The Epistemic View of Concurrency Theory

Sophia Knight

The Epistemic View of Concurrency Theory

Sophia Knight

September 2013

Abstract

This dissertation describes three distinct but complementary ways in which epistemic reasoning plays a role in concurrency theory. The first and perhaps the one least explored so far is the idea of using epistemic modalities as programming constructs. Logic programming emerged under the slogan "Logic as a programming language" and the connection was manifest in a very clear way in the concurrent constraint programming paradigm. In the first part of the present thesis, we explore the role of epistemic, and closely related spatial modalities, as part of the programming language and not just as part of the meta-language for reasoning about protocols.

The next part explores a variant of dynamic epistemic logic adapted to labelled transition systems. In contrast to the previous part, one might be tempted to think that everything that can be said on this topic has already been said. However, the new ingredient that we propose is a tight connection between epistemic logic and Hennessy-Milner logic: *the* logic of labelled transition systems. We provide an axiomatization and prove a weak completeness theorem. This proof follows the general plan that one uses for logics like dynamic logic but requires some non-trivial adaptations.

The final part of the thesis focuses on the study of interacting agents in concurrent processes. We present a game semantics for agents' interaction which makes manifest the role of knowledge and information flow in the interactions between agents, and makes it possible to control the information available to the interacting agents. We use processes as the game board and define strategies for the agents so that two agents interacting according to their strategies determine the execution of the process, replacing the traditional scheduler. We show that different restrictions on the strategies represent different amounts of information being available to the scheduler. These restrictions on the strategies have an explicit epistemic flavour, and we present a modal logic for strategies and a logical characterization of several different possible restrictions on strategies.

These three approaches to the analysis and representation of epistemic information in concurrency theory provide a new way to understand agents' knowledge in concurrent processes, which is vital in today's world of ubiquitous distributed multi-agent systems.

Résumé

Le raisonnement epistémique joue un rôle en théorie de la concurrence de plusieurs manières distinctes mais complémentaires; cette thèse en décrit trois. La première, et presque certainement la moins explorée jusqu'à présent, est l'idée d'utiliser les modalités épistémiques comme éléments d'un langage de programmation. La programmation logique émergea sous le slogan «la logique en tant que langage de programmation» et dans le paradigme de la programmation concurrente par contraintes, le lien est manifeste de manière très claire. Dans la première partie de cette thèse, nous explorons le rôle des modalités épistémiques, ainsi que celui des modalités spatiales qui leur sont étroitement liées, en tant que partie intégrante du langage de programmation et non simplement en tant que partie du metalangage du raisonnement à propos des protocoles.

La partie suivante explore une variante de la logique épistémique dynamique adaptée aux systèmes de transitions étiquetées. Contrairement à la partie précédente, on serait tenté de croire que tout ce qu'on pouvait dire à ce sujet a déjà été dit. Cependant, le nouvel ingrédient que nous proposons est un lien étroit entre la logique épistémique et la logique de Hennessy-Milner, cette dernière étant *la* logique des systèmes de transitions étiquetées. Plus précisément, nous proposons une axiomatisation et une preuve d'un théorème de complétude faible, ce qui est conforme au principe général qu'on utilise pour des logiques telles que la logique dynamique mais nécessite des adaptations non triviales.

La dernière partie de la thèse se concentre sur l'étude d'agents en interaction dans les processus concurrents. Nous présentons une sémantique des jeux pour l'interaction d'agents qui rend manifeste le rôle de la connaissance et du flux d'information dans les interactions entre agents, et qui permet de contrôler l'information disponible aux agents en interaction. Nous utilisons les processus comme support de jeu et définissons des stratégies pour les agents de telle sorte que deux agents qui interagissent conformément à leurs stratégies respectives déterminent l'exécution du processus, remplaçant ainsi

l'ordonnanceur traditionnel. Nous démontrons que des restrictions différentes sur les stratégies réprésentent des quantités d'information différentes disponibles à l'ordonnanceur. Ces restrictions sur les stratégies ont un aspect épistémique explicite, et nous présentons une logique modale pour les stratégies et une caractérisation logique de plusieurs restrictions possibles sur les stratégies.

Ces trois approches d'analyse et de représentation de l'information épistémique en théorie de la concurrence apportent une nouvelle manière de comprendre la connaissance des agents dans des processus concurrents, ce qui est vital dans le monde d'aujourd'hui, dans lequel les systèmes distribués composés de multiples agents sont omniprésents.

Acknowledgments

I am deeply grateful to my supervisors, Frank Valencia and Catuscia Palamidessi. Frank has been incredibly supportive and dedicated. I greatly appreciate his encouragement and enthusiasm, and the time he has spent working with me, always listening to my ideas and helping me to realize them, and providing excellent guidance and wisdom about how to approach problems. I would like to thank Catuscia for her constant advocacy and help. I have benefitted greatly from her indispensable insight and expertise. Working in the team of Catuscia and Frank has been a unique and wonderful experience. I would also like to thank Prakash Panangaden for being deeply involved in the work presented here. Prakash is an inspiring person and has always been willing to help me and answer all my questions and I have been lucky and grateful to be able to work with him.

I would like to express my gratitude to the members of my jury for their careful evaluation and their many insightful and useful questions and comments which allowed me to improve this dissertation. Thanks to Jean-Pierre Jouannod, Stéphane Lengrand, and Hans van Ditmarsch. And I am especially grateful to Larry Moss, Vijay Saraswat, and Björn Victor, who took the time to read and critique my thesis in great detail and write detailed and helpful reports. I am honored to have had the chance to have such a qualified jury.

Many other people have helped me with this work. Thanks to Monica Dinculescu, Julia Evans, Sardaouna Hamadou, Kamal Marhubi, Mathieu Petitpas and Luis Pino for carefully reading my drafts and helping me to improve them greatly. Thanks to Matteo Cimini, Thomas Given-Wilson and Tobias Heindel for helping me to practice my defense and asking me so many questions. Thanks to Kostas Chatzikokolakis, Raluca Diaconu and Matteo Mio for numerous interesting and productive discussions of different aspects of this work. Thanks to Nico Bordenabe and Marco Stronati for helping me to organize the defense. Thanks to Marie-Jeanne Gaffard and Valérie Berthou for helping me deal with problems and adjust to living in

France. And thanks to Mario Alvim, Andrés Aristizabal, Alex Lang, Norm Ferns, Lili Xu, Agnes, and Rose for providing much needed moral support during my PhD.

Finally, I would like to thank my family, and most of all my husband Justin Dean. I couldn't have done this without you.

Contents

List of Figures

One

Introduction

This thesis describes the role that epistemic reasoning can play in concurrent systems. Concurrent processes are a natural and widely used model of interacting agents. Epistemic logic, on the other hand, is a formalism for reasoning about the knowledge of agents in such situations. Epistemic logic has played a central role in the analysis of distributed systems, [HM84, FHMV95] but in the area of concurrent processes, it has only been employed in a few specific ways. Work has been done using epistemic logic to analyze process calculi, for example [CDK09, DMO07, HS04] but to the best of our knowledge, this is the only way that epistemic logic has been applied to concurrent processes.

In this thesis, we explore three new ways of applying epistemic reasoning and modal logic to problems in concurrency theory. First, we develop a process calculus which uses epistemic modalities as programming constructs, allowing the expression of epistemic information within the process calculus. Second, we introduce a variant of dynamic logic adapted to labelled transition systems, enabling us to analyze the effects of actions on agents' knowledge in transition systems. Finally, we present a game semantics for interacting agents in concurrent processes. This semantics makes it possible to model agents with different epistemic capabilities, and the effect that their epistemic information has on the actions they are able to take. We hope to convince the reader that these three approaches to epistemic infor-

mation in concurrent systems are interesting and applicable to problems in distributed systems.

1.1 Context

Before introducing the main material in this thesis we give some context about the areas of study.

1.1.1 Epistemic Logic

Epistemic logic is a species of modal logic where the basic modality is of the form "Agent i knows fact phi": it is the logic of knowledge. Philosophers from all cultures, particularly Indian and Chinese Buddhist philosophers, Aristotle and other Greeks, the Port-Royal modal logicians of the middle ages and modern analytic philosophers, such as Chisholm, Ayer and Gettier, have debated the meaning of knowledge. The kind of epistemic logic that we consider is a philosophically simple variant that is nevertheless well adapted to the computational situations that we consider in our applications. For these applications it is not necessary to enter into the philosophical issues of what constitutes human knowledge: it suffices to consider a very simple notion of knowledge that justifies certain actions being taken at certain states of a protocol.

Although epistemic logic was discussed as a variety of modal logic earlier, particularly by von Wright [vW51], the specific form of epistemic logic we focus on was first really developed by Jaakko Hintikka [Hin62] and was based on Saul Kripke's so-called "possible worlds" semantics for modal logics. Hintikka's presentation was essentially semantic and it was an observation by Lemmon that this was an example of one of the well-known modal logics.

Epistemic logic began to be applied to computer science in the 1980s [HM84, FHMV95], when it was realized that epistemic concepts were involved in many important coordination and agreement protocols in distributed systems. Epistemic logic is usually used to verify that protocols

respect desired properties and as such form a powerful adjunct to other program logics. The kind of properties that are particularly well captured by epistemic logic are those related to agreement and coordination as already mentioned, but also to security properties when it becomes important to know what information is potentially being leaked out of a system in the course of the execution of a protocol.

1.1.2 Concurrency

The first models of computation, for example Turing Machines [Tur37], were fundamentally sequential: there was no ambiguity about the order in which computational steps could occur. The sequential computation paradigm evolved around some key central concepts: the λ-calculus as the basic unifying formalism and the notion of state and state transformation as the basic semantic framework. This framework accommodates a rich variety of developments: types and higher-type computation, program logics, operational semantics and denotational semantics to mention a few examples. The world of concurrent computation, by contrast, lacks a single central unifying concept like the λ-calculus.

Concurrency theory involves a much greater variety of possible phenomena than sequential computation. Once there are multiple autonomous processes functioning independently it is possible that these processes cooperate, compete for resources, act simultaneously, communicate but remain autonomous, or synchronize. The very basic temporal notions that we take so much for granted in sequential computing become major decision points when setting up a framework for concurrent computing. These features make concurrent systems quite general and allow them to be a relevant model of systems that are now common, but they also necessitate more decisions about the models of computation. Originally, concurrency was concerned mainly with operating systems, but now that new types of distributed systems such as social networks, interacting mobile devices, and cloud computing have become ubiquitous, concurrency theory is even more relevant and important.

Models of Concurrent Systems

Since concurrent systems are both widespread and inherently different from sequential systems of computation, many efforts have been made to develop accurate and understandable models for them. Ideally, these models are simple but still able to capture all the essential aspects of concurrent systems, and are also capable of being formally analyzed, to allow reasoning about the systems, as well as proofs of desired properties.

The first well known model of concurrent systems was Petri nets, which represent systems as a kind of graph with resources or tokens enabling transitions between states [Pet63]. Since then, there has been a wide variety of models of concurrent systems. For example, a labelled transition system is an extremely simple model of a concurrent system, consisting only of states and actions which may transition from one state to another.

Communication between agents is one of the fundamental aspects of concurrent systems, and models treat communication in one of two ways: synchronous or asynchronous. In a synchronous system, agents synchronize and communicate together at the same time, like in a telephone call. In asynchronous communication, one agent sends a message, which is later received by the other agent at an unspecified time. Email communication, for example, is asynchronous.

An early asynchronous model of concurrent systems was Jack Dennis' data flow model [Den74]. This model had a fixed network of autonomous computing agents with unidirectional communication channels linking them. The processes could read from and write to the communication channels, but if a process tried to read from an empty channel, where no message had yet arrived, it was blocked until it received a message on this channel. As we shall see later, the *Concurrent Constraint Programming* (CCP) paradigm has exactly this kind of asynchronous communication.

On the other hand, process calculi are another important class of models, on which we shall focus in this thesis. Besides a few exceptions, such as CCP, process calculi use synchronous communication. They are quite diverse, but the general idea is to represent computing agents or processes as algebraic

terms built from simple operators, and to provide an operational semantics defining the computational steps a process can take, based on its semantic form. Once a system is represented as a process calculus term, it should be easy to reason about important aspects of its behaviour, for example, what other systems it can be considered as equivalent to, and what behaviours it may or may not do.

One of the earliest process calculi was Robin Milner's Calculus of Communicating Systems, or CCS [Mil80]. This process calculus is quite simple but includes features that became common to many future process calculi, such as basic actions defined based on the intended application, nondeterministic choice between subprocesses, parallel execution of subprocesses, communication between two subprocesses, recursion, and restricting a subprocess from interacting with its environment in specific ways. CCS allows all of these kinds of behaviours but still is simple and general, so it became an important calculus for modelling computations. However, the simplicity of CCS masked a semantic complexity that took a long time to understand. Semantic models of CCS were slow to develop and, in the end were very operational in character.

Besides CCS, many other process calculi were developed for different purposes and with different advantages. For example, Hoare's Communicating Sequential Processes (CSP) was another early process calculus [Hoa85], with a more restricted notion of choice than in CCS. The Pi-Calculus is another process calculus [MPW92]. Pi-Calculus introduced the notion of *mobility*: instead of a fixed network of communicating agents, in the Pi-Calculus, channel names can be communicated over channels, making it possible for the communication structure between agents to change during the execution of a process. There have also been process calculi developed for specific purposes, such as the Spi-calculus [AG99], an extension of the Pi-calculus with built in cryptographic primitives such as encryption and decryption of messages. Innovation continues to occur in process calculi, for example, the family of process calculi known as Ambient Calculi allow a more general notion of mobile processes, to be better able to model modern computational situations such as mobile computing devices [CG00]. A

recent development is the Psi-Calculus [BJPV09], a process calculus with nominal data types which is general enough to encompass all the calculi mentioned above, and can model both synchronous and asynchronous communication.

Concurrent Constraint Programming

For this thesis, the process calculus called Concurrent Constraint Programming (CCP) [SRP91] is one of our main areas of interest. CCP is another formalism for concurrency, more closely based on logic than other process calculi. It was the first process calculus with asynchronous communication. In CCP, communication between agents occurs through shared variables in a *store*. The store contains partial information about variables: rather than simply assigning values to variables, the store can contain more complex information, called constraints, for example, "$X > 5$" or "$X + Y$ is even." Agents in CCP processes can add information to the store or ask the store for information, allowing a kind of synchronization between agents, because asking agents cannot continue until the store entails the information they are asking for.

In fact, the action of the process on the store is one of the most important aspects of CCP. The structure underlying the store is called the *constraint system* and is in fact a complete lattice. One of the important properties of basic CCP is *confluence*, meaning in essence that all reasonable computations of the same process will eventually reach the same outcome. A process can, therefore, be viewed as a function on the underlying constraint system, taking the original store as input and returning the final store that results from the execution of the process. Furthermore, CCP processes can only add information to a store, never remove it, and the final store of a process is a *fixed point* for that process: even if the process executed again starting from this store, it would not change it. These properties of CCP mean that processes can in fact be viewed as *closure operators* on the constraint system. Thus, there is a simple and elegant mathematical semantics for CCP. These results are presented in detail in Section 3.2, but the essential

point is that the asynchronous nature of CCP and its close ties to logic make elegant mathematical characterizations of its behaviour possible.

1.2 This Thesis: Epistemic Reasoning in Concurrent Systems

The goal of this thesis is to develop formalisms for concurrent systems that make epistemic information directly accessible within the formalism. Recently, concurrent systems have changed substantially with the advent of phenomena such as social networks and cloud computing. In past research on concurrent distributed systems [Lyn96] the emphasis has mostly been on consistency, fault tolerance, resource management and related topics; these aspects were all characterized by *interaction between processes*. The new era of concurrent systems is marked by the importance of managing access to information to a much greater degree than before.

Although this kind of analysis has not been common in process calculus research, epistemic concepts have long been understood to be crucial in distributed computing scenarios. Their importance was realized as early as the mid 1980s with Halpern and Moses' groundbreaking paper on common knowledge [HM84]. These ideas led to a flurry of activity in the following few years [FHMV95] with many distributed protocols being understood from an epistemic point of view. The impact of epistemic ideas in the concurrency theory community has been slower in coming.

There has been, however, some work concerning epistemic ideas in concurrency theory. For example, the goal of [CP07] was to restrict the epistemic information available to the scheduler in a probabilistic variant of the Pi-calculus, in order to avoid security problems of information leakage in the execution of processes. But even though epistemic ideas are central to this paper, formal epistemic logic is not used. Epistemic logic has, however, been used to *analyse* concurrent systems, for example in [CDK09], an epistemic logic is developed whose models are terms in the applied Pi-calculus. Similarly, [DMO07] presents a temporal epistemic logic for reasoning about

terms in a certain process calculus. In [HS04], a method is presented for formalizing information hiding properties in many process calculi, which is more general than the two approaches mentioned above, but still falls into the category of using an epistemic logic to analyze the properties of a term in a process calculus. In fact, as far as we know, all applications of epistemic logic to process calculi have consisted of using epistemic logic to analyze process calculus terms which are completely outside of the epistemic logic. In this thesis, we will present several approaches to including epistemic information directly within process calculi. This will allow us to encode agents' knowledge directly within processes, control information flow, analyze the effects of actions on agents' knowledge, and limit the actions agents are able to take based on the epistemic information available to them.

1.3 Outline and Contributions

This thesis has three parts. In Part I, we introduce a new constraint process calculus that allows the expression of epistemic information within the calculus. We also develop a notion of modal constraint system, underlying the process calculus and enabling us to use epistemic information to do computations, and we discuss characterizations of observable behaviour for these processes. In Part II, we present a variant of dynamic epistemic logic adapted to labelled transition systems. This logic allows us to examine the effects of actions on agents' knowledge in labelled transition systems. We give a sound and complete axiomatization of the logic. Finally, in Part III, we describe a game semantics for agents' interaction which makes manifest the role of knowledge and information flow in the interactions between agents, and makes it possible to control the information available to the interacting agents.

Besides these three parts, there are two introductory chapters, the first being the present introduction. Chapter 2 introduces some preliminary information about modal logic, which will be used throughout the rest of the thesis.

1.3.1 Part I- Epistemic Logic as a Programming Language

In Chapter 3 we review some notions in domain theory and concurrent constraint programming. Next, in Chapter 4, we introduce domain-theoretical structures to represent spatial and epistemic information. In Chapter 5, we present two new process calculi, based on the above-mentioned underlying constraint systems: Spatial CCP, with a new operator to represent a computation happening in a space belonging to an agent, and Epistemic CCP, with a new operator to represent an agent's knowledge of a computation. We also give an operational semantics for this process calculus. In Chapter 6, we present three notions of behaviour for the processes we have defined: limits, barbs, and denotational semantics. We prove that these three notions of behaviour coincide. Finally, in Chapter 7, we present some preliminary work on methods of approximating common knowledge as a compact or finite element, rather than as an infinite limit.

1.3.2 Part II- How Knowledge Evolves

This part of the thesis is concerned with a dynamic epistemic logic for a new kind of labelled transition systems which include epistemic equivalence relations for agents. In Chapter 8, we begin by defining the models for our dynamic epistemic logic as all the possible paths through a labelled transition system, maintaining the agents' equivalence relations on these paths. In Chapter 9, we define our dynamic epistemic logic and its semantics. In Chapter 10, we give an axiomatization for our logic and prove the completeness of the axiomatization.

1.3.3 Part III- Epistemic Strategies for Concurrent Processes

This part of the thesis presents a way of representing independent agents within a process calculus, and limiting the actions they can take based on their knowledge. In Chapter 11 we present a process calculus where each

action is labelled, and there is an independent choice operator representing a subprocess that is executed independently from the execution of the rest of the process. We give the syntax and labelled operational semantics for this process calculus. In Chapter 12, we present a game semantics for these processes, defining two player games with a process as the game board. We define strategies with epistemic restrictions on these games. The two players' choices of strategy define the execution of the process. In Chapter 13, we review a notion of syntactic schedulers in processes from [CP07], and then we prove that a certain class of the strategies defined in the last chapter corresponds exactly to these schedulers. In Chapter 14, we extend the results from the earlier sections to similar processes, but with an added probabilistic choice operator. In Chapter 9, we present some preliminary work on a modal logic for reasoning about the games we have defined in Chapter 12. This logic allows us to discuss formally the players' knowledge and the actions that are available to them.

1.4 Publications

Most of the results in this thesis have already appeared in scientific publications. More specifically:

- Part I is based on the paper **Spatial and Epistemic Modalities in Constraint-based Process Calculi** [KPPV12] that appeared in the proceedings of the 23^{rd} *International Conference on Concurrency Theory* (CONCUR 2012).
- Part II is based on the paper **Combining Epistemic Logic and Hennessy-Milner Logic** [KMP12], which was published in *Logic and Programming Semantics* in 2012.
- Part III is based on the paper **Epistemic Strategies and Games in Concurrent Processes** [CKPP12] that appeared in *ACM Transactions on Computational Logic* in 2012.

Two

Preliminaries on Modal Logic

In this chapter we present some basic information about modal logic, which will be relevant to the rest of this dissertation, mostly based on [BdRV01]. Of course, modal logic is a vast and complicated subject, but we will focus only on normal modal logics of relational structures. This relatively simple and well defined approach to modal logic is already quite expressive and has a wide range of applications. We only present a very brief overview here, for more details, see, for example, [BdRV01], or for the history of modal logic, see [Gol03].

2.1 Relational Structures and the Semantics of Modal Logic

We begin with the basic definitions of the models and semantics for modal logics of relational structures: what modal formulas are, and what structures they are intended to be interpreted over.

Definition 2.1.1 (Relational structure)**.** *A* relational structure *is a pair* $(W, \{R_i\}_{i \in I})$ *where* W *is any set, called the set of* states*, and each* R_i *is a binary relation on* W*, that is,* $R_i \subseteq W \times W$*.*

In principle, W and I may be arbitrary sets, but in this dissertation we only consider relational structures with countable sets of states and

relations. When $(v, w) \in R_i$, we write vR_iw.

Relational structures, sometimes also called Kripke structures, are ubiquitous in computer science, mathematics, and other fields. We present a few simple examples to illustrate the concept briefly.

Example 2.1.2. *The natural numbers, $(\mathbb{N}, \leq)$ are a relational structure, with $\mathbb{N}$ as the set of states and $\leq$ as the single relation on $\mathbb{N}$.*

Example 2.1.3. *Consider a relational structure $(S, \{\xrightarrow{a}\}_{a\in A})$, where S is a set of states and $\{\xrightarrow{a}\}_{a\in A}$ is a family of binary relations on S. If A represents a set of potential actions, then $(S, \{\xrightarrow{a}\}_{a\in A})$ is called a* labelled transition system. *When $w_1 \xrightarrow{a} w_2$, we say w_1 transitions to w_2 on action a.*

Example 2.1.4. *If we consider the set T to be moments in time, and define for $t_1t_2 \in T$, t_1Bt_2 only if t_1 is strictly before t_2, and t_1At_2 only if t_1 is strictly after t_2, then $(T, \{B, A\})$ is a relational structure.*

Now we will discuss modal languages and the logic of relational structures.

Definition 2.1.5 (Modal language)**.** *A* modal language Φ *is a set of* modal formulas *parametric in a set of propositions, P and a family of relations I, called the* modalities *of Φ. The modal formulas ϕ, ϕ_1, ϕ_2 of the language are defined as follows:*

$$\phi \ ::== \ p \mid \top \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \langle i\rangle\phi$$

where $p \in P$ and $i \in I$.

Definition 2.1.6 (Model)**.** *A* model *for a modal language with propositions P and modalities I is a tuple $M = (s, W, \{R_i\}_{i\in I}, V)$ where $s \in W$, $(W, \{R_i\}_{i\in I})$ is a relational structure, and $V : W \longrightarrow \mathcal{P}(P)$ is called a* valuation function. *It maps states to sets of propositions.*

The valuation function tells what formulas hold at what states. If $p \in V(s)$, we say that p *holds at* s or p *is true at* s.

Definition 2.1.7. *We now introduce the notion of* Kripke semantics. *If* $M = (s, W, \{R_i\}_{i\in I}, V)$ *is a model for a modal language* Φ *and* ϕ *is a modal formula such that* $\phi \in \Phi$ *then we define* M satisfying ϕ, *denoted* $M \models \phi$ *as follows*

$$\begin{array}{lll} M \models p & \textit{iff} & p \in V(s) \\ M \models \top & & \textit{always} \\ M \models \neg\phi & \textit{iff} & M \not\models \phi \\ M \models \phi_1 \wedge \phi_2 & \textit{iff} & M \models \phi_1 \textit{ and } M \models \phi_2 \\ M \models \langle i\rangle\phi & \textit{iff} & \exists t \in W \textit{ such that } sR_it \textit{ and } t \models \phi \end{array}$$

In situations where the relational structure $(W, \{R_i\}_{i\in I})$ and the valuation V are clear, we often omit these and write

$$s \models \phi$$

to mean

$$(s, W, \{R_i\}_{i\in I}, V) \models \phi.$$

Also, for a set of formulas Γ, if for all $\gamma \in \Gamma$ $M \models \gamma$, then we write

$$M \models \Gamma.$$

We can also define some standard derived operators.

Definition 2.1.8. *The following are some useful derived operators in modal logic.*

$$\begin{array}{rcl} \bot & \equiv & \neg\top \\ \phi_1 \vee \phi_2 & \equiv & \neg(\neg\phi_1 \wedge \neg\phi_2) \\ \phi_1 \Rightarrow \phi_2 & \equiv & \neg\phi_1 \vee \phi_2 \\ [i]\phi & \equiv & \neg\langle i\rangle\neg\phi \\ \phi_1 \Leftrightarrow \phi_2 & \equiv & (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1) \end{array}$$

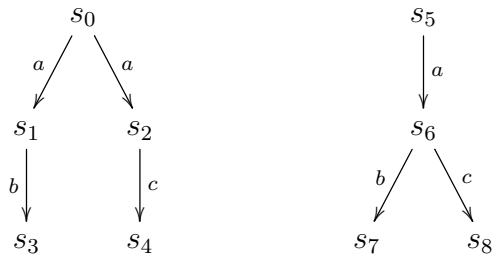
Theorem 2.1.9. *The derived operators defined above are satisfied by a model* $M = (s, W, \{R_i\}_{i\in I}, V)$ *in the following cases*

$$
\begin{array}{lll}
M \models \bot & & \textit{never} \\
M \models \phi_1 \vee \phi_2 & \textit{iff} & M \models \phi_1 \textit{ or } M \models \phi_2 \\
M \models \phi_1 \Rightarrow \phi_2 & \textit{iff} & M \models \neg\phi_1 \textit{ or } M \models \phi_2 \\
M \models [i]\phi & \textit{iff} & \textit{for all } t \in \{t \mid sR_it\}, t \models \phi \\
M \models \phi_1 \Leftrightarrow \phi_2 & \textit{iff} & \textit{either } M \models \phi_1 \wedge \phi_2 \textit{ or } M \models \neg\phi_1 \wedge \neg\phi_2
\end{array}
$$

The proof follows directly from Definition 2.1.7 and the definitions of the derived operators.

Here is a simple example of a relational structure and some formulas that are true at some of the states.

Example 2.1.10. *The picture shows a labelled transition system. The states are* $\{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$ *and the actions are* $\{a, b, c\}$*. The relations are shown below as arrows. We assume the set of propositions is empty, so the only possible valuation maps every state to the empty set of propositions.*



Here are some formulas that are satisfied at certain states:

$$
\begin{array}{ll}
s_0 \models \langle a\rangle\top & s_0 \models \langle a\rangle\langle b\rangle\top \wedge \langle a\rangle\neg\langle b\rangle\top \\
s_0 \models \langle a\rangle\langle b\rangle\top \wedge \langle a\rangle\langle c\rangle\top & s_5 \models \langle a\rangle\left(\langle b\rangle\top \wedge \langle c\rangle\top\right) \\
s_1 \models \neg\langle c\rangle\top & s_5 \models [a]\langle b\rangle\top \\
s_0 \models \langle a\rangle\neg\langle c\rangle\top & s_0 \models \neg[a]\langle b\rangle\top
\end{array}
$$

This kind of modal logic, especially when considered over finitely branching labelled transition systems, is known as Hennessy-Milner logic *[HM80]. It is best known for its relationship with bisimulation.*

Definition 2.1.11. *Consider a labelled transition system $(S, \{\xrightarrow{a}\}_{a\in A})$. A relation $B \subseteq S \times S$ is a* bisimulation *if the following three conditions hold:*

- *B is symmetric,*
- *Whenever $(s,t) \in B$, if $s \xrightarrow{a} s'$ then there exists t' such that $t \xrightarrow{a} t'$ and $(s',t') \in B$,*
- *Similarly, whenever $(s,t) \in B$, if $t \xrightarrow{a} t'$ then there exists s' such that $s \xrightarrow{a} s'$ and $(s',t') \in B$.*

We say that states s and t are bisimilar if and only if there is a bisimulation B such that $(s,t) \in B$.

The following theorem says that Hennessy-Milner logic characterizes bisimulation.

Theorem 2.1.12. *In a finitely branching labelled transition system, states s and t are bisimilar if and only if for all formulas ϕ in Hennessy-Milner logic, $s \models \phi$ if and only if $t \models \phi$.*

2.2 Validity, Soundness and Completeness

First we will define a certain set of formulas called a *normal modal logic*. In the following definitions, we assume a certain set of modalities I that can occur in our formulas.

Definition 2.2.1 (Modus ponens)**.** *A set S of formulas is closed under* modus ponens *if whenever $\alpha \in S$ and $\alpha \Rightarrow \beta \in S$, $\beta \in S$.*

Definition 2.2.2 (Uniform substitution)**.** *A set S of formulas is closed under* uniform substitution *if whenever $\alpha \in S$, if β results from substituting a specific formula for every instance of a specific propositional variable in α, then $\beta \in S$.*

Definition 2.2.3 (Generalization)**.** *A set S of formulas is closed under* generalization *if whenever $\alpha \in S$ then $[i]\alpha \in S$, for all $i \in I$.*

Definition 2.2.4 (Normal modal logic)**.** *A set of modal formulas Λ is a* normal modal logic *if Λ contains all tautologies from propositional logic, the formulas $[i](p_1 \Rightarrow p_2) \Rightarrow ([i]p_1 \Rightarrow [i]p_2)$ for all $i \in I$, and it is closed under modus ponens, uniform substitution, and generalization.*

Proposition 2.2.5. *For any set of modal formulas A, there is a smallest normal modal logic Λ such that $A \subseteq \Lambda$.*

Definition 2.2.6 (Logic generated by a set)**.** *If A is a set of modal formulas, we call the smallest normal modal logic Λ such that $A \subseteq \Lambda$ the* logic generated by *A.*

Definition 2.2.7 (Proof)**.** *Consider a finite set of modal formulas A. A* proof of ϕ_n *from A is a finite sequence $\phi_1, \phi_2, ..., \phi_n$ of modal formulas so that for each $i \in \{1, ..., n\}$ one of the following is true:*

1. *$\phi_i \in A$.*
2. *ϕ_i is a propositional tautology.*
3. *There exists $j < i$ and $k < i$ such that $\phi_k = \phi_j \Rightarrow \phi_i$.*
4. *ϕ_i is the result of uniformly substituting a formula for every instance of a proposition in ϕ_j where $j < i$.*
5. *$\phi_i = [m]\phi_j$ where m is a modality and $j < i$.*

Notice that a normal modal logic Λ contains all the formulas that can be proved from finite subsets of Λ.

