Winter 2015

Calculus for decision systems

Jorge Antonio Samayoa Ranero
Purdue University

Graduate School Form 30
Updated 1/15/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Jorge Antonio Samayoa Ranero

Entitled
CALCULUS FOR DECISION SYSTEMS

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Abhijit Deshmukh
Chair

Omid Nohadani

Suresh Jagannathan

William Crossley

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Abhijit Deshmukh

Approved by: Abhijit Deshmukh 1/8/2015
Head of the Departmental Graduate Program Date

CALCULUS FOR DECISION SYSTEMS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Jorge Antonio Samayoa Ranero

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2015

Purdue University

West Lafayette, Indiana

To my wife *Guisela*

and daughters *Ana Victoria* and *Sabrina*

ACKNOWLEDGMENTS

The work presented in this manuscript has been influenced by many great minds and friends who provided me with the strength, advice and courage to complete it.

I specially thank my advisor, Dr. Abhijit Deshmukh, for his guidance, support and –more importantly– the time he spent listening to my often stubborn and dreamy ideas. I highly appreciate the enthusiasm and joy he injected me towards doing research.

Also, I would like to thank the members of my advisory committee: Dr. Omid Nohadani, Dr. Suresh Jagannathan and Dr. William Crossley, for their support, encouragement and thoughtful input. Their professional quality and attitude complemented greatly my education here at Purdue.

I appreciate the financial and moral support from Galileo University, especially from Dr. Eduardo Suger for instilling my love for science. I thank Dr. Antonio Gillot for introducing me to the theory of automata and for pushing me to understand formal languages during the beginning of my undergraduate program.

I am indebted to my dear friend Dr. Abner Salgado for the hundreds of deep discussions about science, mathematics and the future of education in Guatemala. I specially thank him for proofreading the entire thesis; I will not forget his valuable input and his red ink. Also, I would like to thank my friend Gustavo Petri for his insightful conversations and support during those hard moments when I felt my work was useless.

I want to thank to my friends Carlos Zelada, Jose Ramirez, Keith Hall, Roberto Portillo, KiHyung Kim, Eric Lavetti, Mithun Jacob, Mauricio Gomez and many others with whom I enjoyed many hours of discussion about life and research. I am afraid that I could have forgotten the names of several individuals who in one way or another supported me during my studies, I ask beforehand for their forgiveness.

My deepest appreciation goes to my family who provided the determination to complete this work. My parents, Jorge F. Samayoa and Rosana Ranero for their unconditional support to me, my wife and daughters, and for encouraging me to work hard without jeopardizing my role as a father and husband. My brother, B3B3, and sister, ElenaMa, for all their love and support, and for always having an open ear for me. I could write a book as thick as this dissertation about how thankful I am with my wife, Guisela, and our daughters, Ana and Sabrina. All the memories we collected together and their infinite love and encouragement was the key to complete my doctoral work; las amo!

Finally, I would like to acknowledge and thank the support from the Fulright-LASPAU program during my master's degree which lead to my doctoral studies and the creation of this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Samayoa, Jorge Ph.D., Purdue University, May 2015. Calculus for Decision Systems. Major Professor: Abhijit V. Deshmukh.

The conceptualization of the term "system" has become highly dependent on the application domain. What a physicist means by the term *system* might be different than what a sociologist means by the same term. In 1956, Bertalanffy [1] defined a system as " a set of units with relationships among them". This and many other definitions of system share the idea of a system as a black box that has parts or elements interacting between each other. This means that at some level of abstraction all systems are similar, what eventually differentiates one system from another is the set of underlining equations which describe how these parts interact within the system.

In this dissertation we develop a framework that allows us to characterize systems from an interaction level, i.e., a framework that gives us the capability to capture how/when the elements of the system interact. This framework is a process algebra called Calculus for Decision Systems (CDS). This calculus provides means to create mathematical expressions that capture how the systems interact and react to different stimuli. It also provides the ability to formulate procedures to analyze these interactions and to further derive other interesting insights of the system.

After defining the syntax and reduction rules of the CDS, we develop a notion of behavioral equivalence for decision systems. This equivalence, called bisimulation, allows us to compare decision systems from the behavioral standpoint. We apply our results to games in extensive form, some physical systems, and cyber-physical systems.

Using the CDS for the study of games in extensive form we were able to define the concept of subgame perfect equilibrium for a two-person game with perfect in-

formation. Then, we investigate the behavior of two games played in parallel by one of the players. We also explore different couplings between games, and compare – using bisimulation – the behavior of two games that are the result of two different couplings. The results showed that, with some probability, the behavior of playing a game as first player, or second player, could be irrelevant.

Decision systems can be comprised by multiple decision makers. We show that in the case where two decision makers interact, we can use extensive games to represent the conflict resolution. For the case where there are more than two decision makers, we presented how to characterize the interactions between elements within an organizational structure. Organizational structures can be perceived as multiple players interacting in a game. In the context of organizational structures, we use the CDS as an information sharing mechanism to transfer the inputs and outputs from one extensive game to another. We show the suitability of our calculus for the analysis of organizational structures, and point out some potential research extensions for the analysis of organizational structures.

The other general area we investigate using the CDS is cyber-physical systems. Cyber-physical systems or CPS is a class of systems that are characterized by a tight relationship between systems (or processes) in the areas of computing, communication and physics. We use the CDS to describe the interaction between elements in some simple mechanical system, as well as a particular case of the generalized railroad crossing (GRC) problem, which is a typical case of CPS. We show two approaches to the solution of the GRC problem.

This dissertation does not intend to develop new methods to solve game theoretical problems or equations of motion of a physical system, it aims to be a seminal work towards the creation of a general framework to study systems and equivalence of systems from a formal standpoint, and to increase the applications of formal methods to real-world problems.

1. INTRODUCTION

1.1 Motivation

What is the fundamental difference between any two arbitrary systems? Is there any framework that allows us to describe the interactions between elements of a systems regardless of the domain these systems belong to? These are some the questions that motivate this work. Throughout the years different definitions of a system have appeared in the literature. According to Bertalanffy [1] in 1956, "*A system is a set of units with relationships among them*". Van Gigch [2] in 1991 defined a system as "*an assembly or set of related elements*". A minimalist abstract definition consistent with most of those in the literature is that a *system* is "*a set of interconnected elements forming an integrated whole*". Even though we can convey all the different definitions in a single sentence, the real conceptualization and behavioral analysis of a particular system will depend on the domain this system belongs to. For example, consider two systems, an electrical transformer and a vending machine. On the one hand, the transformer will change its state according to the voltage/current introduced to it. On the other hand, the vending machine changes its state when a token is inserted. Note that at some level of abstraction, both of these systems are simply interconnected elements with inputs and outputs, forming an integrated whole; i.e., these systems, at this are pretty similar. However, when it comes to analyze their behavior, the underlining equations and logics describing the interactions between their elements may be completely different. This motivates us to investigate the creation of a framework that allows us to study systems as interacting elements sharing information, but with the capability of including details pertinent to the domain these systems belong to. Also, since systems will be studied at a level of abstraction where they seem to be almost the same system, this framework should provide us with a mechanism for comparing these systems at this level.

This framework is the first step towards the creation of a general theory that will allow us to relate principles, algorithms and methods between two or more different domains that – from a theoretical standpoint – are currently disconnected. We aim to provide the language and a comparison mechanism that will allow us to describe systems that may have similar behavior but currently may not have a theoretical and practical relationship. Finding similarities between systems that belong to two different domains could suggest new approaches and extensions for current methodologies developed in one domain into another domain.

1.2 Research Approach

If we want to reason about properties and behavior of systems which interact independently and via information sharing, it is inevitable to come across a *process algebra* (also called, *process calculus*, see [3, 4]). Informally, a **process algebra** is a mathematical structure concerned with the properties and relationships of the behavior of a system, satisfying the axioms given for the defined operators of the algebra. Process algebras have been widely use for the specification of a variety of systems in different domains. A process algebra is mainly comprised by a syntax and an operational semantics. Informally, the syntax defines the symbols allowed in the algebra (the alphabet), and the operators that define how these symbols may be related to each other. The operational semantics defines how the elements of the alphabet and operators react under different conditions (the interaction rules).

In this dissertation, we present a process algebra, called *Calculus for Decision Systems*, that allows us to reason about systems at the interaction level. This calculus provides a mean to create mathematical expressions which capture how the systems interact and react to different stimuli. Also it gives us the ability to formulate procedures to analyze these interactions, and to further derive other interesting insights of the system.

The calculus for decision systems, or simply CDS, is a framework for the analysis of the interactions and behavior of – what we defined as – *decision systems*. A decision systems, informally speaking, is a set of interconnected elements called decision makers that make judgments (or decisions) based on the available (internal and external) information. These systems are classified according to the number of decision makers involved in the system, and the interaction of the system with its environment. The latter case only differentiates systems which interact with their environment (open systems), or only interact using information produced and modified within the system (closed system). In the former we study some of the theoretical tools that have been developed for the analysis of their behavior of systems where sub-systems have a conflict of interests, and we use the CDS to reason about the interactions between their elements.

Having decision systems and the CDS well defined, we create an action semantics that relates the reactions rules defined in the operational semantics, with an action that triggers the reactions. This mechanism allows us to define a *Labelled Transition Systems* for decision systems. A labelled transition systems (or LTS), in loose words, is an automaton without an initial or final state. This automaton-like system gives us the flexibility to model systems that can start in any state and may finish in any other state. Using the LTS, we are able to define a notion of equivalence between decision systems.

The equivalence between systems may depend on their response (functional equivalence), their structure (structural equivalence), or their behavior (behavioral equivalence). Using the CDS we are able to study structural congruence between processes, but also behavioral equivalence. The concept of behavioral equivalence was originally developed by Park in [5], and it is considered the hallmark of process algebras. Informally, two processes are considered to be behaviorally equivalent when their externally observed behavior appear to be the same. This concept is referred as bisimulation.

The concept of behavioral equivalence allows us to study equivalences between decision systems. In the case where there are more than one decision maker, the

conflicts of interests are resolved via game theory. There are three forms of mathematical abstractions of a game: the extensive, the normal and the characteristic function forms. The main difference not only relays on the form the information is presented, but in the amount of information captured by of the different forms. In this dissertation we are mainly dealing with the first form of a game, the extensive form. Using the CDS, we explore games in extensive form and are able to study (1) subgame perfect equilibrium, (2) couplings between extensive games, and (3) equivalence between coupled games. Furthermore, we present how the CDS allows us to describe structures comprised by multiple extensive games; i.e., organizational structures, where the output of each element of the structure may be the resolution of a conflict between players of such element.

Another application of the CDS we explored in this dissertation is the specification of cyber-physical systems. Cyber-physical systems or CPS is a class of systems that are characterized by a tight relationship between systems (or processes) in the areas of computing, communication and physics. We described how we deal with *time* in the context of the CDS, and show how to describe some simple physical systems. A typical cyber-physical system used in the literature for exemplifying the specification of a CPS, and various specification and validations methodologies is the Generalized Railroad Crossing Problem (GRC) problem; see [6–10]. We use the CDS to describe the interactions between the elements of the GRC problem. We provide two approaches to the problem, and argue that using some of the theoretical benefits of the CDS simplifies the formulation of the solution.

This dissertation does not intend to develop new methods to solve game theoretical problems or equations of motion of a physical system. As mentioned before, it aims to be a seminal work towards the creation of a general framework to study systems and equivalence of systems from a formal standpoint, and to increase the applications of formal methods to real-world problems.

1.3 Research Outline

As stated, the purpose of this dissertation is to develop a framework that allows us to characterize decision systems. To this end, it is necessary to study the roots of process algebras in order to understand the dynamics and insights of this type of algebras. Also, we need to examine the different approaches that been have utilized to investigate the areas that are related to the work presented here. This is accomplished in Chapter 2.

In Chapter 3, the calculus for decision systems is presented. In order to develop a proper process algebra we present the syntax of the language as well as the operational semantics of it. We start with a minimal process algebra for decision systems (MPADS) and build up our calculus from it. After having the CDS well defined we use it to describe an aircraft acquisition problem. We finish this chapter with a comparison of the CDS with other calculi in the context of decision systems.

Chapter 4 is devoted to the formalization of the notion of bisimulation. This chapter provides the formal insights that relate a labelled transition system to the operational semantics of the CDS. We start by defining an action semantics for our calculus, and then we define the concept of a LTS. Moreover, using these concepts, we formally define bisimulation in the context of the CDS. Furthermore, we defined the notion of *bisimulation up to* $\sim$, which will allow us to use some of the properties of $\sim$ to our advantage, when studying bisimulation.

Chapter 5 is devoted to the study of extensive games using the CDS. We introduce the key concepts of game theory for the study of games in extensive form. Then, we present our characterization of an extensive game using the CDS, and the concept of subgame perfect equilibrium. Furthermore, we study couplings between extensive games. Using the CDS, we were able to describe the interactions between players playing two games concurrently. Also, we use the notion of bisimulation to compare when two games are behaviorally equivalent.

In Chapter 6, we study cyber-physical systems in the context of the CDS. We mention some of the efforts made in the area of formal methods for the specification of CPS. Moreover, we show the work done in the area of process algebras for the study of CPS. Also, we present how we deal with *time* in the CDS and exemplify our concepts by describing the interactions between elements in a mechanical system. We discuss the specification of a CPS, the GRC problem, using the CDS.

Chapter 7 presents the main conclusions of this dissertation, and future research directions.

2. PROCESS CALCULI AND DECISION SYSTEMS

This chapter provides the necessary background for the understanding of the role of process calculi in decision systems. There are four main goals of this chapter: (1) present a formal definition of a decision system, (2) provide the relationship between decision systems and game theory, (3) present an overview of the relationship between game theory and the area of formal methods, and (4) provide a motivation of the utilization of Process Calculi for the description and analysis of decision systems. In this chapter we identify the initial requirements for this dissertation and use them as a driving force for the work presented in the following chapters.

2.1 Decision Systems

In this section we formalize the definition of a Decision System (DS). According to [11] a decision is "*the act or process of deciding*". A minimalist definition of the word decision consistent with those in the literature is that a decision is the process of conveying information in order to make a judgment or draw a conclusion. On the other hand, many definitions of the word *system* can be encountered in the literature. According to Bertalanffy [1] in 1956, "*A system is a set of units with relationships among them*". Van Gigch [2] in 1991 defined a system as "*an assembly or set of related elements*". We can therefore conceive a systems as *a set of interconnected elements forming an integrated whole*. If we combine – in some sense – these two definitions we can state the following informal definition of a decision system.

Definition 2.1.1 (Decision System - Informal)**.** A **Decision System** is a set of interconnected elements – called decision makers – that make judgments (or decisions) based on the available (internal and external) information. □

The schematic of a decision system is depicted in Figure 2.1. Indisputably, information plays an important role in a decision. In fact, J. E. Russo et al, [12] showed how distortions of the available information affect decisions and how current information may lead to the distortion of new information. In our context we make the distinction between current and new information by differentiating between internal and external information. With this and Definition 2.1.1 in mind, we can now state a formal definition of a decision system.

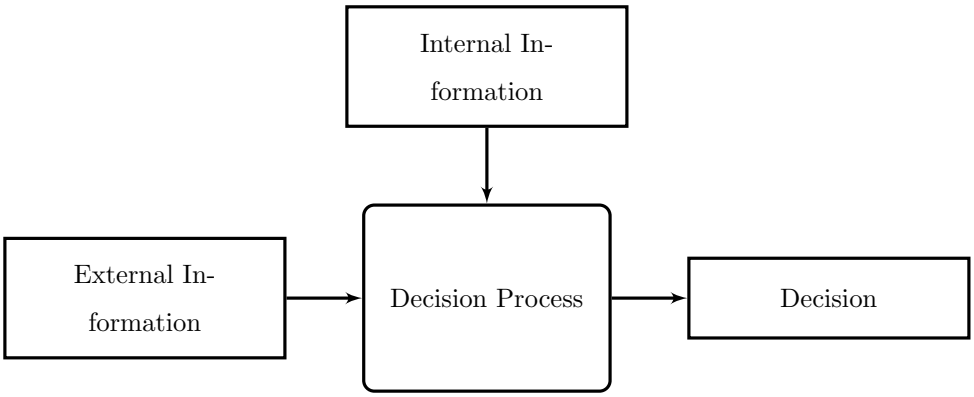


Figure 2.1. Pictorial representation of a Decision System.

Definition 2.1.2 (Decision System)**.** A Decision System (DS), is a triple $(\mathcal{I}, \mathcal{O}, \mathcal{N})$, where

- $\mathcal{I}$=$Int \sqcup Ext$ is the set of input information, where
  - Int= Internal Information set, and
  - Ext= External Information set.
- $\mathcal{O}$ is the output information (Decision).
- $\mathcal{N}$ is a set of interacting decision makers, were every $d \in \mathcal{N}$ is a function $d : \mathcal{I} \to \mathcal{O}$, and $0 < \overline{\overline{\mathcal{N}}} < \infty$[1].

□

[1] $\overline{\overline{\mathcal{N}}}$ denotes the cardinality of the set $\mathcal{N}$.

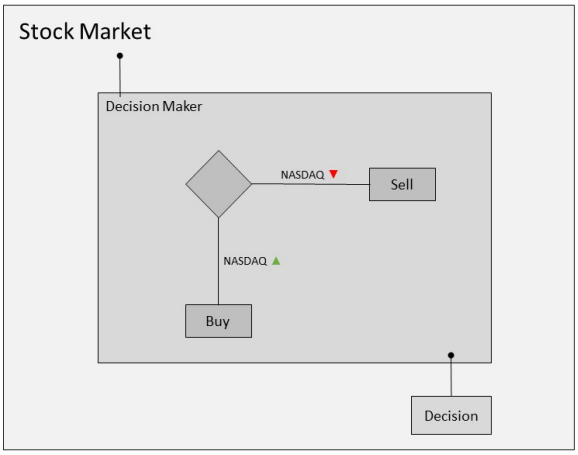
The intuitive interpretation of Definition 2.1.2 is as follows: The input information set $\mathcal{I}{=}Int \sqcup Ext$ is comprised by two subsets of information. The set Int is the information proprietary of the decision maker (decision thresholds, internal bureaucracy, structure of the organization, etc.). The subset Ext is the information transmitted from the environment or other decision system to the decision system (stock price, natural disasters, decisions made by others, etc.). The set $\mathcal{N}$ of interacting decision makers is the set of entities or elements that have control over the decision (when to buy, what to respond to a specific input, etc.). Lastly, the set $\mathcal{O}$ is the output of the decision system, i.e., the decision made by the decision maker to a specific input.

Figure 2.2 shows different examples of decision systems. Figure 2.2(a) is a decision system with a single decision maker (i.e., $\overline{\overline{\mathcal{N}}} = 1$), where the external information is given by the Stock Market, the internal information is the pre-defined threshold of increment/decrement of the NASDAQ that defines the structure of the decision, and the output information is the decision itself; it can be either Sell or Buy. Figure 2.2(b) is similar than the former case with the difference that $\overline{\overline{\mathcal{N}}} = 2$, and that the decision of Company 1 is an element of the external information of Company 2, i.e., the decision of Company 1 affects Company 2. The decision system depicted in Figure 2.2(c) illustrates the interpretation of an electrical transformer as a decision system. In this case, the external information is given by the input voltage, V_{in}, and the internal information is given by the ratio $n_1 : n_2$. There is only one decision maker (i.e., $\overline{\overline{\mathcal{N}}} = 1$) which makes decision according to

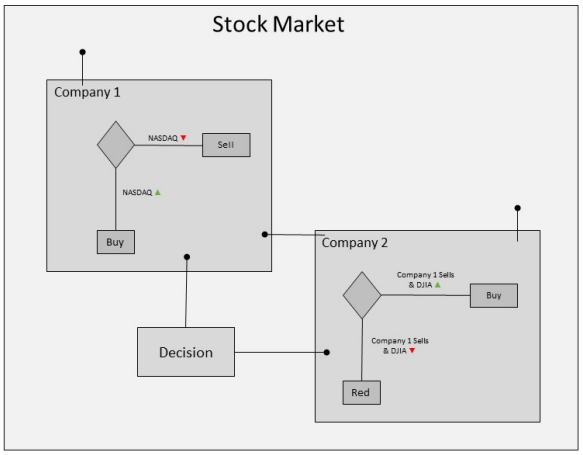
$$V_{out} = V_{in}\frac{n_2}{n_1} \tag{2.1}$$

2.1.1 Classification of Decision Systems

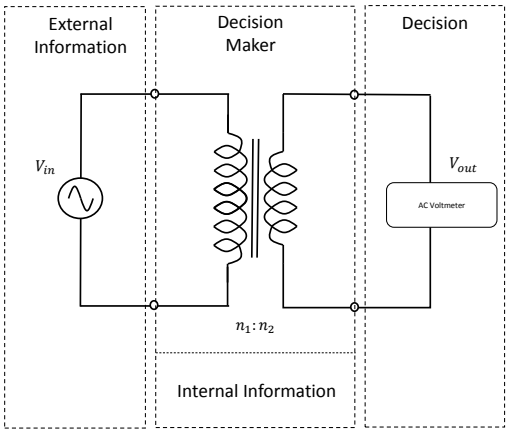
There is a great variety of systems that fit into the definition of a decision system. We classify decision systems according to their interaction with their environment, i.e., Open Decision Systems and Closed Decision Systems. Within this classification we identify a natural sub-classification of decision systems – similar to that suggested



(a) Decision system with a single decision maker.



(b) Decision system with a two decision makers.



(c) Electrical transformer interpreted as a decision system.

Figure 2.2. Examples of different decision systems.

by Luce and Raiffa [13] – based on the number of decision makers involved in the system, i.e., $\overline{\overline{\mathcal{N}}} = 1$ (single decision maker) or $1 < \overline{\overline{\mathcal{N}}} < \infty$ (two or more decision makers).

Open Decision Systems: In systems theory, an open system is characterized by having a continuous interaction with its environment. Similarly, an open decision system (O-DS) is a system that fits into Definition 2.1.2 having a non-empty set of external information ($Ext \neq \emptyset$). i.e., the environment does provide information that is taken into account by the decision maker in order to make a decision. An open decision system interacts with its sub-systems by sharing information such as objective functions, values or other type of decision involved in the decision process.

Closed Decision Systems: A decision system is said to be closed when the set of external information is empty ($Ext = \emptyset$). This means that the decision will be made only based upon the internal information of the system. In a closed decision system the decision maker has all the information necessary to make a decision. A **complex decision system** is a decision system comprised by multiple decision sub-systems. Note that a complex decision system may be closed, whereas all its sub-systems are open decision systems; sharing information such as objective functions, values, etc.

Decision Systems when $\overline{\overline{\mathcal{N}}} = 1$: These decision systems have only one decision maker. There are multiple systems that fit into the definition of a decision system for this particular case. In fact, any system whose output depends solely on the perturbations to the input introduced by the criteria of a single entity (e.g. Nature, a human, etc.) can be perceived as a decision system with only one decision maker. For example, consider the simple electrical system depicted in figure 2.2(c). This is a closed electrical circuit. From the decision systems perspective, however, this is an open decision system where the power source provides the external information to the decision maker (the transformer) and the winding ratio the internal information.

In this example the decision is ruled out by the laws of electromagnetic induction not by a human.

The most important feature of a decision system with a single decision maker is that all decisions are not affected by any sort of conflict between multiple entities. Meaning that all decisions are ruled out by the effect of the input information on a particular utility function, physical law, optimization principle/criteria, etc. The decision is the response of the system to a particular input. In Chapter 6 we provide a deeper understanding of this type of decision systems.

Decision Systems when $1 < \overline{\overline{\mathcal{N}}} < \infty$: A decision system with more than one decision maker is affected by the different interests of each decision maker. This means that the decision (the output of the system) will be a conflict resolution between the two (or more) decision makers. Unlike decision systems with a single decision makers, the tools available for analyzing conflict resolution between entities is not vast. In fact, game theory is a trivial choice for this task.

We have shown that the classification of decision systems depends on two aspects: (1) the interrelation between the system and its environment, and (2) the number of decision makers involved in the system. In Chapter 6 we elaborate more about decision systems with a single decision maker. In the case of multiple decision makers within a decision system we argued that the appropriate tool for the analysis of the behavior of these systems is game theory. In the following sub-section we present some of the formal tools available in the literature to analyze conflict resolution between players.

2.2 Formal Methods and Game Theory

2.2.1 Formal Methods

Before studying the intersection between formal methods and game theory, a brief introduction to formal methods is presented. The area of formal methods began in the 1930's as an intersection between computer science, mathematics and linguistics [14]. According to [15], formal methods are "*the application of mathematical synthesis and analysis techniques to the development of computer controlled systems.*" This area includes a variety of subjects such as predicate logic, automata theory, and formal languages. Formal methods can be viewed as the formal way to describe a problem or model a system [14, 16]. Figure 2.3 shows a basic problem solving framework, presented in [14], which encompasses formal and informal domains.

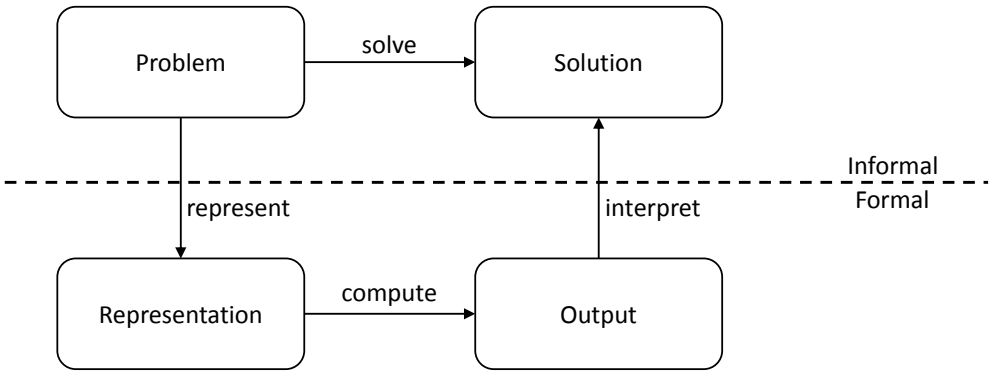


Figure 2.3. Problem solving framework

Even though formal methods have been around for so many years, their popularity in areas outside of software and hardware systems started to increase in the last twenty five years. This effect is the result of the use of intelligent systems to manage complex systems in different industries [17, 18]. We now focus our attention to three main subjects encompassed in the area of formal methods: predicate logic, automata theory and formal languages.

Table 2.1
First order predicate calculus: Symbols and Meanings.

Symbol	**Meaning**
$\vee$	or
$\wedge$	and
$\neg$	not
$\Rightarrow$	logically implies
$\Leftrightarrow$	logically equivalent
$\forall$	for all
$\exists$	there exists

Table 2.2
First order calculus: Rules

$\boldsymbol{p}$	$\boldsymbol{q}$	$\boldsymbol{p \vee q}$	$\boldsymbol{p \wedge q}$	$\boldsymbol{\neg p}$	$\boldsymbol{p \Rightarrow q}$	$\boldsymbol{p \Leftrightarrow q}$
T	T	T	T	F	T	T
T	F	T	F	F	F	F
F	T	T	F	T	T	F
F	F	F	F	T	T	T

Predicate Logic

Predicate logic is essential to understand the fundamentals of formal methods. Logic and propositional calculus are based on statements or propositions, called sentences, which are either true (T) or false (F). This calculus (also called first-order logic) is equipped with a countable set of letters $A, B, C, \ldots$, a set of symbols (see Table 2.1) and rules (see Table 2.2) that allow us to assess the truth value of compound statements depending on the veracity of its primitives, e.g., it allows us to assert about the value of $p \vee q$ depending on the values of p and q.

Note that this calculus contains quantifications. A quantification is a non-logical constant that include names and entities. For example: $\forall X.dots(X) \Rightarrow red(X)$, means that all dots are red. The sentence $\exists X.blue(X)$ means that, among all dots, there exists at least one blue. H. Pospesel [19] provides a more comprehensive introduction to propositional calculus.

Automata Theory

The theory of automata provides a basis for a tremendously useful branch of modeling and analysis: that of computational complexity. Automata theory describes the ability of abstract machines to evaluate mathematical problems given a set of inputs. Such machines surround us in the modern world; from traffic lights at intersections to test stand controllers for aerospace applications, various forms of automata observe, react to, and control the environment around us.

Automata theory possesses a noble history which spans several branches of mathematics and science. The mathematician Alan Turing developed a logical theory in Princeton in 1936, describing a machine which consisted of: [20].

> "...an infinite memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed. At any moment there is one symbol in the machine... called the scanned symbol. The machine can alter the scanned symbol and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine. However, the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine. Any symbol on the tape may therefore eventually have an innings."

This theory is an example of the most abstract and capable automaton, the usefulness of which was not realized until much later. Two other men widely regarded as pioneers in this area were Warren McCulloch and Walter Pitts, a pair of neurophysiologists

from the Massachusetts Institute of Technology. Their seminal 1943 paper, "*A Logical Calculus of the Ideas Immanent in Nervous Activity*" [21], was the first to present a description of finite automata, making significant contributions to neural network theory. This theory was later broadened to include other types of automata by Bell Labs researchers George Mealy [22] and Edward Moore [23].

An *automaton* is an abstract object which recognizes or accepts, in discrete time steps, a string of characters from a particular set (Σ^*), where $\Sigma \supset \Sigma^*$ is a finite alphabet. Given that the elements accepted by the automaton are a subset of Σ^*, we say that these elements, called words, form a language. Therefore, an automaton, M, recognizes or accepts a language, $L(M)$, contained in Σ^*. Definition 2.2.1 is the formal definition of an automaton, similar to that presented by Jirí Adámek and Vera Trnková [24].

Definition 2.2.1 (Automaton)**.** An automaton is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where

Q is a set, called the *set of states*;

Σ is a non-empty set, called the *input alphabet*;

$\delta : Q \times \Sigma \to Q$ is the *transition function*;

$q_0 \in Q$ is the *initial state*;

$F \subseteq Q$ is the set of *accept states*.

□

There are several different types of automata, varying in their types of transitions, the number of states which they contain, the type of inputs they require, and the type of acceptance conditions they allow. Here, we present a few which are considered crucial types in the study of automata theory.

Deterministic finite automata (DFA): The main difference between the definition outlined above and the formal definition of a deterministic finite automata is that the cardinality of Q is finite, and the transition function δ is one-to-one, i.e., for each input symbol, the machine's next state may be one, and only one, of its other states. The formal definition of a DFA is given in Definition 2.2.2.

Definition 2.2.2 (DFA)**.** A deterministic finite automaton M is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states; Σ is a non-empty finite set, called the input alphabet; $\delta : Q \times \Sigma \to Q$ is the transition function; $q_0 \in Q$ is the *initial state*; $F \subseteq Q$ is the set of accept states. □

The input of δ is a letter of Σ and a state belonging to Q. The output is a state of Q (possibly the same one). If the automaton is in state q and reads the letter r, then (q, r) is the input for δ and $\delta(q, r)$ is the next state. Given a non-empty string in $\Sigma^* \subset \Sigma = \{a, b\}$ the automaton reads the string or word as follows:

1. It begins in the initial state q_0, and **reads** the first letter in the string. If the first letter is $a \in \Sigma$, then it **moves** to state $s_1 = \delta(q_0, a)$.

2. The automaton reads the the second letter of the string. If the second letter is b, then the automaton moves to state $s_2 = \delta(s_1, b)$.

3. As the automaton continues to read the given string of letters from the alphabet, it moves from one state to another. Eventually, the automaton reads every letter in the string and then **stops**.

4. After reading the last letter of the string, if the current state belongs to the set of acceptance states, then the automaton **accepts** the string. Otherwise, it rejects it.

Definition 2.2.3 (Regular Language)**.** A language L is defined to be regular if there is a DFA, D, such that $L = \mathcal{L}(D)$, i.e., L is recognized by D. □

Table 2.3
Example of a DFA.

	$\boldsymbol{a}$	$\boldsymbol{b}$
$\boldsymbol{s_1}$	s_2	s_1
$\boldsymbol{s_2}$	s_1	s_2

Example 2.2.1. Let $M = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{s_1, s_2\}$, $\Sigma = \{a, b\}$, $q_0 = s_1$, $F = \{s_2\}$, and δ is defined by Table 2.3.

Figure 2.4 shows the **state diagram** of the DFA. Note that we identify q_0 with an *arrow*, and the elements of F with a *double circle*.
Some of the strings recognized by this automaton are *a, ab, aaabb, bbaaa, baaab, bbbabb* and *baababb*. In general, the language recognized by this automaton is a regular language (see Definition 2.2.3) given by $b^*a(ab^*ab^*)^*b^*$, where * is the *Kleene star* (see [25] for a formal definition of the Kleene star. Informally, a^* denotes any non-negative number (including zero) of symbols a). ▲

Non-deterministic Finite Automata (NFA): A non-deterministic finite automaton is a finite state machine very similar to a DFA. The main difference is that the transition function δ is not one-to-one, i.e., for each input symbol, its next state may be any one of several possible states. Thus, the next state is an element of 2^S, where S is the number of states. With this in mind, the formal definition of a NFA is as follows:

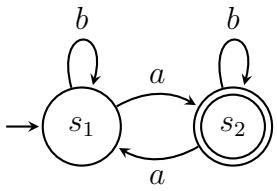


Figure 2.4. State diagram of a DFA.

Definition 2.2.4. Let λ be the empty string. A Deterministic Finite Automaton M is a quintuple $M = (Q, \Sigma, \Delta, q_0, F)$, where Q is a finite set of states; Σ is a non-empty finite set, called the input alphabet; $\Delta : Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$ is the transition function; $q_0 \in Q$ is the *initial state*; $F \subseteq Q$ is the set of accept states. □

NFAs and DFAs work in a similar fashion, with the following main differences:

- The number of possible candidate states to which a move can occur after each step can be greater than one.
- The automaton can move from one state to another using the empty string.
- The automaton accepts a string if the machines stops in any accept state, or can move to another accept state by the empty string λ.

In fact, the automaton M accepts the string (word) if a sequence of states, $r_0, r_1, \ldots, r_n$ exists in Q with the following conditions [26]:

$r_0 = q_0$

$r_{i+1} \in \Delta(r_i, a_{i+1})$, for $i = 0, \ldots, n-1$

$r_n \in F$.

These conditions are suggesting that the automaton has to start in q_0. Transitions are ruled by the transition relation Δ, and the automaton will accept a string w if the last input of w causes the machine to stop in one of the accepting states. Otherwise, the string is rejected by the automaton.

Example 2.2.2. Let $M = (\{s_0, s_1, s_2, s_3, s_4\}, \{0, 1\}, \Delta, s_0, \{s_1, s_3\})$, where the transition relation is given in Table 2.4. Figure 2.5 is the *state diagram* of this automaton.

Some of the strings recognized by this automaton are *λ,11,000,1010,0101* and *001100*. In general, the language recognized by this automaton is a regular language given by $(1^*(01^*01^*)^*) \cup (0^*(10^*10^*)^*)$. ▲

Example 2.2.3. The state diagram of a non-deterministic automata can be represented in a more compact way. Figure 2.6 shows an example of an automaton that recognizes the language $(a \vee b)^* a (a \vee b)^* a (a \vee b)^*$. ▲

Table 2.4
Example of a Non-deterministic Finite Automaton.

	0	**1**	$\boldsymbol{\lambda}$
$\boldsymbol{s_0}$	$\{\}$	$\{\}$	$\{s_1, s_3\}$
$\boldsymbol{s_1}$	$\{s_2\}$	$\{s_1\}$	$\{\}$
$\boldsymbol{s_2}$	$\{s_1\}$	$\{s_2\}$	$\{\}$
$\boldsymbol{s_3}$	$\{s_3\}$	$\{s_4\}$	$\{\}$
$\boldsymbol{s_4}$	$\{s_4\}$	$\{s_3\}$	$\{\}$

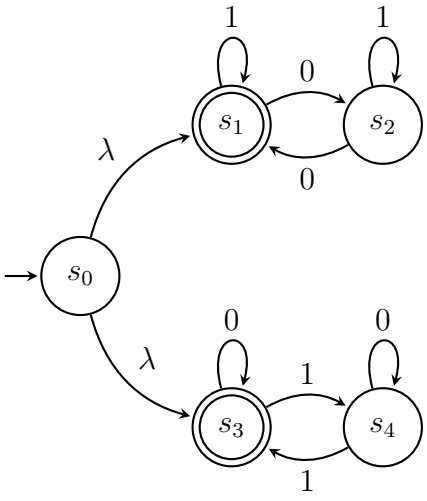


Figure 2.5. State diagram of a Non-deterministic Finite Automaton.

Turing Machine A Turing Machine, named after the mathematician Alan Turing, is an abstract machine capable of recognizing and manipulating a language. In the context of Turing Machines, we assume that the input strings are on a strip of tape, and the machine can erase, print and re-print symbols on the strip [27].

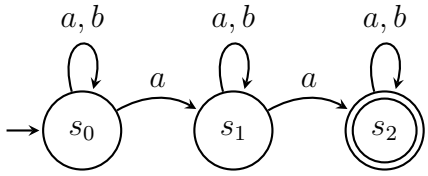


Figure 2.6. Compact state diagram representation of an NFA.

Definition 2.2.5. A **Deterministic** Turing Machine is a quintuple $(Q, \Sigma, \Gamma, \delta, s_0, h)$, where where Q is the set of states, Σ is a finite set of tape symbols, which includes the alphabet and #, Γ is a finite, non-empty set of the type symbols, s_0 is the starting state, h is the halt state, and δ is a function from $Q \times \Gamma$ to $Q \times \Gamma \times N$ where N consists of either L, which indicates a movement on the tape one position to the left, R, which indicates a movement on the tape one position to the right, or #, which indicates that no movement takes place. □

Example 2.2.4. An example of a rule is (s_1, a, s_2, b, L), which means that if the machine is in state s_1 and reads the letter a, it is to change to state s_2, print the letter b in place of the letter a and move one square to the left. ▲

Example 2.2.5. The rule $(s_1, a, s_2, \#, R)$ means that if the machine is in state s_1 and reads the letter a, it changes to state s_2 , erases the a and moves one square to the right. ▲

Example 2.2.6. The rule $(s_1, \#, h, \#, \#)$ means that if the machine is in state s_1 and reads a *blank* then it **halts**, and thus does not print anything or move the position on the tape. ▲

Turing Machines play an important role in theory of computing and formal languages. In general, automata theory has been applied to a great variety of areas [28], in particular in the area of verification and model checking [29].

Formal Languages

In general, languages are divided in two classes: formal languages and natural languages. A **formal language** is comprised by a set of symbols (or syntax) and rules of formation (or semantics) by which these symbols can be combined into objects called sentences [30]. **Natural languages** [31], also known as ordinary languages, are those languages used by humans to communicate such as English, Spanish, etc. The study of natural languages has had a great impact in the area of formal languages. One of the great contributors to the area of natural languages was Noam Chomsky with his work, among others, on formal grammars for describing fragments of natural languages in 1959 [32, 33]. That same year, John Backus [34] introduced the well-known Backus Normal Form (or Backus-Naur Form, or simply BNF) to give a formal description of the context-free syntax of the programming language Algol 60 [35].

Informally, a **Context-free Grammar** (or CFG) is a grammar whose production rules can be applied to a non-terminal symbol, regardless of the surrounding symbols. The formal definition of a CFG is given in Definition 2.2.6.

Definition 2.2.6 (Context-free Grammar)**.** A grammar $G = (V, T, S, P)$ is said to be context-free if all productions in P have the form

$$A \to x$$

Where $A \in V$ and $x \in (V \cup T)^*$. Where V is the set of non-terminal symbols, T the set of terminal symbols, $S \in V$ is the start symbol, and P is the set of rules (or productions).

A language L is said to be context-free if and only if there is a context-free grammar G such that $L = L(G)$, see [30]. □

One of the most common forms of expressing grammars on programming languages and formal languages in general is by their BNF. In this notation, variables are enclosed in triangular brackets, terminal symbols are written without any special marking, and it utilizes symbols such as $|$,$+$ or $*$ to express operations between the expressions or terms. Figure 2.7 is an example of a BNF grammar.

$$< expression > ::= < term > \mid < term > + < expression >$$
$$< term > ::= < factor > \mid < term > * < factor >$$

Figure 2.7. Example of a BNF grammar.

There is a vast number of formal languages in the literature. In this section we present some of the languages that are somehow related with this dissertation. For a more comprehensive list of formal languages we refer to [36]. We focalize our attention to three formal languages: Abstract State Machines (ASM), Petri Nets and Process Calculi. Since Process Calculi is tightly related to this research we will present a more comprehensive overview in posterior sections and present a brief introduction to the first two languages (ASM and Petri Nets) in this section.

