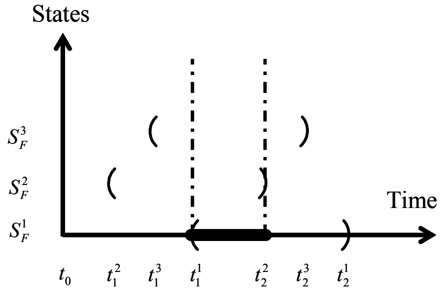

Figure 4-21: Overlap between agents

4.5.2.1 Game Theory

In this section we look at those aspects of game theory that can be used to analyze games with simultaneous moves. We look only at games where the players make only one move. Game theory can be used in two ways: *agent design* and *mechanism design*. In mechanism design which is the approach in this section, the collective good of all agents (avoiding the global fault state) is maximized when each agent adopts the game-theoretic solution that maximizes its own utility [Russell 03].

A **game** is defined by [Russell 03]:

- *Players* (Agents) who will be making decisions.

- *Actions* that the players can choose
- A *payoff matrix* that gives the utility to each player for each combination.

Each player must adopt and then execute a *strategy* (policy). A *pure* strategy is a deterministic policy specifying a particular action to take in each situation (in a one move game the pure strategy is a single action.). On the other hand, a *mixed* strategy is a randomized policy that selects particular actions according to a specific probability distribution over actions. A *strategy profile* is an assignment of a strategy to each player; given the strategy profile the game's *outcome* is a numeric value for each player. A *solution* to a game is a strategy profile in which each player adopts a rational strategy. Outcomes are actual results of playing a game, while solutions are theoretical constructs used to analyze a game.

Now let us look at an example in our framework (figure 4-22) where we only have two agents involved and where we assume that these agents move in opposite directions.

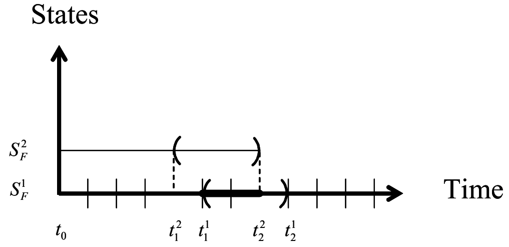

Figure 4-22: Case of Two Agents

In the sequel we will assume that agents involved in condition (4-2) announce their membership to one of the two sets; left-moving and right-moving agents. We define left-moving agents the ones who are willing to expedite their schedules. We define right-moving agents who are willing to delay their schedules.

<u>Case of two agents (figure 4-22)</u>

Assumptions:

- There are two agents Φ_1 and Φ_2 with an overlap of two time units.
- It is given that both agents can change their schedule by at most 4 time units.
- Φ_1 can move only to the right and Φ_2 can move only to the left (reasonable moves – shortest path moves are in the opposite direction).
- Each move per time unit costs $1 and is the same for both agents

We can consider the two player game of figure 4-23. The objective of the game is to clear the overlap.

Φ_1 \ Φ_2	0	-1	-2	-3
0	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(0,-2)$	$(0,-3)$
1	$(-\infty,-\infty)$	$(-1,-1)$	$(-1,-2)$	$(-1,-3)$
2	$(-2,0)$	$(-2,-1)$	$(-2,-2)$	$(-2,-3)$
3	$(-3,0)$	$(-3,-1)$	$(-3,-2)$	$(-3,-3)$

Figure 4-23: Two Player Game Payoff Matrix

The points (-2,0), (-1,-1) and (0,-2) are Nash Equilibrium. Because any deviation around these points wouldn't improve the cost (profit) of player one and two (Agents Φ_1 and Φ_2).

Even though there might be different Nash Equilibrium points, but in order to find a reasonable solution the player might agree on choosing the *Pareto-optimal* solution (i.e. when there is no other outcome that all player would prefer.) It is known that every game has at least one Pareto-optimal solution. In cases where the solutions have the same outcome (two or more equal Pareto-optimal equilibrium points) the players can "*communicate*." The communication can be based on establishment of convention that orders the solutions before the game begins or it can be based on negotiation to reach a mutually beneficial solution during the game. Games in which players need to communicate like our framework are called *coordination games*. In our framework we assume that the agents strive to minimize the joint cost and share it as fairly as possible. Hence the outcome of the game introduced in our example would be (-1,-1).

Case of two agents (figure 4-24) – More General Case

Assumptions:

- There are two agents Φ_1 and Φ_2 with an overlap of two time units.
- There is no limitation on number of moves.
- Φ_1 and Φ_2 can both move on either direction.

Φ_2 Φ_1	-3	-2	-1	0	1	2	3
-3	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$	(-3,-1)	(-3,-2)	(-3,-3)
-2	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$	(-2,-2)	(-2,-3)
-1	(-1,-3)	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$	(-1,-3)
0	(0,-3)	(0,-2)	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$
1	(-1,-3)	(-1,-2)	(-1,-1)	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$
2	(-2,-3)	(-2,-2)	(-2,-1)	(-2,0)	$(-\infty,-\infty)$	$(-\infty,-\infty)$	$(-\infty,-\infty)$
3	(-3,-3)	(-3,-2)	(-3,-1)	(-3,0)	(-3,-1)	$(-\infty,-\infty)$	$(-\infty,-\infty)$

Figure 4-24: Two Player Game Payoff Matrix – More general case

Let X_1 and X_2 be the number of units that agents Φ_1 and Φ_2 move, respectively. Let us also consider $C_1(X_1)$ and $C_2(X_2)$ as the cost of the movements of Φ_1 and Φ_2, which are increasing functions of X_1 and X_2. Figure 4-24 shows the payoff matrix in case where the cost per move unit is \$1.

Let us suppose that Φ_1 knows $C_1(X_1)$ and that there exists $\hat{C}_2(X_2)$ Likewise, Φ_2 knows $C_2(X_2)$ and that there exists $\hat{C}_1(X_1)$.

Let us also consider the following utility functions:

$$U_1(X_1, X_2) = \begin{cases} -\infty & \text{if } X_1 + X_2 \prec k \\ C_1(X_1) & \text{if } X_1 + X_2 \geq k \end{cases} \qquad (4\text{-}41)$$

$$U_2(X_1, X_2) = \begin{cases} -\infty & \text{if } X_1 + X_2 \prec k \\ C_2(X_2) & \text{if } X_1 + X_2 \geq k \end{cases} \qquad (4\text{-}42)$$

In these functions k is the amount of overlap. Nash equilibrium is when $X_1 + X_2 = k$ (regardless of the outcome). We assume that the less an agent reschedules its task plan the less cost it incurs.

Nash equilibrium from agent Φ_1 point of view is where $(X_1, k - X_1)$. Therefore: $U_1 = C_1(X_1)$ and $\hat{U}_2 = \hat{C}_2(k - X_1)$. Nash equilibrium from agent Φ_2 point of view is where $(X_2, k - X_2)$, therefore: $U_2 = C_2(X_2)$ and $\hat{U}_1 = \hat{C}_1(k - X_2)$. A rational agent would think that opponent has the same cost structure, that is the more the agent reschedules its task plans (number of moves) the more it has to pay.

4.5.2.2 Collaboration Methodology

It is obvious that an agent can always delay its schedule to break up the overlap, thus remove the fault condition. But it is always in the best interest of agents to minimize rescheduling. The following theorem shows that an *n*-agent problem reduces to a two-agent problem.

Here we will drop the trivial case of when an agent can move in one direction, indefinitely. So the solution must at least include two agents. We will show that two-agent solution is sufficient.