Definition 2.2.8 (Validity for a state)**.** *Consider a modal language Φ with propositions P and modalities I, let $M = (W, \{R_i\}_{i \in I})$ be a model for Φ, and let s be a state in W. Let ϕ be a formula in Φ. We say that ϕ is* valid *at s in M if for all valuations $V : W \longrightarrow \mathcal{P}(P)$,*

$$(s, W, \{R_i\}_{i \in I}, V) \models \phi.$$

We write this as

$$(s, M) \models \phi.$$

Definition 2.2.9 (Validity for a structure)**.** *Consider a modal language* Φ *with propositions* P *and modalities* I*, and let* $M = (W, \{R_i\}_{i \in I})$ *be a model for* Φ*. Let* ϕ *be a formula in* Φ*. We say that* ϕ *is* valid *for* M *if for all* $s \in W$*,*

$$(s, M) \models \phi.$$

We write this as

$$\models_M \phi.$$

In other words, a formula is valid for a relational structure if the formula is true at every state, regardless of the valuation on the structure.

Definition 2.2.10 (Validity for a class)**.** *Let* $\mathbf{M}$ *be a class of relational structures. For a modal formula* ϕ*, we say that* ϕ *is* valid *for* $\mathbf{M}$ *if for all* $M \in \mathbf{M}$*,* ϕ *is valid for* M*. We write this as*

$$\models_{\mathbf{M}} \phi.$$

Definition 2.2.11 (Soundness)**.** *Consider a set of modal formulas* A *and a class of relational structures* $\mathbf{M}$*. Let* Λ *be the normal modal logic generated by* A*. If for every* $\phi \in \Lambda$*,* ϕ *is valid for* $\mathbf{M}$*, then we say that* A *is* sound *for* $\mathbf{M}$*.*

Definition 2.2.12 (Deducibility in a logic)**.** *If* Λ *is a normal modal logic, we say that a formula* ϕ is deducible in Λ *if* $\phi \in \Lambda$*. We denote this as*

$$\vdash_\Lambda \phi.$$

Notice that with this notation, the soundness of normal modal logic Λ for a class of relational structures $\mathbf{M}$ can be expressed as

$$\text{if } \vdash_\Lambda \phi \text{ then } \models_{\mathbf{M}} \phi.$$

Definition 2.2.13 (Deducibility from a set)**.** *If* Λ *is a normal modal logic and* Γ *is a set of formulas, we say that a formula* ϕ is deducible from Γ in Λ *if there are formulas* $\gamma_1, ..., \gamma_n \in \Gamma$ *such that*

$$\vdash_\Lambda (\gamma_1 \wedge \gamma_2 \wedge ... \wedge \gamma_n) \Rightarrow \phi.$$

We denote this as

$$\Gamma \vdash_\Lambda \phi.$$

Definition 2.2.14 (Local semantic consequence)**.** *Let* $\mathbf{M}$ *be a class of relational structures, let* Γ *be a set of formulas and let* ϕ *be a formula. We say that* ϕ *is a* local semantic consequence *of* Γ *in* $\mathbf{M}$ *if for all* $(W, \{R_i\}_{i\in I}) \in \mathbf{M}$*, for all* $s \in W$ *and for all valuations* V*, if whenever*

$$(s, W, \{R_i\}_{i\in I}, V) \models \Gamma$$

then

$$(s, W, \{R_i\}_{i\in I}, V) \models \phi.$$

We denote this as

$$\Gamma \models_{\mathbf{M}} \phi.$$

Definition 2.2.15 (Strong completeness)**.** *Let* $\mathbf{M}$ *be a class of relational structures, let* A *be a set of formulas, and let* Λ *be the normal modal logic generated by* A*. We say that* A *is* strongly complete *for* $\mathbf{M}$ *if for all sets of formulas* Γ *and all formulas* ϕ*,*

$$\textit{if } \Gamma \models_{\mathbf{M}} \phi \textit{ then } \Gamma \vdash_\Lambda \phi.$$

In other words, A being strongly complete for $\mathbf{M}$ means that if any ϕ is a local semantic consequence of any Γ in $\mathbf{M}$ then ϕ is deducible from Γ in the normal modal logic generated by A.

Definition 2.2.16 (Weak completeness)**.** *Let* $\mathbf{M}$ *be a class of relational structures, let* A *be a set of formulas, and let* Λ *be the normal modal logic generated by* A*. We say that* A *is* weakly complete *for* $\mathbf{M}$ *if for any formula* ϕ*,*

$$\textit{if } \models_{\mathbf{M}} \phi \textit{ then } \vdash_\Lambda \phi.$$

So A is weakly complete for $\mathbf{M}$ if every formula that is valid for $\mathbf{M}$ is deducible in the logic generated by A. Note that strong completeness implies weak completeness.

We need the following definitions for an important result about completeness.

Definition 2.2.17 (Satisfiability for a formula)**.** *Consider a relational structure $M = (W, \{R_i\}_{i \in I})$ and a formula ϕ. ϕ is* satisfiable *in M if there exists a state $s \in W$ and a valuation V such that*

$$(s, W, \{R_i\}_{i \in I}, V) \models \phi.$$

Definition 2.2.18 (Satisfiability for a set)**.** *Consider a relational structure $M = (W, \{R_i\}_{i \in I})$ and a set of formulas Γ. Γ is* satisfiable *in M if there exists a state $s \in W$ and a valuation V such that*

$$(s, W, \{R_i\}_{i \in I}, V) \models \Gamma.$$

Definition 2.2.19 (Consistency)**.** *Let Λ be a normal modal logic and let Γ be a set of formulas. We say that Γ is* Λ-consistent *if $\Gamma \nvdash_\Lambda \bot$. For a formula ϕ, we say that ϕ is Λ-consistent if the set $\{\phi\}$ is Λ-consistent.*

The following result provides useful characterizations of both strong and weak completeness.

Proposition 2.2.20. *Let Λ be a normal modal logic and let $\mathbf{M}$ be a class of relational structures.*

- *Λ is strongly complete for $\mathbf{M}$ if and only if for every Λ-consistent set of formulas Γ, there exists $M \in \mathbf{M}$ such that Γ is satisfiable in M.*
- *Λ is weakly complete for $\mathbf{M}$ if and only if for every Λ-consistent formula ϕ, there exists $M \in \mathbf{M}$ such that ϕ is satisfiable in M.*

The proof can be found for example in [BdRV01].

2.3 Specific Modal Logics

The notions of soundness and completeness allow us to use a set of axioms to characterize a class of structures. In this section, we present several specific modal logics based on certain structures, and the axioms that characterize them. We say that a set of axioms characterizes a class of structures if the normal modal logic generated by the set of axioms is sound and (strongly or weakly) complete for that class of structures. The proofs for all of the results in this section can be found in [BdRV01] or [FHMV95].

2.3.1 $\mathbf{K}_n$

First we discuss a very general normal modal logic, $\mathbf{K}_n$, where $n \in \mathbb{N}$. It is based on instances of the K axiom, which stands for Kripke, and it is the smallest normal modal logic for a set of n modalities.

Definition 2.3.1 (K axiom)**.** *For a modality i, the K* axiom *is*

$$[i](p \Rightarrow q) \Rightarrow ([i]p \Rightarrow [i]q).$$

In fact, for technical reasons we already defined a normal modal logic for modalities I already to contain axiom K for all $i \in I$. But we also include it as an axiom to simplify some of the following results.

Definition 2.3.2 ($\mathbf{K}_n$)**.** $\boldsymbol{K}_n$ *is the normal modal logic generated by the axioms*

$$\begin{gathered}[1](p \Rightarrow q) \Rightarrow ([1]p \Rightarrow [1]q) \\ [2](p \Rightarrow q) \Rightarrow ([2]p \Rightarrow [2]q) \\ \vdots \\ [n](p \Rightarrow q) \Rightarrow ([n]p \Rightarrow [n]q)\end{gathered}$$

Theorem 2.3.3. *Let $\mathbf{M}_n$ be the class of all relational structures with modalities $\{1, ..., n\}$. $\boldsymbol{K}_n$ is sound and strongly complete for $\mathbf{M}_n$.*

2.3.2 S4

Definition 2.3.4 (T axiom)**.** *For a modality i, the T* axiom *is*

$$p \Rightarrow \langle i \rangle p.$$

Definition 2.3.5 (**T**)**.** ***T*** *is the normal modal logic generated by the T axiom.*

Definition 2.3.6 (Reflexivity)**.** *A relation $R \subseteq W \times W$ is* reflexive *if for all $w \in W$, wRw.*

Theorem 2.3.7. *Let $\mathbf{M}^r$ be the class of all relational structures with a single, reflexive relation.* ***T*** *is sound and strongly complete for $\mathbf{M}^r$.*

Definition 2.3.8 (4 axiom)**.** *For a modality i, the* 4 axiom *is*

$$[i]p \Rightarrow [i][i]p.$$

Definition 2.3.9 (**K4**)**.** ***K4*** *is the normal modal logic generated by the 4 axiom.*

Definition 2.3.10 (Transitivity)**.** *A relation $R \subseteq W \times W$ is* transitive *if for all $w_1, w_2, w_3 \in W$, whenever w_1Rw_2 and w_2Rw_3, it follows that w_1Rw_3.*

Theorem 2.3.11. *Let $\mathbf{M}^t$ be the class of all relational structures with a single, transitive relation.* ***K4*** *is sound and strongly complete for $\mathbf{M}^t$.*

Definition 2.3.12 (**S4**)**.** ***S4*** *is the normal modal logic generated by the set consisting of the T axiom and the 4 axiom.*

Theorem 2.3.13. *Let $\mathbf{M}^{rt}$ be the class of all relational structures with a single relation which is transitive and reflexive.* ***S4*** *is sound and strongly complete for $\mathbf{M}^{rt}$.*

2.3.3 S5

Definition 2.3.14 (B axiom)**.** *For a modality i, the B* axiom *is*

$$p \Rightarrow [i]\langle i \rangle p.$$

Definition 2.3.15 (**B**). ***B*** *is the normal modal logic generated by the B axiom.*

Definition 2.3.16 (Symmetry). *A relation $R \subseteq W \times W$ is* symmetric *if for all $w_1, w_2 \in W$, if $w_1 R w_2$ then $w_2 R w_1$.*

Theorem 2.3.17. *Let $\mathbf{M}^s$ be the class of all relational structures with a single symmetric relation. **B** is sound and strongly complete for $\mathbf{M}^s$.*

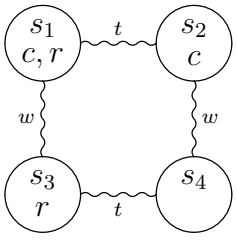
Definition 2.3.18 (**S5**). ***S5*** *is the normal modal logic generated by the set consisting of the T, 4, and B axioms.*

Definition 2.3.19 (Equivalence relation). *A relation R is an* equivalence relation *if and only if R is reflexive, transitive, and symmetric.*

Theorem 2.3.20. *Let $\mathbf{M}^{rst}$ be the class of all relational structures with a single relation which is an equivalence relation. **S5** is sound and strongly complete for $\mathbf{M}^{rst}$.*

S5 is often used to represent *epistemic logic*. In this case, there is a set of agents often labelled $i, j, ...$ and the box modality is written as K, where the formula $\mathsf{K}_i\phi$ means agent i *knows* ϕ. In this case, there is a relation for each agent, and it may be referred to as the agent's *indistinguishability relation*. The relation for agent i is usually denoted $\sim_i$, and the idea is that if $s_1 \sim_i s_2$, then agent i cannot distinguish state s_1 from state s_2. So if s_1 is the state of the system, agent i believes that s_2 may in fact be the actual state. We demonstrate these ideas with the following simple example.

Example 2.3.21. *Consider a system where it may be cold or not cold, and raining or not raining. There are two agents, t and w. t of them can only see a thermometer, and w can only look out the window. Our propositions will be c for cold and r for raining. We name our states s_1 through s_4.*



In state s_1, agent w thinks either state s_1 or s_3 is possible. This is because he can see out the window that it is raining, so he knows states s_2 and s_4 are impossible, but he cannot tell whether it is cold or not. Mirroring this intuition, $s_1 \models \mathsf{K}_w r$. On the other hand, $s_1 \models \neg\mathsf{K}_w c$. Similarly, $s_3 \models \mathsf{K}_t \neg c$, and $s_2 \models (\mathsf{K}_t c) \wedge (\neg\mathsf{K}_w c)$.

Part I

Epistemic Logic as a Programming Language: Epistemic Modalities in Process Calculi

Introduction

The goal of the present part of the thesis is simple: to put epistemic concepts in the hands of programmers rather than just having them appear in post-hoc theoretical analyses. One could imagine the incorporation of these ideas in a variety of process algebraic settings, but what is particularly appealing about the *concurrent constraint programming (CCP)* paradigm [Sar89, SJPR91] is that it was designed to give programmers explicit access to the concept of *partial information* and, as such, had close ties with logic [PSSS93, MPSS95]. This makes it ideal for the incorporation of epistemic concepts by expanding the logical connections to include modal logic [Kri63]. In particular, agents posting and querying information in the presence of *spatial hierarchies for sharing information and knowledge*, such as friend circles and shared albums in social networks or shared folders in cloud storage, provide natural examples of managing information access. These domains raise important problems such as the design of models to predict and prevent privacy breaches, which are now commonplace.

Contributions

In CCP [Sar89, SJPR91] processes interact with each other by querying and posting information to a single centralized shared store. The information and its associated partial order are specified as a *constraint system*, which can be seen as a *Scott information system* without consistency structure [AJ94b]. The centralized notion of store, however, makes CCP unsuitable for systems where information and processes can be shared or spatially distributed among certain groups of agents. In this thesis we enhance and

generalize the theory of CCP for systems with spatial distribution of information.

In Chapter 4 we generalize the underlying theory of constraint systems by adding *space functions* to their structure. These functions can be seen as topological closure operators and they provide for the specification of spatial and epistemic information. In Chapter 5 we extend CCP with a *spatial* and *epistemic* operator. The spatial operator can specify a process, or a local store of information, that resides within the *space* of a given agent (for example an application in some user's account, or some private data shared with a specific group). This operator can also be interpreted as an *epistemic* construction to specify that the information computed by a process will be known to a given agent. In many process calculi, it is traditional to refer to processes as agents. It is crucial to understand that in our setting processes and agents are completely different things. The processes are programs, they are mindless and do not "know" anything; the agents are separate, primitive entities in our model that can be viewed as spatial locations (a passive view) or as active entities that control a locus of information and interact with the global system by launching processes. This distinction will become more clear in Chapter 5.

It is also worth noting that the CCP concept of local variables cannot faithfully model what we are calling local spaces, since in our spatial framework we can have inconsistent local stores without propagating their inconsistencies towards the global store.

In Chapter 6 we give a natural notion of observable behaviour for spatial and epistemic processes. Recursive processes are part of our framework, so the notion of observable may involve limits of the spatial information in fair, possibly infinite, computations. These limits may result in infinite or, more precisely, *non-compact* objects involving unbounded nestings of spaces, or epistemic specifications such as *common knowledge*. We then provide a finitary characterization of these observables which avoids complex concepts such as fairness and limits. We also provide a compositional denotational characterization of the observable behaviour. Finally, in Chapter 7 we present some preliminary work on the technical issue of giving *finite*

approximations of non-compact information.

Three

Preliminaries

In this chapter we present the preliminaries for this section: a brief overview of basic domain theory as well as the concurrent constraint programming formalism.

3.1 Domain theory

Here we will briefly review some definitions from basic domain theory. For a complete exposition, see for example [AJ94b].

Definition 3.1.1 (Partially ordered set)**.** *For a set S with a binary relation $\sqsubseteq$, we say that $(S, \sqsubseteq)$ is a* partially ordered set*, or a* poset*, if the following three properties hold for all $x, y, z \in S$:*

1. *Reflexivity: $x \sqsubseteq x$.*
2. *Transitivity: if $x \sqsubseteq y$ and $y \sqsubseteq z$ then $x \sqsubseteq z$.*
3. *Antisymmetry: if $x \sqsubseteq y$ and $y \sqsubseteq x$ then $x = y$.*

If S with relation $\sqsubseteq$ is a partially ordered set then $\sqsubseteq$ is called a partial order*.*

Definition 3.1.2 (Upper bound)**.** *If $(S, \sqsubseteq)$ is a poset and $A \subseteq S$, then u is an* upper bound *for A if for all $a \in A$, $a \sqsubseteq u$.*

Definition 3.1.3 (Least upper bound)**.** *If $(S, \sqsubseteq)$ is a poset and $A \subseteq S$, then u is the* supremum *or* least upper bound *of A if u is an upper bound for A and for all $u' \in S$, if u' is an upper bound for A then $u \sqsubseteq u'$. We denote the least upper bound of a set a A as $\bigsqcup A$, and we denote the least upper bound of the set $\{a_1, a_2\}$ as $a_1 \sqcup a_2$. We also refer to $a_1 \sqcup a_2$ as the* join *of a_1 and a_2.*

Proposition 3.1.4. *For $(S, \sqsubseteq)$ a poset and $A \subseteq S$, if A has a supremum then it is unique.*

Proof. Suppose u_1 and u_2 are suprema of A. Then by definition, u_1 and u_2 are both upper bounds for A, and so $u_1 \sqsubseteq u_2$, since u_1 is the *least* upper bound of A, and similarly $u_2 \sqsubseteq u_1$. Therefore by reflexivity, $u_1 = u_2$. ∎

Definition 3.1.5 (Directed set)**.** *Suppose $(S, \sqsubseteq)$ is a poset, and $D \subseteq S$. We say that D is* directed *if for all $a, b \in D$ there exists $c \in D$ such that $a \sqsubseteq c$ and $b \sqsubseteq c$.*

Definition 3.1.6 (Directed-complete partial order)**.** *Suppose $(S, \sqsubseteq)$ is a poset. $(S, \sqsubseteq)$ is a* directed-complete partial order*, or* dcpo*, if every directed subset of S has a least upper bound.*

Definition 3.1.7 (Compact element)**.** *Suppose $(S, \sqsubseteq)$ is a dcpo and $d \in S$. d is a* compact element *of S if whenever D is a directed subset of S and $d \sqsubseteq \bigsqcup D$ then there exists $d^* \in D$ such that $d \sqsubseteq d^*$.*

Proposition 3.1.8. *If $(S, \sqsubseteq)$ is a dcpo and C is a finite set of compact elements of S, then if $\bigsqcup C$ exists, it is a compact element of S as well.*

Proof. We only prove that the join of two compact elements is compact. The result for finite sets of compact elements follows by induction.

Let d_1 and d_2 be compact elements of S, and suppose $d_1 \sqcup d_2$ exists. Suppose there is a directed set $D \subseteq S$ such that $d_1 \sqcup d_2 \sqsubseteq \bigsqcup D$. It follows that $d_1 \sqsubseteq \bigsqcup D$ and $d_2 \sqsubseteq \bigsqcup D$. Since d_1 and d_2 are compact by assumption, there must be $d_1^*, d_2^* \in D$ such that $d_1 \sqsubseteq d_1^*$ and $d_2 \sqsubseteq d_2^*$. And since D is directed, there must be $d^* \in D$ such that $d_1^* \sqsubseteq d^*$ and $d_2^* \sqsubseteq d^*$. But this

means that $d_1 \sqsubseteq d^*$ and $d_2 \sqsubseteq d^*$, and therefore $d_1 \sqcup d_2 \sqsubseteq d^*$. So we have shown that if d_1 and d_2 are compact, then for an arbitrary directed set D, if $d_1 \sqcup d_2 \sqsubseteq \bigsqcup D$, then there exists $d^* \in D$ such that $d_1 \sqcup d_2 \sqsubseteq d^*$. This proves that $d_1 \sqcup d_2$ is compact.

∎

Definition 3.1.9 (Algebraic)**.** *Suppose $(S, \sqsubseteq)$ is a dcpo. Let $K(S)$ denote the compact elements of S. $(S, \sqsubseteq)$ is* algebraic *if for every $a \in S$,*

$$a = \bigsqcup \{d \mid d \sqsubseteq a \text{ and } d \in K(S)\}.$$

Definition 3.1.10 (Complete lattice)**.** *$(S, \sqsubseteq)$ is a* complete lattice *if $(S, \sqsubseteq)$ is a poset and for every subset A of S, A has a least upper bound.*

Note that a complete lattice is always a dcpo.

Proposition 3.1.11. *Every complete lattice has a unique greatest element and a unique least element.*

Proof. Suppose $(S, \sqsubseteq)$ is a complete lattice. Since $\emptyset \subseteq S$, there must exist $\bigsqcup \emptyset$, which we will call *bot*. Now, for an arbitrary element $s \in S$, s is an upper bound for $\emptyset$ since s is above any element of $\emptyset$. So, because *bot* is the least upper bound of $\emptyset$, $bot \sqsubseteq s$ by definition of least upper bound. So *bot* is a least element of S, and from reflexivity it follows that *bot* is the unique least element of S.

Similarly, $S \subseteq S$, so S must have a least upper bound which we will call *top*. Since *top* is an upper bound for S, for any $s \in S$, $s \sqsubseteq top$. And since $top \in S$, it follows from reflexivity that *top* is the unique greatest element of S. ∎

Now we will discuss some kinds of functions on complete lattices which will be important later.

Definition 3.1.12 (Monotone function)**.** *Let $(S, \sqsubseteq)$ be a complete lattice and consider a function $f : S \longrightarrow S$. f is* monotone *if for all $a, b \in S$, if $a \sqsubseteq b$ then $f(a) \sqsubseteq f(b)$.*

Definition 3.1.13 (Fixed point)**.** *Let $(S, \sqsubseteq)$ be a complete lattice and consider a function $f : S \to S$. $a \in S$ is a* fixed point *for f if $f(a) = a$.*

Theorem 3.1.14 (Knaster-Tarski Theorem)**.** *If $(S, \sqsubseteq)$ is a complete lattice and f is a monotone function, then the set of fixed points of f is also a complete lattice.*

The proof can be found in [Tar55] or in [GKK+03].

Definition 3.1.15 (Closure operator)**.** *Let $(S, \sqsubseteq)$ be a complete lattice and consider a function $f : S \to S$. f is a* closure operator *on S if the following three conditions hold:*

- *f is* extensive*: for all $a \in S$, $a \sqsubseteq f(a)$*
- *f is* monotone*: for all $a, b \in S$, if $a \sqsubseteq b$ then $f(a) \sqsubseteq f(b)$*
- *f is* idempotent*: for all $a \in S$, $f(a) = f(f(a))$.*

For the next theorem, we need to define the greatest lower bound function on a complete lattice.

Definition 3.1.16. *First, note that on a complete lattice $(S, \sqsubseteq)$, we can define a* greatest lower bound *operator $\bigsqcap$ for $X \subseteq S$ as follows:*

$$\bigsqcap X = \bigsqcup \{s \in S \mid \forall x \in X, s \sqsubseteq x\}.$$

It is easy to see that this is the correct definition of the greatest lower bound of a set, and that it is defined for all sets in a complete lattice.

Theorem 3.1.17. *Let $(S, \sqsubseteq)$ be a complete lattice with a function $f : S \to S$. If f is a closure operator, then f is determined by its set of fixed points. If we let $C \subseteq S$ be the set of fixed points of f. We claim that for any $s \in S$,*

$$f(s) = \bigsqcap \{c \in C \mid s \sqsubseteq c\}.$$

Proof. To show that this definition of f is correct, first note that if $c \in C$ and $s \sqsubseteq c$ then by monotonicity $f(s) \sqsubseteq f(c)$ but since $c \in C$, $c = f(c)$. So for all $c \in C$, $f(s) \sqsubseteq c$, and therefore

$$f(s) \sqsubseteq \bigsqcap \{c \in C \mid s \sqsubseteq c\}.$$

On the other hand, by idempotence, $f(s) = f(f(s))$, so $f(s) \in C$, and by extensiveness, $s \sqsubseteq f(s)$, so $f(s) \in \{c \in C \mid s \sqsubseteq c\}$ and therefore

$$\bigsqcap\{c \in C \mid s \sqsubseteq c\} \sqsubseteq f(s).$$

This two inequalities prove that $f(s) = \bigsqcap\{c \in C \mid s \sqsubseteq c\}$, which means that f is determined by its set of fixed points. ∎

3.2 Concurrent constraint programming

Before presenting our models of spatial and epistemic CCP, we briefly present traditional CCP. The first complete presentation of concurrent constraint programming was [Sar93]. CCP is a model for concurrency based on logic and partial information. Processes communicate asynchronously through shared variables in a store, which contains *constraints*, assertions consisting of partial information about these variables. For example, the constraint $X + Y < 10$ gives some information about the variables X and Y without assigning them a specific value. The processes in CCP communicate with one another only by asking and telling information from the store: the tell operator allows a process to add information to the store, and the ask operator allows a process to query the store and take actions based on the results of the query.

3.2.1 Constraint systems

Formally, the CCP model is parametric in a *constraint system* specifying the structure and interdependencies of the information that processes can add to and ask about from the central shared store.

Definition 3.2.1 (Constraint system)**.** $\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false)$ *is a* constraint system *(cs) if* $(Con, \sqsubseteq)$ *is a complete algebraic lattice,* Con_0 *is the set of compact elements of* $(Con, \sqsubseteq)$*,* $\sqcup$ *is the least upper bound operation,* $true$ *is the least element of* $(Con, \sqsubseteq)$*, and* $false$ *is the greatest element of* $(Con, \sqsubseteq)$*.*

The elements of Con are called constraints and they represent partial information. $\sqsubseteq$ is the *information ordering* or *reverse entailment* relation, because $c \sqsubseteq d$ means that d entails c, or d contains more information than c. The top element *false* represents inconsistent information, and the bottom element *true* is the empty constraint. The least upper bound, $\sqcup$ of a set represents the combination of all the information in that set.

Example 3.2.2. *We briefly present an example based on the* Herbrand constraint system *from [Sar89, SJPR91]. This constraint system is built from a first-order alphabet $\mathcal{L}$ with countably many variables $x, y, \ldots$ and equality $=$. The constraints are equivalence classes of equalities over terms, quotiented by logical equivalence. For example, $\{x = t, y = t\}$ is a constraint, and we consider it to be the same constraint as $\{x = t, y = t, x = y\}$. The relation $c \sqsubseteq d$ holds if the equalities in c follow logically from those in d, for example, $\{x = y\} \sqsubseteq \{x = t, y = t\}$. The constraint false is the equivalence class of inconsistent sets of equalities, and and true is the equivalence class of the empty set. The compact elements are the equivalence classes of finite sets of equalities. The least upper bound is the equivalence class of the set union (see [Sar89, SJPR91] for full details). Figure 3.1 is an instance of this constraint system with two variables, x and y, and two constants, a and b, with $a \neq b$.*

3.2.2 Processes

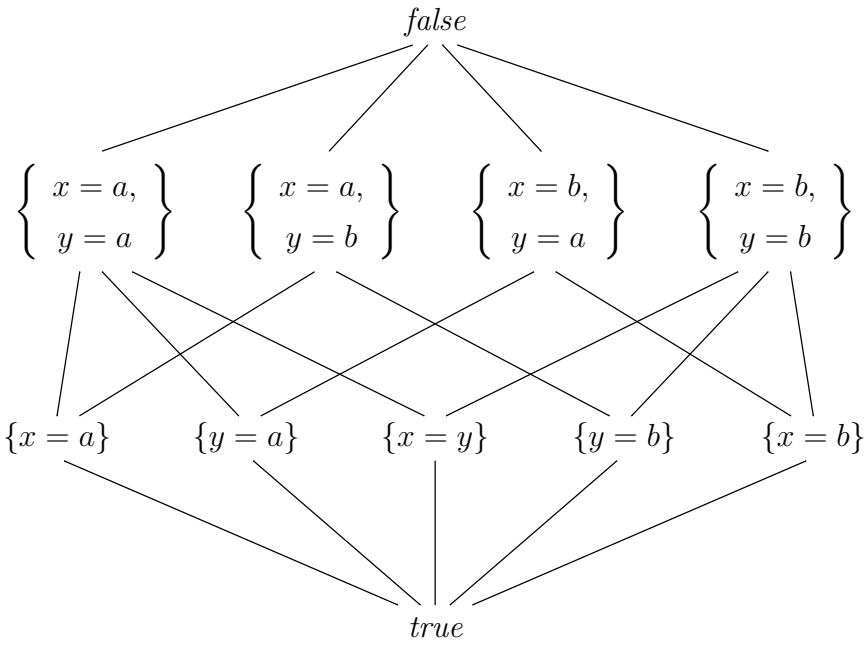
Now we briefly present the process calculus CCP. We will discuss the syntax and semantics of CCP processes.

Definition 3.2.3 (CCP process)**.** *Assume a constraint system* $\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false)$*, and a countable set of variables* $Vars = \{X, Y, ...\}$*. The terms are given by the following syntax:*

$$P \ ::== \ 0 \ \mid \ \textbf{\textit{tell}}(c) \ \mid \ \textbf{\textit{ask}}\ (c) \ \rightarrow \ P \ \mid \ P \parallel Q \ \mid \ X \ \mid \ \mu X.P$$

where $c \in Con_0$ and $X \in Vars$.

Figure 3.1: A Herbrand constraint system



A term T is said to be closed if every variable X in T occurs in the scope of an expression $\mu X.P$. We refer to closed terms as processes *and use $Proc$ to denote the class of all processes.*

In most presentations of CCP, for example [Sar89], there is also a hiding operator, denoted $\exists_x P$. To simplify our presentation, we omit this operator, because we believe is it orthogonal to the extensions of CCP that we are going to present.

Before giving the semantics of processes, we give some intuitions about their behaviour. First, processes execute in the context of a *store*, which contains certain information, and to which the processes may sometimes add information. The basic processes are tell, ask, and parallel composition. Intuitively, $\mathbf{tell}(c)$ in a store d adds c to d to make c available to other processes with access to this store. This addition, represented as $d \sqcup c$, is performed whether or not $d \sqcup c = \mathit{false}$. The process $\mathbf{ask}(c) \rightarrow P$ in a store e may execute P if c is entailed by e, i.e., $c \sqsubseteq e$. The process $P \parallel Q$ stands for the *parallel execution* of P and Q. Also, given $I = \{i_1, \ldots, i_m\}$ we use

$\prod_{i \in I} P_i$ as a shorthand for $P_{i_1} \parallel \ldots \parallel P_{i_m}$. Finally, $\mu X.P$ is a recursive process, in which X is a variable itself representing a process.

To give an idea of the behaviour of a CCP process, we present a simple example. Variations of the following example will be referred to throughout the paper.

Example 3.2.4. *Let us take $P = \textbf{\textit{tell}}(c)$ and $Q = \textbf{\textit{ask}}(c) \rightarrow \textbf{\textit{tell}}(d)$. In the execution of the process $P \parallel Q$, first Q will be blocked and P will execute, adding c to the store, and then Q will be able to act and d will be added to the store. So the end result will be $c \sqcup d$ in the store.*

We now define a structural operational semantics (sos) for CCP. But the behaviour of a process depends on the information in the store, so the operational semantics must be described in terms of processes and stores, as defined below.