Abstract State Machines: An abstract state machine (ASM), is a formal language for specification and verification originally developed by Yuri Gurevich in the 1980's. It is mostly known as the ASM method. From [37], the ASM method is characterized by providing a framework that allows the system engineer to (a) systematically separate multiple concerns, concepts and techniques of systems development activities, and (b) freely choose for each task an appropriate combination of concepts and techniques at the given level of abstraction and precision where the task occurs.

Egon Brger, in [38] states the following mayor activities of the typical software life cycle that are linked via the ASM method:

- Requirements capture by constructing satisfactory ground models, i.e., accurate high-level system blueprints, serving as precise contract and formulated in a language which is understood by all stakeholders,

- Detailed design by stepwise refinement, bridging the gap between specification and code design by piecemeal, systematically documented detailing of abstract models down to executable code,

- Validation of models by their simulation, based upon the notion of ASM run and supported by numerous tools to execute ASMs and others,
- Verification of model properties by proof techniques, also tools supported,
- Documentation for inspection, reuse and maintenance by providing, through the intermediate models and their analysis, explicit descriptions of the software structure and of the major design decisions.

The book [39], E. Brger and R. F. Strk present some real-life case studies and industrial applications analyzed using the ASM method.

Petri Nets: A Petri Net is a graphical and mathematical modeling tool for systems that are characterized as being concurrent, asynchronous, distributed and/or non-deterministic. This tool was created by Carl Adam Petri in 1962 in his dissertation at the Technische Universität Darmstadt, Germany [40, 41]. Petri nets can be used as visual communication aid similar than flow charts, block diagrams, etc. To simulate the dynamic and concurrent behavior of a system, Petri nets utilize tokens that are passed along the nodes of the network. These nets can be used by both practitioners and theoreticians, making this approach a powerful medium of communication between them [42].

A Petri net is comprised by a graph N which is directed, weighted and bipartite having two kinds of nodes, called places (drawn as circles) and transitions (drawn as bars or boxes). The graph N has an initial state, called initial marking, M_0. The arcs can go either from a place to a transition or vice versa. The weights of the arc are positive integers, where a k-weighted arc can be interpreted as the set of k parallel arcs; unitary weights are usually omitted. Definition 2.2.7 presents the formal definition of a Petri Net, from [42].

Definition 2.2.7 (Petri Net)**.** A Petri net is a tuple, $PN = (P, T, F, W, M_0)$ where:

$P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places,

$T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions,

$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs,

$W : F \to \{1, 2, 3, \ldots\}$ is a weight function,

$M_0 : P \to \{0, 1, 2, 3, \ldots\}$ is the initial marking,

$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

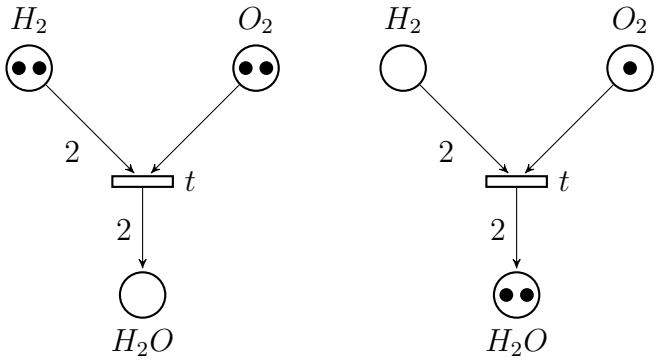
A Petri net structure $N = (P, T, F, W)$ without any specific initial marking is denoted by N. A Petri net with the given initial marking is denoted by (N, M_0). □

The transitions of a Petri net change according to the following (firing) rule [43]:

1. Tokens are moved by the *firing* of the transitions of the net.
2. A transition must be enabled in order to fire. (A transition is enabled when all of its input places have a token in them.)
3. The transition fires by removing the enabling tokens from their input places and generating new tokens which are deposited in the output places of the transition.

Note that a *source transition* (one without any inputs) is unconditionally enabled, and the firing of a sink transition (one without any output place) consumes tokens, but does not produce any. Example 2.2.7 taken from [42] exemplifies the firing rule.

Example 2.2.7. In this example, the well-known chemical reaction: $2H_2 + O_2 \to 2H_2O$ is presented. Two tokens in each input place. Fig. 2.8(a) shows that two units of H_2 and O_2 are available, and the transition t is enabled. After firing t, the marking will change to the one shown in Fig. 2.8(b), where the transition t is no longer enabled. ▲



(a) Marking before firing the enabled transition t.

(b) Marking after firing t, where t is disabled.

Figure 2.8. Example of a transition rule in Petri Nets.

Applications of Petri Nets: Petri nets have a wide range of applications. In the area of formal languages Petri nets are used to model the flow of information and control of actions in a system (see [44, 45]). According to [43] a Petri net properly models a system if every sequence of actions in the modeled system is possible in the Petri net and every sequence of actions in the Petri net represents a possible sequence in the modeled system. Petri nets have been applied to failure analysis [46], performance analysis [47], Manufacturing systems [48], workflow management [49], and other dynamic systems [50]. For a more comprehensive introduction to Petri nets we refer the reader to [42, 51].

Applicability of Formal Methods

Formal Methods have a wide range of applications. According to [52] Formal Methods, in particular Formal Languages, are used in most of the engineering applications to reduce the time and efforts required to communicate and manage the underlining system or process. The accurate and precise definitions of the underlining system in a formal language can be used to validate the system and also to use it as a base for computer-aided solutions. Woodcock et al, [53] presented an extensive

survey about the state of the art of formal methods emphasizing their applicability in engineering and software specification. Hall [54], identified the following seven "myths" of formal methods:

1. Formal Methods can guarantee that software is perfect. "The fact is that no method can guarantee perfection." Fortunately, the usefulness of these methods does not depend on this utter perfection.

2. Formal Methods are all about program proving. Formal methods are more about (formal) specifications a precise definition of what the software is intended to do. "Program verification is only one aspect of formal methods."

3. Formal Methods are only useful for safety-critical systems. "The fact is that formal specifications help with any system." Formal methods has been implemented in non-critical systems (see [54], p.13).

4. Formal methods require highly trained mathematicians. "The fact is that mathematics for specifications is easy." A much higher level of mathematical skills is needed if formal methods are being implemented for safety-critical projects. "The main difficulty is making the right connections between the real world and the mathematical formalism."

5. Formal Methods increase the cost of development. "The fact is that writing a formal specification decreases the cost of development." According to [55] - Rolls-Royce and Associates - the use of formal methods "did not give rise to an increase in cost or timescale of software development. They did, however, require a far larger specification effort than had been seen in previous projects, representing 7% of the total project cost."

6. Formal Methods are unacceptable to users. "The fact is that formal specifications help users understand what they are getting." The specification of a system captures what the user wants before it is built.

7. Formal Methods are not used on real, large-scale software. "The fact is that formal methods are used daily on industrial projects." This has been shown above.

Advantages and Disadvantages of the use of Formal Methods: This list of advantages and disadvantages of using formal methods for the specification and development of software (or systems) is mostly taken from [56].

Advantages

1. Developing a formal specification in detail will require a deep and detailed understanding of the system requirements. Even if the specification is not used in a formal development process, the detection of errors in the requirements is a potent argument for developing a formal specification [54]. Early discovery of requirements problems are usually much cheaper to correct than if they are found at later stages in the development process.
2. As the specification is expressed in a language with formally defined semantics, you can analyze it automatically to discover inconsistencies and incompleteness.
3. Using methods such as the B-method [57], the formal specification of the system can be transformed into a program through a sequence transformations so that the specification is met.
4. Program testing costs may be reduced because the program is verified against its specification.

Disadvantages

1. Domain experts may not understand a formal specification so they cannot check that it accurately represents their requirements. On the other hand, software engineers, who understand the formal specification, may not understand the application domain so they may not be sure that the formal specification is an accurate reflection of the system requirements.

2. Quantifying the costs of creating a formal specification may be easy, but estimating the possible cost savings that will result from its use is more difficult. As a result, managers may be reluctant to take the risk of implementing this approach.
3. Software engineers may not have training on the use formal specification languages. Hence, they may be reluctant to propose their use in development processes.
4. Scaling current approaches to formal specification up to a very large systems is difficult. Formal specification has been mostly used for specifying critical kernel software rather than complete systems.

In this dissertation we are mostly concerned about the relationship between formal methods and decision systems. As we have stated above, game theory plays an important role in the analysis of decision systems. Therefore we now turn our attention specifically on the applications of formal methods to game theory.

2.2.2 Games and Logic

Games and logic have a long history. Using game theoretical ideas to express quantifications goes back to Pierce in the early 1800's [58]. On the other hand, more modern connections between game theory and logic have been developed in the other direction using modal logic (to be discussed later in this section) to study game theoretical problems. Game theory has provided a set of new ideas for describing interactions which may involve a conflict of interest between two or more parties. Even though currently there have been efforts to enrich the utilization of logic to different situations presented in game theory (e.g. cooperative games and imperfect information) the logic community has focused its attention to mainly studying two-player extensive games of perfect information which are strictly competitive [59].

In this section we present some of the most relevant areas that have either used the concepts of game theory applied to formal methods or the concepts of formal

methods applied to game theory. We are interested in the latter, however we first provide literature review of the area that makes use of game theoretical concepts to perform semantical evaluations of statements; this area is known as game semantics.

Game Semantics

One of the first people to use the concepts of a two-player game to perform semantical evaluation of assertions made with respect to some give situation was Jaakko Hintikka [60]. He developed what is known as the Game-Theoretical Semantics (or GTS), [61]. Around the same time Paul Lorenzen first introduced the concept of game semantics for logic [62], where he used the concepts of game theory in argumentation between a defender and critic of a claim. In game semantics the word "game", "debate" and "dialogue" are used synonymously, and analyzes, for example, proofs from the perspective of a two-person game. For example, the winner of the debate (or game) is P (the proponent) or O (the opponent) according to whether the proof is correct or not. Note that in this case the opponent O has actions to perform (i.e., no possible strategies to pick from); these are known as proof-base semantics, which is a special case of game semantics.

In programming languages, a type determines the kind of computation that may take place, in this case types will be modelled as games; a program of type A determines how the systems behaves, so programs will be represented as strategies for P, that is, predetermined responses to the moves O may make.

The concepts of game semantics have been applied to different types of logics, such as linear logic [63] and affine logic [64], and several variations of it have been proposed, such as probabilistic game semantics [65], algorithmic game semantics [66] and games for true concurrency [67]. Hyland and Ong [68] proposed a semantical analysis of sequential functional languages using polyadic π-calculus, reading input π-actions as opponents moves, and output π-actions as proponents moves. Currently there is not a book in this area, but there is popular introduction made by Abramsky and McCusker [69, 70].

Formal Methods applied to Game Theoretical problems

Currently there are three main approaches that use logic to study properties of games: modal logic (or modal languages) [71], temporal logic [72], and propositional dynamic Logic [73]. The main objective of these approaches is to study the structural properties of games in extensive form (we introduce games in extensive form on Chapter 5) , all from either a verification standpoint or from the perspective of *model-checking*. Therefore most of the different languages based on these logics do not take into account the payoffs of the players in the game, but only the structure of the game tree. We show this fact more in detail next.

Modal Logic: P. Blackburn et al, [74] begin their book with the following statement:

> "*Ask three modal logicians what modal logic is, and you are likely to get al least three different answers.*"

In this section we provide a short introduction and literature review of the applications of modal logic to game theory. We will not attempt to present a clear definition of modal logic, but the reader will have a flavor of what this type of logic is about, and what the advantages and disadvantages of modal logic are when it comes to characterize game theoretical problems. There are a few concepts and terminology that will be used throughout this section that presumes a previous knowledge on game theory. For a short introduction to these concepts we refer to Chapter 5. Before introducing modal logic we give an intuition of what is called a "modality". Modalities indicate the mode in which certain statement is said to be true. For example, consider the statement "*it is a sunny day*". We can think of different ways in which we might interpret its truth or falsity. Is it necessary a sunny day? It is known that it is a sunny day? Is it believed that it is a sunny day? Is it a sunny day now, or will it be a sunny day in the future? All of these modifications of the original statement are called *modalities*. These type of modifications of an assertion

Table 2.5
Syntax of modal logic.

$A, B, C, \ldots$	A countable, infinite set of letters
$\vee$, $\wedge$, and $\Rightarrow$	A set of binary operators
$\Box$, $\Diamond$, and $\neg$	A set of unitary operators
(, and)	Brackets

are, obviously, not easily handle by the classical propositional and predicate logic. Therefore, logicians developed a logic that can handle this type of assertions, the modal logic.

Modal logic extends the classical propositional calculus to modal operators [71]. For example, "*player A chooses to go up*" might be qualified by saying "*player A possibly chooses to go up*". In modal logic the assertion is represented as an operator '*possibly*', attached to the assertion "*player A will go up*". Now we provide a more tangible syntax for modal logic.

Table 2.5 presents the (classical) syntax of modal logic [75, 76]. Most of this symbols and operators were already defined in section 2.2.1. The new operators are the 'box' operator, $\Box$, denoting "necessity", and the diamond operator, $\Diamond$, denoting possibility. For example, if P is true, then the equation $\Box P$ means 'P is necessarily true'. Similarly, if P is true, then the equation $\Diamond P$ means that 'P is possibly true'.

van Benthem has vastly studied games from a modal logic perspective, [59, 77]. He analyzes extensive games as interactive process models, using modal logic and notions of bisimulation[2] [78]. Pauly [79–81] created a modal logic for reasoning about what groups of agents can bring about by collective action, i.e. coalitional games. Pauly and Wooldridge [82] use modal logic for representing and reasoning about the properties of game theoretic mechanisms.

[2]The concept of bisimulation is briefly discussed in Section 2.2.3. In Chapter 4 we explain in depth the role of bisimulation in the context of process algebras and decision systems.

One natural extension of modal logic is the **epistemic modal logic**. Epistemic modal logic is concerned with reasoning about knowledge and belief [83], which has its roots in modal logic [71]. Stalnaker [84] studied the concept of rationality and backward induction using an epistemic approach, without making any connections to modal logic. Lorini and Moisan [85] proposed a modal epistemic logic that allows to characterize the concept of rationality and backward induction as defined by Aumann [86–88]; they provide a syntactic proof of Aumanns theorem. In [89] J. van Benthem uses an epistemic logic to investigate the question when are two games equal? by means of bisimulation between epistemic models. The latter work together with the work of [90] show the applicability of modal logic to extensive games and present a characterization of backward induction and the notion Nash equilibrium. Neither of them provide a clear instantiation of a game where its subgame perfect equilibrium is found using their characterization of backward induction. For further reading on the role of knowledge and beliefs in game theory refer to [91]. In Chapter 3 of [92], Hoek and Pauly present other formalisms that use modal logic to express game forms. For a more comprehensive introduction to modal logic we refer to [71, 75, 93] and [94].

Temporal Logic Temporal logic is an special type of modal logic that provides a formal framework for describing and reasoning about how the truth values of assertions change over time [94]. Two new temporal operators are incorporated in a temporal logic:

- '*Sometimes* P' operator, which is true now, if there is a future moment at which P becomes true, and
- '*Always* Q' operator, which is true now, if Q is true at all future moments.

There are three mayor varieties of temporal logic: linear-time temporal logic (LTL), branching-time temporal logic (BTL), and alternating-time temporal logic (ATL). The main difference between the three of them is the way they quantify paths. Both LTL [95] and BTL [96,97] are focused on the specification and verification of computer

programs, while ATL was introduced as a logic that offers a selective quantification over those paths that are possible outcomes of games [72].

The **syntax of ATL** is defined with respect to a finite set Π of prepositions and a finite set $\Sigma = \{1, \ldots, k\}$ of players, with formulas as either one of the following (from [72]):

1. p, for propositions $p \in \Pi$.
2. $\neg\varphi$ or $\varphi_1 \vee \varphi_2$, where φ, φ_1 and φ_2 are ATL formulas.
3. $\langle\langle A\rangle\rangle \bigcirc \varphi$, $\langle\langle A\rangle\rangle \Box\varphi$, or $\langle\langle A\rangle\rangle \varphi_1 \mathcal{U} \varphi_2$, where $A \subseteq \Sigma$ is a set of players, and φ, φ_1, and φ_2 are ATL formulas.

The operator $\langle\langle\ \rangle\rangle$ is a '*path finder*', and $\bigcirc$ ("next"), $\Box$ ("always"), and $\mathcal{U}$ ("until") are temporal operators.

van der Hoek et al, [98] present an extension of ATL, called CATL, which supports reasoning about the abilities of agents and their coalitions in games. This extension was done by adding a '*commitment*' operator, $C_i(\sigma, \varphi)$, read "if it were the case that agent i committed to strategy σ, then φ". This operator adds the ability to reason about possible different choices by the players in a game allowing CATL to express Nash equilibrium and Pareto optimality [99]. Some of the contributions provided by CATL are (from [98] p.157-158):

- According to the authors, CATL is the first logic which combines reasoning about strategic ability with counterfactual reasoning.
- The combination of ability operators and the strategic counterfactual operator enables to express characteristics of games much more elegantly and intuitively than the previously proposed.
- CATL extends ATL by introducing strategies as first-class components of the language.

Unlike [72] in their work on ATL, [98] presents instantiations of a prototype game characterized with CATL where Nash equilibrium and Pareto optimality are satisfied.

In CATL utility functions of players are defined as $\hat{u}_i : \hat{J}_{\hat{Ag}} \rightarrow \mathbb{R}$. This function assigns a real-value utility to each combination of players strategies. Even though [98] is able to characterize Nash equilibrium, this abstract characterization of utility does not allow them to find the expected utility of a game at equilibrium.

Propositional Dynamic Logic Propositional dynamic logic (or PDL) is another mayor approach to characterize properties of games. This logic was introduced by Fischer and Ladner [100] in 1979, based on Pratts dynamic logic [101]. The main idea provided by DL that triggered the creation of PDL was the idea of associating a modality $[\alpha]$ to each computer program α of a programming language. For example, the formula $[\alpha]\phi$ is read whenever program α terminates, it must do so in a state satisfying formula ϕ. This logic was first applied to games by Parikh [102]. As previously mentioned, applications to coalitions of players were developed later by Pauly [79]. Ramanujam and Simon [103] use propositional dynamic logic to characterize situations where a player's strategy may depend on properties of the opponent's strategy. This logic is designed to represent and reason about propositional properties of programs (in our case, games). The syntax presented in [104] is based upon two sets of symbols: a countable set Φ_0 of atomic formulas and a countable set Π_0 of atomic programs. These formulas and programs over such base are defined as follows:

- Every atomic formula is a formula.
- 0 (false) is a formula.
- If A is a formula then $\neg A$ (not A) is a formula.
- If A and B are formulas then $(A \vee B)$ (A or B) is a formula.
- If α is a program and A is a formula then $[\alpha]A$ (every execution of α from the present state leads to a state where A is true) is a formula.
- Every atomic program is a program.

- If α and β are programs then $(\alpha;\beta)$ (do α followed by β) is a program.
- If α and β are programs then $(\alpha \cup \beta)$ (do α or β, non-deterministically) is a program.
- If α is a program then α^* (repeat α a finite, but non-deterministically determined, number of times) is a program.
- If A is a formula then $A?$ (proceed if A is true, else *fail*) is a program.

Syntactically, propositional dynamic logic is a modal logic that provides an algebraic structure to the set of modalities. This approach has been mostly used for concurrent games. Even though van Benthem et al, argued in [89] that no "exotic new formalisms" were needed to study extensive games, six years later, in [105], they developed developed a dynamic logic to reason about (simultaneous) extensive games in terms of a parallel operator. Gosh et al, [106] also developed a dynamic logic to study the behavior of concurrent games. The latter presented a sound and complete axiomatization of the logic (an extension of the standard PDL) and showed that the satisfiability problem for the logic is decidable. In this section we have shown the main connections between logic and game theory. Some authors such as [89, 106] mentioned the potential relationships between process algebras and game theory. In fact, some authors such as Ghosh et al, [106] literary state the evident applicability of process algebras to reason about games (see Section 2.3). As mentioned in Chapter 1, one of the main goals of this dissertation is to study the applicability of process algebras to decision systems, and game theory is an important tool for analyzing such systems. Therefore, we now turn our attention to process algebras and show their connections with game theory and decision systems.

2.2.3 Overview of Process Calculi

2.2.3.1 Introduction

To introduce the term 'process algebra' let us consider the meaning of the word *process* and *algebra* separately. According to [107], the word *process* refers to *behavior of a system*. According to [11], the word *algebra* refers to "*the field of mathematics concerned with the properties and relationships of abstract entities manipulated in symbolic form under operations often analogous to those of arithmetic*". A **process algebra** is then a mathematical structure concerned with the properties and relationships of the behavior of a system, satisfying the axioms given for the defined operators of the algebra. In this context, a process is an element of the universe of a process algebra. Using these axioms and operators, calculations can be performed with processes. This calculations are often referred to as equational reasoning. Therefore, a process algebra is also known as process calculus (plural: process calculi).

One of the simplest models for computation since the beginning of the twentieth century is the *automaton*. We already introduced automata and showed some of the benefits of *automata theory* (see Section 2.2.1. Even though automata provide a rich theory for describing the behavior of certain types of systems, the theory was found to be lacking of expressiveness for systems that interact with another systems, i.e., automata theory lacks the notion of *interaction*. This notion is needed in order to describe parallel (concurrent) or distributed systems, or so-called *reactive systems* (see [108, p. 479]). The theory of interacting, parallel and distributed systems is called *concurrent theory*. A process algebra is one of the approaches to concurrent theory. Therefore, most of the process algebras include a *parallel operator* as one of the basic operators of the algebra [109].

In the context of process algebras, automata are called *transition systems*, and, unlike automata, the notion of equivalence is based on the behavior of the transition system instead of the language it is generated by the transition system (as it is the case in automata theory). This notion of behavioral equivalence is called *bisimula-*

tion, which considers two transition systems equal if and only if they can mimic the behavior of each other in any reachable state.

In this section we provide a short introduction to process algebras starting with some historical remarks, continuing with important definitions pertinent to the area. We also present some of the most popular applications of process algebras. We dedicate a great portion of this section to one of the most popular process algebras: The π-calculus. Most of the topics discussed here are paramount for this dissertation, especially some of the characteristics discussed about π-calculus.

2.2.3.2 Historical Remarks

In the 1920's scientists started wondering about the necessity of a formal framework to study computation from the logic standpoint. It was not until the 1930s when Alonzo Church created the so-called λ-calculus [110, 111] as a simple semantics for computation, enabling computation to be study formally. This calculus expresses computation based on function abstraction and application using variable binding and substitution. Churchs contribution lead to the creation of several mayor contributions such as the Turing Machines and the area known as formal methods (see Section 2.2.1). For a comprehensive introduction to λ-calculus we refer to [112–116]. Also we refer to [113, 116] for better understanding of some of the languages derived from λ-calculus such as the STLC. Cardone and Hindley provide a detailed history of λ-calculus in [117].

In the early 1960's, computer scientists like Carl Petri and Carl Hewitt started studying computation in concurrent systems [40, 118]. The actor model [119] and Petri nets [42] (see Section 2.2.1) where the two most popular tools for the study of this type of systems. Bekič with his studies of semantics of parallel programs [120] moved the study of concurrent systems a step forward by providing a more rigorous and formal approach to analyze concurrent systems. It was until the period from 1973 to 1980 where the concept of process algebra was well defined with the creation of

the calculus for communicating systems (CCS) by Milner [121, 122] and the language CSP (Communicating Sequential Processes) by Hoare [123, 124]. Even though the concept of a Process Algebra was well conceived by Milner and Hoare, it was not until 1982 with the work of Bergstra and Klop [125] when the phrase "process algebra" was used for the first time. Bergstra and Klop created the theory called *algebra of communicating processes* or ACP [126]. After this a great variety of process calculi have been developed. Some of the more popular are π-calculus [127] (the predecessor of CCS), The ambient calculus [128], and many others that will be mentioned later in this chapter.

2.2.3.3 Applications of Process Calculi

Several important developments have occurred since the creation of the basic process algebras CCS, CSP and ACP. Most of these developments and applications have been extracted from [107] and [4]. We present some of the developments in the theory of process algebras, and also some of the most relevant process algebras having extensions to reason about time, probability, and mobility. Finally we state some of the applications to hybrid systems and other areas.

Theory of Process Algebras: There have been multiple advances in the theory of process algebras since the creation of CCS, CSP and ACP. Aceto, in [129], and Baeten, in [107], list some of the most important advances made over the last three decades. Without any debate, one of the most important theoretical advances in the theory of process algebras is the formulation of bisimulation by Park [5]. Since its creation, the notions of strong and weak bisimulation have been the main mechanism for the study of behavioral equivalence. In Chapter 4 we introduce the notion of bisimulation in the context of process algebras, and then we extend this notion to study behavioral equivalence of decision systems.

There are two important theoretical developments in the theory of process algebras that we want to highlight in this dissertation:

1. Recursion: There is a vast number of process calculi that have implemented some form *recursion* in their syntax. Recursion increases the expressiveness of a process algebra significantly by allowing the definition of infinite processes. We will extend our investigation regarding recursion in Section 3.2.3.

2. Session Types: The study of data types within any programming language has been paramount in computer science. Session types provide a formal framework to study communications between two parties. Session types was developed by Honda in [130] and has been applied to a wide range of process calculi and programming languages over the last decade. For a more extensive introduction to session types we refer the reader to [131], [132], and [133].

There are many relevant results in the theory of process calculi, in the rest of this section we turn our attention to process algebras that have been extended with a notion of time, probability, and mobility.

Time and Process Algebras: One of the first extensions of a process algebra to include a notion of time was presented by [134] as an extension of the CSP. Later on, variants of the CCS with timing appeared on the literature, see [135]. J.C.M. Baeten and J.A. Bergstra extended the ACP to real time actions in [136]. More recent extensions have been proposed, for example [137] propose an extension of π-calculus to reason about time (we will discuss this in more detail in Chapter 6). A more comprehensive overview of timed process algebras can be found in [138] and [109].

Probability and Stochastics in Process Algebras: Several extensions of process algebras to include probabilities and stochastic information have been proposed. There have been extensions of the CSP [139], CCS [140] and ACP [141] to include stochasticity. J. Hillston in [142] proposed the *performance enhanced process algebra* or PEPA to reason about stochastic processes (with the restriction that the activities have to be exponentially distributed). A generic process algebra with probabilistic

choice can be found in [109, Ch. 11].Even though there has been progress in this area, much further work is needed, as stated in [107].

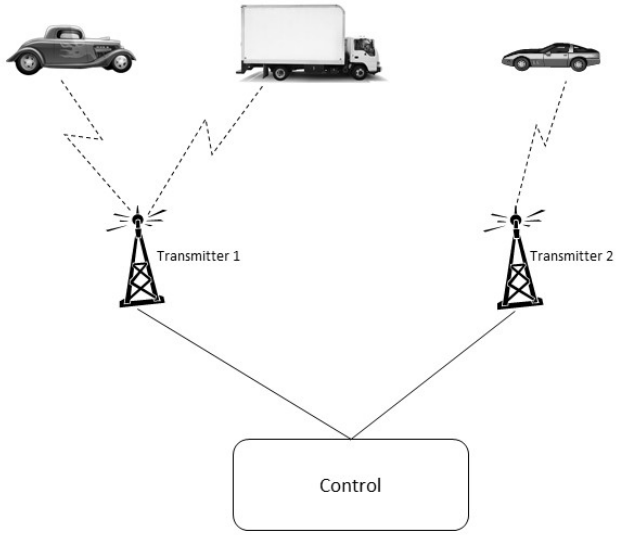
Mobility: According to [127] there are roughly three types of mobility:

1. Processes move in the physical space of computing sites;
2. Processes move in the virtual space of linked processes;
3. Links move in the virtual space of linked processes.

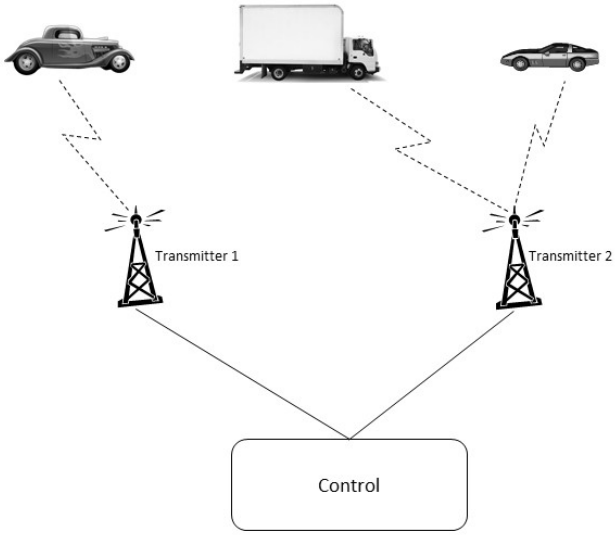
In the context of process algebras, the concept of mobility refers to the fact that processes can move around in space, making them change their communications links when doing so. Note that the move of a process can be represented entirely by the movement of its links. Therefore, option 2 can be reduced to option 3; in fact, choice 3 is more general than 2, since a process can have multiple links, and you can remove only one of those links without moving the others. Therefore, in the context of process algebras, the concept of mobility is usually referred to links than move in the virtual space of linked processes; option 3.

Figure 2.9 in a pictorial exemplification of the concept of mobility [127]. Figure 2.9(a) represents a network of cars that are connected to a control station through transmitters. The first (transmitter 1) has two cars connected to it. In Figure 2.9(b) it is shown a potential "evolution" of Figure 2.9(a), where one of the cars had to move from *transmitter 1* to *transmitter 2* due to a virtual event (e.g. signal fading). This switch of transmitters could have been done by the physical movement of the cars. However, in the context of process algebras we are concern concerned to understand the former case, when the linkage changed due to some virtual event.

The concept of mobility has been applied on networks with dynamic topology, and has been – without a question – dominated by the π-calculus [127,143]. Later on, see L. Cardeli and A. Gordon in [128] developed the *ambient calculus* that extends the concept of process mobility to mobile computation (i.e., executable code that moves around the network).



(a) Before signal fading.



(b) After signal fading.

Figure 2.9. Example of mobility.

Other Applications of Process Algebras: There have been multiple efforts to apply process algebras to different fields. For example, Flight Control [144], commercial clusters [15], business transaction systems [145] and others [109, 146, 147]. More recently applications to hybrid systems (using hybrid automata, see [148, 149]) and cyber-physical systems, see [150].

We have shown some of the relevant extensions of process algebras to reason about a great variety of systems in different fields. One of the most popular process algebras, as previously mentioned, is the π-calculus [127]. This calculus introduced the notion of message passing through communication channels, which extended the expressiveness of processes algebras significantly. In the next section we provide a more deep introduction to Milner's π-calculus.

π-calculus:

In this section we present the π-calculus. This calculus is a process algebra where process interact by sending communication links to each other. One of the most relevant features is that process can transfer communication links between two processes. The recipient can then use such link for further communication with other processes. In this section we present the syntax, structural congruence and reaction rules of the monadic version of the calculus (the term *monadic* refers to the version of the calculus in which a message consists of exactly one name). We also present an example to show the expressiveness of the calculus. We finalize the section with a list of some of the application areas of this calculus.

Syntax of the π-calculus: The syntax of the π-calculus presented in this section can be found in Milner's original book [127]. We need to assume that there exists an infinite set of $\mathcal{N}$ names, usually represented by lower case letters $x, y, z, \ldots, \text{Rang}(\mathcal{N})$. The syntax of the *action prefixes*, π, is described in Table 2.6, and the processes, in π-calculus, are defined in Definition 2.2.8.

Table 2.6
Atomic prefixes of π-calculus

$\pi ::=$	$x(y)$	receive y along channel x.
	$\bar{x}\langle y\rangle$	send y along channel x.
	τ	unobservable (or silent) action

Definition 2.2.8. The set $\mathcal{P}^\pi$ of π-calculus process expressions is defined by the following syntax:

$$P ::= \sum_{i\in I} \pi_i.P_i \mid P_1|P_2 \mid (\nu x)P \mid !P$$

where I is any finite indexing set. The process $\sum_{i\in I} \pi_i.P_i$ are called summations or sums, where 0 is the empty sum (often omitted after an action, e.g. $x(y).0$ is written $x(y)$. □

Before given an example of the utilization of π-calculus, we present its structural congruence and reaction rules, from [127].

Definition 2.2.9. (Structural congruence of π-calculus) Two process expressions P and Q in the π-calculus are structurally congruent, written $P \equiv Q$, if we can transform one into the other by using the following equations (in either direction):

1. Change of bound names.
2. Reordering of terms in a summation.
3. $P|0 \equiv P$, $P|Q \equiv Q|P$, $P|(Q|R) \equiv (P|Q)|R$.
4. $newx(P|Q) \equiv P|new\ xQ$ if $x \notin fn(P)$,
   $new\ x0 \equiv 0$, $new\ xy\ P \equiv new\ yx\ P$.
5. $!P \equiv P|!P$.

□

Definition 2.2.10. (Reaction rules of the π-calculus) The reaction relation $\rightarrow$ over P^{π} contains exactly those transition which can be inferred from the rules in Table 2.7. □

Table 2.7
Reaction rules of the π-calculus

Identifier	**Reaction**
TAU	$\tau.P + M \rightarrow P$
REACT	$(x(y).P + M) \mid (\bar{x}.Q + N) \rightarrow \{x/y\}P \mid Q$
PAR	$\frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}$
RES	$\frac{P \rightarrow P'}{new\ x\ P \rightarrow new\ x\ P'}$
STRUCT	$\frac{P \rightarrow P'}{Q \rightarrow Q'}$ if $P \equiv Q$ and $P' \equiv Q'$

The extension from the monadic π-calculus to its polyadic version can be found on [127] p. 93, and [151]. To illustrate the applicability of the π-calculus we present the following example.

Example 2.2.8. This example has been taken directly from Wing in [152].

> Suppose you want to model a remote procedure call between a client and a server. Consider the following function, `incr`, running on the server. `incr` returns the integer one greater than its argument, x:
>
> ```
> int incr(int x) {return x+1;}
> ```
>
> First, we model the "`incr`" server as a process in π-calculus as follows:
>
> $$!\texttt{incr(a, x)}.\bar{\texttt{a}}\langle\ \texttt{x+1}\ \rangle$$
>
> Ignoring the ! for now, this process expression says that the `incr` channel accepts two inputs: one is the name of the channel, a, which we will use to return the result of calling `incr`, and the other is the argument, `x`, which will be instantiated with an integer value upon a client call. After the call,

the process will send back the result of incrementing its argument, `x`, on the channel `a`. The use of the replication operator, !, in the above process expression means that the "`incr`" server will happily make multiple copies of itself, one for each client interaction.

Now let us model a client call to the "`incr`" server. In the following assignment statement, the result of calling `incr` with 17 gets bound to the integer variable y:

$$\texttt{y := incr(17)}$$

and would look like this in π-calculus:

$$(\nu\ \texttt{a})(\texttt{incr}\ \langle \texttt{a, 17} \rangle\ |\ \texttt{a(y)}\ )$$

which says, in parallel: (1) send on the `incr` channel both the channel a (for passing back the result value) and the integer value 17, and (2) receive on the channel `a` the result `y`. The use of the ν operator (sometimes we write *new* instead of ν) guarantees that a private channel of communication is set up for each client interaction with the `incr` server. Putting the client and server processes in parallel together we get the final process expression:

$$\texttt{!incr(a, x).}\bar{\texttt{a}}\langle x+1 \rangle\ |\ (\nu\texttt{a})(\ \overline{\texttt{incr}}\langle\ \texttt{a, 17}\rangle\ |\ \texttt{a(y)}\ )$$

which expresses the client call to the "`incr`" server with the argument 17 and the assignment of the returned value 18 to `y`.

▲

This example showed most of the features of the calculus. More importantly, we want to highlight the ability of sending the name of the channel along the channel itself (This concept will be useful in the next chapter). Without this capability sending a value through a channels would not be possible. A more simplistic (but

visual) example, and a more comprehensive introduction, can be found in [153]. We shall not attempt an overview of all the variations and applications of the calculus. However, we present the most relevant.

Variants of the π-calculus: Let us present some of the most relevant variations of π-calculus. Throughout the section we present only the particular construct of the variation discussed at that point. At the end of the section we present a summary table with all the variants discussed in the section.

- **Match and Mismatch:** The match and mismatch operators are not included in the *pi*-calculus. The operator $[x = y].P$ is called the match operator; it is read: If $x = y$ then P. The mismatch operator, $[x \neq y].P$ is called mismatch. Without these operators the calculus has a certain loss of expressiveness.

- **Asynchronous π-Calculus:** The asynchronous π-calculus only allows outputs with no suffix (e.g. $\bar{x}\langle y\rangle$, instead of $\bar{x}\langle y\rangle.P$ as in the regular π-calculus). This modification yields to a smaller, but expressive, calculus. Parrow [153] shows how any process in the original calculus can be expressed using the asynchronous version. Hennessy presents an asynchronous π-calculus with match operator in [154, Ch. 2].

- **Higher-Order π-Calculus:** The main difference between this calculus and the original π-calculus is that the higher-order π-calculus allows processes (or agents) to be transmitted through communication channels. For example, given processes P and Q, and channel a. The following output prefix is allowed in the calculus: $\bar{a}\langle P\rangle.Q$.

- **Stochastic π-calculus:** Introduced by C. Priami in [155]. This extension of π-calculus allows prefixes to be exponentially distributed. Thus, the atomic components of processes are pairs of the form (π, r), where π is the action (as presented in Table 2.6) and r is the rate of the exponential distribution.

- **Distributed π-calculus:** This variant of π-calculus was introduced by M. Hennessy in [154]. It extends the calculus to reason about mobile agents in a distributed world by adding two contructs: *goto* $l.P$ (the migration construct) and $(newloc\ k : D_k)\ R$ (the location name creation construct). The former is read: If agent *goto* $l.P$ is currently residing at k it can migrate to the domain l and continuew there with the execution of P. The agent $(newloc\ k : D_k)\ R$ creates a new location k of type D_k and then launches the code R.

- **Typed π-calculus:** This variant explores the issue of assigning types to π-calculus expressions. This variant was created with the idea of defining type systems to detect errors statically. By adding channel names to the values that can be exchanged, and modifying the type system according to those changes, [156] presents the simply-typed π-calculus.

Table 2.8 presents and overview of the variants of π-calculus. We now present some of the areas that have utilized π-calculus for modeling and verification of concurrent systems.

Some Applications of the π-calculus The π-calculus has been used to describe several different types of concurrent systems, in particular protocols. Other extensions not mentioned above include the application to cryptographic protocols [157] (the Spi-calculus) and business processes [158, 159] (BPML or business process modeling language). In biology, processes of the π-calculus are seen as chemical solutions, in which molecules interact concurrently [160]. Laird in [161] presented a game semantics (see Section 2.2.2) of the asynchronous π-calculus.

We introduced the π-calculus and some of the areas where this formalism has been utilized. The applicability of this calculus to different areas of science is still an active area of research by the time of this manuscript. Without a doubt, this calculus and process algebras in general, have provided a wealth of perspectives for the study of concurrent systems. In the next section we present the motivation for the use of process algebras to reason about decision systems.

Table 2.8
Variants of the π-calculus

Variant	**Construct Added**	**Note**
Match and Mismatch	$[x = y].P$	If $x = y$ then P
	$[x \neq y].P$	If $x \neq y$ then P
Asynchronous π-calculus	$\bar{x}\langle y \rangle$	Only allows outputs with no suffix
Higher-Order π-calculus	$\bar{a}\langle P \rangle.Q$	Processes can be sent through channels
Stochastic π-calculus	$(\pi, r).P$	Actions follow a $exp(r)$ distribution
Distributed π-calculus	$goto\ l.P$	Migration construct
	$(newloc\ k : D_k)\ R$	Location name creation construct
Typed π-calculus	N/A	Assigns types to π-calculus expressions.

2.3 Motivation for Studying Decision Systems using Process Calculi

In this section we answer the following questions: (1) Are process calculi a good tool for the study of decision systems? (2) If so, what characteristics does this process calculi has to have? In the beginning of this chapter we presented the classification of decision systems. This classification not only depends on the interaction between the decision system and its environment, but also on the cardinality of the set of decision makers of it. On the one hand, in the case where there are more than one decision maker, we discussed that game theory may provide the necessary means to reason about decision systems. On the other hand, in the case where there is only one decision maker in the decision system, there is a great variety of systems that fit into this class of decision systems. In fact, the only characteristic that this type of decision systems share is that all receive, manipulate and output/send information (by information here we mean either energy or momentum or information bits, etc.). Therefore, we need a tool that is abstract and flexible enough to model not only game theoretical problems but also systems that involve information sharing capabilities, among others. Moreover, this tool has to allow information sharing across systems that belong to different domains.