Theorem: In case of *n* agents, a two agent solution has always the minimal cost (outcome) measured in distance units, provided that there is at least one agent in set of left-moving agents and right-moving agents.

Proof: Without loss of generality we consider a three-agent case.

Assumptions:

- Agents move to a reasonable direction – shortest path direction
- There is at least one agent which tries to move to the left and one agent which is prone to move to the right
- Without loss of generality we consider the time units to be discrete.

Definition: The *shortest path* of agent Φ_i is defined as:

$$p_i = y_i + k \qquad (4\text{-}43)$$

agent Φ_i can prevent the global overlap by moving shortest possible units p_i to the suitable direction. In this equation k is the amount of overlap and y_i is the minimum additional move required to break the overlap.

Then any displacement p_i is a solution. In other words with these assumptions we will have always a solution.

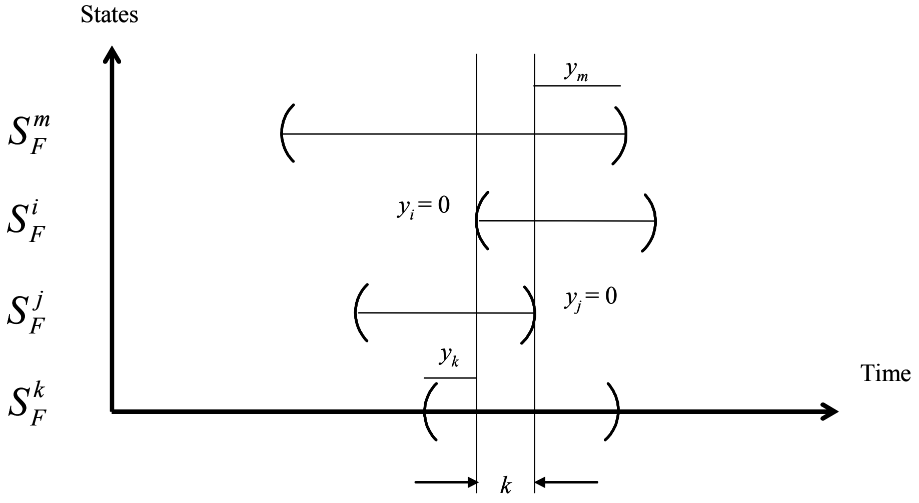

Figure 4-25: General Case

Let us define L = set of left moving agents and R = set of right moving agents. We show the total number of units that agent Φ_i moves by x_i. If agent Φ_i is a member of L then we show the movements by x_i^l and if it is a member of R then we show the movements by x_i^r.

Observations:

- Having these assumptions, it follows that any pair $\left(x_i^r, x_j^l\right)$ of displacement is a solution if: $x_i^r + x_j^l \geq p_i + p_j - k = y_i + y_j + k$ (4-44)
- A move reduces overlapping if $x_i \succ y_i$. Then $k \rightarrow k' = k + y_i - x_i$
- If Φ_i moves by x_i, then p_j does not change if Φ_j is in the same set as Φ_i (y_i changes).
- If Φ_i moves by x_i, then p_j decreases by $x_i - y_i$ if Φ_j is not in the same set as Φ_i.

Suppose $x_i^r + x_j^l + x_k^l$ is such that we have a solution. Then we have:

$p_i = y_i + k$, $p_j = y_j + k$ and $p_k = y_k + k$ (Figure 4-26).

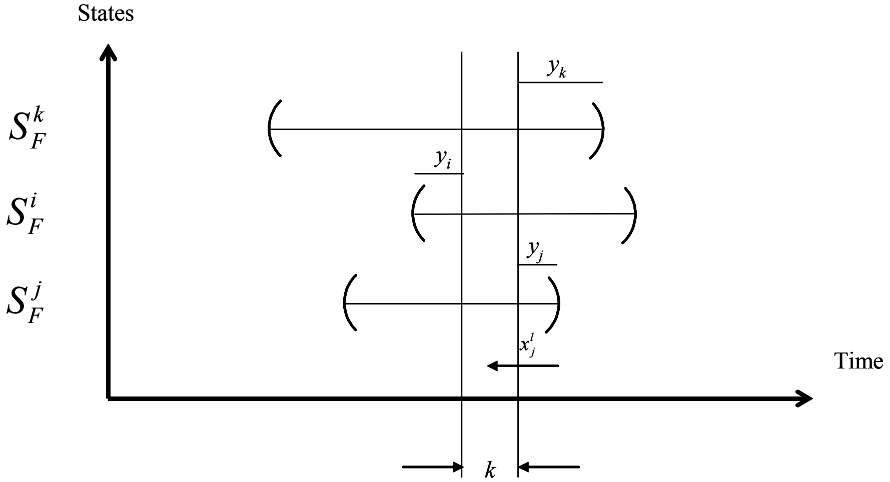

Figure 4-26: Three agent solution

Assume Φ_j moves x_j^l, which is not a solution but reduces the overlap by x_j^l ($x_j^l \succ y_j$), therefore $k' = k + y_j - x_j^l$ (Figure 4-27).

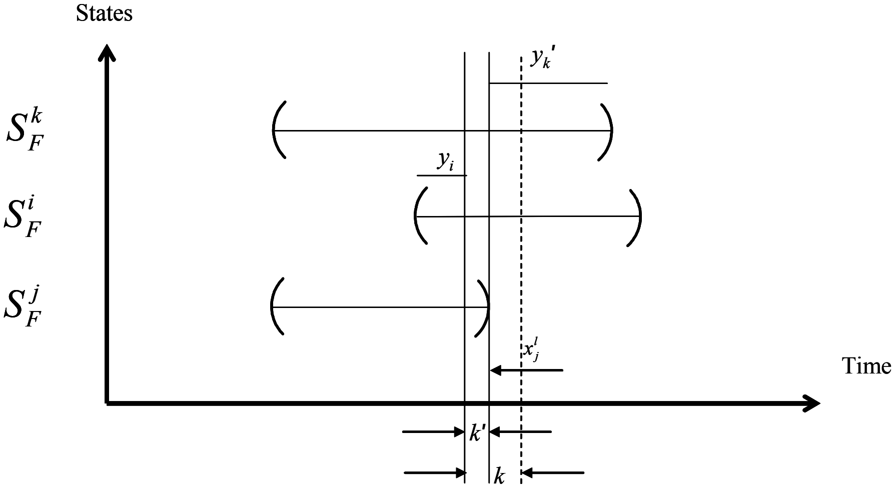

Figure 4-27: Φ_j moves x_j^l

Then based on (4-44) $x_i^r + x_k^l$ is a solution if:

$$x_i^r + x_k^l \geq p_i' + p_k' - k' \qquad (4\text{-}45)$$

But we know that

$$p_i' = y_i' + k' = y_i + (k - x_j^l + y_j) = p_i - x_j^l + y_j \qquad (4\text{-}46)$$

and

$$p_k' = y_k' + k' = y_k + (k - k') + k' = y_k + k = p_k \qquad (4\text{-}47)$$

Substituting (4-46) and (4-47) in (4-45):

$$\begin{aligned} x_i^r + x_k^l &\geq p_i' + p_k' - k' = p_k + p_i - (x_j^l - y_j) - (k + y_j - x_j^l) \\ &= y_i + y_k + 2k - x_j^l + y_j - k - y_j + x_j^l = y_i + y_k + k \end{aligned}$$

This means that x_j^l does not improve the solution cost. In other words we always have a 2 agent move solution.

How does this information affect the risk calculations?