Definition 3.2.5 (Configuration)**.** *Let $\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false)$ be a constraint system, P a process over $\mathbf{C}$, and $c \in Con_0$. The pair $\langle P, c \rangle$ is a* configuration*. We denote the set of all configurations over $\mathbf{C}$ as $Conf(\mathbf{C})$, or just $Conf$ when $\mathbf{C}$ is evident or irrelevant.*

We often use $\gamma, \gamma', \gamma_1, \gamma_2, \ldots$ to represent configurations.

Definition 3.2.6 (Operational semantics of CCP)**.** *For a constraint system $\mathbf{C}$, the CCP transition relation $\longrightarrow \subseteq Conf(\mathbf{C}) \times Conf(\mathbf{C})$ is defined in Table 3.2.*

$$\mathbf{T}\ \frac{}{\langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{0}, d \sqcup c\rangle}$$

$$\mathbf{A}\ \frac{c \sqsubseteq d}{\langle \mathbf{ask}\ (c)\ \ \rightarrow\ \ P, d\rangle \longrightarrow \langle P, d\rangle}$$

$$\mathbf{PL}\ \frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle P \parallel Q, d\rangle \longrightarrow \langle P' \parallel Q, d'\rangle}$$

$$\mathbf{PR}\ \frac{\langle Q, d\rangle \longrightarrow \langle Q', d'\rangle}{\langle P \parallel Q, d\rangle \longrightarrow \langle P \parallel Q', d'\rangle}$$

$$\mathbf{R}\ \frac{\langle P[\mu X.P/X], d\rangle \longrightarrow \gamma}{\langle \mu X.P, d\rangle \longrightarrow \gamma}$$

Figure 3.2: Structural operational semantics for CCP.

Four

Space and Knowledge in Constraint Systems

In this chapter we introduce two new notions of constraint systems for reasoning about distributed information and knowledge in CCP.

4.1 Spatial Constraint Systems.

A crucial issue in distributed and multi-agent scenarios is that agents may have their own spaces for their local information and for performing their computations. We address this issue by introducing a notion of space for agents directly into our constraint systems. In our approach each agent i has a *space* $\mathfrak{s}_i$. We consider $\mathfrak{s}_i(c)$ to be an assertion stating that c holds *within a space attributed to agent* i. Thus, given a store $s = \mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) \sqcup e$ we may think of c and d as holding within the spaces that agents i and j have in s, respectively. Similarly, $\mathfrak{s}_i(\mathfrak{s}_j(c))$ can be viewed as a hierarchical spatial specification stating that c holds within the space that the agent i attributes to agent j.

An n-agent *spatial constraint system (n-scs)* is a constraint system parametric in n structure-preserving constraint mappings $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$ capturing the above intuitions.

Definition 4.1.1 (scs)**.** *An n-agent* spatial constraint system (n-scs) $\boldsymbol{C}$ *is a constraint system equipped with n lub and bottom preserving maps $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$ over its set of constraints Con. More precisely,*

$$\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false, \mathfrak{s}_1, \ldots, \mathfrak{s}_n)$$

where $(Con, \sqsubseteq)$ is a complete algebraic lattice with compact elements Con_0, least upper bound operation $\sqcup$, bottom element true, and top element false, and furthermore, each $\mathfrak{s}_i : Con \to Con$ satisfies the following properties:

S.1 $\mathfrak{s}_i(true) = true$

S.2 $\mathfrak{s}_i(c \sqcup d) = \mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)$

From now on, for an n-scs $\mathbf{C}$, we refer to each $\mathfrak{s}_i$ as the *space* (function) of agent i in $\mathbf{C}$. Also, we write only "scs" when n is unimportant.

Intuitively, S.1 states that the minimal piece of information, *true*, holds in every agent's space. S.2 says that agents can join together the pieces of information in their spaces. From S.2 it follows immediately that the space functions are monotone: Property S.3 below says that if c can be derived from d then any agent can derive c from d within its own space.

Corollary 4.1.2. *Let $\boldsymbol{C}$ be an n-scs with space functions $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$. Then for each $\mathfrak{s}_i$ the following property holds:*

S.3 *If $c \sqsubseteq d$ then $\mathfrak{s}_i(c) \sqsubseteq \mathfrak{s}_i(d)$.*

Proof. If $c \sqsubseteq d$ then $d = c \sqcup d$ so $\mathfrak{s}_i(d) = \mathfrak{s}_i(c \sqcup d) = \mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)$. But if $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d) = \mathfrak{s}_i(d)$ then $\mathfrak{s}_i(c) \sqsubseteq \mathfrak{s}_i(d)$. ∎

We should discuss here some differences between our notion of information holding in an agent's space and the earlier notion of a variable which is local to an agent in CCP. It is true that traditional (non-spatial) CCP agents may have local information within the global store by using local variables. Formally, this is achieved by using the elegant notion of a *cylindric constraint system* [SRP91]. A cylindric constraint system is a constraint system with variable hiding operations of the form $\exists_X(.) : Con \to Con$ that

act much like existential quantifiers. The constraint $\exists_X(c)$ represents the constraint c where the variable X is hidden. Intuitively, the information in c about X is only available to the agent that locally declared X.

Nevertheless, in CCP local variables cannot be used to model the fact that in some *distributed systems* agents may produce inconsistent information within their own spaces, for example as the result of a failed computation, without rendering the global store inconsistent. This is because any cylindric constraint system requires $\exists_X(\mathit{false}) = \mathit{false}$. This suggests that the use of local variables does not fully provide for local computational spaces since local inconsistencies cannot be confined to an agent's space. A closely related issue is that a given agent may compute information about a *global object*, that is inconsistent with that of other agents. For example, given a global system variable X an agent may compute in its own space that $X = 42$ while another agent may compute that $X > 42$. In previous CCP approaches this could be modelled at best indirectly, for example by treating "agent 1 believes that $X = 42$" as a proposition.

4.1.1 Inconsistency Confinement.

In a spatial constraint system nothing prevents us from having $\mathfrak{s}_i(\mathit{false}) \neq \mathit{false}$. Intuitively, inconsistencies generated by an agent may be confined within the agent's own space. It is also possible to have $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) \neq \mathit{false}$ even when $c \sqcup d = \mathit{false}$; i.e. we may have agents whose information is inconsistent with the information of other agents. This reflects the distributive nature of the agents as they may have different information about the same incident. The following definitions capture these situations.

Definition 4.1.3. *[Space Consistency] An n-scs*

$$\boldsymbol{C} = (\mathit{Con}, \mathit{Con}_0, \sqsubseteq, \sqcup, \mathit{true}, \mathit{false}, \mathfrak{s}_1, \ldots, \mathfrak{s}_n)$$

is said to be (i, j) space consistent with respect to $(c, d) \in \mathit{Con} \times \mathit{Con}$ *if and only if* $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) \neq \mathit{false}$. *Also,* $\boldsymbol{C}$ *is said to be* (i, j) space consistent *if and only if it is* (i, j) *space consistent with respect to each* $(c, d) \in \mathit{Con} \times \mathit{Con}$.

Furthermore, $\boldsymbol{C}$ *is space consistent if and only if it is* (i, j) *space consistent for all* $i, j \in \{1, \ldots, n\}$.

We will see an important class of logical structures characterized as space consistent spatial constraint systems in section 4.3. From the next proposition we conclude that to check (i, j) space-consistency it is sufficient to check that $\mathfrak{s}_i(\mathit{false}) \sqcup \mathfrak{s}_j(\mathit{false}) \neq \mathit{false}$.

Proposition 4.1.4. *Let* $\boldsymbol{C}$ *be an n-scs with space functions* $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$. *Then*

1. $\boldsymbol{C}$ *is* (i, j) *space consistent if and only if* $\mathfrak{s}_i(\mathit{false}) \sqcup \mathfrak{s}_j(\mathit{false}) \neq \mathit{false}$ *and*

2. *If* $\boldsymbol{C}$ *is* (i, j) *space consistent then* $\mathfrak{s}_i(\mathit{false}) \neq \mathit{false}$.

Proof. 1. $\mathbf{C}$ is (i, j) space consistent if and only if $\mathfrak{s}_i(\mathit{false}) \sqcup \mathfrak{s}_j(\mathit{false}) \neq \mathit{false}$

First, if $\mathbf{C}$ is (i, j) space consistent then by definition $\mathfrak{s}_i(\mathit{false}) \sqcup \mathfrak{s}_j(\mathit{false}) \neq \mathit{false}$.

To prove the other direction, suppose that $\mathfrak{s}_i(\mathit{false}) \sqcup \mathfrak{s}_j(\mathit{false}) \neq \mathit{false}$. Then for any $c, d \in \mathit{Con}$, $c \sqsubseteq \mathit{false}$ and $d \sqsubseteq \mathit{false}$, so by property S.3, $\mathfrak{s}_i(c) \sqsubseteq \mathfrak{s}_i(\mathit{false})$ and $\mathfrak{s}_j(d) \sqsubseteq \mathfrak{s}_j(\mathit{false})$, so

$$\mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) \sqsubseteq \mathfrak{s}_i(\mathit{false}) \sqcup \mathfrak{s}_j(\mathit{false}) \neq \mathit{false},$$

and since *false* is the top element of the lattice it follows that

$$\mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) \neq \mathit{false}.$$

This means that $\mathbf{C}$ is (i, j) space consistent.

2. If $\mathbf{C}$ is (i, j) space consistent then $\mathfrak{s}_i(\mathit{false}) \neq \mathit{false}$.

Proof by contradiction: if $\mathfrak{s}_i(\mathit{false}) = \mathit{false}$, then $\mathfrak{s}_i(\mathit{false}) \sqcup \mathfrak{s}_j(d) = \mathit{false}$ for any $d \in \mathit{Con}$, which would contradict (i, j) space consistency. Therefore, $\mathfrak{s}_i(\mathit{false}) \neq \mathit{false}$.

∎

Distinctness preservation.

Analogous to inconsistency confinement, a spatial constraint system may have $\mathfrak{s}_i(c) = \mathfrak{s}_i(d)$ for $c \neq d$. Depending on the intended model, this could be interpreted as meaning that agent i cannot distinguish c from d. For example, in a model r and g may represent that a certain object is $r$ed or $g$reen. In this situation, it makes sense to have $\mathfrak{s}_i(r) = \mathfrak{s}_i(g)$ for a colour-blind agent i. For some applications, however, it may be necessary for the space functions to preserve distinctness:

Definition 4.1.5. *[Distinctness preservation] An n-scs* $\boldsymbol{C}$ preserves distinctness *if and only if for all* $i \in \{1, ..., n\}$ *and for all* $c, d \in Con$, $\mathfrak{s}_i(c) = \mathfrak{s}_i(d)$ *if and only if* $c = d$. *In other words, all the space functions are injective.*

Shared and Global Information.

We conclude by introducing constructions capturing the intuition that a given constraint holds in a shared space for a certain group of agents, and that a constraint holds globally, or globally for a certain group.

Definition 4.1.6 (Group Space)**.** *Let* $\boldsymbol{C}$ *be an n-scs. For a set of agents* $G \subseteq \{1, ..., n\}$, *a* group-space *for* G $\mathfrak{s}_G(\cdot)$ *is defined as*

$$\mathfrak{s}_G(c) = \bigsqcup_{i \in G} \mathfrak{s}_i(c)$$

Definition 4.1.7 (Global Information)**.** *Let* $\boldsymbol{C}$ *be an n-scs. For a set of agents* $G \subseteq \{1, ..., n\}$, *we define* global information for G, $\mathfrak{g}_G(\cdot)$ *as*

$$\mathfrak{g}_G(c) = \bigsqcup_{j=0}^{\infty} \mathfrak{s}_G^j(c),$$

where $\mathfrak{s}_G^0(c) = c$ *and* $\mathfrak{s}_G^{k+1}(c) = \mathfrak{s}_G(\mathfrak{s}_G^k(c))$. *If* $\mathfrak{g}_G(c)$, *then we say that information c holds* globally for group G

Recall that a spatial constraint system is a complete lattice, so the global information operator is well defined. It is easy to see that the constraint

$\mathfrak{g}_G(c)$ entails c and $\mathfrak{s}_{i_1}(\mathfrak{s}_{i_2}(\ldots(\mathfrak{s}_{i_m}(c))\ldots))$ for any sequence of $i_1, \ldots, i_m \in G$. Thus this operator realizes the intuition that c holds *globally* with respect to G: c holds in each nested space involving only the agents in G.

4.2 Epistemic Constraint Systems.

We now wish to use $\mathfrak{s}_i(c)$ to represent not only information that holds in an agent's space, but rather a *fact* that an agent *knows*. Representing knowledge necessitates additional properties of the space functions in the constraint system. In fact, the domain theoretical nature of constraint systems allows for a simple and elegant characterization of knowledge by requiring our space functions to be *Kuratowski closure operators* [MT44]: closure operators that preserve least upper bounds and bottom (*true*).

Definition 4.2.1 (n-ecs)**.** *An n-agent* epistemic constraint system (n-ecs) ***C*** *is an n-scs whose space functions $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$ are also* closure operators*. Thus, the space functions must satisfy all of the following properties:*

S.1 $\mathfrak{s}_i(\mathit{true}) = \mathit{true}$

S.2 $\mathfrak{s}_i(c \sqcup d) = \mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)$

E.1 $c \sqsubseteq \mathfrak{s}_i(c)$

E.2 $\mathfrak{s}_i(\mathfrak{s}_i(c)) = \mathfrak{s}_i(c)$

Intuitively, in an n-ecs, $\mathfrak{s}_i(c)$ states that the agent i knows information c. The axiom E.1 says that knowledge is accurate: if agent i knows c then c must be true, hence $\mathfrak{s}_i(c)$ entails c. The epistemic principle that an agent i is aware of its own knowledge (the agents know what they know) is realized by E.2. The epistemic assumption that agents are idealized reasoners follows from S.3 in Corollary 4.1.2; if d entails c ($c \sqsubseteq d$) then if d is known to agent i, so is c ($\mathfrak{s}_i(c) \sqsubseteq \mathfrak{s}_i(d)$).

Common Knowledge.

Epistemic constructions such as "agent i knows that agent j knows c" can be expressed as $\mathfrak{s}_i(\mathfrak{s}_j(c))$. *Group knowledge* of a fact c for a group of agents G means that all the agents in G know c. This can be represented as $\mathfrak{s}_G(c)$ as in Definition 4.1.6. Similarly, *common knowlege* of a fact c in a group G happens when all the agents in G know c, they all know that they know c, and so on ad infinitum. This is captured by the construction $\mathfrak{g}_G(c)$ in Definition 4.1.6.

Remark 4.2.2. *Consider an n-ecs* $\boldsymbol{C}$ *whose compact elements* Con_0 *are closed under the space functions: i.e., if* $c \in Con_0$ *the* $\mathfrak{s}_i(c) \in Con_0$. *By proposition 3.1.8,* Con_0 *is closed under group knowledge* $\mathfrak{s}_G(c)$ *since* G *is finite. It is not necessarily closed under common knowledge* $\mathfrak{g}_G(c)$, *however, because, in general,* $\bigsqcup_{j=1}^{\infty} \mathfrak{s}_G^j(c)$ *cannot be finitely approximated. Nevertheless, in Examples (Section 4.3) we shall identify families of scs's where* Con_0 *is closed under common knowledge, and in Section 7 we address the issue of using suitable over-approximations of common knowledge.*

The following proposition states two distinctive properties of epistemic constraint systems: They are not space consistent and those with space functions other than identity do not preserve distinctness. We use *id* to denote the identity space function.

Proposition 4.2.3. *Let* $\boldsymbol{C}$ *be an n-ecs with space functions* $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$. *For each* $i, j \in \{1, \ldots, n\}$:

1. $\mathbf{C}$ *is not* (i, j)*-space consistent*
2. *If there is* $i \in \{1, ..., n\}$ *such that* $\mathfrak{s}_i \neq id$ *then* $\mathbf{C}$ *does not preserve distinctness.*

Proof. 1. Since $\mathbf{C}$ is an ecs, $false \sqsubseteq \mathfrak{s}_i(false)$ by **E.1**, and $false$ is the top element so $\mathfrak{s}_i(false) = false$. Recall from Proposition 4.1.4 that if $\mathbf{C}$ is (i, j) space consistent then $\mathfrak{s}_i(false) \neq false$. So $\mathbf{C}$ is not (i, j) space consistent and therefore not space consistent at all.

2. Recall from Definition 4.1.5 that **C** preserves distinctness if all the space functions are injective. Suppose there is some $i \in \{1, ..., n\}$ so that $\mathfrak{s}_i \neq id$, so there is some $c \in Con$ such that $c \neq \mathfrak{s}_i(c)$. But by **E.2**, $\mathfrak{s}_i(c) = \mathfrak{s}_i(\mathfrak{s}_i(c))$. So, even though $c \neq \mathfrak{s}_i(c)$, $\mathfrak{s}_i(c) = \mathfrak{s}_i(\mathfrak{s}_i(c))$, meaning that $\mathfrak{s}_i$ is not injective. Therefore, **C** is distinctness preserving if and only if all the space functions are the identity.

∎

4.3 Examples.

We now illustrate two important families of spatial constraint systems. The families reveal meaningful connections between our spatial constraint systems and models of knowledge and belief [FHMV95].

Aumann Constraint Systems

Aumann structures [FHMV95] are an *event-based* approach to modelling knowledge. An Aumann structure is a tuple $\mathbf{A} = (S, P_1, ..., P_n)$ where S is a set of states and each P_i is a partition on S for agent i: for each agent i, $P_i = \{S_{i_1}, S_{i_2}, ..., S_{i_{k_i}}\}$, where $S = \bigcup_{j=i_1}^{i_{k_i}} S_j$ and for all i_j, i_k, $S_{i_j} \cap S_{i_k} = \emptyset$. We call these partitions *information sets*. If two states t and u are in the same information set for agent i, it means that in state t agent i considers state u possible, and vice versa. An *event* in an Aumann structure is any subset of S. For example, an event could be "I am wearing a red shirt," which would consist of all the states where I am wearing a red shirt. Event e holds at state t if $t \in e$. The conjunction of two events is their intersection and knowledge of an event is itself an event: for $i \in \{1, ..., n\}$, the knowledge operator $\mathsf{K}_i : \mathcal{P}(S) \longrightarrow \mathcal{P}(S)$ is defined as

$$\mathsf{K}_i(e) = \{t \in S \mid P_i(t) \subseteq e\}$$

where $P_i(t)$ denotes the cell that t is in in the partition P_i.

Thus, we can define group knowledge for group G as

$$\mathsf{E}_G(e) = \bigcap_{i \in G} \mathsf{K}_i(e).$$

And we can define common knowledge of event e for group G as

$$\mathsf{C}_G(e) = \bigcap_{j=1}^{\infty} \mathsf{E}_G^j(e).$$

Definition 4.3.1 (Aumann Constraint System)**.** *We define the* Aumann n-ecs $\mathbf{C}(\mathbf{A})$ *as follows: The constraints are the events:* $Con = \{e \mid e \subseteq \mathbf{A}\}$*, the order is the reverse inclusion order:* $e_1 \sqsubseteq e_2$ *iff* $e_2 \subseteq e_1$*, the least upper bound of a set of events is just the the set intersection of the events, true is the entire set,* S*, and false is the empty event,* $\emptyset$*. The space function for each agent* i *is given by* $\mathfrak{s}_i(e) = \mathsf{K}_i(e)$*.*

Theorem 4.3.2. *For any Aumann structure* $\mathbf{A} = (S, P_1, ..., P_n)$*,* $\boldsymbol{C}(\mathbf{A})$ *is an* n*-ecs.*

Proof. This proof has several parts: we must prove that $\mathbf{C}(\mathbf{A})$ is a complete algebraic lattice, and we must prove that the $\mathfrak{s}_i$ functions satisfy the four properties of an ecs.

Complete algebraic lattice : It is a standard result that the reverse inclusion ordering on the powerset of a set is a complete algebraic lattice; the compact elements are the co-finite subsets of S (the sets $t \subseteq S$ such that $S \backslash t$ is a finite set), and it is easy to see that every set is equal to the intersection of the cofinite sets that it is included in, so the lattice is algebraic, and it is also easy to see that the intersection of sets is their least upper bound, and that is the top element and S is the bottom element.

S.1 : Now we must show that $\mathfrak{s}_i(\mathit{true}) = \mathit{true}$, which is equivalent to $\mathsf{K}_i(S) = S$. Recall that $\mathsf{K}_i(e) = \{t \in S \mid P_i(t) \subseteq e\}$, so $\mathsf{K}_i(S) = \{t \in S \mid P_i(t) \subseteq S\}$, which is of course equal to S.

S.2 : We want to show that $\mathfrak{s}_i(c \sqcup d) = \mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)$.

$$\begin{aligned}
\mathfrak{s}_i(c \sqcup d) &= \mathsf{K}_i(c \cap d) \\
&= \{t \in S \mid P_i(t) \subseteq c \cap d\} \\
&= \{t \in S \mid P_i(t) \subseteq c \text{ and } P_i(t) \subseteq d\} \\
&= \{t \in S \mid P_i(t) \subseteq c\} \cap \{t \in S \mid P_i(t) \subseteq d\} \\
&= \mathsf{K}_i(c) \cap \mathsf{K}_i(d) \\
&= \mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)
\end{aligned}$$

E.1 : We want to show that $c \sqsubseteq \mathfrak{s}_i(c)$. To prove this, first note that for any state t, $t \in P_i(t)$ (because $P_i(t)$ denotes t's cell in the partition P_i). So, if $P_i(t) \subseteq c$ then $t \in c$. Therefore, $\mathsf{K}_i(c) = \{t \mid P_i(t) \subseteq c\} \subseteq c$. This is equivalent to $c \sqsubseteq \mathfrak{s}_i(c)$.

E.2 : We must show that $\mathfrak{s}_i(\mathfrak{s}_i(c)) = \mathfrak{s}_i(c)$. This is the same as showing that $\mathsf{K}_i(\mathsf{K}_i(c)) = \mathsf{K}_i(c)$, or $\{r \mid P(r) \subseteq \{t \mid P(t) \subseteq c\}\} = \{u \mid P(u) \subseteq c\}$, which is true.

∎

Aumann constraint systems are epistemic constraint systems, thus they are not space consistent (Proposition 4.2.3). We shall now identify a meaningful spatial constraint system that is space consistent.

Kripke Constraint Systems.

Recall that a Kripke structure, also called a relational structure, is a set of states and a family of relations on the states indexed by the agents. We denote the relations as $s \xrightarrow{i} t$ if s is related to t, and they can be thought of as accessibility relations for the agents: if $s \xrightarrow{i} t$ then in state s, agent i considers t possible. An epistemic Kripke structure is a Kripke structure where the accessibility relations are equivalence relations (reflexive, transitive, and symmetric). In the following spatial constraint system, the constraints are sets of *pointed Kripke structures*: sets of pairs (M, s)

where M is a Kripke structure and s is a state of M. In the definition below we consider multiple Kripke structures, so we index the relations by the structure they belong to: if s and t are states in M and s is related to t, we denote this as $s \xrightarrow{i}_M t$.

Definition 4.3.3 (Kripke Constraint System)**.** *Consider a set of Kripke structures $\mathfrak{M}$ over agents $\{1, ..., n\}$. Let $\Delta_{\mathfrak{M}}$ be the set $\{(M, t) \mid M \in \mathfrak{M}$ and $t \in St(M)\}$ where $St(M)$ denotes the set of states of M. Define an n-scs $\boldsymbol{C}(\Delta_{\mathfrak{M}})$ as follows:*

- *Let $Con = \mathcal{P}(\Delta_{\mathfrak{M}})$,*
- *Con_0 is the set of cofinite sets, that is, if $\Delta_{\mathfrak{M}} \backslash c$ is a finite set, then c is a compact element in the lattice,*
- *$c_1 \sqsubseteq c_2$ iff $c_2 \subseteq c_1$,*
- *$c_1 \sqcup c_2$ is the set intersection of c_1 and c_2,*
- *true is the set $\Delta_{\mathfrak{M}}$,*
- *false is $\emptyset$,*
- *and finally, define*

$$\mathfrak{s}_i(c) = \{(M, t) \in \Delta_{\mathfrak{M}} \mid \forall t' \in St(M) \left[t \xrightarrow{i}_M t' \Rightarrow (M, t') \in c \right]\}.$$

Notice that the definition of the space functions is reminiscent of the semantics of the box modality in modal logic [Pop94].

The following theorem gives us a taxonomy of spatial constraint systems for the above construction.

Theorem 4.3.4. *Let $\mathfrak{M}$ be a non-empty set of Kripke structures over agents $\{1, ..., n\}$,*

1. *$\boldsymbol{C}(\Delta_{\mathfrak{M}})$ is an n-scs for any $\mathfrak{M}$,*
2. *If $\mathfrak{M}$ is the class of all n agent pointed Kripke structures, then $\boldsymbol{C}(\Delta_{\mathfrak{M}})$ is a space consistent n-scs, and*

3. *If $\mathfrak{M}$ is a set of n-agent pointed Kripke structures whose accessibility relations are equivalences then $\boldsymbol{C}(\Delta_{\mathfrak{M}})$ is an n-ecs.*

Proof. There are three parts to the proof. Again, we omit the proof that the powerset lattice with the reverse subset ordering is a complete algebraic lattice, with the compact elements, least upper bound operation, and top and bottom elements as defined, because this is a standard and straightforward proof. We prove below that the space functions meet the required properties in each case. Recall that the space functions are defined as

$$\mathfrak{s}_i(c) = \{(M,t) \in \Delta_{\mathfrak{M}} \mid \forall t' \in St(M) \left[t \xrightarrow{i}_M t' \Rightarrow (M,t') \in c\right]\}.$$

1. **S.1** We must show that $\mathfrak{s}_i(\mathit{true}) = \mathit{true}$. Recall that $\mathit{true} = \Delta_{\mathfrak{M}}$. So $\mathfrak{s}_i(\Delta_{\mathfrak{M}}) = \{(M,t) \in \Delta_{\mathfrak{M}} \mid \forall t' \in St(M) \left[t \xrightarrow{i}_M t' \Rightarrow (M,t') \in \Delta_{\mathfrak{M}}\right]\}$. And since $\Delta_{\mathfrak{M}}$ is the entire set of pointed Kripke structures under consideration, every $(M,t') \in \Delta_{\mathfrak{M}}$, so $\mathfrak{s}_i(\Delta_{\mathfrak{M}}) = \Delta_{\mathfrak{M}}$ as desired.

   S.2 This part is straightforward:

$$\begin{aligned}
\mathfrak{s}_i(c \sqcup d) &= \{(M,t) \mid \forall t' \in St(M) \left[t \xrightarrow{i}_M t' \Rightarrow (M,t') \in c \sqcup d\right]\} \\
&= \{(M,t) \mid \forall t' \in St(M) \left[t \xrightarrow{i}_M t' \Rightarrow (M,t') \in c \cap d\right]\} \\
&= \{(M,t) \mid \forall t' \in St(M) \left[t \xrightarrow{i}_M t' \Rightarrow (M,t') \in c\right]\} \\
&\quad \cap \{(M,t) \mid \forall t' \in St(M) \left[t \xrightarrow{i}_M t' \Rightarrow (M,t') \in d\right]\} \\
&= \mathfrak{s}_i(c) \cap \mathfrak{s}_i(d) \\
&= \mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)
\end{aligned}$$

2. We must show that if $\mathfrak{M}$ is the class of all pointed Kripke structures, then $\mathbf{C}(\Delta_{\mathfrak{M}})$ is a space consistent n-scs. Recall that $\mathbf{C}$ is space consistent if and only if $\mathbf{C}$ is (i,j) space consistent for all $i,j \in \{1,...,n\}$, and that from Proposition 4.1.4, a spatial constraint system is (i,j) space consistent if and only if $\mathfrak{s}_i(\mathit{false}) \sqcup \mathfrak{s}_j(\mathit{false}) \neq \mathit{false}$. So we will show that for all $i,j \in \{1,...,n\}$, $\mathfrak{s}_i(\mathit{false}) \sqcup \mathfrak{s}_j(\mathit{false}) \neq \mathit{false}$.

   Since we are considering the class of *all* pointed Kripke structures over n agents, this includes the Kripke structure we will call M^*, which we

define as having a single state s^*, and every relation $\xrightarrow{i}$ empty. Now, notice that

$$\mathfrak{s}_i(false) = \{(M,t) \mid \forall t' \in St(M) \left[t \xrightarrow{i}_M t' \Rightarrow (M,t') \in \emptyset\right]\}$$

Since s^* is not related to any other states by the $\xrightarrow{i}_{M^*}$ relation, $(M^*, s^*) \in \mathfrak{s}_i(false)$, and similarly $(M^*, s^*) \in \mathfrak{s}_j(false)$. Thus, $\mathfrak{s}_i(false) \sqcup \mathfrak{s}_j(false)$ is nonempty, and so $\mathbf{C}(\Delta_{\mathfrak{M}})$ is space consistent.

3. Now we must show that if $\mathfrak{M}$ is a set of n-agent pointed Kripke structures whose accessibility relations are equivalence relations then $\mathbf{C}(\Delta_{\mathfrak{M}})$ is an n-ecs. It follows from the first item that $\mathbf{C}(\Delta_{\mathfrak{M}})$ is an scs, so we must prove that the space functions respect properties **E.1** and **E.2**. Since we are now discussing equivalence relations, instead of $s \xrightarrow{i}_M t$, we will write $s \sim^i_M t$.

   E.1 We must show that for all c, $c \sqsubseteq \mathfrak{s}_i(c)$, which is equivalent to $\mathfrak{s}_i(c) \subseteq c$. Now, if $(M,t) \in \mathfrak{s}_i(c)$, then $\forall t' \in St(M)$, if $t \sim^i_M t'$ then $(M,t') \in c$. But $\sim^i_M$ is reflexive, so $t \sim^i_M t$, and it follows that $(M,t) \in c$. Thus, $c \sqsubseteq \mathfrak{s}_i(c)$.

   E.2 We want to show that $\mathfrak{s}_i(\mathfrak{s}_i(c)) = \mathfrak{s}_i(c)$. The fact that $\mathfrak{s}_i(c) \subseteq \mathfrak{s}_i(\mathfrak{s}_i(c))$ follows from **E.1**. The fact that $\mathfrak{s}_i(\mathfrak{s}_i(c)) \subseteq \mathfrak{s}_i(c)$ follows from the transitivity of $\sim^i$. We omit the details, which are straightforward but tedious.

∎

Remark 4.3.5. *Consider a modal language Φ over propositions P and family of relations I. Consider the modal formulas given by*

$$\phi := p \mid \phi_1 \wedge \phi_2 \mid [i]\phi,$$

where p is a basic proposition, with the corresponding usual notion of satisfaction over Kripke models for propositions, conjunction and the box modality, as defined in Chapter 2. We abuse the notation and use a formula ϕ to denote the set of all pointed Kripke structures that satisfy ϕ. With the

help of the above theorem, one can establish a correspondence between the n-scs satisfying the premise in (2) and the modal system K_n [FHMV95] in the sense that ϕ is above ϕ' in the lattice if and only if we can derive in K_n that ϕ implies ϕ' (written $\vdash_{K_N} \phi \Rightarrow \phi'$). Similarly, for the n-scs satisfying (3) and the epistemic system $S4_n$ [FHMV95].