We have shown that formal methods indeed provide a wealth of approaches to reason about game theoretical problems. Moreover, we have discussed that some calculi have been encoded into other calculi, which seem beneficial in the context of a general theory of decision systems. Therefore, using a formal method to describe and analyze the behavior of decision systems seems appealing. Also we have shown that process algebras, in particular π-calculus, have a simple yet expressive way of characterizing systems that involve concurrency and message/information sharing. Moreover, as π-calculus has proven to be a powerful tool for modeling computation, using some of the ideas of this process algebra may provide a well-developed formal basis upon which to build a mathematical theory for decision systems. The drawback is that π-calculus lacks of some of the needed characteristics to characterize decision system.

In fact, there is not a process algebra that gathers all of the following characteristics:

1. A set of decision functions that characterize the utility functions of the decision makers.

2. A construct that allows recursive definitions.

3. A construct that allows to evaluate a function with the value received through certain communication channel. This construct is necessary to evaluate objective functions with values generated by other processes or sub-systems.

4. In addition to the standard communication capabilities such as communication channels and the ability to send the name of the channel along the channel itself, this process algebra has to have:

   (a) An internal choice operator. This is the typical operator "+" defined in many process algebras.

   (b) A sequential operator for processes. This is the "$\cdot$" operator included in many process algebras.

   (c) A conditional choice operator. This is the typical $if\ then\ else$ operator to characterize decisions.

   (d) A standard parallel operator "||" for concurrent processes.

   (e) A "fresh channel" operator "ν" for defining the scope of channels.

Note that except form characteristics 1, 3 and 4c, π-calculus has all of this features (where characteristic 2 is included in the *replication* construct of the π-calculus). Therefore, we need to create a process algebra, based on π-calculus, that includes all the features listed above. We have shown that there have been a great number of process algebras created over the years, each making its own set of choices for the different domains. The reader may wonder how good this is for science. In fact, Baeten et al, [162] argued that this is actually a good thing. Each different process algebra will have its own set of advantages and disadvantages, and as long as

there is an exchange of information between researchers, this is of benefit to science. Therefore, in the next chapter we propose a process algebra for decision systems based of the π-calculus called *Calculus for Decision Systems*.

2.4 Summary

In this chapter we presented a formal and informal definition of a decision system. Also, we presented a classification of this type of systems. This classification depends on two things: (1) the interaction of the system with its environment, and (2) the number of decision makers involved in the decision system. In the latter case we argued that game theory may provide the necessary means to reason about these systems. Therefore we presented a vast variety of tools that have been used to reason about game theoretical problems. In the former case, case (1), we argued that there is a variety of systems that fit into the definition of a decision system. We presented a class of formalisms called process calculi and provided the motivation for their utilization to study decision systems. We provided the requirements for a new process calculus for decision system.

3. THE CALCULUS FOR DECISION SYSTEMS

3.1 Introduction

Even if made by a machine or by ourselves, decisions are a crucial part of our daily life. Nowadays technologists have made huge steps towards making our decisions easier by providing technological tools/systems that either make decisions for us, or reduce our decision sample space. Even though the technologies to support these (decision) systems is well advanced, the design principles and techniques for analyzing and characterizing their behavior are in a more primitive stage, i.e., there is a big gap between implementation and formal characterization and analysis of systems' behavior. Closing this gap by providing solid foundations for formal characterizations of interactive systems is one of the goals of theoretical computer science, in particular, the area of process calculi.

Several different process calculi have been developed for the study of interactive systems (see section 2.2.3). Two of the more well-known are the Calculus for Communicating Systems or CCS [121] and its successor the π-calculus [127]. From [154], the CCS consists of (1) a simple formal language for describing systems in terms of their structure (i.e., how they are constructed from individual, but interconnected, components), and (2) a semantic theory that seeks to understand the behavior of systems described in the language, in terms of their ability to interact with users. Even though CCS was one of the most successful calculi ever created, its expressiveness is restricted to systems with static connection topology. This drawback was addressed by R. Milner in the π-calculus. In the latter, two new concepts were introduced to the theory of process calculi:

1. The ability to send the name of a channel along with the channel. This feature allows processes to send information such as values through channels.

2. Scope extrusion. This allows mobility between processes (See Section 2.2.3.3).

Since its creation, the π-calculus became one of the most (if not the most) popular process calculi in the literature, making it the focus of intensive research and the inspiration of many other process calculi (see Table 2.8). As we discussed in Chapter 2, it is of our interest to study decision systems using a process calculus. This calculus, called Calculus for Decision Systems (CDS), is also inspired by some of the ideas of Milner's π-calculus.

In this chapter we present the building blocks of the CDS. Throughout the chapter we use the term "Process Algebra" and "Process Calculus" interchangeably. We start by introducing a minimal calculus for the study of decision systems called MPADS. Though expressive, this calculus lacks of some semantic constructs, constraining its ability to express certain decision systems. In section 3.3 we present the syntax of the CDS and some examples of its basic use. Sections 3.4 and 3.5 introduce the operational semantics and the polyadic version of the calculus, respectively. Having the CDS well defined we proceed, in section 3.7, to discuss – in general terms – the notion of behavioral equivalence in the context of decision systems and the CDS. Before concluding the chapter we present a general comparison of the CDS with other calculi in the context of decision systems. We conclude with a summary of the results and concepts presented in this chapter.

3.2 Minimal Process Algebra for Decision Systems (MPADS)

In this section we introduce a Minimal Process Algebra for Decision Systems (MPADS). Before presenting its syntax, we need to introduce some of the concepts and notation that will be used throughout this section and the rest of this dissertation. Most of these concepts are standard in the theory of process calculi, and can be found in any standard book on Process Algebras such as [109].

3.2.1 Basic Concepts and Notation

Each state of a system will be represented by a *process expression*. This expression will carry information about the behavior and the structure of such system. For example, it will indicate the potential precursors of the process, or the processes which can establish communication with it. Processes rage over a set of process identifiers, usually denoted by a capital letter like $A, B, \ldots$, however in most of the examples and practical applications processes are given more meaningful names such as "Car", "Doctor" and so on. Processes are parametric on names; for example, a process $P\langle a, b\rangle$ is the process P with name parameters a and b. Usually these parameters are omitted and deducted from the context; e.g., $P\langle a\rangle = a$ will be written $P = a$. If a and b are names and P a process expression, then the process $\{^b/_a\}P$ is a process where all occurrences of a are replaced by the name b. The set of all names used in a process is denoted by $fn(P)$, where fn stands for *free names*. For example, if $a \notin fn(P)$, then what happens to a does not affect P (we will elaborate more into this when we present the structural congruence of the CDS).

3.2.2 Minimum requirements for MPADS

In Chapter 2 we stated the requirements for a calculus for decision systems. In this section we define the ground base for this calculus, we call it the Minimal Process Algebra for Decision Systems (MPADS). This process algebra is minimal in the sense that this process algebra will be able to express the most simple type of decision systems: closed decision systems with $\overline{\overline{\mathcal{N}}} = 1$, where decisions are binary, single-shot, and based on simple comparisons between values or alternatives. Therefore we need a process algebra with the following capabilities:

1. A set of two alternatives names, say $Alt = \{x, y\}$, which can lead to an output.
   Discussion: Since we require only binary decisions, we will have a maximum of two alternatives in a decision system. These alternatives are the potential paths a decision maker can pursue; e.g. "buy" or "sale".

2. A set of values $Val = \{v_1, v_2, \dots\}$ that are related to the different alternatives.
   Discussion: These are values such as real numbers, which are used to define the different alternatives or the responses to such alternatives, e.g., If the decision maker decides to "buy" he/she will get a value (or payoff) of 3. Note that one alternative may use multiple values.

3. A set of expressions $Exp = \{e_1, e_2, \dots\}$ which range over by the set of values and alternatives.
   Discussion: Expressions relate alternatives with values; e.g. "buying price$<$ 3" or "$ball \vee red$". These expressions are of the type Boolean, integer, string, etc. and may be of the form $e = e'$, $e \leq e'$, $e \vee e'$, etc.

4. A set of channel names $c_1, c_2, \dots$ for sharing internal information.
   Discussion: These channels will allow us to define the structure of the decision system, i.e., the information sharing structure of the system. These channels are known as communication channels (see Section 2.2.3.3). Using these channels, processes can send and receive information from other processes. In the case of MPADS, since $\overline{\overline{\mathcal{N}}} = 1$ and decisions are single-shot, we may only need a few of these channels – most likely, only one – to describe decision systems.
   Notation: It is customary to use round brackets when a channel is used for receiving information (e.g. $c(x)$) and angle brackets with a bar on the channel name for sending information (e.g. $\overline{c}\langle x\rangle$). There are more insights regarding communications channels that will be discussed later in this section.

5. An operator that allows us to share information between processes.
   Discussion: This operator, together with the communication channels, will allow us to define the structure of the decision system. This operator is the composition operator $P||Q$ to run P and Q concurrently. Using this operator and communication channels we can characterize the information sharing capability that is needed.

6. A decision process that allows us to make binary decisions.

   Discussion: In order to make binary decisions we need an operator (or process) that allows us to make a decision that depends on a single criterion; i.e., depending on certain value or expression, say e, we need to provide a response, say P, if e is satisfied, and a response, Q, if it is not satisfied. Therefore we incorporate the process `if` e `then` P `else` Q . Example: `if` *buying price* $<$ 3 `then` *buy* `else` *sell* .

7. A termination process that allows us to know when we have reached a terminal state (e.g. a decision).

   Discussion: Processes can *deadlock*, i.e., reach a state that is not terminal and there is no way out. We need to have an indicator that tells us that a process terminated, and it is in a state where "it is allowed" to terminate; a terminal state. For this purpose, we propose the process *nil*. This processes is the process that does nothing, written **0**. This process is usually omitted; for example, $a.b.0$ is written $a.b$. For example, a process whose only objective is to send a message a through a channel c and then "die". We can write this process as $c\langle a\rangle.0$, of simply $c\langle a\rangle$.

We can now define the syntax of the MPADS, see Fig 3.1. It presupposes a set *Exp* of expressions, ranged over by $e_1, e_2, \ldots$ that can be transmitted via communication channels. These expressions will include expressions of the type Boolean, integer, string, etc., ranged over by sets *Alt* and *Val*, but more importantly channel names themselves. The operation $\oslash$ is any binary Boolean operation ($\wedge, \vee$, etc.), $\oplus$ is any binary arithmetic operation ($+, -$, etc.), and $\ominus$ any relational operator ($=, \leq, \geq$, etc.). None of these operators add semantics to the MPADS.

The intuitive meaning of each of the syntactic construct is as follows:

- The simplest process, which does nothing, is represented by the term **0**. This process can be omitted after an action, for example $\bar{c}\langle x\rangle.0$ can be written as $\bar{c}\langle x\rangle$.

x, y	::= Alternatives names.
$v_1, v_2, v_3, \ldots$	::= Values.
$c_1, c_2, c_3, \ldots$	::= Channels names.
$e \in Exp$	::= $x \mid v \mid e \oslash e' \mid e \oplus e' \mid e \otimes e' \mid \neg e \mid -e \mid e = true \mid e = false \ldots$
ψ: Prefixes	::= $c(e) \mid \bar{c}\langle e \rangle$
P, Q: Processes	::= $\mathbf{0} \mid \psi.P \mid$ `if` e `then` P `else` $Q \mid P \mid\mid Q$

Figure 3.1. Syntax of the MPADS.

- The term **c(e).P** represents the next simplest process, which can receive e along channel c and then it behaves as P, binding the name received to the name e (characterized by using round brackets).
- $\bar{\mathbf{c}}\langle\mathbf{e}\rangle$**.P** denotes a process that sends e along channel c and then it behaves as P. In this process e is free; this is characterized by the angle brackets.
- **P||Q** represents two processes running in parallel; they may exchange values using input/output channels.
- The term `if` e `then` P `else` Q is the process that behaves as P if e holds; otherwise it behaves as Q.

In order to study how processes interact we need to have a notion of process *reaction*. Before we formally define the reaction relation between processes, we provide an intuitive example that shows a reaction occurring within a process P, leading to a new state P'. This reaction is written $P \to P'$.

Example 3.2.1. Let $P = P_1||P_2$ where $P_1 = \bar{c}.A$ and $P_2 = c.B$. Then $P \equiv \bar{c}.A||c.B$. Note that there is a reaction between $\bar{c}$ and c, we call this reaction a *redex* (we will explain this concept later on this chapter). Thus we have the reaction

$$P \to A||B$$

▲

Example 3.2.1 provides an intuition of the reaction between two processes that communicate. Before formally defining how the different constructs presented in Figure 3.1 react, we need to formally define when two processes are structurally congruent, i.e., we need to define the meaning of the expression $P \equiv Q$. The following lemma is the building block for Definition 3.2.1.

Lemma 3.2.1. *In the context of MPADS, the order of execution of two (or more) independent decisions is irrelevant.*

Definition 3.2.1 (Structural Congruence)**.** Two processes P and Q in the MPADS are structurally congruent, written $P \equiv Q$, if we can transform P into Q or vice versa by using the following equations:

- $P||0 \equiv P$
- $P||Q \equiv Q||P$
- $P||(Q||R) \equiv (P||Q)||R$

□

The following lemma is an easy consequence of the previous definition. We need to point out that such lemma is in the (restricted) context of the MPADS. We will extend the scope of this result in later sections of this chapter.

Lemma 3.2.2. *In the context of MPADS, if two decisions systems are congruent, then they have the same set of alternatives.*

Proof . Assuming two MPADS-processes P and Q, a more general result of this lemma can be stated as: If $P \equiv Q$ then $fn(P) = fn(Q)$, where $fn(P)$ and $fn(Q)$ are the sets of free names of process P and Q, respectively. To prove this, it is enough to show that this result holds separately for each of the equations of Definition 3.2.1. Indeed,

1. If $\boldsymbol{P||0 \equiv P}$, then $fn(P||0) = fn(P)$

2. If $\boldsymbol{P||Q} \equiv \boldsymbol{Q||P}$, then $fn(P||Q) = fn(P) \cup fn(Q) = fn(Q) \cup fn(P) = fn(Q||P)$.

3. if $\boldsymbol{P||(Q||R)} \equiv \boldsymbol{(P||Q)||R}$, then

$$\begin{aligned} fn(P||(Q||R)) &= fn(P) \cup fn(Q||R) \\ &= fn(P) \cup fn(Q) \cup fn(R) \\ &= fn(P||Q) \cup fn(R) \\ &= fn((P||Q)||R \end{aligned}$$

■

We can now proceed to formally define the reaction rules of the MPADS. Note that the structural congruence between two processes is used in the last rule. The rules are to be read as follows: if the transition above the inference line holds (or can be inferred), then we can infer the transition below the line. The symbol $\downarrow$ means that we can infer that the left-hand side of $\downarrow$ evaluates to the right-hand side. For example, $e \downarrow true$ means that we can infer that the expression e evaluates to *true*.

Definition 3.2.2 (Reaction)**.** The reaction relation *to* over the processes defined on Figure 3.1 contains the transition which can be inferred from the following rules:

Decision [DES]:

$$\frac{e \downarrow \text{true}}{\texttt{if } e \texttt{ then } P \texttt{ else } Q \ \rightarrow P}$$

$$\frac{e \downarrow \text{false}}{\texttt{if } e \texttt{ then } P \texttt{ else } Q \rightarrow Q}$$

Parallel composition [P-COMP]:

$$\frac{P \rightarrow P'}{P \ || \ Q \rightarrow P' \ || \ Q}$$

Reaction [REACT]:

$$\frac{}{\bar{c}\langle e \rangle.P \ || \ c(e').Q \rightarrow P \ || \ \{^{e}/_{e'}\}Q}$$

Structural congruence [STRUC]: If $P \equiv Q$ and $P' \equiv Q'$

$$\frac{P \rightarrow P'}{Q \rightarrow Q'}$$

□

Example 3.2.2. In general terms, the Simplex method finds the basic feasible solution (BFS) having the maximum (or minimum) objective value z^*. In any BFS, the method compares the current objective value with that of its neighboring BFS's. If any of the neighbors has a higher (or lower) objective value, it updates the basis with the basis of that with the higher (or lower) objective value. Figure 3.2 shows the feasible region of an arbitrary linear programming problem (LPP), for some c, A and b:

$$\begin{aligned} \text{Maximize} \quad & z = c^T \boldsymbol{x} \\ \text{s.t.} \quad & \\ A\boldsymbol{x} \quad & \leq b \\ \boldsymbol{x} \quad & \geq 0 \end{aligned}$$

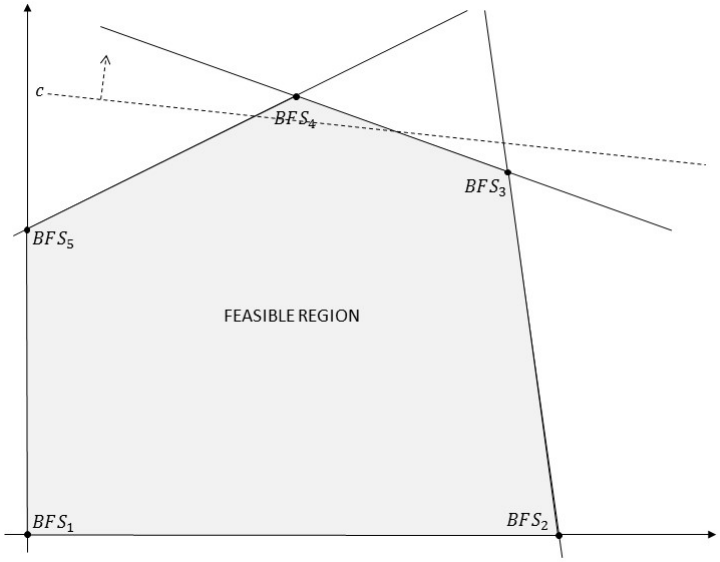


Figure 3.2. Feasible region of an arbitrary LPP.

It is obvious from the figure that the objective value z^* is reached at BFS_4. Let us assume that there exists a process $Basis_{update}$ that updates the current basis to

the new basis, whenever this is found. At any stage of the method, the decision of the methods is characterized by the following process:

$$Simplex\langle z_{current}, z_{new}\rangle = \bar{c}\langle z_{new}\rangle.0 \mid\mid c(e).\texttt{if } z_{current} \leq e \texttt{ then } Basis_{update} \texttt{ else } 0 \tag{3.1}$$

The process $Simplex$ has two parameters, $z_{current}, z_{new}$, and two concurrent processes:

- "$P = \bar{c}\langle z_{new}\rangle.0$" which can be written as "$\bar{c}\langle z_{new}\rangle$", and
- "$Q = c(e).\texttt{if } z_{current} \leq e \texttt{ then } Basis_{update} \texttt{ else } 0$ " which can be written as "$c(e).\texttt{if } z_{current} \leq e \texttt{ then } Basis_{update}$ ".

Process P simply sends z_{new} through channel c. Process Q receives an expression e through channel c, via the rule [REACT]. Then it proceeds as $\texttt{if } z_{current} \leq e \texttt{ then } Basis_{update}$ where all instances of e are substituted by the expression received through channel c, in this case z_{new}. Then, if $z_{current} \leq z_{new}$ is true, the basis is updated through the presumed process $Basis_{update}$, otherwise it just terminates. Let us assume that the method is currently on BFS_3. Obviously the objective value $z_3 < z_4$. Then using [DES] the following reaction occurs:

$$\bar{c}\langle z_4\rangle \mid\mid c(e).\texttt{if } z_{current} \leq e \texttt{ then } Basis_{update} \texttt{ else } 0 \longrightarrow Basis_{update}$$

Or simply,

$$Simplex\langle z_3, z_4\rangle \rightarrow Basis_{update}$$

As described above, we can omit the parameters of a process. Therefore, the process $Simplex\langle z_{current}, z_{new}\rangle$ could have been written as $Simplex$ throughout the example.

▲

3.2.3 Recursion

In Section 2.2.1 we studied automata having (potentially) infinite sequences of a single element (or multiple elements) of the alphabet Σ. These sequences are represented by the so-called *Kleen Star*, *; e.g. a^* denotes any non-negative number

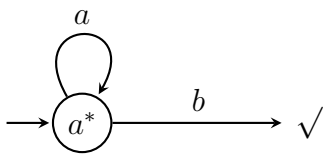


Figure 3.3. Pictorial representation of the expression a^*b

(including zero) of symbols a. Figure 3.3 shows a process $a * b$ that chooses between a and b, and upon termination of a it has the same choice over again; the symbol $\surd$ denotes successful termination. The Kleen Star is considered the most fundamental recursive operator [163].

So far, using the syntax of the MPADS to describe processes similar than the one depicted on Figure 3.3 is not possible; constraining the expressiveness of the language because recursion is indispensable in order to describe repetitive behavior [164]. In this section we propose an extension of this syntax to allow the definition of recursive process expressions. The following example shows a situation where recursion is necessary to successfully describe a simple decision system.

Example 3.2.3 (Vending machine)**.** Consider a simple vending machine that sells a single type of soda for one token. The operation of the machine is as follows: After the insertion of a token by a customer, a soda is dispensed. Assume that there exists a Boolean expression *token* that characterizes the insertion of a token ($token = true$ if a token is inserted) and an atomic action *dispatch* that characterizes the dispatch of the soda. Then, using the syntax of the MPADS, a simple characterization of this machine, called VM, could be given as follows:

$$VM = if\ token\ then\ dispatch \tag{3.2}$$

Assuming that a token is inserted in the machine, one problem with the description represented by (3.2) is that after a soda is dispatched, no more sodas will be handed out ever again. A more accurate description of the vending machine would be the following equation:

$$VM = if\ token\ then\ dispatch.VM \tag{3.3}$$

The decision system described in (3.3), namely the vending machine, is represented by a process VM that occurs not only as the left-hand side of the equation but also again in the right-hand side of it. Such equation is called a *recursive equation.* ▲

There are different ways to extend the syntax of a process theory to admit recursive definitions. Milner [127] proposed the operator '!', called *replication operator*. If P is a π-calculus process, then $!P$ is a process that can replicate itself an infinite number of times, i.e., $!P \equiv P|!P$. Hennessy [154] uses 'rec x. B' as a mechanism for defining recursive processes for the distributed asynchronous π-calculus, aDpi. In that expression, B is a process in the aDpi calculus, called the body of the recursive definition, and x the recursive variable. The occurrences of x within B stand for recursive calls. The reduction of this recursive processes is as follows:

$$\text{rec } x.\ B \rightarrow B\{^{\text{rec } x.\ B}/_{x}\}$$

We would like to extend the MPADS to allow recursive definitions, we will call this theory MPADS$_{rec}$. We will adopt the recursive mechanism similar than that proposed by Hennessy in [154]. To that end, we need to equip our process algebra with two processes:

1. A process X, called *process variable*, that can adopt the behavior of other processes.

2. A process $X \boxdot S$, called *recursive definition process*, defined as in Definition 3.2.3.

Definition 3.2.3 (Recursive Mechanism)**.** Let X be a guarded process variable [1] and P a process in the context of MPADS$_{rec}$. Then, the process $X \boxdot P$ is the process that binds the free occurrences of X in P. We shall often use the notation $X \stackrel{\text{def}}{=} P$ instead of $X \boxdot P$ for simplicity. □

[1]By *guarded process variable* we mean that the value of this variable is "protected" by the value that another variable my take; e.g. in the process `if` e `then` X `else` Y, the variables X and Y are guarded by the expression e.

The **reduction semantics** of the recursive mechanism described in Definition 3.2.3 is characterized by the reduction rule [REC] as follows:

$$X \boxdot P \longrightarrow \{^{X \boxdot P}/_{X}\} \tag{3.4}$$

Now we shall use the MPADS$_{rec}$ to define (3.3) of the vending machine example in a recursive fashion as follows:

$$VM \boxdot if\ token\ then\ dispatch.VM, or$$

$$VM \stackrel{\text{def}}{=} if\ token\ then\ dispatch.VM$$

Meaning that each free occurrence of VM in $if\ token\ then\ dispatch.VM$ will be substituted by VM. Note that VM is a guarded process variable guarded by *token*. This process will dispatch a soda every time a token is inserted; as it is expected by a vending machine.

The process algebra MPADS$_{rec}$ has increased the expressiveness of MPADS to allow recursive definitions. This extension allows us to specify decision systems where decision are not only single-shot, but multiple decision. In the following section we want to relax the constraint of having a single decision maker by extending our algebra to allow the use of sequential of processes.

3.2.4 Sequential Composition

Up to this point, our algebra MPADS$_{rec}$ can combine processes by means of the parallel operator $||$. For the specification of more complex decision systems, additional mechanisms of composition are useful; e.g. a decision system where the order in which decision are made is essential for the specification of the system. In this section we extend the MPADS$_{rec}$ by means of a *sequential composition* operator.

Definition 3.2.4 (Sequential Operator)**.** Given two process expressions P and Q of the MPADS, the term $P \cdot Q$ denotes the sequential composition of P and Q. □

The intuition of this operation is that upon the successful termination of P, the process Q is started. If P does not terminates or ends in a *deadlock* (a process that is not terminal and there is possible transition to other processes), also the sequential composition $P \cdot Q$ does not terminate or deadlocks, respectively. We can state this formally: The **reduction semantics** of the sequential operator of Definition 3.2.4 is characterized by the reduction rule [S-COMP] as follows (remember that $\surd$ denotes that P terminates successfully):

$$\frac{P \to P'}{P \cdot Q \to P' \cdot Q}$$

$$\frac{P \to \surd}{P \cdot Q \to Q}$$

With the sequential operator our theory is able to specify decision systems comprised by more than one decision maker whose decisions are made based upon comparison of values. In the next section we equip this theory with other constructs that will allow us to describe more complex decision systems. The MPADS with all theses extensions yields to our Calculus for Decision Systems (CDS).

3.2.5 From MPADS to CDS

The syntax of the minimum process algebra for decision systems, MPADS, defined on Figure 3.1 has been extended to allow recursion (see Definition 3.2.3) and sequential processes (see Definition 3.2.4). These extensions improved its expressiveness to describe decision systems with recursive properties and with more than one decision maker. In this section we present four more extensions to MPADS that will yield to the Calculus for Decision System, CDS. First, we present two well known constructs in the theory of process algebras, the "+" and the "ν" operators. Then, we extend the set of terminal processes to allow values as terminal nodes. Lastly, we present a new prefix which allows us to evaluate a function with the values received through communication channels.

The "+" and "ν" operators: There are two constructs that are standard in most of process calculi. These constructs will add the ability to specify internal choice in a decision system, and the capacity of bounding the scope of names within a process term. Informally, these constructs are defined as follows:

- Given two processes P and Q, the process $P + Q$, called the "internal choice" process, is a process that randomly chooses to behave either as P or Q. For example: Assume process P and Q, and channel c. Then the process $\bar{c} \mid\mid cP + cQ$ produces, concurrently, two reactions, one to P and the other to Q. i.e., $\bar{c} \mid\mid cP + cQ \longrightarrow P$, and $\bar{c} \mid\mid cP + cQ \longrightarrow Q$.

- Given a process P and a channel name c, the process $(\nu c)P$, called "restriction", is a process that guarantees that channel c is a fresh channel in P. Sometimes we write this process as "*new* c" or "νc" (without parenthesis). This process binds more tightly than *internal choice* and *composition* processes (sequential and parallel); for example $(\nu c)P||Q$ means $(\nu c\ P)||Q$, not $(\nu c)(P||Q)$. This process allows us to define the scope of a name. For example, assuming that channel a is unknown to P and Q, in the process $P \mid\mid (\nu a)Q$, channel a is known to Q but not to P. This process adds multiple benefits to a process algebra, in our particular case, this process will be useful when we define the chain of command and the reachability of information in a organizational structure.

Formally, we can define these processes – in the context of MPADS – as in Definitions 3.2.5 and 3.2.6.

Definition 3.2.5. Given two MPADS-processes P and Q. The *internal choice* process $P + Q$ represents a decision system which may behave either as process P or as Q. □

The process $P + Q$ enables all the current actions of P and all the current actions of Q. After P terminates in $P + Q$, the process term Q is discarded, and vice versa.

This is reflected in the following reduction semantics rule [I-CHOICE]:

$$\frac{}{P + Q \rightarrow P} \qquad \frac{}{P + Q \rightarrow Q}$$

The process term $(\nu c)P$ allows us to define the scope of a name. For example, assuming that channel a is unknown to P and Q, in the process $P \mid\mid (\nu a)Q$, channel a is known to Q but not to P.

Definition 3.2.6. Given a MPADS-process P and a channel c, the *fresh channel creation* process, $(\nu c)P$, is a process that bounds the name c in P. i.e. $c \notin fn(P)$. □

This process adds multiple benefits to a process algebra. In our particular case, this process will by useful when we characterize the chain of command and the reachability of information within an organizational structure.

Termination Processes: Definitions 3.2.5 and 3.2.6 present two constructs that are known to add expressiveness to many process calculi. We have discussed how these process terms will be of use in the specification of decision systems. Another technicality that needs to be addressed in order to increase the capabilities of our process algebra is related to the terminal processes.

In Section 3.2.2 we defined the process **0** as the termination process of MPADS. This process is necessary and useful, but limited in the context of decision systems because some decision systems, such as a game theoretical problems, may be reduced to a value not to a process. Therefore, we need to extend the set of terminal processes to the processes **0** and v, where $v \in Val$. This change is reflected in Figure 3.3.

Function Evaluation: In game theory and utility theory decision makers are characterized by their utility function. Neumann and Morgenstern proved that any individual whose preferences satisfy a set of four axioms (Completeness, transitivity, continuity and independence; see [165]) have a utility function. This function captures the order of preferences of an individual or company. Definition 2.1.2 characterizes decision makers of a function that relates internal and external information to produce an output (i.e., a decision).

So far, we have been able to specify some decision systems using MPADS. These decision systems have had the constraint of having decision makers who make decisions based upon simple comparisons between values. We have not discussed a decision system in which the decision maker makes his decisions based on a slightly more sophisticated function such as the one presented in Example 2.2(c). In order to specify this type of decision systems, we first need to define a set of function names $Func = \{f_1, f_2, \dots\}$. In the most simple case, any element of $Func$, is a predefined function of one variable, say x; i.e., $f_i = f_i(x)$ for some $i \in \mathcal{I}$; where $\mathcal{I}$ is some index set. We will extend these functions to multiple variables in Section 3.5. Now we can define the new prefix which allows communication and function evaluation.

Definition 3.2.7. Assume a process P and a channel c. Let $f \in Func$ be a decision function such that $f(e) \downarrow e'$ where $e, e' \in Exp$. Then, assuming that e is received though channel c, the process $cf[x].P$ binds occurrences of x in P, replacing x with the expression e' and then behaves as P. □

The prefix introduced in Definition 3.2.7 is similar to the *β-reduction* of λ-calculus [2](see [116]). This construct captures the idea of *function application* in the context of MPADS. This is reflected in the following reduction semantics rule:

$$\frac{f(e) \downarrow e'}{\bar{c}\langle e\rangle.P \parallel cf[y].Q \longrightarrow P \parallel \{^{e'}/_{y}\}Q} \tag{3.5}$$

Note that (3.5) may introduce some ambiguity to the expressions of the CDS, because e' may introduce new names. Therefore, this rule can defined as follows:

$$\frac{f(e) \downarrow v}{\bar{c}\langle e\rangle.P \parallel cf[y].Q \longrightarrow P \parallel \{^{v}/_{y}\}Q} \tag{3.6}$$

where v is a value. However, in some decision systems, the outcome can be an string, as will be shown in Example 3.2.4. Thus, we assume that decision functions

[2]In λ-calculus, the β-reduction –in general terms – captures the concept of function application. The computation rule of the β-reduction is as follows: $(\lambda x.t)\ E \longrightarrow \{^{E}/_{x}\}t$, where t is an expression of x and E and application term such as a value; e.g., $(\lambda x.x^2)\ 2 \longrightarrow 4$. For a proper introduction to λ-calculus, we refer to [116].

will be defined in a way such that the output of the function is a value. Therefore, the rule [FUNC-A] is defined according to (3.6).

Example 3.2.4. Let f be a decision function defined as follows:

$$f(x) = \begin{cases} Buy & x < \epsilon \\ Sell & x \geq \epsilon \end{cases}$$

for some threshold ϵ. Then, the following process will "print" the decision made, depending on the values of ϵ and x on the definition of f:

$$P = \overline{c}\langle x \rangle.0 \mid\mid cf[y].print(y).0$$

where $print(y)$ is a terminal process that returns y. i.e. $print(y) \longrightarrow y$. Now, let $\epsilon = 0.7$ and $x = 0.3$. Since $f(0.3) \downarrow Buy$ when $\epsilon = 0.7$, then – using [FUNC-A] – the process reacts as follows:

$$P = \overline{c}\langle 0.3 \rangle.0 \mid\mid cf[y].print(y) \longrightarrow Buy$$

▲

Example 3.2.5. Assume that we want to characterize the decision system depicted in Figure 2.2(c) of Chapter 1. As mentioned before, the decision maker makes his decisions according to (2.1). Then, the following process, called *transformer*, carries out the decision made by this decision system:

$$transformer = \overline{c}\langle V_{in} \rangle \mid\mid cV_{out}(y).print(y)$$

Let $V_{in} = 220V, n_1 = 320$ and $n_2 = 80$, then – using [FUNC-A] – the process P reacts as follows:

$$transformer = \overline{c}\langle 220 \rangle \mid\mid cf[y].print(y) \longrightarrow 55$$

which is the output voltage of the transformer provided the given characteristics.

▲

It is tempting to add recursive definitions to Example 3.2.5 in order to obtain a more meaningful characterization of the response of the decision system. However, we will perform this type of characterizations in Chapter 6.

So far we have presented several constructs that have extended our MPADS process algebra. If we include all the proposed extensions of Definitions 3.2.3, 3.2.4, 3.2.5, 3.2.6, 3.2.7, and include $v \in Val$ as a terminal process to the syntax of MPADS, it yields a new process calculus, called *Calculus for Decision Systems* (CDS). In the next sections we present its syntax, reduction semantics, other extensions and concepts that will allow us to specify more complex decision systems.

3.3 Syntax of the CDS

The syntax of the *calculus for decision systems* is given in Figure 3.4. It presupposes a set $Func = \{f_1, f_2, \dots\}$ of predefined decision functions of one or multiple variables, and a set *Exp* of expressions, ranged over by $e_1, e_2, \dots$ that can be transmitted via communication channels. These expressions will include expressions of the type Boolean, integer, string, etc., ranged over by the sets *Alt* and *Val*, but more importantly channel names themselves. The operation $\oslash$ is any binary Boolean operation ($\wedge, \vee$, etc.), $\oplus$ is any binary arithmetic operation ($+, -$, etc.), and $\ominus$ any relational operator ($=$, $<$, $\geq$, etc.). Note that none of these operators ($\oslash$, $\oplus$, and $\ominus$) add semantics to the calculus.

The names of the syntactic constructs and their intended interpretations are presented in some detail below.

Nil (0): This is the simplest process in the CDS. It represents a terminal and inert process that does nothing. This process is usually omitted after an action, writing for example $\overline{b}\langle x\rangle.0$ as $\overline{b}\langle x\rangle$.

Value (v): The is the second simplest process of the syntax. This process represents a value as a terminal process. This is becomes useful in certain decision systems such as game theoretical problems that terminate in a value.

$$
\begin{array}{ll}
x, y, z, \ldots & ::= \text{Alternatives names.} \\
v_1, v_2, v_3, \ldots & ::= \text{Values.} \\
f_1, f_2, f_3, \ldots & ::= \text{Decision function names.} \\
c_1, c_2, c_3, \ldots & ::= \text{Channels names.} \\
e \in exp & ::= c|x|v|e \oslash e'|e \oplus e'|e \ominus e'|\neg e| - e|e = true|e = false \ldots \\
\psi\text{: Prefixes} & ::= c(e) \mid \bar{c}\langle e \rangle \mid cf[e] \\
P, Q, X\text{: Processes} & ::= \mathbf{0} \mid v \mid X \mid \psi.P \mid P \cdot Q \mid P + Q \\
 & \quad if\ e\ then\ P\ else\ Q \mid P||Q \mid X \boxdot P \mid (\nu c)P
\end{array}
$$

Figure 3.4. Syntax of CDS.

Process Variable (X): This is the process that can adopt the behavior of other processes. This process is necessary for defining recursion and it is assumed that all the occurrences of X are guarded by some expression.

Prefix ($\psi.P$): Prefix is the basic mechanism by which all the behaviors of the components are constructed. The term $\psi.P$ carries out the action ψ and then it behaves as process P. In the case where $\psi = c(e)$, the process $c(e).P$, called *Input*, receives the expression e along channel c and then it behaves as P. The process $\bar{c}\langle e \rangle.P$, called *output*, is the process that can transmit the expression e along channel c and then it behaves as P. It is not a convention, but customary to use round brackets to denote *input* and angle brackets to denote *output*. The process $cf[e].P$, called *function application*, is the process that binds occurrences of e in P, replacing e with the result of evaluating f in the expression received through channel c, i.e., if $\hat{e}$ is received though c and $f(\hat{e}) \downarrow v$, then the process $cf[e].P$ binds occurrences of e in P, replacing e with v and then it behaves as P.

Sequential ($P \cdot Q$): The process term $P \cdot Q$ represents a decision system that behaves as P and once it terminates, it behaves as Q.

Choice ($P + Q$): The process $P + Q$ represent a decision system which may behave either as process P or as process Q. The set of actions of $P + Q$ is the union between the set of actions of P and the set of actions of Q. Note that this is consistent even in the case where P and Q are the same process; i.e., $P + P \longrightarrow P$. This operator assumes that both processes are competing for the same resource. Thus, when one process is selected, the other is discarded.

Decision (if e then P else Q): This process represents a binary decision; if the expression e holds, then it behaves as P, otherwise, if the expression e does not hold, then it behaves as Q.

Parallel ($P \mid\mid Q$): This process represents a decision system comprised by two processes running in parallel. These processes may interchange information (values, alternatives, etc.) using the processes *input* and *output*.

Recursion ($X \boxdot P$): The process term $X \boxdot P$ is a mechanism for defining recursive processes. Here X is a guarded process variable. $X \boxdot P$ binds the free occurrences of X in the process P. As mentioned before, oftentimes we shall use $\stackrel{\text{def}}{=}$ instead of $\boxdot$ for transition and simplicity.

Restriction ($(\nu c)P$): The *restriction* process $(\nu c)P$ represents a process that restricts the scope of c to P. It is also interpreted as a process that guarantees that channel c is "new" or "fresh" in P. We shall use the expression *new* c and νc interchangeably. The operator ν (or *new*) binds more tightly than the composition operators $+, \cdot$ and $||$.

Example

Before presenting the operational semantics of the CDS, we present two examples that illustrate the reaction between processes. Even though in the previous section

we have presented the reaction rules for the different extension of MPADS (which not constitute the CDS). Since we have not formally presented the operational semantics of the CDS, we will utilize the reaction rules of MPADS presented in Definition 3.2.2. Assume that we want to use the CDS to characterize the simple decision system depicted on Figure 3.5 making use of communication channels. Also, assume that *attack* and *hide* are processes that perform some action, and *visible* is an binary expression that can take the values *yes* or *no*.

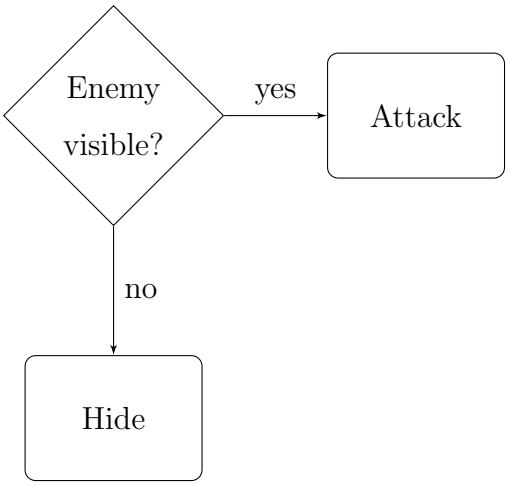


Figure 3.5. Simple Decision System

This decision system can be easily characterized by the process

$$\textbf{if } visible \textbf{ then } Attack \textbf{ else } Hide$$

However, we want to exemplify the use of communication channels. Therefore, we will use a communication channel c to send and receive the information flowing in this system. At first, the system sends the decision about whether or not the enemy is visible. This is characterized by the process $\bar{c}\langle visible\rangle$. Concurrently we have two processes waiting to receive information through channel c so that they can proceed to behave as either *Attack* or $Hide$. These two processes are $c(yes).Attack$ and

$c(no).Hide$. Therefore, the process, P, that characterizes the flow of information of this decision system is given by

$$P = \bar{c}\langle visible? \rangle \mid\mid c(yes).Attack + c(no).Hide$$

Even though the process P captures the flow of information of the decision system, it does not capture the decision itself. It will be shown later that this process may be reduced to the processes *Attack* or $Hide$ without any particular order. i.e., it will randomly reduce to *Attack* or $Hide$ regardless of whether the enemy is visible or not.