Having this theorem, the requester agent can initiate the collaboration algorithm when needed. If all the involved parties agree on the overlap time then the collaboration will start. We know that only two agents need to move in order to avoid the risk. The left moving agents will compare themselves with the individual right moving agents. Each agent announces the best solution (If there exists any) using the method shown in figure 4-23. After collecting all the results, the requester agent will chose the best response for fault state avoidance

4.6 Conclusion

In this chapter we pointed out the issues of risk, communication and collaboration. Generally, agents try to avoid faults. These fault condition could be local or global. Global faults affect a set of agents. Here we introduce a methodology to calculate the risk of transitioning to a global fault state if the fault condition is known. Then we introduced a framework to avoid this global fault state using collaboration between involved agents. Game theory concepts have been used to expand this framework.

5. A Decision-theoretic Control Methodology

5.1 Introduction

In chapters two, three and four we discussed different situations that an agent may face in the framework defined in this dissertation. Chapter five deals with the overall formulation and explains how these different components cope with each other in a live running system in order to reach a desired state by an individual agent and with the cooperation of others.

Tasks will be announced to single agents. A task has a certain value, which will be declared to these agents. Each agent's job is to analyze the task, synthesize a solution and make a cost/benefit calculation. If there exists any reasonable solution, then it will bid for the execution of the job. During the synthesis it may need to cooperate with other agents directly or indirectly to achieve the goal. In order to satisfy this objective, in some cases, an agent must learn the model of its unknown environment as well as its own model (if not fully known). Based on these learned models it then takes appropriate decisions to reach its goals of completing the tasks it has in its task-to-do list.

There are two phases in each agent's life, namely *learning* and *execution*, as shown in figure 5-1. These two phases are quite interrelated. An agent can learn much about itself and its environment during execution of a certain task and must execute different actions while learning the model.

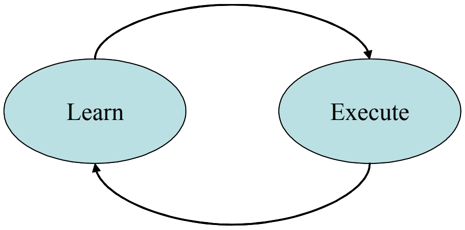

Figure 5-1: Agent's structure – simple view.

Generally speaking, the learning phase can be of two types: One type of learning is to learn the model of the agent (transition probabilities) as well as an optimal control policy (Adaptive dynamic programming approach). Another type is to find an optimal policy without learning the model (Q-learning approach). In any event, this phase must be carried out off-line because of the computation complexity involved with iterative learning stages. In our framework however, the outcome of the learning phase does not quite include a full policy. The reason is that our agents are in a dynamically changing environment and have to interact with other agents. Therefore they cannot make a final decision when they are in an offline state. Consequently the ultimate control policy will be concluded in their execution stage when they initiate another search algorithm.

5.2 Framework

Figure 5-2 summarizes the different components in our framework. The big cloud in this picture is the "environment" that includes various units including distributed agents. This cloud reflects environmental dynamics as well as static structure of the surroundings. From an individual agent's point of view some of these dynamics are known and observable and some others are unknown. The agents go through certain learning process to become familiar with these dynamics to make sequential decisions based on these findings later on.

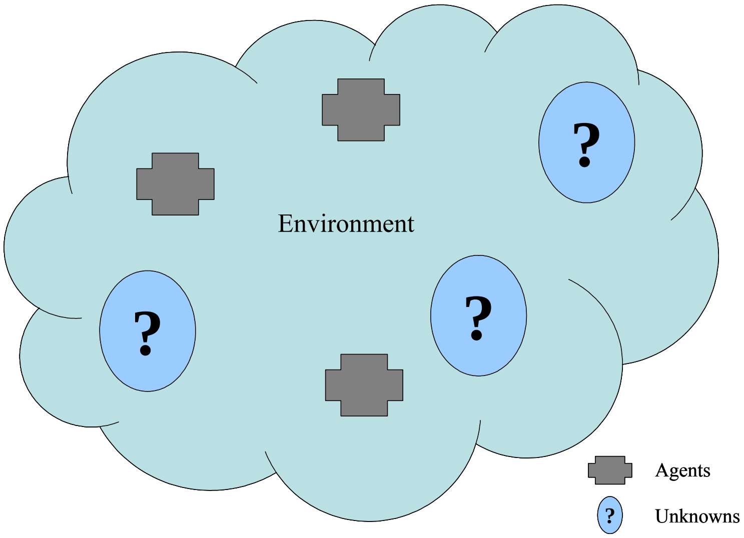

Figure 5-2: Environment

There are some areas in the environment that are initially black boxes to the agents. An agent can model these unknown areas using regular language identification methodology mentioned in chapter three. Then it can learn to control these areas (using its basic functions) if they are controllable, and then it can carry out the learned steps (sequential decisions) at the execution layer. An example that is showing this concept was introduced in chapter three of this dissertation. Another example can be a task for a walking-robot to go from room A to room C, without falling into a trap that is somewhere in room B. Assuming that the structure of room B is unknown to the agent, room B can be considered as an unknown area (black box) and falling into the trap can be assumed as a bad state. Agent wants to avoid the bad state while it passes through room B. Agent learn this process by executing trials. After this initial learning step, the agent can be plugged into the live and running environment, which will be the second phase.

5.3 Learning Stage

5.3.1 Agent's Model - Environment

As we mentioned previously, the agent first learns its environment (Figure 5-2). Learning the environment means:

- Learning the states and the transition probabilities of its own
- Locating the unknown areas of the environment, modeling and controlling them based on the specifications
- Recognizing the situations where the agent needs to calculate the risk, which will be then used at the execution stage

The process of learning the model itself is quite easy, because the environment is fully observable (except the unknown black boxes). This means we have a supervised learning task where the input is a state-action pair and the output is the resulting state. In the simplest case, we can represent the transition model as a table of probabilities. We keep track of how often each action outcome occurs and estimate the transition probability $P_{sa}(s')$ from the frequency with which s' is reached when executing a in s. This is the *maximum likelihood estimate*. A Bayesian update with a Dirichlet prior might work too. The algorithm below shows the above methodology:

Algorithm: *Model-Learning*

Input: Percept indicating the current state s'

```
Output: $P_{sa}(s')$ for all $s$, $a$ and $s'$

procedure Model_Learning (percept, set of actions $a$)
var
    N($s$,$a$): Integer /* A table of frequencies for state-action pairs, initially zero */
    N($s$,$a$, $s'$): Integer /* A table of frequencies for state-action-state triplets, initially zero */
    $t$, $s$, $s'$: State
    $a$: Action /* $s$ and $a$ are the previous state and action */
begin
    Observe the current state $s$ and random action $a$
    Do forever:
        if $s$ is not null then
                if $s$ is a pre black box state then run control_policy
                Select an action $a$ and execute it
                Observe the new state $s'$
                increment N($s$,$a$) and N($s$,$a$, $s'$)
                for each $t$ such that N($s$, $a$, $t$) <> 0
                    $P_{sa}(t)$ = N($s$, $a$, $t$) / N($s$,$a$)
                end for
        end if
    end Do
end
```

When running the algorithm, the agent may face one special case when executing the line: “Observe the new state s',” namely when an agent faces a black box. Figure 5-3 shows how the state space looks like when the agent comes into these types of situations. As we will see later on in this chapter, this special case can be considered as a virtual transition in the global search with only one arc, but with a modified reward (cost) function.