We conclude this chapter with a brief discussion of compactness in the constraint system $\mathbf{C}(\Delta_{\mathfrak{M}})$. We will see in the following chapter that compact elements of the constraint system are particularly important in the context of the process calculi we will define, so it may be useful to have a constraint system that consists entirely of compact elements.

Corollary 4.3.6. *If $\Delta_{\mathfrak{M}}$ is a finite set (this occurs if $\mathfrak{M}$ is a finite set of finite state Kripke structures), then every element of the constraint system is compact.*

This result follows immediately from the fact that in Theorem 4.3.4, we showed that the compact elements of $\mathbf{C}(\Delta_{\mathfrak{M}})$ are the cofinite subsets of $\Delta_{\mathfrak{M}}$. If $\Delta_{\mathfrak{M}}$ is a finite set, then every subset of $\Delta_{\mathfrak{M}}$ is cofinite, and therefore each element of the lattice is compact, even $\mathfrak{g}_G(c)$ (Remark 4.2.2).

Five

Space and Knowledge in Processes

In this chapter we introduce two CCP variants: *Spatial CCP (SCCP)* and *Epistemic CCP (ECCP)*. Spatial CCP is a conservative extension of CCP to model agents with spaces, possibly nested, in which they can store information and run processes. Its underlying constraint system is a spatial constraint system. Epistemic CCP extends the Spatial CCP with additional rules to model agents that interact by asking and computing knowledge within the spatial information distribution. Its underlying constraint system is an epistemic constraint system. In this chapter we will give the syntax of both spatial and epistemic processes, and we will discuss intuitions about their behaviour and the meanings of the different operators. We begin by defining two properties of constraint systems which are necessitated by the syntax of the processes.

For semantic reasons, we require our spatial constraint systems to have the following two properties from now on.

Definition 5.0.7 (Continuous)**.** *An n-scs*

$$\boldsymbol{C} = (\mathit{Con}, \mathit{Con}_0, \sqsubseteq, \sqcup, \mathit{true}, \mathit{false}, \mathfrak{s}_1, \ldots, \mathfrak{s}_n)$$

is said to be continuous *if and only if for every directed set* $S \subseteq \mathit{Con}$ *and every* $\mathfrak{s}_i$, $\mathfrak{s}_i(\bigsqcup S) = \bigsqcup_{e \in S} \mathfrak{s}_i(e)$.

Definition 5.0.8 (Space-compact)**.** *An n-scs*

$$\boldsymbol{C} = (\mathit{Con}, \mathit{Con}_0, \sqsubseteq, \sqcup, \mathit{true}, \mathit{false}, \mathfrak{s}_1, \ldots, \mathfrak{s}_n)$$

is said to be space-compact *if and only if for every* $\mathfrak{s}_i$*, if* $c \in Con_0$ *then* $\mathfrak{s}_i(c) \in Con_0$.

The examples of constraint systems we discussed in the last chapter (Applications, Section 4.3) can be shown to be continuous. Aumann ecs's are space-compact under the additional condition that every set in each partition is finite. A Kripke scs is space-compact if the inverse of the accessibility relation is finitely-branching. In the special case of Kripke ecs's this is the same as requiring each agent's accessibility relation to be finitely-branching since these relations are reflexive.

5.1 Syntax

The following syntax of processes will be common to both calculi.[1]

Definition 5.1.1. *Let* $\boldsymbol{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false, \mathfrak{s}_1, \ldots, \mathfrak{s}_n)$ *be a continuous and space compact n-scs. Let* $A = \{1, ..., n\}$ *be the set of agents. Assume a countable set of variables* $Vars = \{X, Y, \ldots\}$*. The terms are given by the following syntax:*

$$P, Q \ldots ::= \boldsymbol{0} \mid \boldsymbol{tell}(c) \mid \boldsymbol{ask}(c) \rightarrow P \mid P \parallel Q \mid [P]_i \mid X \mid \mu X.P$$

where $c \in Con_0$*,* $i \in A$*, and* $X \in Vars$*. A term* T *is said to be closed iff every variable* X *in* T *occurs in the scope of an expression* $\mu X.P$*. We will refer to closed terms as* processes *and use Proc to denote the class of all processes.*

Notation. Given $I = \{i_1, \ldots, i_m\}$ we use $\prod_{i \in I} P_i$ to represent for

$$P_{i_1} \parallel \ldots \parallel P_{i_m}.$$

5.1.1 Basic Processes

Before giving semantics to our processes, we give some intuitions about their behaviour. The *basic processes* are tell, ask, and parallel composition and

[1] For the sake of space and clarity, we dispense with the local/hiding operator.

they are defined as in standard CCP [SJPR91]. Recall that a process executes in the context of a store, from which it can get information and which the process can also update by adding information. Intuitively, **tell**(c) in store d adds c to d to make c available to other processes with access to this store. This addition, represented as $d \sqcup c$, is performed whether or not $d \sqcup c = \mathit{false}$. The process **ask**$(c) \rightarrow P$ in a store e may execute P if c is entailed by e, i.e., $c \sqsubseteq e$. The process $P \parallel Q$ stands for the *parallel execution* of P and Q. Variants of the following basic example will be referred to throughout the paper.

Example 5.1.2. *Let $P =$ **tell**(c) and $Q =$ **ask**$(c) \rightarrow$ **tell**(d). From the above intuitions, in the execution of the process $P \parallel Q$ starting with the empty store true, first c will be added to the store by P, and then Q will be able to execute and add d to the store, resulting in the final store of $c \sqcup d$.*

5.1.2 Spatial Processes

Our spatial CCP variant can be thought of as a *shared-spaces* model of computation. Each agent $i \in A$ may have computational spaces of the form $[\cdot]_i$ where processes as well as other agents' spaces may exist. It also has a space function $\mathfrak{s}_i$ representing the information stored in its spaces. Recall that $\mathfrak{s}_i(c)$ states that c holds in the space of agent i. Similarly, $\mathfrak{s}_i(\mathfrak{s}_j(c))$ means that c holds in the store that agent j has within the space of agent i. Unlike any other CCP calculus, it is possible to have agents with inconsistent information since $c \sqcup d = \mathit{false}$ does not necessarily imply $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) = \mathit{false}$ (see space consistent constraint system in Definition 4.1.3).

The spatial construction $[P]_i$ represents a process P running within the space of agent i. Any information c that P produces is available to processes that lie within the same space.

Notation. We will use $[P]_G$, where $G \subseteq A$, as an abbreviation of $\prod_{i \in G} [P]_i$.

Example 5.1.3. *Consider the process $[P]_i \parallel [Q]_i$ with $P =$ **tell**(c) and $Q =$ **ask**$(c) \rightarrow$ **tell**(d) as in Example 5.1.2. From the above intuitions we*

see that $[P]_i$ *will act first, adding* c *to the store of agent* i*, and then* $[Q]_i$ *will be able to execute and add* d *to the store of agent* i*, so in the end we will have* $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)$*. Similarly,* $[P \parallel Q]_i$ *will end up producing* $c \sqcup d$ *in the store of agent* i*, i.e.,* $\mathfrak{s}_i(c \sqcup d)$ *which from the scs axioms is equivalent to* $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)$*. In fact, for any processes* P_1 *and* P_2*,* $[P_1]_i \parallel [P_2]_i$ *and* $[P_1 \parallel P_2]_i$ *have the same behaviour.*

On the other hand, consider the process $[P]_j \parallel [Q]_i$ *for* $i \neq j$*. In this case,* d *will not be added to the space of* i *because* c *is not made available for agent* i*. Also in* $P \parallel [Q]_i$*,* d *is not added to the the space of* i*. In this case, however, we may view the* c *told by* P *as being available at an outermost space that* does not belong to any agent*. This does not mean that* c *holds everywhere, i.e.,* globally *(Def. 4.1.6).*

We also allow nesting of the space operator: for the process $[[P]_i]_j$*, the execution will result in* $\mathfrak{s}_j(\mathfrak{s}_i(c))$ *being added to the store. This represents* c *holding in the agent* i*'s space within the space of agent* j*.*

Finally, consider $[P]_{\{i,j\}} \parallel [[Q]_i]_j$*. Here,* $\mathfrak{s}_i(c)$ *and* $\mathfrak{s}_j(c)$ *will both be added to the store, but* d *will not necessarily be added to agent* i*'s space within agent* j*'s space because in an scs although* $\mathfrak{s}_i(c)$ *and* $\mathfrak{s}_j(c)$ *hold,* $\mathfrak{s}_j(\mathfrak{s}_i(c))$ *may not hold.*

5.1.3 Epistemic Processes

For our epistemic CCP variant, we shall further require that the underlying constraint system be an epistemic constraint system. This gives the operator $[\cdot]_i$ additional behaviour. From an epistemic point of view, the information c produced by P not only becomes available to agent i, as in the spatial case, but also it becomes a *fact*. This does not necessarily mean, of course, that c will be available everywhere, as there are facts that some agents may not know. It does mean, however, that unlike the spatial case, we cannot allow agents' spaces to include inconsistent information, as facts *cannot be contradictory*: in an ecs, $c \sqcup d = \mathit{false}$ implies $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) = \mathit{false}$.

Operationally, $[P]_i$ causes any information c produced by P to become available not only in the space of agent i but also in any space in which

$[P]_i$ is included. This is because epistemically $\mathfrak{s}_i(c) = c \sqcup \mathfrak{s}_i(c)$ so if $\mathfrak{s}_j(\mathfrak{s}_i(c))$ holds, then $\mathfrak{s}_j(c \sqcup \mathfrak{s}_i(c))$ also holds, as does $c \sqcup \mathfrak{s}_j(c \sqcup \mathfrak{s}_i(c))$. This can be viewed as saying that c propagates outward in space.

Example 5.1.4. *Consider* $[Q \parallel [P]_i]_j$ *with* $P = \textbf{\textit{tell}}(c)$ *and* $Q = \textbf{\textit{ask}}(c) \to \textbf{\textit{tell}}(d)$ *as in Example 5.1.2. Notice that from executing* P *we obtain* $\mathfrak{s}_j(\mathfrak{s}_i(c))$. *In the spatial case,* Q *will not necessarily tell* d *because in a spatial constraint system,* $\mathfrak{s}_j(\mathfrak{s}_i(c))$ *may not entail* $\mathfrak{s}_j(c)$. *On the other hand, in the epistemic case,* Q *will tell* d *since in an epistemic constraint system,* $\mathfrak{s}_j(\mathfrak{s}_i(c)) = \mathfrak{s}_j(c \sqcup \mathfrak{s}_i(c))$ *which entails* $\mathfrak{s}_j(c)$ *by property* ***S.2****.*

5.1.4 Infinite Processes

Unbounded behaviour is specified using recursive definitions of the form $\mu X.P$ whose behaviour is that of $P[\mu X.P/X]$, i.e., P with every free occurrence of X replaced with $\mu X.P$. We assume that recursion is *ask guarded*: i.e., for every $\mu X.P$, each occurrence of X in P occurs under the scope of an ask process. For simplicity we assume an implicit "**ask**(*true*) $\to$ " in unguarded occurrences of X.

Recursive definitions allow us to represent complex spatial and epistemic situations, like the following.

Definition 5.1.5 (Global process)**.** *Given* $G \subseteq A$ *and a basic process* P *we define*

$$\mathit{global}(G, P) \stackrel{\texttt{def}}{=} P \parallel \mu X.\, [P \parallel X]_G\,.$$

Intuitively, in $\mathit{global}(G, P)$ *any information* c *produced by* P *will be available at any space or any nesting of spaces involving only the agents in* G.

Example 5.1.6. *Consider the process* $\mathit{global}(G, P) \parallel \big[[\ldots[Q]_{k_m}\ldots]_{k_2}\big]_{k_1}$ *where* $G = \{k_1, ..., k_m\} \subseteq A$, *with* $P = \textbf{\textit{tell}}(c)$ *and* $Q = \textbf{\textit{ask}}(c) \to \textbf{\textit{tell}}(d)$ *as in Example 5.1.2. The process* $\mathit{global}(G, P)$ *eventually makes* c *available in the (nested) space* $\big[[\ldots[\cdot]_{k_m}\ldots]_{k_2}\big]_{k_1}$ *and thus* Q *will tell* d *in that space.*

5.2 Reduction Semantics

We now define a structural operational semantics (sos) for SCCP and ECCP. We begin with the structural operational semantics for the spatial case. The structural operational semantics for the epistemic case extends the spatial one with an additional rule and the assumption that the underlying constraint system is epistemic. From now on we will use the following convention:

Convention 1. *The following sections assume an underlying continuous and space-compact spatial constraint system*

$$\boldsymbol{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false, \mathfrak{s}_1, \ldots, \mathfrak{s}_n).$$

Definition 5.2.1 (Configuration)**.** *A* configuration *is a pair* $\langle P, c\rangle \in Proc \times Con$ *where* c *represents the current* spatial distribution *of information in* P*. We use* $Conf$ *with typical elements* $\gamma, \gamma', \ldots$ *to denote the set of configurations.*

Convention 2. *Since we have two process calculi, SCCP and ECCP, with different transition relations, sometimes index the transitions with "*$\mathtt{s}$*" if they are interpreted for SCCP, and with "*$\mathtt{e}$*" if they are interpreted for ECCP. We often omit the index when it is irrelevant or obvious.*

5.2.1 Operational Semantics for SCCP

The structural operational semantics for SCCP is given by means of a transition relation between configurations $\longrightarrow_{\mathtt{s}} \subseteq Conf \times Conf$ in Table 5.1.

The rules A, T, PL, and R for the basic processes and recursion are standard in CCP and it is easy to see that they realize the intuitions discussed above (see [SJPR91]). The rule S for the new spatial operator is more involved and we explain it below. First we introduce the following central notion defining the projection of a spatial constraint c for agent i.

Definition 5.2.2 (Views)**.** *The agent* i*'s* view *of* c*,* c^i*, is given by* $c^i = \bigsqcup\{d \mid \mathfrak{s}_i(d) \sqsubseteq c\}$.

$$\mathbf{T}\ \frac{}{\langle \mathbf{tell}(c), d\rangle \longrightarrow_{\mathtt{s}} \langle \mathbf{0}, d \sqcup c\rangle} \qquad \mathbf{A}\ \frac{c \sqsubseteq d}{\langle \mathbf{ask}\ (c)\ \rightarrow\ P, d\rangle \longrightarrow_{\mathtt{s}} \langle P, d\rangle}$$

$$\mathbf{PL}\ \frac{\langle P, d\rangle \longrightarrow_{\mathtt{s}} \langle P', d'\rangle}{\langle P \parallel Q, d\rangle \longrightarrow_{\mathtt{s}} \langle P' \parallel Q, d'\rangle} \qquad \mathbf{R}\ \frac{\langle P[\mu X.P/X], d\rangle \longrightarrow_{\mathtt{s}} \gamma}{\langle \mu X.P, d\rangle \longrightarrow_{\mathtt{s}} \gamma}$$

$$\mathbf{S}\ \frac{\langle P, c^i\rangle \longrightarrow_{\mathtt{s}} \langle P', c'\rangle}{\langle [P]_i\,, c\rangle \longrightarrow_{\mathtt{s}} \langle [P']_i\,, c \sqcup \mathfrak{s}_i(c')\rangle}$$

Figure 5.1: Rules for SCCP. The projection c^i is given in Definition 5.2.2. The symmetric right rule for PL, PR, is omitted.

Intuitively, c^i represents all the information the agent i may see or have when c is true. For example if $c = \mathfrak{s}_i(d) \sqcup \mathfrak{s}_j(e)$ then agent i sees d, so $d \sqsubseteq c^i$. Notice that if $\mathfrak{s}_i(d) = \mathfrak{s}_i(d')$ then $(\mathfrak{s}_i(d))^i$ entails both d and d'. This is intended because $\mathfrak{s}_i(d) = \mathfrak{s}_i(d')$ means that agent i cannot distinguish d from d'. The constraint c^i enjoys the following property which will be useful later on.

Proposition 5.2.3. *For any constraint c, $c \sqcup \mathfrak{s}_i(c^i) = c$.*

We need a lemma to prove this proposition.

Lemma 5.2.4. *The set $\{d \mid \mathfrak{s}_i(d) \sqsubseteq c\}$ is directed.*

Proof. We prove that the set satisfies the stronger property of being closed under joins. If $\mathfrak{s}_i(a) \sqsubseteq c$ and $\mathfrak{s}_i(b) \sqsubseteq c$, then c is an upper bound for $\mathfrak{s}_i(a)$ and $\mathfrak{s}_i(b)$, so $\mathfrak{s}_i(a) \sqcup \mathfrak{s}_i(b) \sqsubseteq c$, but since $\mathfrak{s}_i$ distributes over joins, $\mathfrak{s}_i(a \sqcup b) \sqsubseteq c$, so $a \sqcup b \in \{d \mid \mathfrak{s}_i(d) \sqsubseteq c\}$. ∎

Now we can prove that $c \sqcup \mathfrak{s}_i(c^i) = c$.

Proof. To show that $c \sqcup \mathfrak{s}_i(c^i) = c$ it is sufficient to show that $\mathfrak{s}_i(c^i) \sqsubseteq c$. Now, $\mathfrak{s}_i(c^i) = \mathfrak{s}_i(\bigsqcup\{d \mid \mathfrak{s}_i(d) \sqsubseteq c\})$, and since we just proved that this is a di-

rected set, from continuity of $\mathfrak{s}_i$ we conclude that $\mathfrak{s}_i(c^i) = \bigsqcup\{\mathfrak{s}_i(d) \mid \mathfrak{s}_i(d) \sqsubseteq c\}$, which is clearly below c. ∎

Now we explain the S rule for the spatial operator. First, in order for $[P]_i$ with store c to make a reduction, the information agent i sees or has in c must allow P to make the reduction. Hence we run P with store c^i. Second, the information d that P's reduction would add to c^i is what $[P]_i$ adds to the space of agent i as stated in Proposition 5.2.5 below.

Proposition 5.2.5. *If*

$$\langle P, c^i\rangle \longrightarrow \langle P', c^i \sqcup d\rangle$$

then

$$\langle [P]_i, c\rangle \longrightarrow \langle [P']_i, c \sqcup \mathfrak{s}_i(d)\rangle.$$

Proof. From the S rule in the operational semantics, if $\langle P, c^i\rangle \longrightarrow \langle P', c^i \sqcup d\rangle$ then $\langle [P]_i, c\rangle \longrightarrow \langle [P]'_i, c \sqcup \mathfrak{s}_i(c^i \sqcup d)\rangle$. Since $\mathfrak{s}_i$ distributes over joins, $c \sqcup \mathfrak{s}_i(c^i \sqcup d) = c \sqcup \mathfrak{s}_i(c^i) \sqcup \mathfrak{s}_i(d)$, and by Proposition 5.2.3 above, this constraint is equal to $c \sqcup \mathfrak{s}_i(d)$, completing the proof. ∎

The following corollary shows that the store for $[P]_i$ only changes in a given transition if the store for P changes in the corresponding transition.

Corollary 5.2.6. *If* $\langle P, c^i\rangle \longrightarrow \langle P', c^i\rangle$ *then* $\langle [P]_i, c\rangle \longrightarrow \langle [P']_i, c\rangle$.

Proof. Follows immediately from Proposition 5.2.5. ∎

Next we show an instructive reduction involving the use of the S rule.

Example 5.2.7. *Take* $[P]_i \parallel [Q]_i$ *with* $P = \boldsymbol{tell}(c)$ *and* $Q = \boldsymbol{ask}(c) \rightarrow \boldsymbol{tell}(d)$ *as in Example 5.1.2. One can verify that*

$$\langle [P]_i \parallel [Q]_i, \mathit{true}\rangle \longrightarrow \langle [\boldsymbol{0}]_i \parallel [Q]_i, \mathfrak{s}_i(c)\rangle \longrightarrow \langle [\boldsymbol{0}]_i \parallel [\boldsymbol{0}]_i, \mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)\rangle.$$

Recall that $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d) = \mathfrak{s}_i(c \sqcup d)$. *A more interesting example is* $[\boldsymbol{tell}(c')]_i \parallel [Q]_i$ *under the assumption that* $\mathfrak{s}_i(c) = \mathfrak{s}_i(c')$. *We have*

$$\langle [\boldsymbol{tell}(c')]_i \parallel [Q]_i, \mathit{true}\rangle \longrightarrow \langle [\boldsymbol{0}]_i \parallel [Q]_i, \mathfrak{s}_i(c')\rangle \longrightarrow \langle [\boldsymbol{0}]_i \parallel [\boldsymbol{0}]_i, \mathfrak{s}_i(c') \sqcup \mathfrak{s}_i(d)\rangle.$$

At first glance, it may seem strange that Q is allowed to execute when c' holds in i's store, rather than c. But actually, this is the desired behaviour because $\mathfrak{s}_i(c) = \mathfrak{s}_i(c')$, so c and c' are regarded as equivalent by i, and so a process in i's space must behave the same way whether c' or c is in i's store.

5.2.2 Operational Semantics for ECCP

Now we present the operational semantics for the epistemic case. The ECCP structural operational semantics assumes that the underlying constraint system is an epistemic constraint system. As explained earlier, in the epistemic setting, for the process $[P]_i$, the information c produced by P not only becomes available to agent i but also becomes a *fact* within the hierarchy of spaces in which $[P]_i$ is included. This means that c is available not only in the space of agent i but also in any space in which $[P]_i$ is included. We can view this as saying that c propagates *outward* through the spaces $[P]_i$ is in and this is partly realized by the equation $\mathfrak{s}_i(c) = c \sqcup \mathfrak{s}_i(c)$ which follows from **E.1** (Definition 4.2.1). Mirroring this constraint equation and epistemic reasoning, the behaviour of $[P]_i$ and $P \parallel [P]_i$ must also be equated, since $[P]_i$ can only produce factual information. This makes $[P]_i$ somewhat reminiscent of the replication/bang operator in the π-calculus [MPW92]. For ECCP we include the new E Rule in Table 5.2. As illustrated in Example 5.2.8, Rule E is necessary for the behaviour of $[P]_i$ and $P \parallel [P]_i$ to be the same, corresponding to the epistemic principles we wish to represent.

The structural operational semantics of ECCP is given by the transition relation between configurations $\longrightarrow_{\mathsf{e}} \subseteq \mathit{Conf} \times \mathit{Conf}$ defined in Table 5.2 and assuming the underlying constraint system to be epistemic.

Example 5.2.8. *Let $R = [P \parallel [Q]_i]_j$ and $T = [P \parallel [Q]_i \parallel Q]_j$ with $P = \boldsymbol{tell}(c)$ and $Q = \boldsymbol{ask}(c) \rightarrow \boldsymbol{tell}(d)$ as in Example 5.1.2. Our operational semantics allows us to equate R and T, which mimics epistemic principles. Even assuming an epistemic constraint system, with only the rules of SCCP (i.e., without Rule E), T can produce $\mathfrak{s}_j(d)$, d in the store of agent j, but R is not necessarily able to do this: One can verify that there are T', e' s.t.*

$$\mathbf{T}\ \frac{}{\langle \mathbf{tell}(c), d\rangle \longrightarrow_{\mathsf{e}} \langle \mathbf{0}, d \sqcup c\rangle} \qquad \mathbf{A}\ \frac{c \sqsubseteq d}{\langle \mathbf{ask}\ (c)\ \rightarrow\ P, d\rangle \longrightarrow_{\mathsf{e}} \langle P, d\rangle}$$

$$\mathbf{PL}\ \frac{\langle P, d\rangle \longrightarrow_{\mathsf{e}} \langle P', d'\rangle}{\langle P \parallel Q, d\rangle \longrightarrow_{\mathsf{e}} \langle P' \parallel Q, d'\rangle} \qquad \mathbf{R}\ \frac{\langle P[\mu X.P/X], d\rangle \longrightarrow_{\mathsf{e}} \gamma}{\langle \mu X.P, d\rangle \longrightarrow_{\mathsf{e}} \gamma}$$

$$\mathbf{S}\ \frac{\langle P, c^i\rangle \longrightarrow_{\mathsf{e}} \langle P', c'\rangle}{\langle [P]_i\,, c\rangle \longrightarrow_{\mathsf{e}} \langle [P']_i\,, c \sqcup \mathfrak{s}_i(c')\rangle} \qquad \mathbf{E}\ \frac{\langle P, c\rangle \longrightarrow_{\mathsf{e}} \langle P', c'\rangle}{\langle [P]_i\,, c\rangle \longrightarrow_{\mathsf{e}} \langle [P]_i \parallel P', c'\rangle}$$

Figure 5.2: Rules for ECCP (see Convention 2). Recall that the projection $c^i = \bigsqcup\{d \mid \mathfrak{s}_i(d) \sqsubseteq c\}$ as in Definition 5.2.2. The symmetric right rule for PL, PR, is omitted.

$\langle T, true\rangle \longrightarrow_s^* \langle T', e'\rangle$ and $\mathfrak{s}_j(d) \sqsubseteq e'$, while, in general, for all R', e'' such that $\langle R, true\rangle \longrightarrow_s^* \langle R', e''\rangle$ we have $\mathfrak{s}_j(d) \not\sqsubseteq e''$. With the rules of ECCP, however, one can verify for each e' such that $\langle T, true\rangle \longrightarrow_e^* \langle T', e'\rangle$ there exists e'', $\langle R, true\rangle \longrightarrow_e^* \langle R', e''\rangle$ such that $e' \sqsubseteq e''$ (and vice-versa with the roles of R and T interchanged).

Six

Observable Behaviour of Space and Knowledge

A standard notion of observable behaviour in CCP involves infinite fair computations and information constructed as the limit of finite approximations. For our calculi, however, these limits may result in infinite (or non-compact) objects involving arbitrary nesting of spaces, or epistemic specifications such as common knowledge. In this chapter we provide techniques useful for analyzing the observable behaviour of such processes using simpler finitary concepts and compositional reasoning.[1]

The notion of *fairness* is central to the definition of observational equivalence for CCP. We introduce this notion following [FGMP97]. To define fairness we need several subsidiary definitions. First, note that any derivation of a transition involves an application of Rule A or Rule T.

Definition 6.0.9 (Active)**.** *We say that P is* active *in a transition $t = \gamma \longrightarrow \gamma'$ if there exists a derivation of t where rule A or T is used to produce a transition of the form $\langle P, d\rangle \longrightarrow \gamma''$.*

Definition 6.0.10 (Enabled)**.** *We say that P is* enabled *in γ if there exists γ' such that P is active in $\gamma \longrightarrow \gamma'$.*

Now we can define a fair computation.

[1] See Convention 2.

Definition 6.0.11 (Fair Computation)**.** *A computation*

$$\gamma_0 \longrightarrow \gamma_1 \longrightarrow \gamma_2 \longrightarrow \dots$$

is said to be fair *if for each process P enabled in some γ_i there exists $j \geq i$ such that P is active in γ_j.*

Note that a finite fair computation is guaranteed to be *maximal*, namely no outgoing transitions are possible from its last configuration.

6.1 Observing Limits.

A standard notion of observables for CCP is the *result* computed by a process for a given initial store. The result of a computation is defined as the least upper bound of all the stores occurring in the computation, which, thanks to the monotonic properties of our calculi, form an increasing chain (this is easy to verify by looking at the operational semantics). Here is the formal definition.

Definition 6.1.1 (Result of a computation)**.** *Let ξ be a computation (finite or infinite) of the form*

$$\langle Q_0, d_0\rangle \longrightarrow \langle Q_1, d_1\rangle \longrightarrow \langle Q_2, d_2\rangle \longrightarrow \dots$$

We define the result of ξ as

$$\mathit{Result}(\xi) = \bigsqcup_i d_i.$$

In our calculi all fair computations from a configuration have the same result.

Proposition 6.1.2. *Let γ be a configuration and let ξ_1 and ξ_2 be two computations of γ. If ξ_1 and ξ_2 are fair, then $\mathit{Result}(\xi_1) = \mathit{Result}(\xi_2)$.*

This means that we can define the result of a process, rather than just a computation.

Definition 6.1.3 (Result of a process)**.** *Now, for a configuration γ, if ξ is any fair computation of γ, we can set*

$$\mathit{Result}(\gamma) \stackrel{\mathtt{def}}{=} \mathit{Result}(\xi).$$

Now we can define the observation function, which also takes the initial store into account.

Definition 6.1.4 (Observation)**.** *The* observation *is a function $\mathcal{O} : \mathit{Proc} \to \mathit{Con}_0 \to \mathit{Con}$ mapping a process and an initial store to a final store. For a process P and an initial store d, we define*

$$\mathcal{O}(P)(d) = \mathit{Result}(\langle P, d\rangle).$$

Example 6.1.5. *The observation we make of the recursive process $\mathit{global}(G, \boldsymbol{tell}(c))$ on input true is the limit $\mathfrak{g}_G(c)$ (Definition 4.1.6). In other words,*

$$\mathcal{O}(\mathit{global}(G, \boldsymbol{tell}(c)))(\mathit{true}) = \mathfrak{g}_G(c).$$

Now we define an equivalence between processes based on the observation function.

Definition 6.1.6 (Observational equivalence)**.** *We say that P and Q are observationally equivalent, written $P \sim_o Q$, if and only if for all $d \in \mathit{Con}_0$,*

$$\mathcal{O}(P)(d) = \mathcal{O}(Q)(d).$$

Example 6.1.7. *Let $P = \boldsymbol{tell}(c)$ and $Q = \boldsymbol{ask}(c) \to \boldsymbol{tell}(d)$ as in Example 5.1.2. Let $R = [P]_i \parallel [Q]_i$, and let $T = [P \parallel Q]_i$. R and T are observationally equivalent, that is, $R \sim_o T$.*

The relation $\sim_o$ can be shown to be a congruence, that is, it is preserved under arbitrary contexts. Recall that a context C is a term with a hole $\bullet$, so that replacing it with a process P yields a process term $C(P)$. For example, if $C = [\bullet]_i$ then $C(\mathbf{tell}(d)) = [\mathbf{tell}(d)]_i$.

Theorem 6.1.8. *$P \sim_o Q$ if and only if for every context C, $C(P) \sim_o C(Q)$.*

We will not prove this theorem, because at the end of the chapter it will follow directly from some other results.[2]

6.2 Observing Barbs

In the next section we will show that the above notion of observation has pleasant and useful closure properties like those of basic CCP. Some readers, however, may feel uneasy with observable behaviour involving notions such as *infinite fair* computations and limits, as well as possibly non-compact elements in the constraint system. Fortunately, we can give a finitary characterization of behavioural equivalence for our calculi, involving only finite computations and compact elements.

Definition 6.2.1 (Barb)**.** *In the constraint system*

$$\boldsymbol{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false, \mathfrak{s}_1, \ldots, \mathfrak{s}_n),$$

A barb *is an element of* Con_0.

Definition 6.2.2 (Barb satisfaction)**.** *We say that* $\gamma = \langle P, d\rangle$ satisfies *the barb* c*, written* $\gamma \downarrow_c$*, if and only if* $c \sqsubseteq d$*. We also say that* c *is a* strong barb *for* P.

Definition 6.2.3 (Weak barb satisfaction)**.** *We say that the configuration* γ weakly satisfies *the barb* c*, written* $\gamma \Downarrow_c$*, if and only if there is a sequence of configurations* $\gamma_1, \gamma_2, ..., \gamma_n$ *such that*

$$\gamma \longrightarrow \gamma_1 \longrightarrow \gamma_2 \longrightarrow ... \longrightarrow \gamma_n$$

and $\gamma_n \downarrow_c$*. We also say that* c *is a* weak barb *for* P.