It is said that two actions are *complementary* whenever they utilize the same channel. In the previous example, the processes $\bar{c}\langle visible \rangle$ and $c(yes).Attack$ are complementary. If these actions are not alternative to each other, then they constitute a *redex*, resulting in a reaction $P \longrightarrow P'$, which invokes a substitution $\{^{visible}/_{yes}\}$ (where all the instances of *yes* are substituted by *visible*) which is not necessary here since the resulting processes *Attack* and $Hide$ have no instantiations of *yes* or *no*, respectively. Therefore, in the previous example there are two redexes: the pair $(\bar{c}\langle visible \rangle, c(yes).Attack)$ and the pair $(\bar{c}\langle visible \rangle, c(no).Hide)$. So there are two possible reactions $P \longrightarrow P'_1$ and $P \longrightarrow P'_2$, where $P'_1 = \mathbf{0} \mid\mid Attack$ and $P'_2 = \mathbf{0} \mid\mid Hide$, or simply $P'_1 = Attack$ and $P'_2 = Hide$.

In order to use all the constructs of the syntax of the CDS presented in Figure 3.4, we need to formally define how these constructs react. Before presenting these formalizations we want to formalize the idea of two processes being structurally congruent.

Definition 3.3.1 (Structural Congruence)**.** Two processes P and Q are structurally congruent in the CDS, written $P \equiv Q$, if we can transform P into Q, or vice versa, by using the equations in Figure 3.6. □

There are several results that can be proved using the rules presented in Figure 3.6. For example, if a name is not in the set of free names of a process, meaning that it is already being used in the processes, then constraining the scope of that name to that process does not cause any changes in that process. In a different angle, if we

$$P||Q \equiv Q||P,\ P||\mathbf{0} \equiv P$$

$$(\nu cc')P \equiv (\nu c'c)P$$

$$(\nu c)\mathbf{0} \equiv \mathbf{0},\ \ (\nu c)v \equiv v,\ \ (\nu c)v \equiv v$$

$$P + Q \equiv Q + P$$

$$(P||Q)||R \equiv P||(Q||R)$$

$$P + (Q + R) \equiv (P + Q) + R$$

$$P \cdot (Q \cdot R) \equiv (P \cdot Q) \cdot R$$

$$(\nu c)(P \cdot Q) \equiv (\nu c)P \cdot Q \text{ if } c \text{ is not a free name of } Q$$

$$(\nu c)(P||Q) \equiv (\nu c)P||Q \text{ if } c \text{ is not a free name of } Q$$

Figure 3.6. Structural Congruence of the CDS.

assume that certain information is carried out by a channel, say c, that belongs to an arbitrary decision system P, constraining the scope of c to the decision system P is redundant, since such information is already known by P. We can write this formally in the following corollary.

Corollary 3.3.1. *If c is not a free name in P, then $(\nu c)P \equiv P$.*

Proof . Assume c is not a free name in P; i.e., $c \notin fn(P)$. Then c is bound by the scope of P. Therefore $(\nu c)P \equiv (\nu c)(P \parallel \mathbf{0}) \equiv P \parallel (\nu c)\mathbf{0} \equiv P \parallel \mathbf{0} \equiv P$. ■

The next theorem is an extension of Lemma 3.2.2 to CDS-processes.

Theorem 3.3.1. *Two decision systems are structurally equivalent, if they possess the same alternatives.*

Proof . Use Lemma 3.2.2 and apply the same procedure to the rules of Figure 3.6 that are not in Definition 3.2.1. ■

The structural congruence will be needed when defining the reaction rules of the CDS. This set of rules is known as the operational semantics.

3.4 Operational Semantics of the CDS

The Operational Semantics is given by the reduction relation defined by a binary relation $\rightarrow$ between closed terms or processes. The expression

$$P \rightarrow Q$$

intuitively means that in one computation step, the process P is reduced to Q. A computation from P to Q then consists of any arbitrary number of such steps from P that eventually may lead to Q. This is denoted by $\rightarrow^*$ (the reflexive transitive closure of $\rightarrow$). Thus, the expression

$$P \rightarrow^* Q$$

denotes that the process P may lead to the process Q after many steps. The definition presents the operational semantics of the CDS, which formalizes the reaction rules of the CDS. Similar than in the previous section, note that the structural congruence between two processes is utilized in the last rule of the operational semantics.

Definition 3.4.1 (Reduction Semantics)**.** The reduction semantics of the CDS is the smallest relation on processes generated by the following reduction rules ($e \downarrow v$ denotes that the expression e evaluates to values v.):

Internal Choice [I-CHOICE]:

$$\frac{}{P + Q \rightarrow P} \qquad \frac{}{P + Q \rightarrow Q}$$

Sequential Composition [S-COMP] ($\surd$ denotes that P terminates successfully):

$$\frac{P \longrightarrow P'}{P \cdot Q \longrightarrow P' \cdot Q} \qquad \frac{P \longrightarrow \surd}{P \cdot Q \longrightarrow Q}$$

Restriction [RES]:

$$\frac{P \longrightarrow Q}{(\nu c)P \longrightarrow (\nu c)Q}$$

Decision [DES]:

$$\frac{e \downarrow \ true}{if\ e\ then\ P\ else\ Q \longrightarrow P} \qquad \frac{e \downarrow \ false}{if\ e\ then\ P\ else\ Q \longrightarrow Q}$$

Parallel composition [P-COMP]:

$$\frac{P \longrightarrow P'}{P \mid\mid Q \longrightarrow P' \mid\mid Q}$$

Reaction [REACT]:

$$\frac{}{\bar{c}\langle e\rangle.P \mid\mid c(e').Q \longrightarrow P \mid\mid \{^{e}/_{e'}\}Q}$$

Function Application [FUNC-A]:

$$\frac{f(e) \downarrow v}{\bar{c}\langle e\rangle.P \mid\mid cf[y].Q \longrightarrow P \mid\mid \{^{v}/_{y}\}Q}$$

Recursion [REC]:

$$\frac{}{X \circledcirc P \longrightarrow \{^{X \circledcirc P}/_{X}\}P}$$

Structural congruence [STRUC]: If $P \equiv Q$ and $P' \equiv Q'$

$$\frac{P \longrightarrow P'}{Q \longrightarrow Q'}$$

□

Having the operational semantics defined allows us to use the CDS almost in its full capacity.There is an small constraint to the current language that needs to be addressed: The CDS is monoadic; i.e., we can only send one name at a time through a channel, constraining our capacity to specify decision systems having tuples of information instead of singletons. Fortunately this is a technicality that can be resolved without much harm, and will be done in Section 3.5. In the meantime we present some examples to show how to use some of the CDS-constructs.

3.4.1 Examples

Example 3.4.1. (Channel re-utilization) This example shows how a name received on a channel can itself be used as an output (or input) channel. We will use channel c to receive the input x, and then it will be used to output the value 5:

$$\bar{c}\langle x\rangle \mid\mid c(y).\bar{y}\langle 5\rangle$$

This process is reduced to $\bar{x}\langle 5\rangle$. ▲

Example 3.4.2. (Value Passing) A value can be interpreted as a channel name. This allows channels to send and receive values. Assume we want to send the value 3 along the channel c. This process (and its reduction) is given by:

$$\bar{c}\langle 3\rangle \mid\mid c(x).\bar{y}\langle x\rangle \rightarrow (\bar{y}\langle x\rangle)\{^{3}/_{x}\} = \bar{y}\langle 3\rangle$$

▲

Example 3.4.3. (Scope Extrusion) A restricted channel name can be sent outside of its original scope. In this example, channel y is sent on channel c outside the scope of the binder νy. This process (and its reductions) are as follows:

$$\begin{aligned}
&(\nu y)(\bar{c}\langle y\rangle \mid\mid y(x).if\ (x = 7)\ then\ P\ else\ Q) \mid\mid c(u).\bar{u}\langle 7\rangle \\
&\longrightarrow (\nu y)(y(x).if\ (x = 7)\ then\ P\ else\ Q \mid\mid \overline{y}\langle 7\rangle) \\
&\longrightarrow (\nu y)(if\ (x = 7)\ then\ P\ else\ Q)\{^{7}/_{x}\} \\
&\longrightarrow (\nu y)P
\end{aligned}$$

▲

Example 3.4.4. (Function Evaluation) Assume that we have a decision function $f(x) = x^2$. We can send 3 along channel c and evaluate the decision function f.

$$\bar{c}\langle 3\rangle \mid\mid cf[x].x \longrightarrow 9$$

▲

As mentioned above, these examples showed how to use some of the most important constructs of the CDS. We can now proceed to address the previously mentioned technicality regarding the communication of tuples through channels. We call this extension the polyadic-CDS.

3.5 The polyadic CDS

Multiple decision can be made during a single decision system, therefore we clearly wish to have the ability of sending messages consisting of more than one name (i.e. tuples of names). In this section we present an straightforward extension that allows multiple objects during communication between channels. This will extend the CDS to allow outputs of type $\overline{c}\langle e_1, \ldots, e_n\rangle.P$ and inputs of type $c(e_1, \ldots, e_n).Q$. This extension is called the polyadic-CDS.

To extend the CDS to its polyadic version we will use the encoding proposed by Milner in [127, p.93]. This encoding only requires to send a fresh name along the communication channel and then send the tuple of names one by one along the new name. i.e.,

$$c(y_1, \ldots, y_n).P \longmapsto c(w).w(y_1).\ldots.w(y_n).P \tag{3.7}$$

$$\overline{c}\langle z_1, \ldots, z_n\rangle.Q \longmapsto new\ w\,(\overline{c}\langle w\rangle.\overline{w}\langle z_1\rangle.\ldots.\overline{w}\langle z_n\rangle.Q) \tag{3.8}$$

This extension suggests us to make the following assumptions:

1. Whenever we send tuples of names instead of singletons, the communication will be done utilizing the polyadic CDS. This is for simplicity of notation.

2. The arity of the output is the same than the arity of the input; this also holds for [FUNC-A]. If this assumption does not hold, we would need a type system to detect it. Creating a type system for the CDS is out of the scope of this dissertation.

The following example shows how to pass tuples of values instead of singletons through communication channels.

Example 3.5.1. (Function Application-polyadic) Assume that we have a decision function $f(x_1, x_2) = (x_1^2, x_2^3)$. We can send the pair (2,3) along channel c and evaluate the decision function f.

$$\overline{c}\langle 2, 3\rangle \mid\mid cf[x].x \longrightarrow (4, 27)$$

▲

During the rest of this document, we will use the *polyadic CDS* to admit multiple action prefixes. Note that the empty action prefixes $\overline{c}\langle\rangle$ and $c()$ are simply written $\overline{c}$ and c, respectively. We are now in position to prove more interesting results. First, we state the following (trivial) lemma.

Lemma 3.5.1. *If A is a square $n \times n$ matrix, and $\boldsymbol{x}$ and $\boldsymbol{b}$ two n-dimensional vectors, then $A\boldsymbol{x} \leq \boldsymbol{b} \equiv f(\boldsymbol{x}) \leq \boldsymbol{b}$, for some $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))^T$.*

Using Lemma 3.5.1 and the polyadic version of the CDS we can state the following result.

Theorem 3.5.1. *Let A be a square $n \times n$ matrix, and $\boldsymbol{x}$ and $\boldsymbol{b}$ two n-dimensional vectors. Let $\boldsymbol{v}$ be a sequence of values of arity n such that $v_i = x_i$ for all i. Then, the process "$\overline{c}\langle\boldsymbol{v}\rangle \mid\mid cf[\boldsymbol{y}].\texttt{if } \bigwedge \boldsymbol{y} \leq \boldsymbol{b} \texttt{ then } \boldsymbol{v}$ ", is reduced to a* feasible solution *(FS) of the program* min $\boldsymbol{c}^T\boldsymbol{x}$, *s.t. $A\boldsymbol{x} \leq \boldsymbol{b}$, where $\boldsymbol{c}$ is an n-dimensional vector.*

Proof . First, let P and Q be two CDS-process terms. Using 3.8 we can write $\overline{c}\langle z_1, \ldots, z_n\rangle.P$ as *new* $w\,(\overline{c}\langle w\rangle.\overline{w}\langle z_1\rangle.\cdots.\overline{w}\langle z_n\rangle.P)$. We can perform the same encoding such that $z_i = \boldsymbol{v}$ for $i = 1 \ldots n$, where $\boldsymbol{v}$ is a sequence of values of arity n; we can denote this by $\overline{c}\langle\boldsymbol{v}\rangle.P$. Second, assume a sequence of decision functions $f(\boldsymbol{y}) = (f_1(\boldsymbol{y}), f_2(\boldsymbol{y}), \ldots, f_n(\boldsymbol{y}))$, then, by similar encoding to the one presented on (3.7), the process $cf(\boldsymbol{y}).Q$ can be written as $c(w).wf_1(y_1).\cdots.wf_n(y_n).Q$. Third, a vector $\boldsymbol{v}$ is a FS of the program min $c^T\boldsymbol{x}$, s.t. $A\boldsymbol{x} \leq \boldsymbol{b}$, if $A\boldsymbol{v} \leq \boldsymbol{b}$. If $\boldsymbol{y}$ and $\boldsymbol{b}$ are two sequences of values of arity n, then the expression $\bigwedge \boldsymbol{y} \leq \boldsymbol{b}$ evaluates to the Boolean expression *true* if $y_i \leq b_i$ for all $i = 1, \ldots, n$. Using the latter fact, together with the assumptions of the theorem, the former encoding for the processes $\overline{c}\langle\boldsymbol{v}\rangle.P$ and $cf[\boldsymbol{y}].Q$, and Lemma 3.5.1, the result falls immediately by applying the reduction rule [DES]. ■

Using Theorem 3.5.1 we can specify the set of all feasible solutions, and therefore –assuming that the objective function has a minimum (or maximum) on the feasible region– find the optimal value of the program min $\boldsymbol{c}^T\boldsymbol{x}$, s.t. $A\boldsymbol{x} \leq \boldsymbol{b}$, which is the

smallest (or highest in the case of maximization) value of the function $z = \boldsymbol{c}^T\boldsymbol{x}$, such that $\boldsymbol{x}$ is a basic feasible solution. Now we present a more sophisticated example that make use of many of the constructs of the CDS for the specification of a more complex decision system.

3.6 Example: Aircraft Acquisition Problem

In this section we use the CDS to specify the decision system described in [166]. This system is interesting because of the proposed decomposition strategy of the monolithic optimization problem. The rest of this section is organized as follows: first, we present the definition of the problem. Second, we exhibit the strategy proposed by the authors. Third, we characterize the system using the CDS. Lastly, we present some of the conclusions drawn from the specification of this system using our calculus.

Problem Statement and Formulation

The problem statement has been mostly extracted from [166]:

> Aviation fuel contributes the largest percentage of energy consumption in the Department of Defense (DoD), with the Air Mobility Command (AMC) being single largest consumer. AMCs mission profile mainly consists of worldwide cargo and passenger transport, air refueling and aeromedical evacuation. Platforms in operation include C-5 Galaxy and C-17 Globemaster III for long range strategic missions, C-130 Hercules for tactical missions, KC-135 Stratotanker and KC-10 Extender for aerial refueling missions, and various VIP transport platforms including Air Force One. AMC also charters aircraft from Civil Reserve Air Fleet during peacetime contractually committed from U.S. airlines.
>
> The complex logistics involved in the transportation of various cargos across the AMCs service network requires efficient deployment of its fleet of cargo aircraft in meeting daily cargo delivery requirements, while

minimizing fuel consumption and subsequent costs. These fuel costs are naturally driven by the choice of aircraft design and individual flight legs flown by the AMC fleet, in meeting cargo obligations within a prescribed schedule timeframe. The design of the aircraft itself, operations across routes flown and manifestation of uncertainty in daily cargo transportation demand creates a highly complex hierarchy of interwoven systems or a 'system-of-systems'.

The AMC is in the process of modernizing the current strategic fleet, consisting of C-5s and C-17s, by incorporating new materials and engines on existing airframes to operate the current fleet more efficiently. However, the design of new, more fuel efficient aircraft may potentially provide the biggest cost and fuel consumption savings. The uncertain nature of AMC operations, coupled with its complex logistics, makes it difficult to identify a fuel efficient aircraft designs that achieves target performance, while simultaneously minimizing fuel consumption across the range of day-to-day operational scenarios. With the many variables available to a systems designer, a computational approach becomes necessary to determine which variables to change and determine the magnitude of change to satisfy constraints while minimizing or maximizing an objective (or multiple objectives). Solutions obtained from properly formulated optimization problems provide insight into decisions about new systems and help to inform acquisition decisions.

The monolithic problem is formulated as a blend of stochastic integer programming and non-linear programming. In involves resource allocation under uncertainty and aircraft design perspectives. In this example we focus on the decomposition strategy taken by the authors to solve the program formulated on [166, p.3-4]. We disregard the uncertainty aspect of the problem for simplicity, but given the simplicity of the formulation, extending it to include uncertainty should not be of great difficulty. We now present the decomposition strategy of the original problem.

Problem Decomposition Strategy

The decomposition strategy is comprised of three mayor blocks:

1. Top Level (TL): In this block, the requirements of the problem are formulated. The objective is to minimize the expected fleet Direct Operating Cost (DOC) using the pallet capacity and range of the new aircraft of type X. This problem is addressed using a simple enumeration scheme.

2. Aircraft Sizing problem (AS): In this block, the problem of the design of the new aircraft is formulated. The objective is to minimize DOC subject to the take-off distance, which is a function of a set of design variables. This problem uses as input the pallet capacity ($Pallet_X$) and range ($Range_X$) calculated on the top-level block.

3. Air Mobility Command problem (AMC): In this block, the output of the aircraft sizing problem (pallet capacity) and top-level problem (operation costs) are used as input of a problem that seeks to minimize the DOC subject to the pallet capacity and trip limits.

The decomposition structure, taken directly from [166], is presented in Figure 3.7.

Enumeration Scheme: This simple enumeration scheme is used by the authors to find the minimum DOC in the top-level optimization problem. The idea is to compare the minimum expected DOC calculated in the AMC block with minimum expected DOC calculated in the previous iteration. The minimum expected DOC of the top-level block is then the minimum among them. If the value of the minimum expected DOC calculated during the current iteration and the one computed in the previous iteration match, then a counter is increased by 1. If a value is repeated multiple times, say N, then the program stops and selects that value to be the optimal value for the top-level optimization problem.

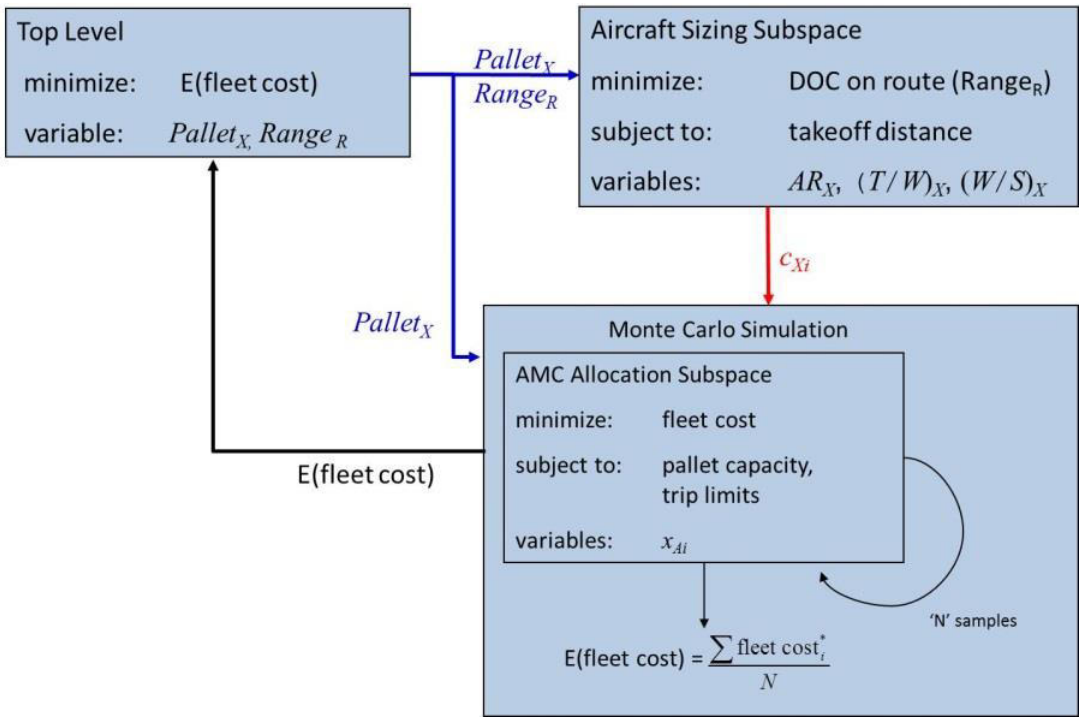


Figure 3.7. Decomposition of the (monolithic) aircraft acquisition problem.

Using the CDS for the specification of the Aircraft Acquisition Problem

Before using the CDS for the specification of this problem, we want to illustrate the suitability of this problem into the definition of a decision system (see Definition 2.1.2). There are three important aspects that need to be highlighted:

1. The initial requirements of the yet-to-be designed aircraft are specified to the problem by its environment; i.e., the initial requirements are the external information of the decision system. Given the enumeration scheme utilized for solving the top-level optimization problem, these initial requirements are directly fed into the aircraft sizing problem (AS).

2. There are three interactive subsystems; i.e. three interactive decision makers, each of them with internal and external information used to make their decisions.

3. The decision to be made by the decision system is: What is the pallet capacity and range of the new aircraft, so that the expected DOC is minimum?

The state diagram presented on Figure 3.8 shows our conceptualization of the problem. It shows the conditions that need to be satisfied in order to go from one state to the other. The labels inside of angle brackets, $\langle\ \rangle$, represent the information that is to be sent from one state to the other. Table 3.1 presents labels used in the state diagram, together with their meanings.

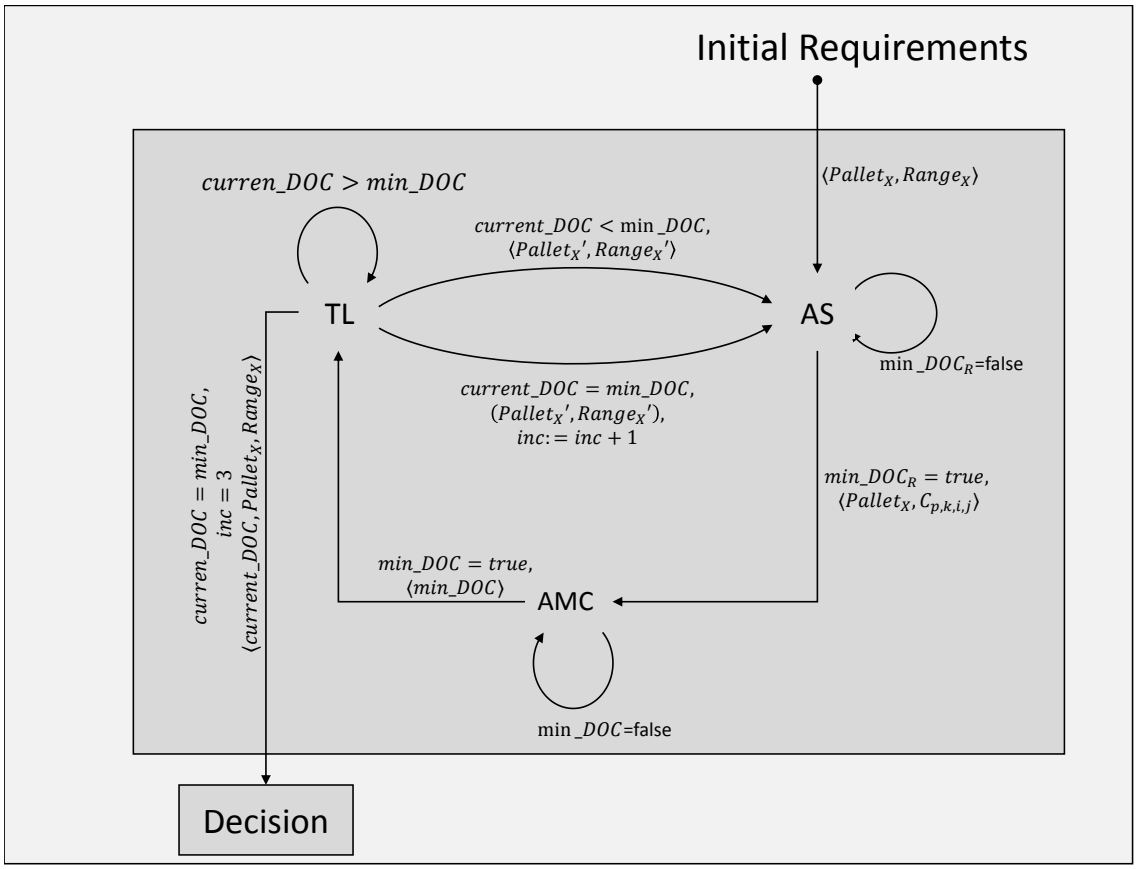


Figure 3.8. State Diagram of the aircraft acquisition problem presented in [166]

.

Using the state diagram presented in Figure 3.8 and some of the labels defined on Table 3.1, we can describe the aircraft acquisition problem in the CDS. For simplicity of notation we use P_X, R_X and C_X instead of $Pallet_X$, $Range_X$ and $C_{p,k,i,j}$,

Table 3.1
List of labels used on the aircraft acquisition problem.

Label	**Meaning**
TL	Top-level problem.
AS	Aircraft Sizing problem.
AMC	Air Mobility Command problem.
current_DOC	Value of the current DOC (this is used as a result of the enumeration scheme implemented by the authors)
min_DOC	Value of the minimum DOC; when this label is equal to *true* (or $false$) in the diagram, it means that such value is indeed the minimum. This mix of types between values and Boolean expressions is just for illustration purposes of the state diagram. In the characterization of the problem using the CDS, we avoid that confusion.
min_DOC_R	Boolean expression that becomes "true" when the value of the DOC subject to the $range_X$ is the minimum.
inc	This is a variable that has increments of 1 every time it is called.
$Pallet_X$	This is the pallet capacity of aircraft of type X.
$Pallet'_X$	This is the new pallet capacity after incrementing its value according to the enumeration scheme.
$Range_X$	This is the range of aircraft of type X.
$Range'_X$	This is the new range after incrementing its value according to the enumeration scheme.
$C_{p,k,i,j}$	These are the operation costs of the aircraft.

respectively. We shall start by defining an auxiliary process, call $Assign\langle x, x', P\rangle$, that assigns the value x' in process P to the value x; assuming that $x \in fn(P)$.

$$Assign\langle x, x', P\rangle = new\ x(\overline{c}\langle x'\rangle \mid\mid c(x).P) \tag{3.9}$$

Assume the atomic action *inc* that increments in 1 every time it is called. This can be defined similar than the process `incr` of Example 2.2.8. Using *inc*, (3.9), and Figure 3.8 we can now define the process TL:

$$\begin{aligned} TL \stackrel{\text{def}}{=}\ & c(min\_DOC).if\ (current\_DOC = 0)\ \vee\ (current\_DOC > min\_DOC)\ then \\ & Assign\langle current\_DOC, min\_DOC, TL\rangle\ else \\ & if\ current\_DOC < min\_DOC\ then\ \overline{c_1}\langle P'_X, R'_X\rangle.AS\ else \\ & if\ current\_DOC = min\_DOC\ then \\ & if\ inc = N\ then\ decision(current\_DOC, P_X, R_X)\ else \\ & inc.\overline{c_1}\langle P'_X, R'_X\rangle.AS \end{aligned} \tag{3.10}$$

where N is the number of times previously selected as the threshold to conclude that min_DOCis the optimal value of the top-level optimization problem (in Figure 3.8, we assumed $N = 3$). Let FDV be the sequence of Feasible Design Variables for the optimization problem defined in the AS block. Then, the process AS is defined as follows:

$$AS = c_1(P_X, R_X).new\ w(\overline{w}\langle FDV, P_X, R_X\rangle \mid\mid OP_{AS}) \tag{3.11}$$

where,

$$OP_{AS} \stackrel{\text{def}}{=} wf[x].if\ x \leq x'\ then\ \texttt{Estimate}\ else\ Assign\langle x', x\rangle.\overline{w}\langle FDV', P_X, R_X\rangle.OP_{AS}$$

and,

$$\texttt{Estimate} = cost\_estimator\langle FDV\rangle \mid\mid a(C_X).\overline{c_2}\langle P_X, C_X\rangle.AMC \tag{3.12}$$

where, $cost\_estimator\langle x\rangle$ returns, through channel a, the operating costs of aircraft with design variables x. Let FAV be the feasible allocation variables for

the optimization problem defined in the block AMS. Also, let $f(FAV, P_X, C_X) = DOC(FAV, P_X, C_X)$. Then the AMC process is defined as follows:

$$AMC = c_2(P_X, C_X).new\ w(\overline{w}\langle FAV, P_X, C_X\rangle \mid\mid OP_{AMC}) \tag{3.13}$$

where,

$$OP_{AMC} \stackrel{\text{def}}{=} wf[x].if\ x \leq x'\ then\ \overline{c}\langle x\rangle.TL\ else\ Assign\langle x', x\rangle.\overline{w}\langle FAV', P_X, C_x\rangle.OP_{AMC} \tag{3.14}$$

Then, the process that characterizes the aircraft acquisition problem is

$$new\ cc_1c_2\ \ (\overline{c_1}\langle P_0, R_0\rangle \mid\mid TL \mid\mid AS \mid\mid AMC).$$

Discussion

The aircraft acquisition problem presented in this section provides an interesting decomposition methodology of an optimization problem. Using the CDS we were able to characterize the behavior of this system-of-systems, and assure that the communication between the different blocks (TL, AS, and AMC) is consistent. Using the CDS provided the following insights regarding this system:

1. Even though the TL problem is the control system, it comes into play until the end of the first iteration of the optimization process; i.e. the enumeration scheme utilized in the TL problem requires the comparison of two computed fleet costs (DOC's), therefore the system has to be initially fed through the AS problem (as depicted in Figure 3.8).

2. All communication channels used to share information had the pallet capacity as part of the shared tuples. If we were to allocate resources to accurately estimate the parameters needed for the successful solution of this problem, the pallet capacity of the aircraft would have to have high priority.

3. The aircraft allocation problem, represented by the AMC block, requires both the pallet capacity ($Pallet_X$) and the operation costs ($C_{p,k,i,j}$) in order to find the minimum DOC. Thus, feeding the AMC block with the pallet capacity of the TL block is not necessary. The pallet capacity can be fed from the AS block to the AMC block.

These insights become more important as the estimation of the parameters becomes more expensive. The impact of the insights that can be potentially gathered using formal tools such as process calculi – in particular the CDS – increases when the system can be modified and tested for comparison with the original system. To that end, we need to have a formal framework for comparing the behavior of two systems. This formal framework is known in process calculi theory as Bisimulation. In the next section we briefly discuss the concept of behavioral equivalence in the context of decision systems and the CDS.

3.7 Behavioral Equivalence for Decision Systems

In automata theory, two automata are said to be equivalent if they recognize the same language. Sometimes these automata may recognize the same language, but the behavior (or paths) adopted by each string (or word) in order to be recognized by the automaton may be different; i.e., these automata are behaviorally different. In process algebra, there exists an instrument to compare two systems from the behavioral stand point. This apparatus is called Bisimulation, and oftentimes it is refereed to as the Hallmark of process calculi.

In the next chapter we investigate in depth the notion of behavioral equivalence in the context of decision systems. We start by defining the concept of a labelled transition system (LTS), which constitutes the basis for behavioral equivalence. This concept will be studied from the process algebra perspective, and they define it in the context of decision systems. Having the LTS well defined, we introduce the concepts of simulation and strong bisimulation in the context of process algebras and then we

provide the extension to CDS. The concept of strong bisimulation can be utilized to study behavioral equivalence between two decision systems, allowing us to compare not only two decision systems, but also a decision system against its modified version.

3.8 The CDS versus other Calculi in the context of Decision Systems

In this section we compare other calculi in the context of decision systems. As discussed in Chapter 2, decision systems are naturally classified by the number of decision makers (single or multiple decision makers) and by their interaction with their environment (if the decision system interacts with its environment, it is said to be an open decision system; otherwise it is a closed decision system).

In the case of a single decision maker, we have discussed that there is an endless number of systems that fit into the definition of a decision system. Given the vast number of modeling and analysis tools for these type of systems, even stating such tools seems an impossible task. However, we can certainly argue that process algebras are a natural choice of formal framework for reasoning about the structure and behavior of (concurrent) systems. The ability to compare two systems by their behavior rather than their structure or their output is a capability not included in any other approach outside of the formal methods area; making these methods a proper tool for behavioral comparison over any other tool. The CDS was based upon the definition of a decision system, therefore it is equipped with constructs that make the calculus a natural choice for decision systems. In the particular case of a single decision maker, the system will have sets of input and output information, and the decision are made based upon the function that characterizes the decision maker. In this case, the CDS utilizes the *input* and *output* constructs to gather and share the input and output information, and makes use of the *function application* construct to withdraw decision based on the inputted information. The key difference between the CDS and other calculi in this particular case, relays on the latter construct, the *function application*. Even though the *application* rule of λ-calculus is similar –in

principle – to the *function application* construct of the CDS, the latter allows us to evaluate a broader type of functions in a more natural way than the former. The polyadic version of the CDS allows for evaluation of multivariable functions, and the flexibility in the type of expressions allowed in the *input* and *output* constructs allow the evaluation of functions with cases that return not only numbers, but expressions of the type string or Boolean.

In Chapter 2 we presented different theories for the case where there are two or more decision makers in a decision system. All these theories – in one way or another – have improved the analysis and specification of game theoretical problems. As discussed in Section 2.2.2, most of these theories are based on modal operators (or modal logic) for reasoning about the structure of games. Even though these approaches are great logical theories, they lack communication capabilities to share information between processes, constraining the scope of application to only game theoretical problems, which is acceptable since it is one of the goals of those theories. However, in the case of decision systems, we need to have more flexibility in the type of systems we want to reason about. We need the ability to reason about decision systems comprised by system that belong to a domain different than game theory; e.g. a game between a computer and a physical system. Therefore, we need to have the information sharing capabilities of a process algebra, in addition to constructs that support the characterization of systems other than those provided by game theory. In Section 2.2.3 we presented different process calculi for the analysis of different type of systems. All these calculi provide different perspectives, advantages and disadvantages over the CDS. In fact, some of them include aspects that could increase the expressiveness of the CDS in the context of decision systems; e.g., probabilistic choice. The main differences between the CDS and other process calculi are the following (the justification of the necessity of the following characteristics was discussed at the beginning of this chapter):

1. A set of decision functions that characterizes the utility (or payoff) functions of the decision makers.

2. A construct that allows to evaluate a function with the value (or expression) received through certain communication channel.

3. Have the capability of sending expressions through communication channels, providing great flexibility to the *function application* construct.

4. A theory that has all the previous characteristics combined with the operators: *Sequential*, *Choice*, *Decision*, *Parallel*, and the *Nil* and *Value* constructs.

5. A well defined notion of behavioral equivalence of decision systems.

There are particular differences, advantages and disadvantages between other calculi and the CDS that will be discussed in Chapters 5 and 6. Before exploring these differences and applications of the CDS to different areas, we need to formalize the notion of behavioral equivalence of decision systems. Chapter 4 is devoted to such definition. Then, we will turn our attention to the application of our calculus to game theoretical problems, in particular games in extensive form. To finalize this chapter, we provide a summary of the most relevant results and concepts that were presented.

Discussion

If we could develop a well defined general theory for the study of systems, we would be able to not only describe and compare systems but also to apply approaches developed for one domain to another domain. Moreover, similar to the general idea of dynamic programming of solving different parts of a problem separately and then find the general solution, we could use a formal language, such as the CDS, to describe two or more interacting sub-systems belonging to different domains, and then combining them to find the overall output of the system. To that end, we first need a well define (formal) language to describe and compare (decision) systems; this language is the CDS.

As we have shown in Chapter 2 there are multiple calculi used for different purposes. If, for example, we want to compare the behavior of one system which has been

described in one calculus, with another system described using the CDS, we could use the notion of bisimulation. Moreover, if there is a particular interest in a set of systems described using another calculus, we may be able to encode such calculus in the CDS and then use the comparison mechanism presented in this dissertation to gather information about the similarities in behavior between the systems. This will allow us to study systems that interact with each other, but the underlining principles of them lie on different domains.

3.9 Summary

In this chapter we presented the *Calculus for Decision Systems*. This process calculus uses some of the ideas proposed by Milner in the popular π-calculus. We started the chapter by defining a minimal process algebra for decision systems called MPADS. This process algebra had some limitations, but it was shown to be expressive enough to characterize simple binary decisions. The structural congruence and operational semantics of the MPADS was presented. Using the MPADS as the ground base, we presented several extensions that increased the expressiveness this process algebra that eventually lead to the CDS. Once the syntax of the CDS was presented, we defined the structural congruence, reaction rules and the polyadic version of the calculus. Having the CDS well defined, we proved that every feasible solution of the program min $\boldsymbol{c}^T\boldsymbol{x}$, s.t. $A\boldsymbol{x} \leq \boldsymbol{b}$ has CDS-term that characterizes it. After presenting multiple examples of the utilization of the reaction rules, we utilized our calculus to characterize the aircraft acquisition problem presented in [166]. We withdrew some basic insights of the problem using the characterization of the behavior in the CDS. At the end of the chapter we briefly discussed the behavioral equivalence of decision systems using the CDS. We conclude the chapter with a general comparison between the CDS and other calculi in the context of decision systems.

4. BEHAVIORAL EQUIVALENCE

4.1 Introduction

In this chapter we develop the notion of behavioral equivalence in the context of decision systems. The concept of behavioral equivalence was originally developed by Park in [5], and it is considered the hallmark of process algebras. Informally, two processes are considered to be behavioral equivalent when their externally observed behavior appear to be the same. In the context of decision systems, two decision systems may have the same outcome (or reach the same decision), but the path taken in order to reach such outcome could have been drastically different; making decisions – for example – more expensive or inefficient. Therefore, having a mechanism to differentiate when two decision systems are behaviorally different is important.

According to [142], there are three different classes of entity-to-entity equivalence in a modeling study: system-to-model equivalence, model-to-model equivalence and state-to-state equivalence. In process algebras, all modeling entities – system, model and states– are represented by agents or processes, therefore, as it is argued in [142], the notion of bisimulation can capture all three of these equivalences. In the case of decision systems, we represent decision systems by processes, without differentiating between a system, a model or a state. Therefore, the notion of bisimulation will suffice for the study of behavioral equivalence.

Before we formally define the concept of bisimulation in the context of decision systems, let us to introduce this mechanism, together with the necessary background concepts, in the context of process algebras. Therefore, we first define the notion of bisimulation in the context of process algebras in Section 4.2, and then we present, in Section 4.3, the preliminary concepts needed in order to extend the notion of bisimulation to decision systems. In Section 4.4, we present the formal definition of

bisimulation in the context of the calculus of decision systems. Before summarizing the chapter in Section 4.5, in Section 5.6 we present an example to show the insights that bisimulation provides in the context of games in extensive form.

4.2 Process Algebras and Bisimulation

The theory of process algebras has advanced significantly since the creation of the CCS, CSP and ACP. In this section we introduce two of the most important concepts relevant to the theory of process algebras: *labeled transition system*, and *Bisimulation*.

4.2.1 Labeled Transition Systems

A **Labelled Transition System** or LTS informally speaking is similar than an automaton without the set of initial and final states. This modification gives us the flexibility to model systems that can start in any state and may finish in any other state. A formal definition of an LTS is presented in Definition 4.2.1; it consists of (1) a collection of states and (2) a collection of transitions between them. The transitions are labelled by actions from a set Act.

Definition 4.2.1 (Labeled Transition System)**.** Let Act be a set of actions. A *labelled transition system* (LTS) over Act is a tuple $(S, \rightarrow)$ with

1. S a set of (finite) states
2. $\rightarrow \subseteq S \times Act \times S$ is a transition relation

□

The transition $s \xrightarrow{a} t$ means that system s can evolve into system t while performing the action a. In a LTS the same label may cause a reaction in more than one transition. If the set of actions is a singleton (i.e. $\overline{\overline{Act}}=1$), then we can assume that the transition system is unlabeled and replace the transition for the silent transition τ. Labelled transition systems are essential to the study of behavioral equivalence of decision systems and processes in general. Before defining the concept of behavioral

equivalence (or bisimulation) we need to discuss the concept of *simulation*. Informally, we say that a state *simulates* another state, if they transition to one or many states using the exact same label(s). Formally, we can state this concept as follows.