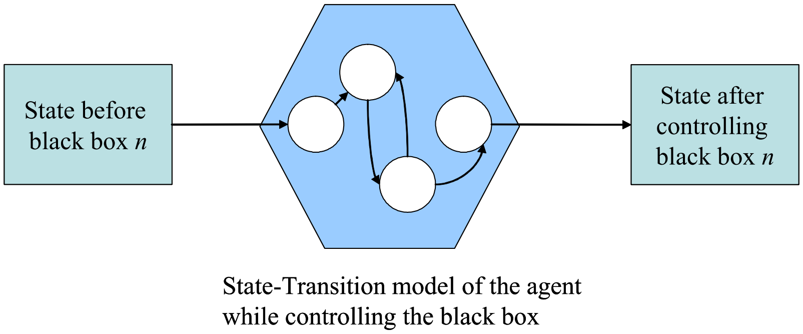
State before black box n
State after controlling black box n
State-Transition model of the agent while controlling the black box

Figure 5-3: Transition model of the agent when facing a black box

5.4 Execution Stage

After the agent learns a preliminary model of the system, it will be time to execute the task. Because of some of the properties of the system, the agent needs to make the final synthesis at the start of this stage. In order to understand our approach, we first introduce three different types of nodes in the global search tree. There are three different types of nodes that the agent is facing:

- Regular Nodes
- Black Box Nodes
- Risk Analysis Nodes

During the search the agent is usually in a regular node, where it applies one of its basic functions, and receives a fixed reward from the environment. It may also be in one of the black box nodes where it must control its uncontrolled environment by applying the control solution that it has learned during the learning stage. Finally it may be in a risk analysis node. These nodes are the ones that lead to a potential global fault state situation. Figure 5-4 shows a sample search tree.

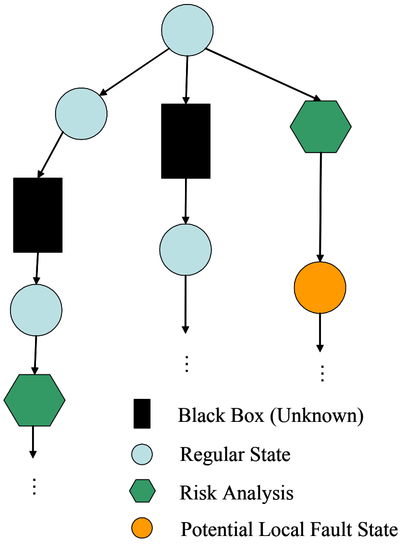

Figure 5-4: Execution Search Tree

5.4.1 Regular Nodes

Now we formulate the reward function and transition out of these nodes to complete our Markov Decision Process model. When applying a basic function *a* in state *s*, the agent will make an immediate transition to *s′* and will receive reward *r*(*s*,*a*). If state *s* is a regular node we then define the reward as:

$$r(s,a) = -c(a) \quad \text{if } s \text{ is a regular node} \qquad (5\text{-}1)$$

Where *c*(*a*) is the cost of using action *a*.

5.4.2 Black Box Nodes

We saw in figure 5-3 that executing a set of control actions in the black box to reach the desired objective can be combined into "one" virtual transition – in and out of the black box. Relevant to the global search is the total cost of these actions, which can be described as below:

$$r(s,a) = \left(- \sum_{a_j \text{in black box}} c(a_j) \right) + K \quad \text{if } s \text{ is a black box node and the outcome is successful} \quad (5\text{-}2)$$

where *K* is a constant positive reward that the agent receives when the job in the black box is performed successfully.

$$r(s,a) = \left(- \sum_{a_j \text{in black box}} c(a_j) \right) - K' \quad \text{if } s \text{ is a black box node and the outcome is not successful} \quad (5\text{-}3)$$

where *K′* is a constant negative reward that the agent receives when the job in the black box is performed unsuccessfully. We notice that usually $K' \succ\succ K$.

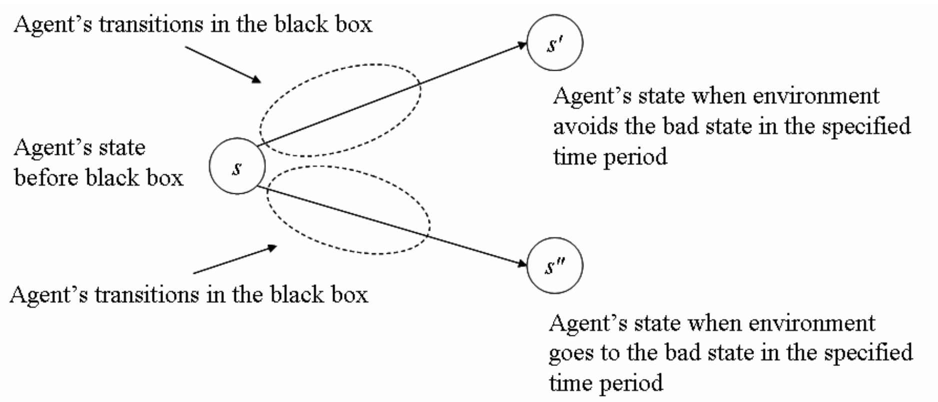
Agent's transitions in the black box
s'
Agent's state when environment avoids the bad state in the specified time period
Agent's state before black box
s
s''
Agent's transitions in the black box
Agent's state when environment goes to the bad state in the specified time period

Figure 5-5: Agent's transitions during interaction with a black box

Figure 5-5 shows how transitions from a pre-black box state look like if we don't consider the detailed state transition model of an agent during the interaction with a black box. From this point of view, only two issues must be addressed in the global search. One is whether the outcome of the control policy was successful and other one is the state of agent when the interaction with the environment is complete.

5.4.3 Risk Analysis Nodes

In chapter 4 we comprehensively discussed how an agent can calculate the risk of transitioning to a fault state and how it can reduce this risk using collaboration. Here in we will use this information in agent's global search.

First we need to calculate the amount of reward an agent gains when executing an action in its pre-bad (fault) state. This state may or may not lead the agent into a global fault state, depending on the other agents. In general, we have:

$$r(s,a) = -c(a) - c(RC) \quad \text{if } s \text{ is a pre-risk analysis node} \qquad (5\text{-}4)$$

In this equation $c(RC)$ is the cost of risk analysis and it may have different values according to different situations:

- if the initial risk analysis shows that the risk is not too high

$$\begin{aligned} C(RC) &= -c_1(RC) \\ &= -c_1(\text{number of agents in (4-1) and number of local states in each perception model}) \end{aligned} \tag{5-5}$$

- if the risk is too high, the communication will be initiated and if communication alone shows a low risk based on new information then:

$$\begin{aligned} C(RC) &= -c_1(RC) - c_2(RC) \\ &= -c_1(RC) - c_2(\textit{number of communication packages}) \end{aligned} \tag{5-6}$$

- if the risk is too high, and communication alone does not reduce it very much and collaboration is required, then:

$$\begin{aligned} C(RC) &= -c_1(RC) - c_2(RC) - c_3(RC) - c_4(RC) = -c_1(RC) - c_2(RC) \\ &- c_3(\textit{additional communication packages}) - c_4(\textit{number of moves in rescheduling}) \end{aligned} \tag{5-7}$$

- if the risk is too high, and collaboration cannot help reducing it (or no possibility of collaboration), then:

$$\begin{aligned} C(RC) &= -c_1(RC) - c_2(RC) - c_3(RC) - c_5(RC) = -c_1(RC) - c_2(RC) - c_3(RC) \\ &- c_5(\text{potential global fault state}) \end{aligned} \tag{5-8}$$

We notice that usually $c_5(RC) \succ\succ c_1(RC), c_2(RC), c_3(RC), c_4(RC)$.