Example 6.2.4. *Consider the process* $R = \boldsymbol{ask}\ c \ \rightarrow\ [\boldsymbol{tell}(d)]_i$ *and the configuration* $\langle R, c\rangle$*. Notice that* $\langle R, c\rangle \downarrow_c$ *and also* $\langle R, c\rangle \Downarrow_{\mathfrak{s}_i(d)}$.

[2] This theorem follows from Theorem 6.3.9 which says that two processes are observationally equivalent if and only if their denotations are the same, and since the definition of denotation is completely computational, it is a congruence.

On the other hand, $\langle R, true\rangle \downarrow_{true}$ *and also* $\langle R, true\rangle \Downarrow_{true}$*, and these are the only weak or strong barbs for this configuration.*

Finally, $\langle \textbf{\textit{tell}}(c) \parallel R, true\rangle \Downarrow_c$ *and* $\langle \textbf{\textit{tell}}(c) \parallel R, true\rangle \Downarrow_{\mathfrak{s}_i(d)}$*, even though the only weak barb for this configuration is true.*

Definition 6.2.5 (Barb equivalence)**.** *P and Q are* barb equivalent*, written* $P \sim_b Q$*, if and only if for all stores, P and Q weakly satisfy the same barbs:* $\forall c, d \in Con_0$*,*

$$\langle P, d\rangle \Downarrow_c \iff \langle Q, d\rangle \Downarrow_c .$$

We now establish the correspondence between our process equivalences. First we recall some notions from domain theory central to our proof of the correspondence.

Definition 6.2.6 (Chain)**.** *In a partial order* $(S, \sqsubseteq)$ *we call a totally ordered subset of S with a least element a* chain*. That is,* $D \subseteq S$ *is a chain if* $D = \{d_0, d_1, d_2, ...\}$ *where*

$$d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq ... \sqsubseteq d_n \sqsubseteq ...$$

Definition 6.2.7 (Cofinal)**.** *Two (possibly infinite) chains* $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ *and* $e_0 \sqsubseteq e_1 \sqsubseteq \cdots \sqsubseteq e_n \sqsubseteq \ldots$ *are said to be* cofinal *if for all* d_i *there exists an* e_j *such that* $d_i \sqsubseteq e_j$ *and vice versa.*

The following lemma is very useful for the correspondence between our process equivalences.

Lemma 6.2.8. *Consider a complete lattice* $(S, \sqsubseteq)$*, and chains* $D, E \subseteq S$*. The following results hold:*

1. *If D and E are cofinal, then they have the same limit,, that is* $\bigsqcup D = \bigsqcup E$*.*
2. *If all elements of D and E are* compact *and* $\bigsqcup D = \bigsqcup E$*, then the two chains are cofinal.*

Proof. Let $D = \{d_0, d_1, ...\}$ and $E = \{e_0, e_1, ...\}$ such that

$$d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq ... \sqsubseteq d_n \sqsubseteq ...$$

and

$$e_0 \sqsubseteq e_1 \sqsubseteq e_2 \sqsubseteq ... \sqsubseteq e_n \sqsubseteq ...$$

1. For each $d_i \in D$, there exists $e_{k_i} \in E$ such that $d_i \sqsubseteq e_{k_i}$. Define $f_l = \bigsqcup_{i=0}^{l} e_{k_i}$. Then for all l, $d_l \sqsubseteq f_l$ and $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq ....$ So the chain $d_0, d_1, d_2, ...$ is dominated by the chain $f_0, f_1, f_2, ...$ and therefore $\bigsqcup D \sqsubseteq \bigsqcup_{i\in\mathbb{N}} f_i$. However $\bigsqcup_{i\in\mathbb{N}} f_i = \bigsqcup_{i\in\mathbb{N}} e_{k_i} = \bigsqcup E$, proving that $\bigsqcup D \sqsubseteq \bigsqcup E$. But we can prove in exactly the same way that $\bigsqcup E \sqsubseteq \bigsqcup D$, so we have that $\bigsqcup D = \bigsqcup E$.

2. If $\bigsqcup D = \bigsqcup E$ then for arbitrary d_i, since $d_i \sqsubseteq \bigsqcup D$,$d_i \sqsubseteq \bigsqcup E$, and since d_i is compact, by definition there must be e_j such that $d_i \sqsubseteq e_j$. The same reasoning can be used to prove that for every e_i, there exists d_j with $e_i \sqsubseteq d_j$. Therefore D and E are cofinal.

∎

Now we are almost ready to show that two processes are observationally equivalent if and only if they are barb equivalent. The proof of this correspondence shows that the stores of any pair of *fair* computations of equivalent processes form pairs of cofinal chains. But it also uses the following result about a relation between weak barbs and fair computations.

Lemma 6.2.9. *Let* $\langle P_0, d_0\rangle \longrightarrow \langle P_1, d_1\rangle \longrightarrow \ldots \longrightarrow \langle P_n, d_n\rangle \longrightarrow \ldots$ *be a fair computation. If* $\langle P_0, d_0\rangle \Downarrow_c$ *then there exists a store* d_i *s.t.,* $c \sqsubseteq d_i$.

Proof. Since $\langle P_0, d_0\rangle \Downarrow_c$, it means that there exists a computation

$$\langle P_0, d_0\rangle \longrightarrow \langle P'_1, d'_1\rangle \longrightarrow ... \longrightarrow \langle P'_n, d'_n\rangle$$

such that $c \sqsubseteq d'_n$. Any finite computation can be extended to a fair computation which is either finite and maximal or infinite. We only include the

proof for the case where the computation is infinite, because the finite case is similar. So let

$$\xi = \langle P_0, d_0\rangle \longrightarrow \langle P_1', d_1'\rangle \longrightarrow ... \longrightarrow \langle P_n', d_n'\rangle \longrightarrow$$

be an infinite fair computation. Then

$$Result(\langle P_0, d_0\rangle) = Result(\xi) = \bigsqcup_{i\in\mathbb{N}} d_i'$$

and $c \sqsubseteq d_n' \sqsubseteq \bigsqcup_{i\in\mathbb{N}} d_i'$. But we know that all fair computations have the same results, so $Result(\langle P_0, d_0\rangle) = \bigsqcup_{i\in\mathbb{N}} d_i$ also, and $c \sqsubseteq \bigsqcup_{i\in\mathbb{N}} d_i$. Finally, since c is a barb, c is compact, and so there is some d_i such that $c \sqsubseteq d_i$. ∎

Lemma 6.2.10. *For a process P and $c, d \in Con_0$,*

$$\langle P, d\rangle \Downarrow_c \text{ if and only if } c \sqsubseteq \mathcal{O}(P)(d).$$

Proof. From Lemma 6.2.9 it follows that if $\langle P, d\rangle \Downarrow_c$ then $c \sqsubseteq \mathcal{O}(P)(d)$. On the other hand, suppose that

$$\langle P, d\rangle \longrightarrow \langle P_1, d_1\rangle \longrightarrow \ldots \longrightarrow \langle P_n, d_n\rangle \longrightarrow \ldots$$

is a fair computation. If c is compact and $c \sqsubseteq \mathcal{O}(P)(d) = \bigsqcup_{i\in\mathbb{N}} d_i$ it follows from c's compactness that for some $d_i \in \{d, d_1, d_2, ...\}$, $c \sqsubseteq d_i$, and therefore $\langle P, d\rangle \Downarrow_c$. ∎

With these observations we can show that two processes are not observationally equivalent on a given input if and only if there is a compact element that tells them apart.

Theorem 6.2.11. *Observational equivalence and barb equivalence correspond: for all $P, Q \in Proc$,*

$$P \sim_o Q \iff P \sim_b Q.$$

Proof. Recall that our constraint systems are complete algebraic lattices, and that algebraicity means that every element is the supremum of the

compact elements below it. Since all barbs are compact elements, it follows from Lemma 6.2.10 that for any process P and any $d \in Con_0$,

$$\mathcal{O}(P)(d) = \bigsqcup\{c \in Con_0 \mid \langle P, d\rangle \Downarrow_c\}.$$

It follows immediately that if $P \sim_b Q$ then $P \sim_o Q$.

On the other hand, since Lemma 6.2.10 says that if $c \sqsubseteq \mathcal{O}(P)(d)$ then $\langle P, d\rangle \Downarrow c$, it follows that if $P \sim_o Q$ then $P \sim_b Q$. ∎

6.3 Denotational Semantics.

Now we give a denotational characterization of observable behaviour that allows us to reason compositionally about our spatial and epistemic processes. First we can show that the behaviour of a process P, $\mathcal{O}(P)$, is a closure operator on $\sqsubseteq$. The importance of $\mathcal{O}(P)$ being a closure operator on $\sqsubseteq$ is that a closure operator is fully determined by its set of fixed points, as discussed in Chapter 3. This property will later allow us to define the behaviour of a process compositionally.

Lemma 6.3.1. *For every P, $\mathcal{O}(P)$ is a closure operator on $\sqsubseteq$.*

Before the proof of the lemma, we need several more lemmas which will be used in the idempotence part of the proof.

Lemma 6.3.2. *If*

$$\langle Q, d\rangle \longrightarrow \langle Q', d'\rangle$$

and $d' \sqsubseteq e$ then

$$\langle Q, e\rangle \longrightarrow \langle Q', e\rangle$$

Proof. We prove this lemma by structural induction on Q. The cases for tell, parallel, and ask are the same as in standard CCP, so we only discuss the case where $Q = [P]_i$ for some P.

In SCCP, if $\langle [P]_i, d\rangle$ has a transition available, the derivation must be of the form

$$\frac{\langle P, d^i\rangle \longrightarrow \langle P', d''\rangle}{\langle [P]_i, d\rangle \longrightarrow \langle [P']_i, d \sqcup \mathfrak{s}_i(d'')\rangle}$$

Now since we assume that $d' \sqsubseteq e$ and we know that $d \sqsubseteq d'$ because $\langle Q, d\rangle \longrightarrow \langle Q', d'\rangle$, we know that $d \sqsubseteq e$, so clearly $d^i \sqsubseteq e^i$. Furthermore, $d' = d \sqcup \mathfrak{s}_i(d'') \sqsubseteq e$ so clearly $d'' \sqsubseteq e^i$. So by induction hypothesis, $\langle P, e^i\rangle \longrightarrow \langle P', e^i\rangle$. Now we can conclude that

$$\frac{\langle P, e^i\rangle \longrightarrow \langle P', e^i\rangle}{\langle [P]_i, e\rangle \longrightarrow \langle [P']_i, e \sqcup \mathfrak{s}_i(e^i)\rangle}$$

Of course, we already know that for any e, $e \sqcup \mathfrak{s}_i(e^i) = e$, so we conclude that if $\langle [P]_i, d\rangle \longrightarrow \langle [P']_i, d'\rangle$ and $d' \sqsubseteq e$ then $\langle [P]_i, e\rangle \longrightarrow \langle [P']_i, e\rangle$.

Now, in the ECCP case, where $Q = [P]_i$ for some P, there are two possibilities. If $\langle [P]_i, d\rangle \longrightarrow \langle [P']_i, d'\rangle$ then the reasoning is exactly the same as above. If, on the other hand, $\langle [P]_i, d\rangle \longrightarrow \langle [P]_i \parallel P', d'\rangle$, then the derivation must be as follows:

$$\frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle [P]_i, d\rangle \longrightarrow \langle [P]_i \parallel P', d'\rangle}$$

So if $d' \sqsubseteq e$ then by induction hypothesis, it is immediate that $\langle P, e\rangle \longrightarrow \langle P', e\rangle$, and we can make the following derivation:

$$\frac{\langle P, e\rangle \longrightarrow \langle P', e\rangle}{\langle [P]_i, e\rangle \longrightarrow \langle [P]_i \parallel P', e\rangle}$$

This completes the proof of the lemma. ∎

Lemma 6.3.3. *If $\langle P, e\rangle \longrightarrow \langle P', e'\rangle$ then there exists $c \in Con_0$ such that $c \sqsubseteq e$ and $\langle P, c\rangle \longrightarrow \langle P', c'\rangle$ for some c'.*

Proof. The proof is by structural induction on P. Since the other cases are the same as in standard CCP, we only include the case where $P = [Q]_i$ for some Q.

In SCCP or ECCP, if P's transition is justified by the S rule then there is a derivation of the form

$$\frac{\langle Q, e^i\rangle \longrightarrow \langle Q', e'\rangle}{\langle [Q]_i, e\rangle \longrightarrow \langle [Q']_i, e \sqcup \mathfrak{s}_i(e')\rangle}$$

So by the induction hypothesis, there must be $c \in Con_0$ such that $c \sqsubseteq e^i$ and $\langle Q, c\rangle \longrightarrow \langle Q', c'\rangle$ for some c'. Now, since $\mathfrak{s}_i$ is a compactness preserving

function, $\mathfrak{s}_i(c) \in Con_0$. Furthermore, we know that $c \sqsubseteq (\mathfrak{s}_i(c))^i$ so the following derivation holds:

$$\frac{\langle Q, (\mathfrak{s}_i(c))^i\rangle \longrightarrow \langle Q', x\rangle}{\langle [Q]_i, \mathfrak{s}_i(c)\rangle \longrightarrow \langle [Q']_i, \mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(x)\rangle}$$

for some x. Since $\mathfrak{s}_i(c) \in Con_0$, this proves the lemma in this case.

If we are in the ECCP case and the transition of $\langle [Q]_i, e\rangle$ is justified by the E rule, then we have that

$$\frac{\langle Q, e\rangle \longrightarrow \langle Q', e'\rangle}{\langle [Q]_i, e\rangle \longrightarrow \langle [Q]_i \parallel Q', e'\rangle}$$

so by the induction hypothesis, there exists $c \in Con_0$ such that $\langle Q, c\rangle \longrightarrow \langle Q', c'\rangle$. Then of course we can conclude that

$$\frac{\langle Q, c\rangle \longrightarrow \langle Q', c'\rangle}{\langle [Q]_i, c\rangle \longrightarrow \langle [Q]_i \parallel Q', c'\rangle}$$

which completes the proof of the lemma for this case. ∎

Lemma 6.3.4. *If*

$$\zeta_1 = \langle P, c_0\rangle \longrightarrow \langle P_1, c_1\rangle \longrightarrow \langle P_2, c_2\rangle \longrightarrow \ldots \longrightarrow \langle P_n, c_n\rangle \longrightarrow \ldots$$

is a fair computation, $c_0 \in Con_0$*, and* $e = \bigsqcup_i c_i$*, then*

$$\zeta_2 = \langle P, e\rangle \longrightarrow \langle P_1, e\rangle \longrightarrow \langle P_2, e\rangle \longrightarrow \ldots \longrightarrow \langle P_n, e\rangle \longrightarrow_{\ldots}$$

is also a fair computation.

Proof. First, recall that a computation $\gamma_0 \longrightarrow \gamma_1 \longrightarrow \ldots \longrightarrow \gamma_n \longrightarrow \ldots$ is fair if for each process enabled in some γ_i, there exists $j \geq i$ such that the process is active in γ_j.

Next, note that it is easy to verify from the operational semantics that whenever $\langle P, c\rangle \longrightarrow \langle P', c'\rangle$ and $c \in Con_0$, $c' \in Con_0$. So we know that for all $i \in \mathbb{N}$, $c_i \in Con_0$.

Suppose a process P^* is enabled in some $\langle P_i, e\rangle$. From Lemma 7, it follows that there is some $d \in Con_0$ such that $d \sqsubseteq e$ and the P^* is enabled in $\langle P_i, d\rangle$. Now since d is a compact element and $d \sqsubseteq e = \bigsqcup_i c_i$, there

exists j such that $d \sqsubseteq c_j$. Therefore P^* is enabled in $\langle P_j, c_j\rangle$, and since ζ_1 is a fair computation, this means that for some $k \geq j$ P^* is active in $\langle P_k, c_k\rangle \longrightarrow \langle P_{k+1}, c_{k+1}\rangle$. Furthermore, it it easy to see that P^* must also be active in the transition $\langle P_k, e\rangle \longrightarrow \langle P_{k+1}, e\rangle$. This proves that ζ_2 is a fair computation.

∎

Proof. Finally we can prove that $\mathcal{O}(P)$ is a closure operator. We must prove three properties: extensiveness, monotonicity and idempotence.

1. $c \sqsubseteq \mathcal{O}(P)(c)$.

   First, note that whenever $\langle P, c\rangle \longrightarrow \langle P', c'\rangle$, $c \sqsubseteq c'$. This is easy to see because the only rules that directly change the store are the T, and S, rules, and in this case the store goes from c to $c \sqcup c'$, and of course $c \sqsubseteq c \sqcup c'$. The rules PL, R, and E may change the store but only if the store in the hypothesis is similarly changed, so the stores in the hypothesis must also be increasing. The A rule does not change the store. Therefore, from the fact that a transition can only increase the store, it follows from the induction of $\mathcal{O}(P)$ that $c \sqsubseteq \mathcal{O}(P)(c)$.

2. If $c \sqsubseteq d$ then $\mathcal{O}(P)(c) \sqsubseteq \mathcal{O}(P)(d)$.

   We will show by structural induction on the derivation tree for the transition that if $\langle P, c\rangle \longrightarrow \langle P', c'\rangle$ and $c \sqsubseteq d$ then $\langle P, d\rangle \longrightarrow \langle P', d'\rangle$ and $c' \sqsubseteq d'$. It is clear that this fact implies the result.

   - If the transition follows from the T, A, PL or R rule, the proof is the same as in standard CCP.
   - If the transition follows from the S rule, then the process must be of the form $[P]_i$. If $c \sqsubseteq d$ then $c^i \sqsubseteq d^i$, so if $\langle P, c^i\rangle \longrightarrow \langle P', c'\rangle$ then $\langle P, d^i\rangle \longrightarrow \langle P', d'\rangle$ where $c' \sqsubseteq d'$ by the induction hypothesis. Then since $\mathfrak{s}_i$ is order-preserving by assumption, $\mathfrak{s}_i(c') \sqsubseteq \mathfrak{s}_i(d')$, and therefore $c \sqcup \mathfrak{s}_i(c') \sqsubseteq d \sqcup \mathfrak{s}_i(d')$.
   - If the transition follows from the E rule, then the process is of the form $[P]_i$, and if $\langle [P]_i, c\rangle \longrightarrow \langle [P]_i \parallel P', c'\rangle$ then we know

that $\langle P, c\rangle \longrightarrow \langle P', c'\rangle$. From the induction hypothesis, if $c \sqsubseteq d$ then $\langle P, d\rangle \longrightarrow \langle P', d'\rangle$ and $c' \sqsubseteq d'$. Of course this means that $\langle [P]_i, d\rangle \longrightarrow \langle [P]_i \parallel P', d'\rangle$, proving the desired property.

3. $\mathcal{O}(P)(c) = \mathcal{O}(P)(\mathcal{O}(P)(c))$.

   We prove this property for the case of a process with an infinite execution. The proof for a process with finite execution is similar.

   Assume that P has a fair computation (let $c = c_0$)

   $$\langle P, c_0\rangle \longrightarrow \langle P_1, c_1\rangle \longrightarrow \langle P_2, c_2\rangle \longrightarrow \ldots \longrightarrow \langle P_n, c_n\rangle \longrightarrow \ldots$$

   Let $e = \mathcal{O}(P)(c) = \bigsqcup_i c_i$.

   From Lemma 6.3.2, it follows that

   $$\langle P, e\rangle \longrightarrow \langle P_1, e\rangle \longrightarrow \langle P_2, e\rangle \longrightarrow \ldots \longrightarrow \langle P_n, e\rangle \longrightarrow \ldots$$

   From lemma 6.3.4 we know that this is a fair computation. Therefore $\mathcal{O}(P)(e) = e$. Since $e = \mathcal{O}(P)(c)$, this means that $\mathcal{O}(P)(c) = \mathcal{O}(P)(\mathcal{O}(P)(c))$, so $\mathcal{O}(P)$ is an idempotent function.

∎

Now we recall the definition of the fixed points of the observation function.

Definition 6.3.5. *For a process P, the set of fixed points of $\mathcal{O}(P)$ is defined as*

$$fix(\mathcal{O}(P)) = \{d \in Con \mid \mathcal{O}(P)(d) = d\}.$$

Proposition 6.3.6. *The observation function is defined by its set of fixed points:*

$$\mathcal{O}(P)(c) = \bigsqcap\{d \in Con \mid c \sqsubseteq d \text{ and } d \in fix(\mathcal{O}(P))\}.$$

This proposition is just an instance of Theorem 3.1.17.

Corollary 6.3.7. $\mathcal{O}(P) = \mathcal{O}(Q)$ *iff* $fix(\mathcal{O}(P)) = fix(\mathcal{O}(Q))$.

DX $[\![X]\!]_I = I(X)$
DP $[\![P \parallel Q]\!]_I = [\![P]\!]_I \cap [\![Q]\!]_I$
D0 $[\![0]\!]_I = \{d \mid d \in Con\}$
DT $[\![\mathbf{tell}(c)]\!]_I = \{d \mid c \sqsubseteq d\}$
DA $[\![\mathbf{ask}(c) \rightarrow P]\!]_I = \{d \mid c \sqsubseteq d \text{ and } d \in [\![P]\!]_I\} \cup \{d \mid c \not\sqsubseteq d\}$
DR $[\![\mu X.P]\!]_I = \bigcap\{S \subseteq \mathcal{P}(Con) \mid [\![P]\!]_{I[X:=S]} \subseteq S\}$
DS $[\![[P]_i]\!]_I = \{d \mid d^i \in [\![P]\!]_I\}$

Figure 6.1: Denotational Equations for SCCP. $I : Var \rightarrow \mathcal{P}(Con)$.

DX $[\![X]\!]_I = I(X)$
DP $[\![P \parallel Q]\!]_I = [\![P]\!]_I \cap [\![Q]\!]_I$
D0 $[\![0]\!]_I = \{d \mid d \in Con\}$
DT $[\![\mathbf{tell}(c)]\!]_I = \{d \mid c \sqsubseteq d\}$
DA $[\![\mathbf{ask}(c) \rightarrow P]\!]_I = \{d \mid c \sqsubseteq d \text{ and } d \in [\![P]\!]_I\} \cup \{d \mid c \not\sqsubseteq d\}$
DR $[\![\mu X.P]\!]_I = \bigcap\{S \subseteq \mathcal{P}(Con) \mid [\![P]\!]_{I[X:=S]} \subseteq S\}$
DE $[\![[P]_i]\!]_I = \{d \mid d^i \in [\![P]\!]_I\} \cap [\![P]\!]_I$

Figure 6.2: Denotational Equations for ECCP. $I : Var \rightarrow \mathcal{P}(Con)$.

We now give a compositional denotational semantics $[\![\, P \,]\!]$ that exactly captures the set of fixed points of $\mathcal{O}(P)$. More precisely, let I be an assignment function from Var, the set of process variables, to $\mathcal{P}(Con)$. Given a term T, $[\![\, T \,]\!]_I$ is meant to capture the fixed points of T under the assignment I. Notice that if T is a process P, i.e., a closed term, the assignment is irrelevant so we simply write $[\![\, P \,]\!]$. The denotation for processes in SCCP is given by the equations DX, D0, DT, DA, DP and DS in Table 6.1. The denotation for the processes in ECCP is given by the same rules except that the rule DS is replaced with the rule DE in Table 6.2.

The denotations of the basic operators are the same as in standard CCP [SJPR91] and are given by equations D0, DT, DA and DP. For example, DA says that the set of fixed points of **ask** $c \rightarrow P$ are those d that do not

entail c (these are fixed points because the **ask** process is blocked so the store can no longer change) or that if they do entail c then they must be fixed points of P. The denotation of a term X under I is $I(X)$ (see DX). The equation DR for $\mu X.P$ follows from the Knaster-Tarski theorem in the complete lattice $(\mathcal{P}(Con), \subseteq)$.

The denotation of $[P]_i$ in the spatial case is given by equation DS. It says that d is a fixed point for $[P]_i$ if $d^i \in [\![P]\!]$. Recall that d^i is i's view of d, so if $d^i \in [\![P]\!]$, then i's view of d is a fixed point for P. In the operational semantics, the S rule is the only applicable rule for this case. We can use Lemma 5.2.3, which says that $d = d \sqcup \mathfrak{s}_i(d^i)$, to prove that if d^i is a fixed point for P then d is a fixed point for $[P]_i$.

The denotation of $[P]_i$ in the epistemic case is given by DE instead of DS. It says that d is a fixed point for $[P]_i$ if $d^i \in [\![P]\!]$, as in the spatial case, and d is also a fixed point of P. The additional requirement follows from the operational semantics rule E which amounts to running $[P]_i$ in parallel with P.

The above observations suggest that $[\![P]\!] = \mathit{fix}(\mathcal{O}(P))$, which we will now prove formally.

Lemma 6.3.8. *For any P, $[\![P]\!] = \mathit{fix}(\mathcal{O}(P))$.*

Proof. We prove the proposition by structural induction on P.

- The proofs for the cases $P = \mathbf{0}$, $P = \mathbf{ask}\ (c)\ \rightarrow\ P'$, $P = \mathbf{tell}(c)$, and $P = P_1 \parallel P_2$ are the same as in traditional CCP, so we omit these proofs.

- In SCCP, for the $P = [P']_i$ case, first assume that $d \in [\![[P']_i]\!]$. Then $d^i \in [\![P']\!]$. So by the induction hypothesis, $\mathcal{O}(P')(d^i) = d^i$, and therefore by definition of $\mathcal{O}$, if $\langle P', d^i\rangle \longrightarrow \langle P'', d'\rangle$ then $d' = d^i$. So the derivation for the transition of $[P']_i$ must look like this:

$$\frac{\langle P', d^i\rangle \longrightarrow \langle P'', d^i\rangle}{\langle [P']_i\,, d\rangle \longrightarrow \langle [P'']_i\,, d \sqcup \mathfrak{s}_i(d^i)\rangle}$$

  Finally, in lemma 5.2.3 we showed that $d \sqcup \mathfrak{s}_i(d^i) = d$, so therefore $d \in \mathit{fix}(\mathcal{O}(P))$.

Now assume that $d \in \mathit{fix}(\mathcal{O}(P))$. If $\langle [P']_i, d\rangle$ has a transition available, this derivation must hold:

$$\frac{\langle P', d^i\rangle \longrightarrow \langle P'', d'\rangle}{\langle [P']_i, d\rangle \longrightarrow \langle [P'']_i, d \sqcup \mathfrak{s}_i(d')\rangle}$$

But since $d \in \mathit{fix}(\mathcal{O}(P))$, $d = d \sqcup \mathfrak{s}_i(d')$, which means that $\mathfrak{s}_i(d') \sqsubseteq d$. Recall that $d^i = \bigsqcup\{c \mid \mathfrak{s}_i(c) \sqsubseteq d\}$, meaning, therefore, that $d' \sqsubseteq d^i$. However, we know that if $\langle P', d^i\rangle \longrightarrow \langle P'', d'\rangle$ then $d^i \sqsubseteq d'$. So we conclude that $d^i = d'$, and since we considered any arbitrary transition available to $\langle P', d^i\rangle$, we conclude that $d^i \in \mathit{fix}(\mathcal{O}(P'))$. From the induction hypothesis, therefore, $d^i \in [\![P']\!]$, so by D4, $d \in [\![P]\!]$.

- In ECCP, for the $P = [P']_i$ case, first assume that $d \in [\![[P']_i]\!]$. Then $d^i \in [\![P']\!]$ and $d \in [\![P']\!]$. So by the induction hypothesis, $\mathcal{O}(P')(d^i) = d^i$ and $\mathcal{O}(P')(d) = d$, and therefore by definition of $\mathcal{O}$, if $\langle P', d^i\rangle \longrightarrow \langle P'', d'\rangle$ then $d' = d^i$, and if $\langle P', d\rangle \longrightarrow \langle P'', d'\rangle$ then $d' = d$. So there are two choices for the derivation of the transition: First,

$$\frac{\langle P', d\rangle \longrightarrow \langle P'', d\rangle}{\langle [P']_i, d\rangle \longrightarrow \langle [P']_i \parallel P'', d\rangle}.$$

In this case, we immediately have that $d \in \mathit{fix}(\mathcal{O}(P))$. The second possibility is

$$\frac{\langle P', d^i\rangle \longrightarrow \langle P'', d^i\rangle}{\langle [P']_i, d\rangle \longrightarrow \langle [P'']_i, d \sqcup \mathfrak{s}_i(d^i)\rangle}$$

As above, $d \sqcup \mathfrak{s}_i(d^i) = d$, so $d \in \mathit{fix}(\mathcal{O}(P))$.

Now assume that $d \in \mathit{fix}(\mathcal{O}(P))$. If $\langle [P']_i, d\rangle$ has a transition available, it may be of the form

$$\frac{\langle P', d^i\rangle \longrightarrow \langle P'', d'\rangle}{\langle [P']_i, d\rangle \longrightarrow \langle [P'']_i, d \sqcup \mathfrak{s}_i(d')\rangle}.$$

In this case, since $d \in \mathit{fix}(\mathcal{O}(P))$, $d = d \sqcup \mathfrak{s}_i(d')$, which means that $\mathfrak{s}_i(d') \sqsubseteq d$. As in the SCCP case, this means that $d^i = d'$, $d^i \in \mathit{fix}(\mathcal{O}(P'))$. From the induction hypothesis, therefore, $d^i \in [\![P']\!]$, but we must also prove that $d \in [\![P']\!]$. Because of the induction hypothesis, to do this it is sufficient to prove that $d \in \mathit{fix}(\mathcal{O}(P'))$. If $\langle P', d\rangle$ has

no transitions available, then $d \in \mathit{fix}(\mathcal{O}(P'))$ vacuously. On the other hand, if $\langle P', d\rangle \longrightarrow \langle P'', d'\rangle$ then we can use the E rule to conclude that

$$\frac{\langle P', d\rangle \longrightarrow \langle P'', d'\rangle}{\langle [P']_i, d\rangle \longrightarrow \langle [P']_i \parallel P'', d'\rangle}$$

but since we assumed that $d \in \mathit{fix}(\mathcal{O}(P))$, $d' = d$, and therefore $d \in \mathit{fix}(\mathcal{O}(P'))$ and $d \in [\![P']\!]$. Note that the above reasoning also holds if $[P']_i$ only has a transition available according to the E rule and no transition available according to the S rule, so in all cases, if $d \in \mathit{fix}(\mathcal{O}([P']_i))$ then $d^i \in [\![P']\!]$ and $d \in [\![P']\!]$, so therefore $d \in [\![[P']_i]\!]$.

∎

So now we can prove

From Corollary 6.3.7 we obtain a compositional characterization of observational equivalence, and thus from Theorem 6.2.11 also for barb equivalence.

Theorem 6.3.9. *$P \sim_o Q$ if and only if $[\![P]\!] = [\![Q]\!]$.*

This theorem follows directly from Lemma 6.3.8 and 3.1.17.

Seven

Future Work and Conclusions

In the first section of this chapter, we present some preliminary results about future work we hope to accomplish concerning the representation of common knowledge in our process calculi. Next, we provide a more in-depth discussion of other work related to ours, and finally our conclusions.