Definition 4.2.2 (Simulation)**.** Let $(\mathcal{Q}, \mathcal{T})$ be an LTS, and let $\mathcal{S}$ be a binary relation over $\mathcal{Q}$. Then $\mathcal{S}$ is called a strong simulation over $(\mathcal{Q}, \mathcal{T})$ if, whenever $p\mathcal{S}q$, if $p \xrightarrow{\alpha} p'$ then there exists $q' \in \mathcal{Q}$ such that $q \xrightarrow{\alpha} q'$ and $p'\mathcal{S}q'$. □

4.2.2 Strong Bisimulation

We can now proceed to define the concept of Bisimulation in process algebra theory. Informally, is the mirrored relation of a simulation is also a simulation, we say that there is a bisimulation. Formally, bisimulation is defined in Definition 4.2.3.

Definition 4.2.3 (Bisimulation)**.** A binary relation $\mathcal{S}$ over $\mathcal{Q}$ is said to be a strong bisimulation over the LTS $(\mathcal{Q}, \mathcal{T})$ if both $\mathcal{S}$ and its converse are simulations. We say that p and q are strongly bisimilar, if there exists a strong bisimulation $\mathcal{S}$ such that $p\mathcal{S}q$. □

In loose words, in order to show that there is a bisimulation between two labeled transition systems, say LTS_1 and LTS_2, we need to show that for each node, s_i, in LTS_1 there has to be one node, t_j, in LTS_2 so that $s_i\mathcal{S}t_j$ holds. In the next section we present some examples that clarify this concept.

4.2.3 Examples

Examples 4.1 and 4.2 provide a rather simple situation where bisimulation will detect subtleties in the behavior of two LTS. Example 4.1 shows the existence of a bisimulation between the two LTS, whereas example 4.2 shows the opposite.

Example 4.2.1. Consider the two labeled transition systems depicted in Figure 4.1. One can verify that there exists a bisimulation between LTS 4.1(a) and LTS 4.1(b) by simply verifing that each state of 4.1(a) can be simulated with some state in 4.1(b).

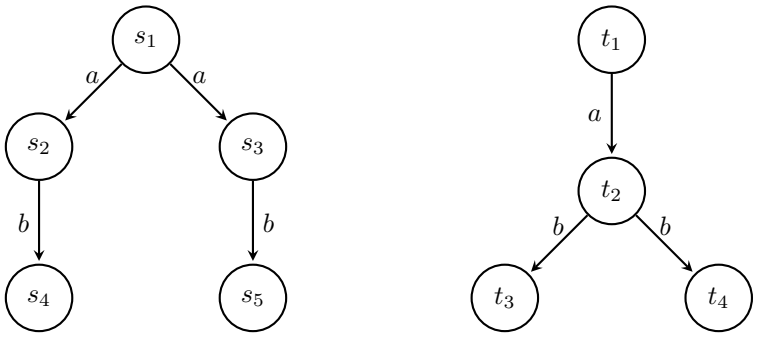


Figure 4.1. Example of a Bisimulation.

Intuitively, automaton 4.1(a) accepts the string $ab + ab$ and automaton 4.1(b) the string $a(b + b)$. ▲

Example 4.2.2. Consider the two labeled transition systems depicted in Figure 4.2. In order to verify if there exists a bisimulation between LTS 4.2(a) and LTS 4.2(b) we need to verify that each state of 4.2(a) can be simulated with some state in 4.2(b) and vice versa. However, we cannot find a node in LTS 4.2(a) that simulates the behavior of node t_2 in LTS 4.2(b), i.e. There is no s_i in LTS 4.2(a), such that $s_i \mathcal{S} t_2$.

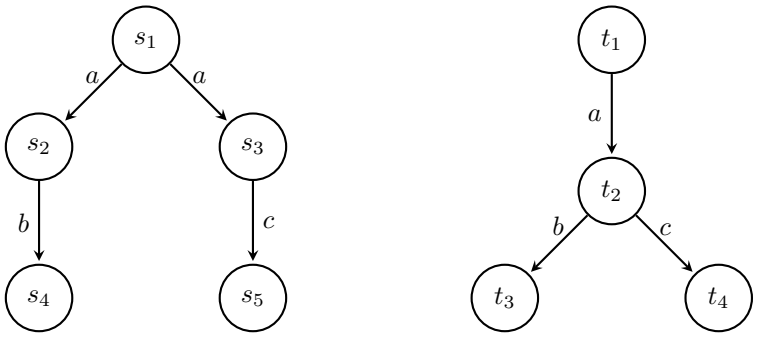


Figure 4.2. Example when there is not a Bisimulation.

Intuitively, automaton 4.2(a) accepts the string $ab + ac$ and automaton 4.2(b) the string $a(b + c)$. ▲

We are now in position to define the concepts of labelled transition systems and bisimulation in the context of decision systems. First, we need to study the concept of

actions in the calculus for decision systems. To that end we will define what actions are feasible in the calculus, and the semantics of those actions in the CDS.

4.3 Action Semantics for the CDS

So far, we have interpreted transitions of CDS-processes as simple reductions, using the reduction semantics presented in Definition 3.4.1. Now we will give a different – more general – view of processes in the context of the CDS, studying reactions as transitions. We start by defining the possible transitions we can have in the CDS, and then we present the action semantics of the calculus. Using this action semantics, we can view CDS-processes as labelled transition systems. After defining the action semantics for the CDS, we can proceed to study the notion of bisimulation in this context. We now define the process transitions $P \xrightarrow{\alpha} Q$ which will extend the reactions $P \to Q$ defined in the previous chapter.

- $P \xrightarrow{\bar{c}\langle e\rangle} Q$ resembles the ability of process P to send the expression e along c, provided that another process, running in parallel, can perform the complementary transition. The process Q represents the resulting change in the process.

- $P \xrightarrow{c(e)} Q$ resembles the ability of process P to receive the expression e along c, with the residual process Q representing the resulting change in the process. Again, provided that another process, running in parallel, can perform the complementary transition. *N.B.:* The use of the prefix $\psi = cf[x]$ is a special case of this transition; this becomes clear in Definition 4.3.1.

- $P \xrightarrow{\tau} Q$; the reduction of P to Q by an arbitrary internal activity. This resembles a reaction following the rules of the operational semantics defined on Definition 3.4.1.

Notation: We use the symbol γ to denote an action that ranges over the set of all input actions, and $\bar{\gamma}$ to denote an action that ranges over the set of all output

Table 4.1
Action semantics for the CDS

Name	**Transition Rule**	
IN_t	$c(e).P \xrightarrow{c(e')} \{^{e'}/_{e}\}P$	
OUT_t	$\bar{c}\langle e\rangle.P \xrightarrow{\bar{c}\langle e\rangle} P$	
SEQ_t	$\dfrac{P \xrightarrow{\alpha} P'}{P \cdot Q \xrightarrow{\alpha} P' \cdot Q}$	
DES_t	`if` e `then` P `else` Q$\xrightarrow{\tau} P$	$e \downarrow true$
	`if` e `then` P `else` Q$\xrightarrow{\tau} Q$	$e \downarrow false$
RES_t	$\dfrac{P \xrightarrow{\alpha} Q}{(\nu c)P \xrightarrow{\alpha} (\nu c)Q}$	c is not a name in α
REC_t	$X \triangleq P \xrightarrow{\tau} \{^{X \triangleq P}/_{X}\}P$	
$FUNC_t$	$cf[x].P \xrightarrow{c(e)} \{^{v}/_{x}\}P$	$f(e) \downarrow v$
$REACT_t$	$\dfrac{P \xrightarrow{\gamma} P' \wedge Q \xrightarrow{\bar{\gamma}} Q'}{P\|\|Q \xrightarrow{\tau} P'\|\|Q'}$	
$CHOICE_t$	$P + Q \xrightarrow{\tau} P$	
	$P + Q \xrightarrow{\tau} Q$	
$CONCRT_t$	$\dfrac{P \xrightarrow{\alpha} P'}{P\|\|Q \xrightarrow{\alpha} P'\|\|Q \vee Q\|\|P \xrightarrow{\alpha} Q\|\|P'}$	

actions; *N.B.:* Neither γ nor $\bar{\gamma}$ include τ. The label α denotes an action that can range over all possible actions; input, output and it may also include τ.

Definition 4.3.1 (LTS for CDS-processes)**.** Let $\mathcal{L}$ be the set of all labels. Then, a *Labelled Transition System* $(\mathcal{P}, \mathcal{T})$ of the CDS over the set of actions $Act = \mathcal{L} \cup \{\tau\}$ is comprised by a set of states $\mathcal{P}$, that has the possible process expressions, and a set of transitions $\mathcal{T}$ which are exactly those that can be inferred from the rules in Table 4.1. □

Most of the transition rules presented on Table 4.1 need no explanation. The transition rule $FUNC_t$ resembles the ability of process P to receive the expression e along c and substitute instantiations of x by the image of e in the mapping f. The rule $CONCRT_t$ represents both left and right transition of the parallel composition operator.

We are now in position to show how our LTS is related with the structural congruence of the CDS. Informally, we can state that two structurally congruent CDS-processes have essentially the same transitions; i.e., the structural congruence relation $\equiv$ is a strong simulation for (pure) process algebras, as defined in Definition 4.2.2. Thus, structural congruence respects transition. This result will be fundamental when we define the behavioral equivalence between decision systems.

Lemma 4.3.1. *Given two CDS-process P and Q. If $P \equiv Q$ and $P \xrightarrow{\alpha} Q$, then there exists a process Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv Q'$.*

The proof of this result is similar than the one found in [127]. Also, [154] presents a proof where a relation $\mathcal{R}$ is assumed; where $P\mathcal{R}Q$ if whenever $P \xrightarrow{\alpha} Q$ there exists some Q' such that $Q \xrightarrow{\alpha} Q'$, and $P' \equiv Q'$. Then, $\mathcal{R}$ is also an equivalence relation that satisfies the axioms of the structural congruence.

The following results relates *reaction* with the silent transition τ.

Lemma 4.3.2. *Given two CDS-processes P and Q. If $P \longrightarrow Q$, then $P \xrightarrow{\tau} Q'$ for some $Q' \equiv Q$.*

Proof . This result is proved by induction on the definition of the relation $P \longrightarrow Q$. We need to show that the relation $\xrightarrow{\tau}$ satisfies all the axioms and rules of $\longrightarrow$, defined on Definition 3.4.1.

> **Case [I-CHOICE]**: Let $P = R + S$, and the transition $P \xrightarrow{\tau} Q'$ follows – in both of the rules – directly by $[CHOICE_t]$.
>
> **Case [S-COMP]**: Let $P = R \cdot S$ and $Q' = R' \cdot S$, with $R \longrightarrow R'$ by a shorter inference. Then, by induction $R \xrightarrow{\tau} R''$, where $R'' \equiv R'$. The result follows by using $[SEQ_t]$. Use similar argument for the other case.

Case [RES]: Let $P = (\nu c)R$ and $Q' = (\nu c)S$, with $R \longrightarrow S$ by a shorter inference. By induction $R \xrightarrow{\tau} S'$, where $S' \equiv S$. Then, using $[RES_t]$, $(\nu c)R \xrightarrow{\tau} (\nu c)S'$ with $S' \equiv S$.

Case [DES]: Let $P =$ `if` e `then` R `else` S and $Q' = R$ with $e \downarrow true$. Then, the transition $P \xrightarrow{\tau} Q'$ follows directly by $[DES_t]$. Use a similar argument for the case where $e \downarrow false$.

Case [P-COMP]: Use a similar argument than the one used for [S-COMP].

Case [REACT]: Let $P = \overline{c}\langle e\rangle.R \mid\mid c(e').S$ and $Q' = R||\{^{e}/_{e'}\}S$. Then,

$$\dfrac{\dfrac{}{\overline{c}\langle e\rangle.R \xrightarrow{\overline{c}\langle e\rangle} R}\,OUT_t \qquad \dfrac{}{c(e).S \xrightarrow{c(e')} \{^{e}/_{e'}\}S}\,IN_t}{\overline{c}\langle e\rangle.R \mid\mid c(e).S \xrightarrow{\tau} R \mid\mid \{^{e}/_{e'}\}S}\,REACT_t$$

Case [FUNC-A]: Let $P = \overline{c}\langle e\rangle.R \mid\mid cf[y].S$ and $Q' = R \mid\mid \{^{v}/_{y}\}S$ with $f(e) \downarrow v$. Then,

$$\dfrac{\dfrac{}{\overline{c}\langle e\rangle.R \xrightarrow{\overline{c}\langle e\rangle} R}\,OUT_t \qquad \dfrac{f(e) \downarrow v}{cf[y].S \xrightarrow{c(e)} \{^{v}/_{y}\}S}\,FUNC_t}{\overline{c}\langle e\rangle.R \mid\mid cf[y].S \xrightarrow{\tau} R \mid\mid \{^{v}/_{y}\}S}\,REACT_t$$

Case [REC]: Let $P = X \boxdot R$ and $Q' = \{^{X \boxdot P}/_{X}\}R$, this follows straight from $[REC_t]$.

Case [STRUC]: Use a similar argument than the one used for [S-COMP], together with Lemma 4.3.1 to prove the result.

■

The converse of Lemma 4.3.2 is straightforward, since the reduction relation can mimic internal actions. Though some of the transition rules are defined in terms of γ and $\bar{\gamma}$, it can be shown (see [154]) that such transition rules also suggest that if $P \xrightarrow{\tau} Q$ then $P \longrightarrow Q$. Then, we can state the following theorem that suggests that reaction agrees with τ-transition.

Theorem 4.3.1. $P \longrightarrow Q \iff P \xrightarrow{\tau} Q'$ *for some* $Q' \equiv Q$.

Theorem 4.3.1 shows the relationship between the reduction relationship and the LTS via structural congruence. This will allow us to define, in the next section, when two decision systems are behaviorally equivalent.

4.4 Strong Bisimulation for CDS

In this section we present the notion of bisimulation in the context of decision systems. Recall that we provided the definition of strong Bisimulation for (pure) process algebras in Definition 4.2.3. It is worth to mention that the difference between strong bisimulation and weak bisimulation relays on whether the silent transition τ is in the set *Act* (stong bisimulation), or not (weak bisimulation). In the previous section we showed how important the silent transition relation ($\xrightarrow{\tau}$) is, since it is tightly related with reduction relation ($\longrightarrow$).

The definition of bisimulation presented in this chapter aims to be an extension of the definition of strong bisimulation presented in Definition 4.2.3 to the CDS. Informally – in the context of the π-calculus – two agents are strongly bisimilar, if any α action of one can be matched by an α action of the other. Recall that in the syntax of the CDS presented in Figure 3.4, the second syntactic construct is the value construct v. As discussed before, this is a necessary construct since the output of a decision system may be a value. This values are usually influenced by the decision maker, and therefore by the decision function that characterizes such a decision maker. Note that the definition of bisimulation does not impose any restrictions on the decision functions of the decision system, which suggests a problem when dealing with decision systems. For example, making the decision between buying stock for either \$1 or \$2, is different than deciding to buy stock for either \$10 or \$100,000; even though these two decisions will be behaviorally the same. Therefore, we assume that the observer basis his comparisons on the current behavior of the decision systems, assuming that the set of decision functions is the same among the two decision systems.

Definition 4.4.1 (Strong Bisimulation)**.** Let $(\mathcal{Q}, \mathcal{T})$ be an LTS, and let $\mathcal{S}$ be a binary relation over $\mathcal{Q}$. Then $\mathcal{S}$ is called a strong bisimulation over $(\mathcal{Q}, \mathcal{T})$ if, for $(P, Q) \in \mathcal{S}$ and $\alpha \in Act$,

1. $Func(P) = Func(Q)$
2. Whenever $P \xrightarrow{\alpha} P'$ then, for some Q', $Q \xrightarrow{\alpha} Q'$, and $(P', Q') \in \mathcal{S}$;
3. Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P', $P \xrightarrow{\alpha} P'$, and $(P', Q') \in \mathcal{S}$.

□

We may now define when two processes are said to be strongly bisimilar.

Definition 4.4.2. Given two processes P and Q. It is said that P and Q are strongly bisimilar, written $P \sim Q$, if $(P, Q) \in \mathcal{S}$ for some strong bisimulation $\mathcal{S}$. □

Note that if we could relax Assumption 1, then Lemma 4.3.1 suggests that the structural congruence ($\equiv$) will constitute a bisimulation for the CDS. Unfortunately, as it was justified before, this is not possible. However, we can state the following theorem.

Theorem 4.4.1. *If $P \equiv Q$ and $Func(P) = Func(Q)$ then $P \sim Q$.*

Proof . Assuming that $Func(P) = Func(Q)$, the result follows immediately from Lemma 4.3.1. ■

N.B: In Definition 4.4.1 we can observe that any processes is trivially a member of a strong bisimulation, since the identity relation $Id_{\mathcal{Q}} = \{(P, P) | P \in \mathcal{Q}\}$ satisfies all the conditions. Moreover, we can state the following lemma:

Lemma 4.4.1. *$\sim$ is an equivalence relation.*

Proof . The proof follows trivially from Definition 4.4.1. ■

Using Lemma 4.4.1 and Definition 4.4.1, it follows that $\sim$ itself is a strong bisimulation; i.e., $\sim$ is the largest relation satisfying the conditions of a strong bisimulation relation, that is an equivalence relation. In order to show that $P \sim Q$, we have to find a strong bisimulation relation $\mathcal{S}$ such that $P\mathcal{S}Q$. These processes can be lengthy process, since we have to consider all possible transitions of P and Q and their possible actions. Therefore we can define a weaker relation, *strong bisimulation up to* $\sim$, which makes use of the the equivalence classes induced on the transition set of each processes by the $\sim$ relation. Thus, two processes satisfy the relation $\mathcal{S}$ if their actions and the set of decision functions are matched. Formally we can state this relation as follows.

Definition 4.4.3 (Strong Bisimulation up to $\sim$)**.** Let $(\mathcal{Q}, \mathcal{T})$ be an LTS, and let $\mathcal{S}$ be a binary relation over $\mathcal{Q}$. Then $\mathcal{S}$ is called a strong bisimulation up to $\sim$ over $(\mathcal{Q}, \mathcal{T})$ if $P\mathcal{S}Q$ implies, for all $\alpha \in Act$,

1. $Func(P) = Func(Q)$
2. Whenever $P \xrightarrow{\alpha} P'$ then, for some Q', $Q \xrightarrow{\alpha} Q'$, and $P' \sim \mathcal{S} \sim Q'$;
3. Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P', $P \xrightarrow{\alpha} P'$, and $P' \sim \mathcal{S} \sim Q'$.

□

Having the notion of strong bisimulation and strong bisimulation up to $\sim$ well defined, we can compare decision systems from a behavioral stand point. In the next chapter we present a complete example were we use the action semantics of the CDS and the notion of bisimulation. We show how bisimulation provides interesting insights regarding the similarity between decision systems. In Chapter 6 we use the action semantics and the LTS to describe a mechanical system. We now present a summary of what we have presented in this chapter.

4.5 Summary

In this chapter we presented the notion of bisimulation in the context of decision systems and the CDS. We started the chapter with a brief introduction on bisimulation in (pure) process algebras, and the concepts that are related with it. In order to define bisimulation for our calculus we created an action semantics. This action semantics provides the type of transitions that are allowed in the CDS and operational rules for them. We proved the relation between transitions and reductions. In particular we proved that, for any two CDS-terms P and Q, $P \to Q \Leftrightarrow P \xrightarrow{\tau} Q'$ for some $Q' \equiv Q$.

After having the transition system well defined for decision systems, we formally defined the notion of strong bisimulation. This notion of bisimulation aims to be an extension of the notion of bisimulation for the CCS (see [121]). We proved that the relation $\sim$ is an equivalence relation. To take advantage of the properties of the $\sim$ relation, we defined a strong bisimulation up to $\sim$; this will help us to avoid proving, for example, the converse of $P \sim Q$ is also a bisimulation (using symmetry of $\sim$). In the next chapters we provide examples on the use of bisimulation to compare decision systems from the behavioral standpoint.

5. CONCURRENCY AND EXTENSIVE GAMES

5.1 Introduction

In Chapter 2 we discussed the importance of game theory in the study of decision systems when there is more than one decision maker in the system. In this chapter we utilize the CDS to study some game theoretical problems expressed in the so-called extensive game form (or simply, extensive games). It is our goal in this chapter to characterize the interactions between two players within an extensive game, viewed from the decision system standpoint. We assume that the reader has some basic knowledge of game theory. However, in order to put some terminology in context, we provide a short introduction to the concepts of game theory in the framework of extensive games.

The chapter is organized as follows. We begin with a short introduction to game theoretical concepts in the context of extensive games. In Section 5.4 we introduce the CDS as a tool for describing interaction between extensive games, and withdraw some properties of games such as Nash Equilibrium. We also present – in Section 5.4 – some examples and the current limitations of our calculus for the study of extensive games. In Section 5.5 we discuss couplings of extensive games using the CDS. We present an overview of what has been done in the study of game couplings and present some examples using our calculus. Section 5.8 presents the advantages and current limitations of the CDS in the context of extensive games. We finalize the chapter with a summary of the results and conclusions.

5.2 Game Theory and Games in Extensive Form

There are three forms of mathematical abstractions of a game: the extensive, the normal and the characteristic function forms. The main difference not only relays on the form the information is presented, but in the amount of information captured by of the different forms. In this dissertation we are mainly dealing with the first form of a game, the extensive form. For a detailed introduction to the normal and the characteristic function forms we refer to [13]. In this section we provide a short introduction to games in extensive form. We discuss some of the relevant theoretical concepts that are necessary for the specification of extensive games using the CDS. For a more detailed introduction to game theory we refer to [167, 168].

The most popular representation of a game is the *normal form*, also called the *strategic form* of a game. Games expressed in strategic form provide a compact way of describing the aspects of a game – Table 5.1 shows a game in strategic form, where players 1 and 2 have to choose between strategy A and strategy B, which leads to a payoff $p_{i,s}$ where s is the strategy chosen by player i, $i = 1, 2$. However, this representation does not show some subtleties of the game that may lead to interesting analysis of its behavior; e.g. players' turns and information available (or privileged) between players.

Table 5.1
Example of a game in strategic form.

	Choose A	**Choose B**
Choose A	$(p_{1,A}, p_{2,A})$	$(p_{1,A}, p_{2,B})$
Choose B	$(p_{1,B}, p_{2,A})$	$(p_{1,B}, p_{2,B})$

As mentioned before, there is another representation of a game called *extensive form* of a game. When a game is represented in an extensive form, we usually refer to it as *extensive game*. The extensive form of a game captures some of subtleties that are not available in strategic form of the game. – Figure 5.1 shows the typical

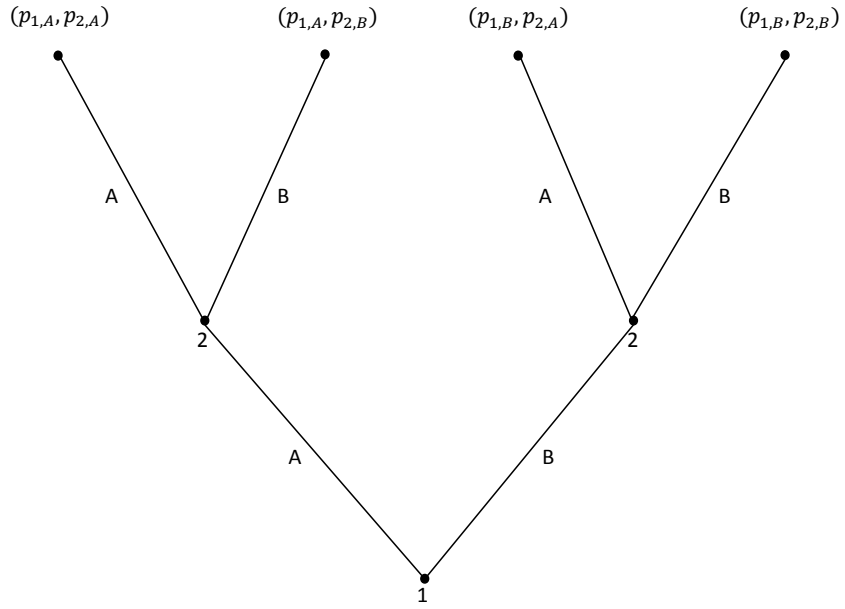


Figure 5.1. Example of a game in extensive form.

representation of an extensive game, where the nodes represent the player, the edges the strategies, and the leaves the payoffs of each path that lead to that payoff (called a *play*); again $p_{i,s}$ is the payoff of a player i, $i = 1, 2$, by playing strategy s. We shall use a slightly different representation of an extensive game throughout this dissertation, see Figure 5.3. There are four important concepts that need to be discussed about extensive games: The concept of a game tree (or game structure), information sets (what players know in their turns), and outcomes (what is the payoff of a play).

5.2.1 Game Trees

Extensive games are modeled using a directed graph, i.e., a pair (T, E), where $T \neq \emptyset$ is a set of vertices (or nodes) and E a set of ordered pairs (or edges) of T. Intuitively, in a game, the vertices represent the position of the game and the arcs the positions that can be reached after that vertex (moves). Also, any path from any

vertex s_0 to a vertex s_i, for some i, is a sequence of vertices (or play) that the game can take. We now present the formal definition of a game tree (a particular type of directed graphs).

Definition 5.2.1. A **tree** is a directed graph, (T, E) where the vertex, s_0, called the root (or the initial vertex), such that for every vertex $s_i \neq s_0 \in E$, there is a unique path starting at s_0 and ending at s_i. □

We say that the game tree is *connected* when there exists a unique path starting from the root and ending in any given terminal node (or leaf) of the game tree. This terminal nodes are the payoff of the game. For a n-person games this payoff is represented by a n-tuple of payoffs. In the particular case of two-person zero-sum game (see Definition 5.3.1), the payoff of player I is what player II loses.

5.2.2 Information Sets

The next concept we need to discuss in extensive games captures the amount of information available for each player on each move. This is called *information sets*. In each turn, a player may or may not know what the other player has done, depending on the information set of that turn. For example, in an arbitrary turn, player I may know the strategy (or move) player II played before him. In the case of games with *complete information*, the information sets are singletons. Meaning that in each turn, each player knows exactly what his or her position on the tree is. In games with *incomplete information*, the cardinality of the information set is greater than one, meaning that a player may not know the stage of the game (i.e., whether the other player has played or not by the time when he/she has to move). This fact is represented by either a dotted line or a dotted enclosing between the nodes that belong to the same information set. Figure 5.2 shows a game with incomplete information, where player 2 does not know whether player 1 played A or B. The information set of player two is the set $\{2_A, 2_B\}$. We shall define a game with complete (or perfect) information in Section 5.3.

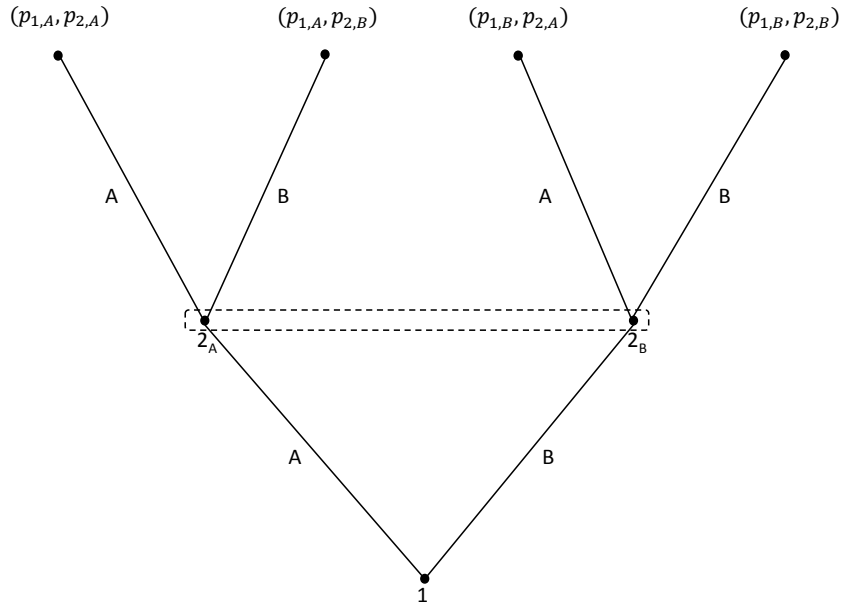


Figure 5.2. Example of a game in extensive form with incomplete information.

5.2.3 Outcomes

The last concept we need discuss regarding a game is the *outcome* of the game, which occurs at the end of each play of the game. The outcome of the game can be represented by the quantification of some kind of reward (or loss). For example the monetary prize received (or lost) in a gamble, or the dissatisfaction or frustration of seeing someone winning what you could have won, or a subjective reward of winning or losing a game. These outcomes are specified a priori in a game, by some sort of function that assigns an outcome to a specific play. Each of the leaves of the game tree represent a possible termination point of the game. Each termination point has an associated outcome with it. Throughout this chapter we use the terms outcome and payoff interchangeably. The set of outcomes of Figure 5.1 is given by the following set:

$$\{(p_{1,A}, p_{2,A}), (p_{1,A}, p_{2,B}), (p_{1,B}, p_{2,A}), (p_{1,B}, p_{2,B})\}$$

After introducing the concepts of game tree, information sets and outcomes, we shall now proceed to define the concept of a two-person, zero-sum extensive game.

5.3 Two-person Zero-sum Extensive Game

Two-person games play an essential role in game theory. This section aims to define some of the crucial concepts for the study of two-person zero-sum extensive games in the context of this dissertation. We do not provide a comprehensive introduction to these concepts, but we define those that are more relevant to the work we develop using our calculus in later sections. In this section we formally define the concepts of two-person zero-sum extensive game, strategy, and subgame perfect equilibrium.

Definition 5.3.1. A **finite two-person zero-sum** extensive game is given by

1. A finite tree with vertices T.

2. A payoff function that assigns a real number to each terminal vertex.

3. A set T_0 of non-terminal vertices (representing positions at which decisions occur) and for each $t \in T_0$, a probability distribution on the edges leading from t.

4. A partition of the rest of the vertices (not terminal and not in T_0) into two groups of information sets $T_{11}, T_{12}, \ldots, T_{1k_1}$ (for Player I) and $T_{21}, T_{22}, \ldots, T_{2k_2}$ (for Player II).

5. For each information set T_{jk}, for player j, a set of labels L_{jk}, and for each $t \in T_{jk}$, a one-to-one mapping of L_{jk} onto the set of edges leading from t.

□

Definition 5.3.2. A game of **perfect information** is an extensive game in which each information set of every player is a singleton. □

In games of perfect information, each player knows all the past moves of the game, including the so-called *chance moves*; i.e., moves made by "nature" or moves made by a player with no strategic interests in the outcome of the game. For example, in tic-tac-toe and chess each player has knowledge of all the past moves of each player during the game, even when they made the first (chance) move to decide who will play first in the game.

We are now in position to present an example of a two-person zero-sum extensive game. The objective of Example 5.3.1 is to show the game tree representation of a two-person zero-sum extensive game.

Example 5.3.1. Lets assume we have two firms. Firm 1 is a monopolist and Firm 2 has the opportunity to enter the market. After Firm 2 enters, Firm 1 will have to either "compete" or "share" the market with Firm 2. If Firm 2 does not enter the market, Firm 1 continues with the monopoly. This situation can be depicted as follows:

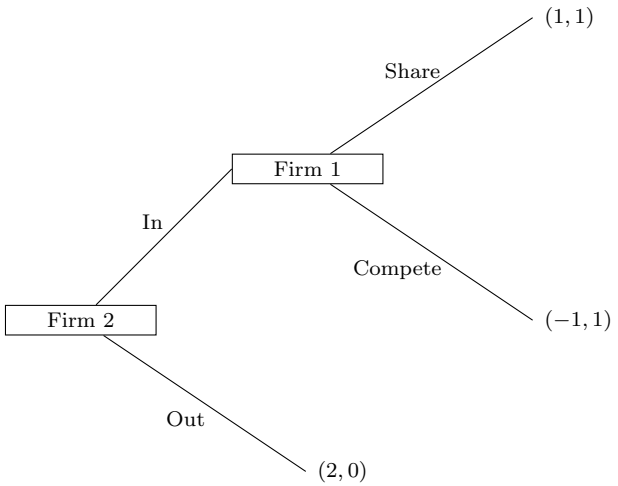


Figure 5.3. Example of a game tree.

▲

We have been discussing different concepts related to games, in particular, we have mentioned several times that players have strategies which lead them to an outcome.

However, we have not formally defined the concept of *strategy*. In loose words, a strategy is a plan that a player will execute in every situation. We have to be careful about differentiating between a strategy and a move. A strategy is a plan of action, given any particular situation throughout the game; i.e., a *strategy* is an algorithm that will tell us what to do if we encounter a particular situation in the game. A *move*, on the other hand, is an action taken by a particular player at some point during the game. We shall now formally define the concept of strategy. There are three types of strategies: *Pure Strategies*, *Mixed Strategies*, and *Behavioral Strategies*. The three concepts diverge in the stochasticity they add to the game.

Definition 5.3.3. A **pure strategy** for player i in an extensive game is a function $s_i : H_i \to A_i$ such that $s_i(h) \in A(h)$ for each $h \in Hi$. Where H_i is the set of information sets at which player i moves, and A_i is the set of actions available to i at any of his information sets. □

The strategy set of a player is the set of pure strategies available to that player in a game. In Example 5.3 Firm 1 has two pure strategies *Compete* and *Share*, and Firm 2, *In* and *Out*. Sometimes a player may be indifferent between the potential outcomes of a game, thus, he/she may decide to randomize between playing one or any of the other strategies of the game. In this case, the player will assign a probability to each of the pure strategies. The resultant (randomized) strategy is called a *mixed strategy*. Formally,

Definition 5.3.4. A **mixed strategy** for player i in an extensive game is a probability distribution (collection of weights) over pure strategies that capture the frequency each move is to be played. □

There is one more type of strategy that instead of assigning a probability to each of the pure strategies, it assigns to each information set a probability distribution over the set of possible actions of the player, injecting more stochasticity than mixed strategies. We refer to this type of strategy as *behavioral strategy*.

We now define the concept of *game of perfect recall*.

Definition 5.3.5. A **game of perfect recall** is a game where:

1. A player never forgets previous decisions made during the game.
2. A player never forgets information he had available for decisions made in the past.

□

In games of perfect recall for any mixed strategy there is an equivalent behavioral strategy and vice versa (Kuhn's Theorem [169]). Therefore, mixed strategies and behavioral strategies are equivalent in this type of games. Throughout this dissertation we deal with games with perfect recall, therefore we will use mixed strategies and behavioral strategies interchangeably.

5.3.1 Nash Equilibrium

Suppose that each player i $(i = 1, \ldots, n)$ chooses a strategy s_i in the extensive game $\mathcal{G}$. The sequence $s = (s_1, \ldots, s_n)$ of strategies is called a *strategy profile* for the game $\mathcal{G}$, if s determines a unique path through the game tree, and hence a unique terminal node $t \in T$. The *payoff* $f_i(s)$ to player i under the strategy profile s is defined to be $f(t)$, where $f = (f_1(s), \ldots, f_n(s))$ is the payoff function for $s \in S$, and S is the strategy profile set.

An strategy profile is said to be a *Nash Equilibrium* if it contains all the strategies in which neither player can increase his expected payoff by unilaterally changing his strategies. Formally,

Definition 5.3.6 (Nash Equilibrium)**.** Let S_i be the strategy set for player i, and let $s_i, s_{-i} \in S_i$ be a strategy profile of player i and the strategy profile of all players except for player i, respectively. A strategy profile $s^* \in S$ is a *Nash Equilibrium* if no unilateral deviation in strategy by any single player is profitable for that player; i.e., for all $s_i \in S_i$,

$$f_i(s_i^*, s_{-i}^*) \geq f_i(s_i, s_{-i}^*) \qquad \forall i \in \{1, \ldots, n\} \tag{5.1}$$

□

If $s^* \in S$ is comprised uniquely by pure strategies, we say that s^* is a *Pure Nash Equilibrium*. On the other hand, if $s^* \in S$ is comprised of at least one mixed strategy, we say that s^* is a *Mixed Nash Equilibrium*. We shall now define the concept of subgame and subgame Perfect Equilibrium.

Definition 5.3.7 (Subgame)**.** Let G be an extensive game. G' is a **subgame** of G. if it is composed by

(i) The same information sets, payoffs and feasible moves at terminal nodes.

(ii) A set $Y \subset T$, where T is the set of nodes of G, where Y is formed by a single non-terminal node x and all its successors such that if $y \in Y, y' \in h(y)$ then $y' \in Y$.

□

The informal intuition of a subgame is that, when considered in isolation, a subgame constitutes itself a game. The concept of subgame is necessary when defining the concept of subgame perfect equilibrium. In Example 5.3, there are two subgames:

- The entire game (which is always a subgame).
- The (sub)game after Firm 2 enters the market.

Definition 5.3.8 (Subgame Perfect Equilibrium)**.** A strategy profile s is a **subgame perfect equilibrium** of a game G if it induces a Nash equilibrium in every subgame of G. □

Subgame perfection rules out incredible threats by assuming that whenever a move is made, players will always optimize by moving forward. Since every extensive game has a Nash Equilibrium [170], and the entire game is always a subgame, any Subgame Perfect Equilibrium must also be a Nash Equilibrium.

In this section we briefly presented concepts that are necessary to have a basic idea of Game Theory, in particular extensive games. Some of this concepts are necessary in

order to study the specification of extensive games using our process algebra. In the next section we put our process algebra to work. Using the CDS we will characterize the interactions between players in an extensive game. Also, we define expressions that help us to prove whether a strategy profile is a Subgame Perfect Equilibrium.

5.4 Using the CDS to study Extensive Games

In this section we use the CDS to characterize the interactions between players in an extensive game. We start by defining some of the concepts presented in previous sections of this chapter. Then, we present an example of an extensive game as a CDS-process. After the example, we define the concept of subgame perfect equilibrium of a two-person zero-sum game with perfect information and sequential moves.

Recall the syntax of the CDS presented in Figure 3.4. We would like to define a shorthand of the process '`if then else` ' that will simplify our notation throughout this section.

Definition 5.4.1. The process $switch(x)\{x_0 \Rightarrow r_0, \ldots, x_{k-1} \Rightarrow r_{k-1}, r_k\}$ is syntactic sugar of a set of k nested '`if then else`' CDS-terms. If r_k is not in the process, it is assumed to be the CDS-process '0'. □

The interpretation of the process $switch(x)\{x_0 \Rightarrow r_0, \ldots, x_{k-1} \Rightarrow r_{k-1}, r_k\}$ for the case where $k = 3$ is as follows:

$switch(x)\{x_0 \Rightarrow r_0, x_1 \Rightarrow r_1, x_2 \Rightarrow r_2, r_3\} \equiv$

$$\texttt{if } x_0 \texttt{ then } r_0 \texttt{ else } (\texttt{if } x_1 \texttt{ then } r_1 \texttt{ else } (\texttt{if } x_2 \texttt{ then } r_2 \texttt{ else } r_3\ )\ )$$

Using this syntactic sugar, equation 3.10 looks as follows:

$$
\begin{aligned}
TL \stackrel{\text{def}}{=}\ & c(x).switch(x)\{ \\
& (current\_DOC = 0) \vee (current\_DOC > x) \Rightarrow Assign\langle current\_DOC, x, TL\rangle, \\
& (current\_DOC < x) \Rightarrow \overline{c_1}\langle P'_X, R'_X\rangle.AS, \\
& (current\_DOC = x) \Rightarrow switch(x)\{ \\
& (inc = N) \Rightarrow decision(current\_DOC, P_X, R_X), \\
& inc.\overline{c_1}\langle P'_X, R'_X\rangle.AS \\
& \}\}
\end{aligned}
$$

We shall now present some definitions relevant to the application of the CDS to games in extensive form.

5.4.1 Definitions

In this section we begin by defining a two-person zero-sum game as a CDS process.

Definition 5.4.2 (Two Person Game)**.** A **two-person game** is the CDS-process $J \parallel P_1 \parallel P_2 \parallel U$. Where

1. J is a process called the structure of the game.
2. P_1 and P_2 are processes representing players 1 and player 2, respectively.
3. U is the payoff structure of the game.

□

This is a broad definition in the sense that (1) it does not constrain the depth of the game tree, represented by J. In fact, it does not even constrain J to be a game tree. (2) Processes P_1 and P_2 are running concurrently, so extending this definition to a n-person game seems trivial; it will depend on the structure of the game and its payoff. (3) The payoff structure of the game, U, does not constrain the payoff to be of a specific type (zero sum, or non-zero sum), this will be defined by the definition of the process U. We now define the set of strategies as a CDS process.

Definition 5.4.3 (Strategy Set)**.** Let s_{ki} be an expression of the CDS. The set of all possible *strategies* of player i in a game $J||P_1||P_2||U$ is given by:

$$S_i(J) = \{s_{ki} | J = c_i(x).switch(x)\{s_{k0} \Rightarrow J_0, \ldots, s_{kn} \Rightarrow J_n\}\}$$

for all $k = 1, \ldots, K$, and $i = 1, 2$. □

Recall from Section 5.3 that "A strategy is a plan of action, given any particular situation throughout the game". All the elements s_{ki} of $S_i(J)$ are those plans of action to be executed in a given situation during the game. *N.B.:* Given two players i and j, a *strategy profile* is a subset of the union of $S_i \cup S_j$.