At this point we need to see how these rewards affect the transition functions. Figure 5-6 shows different transitions of the agent after doing the risk analysis. As we see, in three cases the agent makes a transition to a safe state and in the last one, it makes a transition to a global fault state.

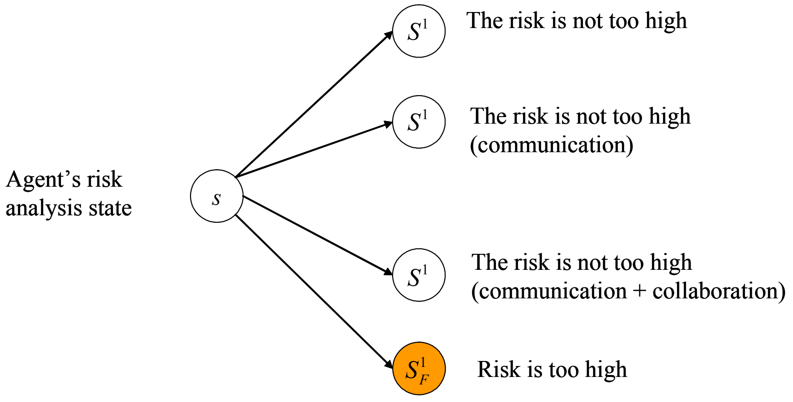
Agent's risk analysis state
s
S^1
The risk is not too high
S^1
The risk is not too high (communication)
S^1
The risk is not too high (communication + collaboration)
S^1_F
Risk is too high

Figure 5-6: Agent's transitions after risk analysis

5.5 Global Search

After introducing different state types in the global search tree, we are now ready to introduce the global search methodology of the agent, which is built into a Markov Decision Process model.

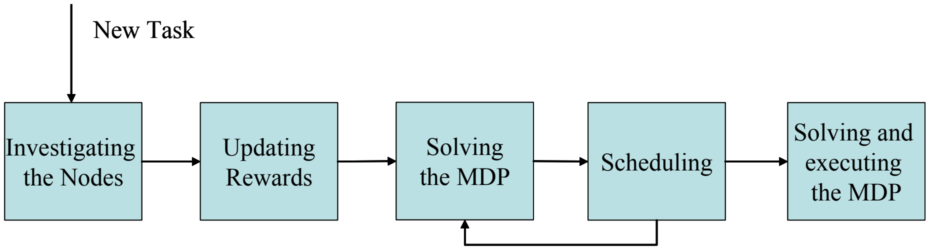

Figure 5-7: Steps in Execution Stage

An agent goes through 5 different steps while at the execution stage. After receiving a task request it first investigates the nodes learned at the learning stage – this step is to identify the special nodes (black box and risk analysis nodes). In the next step it updates the rewards using communication, and collaboration and current states of the other agents based on the equations (5-1) through (5-8). Only then the agent would be ready to solve the corresponding Bellman equations (3-3) – (3-5). Following finding a solution, the agent would schedule the new steps and will execute them accordingly. These steps are shown in figure 5-7.

5.5.1 MDP Model

We show the value of a task T by $V(T)$. If the agent reaches the objective goal it will receive $V(T)$ as reward. In this phase, the objective of the agent is to synthesize a path from its current state to the desired state (current job) with a reasonable cost. We use an MDP model to solve this problem. Having the full perception model $P_{sa}(s')$ and the corresponding rewards $r(s,a)$ we can solve the Bellman equations and find the optimal policy.

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s') \qquad (5\text{-}9)$$

Below you can find the algorithm that does the search using the modified policy iteration method.

Algorithm: *Global Search*

Input: Initial MDP, set of global fault states based on (4-1), set of black box locations and times of minimum interactions with them, set of basic functions

Output: Policy π

```
procedure Global_Search(Initial model MDP, set of actions $a$)
begin
    Identify the black box nodes
    Update related rewards according to (5-2) and (5-3)
    Identify the risk analysis nodes
    for each
        Update related rewards based on (5-3) through (5-8)
    end for
    $\pi$ = modified_policy_iteration_algorithm [Russell 03]
    return optimal policy $\pi$
end
```

5.6 Conclusion

In this chapter we used the results of chapter two to four to build a framework that can be embedded into a single agent. An agent learns its own model and environment's model in an offline state. Then it can be plugged into a live system to interact with others and to perform specific tasks. While in this system, it can update the learned model through collection of information and make more accurate and reliable decisions.

6. Conclusions and Future Work

6.1 Conclusion

In chapter two we presented our preliminary results on control synthesis algorithm for agents which are selfish. We assumed that the agent's environment is deterministic and fully observable. The agent is not subject to failures, but it is possible to reach to fault conditions, specifically deadlocks. Two search algorithms were presented and it was also shown how to embed deadlock detection and avoidance in this algorithms.

In chapter three we introduced an algorithm that utilizes the ideas from reinforcement learning to exert control on an unknown environment. This algorithm identifies a perception model of the environment at each level using regular language theory and adapts its search to the newly identified models. The search algorithm is goal oriented. The agent examines its basic functions around the bad state to save the maximally permissive property of the process and to reduce the computational cost. In this method we showed that even a process that is continuous in time can be controlled using a discrete event control theoretic framework.

In chapter four we pointed out the issues of risk, communication and collaboration. Generally, agents try to avoid faults. These fault conditions could be local or global. Global faults affect a set of agents. Here we introduce a methodology to calculate the risk of transitioning to a global fault state if the fault condition is known. Then we introduced a framework to avoid this global fault state using collaboration between involved agents. Game theory concepts have been used to expand this framework.

In chapter five we used the results of chapter two to four to build a framework that can be embedded into a single agent. An agent learns its own model and environment's model in an offline state. Then it can be plugged into a live system to interact with others and to perform specific tasks. While in this system, it can update the learned model through collection of information as well as collaboration and make more accurate and reliable decisions.

6.2 Future Work

1 - The approximate risk calculation model presented in chapter 4 provides a fundamental framework whereby an agent can calculate its risk of encountering global conflicts or faults conditions, through communication and collaboration with the other agents. At this time, this methodology has some shortcomings in terms of complexity of calculations and accuracy of the results. We believe that extension of this methodology and its simplification will be a major step.

2 - In chapter 3 we presented a control synthesis model which acts upon an unknown environmental black box for the purpose of avoiding some adverse conditions or creating desirable ones. For such a methodology to be fully useful in real applications, it is essential to further extend the underlying learning schemes and simulations. It is also

essential to include cost more directly into the underlying search to reduce the computational complexity.

3 - The results in chapter 2, where controller reconfigures itself to deal with previously unknown conditions can be extended to different situations. Such an embedded intelligence will be essential for real life applications.

4 - The cost models that we presented in Chapter 5 are fairly simple. We believe these can be extended further including a more formalized model for learning.