7.1 Compact Approximation of Space and Knowledge

In this section we present some preliminary, but we hope interesting, ideas about the expression of common knowledge (or global information) in our process calculi.

An important semantic property of global information or common knowledge $\mathfrak{g}_G(c)$ (Definition 4.1.6) in the underlying spatial constraint system is that it preserves the *continuity* of the space functions. Thus, one can verify that $\mathfrak{g}_G(\bigsqcup D) = \bigsqcup_{d \in D} \mathfrak{g}_G(d)$ for any directed set $D \subseteq Con$.

Theorem 7.1.1. *Let* $(Con, Con_0, \sqsubseteq, \sqcup, true, false, \mathfrak{s}_1, \ldots, \mathfrak{s}_n)$ *be an* n*-agent continuous and space-compact scs. Then for any* $G \subseteq \{1, \ldots, n\}$, $\mathfrak{g}_G(\cdot)$ *in Definition 4.1.6 is continuous:* $\mathfrak{g}_G(\bigsqcup D) = \bigsqcup_{d \in D} \mathfrak{g}_G(d)$ *for any directed set* $D \subseteq Con$.

In contrast $\mathfrak{g}_G(c)$ does not preserve the *compactness* of the space functions (Remark 4.2.2). This means that, although, the limit of infinite computation may produce $\mathfrak{g}_G(c)$, we cannot have a process that refers directly to $\mathfrak{g}_G(c)$ since processes can only ask and tell compact elements. The reason for this syntactic restriction is illustrated below:

Example 7.1.2. *Suppose we had a process* $P = \textbf{\textit{ask}}\ \mathfrak{g}_G(c) \rightarrow \textbf{\textit{tell}}(d)$ *asking whether group* G *has common knowledge of* c *and if so posting* d*. Note that* $\mathcal{O}(P)(true) = true$ *and* $\mathcal{O}(P)(\mathfrak{g}_G(c)) = \mathfrak{g}_G(c) \sqcup d$*. Now for* $Q = global(G, \textbf{\textit{tell}}(c))$ *we have* $\mathcal{O}(Q)(true) = \mathfrak{g}_G(c)$*. But one can verify that* $\mathcal{O}(P \parallel Q)(true) = \mathfrak{g}_G(c)$*, and thus* $\mathcal{O}(P \parallel Q)(\mathcal{O}(P \parallel Q)(true)) = \mathcal{O}(P \parallel Q)(\mathfrak{g}_G(c)) = \mathfrak{g}_G(c) \sqcup d$*. This would mean that the observation function is not idempotent, contradicting the fact that* $\mathcal{O}(P)$ *is a closure operator, a crucial property for full abstraction of our denotational semantics.*

Nevertheless, asking and telling information of the form $\mathfrak{g}_G(c)$ could be useful in certain protocols to state in one computational step, rather than computing as a limit, common knowledge or global information about certain states of affairs c (for example, mutual agreement). To address this issue we extend the underlying spatial constraint system with *compact elements* of the form $\mathfrak{a}_G(c)$ which can be thought of as (over-)approximations of $\mathfrak{g}_G(c)$. The approximation $\mathfrak{a}_G(c)$ can then be used in our processes to simulate the use of $\mathfrak{g}_G(c)$. We refer to $\mathfrak{a}_G(c)$ as a *announcement* of c for the group G to convey the meaning that $\mathfrak{g}_G(c)$ is attained in one step as in a public announcement. We can only define the announcements over a finite subset of compact elements S, since an infinite set would conflict with the continuity of $\mathfrak{a}_G(\cdot)$. We only consider announcements for the entire set of agents A (for arbitrary groups the construction follows easily). The above-mentioned extension of a spatial constraint system $\mathbf{C}^1$ into a spatial constraint system $\mathbf{C}^2(S)$ with announcement over S is given below:

Definition 7.1.3. *Let* $\boldsymbol{C}^1 = (Con^1, Con_0^1, \sqsubseteq_1, \mathfrak{s}_1^1, \ldots, \mathfrak{s}_n^1)$ *be a spatial constraint system over agents* $A = \{1, \ldots, n\}$*. For* $S \subseteq_{fin} Con_0^1$*, define lattice* $\boldsymbol{C}^2(S) = (Con^2, Con_0^2, \sqsubseteq_2, \mathfrak{s}_1^2, \ldots, \mathfrak{s}_n^2)$ *as follows. The set* Con^2 *is given by two rules:*

1. $Con^1 \subseteq Con^2$,

2. *For any finite nonempty indexing set* I, *if* $c_i \in S$ *for all* $i \in I$ *then* $\mathfrak{a}_A(\bigsqcup_{i\in I} c) \in Con^2$.

The ordering $\sqsubseteq_2$ *is given by the following rules:*

1. $\sqsubseteq_1 \subseteq \sqsubseteq_2$,

2. $d \sqsubseteq_2 \mathfrak{a}_A(\bigsqcup_{i\in I} c_i)$ *if* $d \in Con^1$ *and* $d \sqsubseteq_1 \mathfrak{g}_A(\bigsqcup_{i\in I} c_i)$, *and*

3. $\mathfrak{a}_A(\bigsqcup_{i\in I} c_i) \sqsubseteq_2 \mathfrak{a}_A(\bigsqcup_{j\in J} c_j)$ *if* $\mathfrak{g}_A(\bigsqcup_{i\in I} c_i) \sqsubseteq_1 \mathfrak{g}_A(\bigsqcup_{j\in J} c_j)$.

Furthermore, for all $i \in A$, *for any* $\mathfrak{a}_A(d) \in Con^2$, $\mathfrak{s}^2_i(\mathfrak{a}_A(d)) = \mathfrak{a}_A(d)$ *and for each* $e \in Con^1$, $\mathfrak{s}^2_i(e) = \mathfrak{s}^1_i(e)$.

The next theorem states the correctness of the above construction. Intuitively, the lattice $\mathbf{C}^2(S)$ above must be a spatial constraint system and the announcement of a certain fact in $c \in S$ must behave similarly to common knowledge or global information of the same fact.

Theorem 7.1.4. *Let* $\boldsymbol{C}^1 = (Con^1, Con^1_0, \sqsubseteq_1, \mathfrak{s}^1_1, ..., \mathfrak{s}^1_n)$ *be a continuous space-compact n-scs (n-ecs) and let* $S \subseteq_{fin} Con^1_0$. *Let*

$$\boldsymbol{C}^2(S) = (Con^2, Con^2_0, \sqsubseteq_2, \mathfrak{s}^2_1, ..., \mathfrak{s}^2_n)$$

as in Def. 7.1.3, then

1. $\boldsymbol{C}^2(S)$ *is a continuous, space-compact n-scs (n-ecs),*

2. $\forall \mathfrak{a}_A(c) \in Con^2$, $\mathfrak{a}_A(c) \in Con^2_0$, *and*

3. $\forall d \in Con^1$, $\forall \mathfrak{a}_A(c) \in Con^2$, $d \sqsubseteq_2 \mathfrak{a}_A(c)$ *iff* $d \sqsubseteq_1 \mathfrak{g}_{\mathcal{A}}(c)$.

7.2 Related Work

There is a huge volume of work on epistemic logic and its applications to distributed systems; [FHMV95] gives a good summary of the subject. This work is all aimed at analyzing distributed protocols using epistemic logic

as a reasoning tool. While it has been very influential in setting the stage for the present work it is not closely connected to the present proposal to put epistemic concepts into the programming formalism.

Epistemic logic for process calculi has been discussed in [CDK09, DMO07, HS04]. In [CDK09], an epistemic logic is presented for the applied calculus. While we find their approach to epistemic logic for the applied pi-calculus compelling, it is quite different from our work because their epistemic logic is defined outside of the process calculus, whereas our processes have epistemic (or spatial) logic terms within the constraint system, as well as knowledge or space constructions on the processes. Furthermore, their epistemic logic only concerns the agent "intruder." While this is satisfactory for the problems they are considering, our calculus enjoys the ability to deal with an arbitrary finite set of agents. Furthermore, we consider both general modal logic, with the modalities interpreted as "spaces," and epistemic logic, and we believe our constraint system could easily be adapted to other specific modal logics, particularly temporal logic. The paper of Dechesne, Mousavi and Orzan [DMO07] also takes an interesting approach to combining epistemic logic and process calculus. Again, their processes provide a model for their logic, rather than having epistemic operators within the process calculus. Furthermore, they deal with a specific temporal epistemic logic which is different from either of the logics we consider. [HS04] combines epistemic logic and process calculi using function views representing partial information. Like ours, this paper presents a domain-theoretical characterization of knowledge in process calculi. Again, however, this paper uses the processes as models for the logic, rather than including modal constructs in the process calculus.

In all of these works, the epistemic logic is defined outside of the process calculus, with the processes as models for the logic, whereas our processes have epistemic (or spatial) logic terms within the constraint system, as well as knowledge or space constructions on the processes.

The issue of extending CCP to provide for distributed information has been previously addressed in [Rét98]. In [Rét98] processes can send constraints using communication channels much like in the π-calculus. This in-

duces a distribution of information among the processes in the system. The extended processes, however, do not have the traditional (closure-operator) denotational semantics of CCP which is one of the sources of its elegance and simplicity. By using a logical approach to the problem of common and distributed information, rather than an operational one based on channel communication, we have a framework faithful to the declarative nature of CCP.

Another closely related work is the Ambient calculus [CG00], an important calculus for spatial mobility. Ambient allows the specification of processes that can move in and out within their spatial hierarchy. It does not, however, address posting and querying epistemic information within a spatial distribution of processes. Adding Ambient-like mobility to our calculi is a natural research direction.

One very interesting approach related to ours in spirit – but not in conception or details – is the spatial logic of Caires and Cardelli [CC03, CC04]. In this work they also take spatial location as the fundamental concept and develop modalities that reflect locativity. Rather than using modal logic, they use the name quantifier which has been actively studied in the theory of freshness of names in programming languages. Their language is better adapted to the calculi for mobility where names play a fundamental role. In effect, the concept of freshness of a name is exploited to control the flow of information. It would be interesting to see how a name quantified SCS would look and to study the relationship with the Caires-Cardelli framework.

Finally, the process calculi in [BJPV09, BM07, FRS01] provide for the use of assertions within π-like processes. They are not concerned with spatial distribution of information and knowledge. These frameworks are very generic and offer several reasoning techniques. Therefore, it would be interesting to see how the ideas here developed can be adapted to them.

7.3 Future Work

One important item that we have already begun is the development of the theory from lower level concepts. In the same way that domains and complete algebraic lattices arise from information systems [Sco82] we have developed modal information systems where the various axioms for constraint systems arise from a structure closer to the logic.

One natural extension of these ideas is to develop the combination of epistemic and mobile constructs. It would be exciting if this would lead to a new epistemic perspective on the spatial logics of Caires and Cardelli. It would also be important to demonstrate that these epistemic concepts extend to calculi beyond CCP.

There are a number of application areas that are important. One immediate task is to explore how well one can capture distributed systems protocols in the ECCP language. In the late 1980s Panangaden and Taylor [PT92] developed *concurrent common knowledge* to capture agreement in asynchronous systems and showed how various protocols, for example the Chandy-Lamport checkpointing algorithm, were effectively protocols for attaining concurrent common knowledge. There are many examples in this vein in the literature which we need to explore to see whether "putting epistemic concepts in the hands of the programmer" leads to more perspicuous presentations of known algorithms or indeed new algorithms.

Finally, there are a number of theoretical ideas to explore. One of the founding fathers of topos theory [Joh02], Lawvere, stated that the point of the geometric logic developed by topos theorists is to capture a modality of the form "it is locally the case that.." Remarkably, these ideas have been present in sheaf theory and categorical semantics since the 1970s but they have not had a direct impact on programming language semantics. Considering the importance in computer science of local information, local computations, mobility, and the flow of information between locations, unravelling this connection is certain to enrich our understanding of the subject, and to provide important theoretical tools for modelling these concepts.

7.4 Conclusion

We have presented constraint systems and process calculi for working with spatially distributed or epistemic information and computation. We believe that our process calculi are relevant to modern problems in computation, because systems with many users and large amounts of information with complicated structures of access to the information are becoming more and more common. In the next part of the thesis, we will see another approach to understanding agents' epistemic states in the context of changing systems.

Part II

How Knowledge Evolves: Epistemic Logic for Labelled Transition Systems

Introduction

Concurrency theory has been built upon the implicit assumption of omniscience of all the agents involved, but for many purposes – notably security applications – it is crucial to incorporate and reason about what agents "know" or do not know. Tracking the flow of information is the essence of analyses of security protocols. Equally crucial is the idea that different participants may have different views of the system and hence know different things. The purpose of this part of the thesis is to meld traditional concurrency concepts with epistemic concepts and define a logic with both dynamic and epistemic modalities.

In the previous part of this thesis, we presented a way to incorporate epistemic modalities directly into the process calculus. In the dynamic processes we looked at, the epistemic information was updated through actions taken by these processes. In this section, however, we present a different way of representing knowledge in dynamic multi-agent systems, by introducing a dynamic epistemic logic of transition systems. Whereas previously we focused on systems with asynchronous agent communication and actions that mainly added information to the store, in the following chapters our models do not include explicit communication between agents. However, an essential and somewhat novel aspect of our systems is that we allow *fact changing actions*. This means that our actions do not just reveal information but may also change the state of the system. For example, a system may start out with a certain fact true, like "it is not raining," and then after an action this fact may change, and "it is raining" will hold. We model these types of situation by taking labelled transition systems as our basic structures, and consider the problem of adding Kripke-style

agent equivalence relations to the structures. Our logic is particularly well adapted to analyzing the effects of the actions in the labelled transition systems because it is closely tied not only to epistemic logic but also to Hennessy-Milner logic, the essential logic of labelled transition systems.

Thus, in this part of the thesis we are again investigating epistemic concepts in a concurrent setting. Epistemic logic has been a major theme within distributed systems ever since the groundbreaking paper of Halpern and Moses [HM84], but has been strangely slow to influence concurrency theory. A few investigations have appeared but, as far as we know, there has not been a thorough integration of epistemic concepts with the traditional theory of labelled transition systems. Typically, one sees a multimodal logic closely tied to the syntax of some particular process calculus with reasoning principles that are not proven complete in any sense [CDK09]. Such logics are interesting and useful, but their close ties to a particular process formalism obscure the general principles. Another closely related strand is, of course, dynamic epistemic logic [vDvdHK08] which, as the name suggests, is all about how knowledge evolves. However, the bulk of this work is about actions that communicate information, perhaps through messages or announcements, rather than about general transitions that could change basic facts about the state. A few papers indeed deal with so-called fact-changing actions but, as far as we know, the theory is still geared toward communication actions. Our goals are to develop the theory for a suitably general class of labelled transition systems and to formulate axioms that are provably complete with respect to this class of models. We provide more detailed comparisons with related work in a later section, after the presentation of our framework.

The standard route to modelling epistemic concepts is to use Kripke models: these are sets of states equipped with indistinguishability (equivalence) relations [FHMV95]. We will equip the states with a labelled transition system structure as well and impose coherence conditions between the two kinds of relations. The resulting modal logic is a blend of Hennessy-Milner logic, epistemic logic and temporal modalities. The essential point is that one can reason about how knowledge changes as transitions occur.

There are many variations that one could contemplate and the particular formalism that we have developed is geared toward representing the unfolding of a labelled transition system through time, taking into account different agents' contrasting views of the labelled transition system.

The background material on labelled transition systems and Hennessy-Milner logic has already been presented in Chapter 2. This part of the thesis is organized as follows. In Chapter 8 we define the class of transition systems that we work with; they are called *history labelled transition systems* and are unfoldings of the usual labelled transition systems, with the addition of equivalence relations on states. In Chapter 9 we define the logic and its semantics. In Chapter 10 we prove the weak completeness theorem. There is an easy argument, which we present in Chapter 10, that shows that a strong completeness theorem is not possible. The final sections discuss related work and conclusions.

Eight

Histories

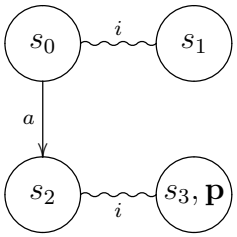
The main contribution of this part of the thesis is to study how an agent's knowledge changes as transitions occur in a labelled transition system. The basic picture is that even when an agent knows the overall structure of a labelled transition system, they have a limited view of the current state of the system. This uncertainty is modelled by an equivalence relation on the states of the system just as in a Kripke structure. The agent does not choose the actions to perform but can see which action has happened and tries to deduce from this where it is. Our temporal-epistemic logic will be designed to handle this type of reasoning.

The semantics of the formulas will be given in terms of histories or runs, as with the semantics of Halpern and Moses [HM84, HM90], but we view the runs as coming from the executions of a labelled transition system (LTS). In fact, we will view the set of runs as forming a labelled transition system in its own right. This will give a "branching-time" logic rather than a linear-time logic. We will use the box and diamond modalities of Hennessy-Milner logic [HM85] rather than the "always" and "eventually" modalities of temporal logic. In this chapter, we motivate the need for this particular combination of modalities.

8.1 Labelled transition systems with agents

The basic setup for a purely epistemic (static) logic is a set of states with equivalence relations, one for each agent. If we wish to incorporate this into a given labelled transition system the natural step is to define equivalence relations on the states of the labelled transition system. If one does this naïvely one gets situations where one cannot say what an agent has learned from its history.

Example 8.1.1. *Consider the following simple labelled transition system:*



where the wiggly line refers to the indistinguishability equivalence relation of agent i and the proposition $\mathbf{p}$ *holds in the state s_3 and in no other state. The agent i in state s_0 cannot tell whether he is in s_0 or in s_1. Similarly, in s_2 he cannot tell whether he is in s_2 or in s_3. However, if the agent is in s_0 and then observes an a action then he "knows" he must have been in s_0 and further, that he is in s_2 now. No purely state-based semantics can say this. It is only because the agent "remembers" how he got there that one can say anything. Thus, a purely state-based semantics is not adequate for even the simplest statements about evolving knowledge for agents with memory and basic reasoning abilities.*

The basic paradigm that we have in mind is that the agent is observing a transition system: the agent can see the actions and can remember the actions but cannot *control* the actions nor see *which actions are available at a given state*. The extent to which an agent can "see" the state is what the indistinguishability relation spells out.

In order to give the semantics of the epistemic modalities we need to extend the equivalence relation from states to histories. We formalize the

definition of labelled transition systems with agents, and the notion of histories and the extended equivalence relation as follows.

Definition 8.1.2 (Labelled transition system with agents)**.** *A* labelled transition system *is a set of states, S, a finite set of actions $\mathcal{A}$, and, for every $a \in \mathcal{A}$, a binary relation, written $\xrightarrow{a}$, on the states. We write $s \xrightarrow{a} s'$ instead of $(s, s') \in \xrightarrow{a}$. In addition, there is a finite set of* agents*, denoted by letters like $i, j, \ldots$. For each agent i there is an equivalence relation, written $\sim_i$ defined on S.*

The relation $\xrightarrow{a}$ can be nondeterministic and does not have to be image-finite [1]. We also assume that all actions are visible, that is, there are no hidden actions (commonly denoted by τ).

Definition 8.1.3 (History)**.** *A* history *is a finite alternating sequence of states and actions*

$$s_0 a_1 s_1 a_2 s_2 \ldots a_n s_n,$$

where, for each $l \in \{0, \ldots, n-1\}$, $s_l \xrightarrow{a_{l+1}} s_{l+1}$.

Given a pair of histories, an agent can tell immediately that they are not the same if they do not have exactly the same sequence of actions. In order to say this it will be convenient to define the notation $\mathsf{act}(h)$ to mean the action sequence extracted from the history h; it has an evident inductive definition. Given a history h, we write $h[n]$ for the n^{th} state in h. Thus if $h = s_0 a_1 s_1 a_2 s_2 a_3 s_3$, $\mathsf{act}(h) = a_1 a_2 a_3$ and $h[0] = s_0$ while $h[2] = s_2$. We write $|h|$ for the length of the sequence of states in h.

Definition 8.1.4 (History Indistinguishability)**.** *We say that the histories h_1 and h_2 are* indistinguishable by agent $\mathbf{i}$*, written $h_1 \sim_i h_2$, if:*

1. $\mathsf{act}(h_1) = \mathsf{act}(h_2)$ *and*
2. *for all $0 \leq n \leq |h_1| (= |h_2|)$, $h_1[n] \sim_i h_2[n]$.*

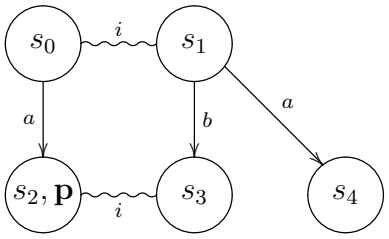
[1] "Image finite" means that for a given s and a the set $\{s' | s \xrightarrow{a} s'\}$ is finite.

The use of the same notation for indistinguishability of states and histories should not occasion anxiety for the reader as the context will disambiguate which we mean; this usage is meant to emphasize the tight connection between the concepts.

It is useful to have both past and future modalities. We will define the syntax precisely in the next chapter, for the moment we note that $\langle - \rangle$ means one step in the past and $\langle + \rangle_a$ means *possibly* after an a-step into the future (we will see later why the future operator is concerned with possibility while the past operator is not). Consider the labelled transition system we have used for our example above. Suppose we introduce the proposition $@s$ to mean "at the state s" then we want to be able to say things like $s_0 a s_2 \models K_i \langle - \rangle @ s_0$. Note that we cannot say $s_0 \models K_i @ s_0$, so we need the past operator to express the idea that agent i learns where he was in the past, or, in general, learns that a fact used to be true. Note that, for this example, $s_0 a s_2 \models \langle - \rangle K_i @ s_0$ does not hold, even though $s_0 a s_2 \models K_i \langle - \rangle @ s_0$ does.

Note that every history has a beginning and every state has a finite number of predecessors: in short the prefix order on histories is well founded. This will cause most of the difficulties in the completeness proof.

Example 8.1.5. *Why do we need the Hennessy-Milner like modalities indexed by actions? Consider the following simple labelled transition system:*



which is like the previous example except for the addition of the extra state and transitions and the fact that $\mathbf{p}$ *is true in* s_2 *instead of* s_3*. We would like to be able to say* $s_0 \models \langle + \rangle_a K_i \mathbf{p}$*. Note that* s_4 *can be distinguished by* i *from any other state.*

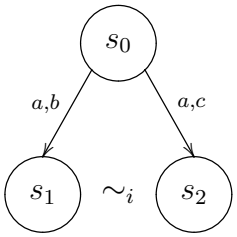
The logic, though its semantics is given in terms of runs, is actually a branching time logic. It is applied to a very specific type of transition system

that arises as the set of histories of general labelled transition system. The "states" are histories and the transitions are of the form

$$s_0 a_1 s_1 \dots a_n s_n \xrightarrow{a} s_0 a_1 s_1 \dots a_n s_n a s$$

whenever $s_n \xrightarrow{a} s$ is a transition of the underlying labelled transition system. The key features of these labelled transition systems of histories are a well-foundedness property for the backward transitions, determinacy for the backward transitions and a few other properties.[2] In the course of the completeness proof we will spell out these properties and then proceed with the axiomatization and completeness theorem.

Example 8.1.6. *Here is an example about why the identity of actions is important.*

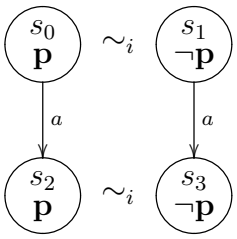


If this system starts out in s_0 and an a action occurs, then agent i will not know which state the system is in, because s_1 and s_2 are equivalent for the agent. But if the system does a b action, then the agent knows it is in s_1 because he observes the b action and knows the overall structure of the system, so he realizes that s_1 is the only state that a b action can lead to. Similarly, if the system does a c action, then the agent knows that the system is in s_2. ■

Example 8.1.7. *This example shows why we want to be able to combine epistemic modalities and (past or future) temporal modalities. Here p rep-*

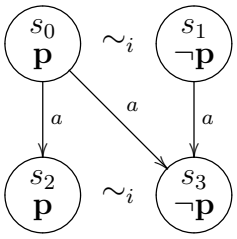
[2] In fact, such transition systems arise naturally as unfoldings of general labelled transition systems.

resents some proposition.



If the system starts out in s_0 or s_1, then after an a action, the agent does not know whether p is true, but he does know that if p is true now, then it must have been true in the first state, and if p is false now, it must have been false in the first state. ■

Example 8.1.8.



If this system starts out in s_0 or s_1 and then an a action occurs, then after the action, the agent does not know whether p is true, but he knows that if p is true now, then it was true in the start state. But he also knows that if p is not true now, then p may or may not have been true in the start state. ■

8.2 History Systems

First we will explain how to translate any LTS with equivalence classes into an equivalent history LTS: an LTS with designated starting states, where the entire history of any run starting from a starting state is determined by its current state.

Definition 8.2.1 (Unfolding)**.** *Given the LTS $(S_0, \mathcal{A}, \mathcal{I}, \underset{0}{\longrightarrow}, \sim^0)$, where S_0 is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{I}$ the set of agents, $\underset{0}{\longrightarrow} \subseteq S_0 \times \mathcal{A} \times S_0$ is the transition relation and $\sim^0 \subseteq S_0 \times \mathcal{I} \times S_0$ is the indistinguishability*

relation, inductively construct the unfolding $(S_1, \mathcal{A}, \mathcal{I}, \xrightarrow[+]{}, \xrightarrow[-]{}, \xrightarrow[+]{*}, \xrightarrow[-]{*}, \sim^1)$*, where* $\xrightarrow[+]{} \subseteq S_1 \times \mathcal{A} \times S_1$*,* $\xrightarrow[-]{} \subseteq S_1 \times \mathcal{A} \times S_1$*,* $\xrightarrow[+]{*} \subseteq S_1 \times S_1$ *and* $\xrightarrow[-]{*} \subseteq S_1 \times S_1$*, as follows:*

1. *If* $s \in S_0$ *then* $s \in S_1$*.*
2. *If* $s_0.a_1.s_1.a_2...s_n \in S_1$ *and* $s_n \xrightarrow[0]{a} s$ *then* $s_0.a_1...s_n.a.s \in S_1$ *and* $s_0.a_1...s_n \xrightarrow[+]{a} s_0.a_1...s_n.a.s$*.*
3. *If* $s_0.a_1...s_n$*,* $s_0.a_1...s_n.a.s \in S_1$ *then* $s_0.a_1...s_n.a.s \xrightarrow[-]{a} s_0.a_1...s_n$*.*
4. *If* $s_0.a_1...s_n \in S_1$ *then* $s_0.a_1...s_n \xrightarrow[+]{*} s_0.a_1...s_n$*.*
5. *If* $s_0.a_1...s_n$*,* $s_0.a_1...s_n.a_{n+1}...a.s \in S_0$ *then* $s_0.a_1...s_n \xrightarrow[+]{*} s_0.a_1...s_n.a_{n+1}...a.s$*.*
6. *If* $s_0.a_1...s_n \in S_1$ *then* $s_0.a_1...s_n \xrightarrow[-]{*} s_0.a_1...s_n$*.*
7. *If* $s_0.a_1...s_n$*,* $s_0.a_1...s_n.a_{n+1}...a.s \in S_0$ *then* $s_0.a_1...s_n.a_{n+1}...a.s \xrightarrow[-]{*} s_0.a_1...s_n$*.*
8. *If* $s, t \in S_0$ *and* $s \sim_i^0 t$ *then* $s \sim_i^1 t$*.*
9. *If* $s, t \in S_1$ *and* $s \sim_i^1 t$ *and* $s \xrightarrow[+]{a} s.a.s'$ *and* $t \xrightarrow[+]{a} t.a.t'$ *and* $s' \sim_i^0 t'$ *then* $s.a.s' \sim_i^1 t.a.t'$*.*

Definition 8.2.2 (History-LTS)**.** *An LTS with agent equivalence classes and with transition relations* $\xrightarrow[+]{} \subseteq S_1 \times \mathcal{A} \times S_1$*,* $\xrightarrow[-]{} \subseteq S_1 \times \mathcal{A} \times S_1$*,* $\xrightarrow[+]{*} \subseteq S_1 \times S_1$ *and* $\xrightarrow[-]{*} \subseteq S_1 \times S_1$ *is called a* history-LTS *if it satisfies the following properties:*

1. *Forward and backward transitions are converse:* $s \xrightarrow[+]{a} t$ *iff* $t \xrightarrow[-]{a} s$*.*
2. *There is only one way to reach each state: if* $s \xrightarrow[+]{a} t$ *then for all states* s' *and all actions* b*, if* $s' \xrightarrow[+]{b} t$ *then* $s = s'$ *and* $a = b$*.*

3. *If we let* $\underset{+}{\longrightarrow} = \bigcup_{a \in \mathcal{A}} \underset{+}{\overset{a}{\longrightarrow}}$, *then* $\underset{+}{\overset{*}{\longrightarrow}}$ *is the transitive reflexive closure of* $\underset{+}{\longrightarrow}$.

4. *If we let* $\underset{-}{\longrightarrow} = \bigcup_{a \in \mathcal{A}} \underset{-}{\overset{a}{\longrightarrow}}$, *then* $\underset{-}{\overset{*}{\longrightarrow}}$ *is the transitive reflexive closure of* $\underset{-}{\longrightarrow}$.

5. *There are no infinite backward paths: it is impossible to have an infinite chain* $s_0 \underset{-}{\longrightarrow} s_1 \underset{-}{\longrightarrow} \ldots \underset{-}{\longrightarrow} s_n \underset{-}{\longrightarrow} \ldots$.

6. $\sim_i$ *is transitive, reflexive and symmetric for each agent* i.

7. *If* $s_1 \sim_i t_1$ *and there exists a state* s_0 *and an action* a *such that* $s_0 \underset{+}{\overset{a}{\longrightarrow}} s_1$ *then there exists a state* t_0 *such that* $t_0 \underset{+}{\overset{a}{\longrightarrow}} t_1$ *and* $s_0 \sim_i t_0$.

These properties capture the idea that a history LTS is exactly what we get when we unfold the paths of an LTS with agent equivalence relations; a formal proof is straightforward. At each stage there is possible future branching but the past is determined in a particular history. Thus the past modalities are like LTL modalities but not the future modalities. The starred modalities give one the power of "always" and "eventually" operators in temporal logics. A history is assumed to have a starting point so it must be well-founded.

Nine

The Logic and its Semantics

In this chapter we present the logic for history LTS's. It allows us to discuss what is true at a certain state, what was true in the past, what agents know at at the current state, and what may or must be true in the future.

9.1 Syntax and Models

We assume a finite set of agents $\mathcal{I}$, a finite set of actions $\mathcal{A}$, and a countable set of propositions Q. In the following definition, $a \in \mathcal{A}$, $i \in \mathcal{I}$, and $q \in Q$.