Definition 5.4.4 (Players)**.** Player i is a process, P_i, that sends a strategy $s_{ki} \in S_i(J)$ using channel c_i and stops. i.e.,

$$P_i = \bar{c}_i\langle s_{ki}\rangle.\mathbf{0}$$

□

Since we are representing a player by a process, every time a player chooses a different strategy will be represented by a different process, i.e., There is one process (player) for each strategy in the game. The set of all 'process-players' is defined in Definition 5.4.5.

Definition 5.4.5 (Process Players)**.** The set of all Process-Players of player i is given by,

$$\mathbb{P}_i(J) = \{P_i | \exists s_{ki} \in S_i(J), \text{such that } P_i = \bar{c}\langle s_{ki}\rangle.\mathbf{0}, \forall k \in |S_i|\}$$

□

Using the concept defined in Section 5.2.3 and the definition of $\longrightarrow^*$ in Section 3.4, we define the set of outcomes of the game J as follows:

Definition 5.4.6 (Outcomes)**.** The *Outcomes of a game* is a set of all possible outcomes that the players can obtain in a particular game. i.e., Given a game $J||P_1||P_2||U$,

$$outcomes(J) = \{(m, n) | \exists P_1 \in \mathbb{P}_1(J), \exists P_2 \in \mathbb{P}_2(J),\ s.t.\ J||P_1||P_2||U \rightarrow^* (m, n)\}$$

□

Using the definitions of these sections we shall now present an example were we use CDS-terms to describe the interactions between players in a two-person game.

5.4.2 Example

Example 5.4.1 (Extensive Game)**.** Assume that we have the two-person zero-sum game with perfect and complete information depicted in Figure 5.4. Also assume that the payoff function is given by $f((n,m)) = (n,m)$. Then this game is the process $J||P_1||P_2||U$.

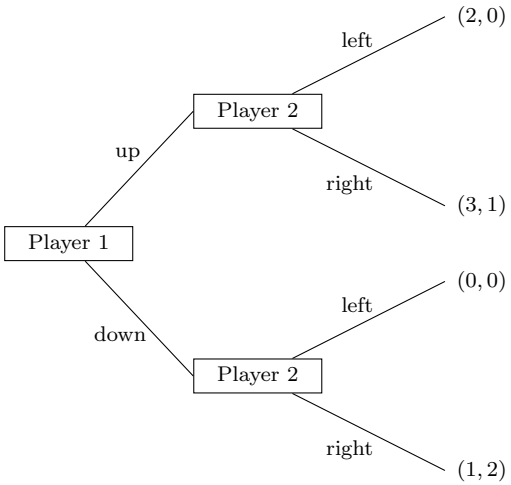


Figure 5.4. Example of a extensive-form game with perfect and complete information.

The structure J of the extensive game depicted in Figure 5.4 is given by the process:

$$J = c_1(x).switch(x)\{$$

$$\text{up} \Rightarrow c_2(y).switch(y)\{\text{left} \Rightarrow \bar{c}_3\langle(2,0)\rangle.\mathbf{0},\ \text{right} \Rightarrow \bar{c}_3\langle(3,1)\rangle.\mathbf{0}\}$$

$$\text{down} \Rightarrow c_2(y).switch(y)\{\text{left} \Rightarrow \bar{c}_3\langle(0,0)\rangle.\mathbf{0},\ \text{right} \Rightarrow \bar{c}_3\langle(1,2)\rangle.\mathbf{0}\}\}$$

Using Definition 5.4.3, we can identify the set of strategies for each player. The set S_1 and S_2 represent such sets for player 1 and player 2, respectively.

$$S_1(J) = \{\text{up}, \text{down}\}$$

$$S_2(J) = \{\text{left}, \text{right}\}.$$

Using Definition 5.4.5, the set of Process-Players containing all processes that represent each possible strategy for each player is given by:

$$\mathbb{P}_1(J) = \{\bar{c}_1\langle\text{up}\rangle.\mathbf{0}, \bar{c}_1\langle\text{down}\rangle.\mathbf{0}\}$$

$$\mathbb{P}_2(J) = \{\bar{c}_2\langle\text{left}\rangle.\mathbf{0}, \bar{c}_2\langle\text{right}\rangle.\mathbf{0}\}$$

Using Definition 5.4.6, the set of all possible outcomes of the extensive game is

$$\mathit{outcomes}(J) = \{(2,0), (3,1), (0,0), (1,2)\}.$$

Recall the reduction semantics of the *function application* prefix, [FUNC-A], in Definition 3.4.1. Since we know that the prefix $\psi = cf(x)$ will receive a payoff of a player (i.e., a value), and we defined on the syntax of the CDS (ref. Figure 3.3.) the process value v, then the payoff process of the extensive game is given by

$$U = c_3 f(x).x$$

Therefore, this process is reduced to the value received through channel c_3; i.e., Assuming that the value v is received through channel c_3, and since $f(v) \downarrow v$, then $c_3 f(x).x \longrightarrow v$.

Note that for $P_1 = \bar{c}_1\langle\text{up}\rangle.\mathbf{0}$, $P_2 = \bar{c}_2\langle\text{right}\rangle.\mathbf{0}$ and $J_1 = c_2(y).\mathit{switch}(y)\{\text{left} \Rightarrow \bar{c}_3\langle(2,0)\rangle.\mathbf{0},\ \text{right} \Rightarrow \bar{c}_3\langle(3,1)\rangle.\mathbf{0}\}$,

$$\begin{aligned} J||P_1||P_2||U &\equiv J \mid\mid \bar{c}_1\langle\text{up}\rangle.\mathbf{0} \mid\mid \bar{c}_2\langle\text{right}\rangle.\mathbf{0} \mid\mid c_3 f(x).x \\ &\to J_1 \mid\mid \mathbf{0} \mid\mid \bar{c}_2\langle\text{right}\rangle.\mathbf{0} \mid\mid c_3 f(x).x \\ &\to \bar{c}_3\langle(3,1)\rangle.\mathbf{0} \mid\mid \mathbf{0} \mid\mid \mathbf{0} \mid\mid c_3 f(x).x \\ &\to (3,1) \end{aligned}$$

which is written as

$$\begin{aligned} J||P_1||P_2||U &\rightarrow J_1 \parallel \bar{c}_2\langle \text{right}\rangle \parallel c_3 f(x).x \\ &\rightarrow \bar{c}_3\langle (3,1)\rangle \parallel c_3 f(x).x \\ &\rightarrow (3,1) \end{aligned}$$

Or simply,

$$J||P_1||P_2||U \rightarrow^* (3,1)$$

▲

5.4.3 Subgame Perfect Equilibrium

In this section we want to use the CDS to compute Subgame Perfect Nash Equilibria as defined by Definition 5.3.8. We will consider the case where the payoff functions are of the form $f(\boldsymbol{x}) = \boldsymbol{x}$.

Definition 5.4.7. Assuming a payoff function of the form $f(\boldsymbol{x}) = \boldsymbol{x}$. The subgame perfect equilibrium of a two-person zero-sum game $J||P_1||P_2||U$, with perfect information and sequential moves, is given by the sequence of strategies $\bigcup_{i\in\{1,2\}} Eq_i(J)$ where

$$Eq_i(J) = \{(s_{ji})_{j=1}^{k}\}|\exists\{s_{ji}\} \subseteq S_i(J),\ s.t.\ s_{i1} \rightarrow^* \text{payoff\_eq}(J)\}$$

and,
payoff_eq$(J) = (m,n) \in outcomes(J)$ is such that

1. If $J = c_1(x).switch(x)\{s_{11} \Rightarrow J_1, \ldots, s_{n1} \Rightarrow J_n\}$ and $P_1 = \bar{c}_1\langle s_{i1}\rangle.P_1'$ then $\forall(m',n') \in outcomes(J),\ s.t.$ payoff_eq$(J_i) = (m',n')$ and $n' \geq n \Rightarrow m' < m$.
2. If $J = c_2(x).switch(x)\{s_{12} \Rightarrow J_1, \ldots, s_{n2} \Rightarrow J_n\}$ and $P_2 = \bar{c}_2\langle s_{i2}\rangle.P_2'$ then $\forall(m',n') \in outcomes(J),\ s.t.$ payoff_eq$(J_i) = (m',n')$ and $m' \geq m \Rightarrow n' < n$

The process payoff_eq(J) carries out the payoff of the profile $\bigcup_{i\in\{1,2\}} Eq_i(J)$. □

Using Definition 5.4.7, in Example 5.4.1 the subgame perfect Nash equilibrium is $\bigcup_{i\in\{1,2\}} Eq_i(J)$, where

$$Eq_1(J) = (\text{up})$$

$$Eq_2(J) = (\text{right})$$

with a payoff of payoff_eq$(J) = (3, 1)$.

5.4.4 Discussion of results

We were able to characterize the behavior of an extensive game. In this case – in the example – we intentionally assumed a typical, rather simple, game structure that allowed us to clearly exemplify the utilization of the CDS in this context. However, expressing a more complex game tree as a CDS-process should not be a difficult task. Since each strategy is represented by a process, the reduction of the game $J||P_1||P_2||U$ depends on the selection of P_1 and P_2, this may not seem ideal, since there will be a reduction for each pair of strategies. However, since the reductions are smoothly done by our operational semantics, the reductions are not complicated to perform; note that we were able to reach a leaf of the game tree in three (logical) computational steps.

Computing the subgame perfect equilibrium was also possible for a particular case of extensive games. The strategy profile that leads to that equilibrium was also found. These notions can be easily extend to the case of more than two players in the game.

5.5 Coupling Extensive Games using the CDS

In this section we want to use the CDS to couple extensive games. The main motivation to study this case of decision systems is due to its strategic application in situations of conflict of interests; i.e., in situations where knowing information about a particular situation can help to resolve favorably other situation that is happening

concurrently. Also, playing games in parallel can resemble the learning from one situation in order to apply it to another, concurrently [171].

Two games can be coupled either sequentially or concurrently. Gosh et al, in [171], studied games played in both fashions, from the perspective of Kripke structures[1] and dynamic logic. Gosh et al, reason about the structure of extensive games, therefore they work with "*extensive form games embedded in Kripke structures rather than with effectivity functions.*" In our case, we take into account the payoff (or decision) functions which define the payoff of the players.

5.5.1 Sequential Composition

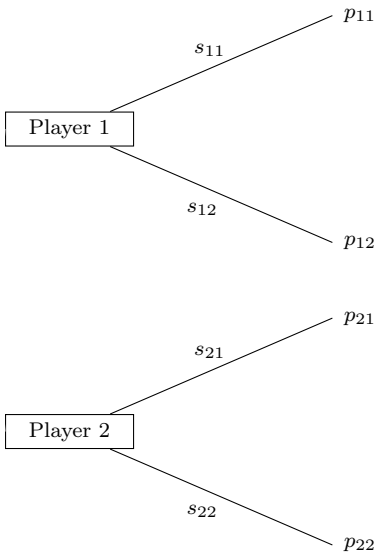
Coupling extensive games in a sequential fashion has been well studied in the literature, not only by logicians like [171], but by game theorists. The main result in sequential composition of extensive games is the following: Suppose we are given two finite extensive games G_1 and G_2, having game structures J_1 and J_2, respectively. Then, the sequential composition of G_1 and G_2, written – in CDS-terms – as $G_1 \cdot G_2$, is generated by concatenating the game structure J_2 to each one of the leaves of the game structure J_1. To illustrate this, assume the game trees depicted in Figure 5.5(a). The resultant game structure of the game $G_1 \cdot G_2$ is depicted in Figure 5.5(b).

5.5.2 Parallel Composition

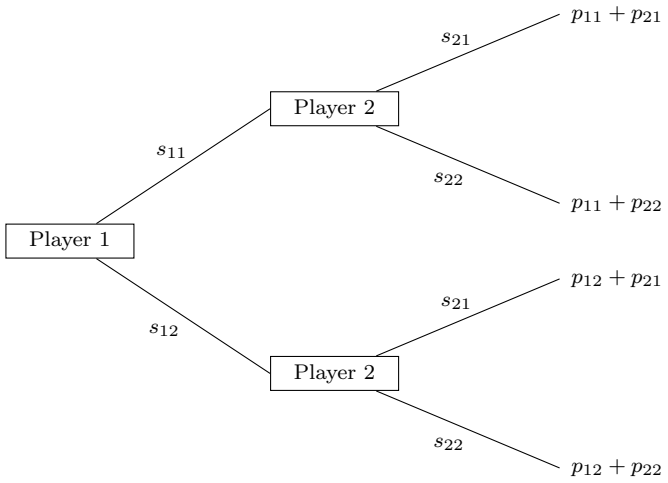
To illustrate how to characterize parallel composition between extensive games using CDS-terms, assume that we have three players playing two two-person zero-sum games with perfect and complete information. Assume that Player 2 is playing the extensive game depicted in Figure 5.6, call it J_1, and concurrently she is playing another game, say J_2, depicted in Figure 5.7.

We want to illustrate the situation where Player 2 mimics the behavior of her

[1] A Kripke structure [172] is a non-deterministic state-transition graph use to represent the behavior of a system; i.e., it is a variation of a NFA.



(a) Atomic game structures J_1 and J_2, of games G_1 and G_2, respectively.



(b) Resultant game tree of the Sequential composition $G_1 \cdot G_2$.

Figure 5.5. Example of a extensive-form game with perfect and complete information.

opponent in game J_1, in another game, J_2, where she is playing versus another opponent; i.e., she learns from Player 1 in J_1 and replicates his actions in J_2 where she is playing against player 3. For simplicity assume that all the payoff functions f_1, f_2 have the form $f_i((n, m)) = (n, m)$ – this assumption can be easily modified.

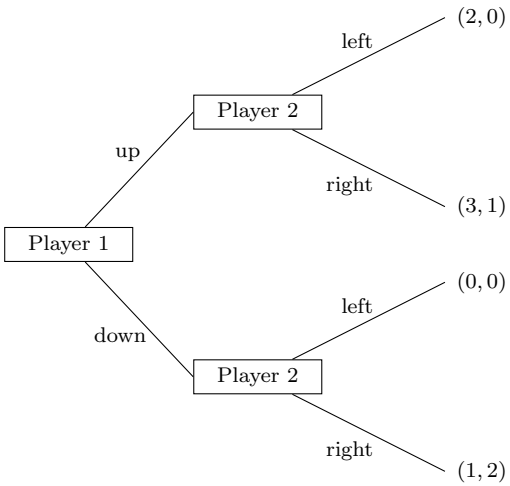


Figure 5.6. Game 1 (J1).

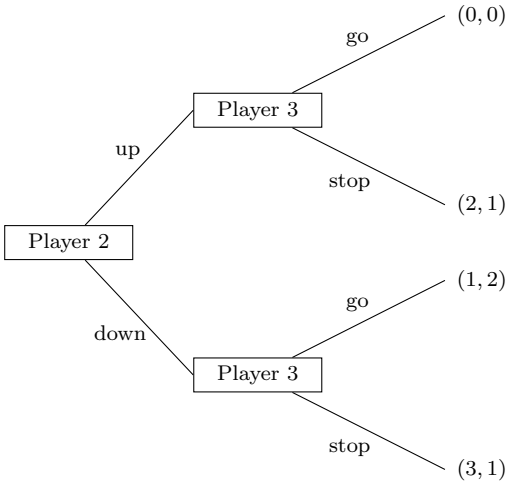


Figure 5.7. Game 2 (J2).

The structure of Game 1 is given by the CDS-process J_1 defined as:

$J_1 = c_1(x).switch(x)\{$

$\text{up} \Rightarrow \bar{c_2}'\langle up\rangle.c_2(y).switch(y)\{\text{left} \Rightarrow \overline{c_4}\langle(2,0)\rangle.\mathbf{0},\ \text{right} \Rightarrow \overline{c_4}\langle(3,1)\rangle.\mathbf{0}\}$

$\text{down} \Rightarrow \bar{c_2}'\langle down\rangle.c_2(y).switch(y)\{\text{left} \Rightarrow \overline{c_4}\langle(0,0)\rangle.\mathbf{0},\ \text{right} \Rightarrow \overline{c_4}\langle(1,2)\rangle.\mathbf{0}\}\}$

The structure of Game 2 is given by the CDS-process J_2 defined as:

$J_2 = c_2'(x).switch(x)\{$

$\text{up} \Rightarrow c_3(y).switch(y)\{\text{go} \Rightarrow \bar{c}_5\langle(0,0)\rangle.\mathbf{0},\ \text{stop} \Rightarrow \bar{c}_5\langle(2,1)\rangle.\mathbf{0}\}$

$\text{down} \Rightarrow c_3(y).switch(y)\{\text{go} \Rightarrow \bar{c}_5\langle(1,2)\rangle.\mathbf{0},\ \text{stop} \Rightarrow \bar{c}_5\langle(3,1)\rangle.\mathbf{0}\}\}$

Thus, the structure of the game is given by

$$J = J_1 \mid\mid J_2.$$

Similar to Example 5.4.1, we will study the reduction of the extensive game assuming $P_1 = \overline{c_1}\langle up\rangle.\mathbf{0}$ and $P_2 = \overline{c_2}\langle right\rangle.\mathbf{0}$. Then, the process $J||P_1||P_2||P_3||U$ is reduced in the following manner:

$$\begin{aligned}
& J||P_1||P_2||P_3||U \equiv \\
& \overbrace{J_1 \mid\mid J_2}^{J} \mid\mid \overbrace{\bar{c}_1\langle\text{up}\rangle.\mathbf{0}}^{P_1} \mid\mid \overbrace{\bar{c}_2\langle\text{right}\rangle.\mathbf{0}}^{P_2} \mid\mid \overbrace{\bar{c}_3\langle\text{stop}\rangle.\mathbf{0}}^{P_3} \mid\mid \overbrace{c_4 f_1(x).x \mid\mid c_5 f_2(x).x}^{U} \\
\rightarrow\ & J_1' \mid\mid J_2 \mid\mid \mathbf{0} \mid\mid \overline{c_2}\langle right\rangle.\mathbf{0} \mid\mid \overline{c_3}\langle stop\rangle.\mathbf{0} \mid\mid c_4 f_1(x).x \mid\mid c_5 f_2(x).x \\
\rightarrow\ & \overline{c_4}\langle(3,1)\rangle \mid\mid J_2' \mid\mid \mathbf{0} \mid\mid \mathbf{0} \mid\mid \overline{c_3}\langle stop\rangle.\mathbf{0} \mid\mid c_4 f_1(x).x \mid\mid c_5 f_2(x).x \\
\rightarrow\ & (3,1) \mid\mid \overline{c_5}\langle(2,1)\rangle.\mathbf{0} \mid\mid \mathbf{0} \mid\mid \mathbf{0} \mid\mid \mathbf{0} \mid\mid \mathbf{0} \mid\mid c_5 f_2(x).x \\
\rightarrow\ & \underbrace{(3,1)}_{\text{Payoff of } J_1} \mid\mid \rightarrow \underbrace{(2,1)}_{\text{Payoff of } J_2}
\end{aligned}$$

where,

- $J_1' = \bar{c_2}'\langle up\rangle.c_2(y).switch(y)\{\text{left} \Rightarrow \bar{c}_4\langle(2,0)\rangle.\mathbf{0},\ \text{right} \Rightarrow \bar{c}_4\langle(3,1)\rangle.\mathbf{0}\}$, and
- $J_2' = c_3(y).switch(y)\{\text{go} \Rightarrow \bar{c}_4\langle(0,0)\rangle.\mathbf{0},\ \text{stop} \Rightarrow \bar{c}_5\langle(2,1)\rangle.\mathbf{0}\}$

The reductions of the process $J||P_1||P_2||P_3||U$ can be written as

$$\begin{aligned}
J||P_1||P_2||P_3||U \ \rightarrow\ & J_1' \mid\mid J_2 \mid\mid \overline{c_2}\langle right\rangle \mid\mid \overline{c_3}\langle stop\rangle \mid\mid c_4 f_1(x).x \mid\mid c_5 f_2(x).x \\
\rightarrow\ & \overline{c_4}\langle(3,1)\rangle \mid\mid J_2' \mid\mid \overline{c_3}\langle stop\rangle \mid\mid c_4 f_1(x).x \mid\mid c_5 f_2(x).x \\
\rightarrow\ & (3,1) \mid\mid \overline{c_5}\langle(2,1)\rangle \mid\mid c_5 f_2(x).x \\
\rightarrow\ & (3,1) \mid\mid \rightarrow (2,1)
\end{aligned}$$

or simply,

$$J||P_1||P_2||P_3||U \rightarrow^* (3,1) \text{ and } J||P_1||P_2||P_3||U \rightarrow^* (2,1)$$

The subgame perfect equilibrium for each individual game is given by $Eq_1(J_1) = (up)$, $Eq_2(J_1) = (right)$ and $Eq_1(J_2) = (up)$, $Eq_2(J_2) = (stop)$ with payoffs of payoff_eq$(J_1) = (3,1)$, and payoff_eq$(J_2) = (2,1)$. The subgame perfect equilibrium and the payoffs of the players are given in Table 5.2.

Table 5.2
Payoffs of the three players.

Player	**Strategy**	**Payoff**
1	$\langle up \rangle$	**3**
2	$\langle right, up \rangle$	1+2=**3**
3	$\langle stop \rangle$	**1**

5.5.3 Example

Example 5.5.1. Assume that we have the two-person zero-sum game with perfect and complete information depicted in Figure 5.8. In this example, the players will play the same game two times concurrently, alternating strategies between games. There is a small delay (one computational step) that allows players to know what strategy they use in the first iteration of the game, so that they can decide which strategy will be used in the other (concurrent) iteration of the game – The case where this delay is not considered is presented in Section 5.6. Also assume that the payoff functions f_1, f_2 have the form $f_i((n,m)) = (n,m)$.

The structure of the first instance of the game is given by the process:

$J_1 = c_1(x).switch(x)\{$

up $\Rightarrow \bar{c_1}'\langle down \rangle.c_2(y).switch(y)\{$left $\Rightarrow \bar{c_2}'\langle right \rangle.\overline{pay_1}\langle(2,0)\rangle.\mathbf{0},$ right $\Rightarrow$ $\bar{c_2}'\langle left \rangle.\overline{pay_1}\langle(3,1)\rangle.\mathbf{0}\}$

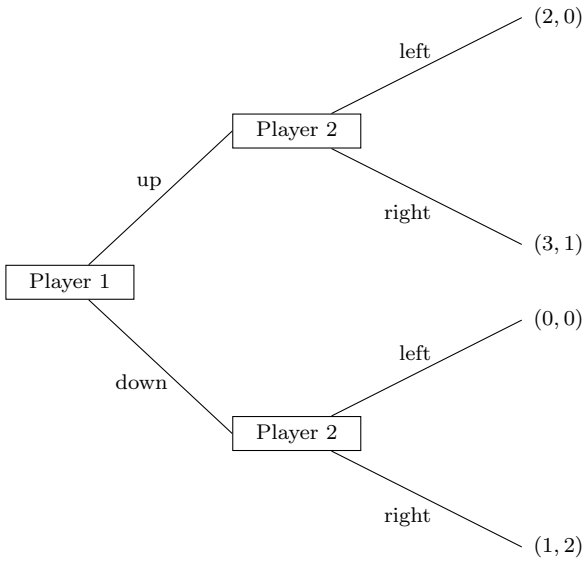


Figure 5.8. Extensive game played two times concurrently.

down $\Rightarrow \bar{c_1}'\langle up\rangle.c_2(y).switch(y)\{\text{left} \Rightarrow \bar{c_2}'\langle right\rangle.\overline{pay_1}\langle(0,0)\rangle.\mathbf{0},\ \text{right} \Rightarrow \bar{c_2}'\langle left\rangle.\overline{pay_1}\langle(1,2)\rangle.\mathbf{0}\}\}$

The structure of the second instance of the game is given by the process:

$J_2 = c_1'(x).switch(x)\{$

up $\Rightarrow c_2'(y).switch(y)\{\text{left} \Rightarrow \overline{pay_2}\langle(2,0)\rangle.\mathbf{0},\ \text{right} \Rightarrow \overline{pay_2}\langle(3,1)\rangle.\mathbf{0}\}$

down $\Rightarrow c_2'(y).switch(y)\{\text{left} \Rightarrow \overline{pay_2}\langle(0,0)\rangle.\mathbf{0},\ \text{right} \Rightarrow \overline{pay_2}\langle(1,2)\rangle.\mathbf{0}\}\}$

Assume that Player 1 decides to play *down* and Player 2 plays *left* in the first instance of the game (J'). Then,

$$\begin{aligned}
J||P_1||P_2||U &\equiv \overbrace{J_1 \mid\mid J_2}^{J} \mid\mid \overline{c_1}\langle down\rangle \mid\mid \overline{c_2}\langle left\rangle \mid\mid \overbrace{\overline{pay_1}f_1(x).x \mid\mid \overline{pay_2}f_2(x).x}^{U} \\
&\rightarrow J_1' \mid\mid J_2 \mid\mid \overline{c_2}\langle left\rangle \mid\mid \overline{pay_1}f_1(x).x \mid\mid \overline{pay_2}f_2(x).x \\
&\rightarrow \bar{c_2}'\langle right\rangle.\overline{pay_1}\langle(0,0)\rangle \mid\mid J_2' \mid\mid \overline{pay_1}f_1(x).x \mid\mid \overline{pay_2}f_2(x).x \\
&\rightarrow \overline{pay_1}\langle(0,0)\rangle \mid\mid \overline{pay_2}\langle(3,1)\rangle \mid\mid \overline{pay_1}f_1(x).x \mid\mid \overline{pay_2}f_2(x).x \\
&\rightarrow (0,0) \mid\mid \rightarrow (3,1)
\end{aligned}$$

▲

5.5.4 Discussion of results

Using the syntax and reduction semantics of the CDS we were able to characterize the interactions between (1) three players playing two extensive games where one of the players is playing in both game concurrently, and (2) two players are playing two instances of the same game concurrently. If we would have had to use a game tree to express either of the situations presented in this section, such structure would have been significantly more complicated than the structure of one of the individual game trees depicted in Figure 5.6 or Figure 5.7. However, using the CDS we were able to reach the leaves of the tree in only one more computational step than if the structure of the game was that of Figure 5.6.

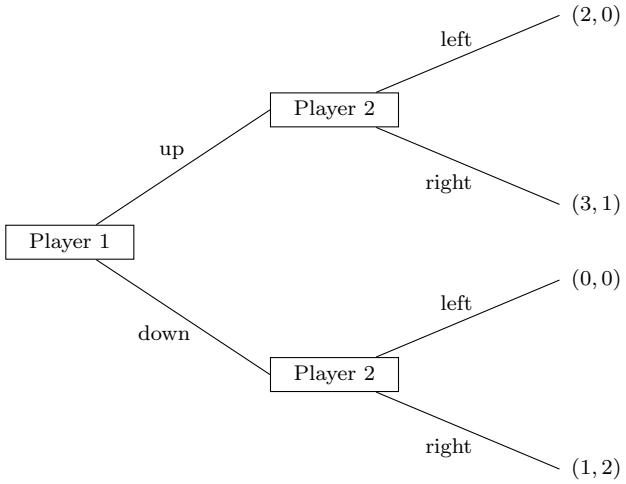
5.6 Bisimilar Extensive Games

The goal of this section is to apply the concepts presented in Chapter 4 in the context of extensive games. The notion of bisimilarity will allow us to compare when two extensive game are similar from the perspective of an observer. Therefore, we present two cases where the resultant extensive game of one case is not trivially comparable with the other.

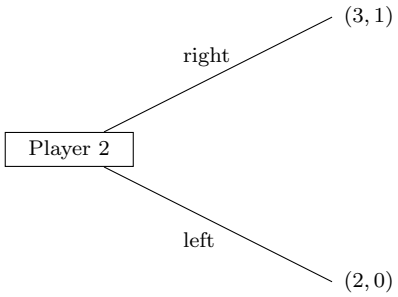
Let us assume the following situation:

(I) G_1 is the resultant game after coupling two games, G_1' and G_1'', such that Player 1 and Player 2 play two instantiations of the same game in parallel. Unlike Example 5.5.1, since players are not alternating strategies, in this case both players are playing the two games exactly at the same time. For illustration purposes we will assume that players will play the same strategy they play in G_1' in G_1''. Figure 5.9(a) shows the game tree of G_1', which is the same than G_1''. As it is customary at this point, assume that the payoff functions f_1, f_2 have the form $f_i((n, m)) = (n, m)$.

(II) G_2 is the resultant game after coupling two games such that Player 2 plays two games in parallel, G_2' and G_2''. Player 2 is the last player for both games. In addition, in G_2'' she just mimics the decision made in G_2'. Figure 5.9 shows the game trees of G_2' (Fig. 5.9(a)) and G_2'' (Figure 5.9(b)). We are assuming that the players involved in G_2 are the same than those involved in G_1.



(a) Game tree of G_1' and G_2'.



(b) Game tree of G_2''.

Figure 5.9. Game trees of G_1 and G_2.

Assuming that Player 1 plays *up* and Player 2 plays *left* in G_1, the reductions are as follows (note that the strategies played by the players are irrelevant to our purposes):

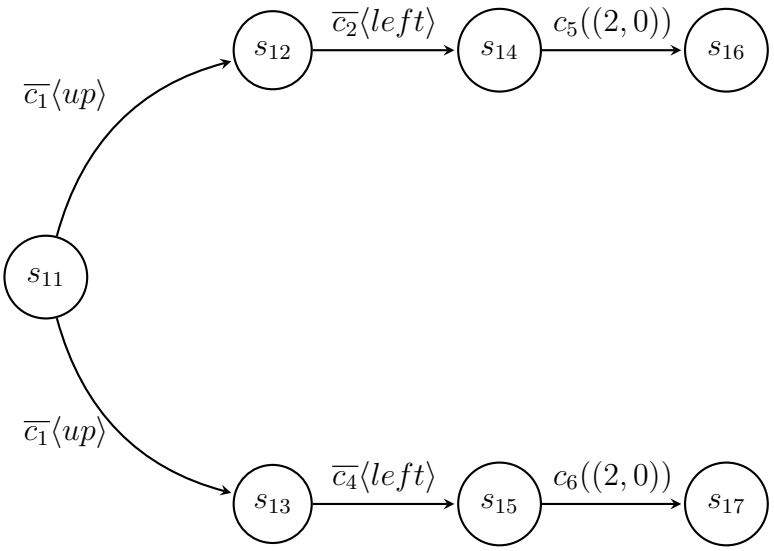


Figure 5.10. LTS of G_1.

$$
\begin{aligned}
& J_{11} \mid\mid J_{12} \mid\mid \overline{c_1}\langle up\rangle.\overline{c_3}\langle up\rangle \mid\mid \overline{c_2}\langle left\rangle.\overline{c_4}\langle left\rangle \mid\mid c_5 f_1(x).x \mid\mid c_6 f_2(x).x \\
\longrightarrow \quad & J'_{11} \mid\mid J'_{12} \mid\mid \overline{c_2}\langle left\rangle.\overline{c_4}\langle left\rangle \mid\mid c_5 f_1(x).x \mid\mid c_6 f_2(x).x \\
\longrightarrow \quad & \overline{c_5}\langle(2,0)\rangle \mid\mid \overline{c_6}\langle(2,0)\rangle \mid\mid c_5 f_1(x).x \mid\mid c_6 f_2(x).x \\
\longrightarrow \quad & (2,0) \mid\mid (2,0)
\end{aligned}
$$

We want to express this game as a LTS as defined in Section 4.3. To that end, we start by naming each state of the system as follows:

$$
\begin{aligned}
& J_{11} \mid\mid J_{12} \mid\mid \overline{c_1}\langle up\rangle.\overline{c_3}\langle up\rangle \mid\mid \overline{c_2}\langle left\rangle.\overline{c_4}\langle left\rangle \mid\mid c_5 f_1(x).x \mid\mid c_6 f_2(x).x && \boldsymbol{(S_{11})} \\
\longrightarrow \quad & J'_{11} \mid\mid J'_{12} \mid\mid \overline{c_2}\langle left\rangle.\overline{c_4}\langle left\rangle \mid\mid c_5 f_1(x).x \mid\mid c_6 f_2(x).x && \boldsymbol{(S_{12}, S_{13})} \\
\longrightarrow \quad & \overline{c_5}\langle(2,0)\rangle \mid\mid \overline{c_6}\langle(2,0)\rangle \mid\mid c_5 f_1(x).x \mid\mid c_6 f_2(x).x && \boldsymbol{(S_{14}, S_{15})} \\
\longrightarrow \quad & (2,0) \mid\mid (2,0) && \boldsymbol{(S_{16}, S_{17})}
\end{aligned}
$$

Using the transition defined in Table 4.1, we can define the LTS presented in Figure 5.10.

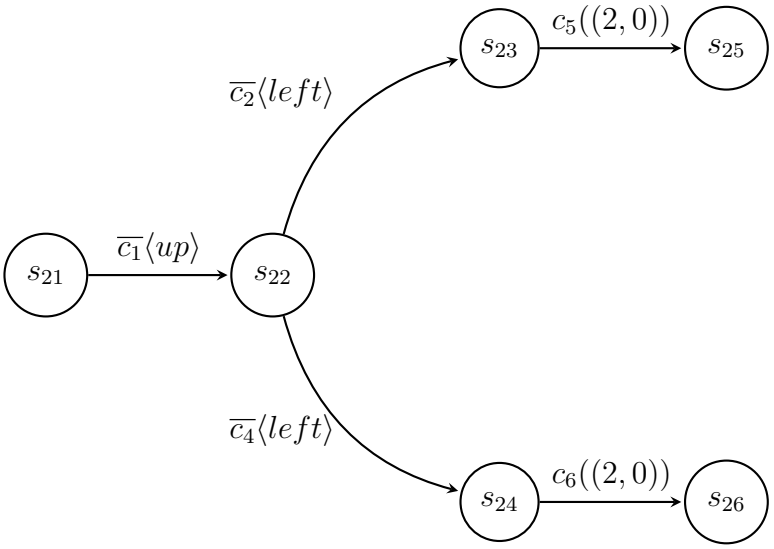


Figure 5.11. LTS of G_2.

Similarly, the reductions, together with the state names, of G_2 is given by

$$J_{21} \mid\mid J_{22} \mid\mid \overline{c_1}\langle up\rangle \mid\mid \overline{c_2}\langle left\rangle.\overline{c_4}\langle left\rangle \mid\mid c_5 f_1(x).x \mid\mid c_6 f_2(x).x \qquad (\boldsymbol{S_{21}})$$

$$\longrightarrow \quad J'_{21} \mid\mid J_{22} \mid\mid \overline{c_2}\langle left\rangle.\overline{c_4}\langle left\rangle \mid\mid c_5 f_1(x).x \mid\mid c_6 f_2(x).x \qquad (\boldsymbol{S_{22}})$$

$$\longrightarrow \quad \overline{c_5}\langle(2,0)\rangle \mid\mid \overline{c_6}\langle(2,0)\rangle \mid\mid c_5 f_1(x).x \mid\mid c_6 f_2(x).x \qquad (\boldsymbol{S_{23}, S_{24}})$$

$$\longrightarrow \quad (2,0) \mid\mid (2,0) \qquad (\boldsymbol{S_{25}, S_{26}})$$

The LTS for G_2 is depicted in Figure 5.11.

Now we would like to investigate if G_1 and G_2 are bisimilar. This is stated in the following theorem.

Proposition 5.6.1. *Given the rules described in (I) and (II). The extensive games G_1 and G_2 are bisimilar in the CDS.*

Proof . We need to verify the conditions of Definition 4.4.1 hold. Indeed,

(1) $Func(P) = Func(Q)$

Condition (1): Since we are assuming that the players involved in G_2 are the same than those involved in G_1, then $Func(G_1) = Func(G_2)$.

(2) Whenever $P \xrightarrow{\alpha} P'$ then, for some Q', $Q \xrightarrow{\alpha} Q'$, and $(P', Q') \in \mathcal{S}$;

(3) Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P', $P \xrightarrow{\alpha} P'$, and $(P', Q') \in \mathcal{S}$.

Conditions (2) and (3): Assume that Player 1 plays *up* and Player 2 plays (left). Then, it is easy to show that for every state S_{1i} in the LTS depicted in Figure 5.10, we can find a state S_{2i} in the LTS depicted in Figure 5.11 such that S_{1i} simulates S_{2i}, and vice versa. We can relax the assumption on the strategies played by the players and obtain the corresponding labelled transition systems. Then, the result follows by applying the same procedure described above.

■

5.6.1 Discussion of results

Using the the LTS and notion of bisimulation for the CDS, we were able to compare two extensive games from the behavioral standpoint. We assumed that the players of the two games G_1 and G_2 were the same so that the decision/payoff functions were the same. For this reason the proof of condition 1 was straightforward. For simplicity of illustration, the form of the decision functions was assumed to be that of a fixed point. However, these functions could have been of any other form as long as $Func(G_1) = Func(G_2)$.

Proposition 5.6.1 suggests that, with some probability, playing two concurrent games as first player or second player is behaviorally equivalent. This result presumes that conditions described in (I) and (II). If we relax those conditions, the two extensive games may not be bisimilar; i.e., if, for example, Player 1 in G_1 does not play the same strategy in both instantiations of the game, the resultant LTS may behave differently than the LTS of G_2.

5.7 Extensive Games and Organizational Structures

The goal of this section is to show an implementation of the CDS to organizational structures. Organizational structures play an important role in the study of decision systems because they can be seen as a decision systems with multiple decision makers. In this section we do not present an introduction to the theory of organizational structures. Our objective is to show how an organizational structure can be characterized using the CDS.

5.7.1 Process Algebras and Organizational Structures

The application of process algebras to the study of organizational structures is relatively new and is still an active area of research. In 2002, Ulrich [173] proposed a framework that claims to be useful for describing the activity of an organization. He acknowledges the potential uses of process algebras to modeling organizations, although he believes they lack of expressiveness of domain level concepts. In that same year, Arkin et al published their work on Business Process Modeling Language [174], which uses π-calculus as its foundations. In 2003, Smith [175] published a work entitled "*Business process management; the third wave: business process modeling language (bpml) and its π-calculus foundations*" where he argues the uses of process algebras – in particular π-calculus – to capture, describe and manage whole processes in an organization. A few years later, in 2006, F. Puhlmann [176] published a paper where he discusses the applicability of process algebras as a formal foundation for Business Process Management (BPM).

Smith and Fingar in [177] claim that workflow patterns and the services provided by work-flow engines can be modeled with languages derived from π-calculus. This publication turned out to be rather controversial [178] and urged the interest in π-calculus as a tool for studying problems in Organizational Theory and related areas. The following section presents a rather simple organizational structure where the interactions between the elements of the structure are characterized using the CDS.

5.7.2 CDS and Organizational Structures

Organizational structures are not organized for the sake of organization, but to produce a product or a service; i.e., a decision. Using mechanisms such as the CDS for the study of organizational structures can be seen as a natural choice, because this type of process algebras can easily characterize the flow of information between processes. Also, in the particular case of the CDS, the theory behind this calculus could be used for comparing two different structures using bisimulation. We have shown how we can couple different extensive games using the CDS. In this context, we could think of an organizational structure as an organized structure of extensive games that eventually lead to a decision. The input of an arbitrary element of the organization is the output of its predecessor; i.e., the solution of the extensive game played by two players in an organization affects all the chain of command (or unity of command) of each of these players. The following example shows how the information can be shared within an organizational structure.

Example 5.7.1. Consider a healthcare institution with the structure depicted in Figure 5.12. Assume the simplest case where the information flows in one direction and there is no back communication after this. Using the syntax provided in Figure 3.3, we have the following process:

> At level 1: The CEO is the process $A.\overline{c_1}\langle x\rangle$, meaning that he behaves like A and sends information using channel c_1.
>
> At level 2: The COO is a process $c_1(x).B.\overline{c_2}\langle y\rangle$ meaning that he receives information through channel c_1, behaves as B and sends out information through channel c_2. Similarly, the CMO is the process $c_1(x).C.\overline{c_3}\langle y\rangle$ and the CFO is the process $c_1(x).D$.
>
> At level 3: Similarly than before, the VPS is the process $c_2(x).E$, the VPP is the process $c_2(x).F$, the PSO is the process $c_2(x).G$ and the PS is the process $c_2(x).H$. Also, the DH is characterized by the process $c_3(x).I$.