Appendix

Algorithm for recognizing the stochastic regular language [Carrasco 94]:

```
algorithm ALEGRIA
input:
    S: sample set of strings
    α: 1-confidence level
ouput:
    DSFA
begin:
    A = Stochastic Prefix Tree Acceptor (SPTA) from S
    do (for j = successor(firstnode(A)) to lastnode(A))
        do (for i = firstnode(A) to j)
            if compatible(i, j)
                determinize(A,i) // making sure all descendents have at most one outgoing transition corresponding to a given symbol
                exit (i-loop)
            end if
        end do
    end do
    return A
end algorithm
```

```
algorithm different
input:
    n, n′: number of strings arriving at each node
    f, f′: number of strings ending/following a given arc
ouput:
    boolean
begin:
    return $\left|\frac{f}{n}-\frac{f'}{n'}\right| \succ \sqrt{\frac{1}{2}\ln\frac{2}{\alpha}}\left(\frac{1}{\sqrt{n}}+\frac{1}{\sqrt{n'}}\right)$
end algorithm
```

```
algorithm compatible
input:
    i, j: nodes
ouput:
    boolean
begin:
    if different($n_i$, $f_i(\#)$, $n_j$, $f_j(\#)$)
        return false
    end if
    do (for $\forall a \in A$)
        if different($n_i$, $f_i(a)$, $n_j$, $f_j(a)$)
            return false
        end if
        if not compatible($\delta(i,a)$, $\delta(j,a)$)
            return false
        end if
    end do
    merge ($A$, $i$, $j$)
    return $A$
end algorithm
```

```
algorithm determinize
input:
        $A$: finite automaton
        $i$: node in $A$, possibly with duplicate arcs
output:
        equivalent deterministic finite automaton
begin
        do (for $\forall a \in A$)
            if (node $i$ has two transitions $\delta_1(i,a)$, $\delta_2(i,a)$ on $a$)
                merge ($A$, $\delta_1(i,a)$, $\delta_2(i,a)$)
                determinize($A$, $\delta_1(i,a)$)
            end if
        end do
end algorithm
```

References

[AAAI 05] http://www.aaai.org/AITopics/, AI Topics**,** a dynamic library of introductory information about Artificial Intelligence, 2005.

[Alpan 97] G. Alpan, Design and Analysis of Supervisory Controllers for Discrete Event Systems, Ph.D. Dissertation, Rutgers University, New Brunswick, New Jersey, 1996.

[Arrow 63] K. Arrow, Social Choice and Individual Values, 2nd edition, New Haven: Cowles Foundation, 1963.

[ASAP 05] http://www.asap.cs.nott.ac.uk/themes/ma.shtml, ASAP, the Automated Scheduling, Optimization and Planning group, School of Computer Science and Information Technology, University of Nottingham, 2005.

[Baase 90] S. Baase, Computer Algorithms: Introduction to Design and Analysis, Addison-Wesley Publishing Company, 2nd Edition, 1990.

[Bernstein 00] D.S. Bernstein, S. Zilberstein, and N. Immerman, The complexity of decentralized control of Markov decision processes, *Proceedings of the 16th International Conference on Uncertainty in Artificial Intelligence*, Stanford, CA, 2000.

[Bhat 84] U.N. Bhat, Elements of applied stochastic processes, John Wiley & Sons, 1984.

[Bigus 01] J.P. Bigus, and J. Bigus, Constructing Intelligent Agents Using Java, Second Edition. Professional Developer's Guide Series, Wiley, 2001.

[Bonasso 96] IL R. Bonasso, D. Kortenkamp, D.P. Miller, and M. Slack, Experiences with an Architecture for Intelligent, Reactive Agents, *Intelligent Agents II,* eds. M. Wooldridge., J.P. Muller, and M. Tambe, pp. 187-202, Lecture Notes In Artificial Intelligence 1037, New York Springer-Verlag, 1996.

[Bond 88] A.H. Bond, and Gasser, I., *Readings in Distributed Artificial Intelligence.* San Francisco, Calif.: Morgan Kaufmann, 1988.

[Bratman 87] M.E. Bratman, Intensions, Plans, and Practical Reasons, Cambridge, MA: Harvard University Press, 1987.

[Brooks 86] R.A. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*, Vol. 2, pp. 14-23, 1986.

[Carrasco 94] R.C. Carrasco, and J. Oncina, Learning stochastic regular grammars by means of a state merging method, *Proceedings of the 2nd International Colloquium on Grammatical Inference and Applications*, ICGI, pp. 139-152, 1994.

[Cassady 67] R. Cassady, Auctions and Auctioneering, Berkeley, University of California Press, 1967.

[Chaib-draa 92] B. Chaib-draa, B. Moulin, R. Mandiau, and P. Millot, Trends in Distributed Artificial Intelligence. *Artificial Intelligence Review*, Vol. 6, No. 1, pp. 35-66, 1992.

[Chandy 82] K.M. Chandy, and J. Misra, A Distributed Algorithm for Detecting Resource Deadlocks in Distributed Systems, *Proceedings of ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Ottawa, Ontario, Canada, pp. 157-164, August 1982.

[Chandy 83] K.M. Chandy, J. Misra, and L.M. Haas, Distributed Deadlock Detection, *ACM Transactions on Computer Systems*, Vol. 1, No. 2, pp. 144-156, May 1983.

[Coffman 71] E.G. Coffman, M.J. Elphick, and A. Shoshani, System deadlocks, *ACM Computing Surveys*, Vol. 3, pp. 67-78, 1971.

[Corkill 83] D. Corkill, and V. Lesser, The Use of Metalevel Control for Coordination in a Distributed Problem- Solving Network., *Proceedings of the Eight International Joint Conference on Artificial Intelligence (UCAI- 83)*, pp. 767-770, Menlo Park. Calif.: International Joint Conferences on Artificial Intelligence, 1983.

[Darabi 00] H. Darabi, Toward Fault Tolerant and Re-configurable Discrete Event Systems, Ph.D. Dissertation, Rutgers University, New Brunswick, New Jersey, 2000.

[Davis 83] R. Davis, and R.G. Smith, Negotiation as a Metaphor for Distributed Problem Solving, *Artificial Intelligence*, Vol. 20, No. 1, pp. 63-100, 1983.

[Debouk 00] R. Debouk, Failure diagnosis of decentralized discrete-event systems, *Ph.D. thesis*, University of Michigan, Ann Arbor, 2000.

[Decker 97] K. Decker, K. Sycara, and M. Williamson, Middle Agents for the Internet, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97*), pp. 578-583, Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence, 1997.

[Dent 92] I. Dent, J. Boticario, J. McDermott, T. Mitchell, and D. Zabowski, A Personal Learning Apprentice. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 96-103, Menlo Park. Calif.: American Association for Artificial Intelligence, 1992.

[Durfee 89a] E.H. Durfee, and V. Lesser, Negotiating Task Decomposition and Allocation Using Partial Global Planning, *Distributed Artificial Intelligence,* Vol. 2, eds. L Gasser and M. Huhns., pp. 229-244, San Francisco, Calif.: Morgan Kaufmann, 1989.

[Durfee 89b] E. Durfee, V. Lesser, and D. Corkill, Cooperative distributed problem solving, *The Handbook of Artificial Intelligence*, Vol. IV, A. Barr, P. Cohen, E. Feigenbaum, eds., Addison Wesley, pp. 83-167, 1989.

[Fanti 04] M.P. Fanti, and M. Zhou, Deadlock Control Methods in Automated Manufacturing Systems, *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans*, Vol. 34, No. 1, January 2004.

[Feller 50] W. Feller, An introduction to probability theory and its applications, John Wiley & Sons, New York, 1950.

[Ferber 96] J. Ferber, Reactive Distributed Artificial Intelligence: Principles and Applications, *Foundations of Distributed Artificial Intelligence,* eds. G. O'Hare and N. Jennings, pp. 287-314, New York Wiley, 1996.

[Ferber 99] J. Ferber, Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Harlow, UK: Addison Wesley Longman, 1999.

[Ferguson 95] I.A. Ferguson, Integrated Control and Coordinated Behavior: A Case for Agent Models, *Intelligent Agents: Theories, Architectures, and Languages,* eds. M. Wooldridge and N.K. Jennings, pp. 203-218, Lecture Notes in Artificial Intelligence, V. 890, New York, Springer-Verlag, 1995.

[FIPA 05] http://www.fipa.org, Foundation for Intelligent Physical Agents.