Definition 9.1.1 (Syntax)**.**

$$\phi \ := \ \top \mid q \mid \langle + \rangle_a \phi \mid \langle - \rangle_a \phi \mid \langle + \rangle_* \phi \mid \langle - \rangle_* \phi \mid K_i \phi \mid \neg\phi \mid \phi \wedge \phi$$

As usual, we assume the boolean constants $\bot = p \wedge \neg p$ and $\top = \neg\bot$ and the boolean operators $\Rightarrow, \vee, \Longleftrightarrow$. In addition we define

$$\begin{aligned}
[-]_a\phi &= \neg\langle - \rangle_a \neg\phi & [+]_a\phi &= \neg\langle + \rangle_a \neg\phi, \\
[-]^*\phi &= \neg\langle - \rangle^* \neg\phi, & [+]^*\phi &= \neg\langle + \rangle^* \neg\phi, \\
\langle - \rangle\phi &= \bigvee_{a \in A} \langle - \rangle_a \phi, & \langle + \rangle\phi &= \bigvee_{a \in A} \langle + \rangle_a \phi, \\
[-]\phi &= \neg\langle - \rangle \neg\phi, & [+]\phi &= \neg\langle + \rangle \neg\phi.
\end{aligned}$$

In order to define the semantics we consider the (oriented) labeled graphs over $\mathcal{A}$. These capture sets of histories as we defined them in the previous

chapter. The nodes of the graph are states and the transitions are labelled by actions in $\mathcal{A}$. A path through the graph is a history.

If $G = (S, \xrightarrow{a})_{a\in\mathcal{A}}$ is a labelled graph, we denote by $\Rightarrow$ the relation $\bigcup_{a\in A} \xrightarrow{a}$ and by $\Rightarrow^*$ the reflexive-transitive closures of $\Rightarrow$ respectively.

Definition 9.1.2 (Labelled forest)**.** *A* labelled forest over $\mathcal{A}$ *is a labelled graph* $G = (S, \xrightarrow{a})_{a\in\mathcal{A}}$ *such that*

1. *for arbitrary* $s, s', s'' \in S$, $s' \Rightarrow s$ *and* $s'' \Rightarrow s$ *implies* $s' = s''$*;*
2. *there exists no infinite sequence* $s_0, s_1, .., s_k, .. \in S$ *such that* $s_{i+1} \Rightarrow s_i$ *for each* $i \in \mathbb{N}$*; i.e. it is well-founded to the past.*

The *support* of a forest $\mathcal{F}$, denoted by $supp(\mathcal{F})$, is the set of its nodes. Give a labelled forest $\mathcal{F}$, we say that an equivalence relation $\approx \subseteq supp(\mathcal{F}) \times supp(\mathcal{F})$ *reflects the branching structure* if whenever $s \approx t$, the existence of a transition $s' \xrightarrow{a} s$ implies the existence of $t' \in supp(\mathcal{F})$ such that $t' \xrightarrow{a} t$ and $s' \approx t'$. Notice that this is a *backward* bisimulation property; it is a backward preservation property.

Definition 9.1.3 (Epistemic Frame)**.** *Given a set* $\mathcal{I}$ *(of* agents*),* an epistemic frame *is a tuple* $\mathcal{E} = (\mathcal{F}, (\approx_i)_{i\in\mathcal{I}})$*, where* $\mathcal{F}$ *is a labelled forest over* $\mathcal{A}$ *and* $(\approx_i)_{i\in\mathcal{I}}$ *is an indexed set of equivalence relations on* $supp(\mathcal{F})$ *such that for each* $i \in \mathcal{I}$*,* $\approx_i$ *preserves the branching structure.*

We call the relation $\approx_i$ the *indistinguishability* relation of agent $i \in \mathcal{I}$. Observe that an epistemic frame defines a unique history-LTS and a history-LTS is supported by a unique epistemic frame.

9.2 Semantics

In the following definition we write s, t, r with or without subscripts for states, p and variants for propositions, ϕ, ψ for formulas and a for actions and i for agents.

Definition 9.2.1 (Semantics)**.** *The semantics is defined for an epistemic frame* $\mathcal{E} = (\mathcal{F}, (\approx_i)_{i\in\mathcal{I}})$*, a state* $s \in supp(\mathcal{F})$ *and an interpretation function* $Prop : supp(\mathcal{F}) \Rightarrow 2^{\mathcal{P}}$*, as follows.*

$s \models \top$ *for all* s.

$s \models p$ *if* $p \in Prop(s)$.

$s \models \langle + \rangle_a \phi$ *if there exists a state* t *such that* $s \xrightarrow{a} t$ *and* $t \models \phi$.

$s \models \langle - \rangle_a \phi$ *if there exists a state* r *such that* $r \xrightarrow{a} s$ *and* $r \models \phi$.

$s \models \langle + \rangle_* \phi$ *if there exist* $s_1, ..., s_n \in S$ *and* $a_1, ..., a_n \in \mathcal{A}$ *such that*
$s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} ... \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n$ *and* $s_n \models \phi$.

$s \models \langle - \rangle_* \phi$ *if there exist* $s_0, ..., s_{n-1} \in S$ *and* $a_1, ..., a_n \in \mathcal{A}$ *such that*
$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} ... \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s$ *and* $s_0 \models \phi$.

$s \models K_i \phi$ *if for all* t *such that* $s \approx_i t$, $t \models \phi$.

$s \models \neg\phi$ *if it is not the case that* $s \models \phi$.

$s \models \phi_1 \wedge \phi_2$ *if* $s \models \phi_1$ *and* $s \models \phi_2$.

Now we have defined our basic operators. For convenience, we also define other operators as shorthand for certain combinations of these basic operators:

$$
\begin{aligned}
\langle + \rangle \phi &:= \bigvee_{a\in\mathcal{A}} \langle + \rangle_a \phi \\
\langle - \rangle \phi &:= \bigvee_{a\in\mathcal{A}} \langle - \rangle_a \phi \\
[+]_a \phi &:= \neg\langle + \rangle_a \neg\phi \\
[-]_a \phi &:= \neg\langle - \rangle_a \neg\phi \\
[+] \phi &:= \bigwedge_{a\in\mathcal{A}} [+]_a \phi \\
[-] \phi &:= \bigwedge_{a\in\mathcal{A}} [-]_a \phi \\
[+]_* \phi &:= \neg\langle + \rangle_* \neg\phi \\
[-]_* \phi &:= \neg\langle - \rangle_* \neg\phi
\end{aligned}
$$

$$L_i\phi \;:=\; \neg K_i \neg \phi$$

Note that $[+]\phi = \neg\langle + \rangle \neg\phi$ and $[-]\phi = \neg\langle - \rangle \neg\phi$. The semantics of these derived operators are:

$s \models \bot$ never.
$s \models [+]_a\phi$ iff for any $t \in supp(\mathcal{F})$ s.t. $s \xrightarrow{a} t$, $t \models \phi$,
$s \models [-]_a\phi$ iff for any $t \in supp(\mathcal{F})$ s.t. $t \xrightarrow{a} s$, $t \models \phi$,
$s \models [+]^*\phi$ iff for any $t \in supp(\mathcal{F})$ s.t. $s \Rightarrow^* t$, $t \models \phi$,
$s \models [-]^*\phi$ iff for any $t \in supp(\mathcal{F})$ s.t. $t \Rightarrow^* s$, $t \models \phi$.

If we have an epistemic frame $\mathcal{E}$, a *valuation* is a map $\rho : supp(\mathcal{F}) \Rightarrow 2^{\mathcal{P}}$ which provides an interpretation of the propositions in the states of $\mathcal{E}$. If a formula ϕ is true in a given epistemic frame $\mathcal{E}$ and state s with a valuation ρ we write $\mathcal{E}, s, \rho \models \phi$ and we say that $(\mathcal{E}, s, \rho)$ is a model of ϕ. In this case we say that ϕ is *satisfiable*. Given an arbitrary $\phi \in \mathcal{L}$, if for any epistemic frame $\mathcal{E} = (\mathcal{F}, (\approx_i)_{i\in\mathcal{I}})$, any state $s \in supp(\mathcal{F})$ and any valuation ρ, $\mathcal{E}, s, \rho \models \phi$ we say that ϕ is *valid* and write $\models \phi$. We also write $\mathcal{E}, s, \rho \models \Phi$, where Φ is a set of formulas if it models every formula in the set Φ. We write $\Gamma \models \phi$ if any model of Γ is a model of ϕ.

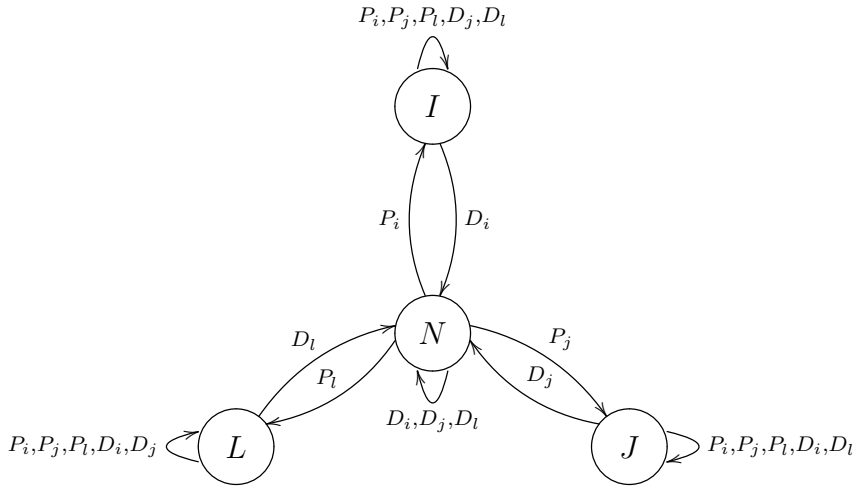
9.3 An example

Example 9.3.1. *Here is a more complicated example with multiple agents which we describe as an illustration of our logic.*

The situation is as follows: There are three agents, one diamond, and a bag. The diamond can either be held by one of the agents or it can be in the bag. Each agent can perform two actions: reach into the bag and take the diamond if it is there*, and drop the diamond into the bag, or pretend to drop it. After dropping or pretending to drop the diamond, the agent shows the other agents that his hands are empty, so it is impossible to keep the diamond while pretending to drop it. On the other hand, if the agent does not have the diamond, he can still pretend to drop it in the bag. If the agent*

reaches into the bag to take the diamond, he will take it if it is there, and will not take it if it is not there.

Here is the transition system:



The agents are i, j, and l. In state N, no one has the diamond, and in states I, J, and L, agents i, j, and l respectively have the diamond. Action P_i represents agent i picking up or pretending to pick up the diamond and action D_i represents agent i dropping or pretending to drop the diamond.

The equivalence classes are as follows:

$N \sim_i J \sim_i L$

$N \sim_j I \sim_j L$

$N \sim_l I \sim_l J$.

We use as propositions $@I$, $@J$, $@L$ and $@N$; each proposition is true only in the corresponding state and in each state only the corresponding proposition is true. For example, the only proposition true in state I is $@I$. We write Prop *for this set of 4 propositions. Now we consider the formulas*

$$\phi_1 \quad = \quad \bigwedge_{X \in \mathsf{Prop}} X \Rightarrow K_l X$$

$$\phi_2 \quad = \quad \langle - \rangle_{P_l} @N$$

$$\phi_3 \quad = \quad \bigvee_{X \in \mathsf{Prop}} K_l X$$

The first formula says that if any of the propositions are true then l knows it: in short l knows where the diamond is. Of course this formula is not universally true, it might or might not be true depending on the situation. The second formula is true for a history where the immediately preceding action is P_l (l picks up the diamond) and in the immediately preceding state nobody had the diamond (i.e. it was in the bag). In other words ϕ_2 describes the situation where the diamond was in the bag and l has just picked it up. The formula ϕ_3 says whatever the state happens to be, l knows it. Here are two formulas that are true in every state of the unfolded labelled transition system (the history LTS):

$$\phi_2 \Rightarrow [+]^* \phi_1 \text{ and } \phi_3 \Rightarrow [+]^* \phi_1.$$

The first is true because l has picked up the diamond and can now track its movements precisely for all future moves since all actions are visible to him. The second statement is slightly more general, it says that once l knows where the diamond is he can track its future exactly.

Here is another example of reasoning within this system. We define ϕ_4 to be like ϕ_1 except that we have K_i instead of K_l and ϕ_5 is like ϕ_1 except that K_j replaces K_l. Now we can conclude that the following formula is true in every state

$$\langle - \rangle_{D_i} \langle - \rangle_{D_j} \langle - \rangle_{D_l} \Rightarrow [+]^* (\phi_1 \wedge \phi_4 \wedge \phi_5).$$

What we cannot say in this logic is that the location of the diamond is common knowledge. ■

Ten

A Complete Axiomatization

In this chapter we will present an axiomatization for our logic, prove that it is complete, and we will also discuss why our logic is weakly complete and not strongly complete.

10.1 Axioms

We assume the axioms and rules of classical propositional logic. Because we have 5 independent modalities in our logic ($K_i, \langle + \rangle_a, \langle - \rangle_a, \langle + \rangle^*$ and $\langle - \rangle^*$) we expect to have, in addition, five classes of axioms (one for each modality) reflecting the behaviour of that modality in relation to Booleans. In addition, we will have a few other classes of axioms describing the relations between various modalities. For instance, $\langle + \rangle_a$ and $\langle - \rangle_a$ are in a certain duality supported by our intuition about time, so we expect to have some axioms relating these two. Similarly between $\langle + \rangle^*$ and $\langle - \rangle^*$. We also have some clear intuition about the relation between time transition and knowledge update that will be characterized by some axioms combining dynamic and epistemic operators.

The axioms of $\mathcal{L}$ are presented in Figure 10.1.

(A1): $\vdash [+]_a\phi \wedge [+]_a(\phi \Rightarrow \psi) \Rightarrow [+]_a\psi$

(A2): If $\vdash \phi$ then $\vdash [+]_a\phi$

(B1): $\vdash [-]_a\phi \wedge [-]_a(\phi \Rightarrow \psi) \Rightarrow [-]_a\psi$

(B2): If $\vdash \phi$ then $\vdash [-]_a\phi$

(B3): $\vdash \langle - \rangle_a \top \Rightarrow \bigwedge_{a \neq b} [-]_b \bot$

(B4): $\vdash \langle - \rangle_a \phi \Rightarrow [-]\phi$

(AB1): $\vdash \phi \Rightarrow [+]_a \langle - \rangle_a \phi$

(AB2): $\vdash \phi \Rightarrow [-]_a \langle + \rangle_a \phi$

(C1): $\vdash [+]^*\phi \wedge [+]^*(\phi \Rightarrow \psi) \Rightarrow [+]^*\psi$

(C2): If $\vdash \phi$ then $\vdash [+]^*\phi$

(C3): $\vdash [+]^*\phi \leftrightarrow (\phi \wedge [+][+]^*\phi)$

(C4): $\vdash [+]^*(\phi \Rightarrow [+]\phi) \Rightarrow (\phi \Rightarrow [+]^*\phi)$

(D1): $\vdash [-]^*\phi \wedge [-]^*(\phi \Rightarrow \psi) \Rightarrow [-]^*\psi$

(D2): If $\vdash \phi$ then $\vdash [-]^*\phi$

(D3): $\vdash [-]^*\phi \leftrightarrow (\phi \wedge [-][-]^*\phi)$

(D4): $\vdash [-]^*(\phi \Rightarrow [-]\phi) \Rightarrow (\phi \Rightarrow [-]^*\phi)$

(BD1): $\vdash \langle - \rangle^* [-] \bot$

(E1): $\vdash K_i\phi \wedge K_i(\phi \Rightarrow \psi) \Rightarrow K_i\psi$

(E2): If $\vdash \phi$ then $\vdash K_i\phi$

(E3): $\vdash K_i\phi \Rightarrow \phi$

(E4): $\vdash K_i\phi \Rightarrow K_iK_i\phi$

(E5): $\vdash \neg K_i\phi \Rightarrow K_i\neg K_i\phi$

(BE1): $\vdash \langle - \rangle_a K_i\phi \Rightarrow K_i \langle - \rangle_a \phi$

Figure 10.1: Hilbert-style axiomatization for $\mathcal{L}$

10.2 Soundness and Completeness

Many of the lemmas apply generically to $\langle\rangle$ or [] modalities and the proofs are essentially identical for the different variants. To streamline some proofs, we use the tuple of symbols $(\Diamond, \Box)$ to represent an arbitrary tuple of type $(\langle - \rangle_a, [-]_a)$, $(\langle + \rangle_a, [+]_a)$, $(\langle - \rangle, [-])$, or $(\langle + \rangle, [+])$. Similarly, $(\Diamond^*, \Box^*)$ represents $(\langle + \rangle^*, [+]^*)$ or $(\langle - \rangle^*, [-]^*)$. We also use $(\Diamond^x, \Box^x)$ to represent an arbitrary tuple of type $(\langle - \rangle_a, [-]_a)$, $(\langle + \rangle_a, [+]_a)$, $(\langle - \rangle, [-])$, $(\langle + \rangle, [+])$, $(\langle + \rangle^*, [+]^*)$ or $(\langle - \rangle^*, [-]^*)$. With these notations, the axioms (A1),(A2), (B1), (B2), (C1), (C2) and (D1), (D2) can be regarded as instances of (X1), (X2). Similarly, (C3), (C4) and (D3), (D4) are instances of (X3), (X4).

(X1): $\vdash \Box^x\phi \wedge \Box^x(\phi \Rightarrow \psi) \Rightarrow \Box^x\psi$

(X2): If $\vdash \phi$ then $\vdash \Box^x\phi$

(X3): $\vdash \Box^*\phi \leftrightarrow (\phi \wedge \Box\Box^*\phi)$

(X4): $\vdash \Box^*(\phi \Rightarrow \Box\phi) \Rightarrow (\phi \Rightarrow \Box^*\phi)$

From (X1) and (X2) alone we can prove a lemma which can be instantiated to all the particular instances. This is a standard lemma of modal logic.

Lemma 10.2.1. *1. If* $\vdash \phi \Rightarrow \psi$, *then* $\vdash \Box^x\phi \Rightarrow \Box^x\psi$ *and* $\vdash \Diamond^x\phi \Rightarrow \Diamond^x\psi$.

2. If $\vdash \phi \Rightarrow \psi$, *then* $\vdash K_i\phi \Rightarrow K_i\psi$.

3. $\vdash \langle - \rangle_a\phi \Rightarrow [-]_a\phi$ *and* $\vdash \langle - \rangle\phi \Rightarrow [-]\phi$.

Proof. 1. From (X2), $\vdash \phi \Rightarrow \psi$ implies $\vdash \Box^x(\phi \Rightarrow \psi)$. If we use this with $\vdash \Box^x(\phi \Rightarrow \psi) \Rightarrow (\Box^x\phi \Rightarrow \Box^x\psi)$, which is equivalent to (X1), we obtain $\vdash \Box^x\phi \Rightarrow \Box^x\psi$.

To prove the second implication, we start from $\vdash \neg\psi \Rightarrow \neg\phi$ and apply the first result which gives us $\vdash \Box^x\neg\phi \Rightarrow \Box^x\neg\psi$. Using De Morgan we derive $\vdash \Diamond^x\phi \Rightarrow \Diamond^x\psi$.

2. It is proved in the same way as 1; in fact K is a box-like modality.

3. From (B4) we have $\vdash \langle - \rangle_a \phi \Rightarrow \bigwedge_a [-]_a \phi$ which implies $\vdash \langle - \rangle_a \phi \Rightarrow [-]_a \phi$. The same axiom implies $\vdash \bigwedge_a (\langle - \rangle_a \phi \Rightarrow [-]\phi)$ which is equivalent to $\vdash \bigvee_a \langle - \rangle_a \phi \Rightarrow [-]\phi$ which implies $\vdash \langle - \rangle \phi \Rightarrow [-]\phi$. ∎

As usual, we say that a formula $\phi \in \mathcal{L}$ is *provable*, denoted by $\vdash \phi$, if it can be proved from the axioms in Table 10.1 and boolean rules. We say that ϕ is *consistent*, if $\neg\phi$ is not provable from the axioms.

Given $\Phi, \Psi \subseteq \mathcal{L}$, Φ proves Ψ if from the formulas of Φ and the axioms we can prove each $\psi \in \Psi$; we write $\Phi \vdash \Psi$. Let $[\Phi] = \{\psi \in \mathcal{L} \mid \Phi \vdash \psi\}$; this is the deductive closure of Φ. Φ is consistent if it is not the case that $\Phi \vdash \bot$.

For a sublanguage $L \subseteq \mathcal{L}$, we call Φ *L-maximally consistent* if Φ is consistent and no formula of L can be added to it without making it inconsistent. The following lemma follows directly from the definition of maximal consistency.

Lemma 10.2.2. *If Γ is a consistent set of formulas then the following assertions are true.*

1. *if $\Diamond^x \top \in [\Gamma]$ and $\Diamond^x \phi \notin [\Gamma]$, then $\{\psi \in \mathcal{L} \mid \Box^x \psi \in [\Gamma]\} \cup \{\neg\phi\}$ is consistent.*
2. *if $\Box^x \phi \notin [\Gamma]$, then $\{\psi \in \mathcal{L} \mid \Box^x \psi \in [\Gamma]\} \cup \{\neg\phi\}$ is consistent.*

Proof. Let $\Lambda = \{\psi \in \mathcal{L} \mid \Box^x \psi \in [\Gamma]\}$. Suppose that $\Lambda \cup \{\neg\phi\}$ is inconsistent. Then there is a finite set $\{f_1, .., f_n\} \subseteq \Lambda$ s.t. $\vdash f_1 \wedge .. \wedge f_n \Rightarrow \phi$. Hence, $\vdash \Box^x(f_1 \wedge .. \wedge f_n) \Rightarrow \Box^x \phi$ implying further $\vdash (\Box^x f_1 \wedge .. \wedge \Box^x f_n) \Rightarrow \Box^x \phi$. Hence, $\Box^x \phi \in [\Gamma]$.

1. If $\Diamond^x \top \in [\Gamma]$, from $\Box^x \phi \in [\Gamma]$ we obtain $\Diamond^x \phi \in [\Gamma]$ - contradiction.
2. $\Box^x \phi \notin [\Gamma]$ is again contradictory. ∎

A basic theorem that holds for the axiom system is the soundness property.

Theorem 10.2.3 (Soundness)**.** *The axiomatic system of $\mathcal{L}$ is sound, i.e., for any $\phi \in \mathcal{L}$,*

$$\vdash \phi \text{ implies } \models \phi.$$

The proof is a routine structural induction. It is sufficient to prove that each axiom is sound and that each rule preserves the soundness.

The more interesting result is the completeness of the axiom system. Moreover, we will show that for each consistent formula a *finite* model can be constructed.

Recall that there are two notions of completeness: *strong* completeness and *weak* completeness. Strong completeness says that

$$\Gamma \models \phi \iff \Gamma \vdash \phi.$$

An important easy consequence of strong completeness is the so-called *compactness* property. A logic is said to be compact if every inconsistent set of formulas has a *finite* inconsistent subset. Our logic is not compact. For example, the set of formulas

$$\{p, [+]p, [+][+]p, [+]^3 p, \ldots, \neg[+]^* p\}$$

is not consistent but any finite subset is consistent. Therefore we cannot hope to prove strong completeness. Instead we prove weak completeness

$$\models \phi \iff \vdash \phi.$$

Many of the basic completeness proofs in the literature are strong completeness proofs and are much easier than weak completeness proofs. The proof that we present shares many of the features of the weak completeness proof for Propositional Dynamic Logic (PDL) [Pra76].

Before proceeding with these proofs we establish some notation that will be useful for future constructions.

We extend, canonically, all the logical operators from formulas to sets of formulas. Thus for arbitrary $\Phi, \Psi \subseteq \mathcal{L}$, $\Phi \wedge \Psi = \{\phi \wedge \psi \mid \phi \in \Phi, \psi \in \Psi\}$, $\langle + \rangle_a \Phi = \{\langle + \rangle_a \phi \mid \phi \in \Phi\}$, and so on for all the modal operators.

If $\Phi \subseteq \mathcal{L}$ is finite, we use Φ to also denote $\bigwedge_{\phi \in \Phi} \phi$; it should be clear from the context when Φ denotes a set of formulas and when it denotes the conjunction of its elements.

A key step in the proof is the construction of models by using maximally consistent sets as states. However, because we are trying to prove a weak completeness theorem we have to ensure that we are constructing finite sets of formulas. The liberal notion of maximal consistency used in strong completeness proofs is not available to us. If we wish to construct a model of a formula ϕ, we need to define a special family of formulas associated with ϕ from which we will construct maximal consistent subsets. Furthermore we need to ensure that the collection of formulas we construct is finite. We adapt a construction due to Fischer and Ladner [FL79] developed in the context of PDL.

For an arbitrary $\phi \in \mathcal{L}$, let $\sim \phi = \psi$ whenever $\phi = \neg\psi$ and $\sim \phi = \neg\phi$ otherwise.
For an arbitrary $\phi \in \mathcal{L}$, let $\overline{k}_i\phi = \phi$ whenever $\phi = K_i\psi$ or $\phi = \neg K_i\psi$ and $\overline{k}_i\phi = K_i\phi$ otherwise.

Definition 10.2.4. *The* (Fischer-Ladner) closure of ϕ, *written* $\mathcal{FL}(\phi)$, *is defined as a set of formulas such that:*

- $\phi, \langle - \rangle_a p, \langle - \rangle_a \top \in \mathcal{FL}(\phi)$,
- *if* $\psi \in \mathcal{FL}(\phi)$, *then* $\sim \psi \in \mathcal{FL}(\phi)$, $\overline{k}_i\psi$ *and any subformula of* ψ *is in* $\mathcal{FL}(\phi)$,
- *if* $\langle - \rangle_a \psi \in \mathcal{FL}(\phi)$ *or* $\langle + \rangle_a \psi \in \mathcal{FL}(\phi)$, *then* $\langle - \rangle \psi, \langle + \rangle \psi \in \mathcal{FL}(\phi)$,
- *if* $\Diamond^* \psi \in \mathcal{FL}(\phi)$, *then* $\Diamond\Diamond^* \psi \in \mathcal{FL}(\phi)$.

The following lemma is immediate but important to state because we have to ensure that we always have finite sets of formulas when we construct models out of sets of formulas.

Lemma 10.2.5. *For any* $\phi \in \mathcal{L}$, $\mathcal{FL}(\phi)$ *is finite.*

In what follows we fix a consistent formula $\theta \in \mathcal{L}$ and we construct a finite model for θ. This means that we construct an epistemic frame $\mathcal{E}_\theta = (\mathcal{F}_\theta, (\approx_i)_{i\in\mathcal{I}})$, a valuation $\rho : supp(\mathcal{F}_\theta) \Rightarrow 2^{\mathcal{P}}$ and we will identify a state $s \in supp(\mathcal{F}_\theta)$ such that $s \models \theta$.

Let Ω_θ be the set of $\mathcal{FL}(\theta)$-maximally consistent sets. Because $\mathcal{FL}(\theta)$ is finite, Ω_θ and any $\Gamma \in \Omega_\theta$ are finite sets. In the construction of the model we will use Ω_θ as the support set for $\mathcal{F}_\theta$. The transitions on Ω_θ are defined as follows. For each $a \in \mathcal{A}$, let $\xrightarrow{a} \subseteq \Omega_\theta \times \Omega_\theta$ be defined by

$$\Gamma \xrightarrow{a} \Gamma' \text{ iff for any } \psi \in \mathcal{L}, [+]_a\psi \in [\Gamma] \text{ implies } \psi \in [\Gamma'].$$

Now we prove a few properties of these transitions that will be important for the rest of the proof.

Lemma 10.2.6. *For arbitrary $\Gamma, \Gamma' \in \Omega_\theta$ the following are equivalent:*

1. *for any $\phi \in \mathcal{L}$, $[+]_a\phi \in [\Gamma]$ implies $\phi \in [\Gamma']$,*
2. *for any $\phi \in \mathcal{L}$, $[-]_a\phi \in [\Gamma']$ implies $\phi \in [\Gamma]$.*

Proof. (1) implies (2): Suppose that $[-]_a\phi \in [\Gamma']$. Then, $\vdash \Gamma' \Rightarrow [-]_a\phi$ and using axiom (AB1), $\vdash \langle+\rangle_a\Gamma' \Rightarrow \phi$. If we prove that $\langle+\rangle_a\Gamma' \in [\Gamma]$, then $\phi \in [\Gamma]$ and the proof is done. Observe that $\langle+\rangle_a\top \in [\Gamma]$ because otherwise $\neg\langle+\rangle_a\top \in [\Gamma]$ implying $[+]_a\bot \in [\Gamma]$ and from the hypothesis we obtain $\bot \in [\Gamma']$ - impossible. Hence, $\langle+\rangle_a\top \in [\Gamma]$ and if $\langle+\rangle_a\Gamma' \notin [\Gamma]$, from Lemma 10.2.2 instantiated to $\Box^x = [+]_a$, we obtain that $\{\psi \mid [+]_a\psi \in [\Gamma]\} \cup \{\neg\Gamma'\}$ is consistent. But this is impossible because, from the hypothesis, $\{\psi \mid [+]_a\psi \in [\Gamma]\} \subseteq [\Gamma']$.

(2) implies (1) Suppose that $[+]_a\phi \in [\Gamma]$. Then, $\vdash \Gamma \Rightarrow [+]_a\phi$ implying $\vdash \langle-\rangle_a\Gamma \Rightarrow \langle-\rangle_a[+]_a\phi$. Now (AB2) guarantees that $\vdash \langle-\rangle_a\Gamma \Rightarrow \phi$. In any normal modal logic we have that $\vdash (\Box\psi \wedge \Diamond\top) \Rightarrow \Diamond\psi$. We use this with the previous formula and we obtain $\vdash ([-]_a\Gamma \wedge \langle-\rangle_a\top) \Rightarrow \phi$.

Note that $\langle-\rangle_a\top \in \Gamma'$ because otherwise $[-]_a\bot \in \Gamma'$ and, from the hypothesis we obtain that $\bot \in [\Gamma]$ - impossible. Now, if we prove that $[-]_a\Gamma \in [\Gamma']$, then $\phi \in [\Gamma']$ and the proof is done. Now note that $[-]_a\Gamma \notin [\Gamma']$ implies, using Lemma 10.2.2 instantiated with $\Box^x = [-]_a$, that $\{\psi \mid$

$[-]_a\psi \in [\Gamma']\} \cup \{\neg\Gamma\}$ is consistent. But this is impossible because, from the hypothesis, $\{\psi \mid [-]_a\psi \in [\Gamma']\} \subseteq [\Gamma]$. ∎

This lemma tells us that we can define the transitions either using $[+]$ or $[-]$.

Lemma 10.2.7. *For arbitrary* $\Gamma \in \Omega_\theta$ *and* $[+]_a\phi \in \mathcal{FL}(\theta)$,

1. $[+]_a\phi \in \Gamma$ *iff for any* $\Gamma' \in \Omega_\theta$, $\Gamma \xrightarrow{a} \Gamma' \Rightarrow \phi \in \Gamma'$;
2. $\langle+\rangle_a\phi \in \Gamma$ *iff there exists* $\Gamma' \in \Omega_\theta$ *such that* $\Gamma \xrightarrow{a} \Gamma', \phi \in \Gamma'$;
3. $[-]_a\phi \in \Gamma$ *iff for any* $\Gamma' \in \Omega_\theta$ *such that* $\Gamma' \xrightarrow{a} \Gamma, \phi \in \Gamma'$;
4. $\langle-\rangle_a\phi \in \Gamma$ *iff there exists* $\Gamma' \in \Omega_\theta$ *such that* $\Gamma' \xrightarrow{a} \Gamma, \phi \in \Gamma'$.