Since there is a hierarchy, some processes happen after others, e.g. the VPS waits until the COO makes a decision before acting, and the COO waits until the CEO sends him his decision before acting. Therefore, the process J that characterizes the organizational structure depicted in Figure 5.12 is the following:

$$\begin{aligned} J = \quad & A.\overline{c_1}\langle x\rangle \quad || \ (c_1(x).B.\overline{c_2}\langle y\rangle \ || \ (c_2(x).E + c_2(x).F + c_2(x).G + c_2(x).H) \\ & || \ (c_1(x).C.\overline{c_3}\langle y\rangle \ || \ c_3(x).I) \\ & || \ c_1(x).D) \end{aligned}$$

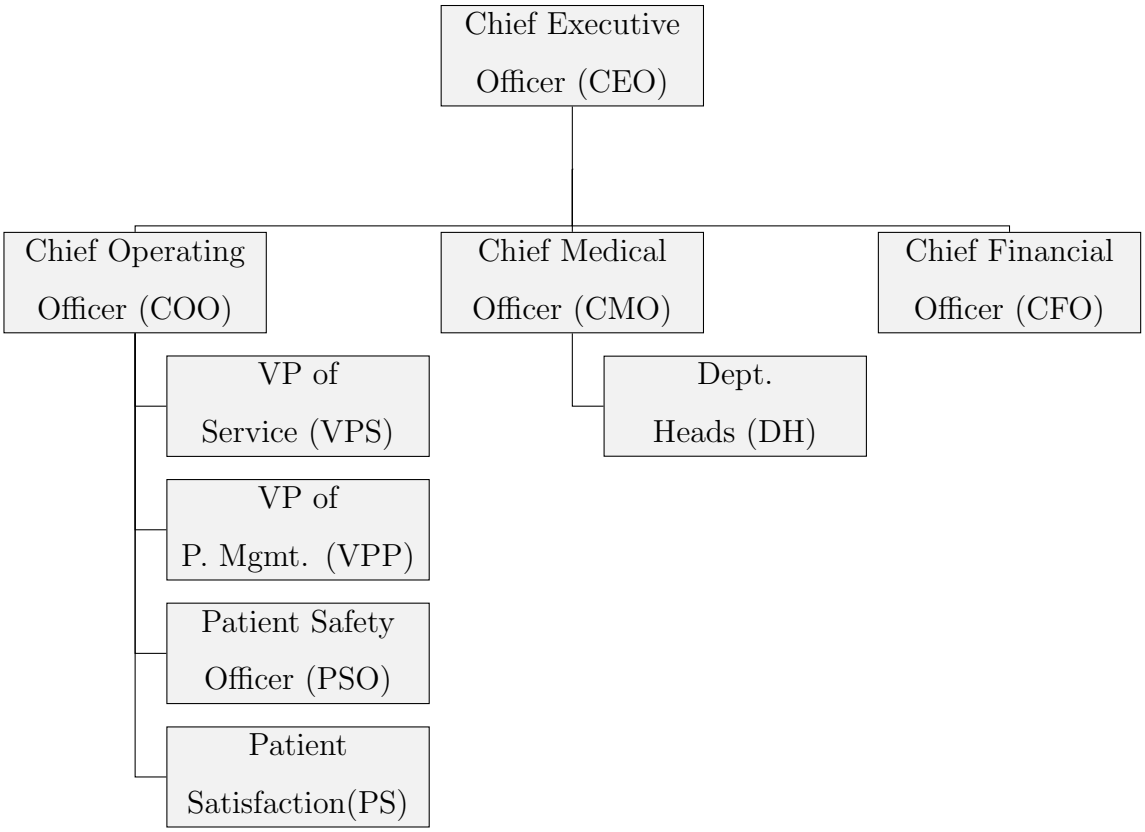
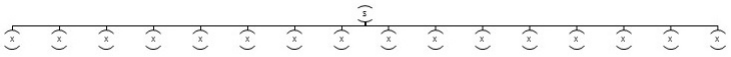
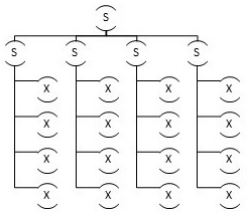


Figure 5.12. Example of an Organizational Structure

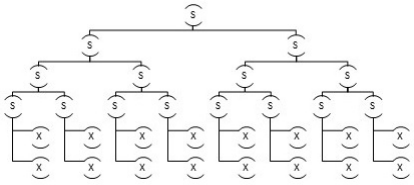
▲



(a) Span ratio of 16:1.



(b) Span ratio of 4:1.



(c) Span ratio of 2:1.

Figure 5.13. Example of Span of Supervision

Unity of Command

According to [179], *unity of command* facilitates order in an organizational structure because it charges one official with an area of responsibility and establishes a chain of command where each member in the organization knows to whom he reports and who reports to him. The unity of command is defined by the *span of supervision*, or *span of control*. The span of supervision determines the ratio between superior and subordinates in the organization. If a superior has a narrow span of supervision, he can devote a considerable amount of time to each subordinate. Therefore, he can supervise "closely" and retain much of the decision-making authority. Figure 5.13 shows different spans of supervision.

Characterizing the span of supervision using the CDS is feasible using the *restriction* process. Recall that this process restricts the scope of a channel in a process. For example, Figure 5.14 has one supervisor, S, and three subordinates, A, B and C.

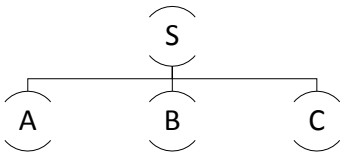


Figure 5.14. Superior-Subordinate relationships

The fact that the three subordinates could receive orders from S can be characterized by the process $new\ c(S.\bar{c} \mid\mid c.A + c.B + c.C)$.

Therefore, the process that characterizes the organizational structure of Example 5.12 is given by the following expression:

$$\begin{aligned} J = \quad new\ c_1\ (A.\bar{c_1} \quad & \mid\mid new\ c_2(c_1.B.\bar{c_2} \mid\mid c_2(x).E + c_2(x).F + c_2(x).G + c_2(x).H) \\ & \mid\mid new\ c_3(c_1(x).C.\bar{c_3}y \mid\mid c_3(x).I) \\ & \mid\mid c_1(x).D) \end{aligned}$$

A limitation on the span of supervision is the number of possible relationships that a supervisor may be required to manage. Graicunas [180] argued that the ability to carry on face-to-face conversations between members in an organization decreases as the number of members increases. Therefore, the quality of supervision highly depend on the number of relationships involved in the span of supervision. Graicunas classified the possible relationships to be supervised in three types:

- Direct single relationships. These are the one-to-one relationships between the supervisor and his subordinates.
- Direct group relationships. These are the one-to-many relationships between the supervisor and all the subordinates.
- Cross relationships. These are the one-to-one relationships between subordinates without including the supervisor.

According to [179] the total number of relationships under each span of supervision, under one supervisor, is determined by the following equation:

$$r = n(2^{n-1} + n - 1) \tag{5.2}$$

where n is the number of persons supervised, and r the number of relationships. In Figure 5.14 the possible relationships within an structure of one supervisor and three subordinates are:

- Direct single relationships:
  (1) S → A
  (2) S → B
  (3) S → C
- Direct Group Relationships:
  (4) S → A with B
  (5) S → A with C
  (6) S → B with A
  (7) S → B with C
  (8) S → C with A
  (9) S → C with B
  (10) S → A with B and C
  (11) S → B with A and C
  (12) S → C with B and A
- Cross relationships
  (13) A → B
  (14) A → C
  (15) B → A
  (16) B → C

(17) C → A

(18) C → B

Note that, using 5.2, we have that $r = 3(2^{3-1} + 3 - 1) = 18$. In the context of the CDS, we can take advantage of this result and relate it with the number of channels in a process. Even though we can use the same channel to characterize the two-way communication between two processes, in the context of organizational theory this two relationships count separately. The following lemma states this fact formally.

Lemma 5.7.1. *Let N be the number of communication channels of a process P, and r the number of superior-subordinate relationships of an organizational structure, with one supervisor, characterized by P. Then $N \leq r$; i.e., the number of channels needed to describe an organizational structure, under one supervisor, using the CDS is right-bounded by the number of superior-subordinate relationships described in (5.2).*

Proof . The proof is trivial since the only scenario where we need as many communication channels as relationships, is in the case where all communications between processes is expected to be confidential; otherwise we need an smaller number of channels. Also, the fact that processes can use the same channel to communicate back and forth reduces the number of cross relationships in half. ■

Lemma 5.7.1 shows one of the many connections that can be made between channels and relationships within an organization. Many results in organizational theory such as Fordham model [179, p. 88] and Lockheed Model [179, p. 93] can be use to study the efficiency of communication channels in a process that characterizes a particular structure. This type of extensions are out of the scope of this dissertation, but they suggest a natural line of future research.

In this section we do not ambition to make a contribution to the area of organizational structures, other than that of describing structures using process algebras. In the context of organizational structures, we use the CDS as an information sharing mechanism to transfer the inputs and outputs from one extensive game to another.

This characterization of an organizational structure becomes more meaningful if it is assumed that the potential conflicts of interests are characterized by an extensive game expressed in the CDS.

5.8 Advantages and Current Limitation of the CDS in GT

Advantages: The syntax of the CDS allows us to describe different extensive games in a relatively simple manner. Coupling of extensive games is also feasible. Even though the structure of the game tree may become complex, the reductions of the CDS-processes are still relatively simple. The action semantics of the CDS allowed us to obtain a labelled transition system out of a game; this is useful to understand how players are interacting in a game. The notion of bisimulation provides insights that cannot be gathered using game trees or other methods in game theory.

Current limitations: The syntax of the CDS cannot deal with ties between payoffs; i.e., it cannot define mixed strategies or information set of cardinality greater than one. This is because – currently – the CDS can only represent handshaking [2] communication. Also, the syntax of the CDS does not support probabilistic choice only internal choice which does not provide control on the probability of the choice.

5.9 Summary

In this chapter we have used our calculus to characterize games in extensive form. We started with a brief introduction of game theory where we defined some of the main concepts that are related with our goals. Later on, we defined some of this concepts in the context of the CDS. This allowed us to define a game by its structure, players and payoff structure. After characterizing an extensive game with CDS-terms, we defined the concept of subgame perfect equilibrium in our context.

[2] *Handshaking communication* refers to the type of communication that can happen only between two parties (see [109] p. 198).

One of the main contributions of this chapter was to couple extensive games in different fashions. Using the CDS we characterized situations were there are players playing more than one game at the same time. Even though the resultant game tree of composing two games may become more complicated than that of one of the singleton game trees, using the CDS we were able to reach a leave in only a few more computational steps; and it was done "automatically" using the reduction semantics.

How to couple two extensive games? This is an interesting enough question. However, we took this question a step forward. How to know if two sets of coupled games behave similarly or differently? Using the notion of strong bisimulation defined in Chapter 4 we were able to compare two set of composed games. We started by converting the reductions of the composed extensive games to labelled transition systems, and then testing if the decision systems behave similarly by a rather simple procedure introduced at the beginning of this chapter.

We concluded the chapter with a discussion on the most relevant advantages provided by the CDS, and some of the disadvantages that constraints the expressiveness of the CDS for characterizing extensive games.

6. CYBER-PHYSICAL SYSTEMS AS DECISION SYSTEMS

6.1 Introduction

In this chapter we discuss the well known area of Cyber-Physical Systems in the context of Decision Systems. We can think of Cyber-Physical Systems (or CPS) as systems comprised by a computing component and a physical component working tightly together. As we have repeatedly discussed during this dissertation, the roots of the CDS lie in the area of computer science, and this calculus is thought off as a formal model of computation in the context of decision systems. It is presumed that this formalization will be eventually implemented in a computer in order to analyze decision systems. The nature of the CDS, therefore, makes – inherently – any physical system – described by its syntax – a cyber-physical system.

We start the chapter with an overview of cyber-physical systems. Then we present some formal tools that have been developed for the study of real-time systems and cyber-physical systems. In Section 6.4, we discuss the notion of time in the context of the CDS. In Section 6.6, we use the CDS to study cyber-physical systems. We start by describing some basic physical systems. Later on, we use the CDS to describe the interactions involved in the Generalized Railroad Crossing problem, in the particular case of two railroads. Before concluding the chapter, we present some of the advantages and current limitations of the CDS for the study of cyber-physical systems.

6.2 Overview of Cyber-Physical Systems

Cyber-Physical Systems (or CPS[1]) is a class of systems that are characterized by a tight relationship between systems (or processes) in the areas of computing, communication and physics. As mentioned in [181], the bond between these processes is so strong that "*it is not possible to identify whether behavioural attributes are the results of computing, communication, control, physical laws or all of them working together.*" The term Cyber-Physical system started to appear in the literature in the decade of the 2000's (see [182] and [183]). Later –after the NSF identified CPS as important field of research [184] – the research in the now called science of cyber-physical systems has evolved tremendously, leading this new science to be classified by many as *the new computing revolution* [185, 186].

6.2.1 Characteristics of CPS

Some of the characteristics of cyber-physical systems – from [187] – are:

- *Heterogeneity*: CPS are comprised by systems from different domains.
- *Unreliable Networking*: CPS usually operate over ad-hoc wireless networks and environments with power constraints.
- *Mobility*: Many CPS include mobile devices, adding complexity to the system.
- *Tight Environmental Coupling*: Most of CPS are greatly affected by the environment they belong.

The integration between physical systems and computing systems is not a new concept. The areas of *embedded systems* and *robotics* has been widely used to describe engineered systems that combine these two areas. As described by Lee in [188], the main difference between embedded systems and CPS is the envision of the networking

[1]Sometimes we may use the acronym '*CPS*' when referring to a single cyber-physical system.

capabilities of CPS. Thus, the communication and concurrency capabilities of CPS are paramount.

6.2.2 Challenges of CPS

The science of CPS promises to solve some of the big problems faced by our society. According to [185], some of the grand challenges for CPS are the following:

- Generation and distribution of –blackout-free – electricity.
- Safety and efficiency in facilities evacuation.
- Efficient automotive traffic management.
- Minimal automotive traffic fatalities.
- Energy efficient/aware buildings and cities, among others.

The science of CPS is still considered an area under development and there is a big opportunity to make small steps towards the creation of more resilient and reliable Cyber-Physical Systems. In the next section we explore some of the formal approaches developed for the analysis of CPS. We start with a discussion on formal methods for real-time systems and then we present the material that is more closely related with our goals in this chapter.

6.3 Formal Methods and Cyber-Physical Systems

Two of the biggest challenges in the science of cyber-physical systems is their design and debugging. In a recent study, Zheng et al [189] identified three important aspects regarding verification and validation of CPS:

1. "*[M]any CPS developers do not use traditional verification and validation methodologies and rely heavily on trial and error*".
2. "*[S]imulation alone is not enough to capture dangerous bugs in CPS*".

3. "*[I]t is widely acknowledged that the main challenges in CPS debugging are related to models of software systems, models of physics, and integration of Cyber and physics models.*"

All these findings support the fact that there is a necessity of (proprietary) formal tools developed for the design and development of CPS [190]. In this section we present some of the recent efforts made in the area of formal methods towards the design, verification and validation of CPS. We discuss some of the efforts made in the areas of automata theory, Petri nets, and process algebras in order to design and analyze CPS. Since the theory of process algebras is more relevant to our work, we shall discuss it in more depth in the next section and devote this section to automata theory and Petri nets only.

Automata Theory: In automata theory, given the hybrid aspects of CPS, the use of *hybrid automata* [148] to model the discrete and continuous dynamics of CPS is becoming popular. Banerjee and Gupta, in [191], developed a linear one-dimensional space, in spatio-temporal hybrid automaton to capture the spatio-temporal aspects of CPS. They utilized their approach to analyze the safe and unsafe conditions of an infusion pump control system. For more applications of hybrid automata to CPS and hybrid systems see [192–194].

Petri Nets: Petri nets (see Section 2.2.1) have been extended to *Hybrid Petri Nets* to capture mixtures between continuous and discrete behavior in CPS. Thacker et al, [195] proposed an approach that combines concepts of hybrid Petri nets and hybrid automata to describe interactions within CPS. Dalton et al, suggested use generalized *stochastic Petri nets*[2] for cyber-attack modeling [196]. Also, [197] applied Petri nets to describe the behavior of CPS in the presence of an intrusion detention and response system (IDRS); an IDRS is a system that must detect malicious nodes in the network

[2]The main difference between stochastic Petri nets and the (pure) Petri nets presented in section 2.2.1 is that transitions occur after random times.

(the CPS network in this case) without unnecessarily wasting energy to extend the lifetime of the system.

6.3.1 Calculi for the study of Cyber-Physical Systems

The theory of process algebra is one of the most promising theories for modeling and analysis of composition in CPS [187]. Some of the most promising extensions of process algebras to reason about CPS are the *real time process algebra* [198] and the *Hybrid Process Algebra* [199, 200].

More relevant to our work, there are two extensions of the π-calculus that have been created with the objective of describing the behavior of cyber-physical systems.

- The first one was proposed by Wang et al. in [150] as an extension of the π-calculus to include (discrete) time and position operators, called *Time-Space π-calculus*. An interpretation and definition of these operators is presented next:

  - The *time* operator $Int(t_r, \Delta t)$ is intended to capture the ability of a process to react only if it is in the interval of time $[t_r, t_r + \Delta t]$; i.e., given the (benchmark) time $\Delta t \geq 0$, the process $Int(t_r, \Delta t)P$ is a process that can start only if $t \in [t_r, t_r + \Delta t]$.
  - The *position* operator $Pos[S]$, is intended to capture the ability of a process to react only if it is in the a particular position; i.e., the process $Pos[S]P$ represents that the process P can start only when $\Pi_{i=1}^{n} c_i \in S$ is true, where c_i represents the ith physical component and S a particular (benchmark) region.

  The time-space π-calculus represents a reasonable extension, specially because they used these operators to define what they called the *CPS service substitution judgment theorem* which is – even though it was not stated in their paper as such– a notion of bisimulation. In this theorem they give the conditions for two processes to be substituted for one another.

- The second extension was proposed by Saeedloei and Gupta in [137]. This extension differs from the previously mentioned extension in the fact that time is considered to range over the real numbers (i.e. real time). Intuitively, Saeedloei and Gupta extended the π-calculus by adding a *clock* and operations over the clocks. This allows us to keep track of when the clock starts of resets and allows processes to be constrained over the clock. For the sake of illustration, consider the process $C\overline{x}\langle y, t_y, c\rangle$. This means that the name y, its time-stamp t_y and the clock c are sent over channel x, after performing the clock operation C. If, for example, the clock operation is of type $Constraint$, the process will send the message if the time constraint is satisfied, otherwise it will become inactive. e.g., $(c < 3)\overline{x}\langle y, t_y, c\rangle$ sends y, its time-stamp, and clock within 3 units of time since the clock c was reset.

  Saeedloei and Gupta provided the operational semantics and and encoding of the operational semantics in logic programming. This is a well posed extension and provides interesting insights of time (and clock) management in the context of π-calculus. They used their extension to describe the interactions between processes in the General Railroad Crossing (GRC) problem for the particular case of one track. We will discuss the GRC problem latter in this chapter.

Both of the discussed extensions of the π-calculus provide interesting ways to manage time and, in Wang's extension, position. We shall now provide our approach to deal with time in the calculus of decision systems.

6.4 Time and the CDS

The role of *time* in most of real systems is paramount. In this section we discuss our approach to dealing with time in the context of decision systems. Without a doubt time plays an important role by the time we are making a decision. However, once the decision is made, even though time continues, we cannot change our decision, only after certain time. Moreover, if we want to make our mind and change our

decision, that is whole new decision and therefore we are back when we started, from the standpoint of a decision system. Let us illustrate this.

Assume that we are to decide whether to buy or sale stock in a market. Time is running, and the market is continuously changing. At some point we decide to buy, this calls the process *buy* that will deal with the transaction. If for some reason, a couple of seconds later we change our mind and we want to sale instead, this will call the process *sale* that takes care of the transaction. Even though the two decisions are somehow connected, the epoch where the decisions are made is independent, i.e., if we disregard the time that process *buy* or process *sale* take to make the transaction, time during the moment we make a decision has a constant behavior. This is stated in the following remark.

Remark 6.4.1. *Given the continuity of time, the probability of making two decisions at exactly the same time is zero.*

Therefore, when a decision is made using the CDS, we (1) fix the time, and then (2) compute the decision process; we assume that reductions are made instantaneously. If a second decision is to be made using the same CDS-process, we need to re-compute using a new time value.

6.4.1 Time as Information

In Chapter 2 we discuss the role of information in decision systems. We argue that decision systems collect information from the environment and from their own internal processes, and then use that information to infer a decision using a decision function. Using a similar argument than Remark 6.4.1, the structure of a system at a fixed point of time is static, and may vary as time goes by. This leads us to consider time as information that is being input to the system; i.e., the structure of the system may change if the time changes. Therefore, in the context of the CDS, we include time the same way we include any other piece of information, by communicating it via communication channels. i.e., we do not add any additional operator to deal with

time in the CDS, we just treat it as a value that can be used in expressions and shared through communication channels.

We shall now present some examples of the use of time in the context of the CDS.

1. Let us assume we want to encode the time operator propped by Wang et al. Recall that the process $Int(t_r, \Delta t)P$ is a process that can start only when $t \in [t_r, t_r + \Delta t]$. We can encode this operator in the CDS as follows:

$$\texttt{if } (t > t_r \wedge t < t_r + \Delta t) \texttt{ then } P \texttt{ else } 0$$

2. Consider the system depicted in Figure 6.1. The force-displacement equation is given by $f(t) = -Kx(t)$. This system is a decision system with $Int = \{K\}$, and the decision maker is characterized by the decision function $f(t) = Kx(t)$. Note that the function $f(t)$ is defined by another function $x(t)$, therefore, we need to evaluate first $x = x(t)$ and then $f(x) = -Kx$. The CDS-term that characterizes this system is the following:

$$cx[y].\overline{c_1}\langle y\rangle \;||\; c_1 f[w].w$$

If we assume that $x(t) = \cos(t)$ and we want to study the response of the system at $t = 0$, we have the following reaction:

$$\overline{c}\langle 0\rangle \;||\; cx[y].\overline{c_1}\langle y\rangle \;||\; c_1 f[w].w \longrightarrow\longrightarrow -K$$

which is the state of the system at $t = 0$ given $x(t) = \cos(t)$. If we write $f(t) = K\cos(t)$, then this process can be simplified as follows:

$$\overline{c}\langle 0\rangle \;||\; cf[x].x \longrightarrow K$$

Giving us the same result in only one computational step.

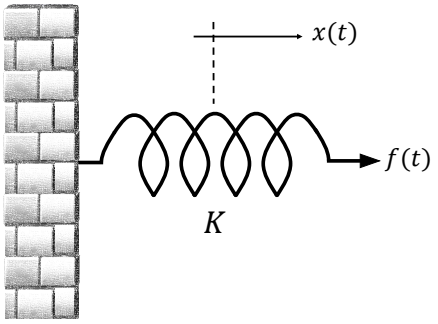


Figure 6.1. Spring with spring constant K.

6.5 Physical Systems in the CDS

In this section we present some basic ideas of how to deal with physical systems in the CDS. To this end, we use as a prototype system a transitional mechanical system. This type of systems is well-known in the area of general systems theory and – because of their analogy with electrical circuits (see [201])– can been used to describe not only mechanical and electrical systems but also systems in the areas of manufacturing, see [202]. By the time of this dissertation no one has implemented a formal method to describe this type of systems.

6.5.1 Transitional Mechanical Systems

In this section we use the CDS to describe the interactions between elements in a Translational Mechanical Systems (TMS) with passive[3] linear mechanical components. We do not intent to provide a comprehensive introduction to TMS. However, we provide the necessary concepts needed to understand our discussion. For a more elaborated introduction to TMS we refer to [203, p. 61]. Table 6.1 shows the three mechanical components and their force-velocity and force-displacement translational relationships for springs, viscous dampers and mass. The constant K (measured in

[3]It is referred as passive mechanical component to a mechanical component with no internal source of energy.

Newtons/meters (N/m)) is the spring constant, the constant V (measured in meter/second (m/s)) is the viscosity constant of the damper, and the constant M (measured in kilograms (kg)) is the mass. The force $f(t)$ is measured in Newtons (N).

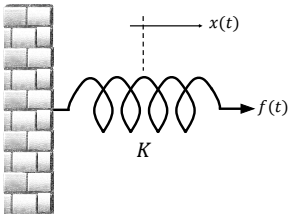
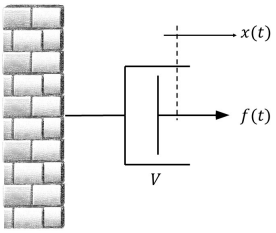
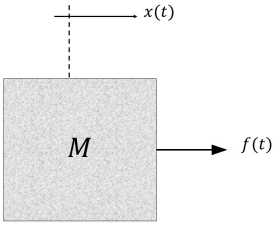
Component	**Force-Velocity**	**Force-displacement**
Spring 	$f(t) = K\int_0^t v(\tau)d\tau$	$f(t) = Kx(t)$
Viscous damper 	$f(t) = Vv(t)$	$f(t) = V\frac{dx(t)}{dt}$
Mass 	$f(t) = M\frac{dv(t)}{dt}$	$f(t) = M\frac{d^2x(t)}{dt^2}$

Table 6.1
Force-velocity and force-displacement translational relationships for mechanical components.

Given the TMS depicted in Figure 6.2. We can use Table 6.1, and set $F_s(x) = Kx$, $F_D(v) = Vv$, and $a_M(f) = \frac{f}{M}$ to be the decision functions defining the force-displacement relationships for the spring, damper and mass, respectively. Then, we can define each of the components as a CDS-term as follows:

(i) $Spring = new\ pos(\overline{c_1}\langle pos\rangle \parallel c_1 F_s[f_s].\overline{c_2}\langle f_s\rangle)$

(ii) $Damper = new\ vel(\overline{c_3}\langle vel\rangle \mid\mid c_3 F_D[f_D].\overline{c_4}\langle f_D\rangle)$

(iii) $Mass = c_2(x).c_4(y).\overline{c_5}\langle x+y\rangle \mid\mid c_5 a_M[a].a$

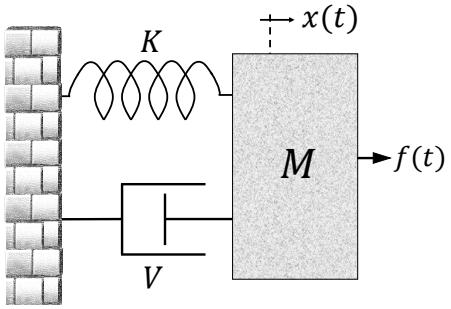


Figure 6.2. Translational Mechanical System.

Where *pos* and *vel* are the position and velocity of the spring and damper, respectively. Therefore, the CDS-process of the TMS depicted in Figure 6.2 is given by the following process:

$$TMS = Spring \mid\mid Damper \mid\mid Mass$$

Note that this process is reduced to the acceleration, a, of the mass; given the position of the spring and velocity of the damper. If we want to compute the acceleration of the mas repeatedly we have to redefine the processes as follows:

(i) $Spring \stackrel{\text{def}}{=} new\ pos(\overline{c_1}\langle pos\rangle \mid\mid c_1 F_s[f_s].\overline{c_2}\langle f_s\rangle) \cdot Spring$

(ii) $Damper \stackrel{\text{def}}{=} new\ vel(\overline{c_3}\langle vel\rangle \mid\mid c_3 F_D[f_D].\overline{c_4}\langle f_D\rangle) \cdot Damper$

(iii) $Mass \stackrel{\text{def}}{=} c_s(x).c_4(y).\overline{c_5}\langle x+y\rangle \mid\mid c_5 a_M[a].a \cdot Mass$

The processes *Spring* and *Damper* are outside of the scope of *pos* and *vel*, respectively, because it is assumed that every time we call each of these processes a new position and velocity will be provided. The LTS of this process is depicted in Figure 6.3. This assumes that the process starts concurrently in states S_1' and S_1''. The

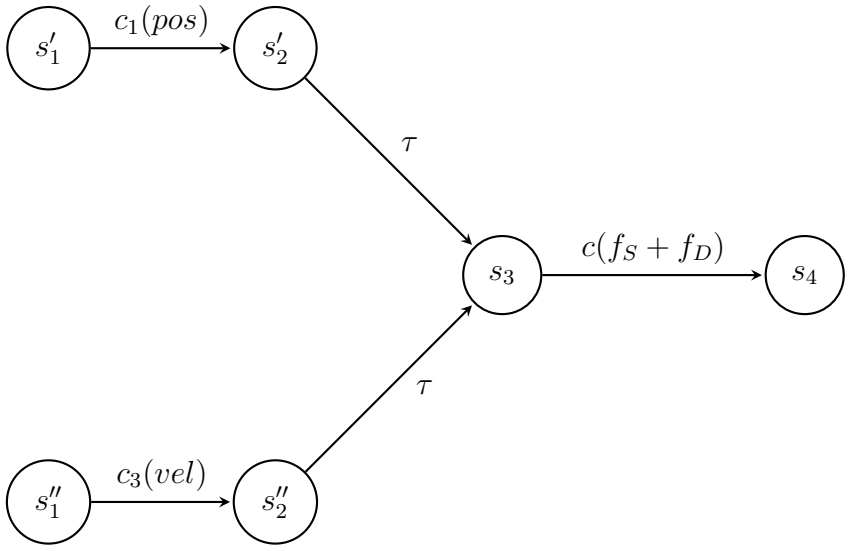


Figure 6.3. Labelled Transition Systems of a TMS.

LTS depicted in Figure 6.3 allows to see how the forces of the spring and damper act together on the mass. In this case we treated each mechanical component as different decision makers interacting in a structure by sharing their forces (decision) given their internal information. We could have treated this system in a more trivial way by assuming that the system as a whole is a single decision maker, with the decision function defined by the equation of motion of the system, but such case does not provide interesting insights about the interactions between the components.

Note that using the LTS and bisimulation we could find similarities between systems that are represented in different spaces; i.e., if two systems represented in two different spaces (e.g. state space or frequency space) are found behaviorally equivalent using the CDS, we will be able to relate the underlining principles of these systems, allowing us to investigate potential equivalences between systems and theories that have not been related yet.

We now turn our attention to systems which relate physical systems with cyber components, this systems are called cyber-physical systems (or CPS).

6.6 Using the CDS to study CPS

CPS are the coupling between cyber and physical systems. Formal methods have been created to reason about a variability of systems using a computational (cyber) approach. Therefore, by specifying physical systems using any formal method we are increasing the interoperability between computers and such physical system. In Section 6.3 we discussed some of the formal tools that have been developed to reason about cyber-physical systems. We have also discussed the importance of time in the study of CPS, and the role of process calculi in the development of this new area. Moreover, we have shown – in Section 6.4.1– how to deal with time in the CDS. The overall goal of this section is to show that the CDS can be used to reason about cyber-physical systems.

A well known problem that has been used as an example of a cyber-physical system in some of the work we have previously mentioned is the Generalized Railroad Crossing Problem. In this section, after giving an introduction to this problem, we use the CDS to characterize the processes involved in the Generalized Railroad Crossing problem for the spacial case of one railroad. It would be easy to see that extending this solution to multiple railroads is not complicated. We finish this section with a discussion of the results found.

6.6.1 The Generalized Railroad Crossing Problem

A typical cyber-physical system used in the literature for exemplifying the specification of a CPS, and various specification and validations methodologies is the Generalized Railroad Crossing Problem (GRC) problem; see [6–10]. This problem was proposed by Heitmeyer and Lynch [204] in 1994 as a case study in formal verification of real-time systems. Informally, the GRC problem consists of multiple train tracks and trains (trains travel in both directions), There is a gate at the crossing that should be operated so that it guarantees the following properties:

1. *Safety*: The gate must be down while one or more trains are crossing.
2. *Utility*: The gate must be up when there is no train in the crossing.

The formal statement of the GRC problem, taken directly from [205], is stated as follows:

> The system to be developed operates a gate at a railroad crossing. The railroad crossing I lies in a region of interest R, i.e., $I \subset R$. A set of trains travel through R on multiple tracks in both directions. A sensor system determines when each train enters and exits region R. To describe the system formally, we define a gate function $g(t) \in [0, 90]$, where $g(t) = 0$ means the gate is down and $g(t) = 90$ means the gate is up. We also define a set $\{\lambda_i\}$ of *occupancy intervals*, where each occupancy interval is a time interval during which one or more trains are in I. The ith occupancy interval is represented as $\lambda_i = [\tau_i, \nu_i]$, where τ_i is the time of the ith entry of a train into the crossing when no other train is in the crossing and ν_i is the first time since τ_i that no train is in the crossing (i.e., the train that entered at τ_i has exited as have any trains that entered the crossing after τ_i).
>
> Given two constants ξ_1 and ξ_2, $\xi_1 > 0$, $\xi_2 > 0$, the problem is to develop a system to operate the crossing gate that satisfies the following two properties:
>
> **Safety Property** $t \in \cup_i \lambda_i \Rightarrow g(t) = 0$
>
> **Utility Property** $t \notin \cup_i [\tau_i - \xi_1, \nu_i + \xi_2] \Rightarrow g(t) = 90$

The system is composed by three subsystems: a gate, a set of tracks and a controller (see Figure 6.4). Similar to [206] and [8] we study the GRC problem in its single-track version. The extension to multiple tracks should be trivial after the work presented here.

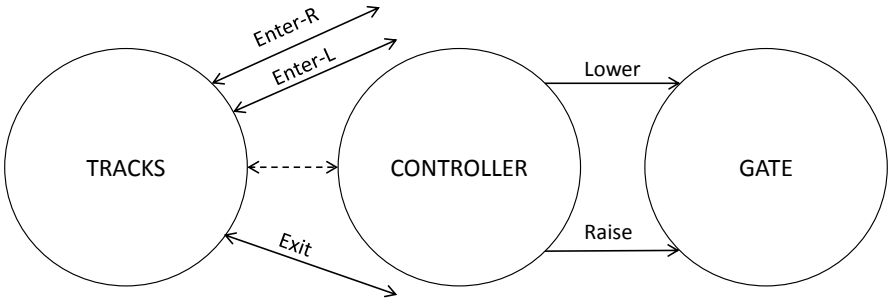


Figure 6.4. The General Railroad Crossing Problem.

6.6.1.1 The GRC Problem with One Track

To exemplify the use of the CDS to specify the GRC problem, we propose a modified version of it (see Figure 6.5) with the following assumptions:

1. There is only one track.

2. Trains are coming from right to left of the track.

3. There are no trains at the beginning.

The system is composed of two sensors, $Sensor_1$ and $Sensor_2$, managed by a controller, $Controller = Sensor_1 \parallel Sensor_2$, and a gate, $Gate$ (see Figure 6.6). Unlike [206], using our sensors we are able to know if the train is approaching or departing, making the system more reliable. The sensors would read 1 if a train is in front of them, and 0 if it is not.

We will solve this problem with two different approaches. First, we specify the system using the monoadic version of the CDS. This approach is similar than those found in [206] and [8]. The second approach makes use of the polyadic version of the CDS, allowing us to specify the system in a very simple manner and, under an

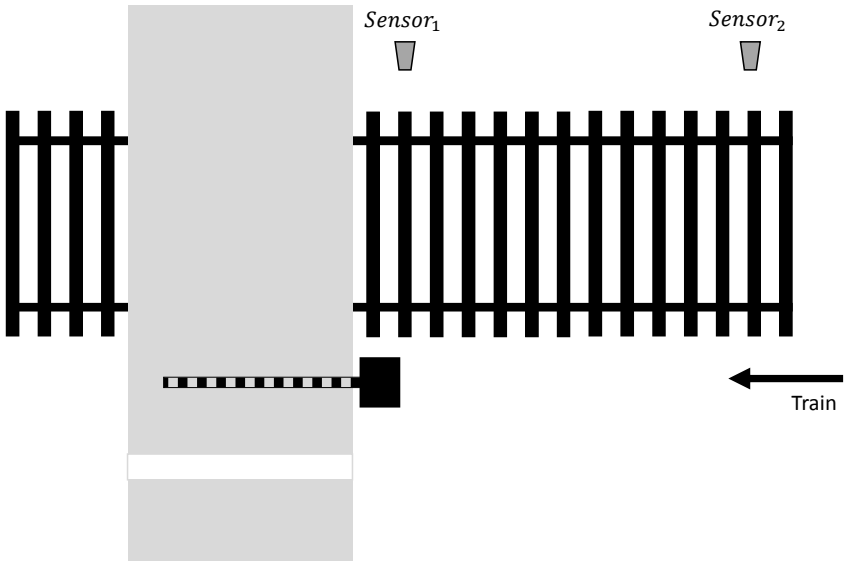


Figure 6.5. Railroad crossing.

extra assumption, allows us to increase the time the gate is opened; guaranteeing an improvement in *utility* – in the sense defined in the previous section.

First Approach Using the monoadic CDS, the railroad crossing problem is specified as follows:

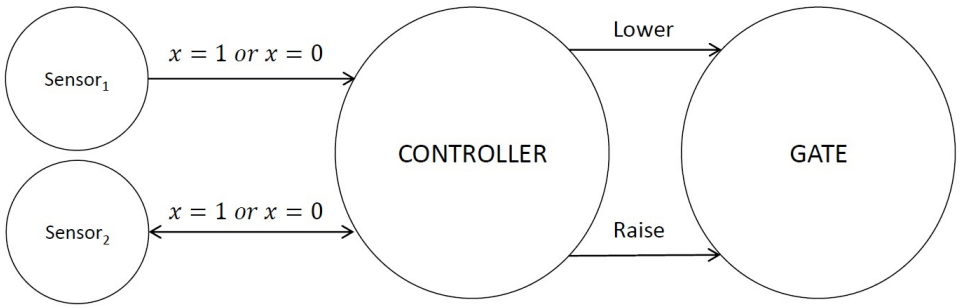


Figure 6.6. Railroad Crossing problem with one track.

$$
\begin{aligned}
Sensor_1 \stackrel{\text{def}}{=}\ & \overline{c_1}\langle x\rangle \mid\mid c_1(x).\texttt{if } x = 1 \texttt{ then } \overline{s_1}\langle appch\rangle.Gate \mid\mid \overline{count}\langle y+1\rangle \cdot Sensor_1 \\
& \texttt{else } count(y).\texttt{if } y > 0 \texttt{ then } \overline{s_1}\langle wait\rangle.Gate \mid\mid Sensor_1 \\
& \texttt{else } \overline{s_1}\langle open\rangle.Gate \mid\mid Sensor_1
\end{aligned}
$$

Discussion: After receiving (via channel c_1) whether the sensor is active ($x = 1$) or not active ($x = 0$), this process decides whether to send the message of *open* or *wait* to the gate. If $x = 1$ is communicated to the gate that a train is approaching; concurrently, it increases a counter in one, and returns to the state of "reading" from the environment. If $x = 0$ two cases might happen. (1) the train just passed, in which case it sends *wait* to the gate (so we can make sure the train passed $Sensor_2$), or (2) the train has not passed yet; this behavior is controlled by the channel *count*, which carries a counter that is reset to zero after both sensors are zero. The behavior of the process $Sensor_2$ is similar.

$$
\begin{aligned}
Sensor_2 \stackrel{\text{def}}{=}\ & \overline{c_2}\langle x\rangle \mid\mid c_2(x).\texttt{if } x = 1 \texttt{ then } \overline{s_1}\langle wait\rangle.Gate.\overline{s_2}\langle wait\rangle \mid\mid Sensor_2 \\
& \texttt{else } count(y).\texttt{if } y > 0 \texttt{ then } \overline{s_1}\langle wait\rangle.Gate \mid\mid Sensor_2 \\
& \texttt{else } \overline{s_2}\langle open\rangle.Gate \mid\mid Sensor_2
\end{aligned}
$$

and the gate behaves as,

$$
\begin{aligned}
Gate \stackrel{\text{def}}{=}\ & s_1(x).switch(x)\{ \\
& wait \Rightarrow s_2(x).\texttt{if } x = wait \texttt{ then } Gate \texttt{ else } \overline{rise} \mid\mid \overline{count}\langle 0\rangle.Sensor_1 \\
& appch \Rightarrow \overline{lower} \mid\mid \overline{count}\langle y+1\rangle.Sensor_2 \\
& open \Rightarrow \overline{rise} \mid\mid \overline{count}\langle 0\rangle.Sensor_1\}
\end{aligned}
$$

Discussion: This process waits until it receives a message from $Sensor_1$ via channel s_1. If both sensors say *wait* (which can only happen after $Sensor_1$ sent the message of approaching, "*appch*", first; which will lead to lowering the gate), it just loops around the process. The gate is only open after $Sensor_2$ sends the message *open*, meaning that the train is departing and it is safe to cross the railroad.

The atomic actions *rise* and *lower* capture the rising and lowering of the gate. Therefore, the GRC problem with one track is the process $\nu count(Controller \mid\mid Gate)$, where $Controller = Sensor_1 \mid\mid Sensor_2$.