[Franklin 96] S. Franklin, and A. Graesser, Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents, *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer Verlag, 1996.

[Garrido 96] L. Garrido, and K. Sycara, Multiagent Meeting Scheduling: Preliminary Experimental Results, *Proceedings of the Second International Conference on Multiagent Systems,* pp. 95-102, Menlo Park, Calif.: *AAAI* Press, 1996.

[Gasser 92] L. Gasser, An Overview of DAI, *Distributed Artificial Intelligence Theory and Praxis,* eds. N. Avouris and L. Gasser, pp. 9-30, Boston, Kluwer Academic, 1992.

[Giua 92] A. Giua, Petri Nets as Discrete Event Models for Supervisory Control, Ph.D. Thesis, Dept. of Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, N.Y., July 1992.

[Gruschke 98] B. Gruschke, Integrated Event Management: Event Correlation Using Dependency Graphs, Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, 1998.

[Han 03] W. Han, and M.A. Jafari, Component and Agent-Based FMS Modeling and Controller Synthesis, *IEEE Transactions On Systems, Man, And Cybernetics —Part C: Applications And Reviews*, Vol. 33, No. 2, May 2003.

[Hart 68] P.E. Hart, N.J. Nilsson, B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions on Systems Science and Cybernetics,*Vol. 4, No. 2, pp. 100-107, 1968.

[Higuera 98] C. De la Higuera, Learning stochastic finite automata from experts, Proceedings of the 4th International Colloquium on Grammatical Inference and Applications, ICGI, pp. 79-89, 1998.

[Hoeffding 63] W. Hoeffding, Probability inequalities for sums of bounded random variables, *American Statistical Association Journal*, Vol. 53, pp. 13-30, 1963.

[Howarth 04] F. Howarth, http://www.it-director.com/article.php?articleid=11774, March 2004.

[Huberman 88] B.A. Huberman, and T. Hogg, The Behavior of Computational Ecologies, *The Ecology of Computation.* ed. B.A**.** Huberman, Amsterdam, The Netherlands: North-Holland, 1988.

[Huhns 97] M. Huhns, and M. Singh, *Readings in Agents.* San Francisco, Calif.: Morgan Kaufmann, 1997.

[Jafari 95] M.A. Jafari, Supervisory Control Specification and Synthesis, *Petri Nets in Flexible and Agile Manufacturing*, Ed. M.C. Zhou, Kluwer Academic Publisher, pp. 337-368, 1995.

[Jennings 98] N. Jennings, K. Sycara, and M. Wooldridge, A Roadmap for Agent Research and Development, *Autonomous Agents and Multiagent Systems*, 1998.

[Knapp 87] E. Knapp, Deadlock Detection in Distributed Databases, *ACM Computing Surveys*, Vol. 19, No. 4, pp. 303-328, December 1987.

[Karp 02] M. Karp, Network World Storage Newsletter, 10/09/02; http://www.research.ibm.com/autonomic/index.html, October, 2002.

[Klemperer 99] P. Klemperer, Auction theory: a guide to the literature, *Journal of Economic Surveys*, Vol.13, No. 3, pp. 227-286,1999.

[Krogh 96] B.H. Krogh, and S. Kowalewski, State Feedback Control of Condition/Event Systems, *Mathematical and Computer Modeling*, Vol. 23, No. 11/12, pp. 161-173, 1996.

[Krothapalli 99] N. Krothapalli, and A. Deshmukh, Design of negotiation protocols for multi-agent manufacturing systems, *International Journal of Production Research*, Vol. 37, No. 7, pp. 1601-1624, 1999.

[Lee 93] Y. K. Lee, S.J. Park, OPNets: An Object-Oriented High-Level Petri Net Model for Real-Time System Modeling, *Journal of Systems and Software,* Vol. 20, No. 1, pp. 69-86, January, 1993.

[Lin 94] F. Lin, Diagnosability of discrete-event systems and its applications, *Journal of Discrete Event Dynamic Systems: Theory and Applications*, Vol. 4, No. 2, pp. 197–212, May 1994.

[Lopomo 98] G. Lopomo, The English Auction is optimal among simple sequential auctions, *Journal of Economic Theory*, No. 82, pp. 144-166, 1998.

[Maes 95] P. Maes, Artificial Life Meets Entertainment: Life like Autonomous Agents, *Communications of the ACM*, Vol. 38, No. 11, pp. 108-114, 1995.

[McAfee 87] R.P. McAfee, and J. McMillan, Auctions and bidding, *Journal of Economic Literature*, No. 25, pp. 699-738, 1987.

[Mitchell 84] D.P. Mitchell, and M.J. Merritt, A Distributed Algorithm for Deadlock Detection and Resolution, *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing, ACM SIGACT SIGOPS*, Vancouver, B.C., Canada, pp. 282-284, August 1984.

[Mitchell 97] T.M. Mitchell, Machine Learning, McGraw Hill, 1997.

[Mocanu 06] A Simple Laplace Transform Calculus, July 21, 2006.

[Muller 94] J.P. Muller, and M. Pischel, Modeling, Interacting Agents in Dynamic Environments, *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, pp. 709-713, Chichester, UK., Wiley, 1994.

[Neuts 81] M.F. Neuts, Matrix-Geometric Solutions in Stochastic Models – An Algorithmic Approach, Baltimore, Maryland, 1981.

[Obermarck 82] R. Obermarck, Distributed Deadlock Detection Algorithm, *ACM Transactions on Database Systems*, Vol. 7, No. 2, pp. 187-208, June 1982.

[O'Hare 96] G. O'Hare, and N. Jennings, *Foundations of Distributed Artificial Intelligence*, New York: Wiley, 1996.

[Oncina 92] J. Oncina, and P. García, Inferring regular languages in polynomial updated time, *Pattern Recognition and Image Analysis*, Series in Machine Perception and Artificial Intelligence, Vol. 1, pp. 49-61, World Scientific, Singapore, 1992.

[Ostroff 89a] J.S. Ostroff, Synthesis of controllers for real-time discrete event systems, *IEEE Proceedings of the 28th Conference on Decision and Control*, Tampa, Florida, Dec. 1989.

[Ostroff 89b] J.S. Ostroff, *Temporal Logic for Real-Time Systems*, Advanced Software Development Series. Research Studies Press Limited (distributed by John Wiley and Sons), England, 1989.

[Patterson 02] D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kıcıman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies, *Computer Science Technical Report*, UCB//CSD-02-1175, U.C. Berkeley, March, 2002.

[Petri 62] C.A. Petri, Communication with Automata, Ph.D. Dissertation, University of Bonn, Bonn, Germany, 1962.

[Pinzon 99] L.E. Pinzon, H.-M. Hanisch, M.A. Jafari, and T. O. Boucher, A Comparative Study of Synthesis Methods for Discrete Event Controllers, *Journal of Formal Methods in System Design*, Vol. 15, No. 2, pp. 123-167, September 1999.

[Pinzon 01] L.E. Pinzon, Developing Sequential Controllers for Discrete Event Systems, Ph.D. Dissertation, Rutgers University, New Brunswick, New Jersey, 2001.

[Ramadge 87a] P.J. Ramadge, and W.M. Wonham, Supervisory control of a class of discrete-event processes, *SIAM Journal of Control and Optimization*, Vol. 25, No. 1, pp. 206-230, January, 1987.

[Ramadge 87b] P.J. Ramadge, and W.M. Wonham, Modular feedback logic for discrete event systems, *SIAM Journal of Control and Optimization*, Vol. 25, No. 5, pp. 1202-1218, September, 1987.

[Rausch 95] M. Rausch, and H.M. Hanisch, Net condition/event systems with multiple condition outputs, *ETFA 95 Conference*, Paris, France, October, 1995.

[Ross 96] S. Ross, Stochastic Processes, John Wiley, 2nd edition, 1996.

[Russell 03] S.J. Russell, and P. Norvig, Artificial Intelligence: A Modern Approach, Englewood Cliffs, NJ: Prentice Hall, 2nd edition, 2003.

[Sampath 95] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis, Diagnosability of discrete-event systems, *IEEE Transactions on Automatic Control*, Vol. 40, pp. 1555–1575, September, 1995.

[Sampath 96] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis, *IEEE Transactions on Control Systems Technology*, Vol. 4, No. 2, March 1996.

[Sampth 01] M. Sampath, A hybrid approach to failure diagnosis of industrial systems, *Proceedings of the American Control Conference*, Arlington, VA, pp. 25-27, June, 2001.

[Sandholm 93] T. Sandholm, An implementation of the contract net protocol based on marginal cost calculations, *Proceedings of the National Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA, pp. 256-262, 1993.

[Schwartz 97] R. Schwartz, and S. Kraus, Bidding mechanisms for data allocation in multi-agent environments, *Agent Theories, Architectures, and Languages*, Springer, New York, pp. 61-75, 1997.

[Srinivasan 93] V.S. Srinivasan, and M.A. Jafari, Fault Detection/Monitoring Using Time Petri Nets, *IEEE Transactions On Systems, Man, And Cybernetics*, Vol. 23, No. 4, July/August, 1993.

[Stone 00] P. Stone, and M. Veloso, Multiagent systems: a survey from a machine learning perspective, *Autonomous Robots*, Vol. 8, No. 3, 2000.

[Sutton 98] R.S. Sutton, and A.G. Barto, Reinforcement Learning: An Introduction, Cambridge, MA, MIT Press, 1998.

[Sycara 98] K. Sycara, Multiagent Systems, American Association for Artificial Intelligence, pp. 79-92 , Summer 1998.

[Sycara 03] K. Sycara, J.A. Giampapa, B.K. Langley, and M. Paolucci, The RETSINA MAS, a Case Study, *Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications*, Alessandro Garcia, Carlos Lucena, Franco Zambonelli, Andrea Omici, Jaelson Castro, ed., Springer-Verlag, Berlin Heidelberg, Vol. LNCS 2603, pp. 232-250, July, 2003.

[Tai 02] T. Tai, and T.O. Boucher, An Architecture for Scheduling and Control in Flexible Manufacturing Systems using Distributed Objects, *IEEE Transactions on Robotics and Automation*, Vol.18, No. 4, pp.452-462, 2002.

[Thomas 98] J. Thomas, and K. Sycara, Stability and Heterogeneity In Multiagent Systems, *Proceedings of the Third International Conference on Multiagent Systems*, Menlo Park, Calif: AAAI Press, 1998.

[Vickrey 61] W. Vickrey, Counterspeculation, auctions, and competitive sealed tenders, *Journal of Finance*, No.16, pp. 8-37, 1961.

[Viswanadham 90] 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*, Vol. 6, pp. 713-723, December 1990.

[Vlassis 03] N. Vlassis, A Concise Introduction to Multiagent Systems and Distributed AI, Introductory Text, University of Amsterdam, September 2003.

[Walsh 98] W. Walsh, M. Wellman, P. Wurman, and J. MacKie-Mason, Auction protocols for decentralized scheduling, *Proceedings of the 18th International Conference on Distributed Computing Systems*, Amsterdam, The Netherlands, pp. 612-621, 1998.

[Wang 98] L.C. Wang, S.Y. Wu, Modeling with colored timed object-oriented Petri nets for automated manufacturing systems, *Computers and Industrial Engineering,* Vol. 34, No. 2, pages 463-480. 1998.

[Wang 03] X. Wang, and T. Sandholm, Reinforcement learning to play an optimal Nash equilibrium in team Markov games, *Advances in Neural Information Processing Systems*, 15, Cambridge, MA, MIT Press, 2003.

[Want 03] R. Want, T. Pering, D. Tennenhouse, Comparing autonomic and proactive computing, *IBM Systems Journal*, Vol. 42, No. 1, p129, 2003.

[Weiβ 95] G. Weiß, Distributed reinforcement learning, *Robotics and Autonomous Systems*, No. 15, pp. 135–142, 1995.

[Wellman 93] M. Wellman, A market-oriented programming environment and its application to distributed multicommodity flow problems, *Journal of Artificial Intelligence Research*, No. 1, pp. 1-22, 1993.

[Wellman 94] M. Wellman, A computational market model for distributed configuration design, *Proceedings of 12th National Conference on Artificial Intelligence*, AAAI-94, Seattle, WA, pp. 401-407, 1994.

[Wooldridge 95] M. Wooldridge, and N. Jennings Intelligent Agents: Theory and Practice, *Knowledge Engineering Review*, Vol. 10, No. 2, pp. 115-152, 1995.

[Yamalidou 96] K. Yamalidou, J. Moody, M. Lemmon, and P. Antsaklis, Feedback control of Petri Nets based on place invariants, *Automatica,* Vol. 32, No. 1, pp. 15–28, 1996.

[Ygge 96] F. Ygge, and H. Akkermans, Power load management as a computational market, *Proceedings of the 2nd International Conference on Multi-Agent Systems*, AAI Press, Menlo Park, CA, pp. 393-400, 1996.

[Zad 99] S.H. Zad, R.H. Kwong, and W.M.Wonham, Fault diagnosis in timed discrete-event systems, *Proceedings of the 38th IEEE Conference on Decision Control*, Phoenix, AZ, pp. 1756–1761, December, 1999.

[Zhao 06] P. Zhao, Distributed System with Self-diagnosis and Self-Healing, Ph.D. Dissertation, New Brunswick, New Jersey, 2006.

Curriculum Vitae

Ardavan Amini

Education

1992-1996	B.S. in Electrical Engineering, University of Tehran, Faculty of Engineering, Tehran, Iran.
2000-2003	M.S. in Industrial and Systems Engineering, Rutgers University, New Brunswick, New Jersey.
2003-2007	Ph.D. in Industrial and Systems Engineering, Rutgers University, New Brunswick, New Jersey.

Experience

1997-1999	Electrical Engineer, Bisan Pars Co. Ltd. (Rep. of HORIBA Europe, Germany), Tehran, Iran.
2000-2002	Research Assistant, Department of Industrial and Systems Engineering, Rutgers University, New Brunswick, New Jersey.
2002-2004	Teaching Assistant, Department of Industrial and Systems Engineering, Rutgers University, New Brunswick, New Jersey.
2004-2007	Research Assistant, Center for Advanced Infrastructure and Transportation (CAIT), Rutgers University, New Brunswick, New Jersey.

Publications

2002	“A Distributed Discrete Event Dynamic Model for Supply Chain of Business Enterprises,” M.A. Jafari, et al., IEEE Computer Society, Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES’02), pp. 279-285.
2005	“Control Synthesis for Reconfigurable Distributed Systems with Applications in Manufacturing,” A. Amini, et al., 16th IFAC World Congress Proceedings, Prague, Czech Republic.

2005	“Modeling Admissible Behavior using Event Signals,” L.E. Pinzon, et al., Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain.
2007	“An Online Fault Detection and Avoidance Framework for Distributed Systems,” P. Zhao, et al., Proceedings of the 2007 IFAC Workshop on Dependable Control of Discrete Systems, Paris, France.