Second Approach The second approach utilizes the polyadic CDS. We will assume that each reduction (i.e., each computational step), takes one unit of time to be performed. Therefore, if –for example – the train takes k units of time to go from $Sensor_1$ to $Sensor_2$, we will make the process perform k reductions before opening the gates. In this approach, we take advantage of the fact that the CDS allows us to use Boolean expressions between expressions. We have only two processes, the first process is the process $Sensors$ defined as follows:

$$\begin{aligned} Sensors \stackrel{\text{def}}{=} \; & \overline{c_1}\langle x\rangle.\overline{c_2}\langle y\rangle \;||\; c_1(x).c_2(y).switch((x,y))\{ \\ & x = 0 \wedge y = 0 \Rightarrow \overline{Gate}\langle(open, 0)\rangle \;||\; Sensors \\ & x = 1 \wedge y = 0 \Rightarrow \overline{Gate}\langle(close, k-l)\rangle \;||\; Sensors \\ & x = 1 \wedge y = 1 \Rightarrow \overline{Gate}\langle(close, 0)\rangle \;||\; Sensors \\ & x = 0 \wedge y = 1 \Rightarrow \overline{Gate}\langle(open, r)\rangle \;||\; Sensors\} \end{aligned}$$

Discussion: This process waits until it receives a message from $Sensor_1$ and $Sensor_2$, via c_1 and c_2, respectively. If both sensors read zero – meaning that there is no reading of a train in front of them – this process send a message requesting to raise the gate immediately (this is captured by the value '0' sent in the message). If $Sensor_1$ reads 0 and $Sensor_2$ reads 1, this means that a train is approaching. Thus, this process requests the lowering of the gate in $k - l$ units of time (recall that we assume that the train takes k units of time from $Sensor_1$ to $Sensor_2$), where l is the number of units of time we want to allow for crossing the railroad. The behavior is similar for the rest of the reading combinations of the sensors. It is worth mentioning that r is the number of units of time we want to allow after know that the train has completely departed.

The second process is called $GATE$. Again, the atomic actions *rise* and *lower* capture the rising and lowering of the gate.

$$
\begin{aligned}
GATE \stackrel{\text{def}}{=} \; & Gate(v,w).switch(v,w)\{ \\
& v = open \wedge w = 0 \Rightarrow \overline{raise} \;||\; GATE \\
& v = open \wedge w > 0 \Rightarrow \overline{Gate}\langle(open, w-1)\rangle \;||\; GATE \\
& v = close \wedge w = 0 \Rightarrow \overline{lower} \;||\; GATE \\
& v = close \wedge w > 0 \Rightarrow \overline{Gate}\langle(close, w-1)\rangle \;||\; GATE \;\}
\end{aligned}
$$

Discussion: This process waits until it receives a message from $Gate$. If the message is $(open, 0)$, it raises the gate immediately. If the message is $(open, w)$, with $w > 0$, it sends the message $(open, w-1)$ back to $GATE$. It will continue doing that, until $w = 0$, in which case it will rise the gate. The behavior of the rest of options is similar. Note that this process allows us to maximize the time we keep the gate open; i.e., the utility.

6.6.2 Discussion of Results

In this section we utilized the CDS for the specification of a transitional mechanical system and the generalized railroad crossing problem in the particular case of one track. The CDS-processes of the former system allowed us to compute the acceleration of the mass, M, given the position and velocity of the damper and spring, respectively. This exercise shows how to interpret a (simple) mechanical system as a decision system. The complexity of modeling this system with the CDS relied on the fact that we treated each component as individual decision makers who share information between each other without any strategy behind. This problem could have been simplified by assuming that the system as a whole is a decision maker having as utility function the equation of motion of the system; this case would have been solved using the CDS in a trivial manner, without providing any insights about the interaction between the mechanical components. Also, this problem could have been stated in a considerably more complex manner assuming that the decision makers (the components) share information strategically; e.g., the (three) decision makers could have been competing for the same resource (the position).

By using the CDS for the specification of the railroad crossing problem we were able to show the importance of polyadicity in process algebras. In the first characterization of the GRC problem we were able to describe the processes in a way that guarantees *safety* by avoiding deadlocks in the formulation. However, even though we maximize the *utility* of the gate as much as possible, the formulation did not allow us to easily consider time steps in the system, forcing us to close the gate right after the first sensor read 1. Fortunately, in the second formulation – using the polyadic CDS – we provided a simple formulation of the problem. This formulation gives us flexibility to select the time we want to allow for vehicles to cross the railroad, and also to specify the time each train takes from the first sensor to the second; therefore we maximized *utility*. Showing that *safety* is complied is obvious, since the process never deadlocks. *Utility* is maximized according to the specification of each instantiation of the problem; i.e., the times from one sensor to another may vary from one instantiation of the problem to another.

6.7 Advantages and Current limitations of the CDS in CPS

Advantages: The CDS presented some advantages in specifying mechanical systems and cyber-physical systems. The *function application* prefix simplified the specification of mechanical components in the transitional mechanical system. The fact that the CDS allows recursion in its syntax allowed us to specify this system in a more precise manner. In the case of cyber-physical systems, when we modeled the GRC problem, the CDS presented several advantages over other calculi. The *decision* process provided us with an easy way of specifying decision points, allowing us to clearly model the potential interactions between processes. Neither the timed π-calculus [8], nor timed automata [206] had the ability to clearly define the decisions made during the GRC problem. Also, none of the aforementioned approaches admit the specification of the allowed times for the railroad crossing and delay between approaching and arriving at the gate. Using the CDS we were able to relax some of the assumptions made in [9].

Current limitations: Currently there are some limitations in the syntax of the CDS, in the context of CPS. First, and foremost, we lack of a way of describing systems in a continuous time manner; we need to discretize time in the CDS. This constraint makes sense in the context of decision systems. As we argued previously, decisions are discretely made, even if the time is continuous. In the CDS we can make decision based on continuous time, but we cannot make decisions continuously. In the CDS we make single-shot decisions (or iterations) of the system, constraining the continuous behavior the system to a step-by-step behavior. This limits the CDS for the specification of CPS, because eventualities my happen during the time each process is been reduced; we partly took care of this limitation by assuming that reductions are made instantaneously. Another limitation of the CDS is the lack of a probabilistic choice operator; many CPS have stochastic behavior. Lastly, specifying mechanical systems using the CDS is not trivial, giving the nature of the CDS. Therefore, one of the limitations of the CDS for the specification of this type of systems is that the notation does not fit naturally, limiting its expressiveness.

6.8 Summary

In this chapter we studied the use of formal methods, in particular the CDS, for the specification of cyber-physical systems (CPS). We presented an overview of the science of CPS, and stated some of the characteristics of these systems. Also, we listed some of the challenge to be tackled by this new science. Subsequently, we presented an overview of automata theory, Petri nets and process calculi applied to CPS. Given the nature of the CDS, we put emphasis in the applications of calculi derived from π-calculus, for the specification of CPS. In Section 6.4 we presented how we deal with time in the CDS. We showed some examples of the use of time in our context. Then, in section 6.6 we use the CDS to specify (1) a transitional mechanical system and (2) the railroad crossing problem for the case of one track. We conclude the chapter with a discussion about the advantages and disadvantages of the CDS for the specification of CPS.

7. CONCLUSIONS

We finalize this dissertation by presenting an overview of the findings reported here, and discussing possible future research directions.

7.1 Summary

The focus of discussion is the creation of a process calculus for the analysis and characterization of decision systems. This dissertation is an attempt at understanding the applicability of process calculi in the context of decision systems. After clearly defining what a decision system is, we focused our attention on presenting the technical aspects of the calculus of decision systems (CDS), the notion of bisimulation in the context of decision systems, and the application of the CDS for the study of extensive games and cyber-physical systems.

The calculus of decision systems is comprised by (1) a syntax containing the alphabet and semantical constructs of the calculus, (2) a structural congruence between CDS-processes, and (3) an operational semantics containing all the reaction rules of the semantical constructs. The syntax defines the type of names, expressions and operations used for the specification of decision systems. The structural congruence is the smallest equivalence relation preserved by the CDS-processes constructs satisfying the set of axioms specified in Figure 3.6; i.e., two processes are structurally congruent, if they are identical up to structure. The operational semantics specifies how the constructs react in the context of CDS processes. More formally, the operational semantics is the smallest relation closed under the set of reduction rules of Definition 3.4.1. Also, we studied the polyadic version of the CDS. This means that expressions can not only be sent in singletons, but also sequences of expressions. In order to show a preliminary application of the CDS, in Section 3.6, we used all the

technicalities here mentioned in the specification of an aircraft acquisition problem.

The term bisimulation refers to the behavioral equivalence between systems. Two systems may deliver the same response, but their behavior could be different. In the context of the CDS, we defined a notion of bisimulation as an extension of the one created by Milner [121] for the CCS. This notion – the one proposed for the CDS – relies on the fact that two decision systems can behave equally, but, in order to be equivalent, the decision makers involved in the process have to be the same. For example, let us assume we have two decisions to make, one is whether to invest \$1 or \$2, and the second one whether to invest \$1,000,000 or \$1,100,000. These decisions may have the same behavior, but it highly depends on who is making the decision. Therefore, these decisions will be called (behaviorally) equivalent if not only they have the same behavior, but also they were made by the same decision maker. In order to study this equivalence, we created an action semantics for CDS-terms. This semantics defines the reductions between processes, where these reactions are triggered by actions.

Using formal methods for the study of games in extensive form allows us to reason about different properties of games. In the particular case of the CDS, we were able to characterize a notion of sub-game perfect equilibrium, and, using the action semantics of the CDS, we utilize bisimulation to compare the behavior of extensive games. The information sharing capabilities of the CDS allowed us to couple extensive games where one of the players is involved in more than one game. Using this couplings we show a potential extension to analyze the behavior of an organizational structures.

The study of cyber-physical systems is currently one of the most progressive areas of research. In this thesis we study the applicability of the CDS for the specification of this type of systems. We present how we manage time in the context of decision systems, and how our syntax allows us to characterize systems which share time or other information such as force. One of the transitional examples of cyber-physical systems is the generalized railroad crossing problem (GRC). We propose two solutions to this problem, for the case of one track. The difference in these solutions lies in

the communication capabilities of the the CDS; i.e., using the polyadic CDS and comparisons between expressions simplifies the solution of the GRC.

The applications presented in this dissertation are just a few of the vast number of applications that we believe can be explored using the concepts discussed here. We now present some of the potential research paths that can be pursued after this thesis.

7.2 Future Research

The application of formal methods for the specification of systems has opened many interesting questions, which can provide more insights into the study of decision systems. Here we present a list of potential topics which could be of interest.

1. Stochasticity plays an important role in many decision systems. Incorporating an operator for probabilistic choice will significantly increase the expressiveness of the CDS. We need to overlap the CDS with probabilistic process algebra which include stochastic process such as PEPA [142]. This will suggest several modifications to the syntax, operational semantics and action semantics of the CDS.

2. Creating a system of types for the CDS would add benefit and consistency to the CDS. In order to maintain communication in a more consistent manner, we could investigate the potential use of session types. This will guarantee that communication between expressions of different types is received and used in processes that recognize such expression as that specific type. Having a system of types will also allow us to prove soundness and completeness of terms.

3. Extending the CDS to a higher order communication where we can send not only names but processes may provide interesting insights in the study of decision systems. This capability will allow us to communicate decision systems between other decision systems; i.e., nested decision systems.

4. Investigating games in coalitional form (or characteristic function form) using the CDS may provide interesting insights about the communication between the different coalitions. This seems feasible since the CDS allows the use of real-valued functions in its syntax.

5. The function that finds the subgame perfect equilibrium, discussed in Section 5.4, can be extended for the general case with n-players, and general payoff functions.

6. The notion of bisimulation can be used to compare equivalences between electrical and mechanical systems. There are multiple (response) equivalences between these type of systems. Investigating these equivalences from the behavioral perspective may provide interesting insights.

7. The specification of cyber-physical systems has been exemplified using a simplification of the GRC problem. Using the CDS to specify more sophisticated CPS will provide a further understanding of the applicability of formal methods to this area.

8. Organizational structures have been briefly discussed in Chapter 5. There are different measures of coupling – such as loose coupling – in this context that could be investigated using the CDS. These measures may allow us to study couplings between organizations in a more formal manner.

9. Investigating the use of efficiency measures of an organizational structure as efficiency measures for communication channels in a CDS-process. Models such as Fordham model and Lockheed model could provide interesting insights to the area of process algebras.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Ludwig Von Bertalanffy. General system theory. *General systems*, 1:1–10, 1956.

[2] John P Van Gigch. *System design modeling and metamodeling*. Springer, 1991.

[3] Robin Milner. *Communication and concurrency*. Prentice-Hall, Inc., 1989.

[4] Jan A Bergstra, Alban Ponse, and Scott A Smolka. *Handbook of process algebra*. Elsevier, 2001.

[5] David Park. *Concurrency and automata on infinite sequences*. Springer, 1981.

[6] Curtis G Northcutt. Security of cyber-physical systems: A generalized algorithm for intrusion detection and determining security robustness of cyber physical systems using logical truth tables. *Vanderbilt Undergraduate Research Journal*, 9, 2013.

[7] Bingqing Xu, Jifeng He, and Lichen Zhang. Specification of cyber physical systems based on clock theory. *International Journal of Hybrid Information Technology*, 6(3):45–54, 2013.

[8] Neda Saeedloei and Gopal Gupta. Timed pi-calculus. *University of Texas at Dallas technical report*, 2008.

[9] Yuri Gurevich and James K Huggins. The railroad crossing problem: an experiment with instantaneous actions and immediate reactions. In *Computer Science Logic*, pages 266–290. Springer, 1996.

[10] Ernst-Rüdiger Olderog, Anders P Ravn, and Jens Ulrik Skakkebaek. Refining system requirements to program specifications. *Formal Methods for Real-Time Computing, Trends in Software*, 5:107–134, 1996.

[11] Merriam-Webster Inc. *Merriam-Webster's collegiate dictionary*. Merriam-Webster, 2004.

[12] J Edward Russo, Victoria Husted Medvec, and Margaret G Meloy. The distortion of information during decisions. *Organizational Behavior and Human Decision Processes*, 66(1):102–110, 1996.

[13] Robert Duncan Luce and Howard Raiffa. *Games and decisions: Introduction and critical survey*. Courier Dover Publications, 1957.

[14] Hossam A Gabbar. Fundamentals of formal methods. In *Modern Formal Methods and Applications*, pages 1–20. Springer, 2006.

[15] Dan Craigen, Susan Gerhart, and Ted Ralston. An international survey of industrial applications of formal methods. In *Z User Workshop, London 1992*, pages 1–5. Springer, 1993.

[16] Egidio Astesiano and Gianna Reggio. Formalism and method. In *TAPSOFT'97: Theory and Practice of Software Development*, pages 93–114. Springer, 1997.

[17] Jim Alves-Foss and Ann E Kelley Sobel. Formal methods and industry. In *Proceedings of the Thirty-Second Annual Hawaii International Conference on System Sciences-Volume 3-Volume 3*, page 3048. IEEE Computer Society, 1999.

[18] Jean-Raymond Abrial. Formal methods in industry: achievements, problems, future. In *Proceedings of the 28th international conference on Software engineering*, pages 761–768. ACM, 2006.

[19] Howard Pospesel. *Introduction to Logic: Propositional Logic, Revised Edition*, volume 3. Pearson, 1999.

[20] AM Turing. Intelligent machinery. report for national physical laboratory. reprinted in ince, dc (editor). 1992. mechanical intelligence: Collected works of am turing, 1948.

[21] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[22] George H Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[23] Edward F Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.

[24] Jiří Adámek, Jirí Adámek, and Vera Trnková. *Automata and algebras in categories*, volume 37. Springer, 1990.

[25] Ganesh Gopalakrishnan. *Computation engineering: applied automata theory and logic*. Springer, 2006.

[26] Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959.

[27] Alan M Turing. Computing machinery and intelligence. *Mind*, pages 433–460, 1950.

[28] Stephen Wolfram. *A new kind of science*, volume 5. Wolfram media Champaign, 2002.

[29] Moshe Y Vardi. Nontraditional applications of automata theory. In *Theoretical Aspects of Computer Software*, pages 575–597. Springer, 1994.

[30] Peter Linz. *An introduction to formal languages and automata*. Jones & Bartlett Publishers, 2011.

[31] John Lyons. *Natural Language and Universal Grammar: Volume 1: Essays in Linguistic Theory*, volume 1. Cambridge University Press, 1991.

[32] Noam Chomsky. On certain formal properties of grammars. *Information and control*, 2(2):137–167, 1959.

[33] Geoffrey K Pullum and Gerald Gazdar. Natural languages and context-free languages. *Linguistics and Philosophy*, 4(4):471–504, 1982.

[34] John W Backus. The syntax and semantics of the proposed international algebraic language of the zurich acm-gamm conference. *Proceedings of the International Comference on Information Processing, 1959*, 1959.

[35] John W Backus, Friedrich L Bauer, Julien Green, C Katz, John McCarthy, P Naur, Alan J Perlis, Heinz Rutishauser, Klaus Samelson, Bernard Vauquois, et al. Revised report on the algorithmic language algol 60. *The Computer Journal*, 5(4):349–367, 1963.

[36] Grzegorz Rozenberg and Arto Salomaa. *Handbook of Formal Languages: Beyonds words*, volume 3. Springer, 1997.

[37] Simona Ronchi Della Rocca. Abstract state machines-a method for high-level system design and analysis. *Computer Journal*, 47(2):270–271, 2004.

[38] Egon Börger. The asm method for system design and analysis. a tutorial introduction. In *Frontiers of Combining Systems*, pages 264–283. Springer, 2005.

[39] Egon Börger and Robert F Stärk. *Abstract State Machines: A Method for High-level System Design and Analysis; with 19 Tables*. Springer, 2003.

[40] Carl Adam Petri. Kommunikation mit automaten. *Bonn: Institut fur lnstrumentelle Mathematik*, 3, 1962.

[41] Carl Adam Petri. Communication with automata. *Technical Report No. RADC-TR-65-377*, 1966.

[42] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[43] James L Peterson. Petri nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252, 1977.

[44] Michel Hack. Petri net language. *Computation Structures Group Memo 124, Project MAC*, 1976.

[45] James L Peterson. Computation sequence sets. *Journal of Computer and System Sciences*, 13(1):1–24, 1976.

[46] TS Liu and SB Chiou. The application of petri nets to failure analysis. *Reliability Engineering & System Safety*, 57(2):129–142, 1997.

[47] Michael K. Molloy. Performance analysis using stochastic petri nets. *Computers, IEEE Transactions on*, 100(9):913–917, 1982.

[48] Alan A Desrochers and Robert Y Al-Jaar. *Applications of Petri nets in manufacturing systems: modeling, control, and performance analysis*, volume 70. IEEE press Piscataway^ eNJ NJ, 1995.

[49] Wil MP van der Aalst. The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998.

[50] René David and Hassane Alla. Petri nets for modeling of dynamic systems: A survey. *Automatica*, 30(2):175–202, 1994.

[51] Richard Zurawski and MengChu Zhou. Petri nets and industrial applications: A tutorial. *Industrial Electronics, IEEE Transactions on*, 41(6):567–583, 1994.

[52] Hossam A Gabbar. Formal methods for process systems engineering. In *Modern Formal Methods and Applications*, pages 21–35. Springer, 2006.

[53] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys (CSUR)*, 41(4):19, 2009.

[54] Anthony Hall. Seven myths of formal methods. *Software, IEEE*, 7(5):11–19, 1990.

[55] JV Hill, P Robinson, and PA Stokes. Safety critical software in control systems-a project view. In *Computers and Safety, 1989. A First International Conference on the Use of Programmable Electronic Systems in Safety Related Applications*, pages 92–96. IET, 1989.

[56] Ian Sommervillw. *Software Engineering*, volume 9th. edition. Addison-Wesley, 2010.

[57] J-R Abrial, Matthew KO Lee, DS Neilson, PN Scharbach, and Ib Holm Sørensen. The b-method. In *VDM'91 Formal Software Development Methods*, pages 398–405. Springer, 1991.

[58] Risto Hilpinen. On c. s. peirce's theory of the proposition: Peirce as a precursor of game-theoretical semantics. *The Monist*, 65(2):pp. 182–188, 1982.

[59] Wiebe van der Hoek and Marc Pauly. Modal logic for games and information. *Handbook of modal logic*, 3:1077–1148, 2006.

[60] Jaakko Hintikka. *Logic, language-games and information: Kantian themes in the philosophy of logic*. Clarendon Press Oxford, 1973.

[61] Jaakko Hintikka. Game-theoretical semantics: insights and prospects. In *The Game of Language*, pages 1–31. Springer, 1983.

[62] Paul Lorenzen and Kuno Lorenz. *Dialogische logik*. Wissenschaftliche Buchgesellschaft Darmstadt, 1978.

[63] Andreas Blass. Degrees of indeterminacy of games. *Fundamenta Mathematicae*, 77(2):151–166, 1972.

[64] Andreas Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56(13):183 – 220, 1992.

[65] Vincent Danos and Russell S Harmer. Probabilistic game semantics. *ACM Transactions on Computational Logic (TOCL)*, 3(3):359–382, 2002.

[66] Samson Abramsky. Algorithmic game semantics. In *Proof and System-Reliability*, pages 21–47. Springer, 2002.

[67] Julian Gutierrez. Logics and games for true concurrency. *arXiv preprint arXiv:1011.1172*, 2010.

[68] J Martin E Hyland and C-HL Ong. Pi-calculus, dialogue games and full abstraction pcf. In *Proceedings of the seventh international conference on Functional programming languages and computer architecture*, pages 96–107. ACM, 1995.

[69] Samson Abramsky and Guy McCusker. Game semantics. In *Computational logic*, pages 1–55. Springer, 1999.

[70] Samson Abramsky et al. Semantics of interaction: an introduction to game semantics. *Semantics and Logics of Computation*, 14:1, 1997.

[71] Brian F Chellas. *Modal logic: an introduction*, volume 316. Cambridge Univ Press, 1980.

[72] Rajeev Alur, Thomas A Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):672–713, 2002.

[73] David Harel. *Dynamic logic*. Springer, 1984.

[74] Patrick Blackburn, Maarten De Rijke, and Yde Venema. *Modal Logic: Graph. Darst*, volume 53. Cambridge University Press, 2002.

[75] G George Edward Hughes and Maxwell John Cresswell. *A new introduction to modal logic*. Psychology Press, 1996.

[76] Melvin Fitting and Richard L Mendelsohn. *First-order modal logic*, volume 277. Springer, 1998.

[77] Johan van Benthem, Jan van Eijck, and Vera Stebletsova. Modal logic, transition systems and processes. *Journal of Logic and Computation*, 4(5):811–855, 1994.

[78] Johan Van Benthem. Extensive games as process models. *Journal of logic, language and information*, 11(3):289–313, 2002.

[79] Marc Pauly. Logic for social software. *University of Amsterdam*, 2001.

[80] Marc Pauly. A logical framework for coalitional effectivity in dynamic procedures. *Bulletin of Economic Research*, 53(4):305–324, 2001.

[81] Marc Pauly. A modal logic for coalitional power in games. *Journal of logic and computation*, 12(1):149–166, 2002.

[82] Marc Pauly and Mike Wooldridge. Logic for mechanism designa manifesto. In *Proc. 5th Workshop on Game-theoretic and Decision-theoretic Agents*. Citeseer, 2003.

[83] Ronald Fagin, Joseph Y Halpern, Yoram Moses, and Moshe Y Vardi. *Reasoning about knowledge*, volume 4. MIT press Cambridge, 1995.

[84] Robert Stalnaker. Belief revision in games: forward and backward induction. *Mathematical Social Sciences*, 36(1):31–56, 1998.

[85] Emiliano Lorini and Frédéric Moisan. An epistemic logic of extensive games. *Electronic Notes in Theoretical Computer Science*, 278:245–260, 2011.

[86] Robert J Aumann. Backward induction and common knowledge of rationality. *Games and Economic Behavior*, 8(1):6–19, 1995.

[87] Robert J Aumann. Interactive epistemology i: knowledge. *International Journal of Game Theory*, 28(3):263–300, 1999.

[88] Robert J Aumann. Interactive epistemology ii: Probability. *International Journal of Game Theory*, 28(3):301–314, 1999.

[89] Johan Van Benthem. Extensive games as process models. *Journal of logic, language and information*, 11(3):289–313, 2002.

[90] Giacomo Bonanno. Modal logic and game theory: two alternative approaches. *Risk Decision and Policy*, 7(3):309–324, 2002.

[91] Pierpaolo Battigalli and Giacomo Bonanno. Recent results on belief, knowledge and the epistemic foundations of game theory. *Research in Economics*, 53(2):149–225, 1999.

[92] Wiebe van der Hoek and Marc Pauly. Modal logic for games and information. *Handbook of modal logic*, 3:1077–1148, 2006.

[93] Alexander Chagrov and Michael Zakharyaschev. *Modal logic*, volume 199. Clarendon Press Oxford, 1997.

[94] E Allen Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 995:1072, 1990.

[95] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[96] Edmund M Clarke and E Allen Emerson. *Design and synthesis of synchronization skeletons using branching time temporal logic*. Springer, 1982.

[97] E Allen Emerson and Joseph Y Halpern. sometimes and not never revisited: on branching versus linear time temporal logic. *Journal of the ACM (JACM)*, 33(1):151–178, 1986.

[98] Wiebe van der Hoek, Wojciech Jamroga, and Michael Wooldridge. A logic for strategic reasoning. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 157–164. ACM, 2005.

[99] Martin J Osborne and Ariel Rubinstein. *A course in game theory*. MIT press, 1994.

[100] Michael J Fischer and Richard E Ladner. Propositional dynamic logic of regular programs. *Journal of computer and system sciences*, 18(2):194–211, 1979.

[101] Vaughan R Pratt. Semantical consideration on floyo-hoare logic. In *Foundations of Computer Science, 1976., 17th Annual Symposium on*, pages 109–121. IEEE, 1976.

[102] Rohit Parikh. Propositional game logic. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 195–200. IEEE, 1983.

[103] Ramaswamy Ramanujam and Sunil Easaw Simon. Dynamic logic on games with structured strategies. In *KR*, pages 49–58, 2008.

[104] Philippe Balbiani. Propositional dynamic logic. *The Stanford Encyclopedia of Philosophy (Spring 2014 Edition)*, 2014.

[105] Johan van Benthem, Sujata Ghosh, and Fenrong Liu. Modelling simultaneous games in dynamic logic. *Synthese*, 165(2):247–268, 2008.

[106] Sujata Ghosh, Ramaswamy Ramanujam, and Sunil Simon. Playing extensive form games in parallel. In *Computational Logic in Multi-Agent Systems*, pages 153–170. Springer, 2010.

[107] Jos CM Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2):131–146, 2005.

[108] David Harel and Amir Pnueli. *On the development of reactive systems*. Springer, 1985.

[109] Jos CM Baeten, Twan Basten, Twan Basten, and MA Reniers. *Process Algebra: Equational Theories of Communicating Processes*, volume 50. Cambridge University Press, 2010.

[110] Alonzo Church. A set of postulates for the foundation of logic. *The Annals of Mathematics*, 33(2):346–366, 1932.

[111] Alonzo Church. *The calculi of lambda conversion*, volume 6. Princeton University Press, 1941.

[112] Raúl Rojas. A tutorial introduction to the lambda calculus. *Im Web unter: http://www. inf. fu-berlin. de/lehre/WS03/alpi/lambda. pdf*, 1997.

[113] Henk Barendregt, W Dekkers, and RICHARD Statman. Lambda calculus with types. *Handbook of logic in computer science*, 2:118–310, 1992.

[114] Gérard Huet. *Logical foundations of functional programming*. Addison-Wesley Longman Publishing Co., Inc., 1988.

[115] John C Mitchell. *Foundations for programming languages*, volume 1. MIT press Cambridge, 1996.

[116] Hendrik Pieter Barendregt. *The lambda calculus: Its syntax and semantics*, volume 103. North Holland, 1985.

[117] Felice Cardone and J Roger Hindley. History of lambda-calculus and combinatory logic. *Handbook of the History of Logic*, 5, 2006.

[118] Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245. Morgan Kaufmann Publishers Inc., 1973.

[119] Carl Hewitt. Actor model of computation: Scalable robust information systems. *arXiv preprint arXiv:1008.1459*, 2010.

[120] H Bekić. Towards a mathematical theory of processes. In *Programming Languages and Their Definition*, pages 168–206. Springer, 1984.

[121] Robin Milner. *A calculus of communicating systems*. Springer-Verlag New York, Inc., 1982.

[122] Robin Milner. Lectures on a calculus for communicating systems. In StephenD. Brookes, AndrewWilliam Roscoe, and Glynn Winskel, editors, *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 197–220. Springer Berlin Heidelberg, 1985.

[123] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[124] C. A. R. Hoare. A model for communicating sequential processes. *In R.M. McKeag and A.M. Macnaghten, editors, onthe Construction of Programs*, pages 229–254, 1989.

[125] Jan A Bergstra and Klop. Fixed point semantics in process algebras. *Mathematisch Centrum - Amsterdam*, 1982.

[126] Jan A Bergstra and Jan Willem Klop. Process algebra for synchronous communication. *Information and control*, 60(1):109–137, 1984.

[127] Robin Milner. *Communicating and mobile systems: the π calculus*. Cambridge university press, 1999.

[128] Luca Cardelli and Andrew D Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures*, pages 140–155. Springer, 1998.

[129] Luca Aceto. *Some of my favourite results in classic process algebra*. Citeseer, 2003.

[130] Kohei Honda. Types for dyadic interaction. In *CONCUR'93*, pages 509–523. Springer, 1993.

[131] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *ACM SIGPLAN Notices*, 43(1):273–284, 2008.

[132] Mariangiola Dezani-Ciancaglini, Dimitris Mostrous, Nobuko Yoshida, and Sophia Drossopoulou. Session types for object-oriented languages. In *ECOOP 2006–Object-Oriented Programming*, pages 328–352. Springer, 2006.

[133] Matthias Neubauer and Peter Thiemann. An implementation of session types. In *Practical Aspects of Declarative Languages*, pages 56–70. Springer, 2004.

[134] George M Reed and A William Roscoe. A timed model for communicating sequential processes. In *Automata, Languages and Programming*, pages 314–323. Springer, 1986.

[135] Faron Moller and Chris Tofts. *A temporal calculus of communicating systems*. Springer, 1990.

[136] Jos C. M. Baeten and Jan A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[137] Neda Saeedloei and Gopal Gupta. Timed pi-calculus. *University of Texas at Dallas technical report*, 2008.

[138] JCM Baeten, CA Middelburg, and MA Reniers. *A new equivalence for processes with timing*. Technische Universiteit Eindhoven, Department of Mathematics and Computer Science, 2002.

[139] Gavin Lowe et al. *Probabilities and priorities in timed CSP*. Oxford University Computing Laboratory, Programming Research Group, 1993.

[140] Jane Hillston. *A compositional approach to performance modelling*, volume 12. Cambridge University Press, 2005.

[141] Jos C. M. Baeten, Jan A. Bergstra, and Scott A. Smolka. Axiomatizing probabilistic processes: Acp with generative probabilities. *Information and Computation*, 121(2):234–255, 1995.

[142] Jane Hillston. *PEPA: Performance enhanced process algebra*. University of Edinburgh, Department of Computer Science, 1993.

[143] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Information and computation*, 100(1):1–40, 1992.

[144] William Denman, Mohamed H Zaki, Sofiène Tahar, and Luis Rodrigues. Towards flight control verification using automated theorem proving. In *NASA Formal Methods*, pages 89–100. Springer, 2011.

[145] Hossein Saiedian. An invitation to formal methods. *Computer*, 29(4):16–17, 1996.

[146] Michael Michael Gerard Hinchey and J Jonathan Peter Bowen. *Applications of formal methods*. Prentice Hall, 1995.

[147] Jean-Raymond Abrial, Egon Börger, and Hans Langmaack. *Formal methods for industrial applications: Specifying and programming the steam boiler control*, volume 9. Springer, 1996.

[148] Thomas A Henzinger. *The theory of hybrid automata*. Springer, 2000.

[149] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995.

[150] Peng Wang, Yang Xiang, and Shaohua Zhang. A novel reliability assurance method for cyberphysical system components substitution. *International Journal of Distributed Sensor Networks*, 2012, 2012.

[151] Robin Milner. *The polyadic π-calculus: a tutorial*. Springer, 1993.

[152] Jeannette M Wing. Faq on π-calculus. *Microsoft Research*, 2002.

[153] Joachim Parrow. An introduction to the-calculus. *Handbook of Process Algebra*, pages 479–543, 2001.

[154] Matthew Hennessy. *A distributed Pi-calculus*. Cambridge University Press, 2007.

[155] Corrado Priami. Stochastic π-calculus. *The Computer Journal*, 38(7):578–589, 1995.

[156] Davide Sangiorgi and David Walker. *The pi-calculus: a Theory of Mobile Processes*. Cambridge university press, 2003.

[157] Martín Abadi and Andrew D Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 36–47. ACM, 1997.

[158] Frank Puhlmann. Why do we actually need the pi-calculus for business process management. In *9th International Conference on Business Information Systems (BIS 2006)*, volume 85, pages 77–89, 2006.

[159] Howard Smith. Business process managementthe third wave: business process modelling language (bpml) and its pi-calculus foundations. *Information and Software Technology*, 45(15):1065–1069, 2003.

[160] Céline Kuttler, Cédric Lhoussaine, and Joachim Niehren. A stochastic pi calculus for concurrent objects. In *Algebraic Biology*, pages 232–246. Springer, 2007.

[161] Jim Laird. A game semantics of the asynchronous π-calculus. In *CONCUR 2005–Concurrency Theory*, pages 51–65. Springer, 2005.

[162] JCM Baeten, JA Bergstra, CAR Hoare, R Milner, J Parrow, and R de Simone. The variety of process algebra. *Deliverable ESPRIT Basic Research Action*, 3006, 1992.

[163] Jan A Bergstra, Wan Fokkink, and Alban Ponse. Process algebra with recursive operations. *Handbook of process algebra*, pages 333–389, 2001.

[164] Eike Best, Raymond Devillers, and Maciej Koutny. Petri nets, process algebras and concurrent programming languages. In *Lectures on Petri Nets II: Applications*, pages 1–84. Springer, 1998.

[165] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton university press, 2007.

[166] J Choi, Parithi Govindaraju, Navindran Davendralingam, and William Crossley. Platform design for fleet-level efficiency under uncertain demand: Application for air mobility command (amc). In *10th Annual Acquisition Research Symposium*, 2013.

[167] Martin J Osborne. *A course in game theory*. Cambridge, Mass.: MIT Press, 1994.

[168] Martin J Osborne. *An introduction to game theory*, volume 3. Oxford University Press New York, 2004.

[169] Harold W Kuhn. Extensive games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(10):570, 1950.

[170] Hans Peters. Extensive form games. *Game Theory: A Multi-Leveled Approach*, pages 197–212, 2008.

[171] Sujata Ghosh, Ramaswamy Ramanujam, and Sunil Simon. Playing extensive form games in parallel. In *Computational Logic in Multi-Agent Systems*, pages 153–170. Springer, 2010.

[172] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[173] Ulrich Frank. Multi-perspective enterprise modeling (memo) conceptual framework and modeling languages. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 1258–1267. IEEE, 2002.

[174] Assaf Arkin et al. Business process modeling language. *BPMI. org*, 2002.

[175] Howard Smith. Business process managementthe third wave: business process modelling language (bpml) and its pi-calculus foundations. *Information and Software Technology*, 45(15):1065–1069, 2003.

[176] Frank Puhlmann. Why do we actually need the pi-calculus for business process management? *BIS*, 85:77–89, 2006.

[177] Howard Smith and Peter Fingar. Workflow is just a pi process. *BPTrends, November*, 2003.

[178] Wil MP van der Aalst. Why workflow is not just a pi-process. *BP Trends*, pages 02–04, 2004.

[179] Rocco Carzo and John N Yanouzas. *Formal organization: A systems approach*. Irwin-Dorsey, 1967.

[180] V A Gralcunas. *Relationship in organization*. Bulletin of the International Management Institute (March), 1933.

[181] Ioan Dumitrache. Cyber-physical systems-new challenges for science and technology. *Journal of Control Engineering and Applied Informatics*, 13(3):3–4, 2011.

[182] Wayne Wolf. The good news and the bad news. *IEEE Computer*, 40(11):104–105, 2007.

[183] W. Wolf. Cyber-physical systems. *Computer*, 42(3):88–89, March 2009.

[184] Radhakisan Baheti and Helen Gill. Cyber-physical systems. *The Impact of Control Technology*, pages 161–166, 2011.

[185] Ragunathan Raj Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th Design Automation Conference*, pages 731–736. ACM, 2010.

[186] Lui Sha, Sathish Gopalakrishnan, Xue Liu, and Qixin Wang. Cyber-physical systems: A new frontier. In *Machine Learning in Cyber Trust*, pages 3–13. Springer, 2009.

[187] Kaiyu Wan, Danny Hughes, Ka Lok Man, and Tomas Krilavicius. Composition challenges and approaches for cyber physical systems. In *Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference on*, pages 1–7. IEEE, 2010.

[188] Edward A Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369. IEEE, 2008.

[189] Xi Zheng, Christine Julien, Miryung Kim, and Sarfraz Khurshid. On the state of the art in verification and validation in cyber physical systems. *Technical Report*, 2014.

[190] Edward A Lee. Cyber-physical systems-are computing foundations adequate. In *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, volume 2. Citeseer, 2006.

[191] Ayan Banerjee and Sandeep KS Gupta. Spatio-temporal hybrid automata for safe cyber-physical systems: A medical case study. In *Cyber-Physical Systems (ICCPS), 2013 ACM/IEEE International Conference on*, pages 71–80. IEEE, 2013.

[192] Alvaro Cardenas, Saurabh Amin, Bruno Sinopoli, Annarita Giani, Adrian Perrig, and Shankar Sastry. Challenges for securing cyber physical systems. In *Workshop on future directions in cyber-physical systems security*, 2009.

[193] Borzoo Bonakdarpour. Challenges in transformation of existing real-time embedded systems to cyber-physical systems. *ACM SIGBED Review*, 5(1):11, 2008.

[194] Tomas Krilavicius. *Hybrid techniques for hybrid systems*. University of Twente, 2006.

[195] Robert A Thacker, Kevin R Jones, Chris J Myers, and Hao Zheng. Automatic abstraction for verification of cyber-physical systems. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, pages 12–21. ACM, 2010.

[196] GC Dalton, Robert F Mills, John M Colombi, and Richard A Raines. Analyzing attack trees using generalized stochastic petri nets. In *Information Assurance Workshop, 2006 IEEE*, pages 116–123. IEEE, 2006.

[197] Robert Mitchell and Ing-Ray Chen. Effect of intrusion detection and response on reliability of cyber physical systems. *IEEE Transactions on Reliability*, 62(1):199–210, 2013.

[198] Jos CM Baeten and Jan A Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[199] Pieter Jan Laurens Cuijpers and Michel A Reniers. Hybrid process algebra. *The Journal of Logic and Algebraic Programming*, 62(2):191–245, 2005.

[200] Jan A Bergstra and Cornelis A Middelburg. Process algebra for hybrid systems. *Theoretical Computer Science*, 335(2):215–280, 2005.

[201] FA Firestone. A new analogy between mechanical and electrical systems. *The Journal of the Acoustical Society of America*, 4(3):249–267, 1933.

[202] Bashar H Sader and Carl D Sorensen. A new technique for modelling production control schemes in manufacturing systems. *International Journal of Production Research*, 48(23):7127–7157, 2010.

[203] Norman S Nise. *CONTROL SYSTEMS ENGINEERING, (With CD)*, volume 6th. Edition. John Wiley & Sons, 2010.

[204] Constance Heitmeyer and Nancy Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. Technical report, DTIC Document, 1994.

[205] Constance L Heitmeyer, BG Labaw, and RD Jeffords. A benchmark for comparing different approaches for specifying and verifying real-time systems. Technical report, DTIC Document, 1993.

[206] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

VITA

VITA

Jorge A. Samayoa holds a BS in Electronics and Computer Science (2003), a MS in Operation Research (2006) and a MS in Applied Mathematics (2009). In 2003 he started teaching undergraduate courses of Mathematics at the Engineering School of Galileo University and has also taught courses at Francisco Marroquín University, Guatemala and Texas A&M University, College Station, TX. In 2004 he founded the Teaching Assistants Department of Galileo University where he was responsible for all the Teaching Assistants of several Schools of Galileo University. In 2006 was awarded the Excellence in Teaching Award of Galileo University. From 2007 to 2014 he was teaching assistant and research assistant at Purdue University and Texas A&M University. In 2011 he started his Ph.D. under the supervision of Dr. Abhijit Deshmukh, in the school of Industrial Engineering at Purdue University in West Lafayette, IN. Subsequently, he returned to Galileo University as head of the Operations Research program and associate professor of Mathematics. His current research interests include Complex Systems, Process Algebras, π-calculus and systems coupling.