Proof. 1. ($\Rightarrow$:) From the definition of $\xrightarrow{a}$.
($\Leftarrow$:) Let ϕ be such that $\phi \in [\Gamma']$ for each $\Gamma' \in \Omega_\theta$ with $\Gamma \xrightarrow{a} \Gamma'$. We need to prove that $[+]_a\phi \in [\Gamma]$. Note that a formula that is in $[\Gamma]$ and also in $\mathcal{FL}(\theta)$ is automatically in Γ.

Let $\Delta = \{\Gamma' \in \Omega_\theta \mid \Gamma \xrightarrow{a} \Gamma'\}$ and let $\delta = \bigvee_{\Gamma' \in \Delta} \Gamma'$. Obviously, $\vdash \delta \Rightarrow \phi$ implying $\vdash [+]_a\delta \Rightarrow [+]_a\phi$. Now, if we prove that $[+]_a\delta \in [\Gamma]$, the proof is done.

Suppose that $[+]_a\delta \notin [\Gamma]$. Lemma 10.2.2 implies that $\Lambda \cup \{\neg\delta\}$ is consistent, where $\Lambda = \{\psi \mid [+]_a\psi \in [\Gamma]\}$. But $[+]_a\psi \in [\Gamma]$ implies $\psi \in \Gamma'$ for each $\Gamma' \in \Delta$ and this proves that $\Lambda \cup \{\neg\delta\}$ cannot be consistent.

(2) is the De Morgan dual of (1).

(3) and (4) are proved in the same way as (1) and (2). ∎

We draw the reader's attention to a minor subtlety in the proof because it recurs in several later proofs. We showed that a formula in $\mathcal{FL}(\theta)$, say ϕ, is in the *deductive closure* of a maximally consistent subset, say Γ, of $\mathcal{FL}(\theta)$, in other words we showed that $\phi \in [\Gamma]$. From the fact that ϕ is itself in $\mathcal{FL}(\theta)$ we were able to deduce that ϕ is in Γ itself precisely because Γ is maximal consistent as a subset of $\mathcal{FL}(\theta)$.

We now need to establish the analogous results for the starred modalities. In what follows, let $\rightarrow = \bigcup_{a \in \mathcal{A}} \xrightarrow{a}$ and $\rightarrow^*$ be its reflexive-transitive closure. This means that $\Gamma \rightarrow^* \Gamma'$ if there exists a sequence $\Gamma_1, \ldots, \Gamma_k \in \Omega_\theta$ such that

$$\Gamma = \Gamma_1 \rightarrow \Gamma_2 \rightarrow \ldots \rightarrow \Gamma_{k-1} \rightarrow \Gamma_k = \Gamma';$$

Because $\rightarrow^*$ is reflexive, k can be 1.

Lemma 10.2.8. *For arbitrary $\Gamma, \Gamma' \in \Omega_\theta$ the following are equivalent*

1. *for any $\phi \in \mathcal{L}$, $[+]^*\phi \in [\Gamma]$ implies $\phi \in [\Gamma']$,*
2. *for any $\phi \in \mathcal{L}$, $[-]^*\phi \in [\Gamma']$ implies $\phi \in [\Gamma]$,*
3. *$\Gamma \rightarrow^* \Gamma'$.*

Proof. (1) $\Longrightarrow$ (3): Let $\Delta = \{\Lambda \in \Omega_\theta \mid \Gamma \rightarrow^* \Lambda\}$ and $\delta = \bigvee_{\Lambda \in \Delta} \Lambda$.

By construction, if $[+]\phi \in [\Lambda]$ for some $\Lambda \in \Delta$, there exists $\Lambda' \in \Delta$ such that $\phi \in [\Lambda']$. This entails $\vdash \delta \Rightarrow [+]\delta$ which guarantees that $\vdash [+]^*(\delta \Rightarrow [+]\delta)$. Using axiom (C4), we obtain $\vdash \delta \Rightarrow [+]^*\delta$. But $\Gamma \in \delta$ (because $\rightarrow^*$ is reflexive), consequently $\vdash \Gamma \Rightarrow \delta$. From here and the previous we derive $\vdash \Gamma \Rightarrow [+]^*\delta$ implying $[+]^*\delta \in [\Gamma]$. Now using 1., $\delta \in [\Gamma']$ implying $\Gamma' \in \Delta$.

(3) $\Rightarrow$ (1): Suppose that $\Gamma = \Gamma_1 \rightarrow \ldots \rightarrow \Gamma_k = \Gamma'$ and $[+]^*\phi \in [\Gamma]$. Axiom (C3) guarantees that $\phi \in [\Gamma_1]$ and $[+][+]^*\phi \in [\Gamma_1]$. Hence $[+]^*\phi \in [\Gamma_2]$ from the definition of $\rightarrow$. The same argument can be repeated for the k cases eventually giving $[+]^*\phi \in [\Gamma_k] = [\Gamma']$ which implies, using axiom (C3), $\phi \in [\Gamma']$.

(2) $\Leftrightarrow$ (3): It is proved in the same way using the axioms (D1) and (D2) in instances of Lemma 10.2.1 and (D3), (D4) respectively. ∎

Lemma 10.2.9. *For arbitrary $\Gamma \in \Omega_\theta$ and $[+]^*\phi \in \mathcal{FL}(\theta)$,*

1. *$[+]^*\phi \in \Gamma$ iff for any $\Gamma' \in \Omega_\theta$ such that $\Gamma \rightarrow^* \Gamma', \phi \in \Gamma'$;*
2. *$\langle+\rangle^*\phi \in \Gamma$ iff there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma \rightarrow^* \Gamma', \phi \in \Gamma'$;*
3. *$[-]^*\phi \in \Gamma$ iff for any $\Gamma' \in \Omega_\theta$ such that $\Gamma' \rightarrow^* \Gamma, \phi \in \Gamma'$;*

4. $\langle - \rangle^* \phi \in \Gamma$ *iff there exists* $\Gamma' \in \Omega_\theta$ *such that* $\Gamma' \rightarrow^* \Gamma, \phi \in \Gamma'$.

Proof. (1) $\Rightarrow$: From Lemma 10.2.8.
($\Leftarrow$:) Let ϕ be such that $\phi \in [\Gamma']$ for each $\Gamma' \in \Omega_\theta$ with $\Gamma \rightarrow^* \Gamma'$. We need to prove that $[+]^*\phi \in [\Gamma]$.

Let $\Delta = \{\Gamma' \in \Omega_\theta \mid \Gamma \rightarrow^* \Gamma'\}$ and let $\delta = \bigvee_{\Gamma' \in \Delta} \Gamma'$. Obviously, $\vdash \delta \Rightarrow \phi$ implying $\vdash [+]^*\delta \Rightarrow [+]^*\phi$. Now, if we prove that $[+]^*\delta \in [\Gamma]$, the proof is done.

Suppose that $[+]^*\delta \notin [\Gamma]$. Lemma 10.2.2 implies that $\Lambda \cup \{\neg\delta\}$ is consistent, where $\Lambda = \{\psi \mid [+]^*\psi \in [\Gamma]\}$. But $[+]^*\psi \in [\Gamma]$ implies $\psi \in \Gamma'$ for each $\Gamma' \in \Delta$ and this proves that $\Lambda \cup \{\neg\delta\}$ cannot be consistent.

(2) is equivalent to (1).

(3) and (4) are proved in the same way. ∎

Now we can proceed with our construction of the model for θ. We start by showing that $(\Omega_\theta, \xrightarrow{a})_{a \in \mathcal{A}}$ is a forest. For this we need to verify that the past is unique and that the graphs have no loops. The precise statement is given in the following theorem.

Theorem 10.2.10. *If* $f \in \mathcal{L}$ *is consistent, then* $\mathcal{F}_\theta = (\Omega_\theta, \xrightarrow{a})_{a \in \mathcal{A}}$ *is a forest over* $\mathcal{A}$.

The proof of this theorem is broken down into two lemmas.

Lemma 10.2.11. *For arbitrary* $\Gamma, \Gamma_1, \Gamma_2 \in \Omega_\theta$, *if* $\Gamma_1 \xrightarrow{a} \Gamma$ *and* $\Gamma_2 \xrightarrow{b} \Gamma$, *then* $a = b$ *and* $\Gamma_1 = \Gamma_2$.

Proof. To prove that $a = b$ it is sufficient to observe that $\langle - \rangle_a \top \wedge \langle - \rangle_b \top$ is inconsistent, result that is a direct consequence of axiom (B3).

Now, from $\Gamma_1 \xrightarrow{a} \Gamma$ and $\Gamma_2 \xrightarrow{a} \Gamma$ we prove that $\Gamma_1 = \Gamma_2$. Suppose that there exists $\phi \in \mathcal{FL}(\theta)$ s.t. $\phi \in \Gamma_1$ and $\neg\phi \in \Gamma_2$. Then, from axiom (AB1) we obtain that $[+]_a\langle - \rangle_a \phi \in [\Gamma_1]$ and $[+]_a\langle - \rangle_a \neg\phi \in [\Gamma_2]$. Now $\Gamma_1 \xrightarrow{a} \Gamma$ guarantees that $\langle - \rangle_a \phi \in [\Gamma]$ while $\Gamma_2 \xrightarrow{a} \Gamma$ guarantees that $\langle - \rangle_a \neg\phi \in [\Gamma]$. Further, using axiom (B4) we obtain that $[-]\phi, [-]\neg\phi \in [\Gamma]$ implying $[-]\bot \in [\Gamma]$. On the other hand, $\langle - \rangle_a \phi \in [\Gamma]$ implies $\langle - \rangle_a \top \in [\Gamma]$ which is equivalent to $\neg[-]\bot \in [\Gamma]$ - contradicts the consistency of $[\Gamma]$. ∎

Now we prove that in the graph $(\Omega_\theta, \xrightarrow{a})_{a\in\mathcal{A}}$ there are no backwards infinite sequences; this will conclude the proof that $(\Omega_\theta, \xrightarrow{a})_{a\in\mathcal{A}}$ is a forest over $\mathcal{A}$.

Lemma 10.2.12. *There exists no infinite sequence* $\Gamma_1, \ldots, \Gamma_k, \ldots \in \Omega_\theta$ *such that*

$$\ldots \Gamma_k \longrightarrow \Gamma_{k-1} \longrightarrow \ldots \longrightarrow \Gamma_1 = \Gamma.$$

Proof. Suppose that there exists such a sequence. Axiom (BD1) guarantees that $\langle - \rangle^*[-]\bot \in [\Gamma]$ and using Lemma 10.2.9 we obtain that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \longrightarrow^* \Gamma$ and $[-]\bot \in \Gamma'$. Lemma 10.2.11 guarantees that Γ' is one of the elements of our sequence, hence $\neg\langle - \rangle \top \in \Gamma'$. But this implies that there exists no $\Gamma'' \in \Omega_\theta$ such that $\Gamma'' \longrightarrow^* \Gamma'$, this contradiction establishes the result. ∎

To complete the construction of the model for θ we need to define the indistinguishability relations on Ω_θ that will eventually organize our forest as an epistemic frame.

For each $i \in \mathcal{I}$, let $\approx_i \subseteq \Omega_\theta \times \Omega_\theta$ be defined as follows:

$$\Gamma \approx_i \Gamma' \text{ iff for any } \phi \in \mathcal{L}, K_i\phi \in [\Gamma] \text{ iff } K_i\phi \in [\Gamma'].$$

By construction, $\approx_i$ is an equivalence relation. Now, to finalize our construction, we must prove that for each $i \in \mathcal{I}$, $\approx_i$ preserves the branching structure of $\mathcal{F}_\theta$ and finally that we have an epistemic frame.

Theorem 10.2.13. $\mathcal{E}_\theta = (\mathcal{F}_\theta, (\approx_i)_{i\in\mathcal{I}})$, *where* $\mathcal{F}_\theta = (\Omega_\theta, \xrightarrow{a})_{a\in\mathcal{A}}$ *and* $\approx_i$ *are defined as before, is an epistemic frame.*

The proof is broken into a number of lemmas. The first lemma that we need is the following.

Lemma 10.2.14. *For arbitrary* $\Gamma, \Gamma' \in \Omega_\theta$, *if for any* ϕ, $K_i\phi \in [\Gamma]$ *implies* $\phi \in [\Gamma']$, *then for any* ϕ, $K_i\phi \in [\Gamma]$ *implies* $K_i\phi \in [\Gamma']$.

Proof. Suppose that for any ϕ, $K_i\phi \in [\Gamma]$ implies $\phi \in [\Gamma']$ and let $K_i\psi \in [\Gamma]$. From our hypothesis we obtain that if $K_i\psi \notin [\Gamma']$, then $K_iK_i\psi \notin [\Gamma]$. From

the axioms (E3) and (E4), $\vdash K_i\psi \leftrightarrow K_iK_i\psi$. Hence, $K_i\psi \notin [\Gamma]$, this contradiction completes the proof. ∎

Now we can prove that for each $i \in \mathcal{I}$, $\approx_i$ preserves the backwards branching structure of $\mathcal{F}_\theta$.

Theorem 10.2.15. *For arbitrary $\Gamma, \Gamma' \in \Omega_\theta$, if $\Gamma \approx_i \Gamma'$ and there exists $\Gamma_0 \in \Omega_\theta$ such that $\Gamma_0 \xrightarrow{a} \Gamma$, then there exists $\Gamma'_0 \in \Omega_\theta$ such that $\Gamma'_0 \xrightarrow{a} \Gamma'$ and $\Gamma'_0 \approx_i \Gamma_0$.*

Proof. Because $\vdash \top$, using (E2) we obtain $\vdash K_i\top$. Because $K_i\top \in [\Gamma_0]$, we obtain that $\langle - \rangle_a K_i\top \in [\Gamma]$ and axiom (BE1) implies $K_i\langle - \rangle_a \top \in [\Gamma]$. Now, from $\Gamma \approx_i \Gamma'$, $\langle - \rangle_a \top \in [\Gamma']$. From Lemma 10.2.7 we obtain that there exists $\Gamma'_0 \in \Omega_\theta$ such that $\Gamma'_0 \rightarrow \Gamma'$.

We prove now that $\Gamma'_0 \approx \Gamma_0$. Suppose that $K_i\phi \in [\Gamma_0]$. Then, $\langle - \rangle_a K_i\phi \in [\Gamma]$ and axiom (BE1) implies $K_i\langle - \rangle_a \phi \in [\Gamma]$. Now from $\Gamma \approx_i \Gamma'$, $\langle - \rangle_a \phi \in [\Gamma']$. Now axiom (B4) implies $[-]\phi \in [\Gamma']$ and because $\Gamma'_0 \rightarrow \Gamma'$, Lemma 10.2.7 implies $\phi \in [\Gamma'_0]$.

Hence, $K_i\phi \in [\Gamma_0]$ implies $\phi \in [\Gamma'_0]$ and Lemma 10.2.14 concludes that $K_i\phi \in [\Gamma_0]$ implies $K_i\phi \in [\Gamma'_0]$. Similarly can be proved that $K_i\phi \in [\Gamma'_0]$ implies $K_i\phi \in [\Gamma_0]$. ∎

Lemma 10.2.14 also establishes the next result that is needed for the proof of the theorem.

Lemma 10.2.16. *For arbitrary $\Gamma \in \Omega_\theta$ and $K_i\phi \in \mathcal{FL}(\theta)$,*

$$K_i\phi \in \Gamma \text{ iff for any } \Gamma' \in \Omega_\theta \text{ such that } \Gamma \approx_i \Gamma', \phi \in \Gamma'$$

Proof. ($\Rightarrow$) This follows directly from Lemma 10.2.14.

($\Leftarrow$) Let ϕ be such that $K_i\phi \in \mathcal{FL}(\theta)$ and $\phi \in \Gamma'$ for each $\Gamma' \in \Omega_\theta$ with $\Gamma \approx_i \Gamma'$. We need to prove that $K_i\phi \in \Gamma$.

Let $\Delta = \{\Gamma' \in \Omega_\theta \mid \Gamma \approx_i \Gamma'\}$, let $\Lambda = \{f_1, \ldots, f_n\} = \bigcap_{\Gamma' \in \Delta} \Gamma'$ and let $F = f_1 \wedge \ldots \wedge f_n$. Then $\vdash F \Rightarrow \phi$ implying $\vdash K_iF \Rightarrow K_i\phi$. Consequently, if we prove that $K_iF \in [\Gamma]$, the proof is done.

Suppose that $K_iF \notin [\Gamma]$. Then, there exists $f_t \in \Lambda$ such that $K_if_t \notin \Gamma$. Then, $\neg K_if_t \in \Gamma$ and axiom (E5) implies $K_i\neg K_if_t \in [\Gamma]$. The definition of $\approx_i$ guarantees that for any $\Gamma' \in \Delta$, $K_i\neg K_if_t \in [\Gamma']$ and axiom (E3) entails that for any $\Gamma' \in \Delta$, $\neg K_if_t \in \Gamma'$. Hence, $\vdash F \Rightarrow \neg K_if_t$ which is equivalent to $\vdash K_if_t \Rightarrow \neg F$. But $\vdash F \Rightarrow f_t$ implying $\vdash K_iF \Rightarrow K_if_t$. Consequently, $\vdash K_iF \Rightarrow \neg F$. But from axiom (E3), $\vdash K_iF \Rightarrow F$, implying $\vdash \neg K_iF$. But Λ is consistent and $K_iF \notin [\Lambda]$, then a similar argument with the one used in Lemma 10.2.2 (notice that K_i is a normal modal operator of type $\Box$) shows that $\Lambda \cup \{\neg F\}$ is consistent, which is impossible. ∎

This completes the proof of the theorem.

We are now ready to complete the construction of the model of θ. $\mathcal{E}_\theta$ is the epistemic frame of the model and the we define a valuation $\rho_\theta : \Omega_\theta \longrightarrow 2^{\mathcal{P}}$ by $\rho_\theta(\Gamma) = \{p \in \mathcal{P} \mid p \in \Gamma\}$. With this definition we prove the Truth Lemma.

Lemma 10.2.17 (Truth Lemma)**.** *If $\theta \in \mathcal{L}$ is consistent, $\mathcal{E}_\theta$ and ρ_θ are defined as before, then for any $\phi \in \mathcal{FL}(\theta)$ and $\Gamma \in \Omega_\theta$,*

$$\phi \in \Gamma \quad \textit{if and only if} \quad \Gamma \models \phi.$$

Proof. Induction on ϕ.

Case $\phi = p \in \mathcal{P}$: from definition of $Prop_\theta$.

Case $\phi = \neg\psi$: ($\Longrightarrow$) Suppose that $\Gamma \not\models \neg\psi$. Then $\Gamma \models \psi$ and from the inductive hypothesis, $\psi \in \Gamma$, hence $\phi \notin \Gamma$.
($\Longleftarrow$) Suppose that $\Gamma \models \neg\psi$ and $\neg\psi \notin \Gamma$. Then, $\psi \in \Gamma$ and the inductive hypothesis guarantees that $\Gamma \models \psi$ - contradiction.

Case $\phi = \phi_1 \wedge \phi_2$: $\phi_1 \wedge \phi_2 \in \Gamma$ iff $\phi_1, \phi_2 \in \Gamma$ which is equivalent, using the inductive hypothesis, to [$\Gamma \models \phi_1$ and $\Gamma \models \phi_2$], equivalent to $\Gamma \models \phi_1 \wedge \phi_2$.

Case $\phi = \langle + \rangle_a\psi$: ($\Longrightarrow$) If $\langle + \rangle_a\psi \in \Gamma$, Lemma 10.2.7 implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma \xrightarrow{a} \Gamma'$ and $\psi \in \Gamma'$. From the inductive

hypothesis, $\Gamma' \models \psi$, implying $\Gamma \models \phi$.
($\Longleftarrow$) $\Gamma \models \langle + \rangle_a \psi$ implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma \xrightarrow{a} \Gamma'$ and $\Gamma' \models \psi$. From the inductive hypothesis, $\psi \in \Gamma'$ and Lemma 10.2.7 implies $\langle + \rangle_a \psi \in \Gamma$.

Case $\phi = \langle - \rangle_a \psi$: ($\Longrightarrow$) If $\langle - \rangle_a \psi \in \Gamma$, Lemma 10.2.7 implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \xrightarrow{a} \Gamma$ and $\psi \in \Gamma'$. From the inductive hypothesis, $\Gamma' \models \psi$, implying $\Gamma \models \phi$.
($\Longleftarrow$) $\Gamma \models \langle - \rangle_a \psi$ implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \xrightarrow{a} \Gamma$ and $\Gamma' \models \psi$. From the inductive hypothesis, $\psi \in \Gamma'$ and Lemma 10.2.7 implies $\langle - \rangle_a \psi \in \Gamma$.

Case $\phi = \langle + \rangle^* \psi$: ($\Longrightarrow$) If $\langle + \rangle^* \psi \in \Gamma$, Lemma 10.2.9 implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma \rightarrow^* \Gamma'$ and $\psi \in \Gamma'$. From the inductive hypothesis, $\Gamma' \models \psi$, implying $\Gamma \models \phi$.
($\Longleftarrow$) $\Gamma \models \langle + \rangle^* \psi$ implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma \rightarrow^* \Gamma'$ and $\Gamma' \models \psi$. From the inductive hypothesis, $\psi \in \Gamma'$ and Lemma 10.2.9 implies $\langle + \rangle^* \psi \in \Gamma$.

Case $\phi = \langle - \rangle^* \psi$: ($\Longrightarrow$) If $\langle - \rangle^* \psi \in \Gamma$, Lemma 10.2.9 implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \rightarrow^* \Gamma$ and $\psi \in \Gamma'$. From the inductive hypothesis, $\Gamma' \models \psi$, implying $\Gamma \models \phi$.
($\Longleftarrow$) $\Gamma \models \langle - \rangle^* \psi$ implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \rightarrow^* \Gamma$ and $\Gamma' \models \psi$. From the inductive hypothesis, $\psi \in \Gamma'$ and Lemma 10.2.9 implies $\langle - \rangle^* \psi \in \Gamma$.

Case $\phi = K_i \psi$: ($\Longrightarrow$) If $K_i \psi \in \Gamma$, Lemma 10.2.16 implies that for any $\Gamma' \in \Omega_\theta$ such that $\Gamma \approx_i \Gamma'$, $\psi \in \Gamma'$. From the inductive hypothesis, $\Gamma' \models \psi$, implying $\Gamma \models \phi$.
($\Longleftarrow$) $\Gamma \models K_i \psi$ implies that for any $\Gamma' \in \Omega_\theta$ such that $\Gamma \approx_i \Gamma'$, $\Gamma' \models \psi$. From the inductive hypothesis, $\psi \in \Gamma'$ and Lemma 10.2.16 implies $K_i \psi \in \Gamma$.

∎

A direct consequence of Truth Lemma is the finite model property.

Theorem 10.2.18 (Finite model property)**.** *For any consistent formula $\phi \in \mathcal{L}$ there exists a finite model. Moreover, the size of the model is bound by the structure of ϕ.*

The finite model property in this context has two important consequences: the weak completeness of the axiomatic system and the decidability of the satisfiability problem.

Theorem 10.2.19 (Weak completeness)**.** *The axiomatic system of $\mathcal{L}$ is complete, i.e., for any $\phi \in \mathcal{L}$,*

$$\models \phi \text{ implies } \vdash \phi.$$

Proof. The proof is based on the fact that any consistent formula has a model. We wish to show that $\models \phi$ implies $\vdash \phi$. Now we have shown that if ϕ is consistent it has a model. Clearly then, if $\neg\phi$ is consistent there is a model of $\neg\phi$. The last statement is equivalent to saying that if $\nvdash \phi$ then $\neg\phi$ is satisfiable. If $\neg\phi$ is satisfiable it follows that not every model models ϕ, i.e. $\not\models \phi$. Thus we have $\nvdash \phi$ implies $\not\models \phi$, or taking the contrapositive, $\models \phi$ implies $\vdash \phi$. ∎

Observe that in the previous construction, the size of Ω_θ depends on the number and type of operators that θ contains. In what follows we refer to the cardinality $|\Omega_\theta|$ of Ω_θ as *the size of* θ.

The satisfiability problem is the problem of deciding, given an arbitrary formula $\phi \in \mathcal{L}$, if ϕ has at least one model. The finite model property entails that the satisfiability problem for our logic is decidable.

Theorem 10.2.20 (Decidability)**.** *The satisfiability problem for $\mathcal{L}$ is decidable.*

Proof. We have proved that θ has at least one model iff it is consistent. And if θ is consistent we have proved that it has a model of size $|\Omega_\theta| \in \mathbb{N}$. But the class of models of size $k \in \mathbb{N}$ is finite. Consequently, we can decide in a finite number of steps if θ does or does not have a model by checking all the models of the appropriate sizes. ∎

Conclusions and Related Work

Related Work

The ground breaking paper of Halpern and Moses [HM84, HM90] showed the importance of common knowledge as a way of formalizing agreement protocols in distributed systems. Very quickly variants of common knowledge were developed [NT87, PT88] and many new applications were explored [NT90]. Extensions to probability [HT89] and zero-knowledge protocols [HMT88] quickly followed. The textbook of Fagin et al. [FHMV95] made these ideas widely accessible and stimulated even more interest and activity. There are numerous recent papers by Halpern and his collaborators, Parikh and his collaborators and students, van Benthem and the Amsterdam school and by several other authors as well. Applications of epistemic concepts range across game theory, economics, spatial reasoning and even social systems.

In the concurrency theory community there is very little work on this topic. Two striking examples are a recent paper by Chadha, Delaune and Kremer [CDK09] and one by Dechesne, Mousavi and Orzan [DMO07]. The former paper defines an epistemic logic for a particular process calculus, a variant of the π-calculus and uses it to reason about epistemic situations. The latter paper explores the connection between operational semantics and epistemic logic and is closer in spirit to our work which is couched in terms of labelled transition systems. Neither of these paper really integrate Hennessy-Milner logic and epistemic logic. A recent paper by Pacuit and Simon [PS11] develops a PDL-style logic for reasoning about protocols. They also prove a completeness theorem for their logic; it is perhaps the

closest in spirit to our work.

Conclusion

Despite their similar subjects, there is often a divide between concurrency theory and distributed systems theory. This is unfortunate because approaches which are effective in one area could often be applicable to the other. Epistemic logic, for example, is one of the areas where the distributed systems community got an early start [FHMV95] in the mid 1980s whereas the concurrency theory community is only just starting to use these ideas. One of the goals of this thesis is to make epistemic logic and reasoning more readily accessible to the concurrency theory community. This part of the thesis is particularly relevant to this goal, addressing it by providing a combination of epistemic logic with the Hennessy-Milner logic that the concurrency community is accustomed to using.

Part III

Knowing What You Are Doing: Epistemic Strategies for Concurrent Processes

Introduction

So far in this thesis we have presented a process calculus that includes epistemic information and a logic for reasoning about agents' knowledge in changing concurrent systems. Now we will discuss another role epistemic information plays in concurrency theory. We will present a process calculus where the agents themselves choose the actions the system takes. Traditional process calculi use the notion of a *scheduler* to resolve nondeterministic choices, but this can lead to problems arising from the scheduler's knowledge. We will show how to use *game semantics*, a powerful computation paradigm, to represent restrictions on agents' knowledge in resolving nondeterministic choices.

As we have seen, concurrent processes are a natural and widely used model of interacting agents. Process algebra combines an operational semantics for processes with equational laws of process behaviour. The most commonly used equivalence is bisimulation. There is also a modal logic which exactly characterizes bisimulation. This combination of algebraic and logical principles is powerful for reasoning about concurrency.

However, process algebra - as traditionally presented - has no explicit epistemic concepts, making it difficult to discuss what agents know and what has been successfully concealed. Epistemic concepts and indeed modal logics capturing "group knowledge" have proven very powerful in distributed systems [HM84, FHMV95], but it has taken a long time for these ideas to surface in the process algebra community.

Epistemic concepts play a striking role in the resolution of nondeterministic choices. Typically one introduces a *scheduler* (or *adversary*) to resolve nondeterminism. This scheduler represents a single global entity that re-

solves all the choices. Furthermore, traditional schedulers are effectively omniscient: they may use the entire past history as well as all other information in order to resolve the choices. This approach is reasonable when one is reasoning about correctness in the face of an unknown environment. In this case one wants a quantification over all possible schedulers in order to deliver strong guarantees about process behaviour.

In security, however, one comes across conditions where omniscient schedulers are unreasonably powerful, creating circumstances where one cannot establish security properties. The typical situation is as follows. One wants to set up protocols that *conceal* some action(s) from outside observers. If the scheduler is allowed to see these actions and reveal them through perverse scheduling decisions, there is no hope for designing a protocol that conceals the desired information. For example, randomness is often used as a way of concealing information; if the scheduler is allowed to see the results of random choices and code these outcomes through scheduling policies then randomness has no power to obfuscate data.

Consider, for instance, a voting system which collects people's votes for candidate a or b, and outputs, in some arbitrary order, the list of people who have voted – for example, to check whether everyone has voted – but is required to do so in a way that does not reveal who voted for whom. Among the possible schedulers, there is one that lists first all the people who voted for a. Clearly, this scheduler completely violates the desired anonymity property. Usually when we want a correctness property to hold for a nondeterministic system we require that it holds for *all* choices of the scheduler: there is no way such universally quantified statements will be true if we permit such omniscient schedulers.

How then is process algebra traditionally used to treat security issues? In fact scrutiny reveals that they do not have a completely demonic scheduler all the time. For example, Schneider and Sidiropoulos [SS96] argue that a system is *anonymous* if the set of (observable) traces produced by one user is the same as the set of traces produced by another user. This is, in fact, an extremely *angelic* view of the scheduler. A perverse scheduler can most definitely leak information in this case by ensuring that certain traces

never appear in one case *even though the operational semantics permits them*. Even a probabilistic (hence not overtly demonic) scheduler can leak information as discussed by Bhargava and Palamidessi[1] [BP05]. Anonymity is a problem where these issues manifest themselves particularly sharply.

Even bisimulation, a notion often used in the analysis of security properties, does not treat non-determinism in a purely demonic way. If one looks at its definition, there is an alternation of quantifiers: s is bisimilar to t if *for every* $s \xrightarrow{a} s'$ *there exists* t' such that $t \xrightarrow{a} t'$ ... This definition implies that the scheduler that chooses the a transition for s is demonic whereas the scheduler that chooses the corresponding transition for t is *angelic*.

One approach to solving the problem of reasoning about anonymity in the presence of demonic schedulers has been suggested in [CP07]: the interplay between the secret choices of the process and the choices of the scheduler is expressed by introducing two *independent* schedulers and a framework that allows one to switch between them.

The ideas of demonic versus angelic schedulers, the idea of independent agents and the presence of epistemic concepts all suggest that *games* are a unifying theme. In this part of the thesis we propose a game-based *semantic* restriction on the information flow in a concurrent process. We introduce a turn-based game that is played between two agents and define strategies for the agents. The game is played with the process as the "playing field" and the players' moves roughly represent the process executing an action. The information to which a player does not have access appears as a restriction on its allowed strategies. This is in the spirit of game semantics [AJ94a, HO00, AJM00] where restrictions on strategies are used to describe limits on what can be computed. The restrictions we discuss have an epistemic character which we model using Kripke-style indistinguishability relations.

We show that there is a particular epistemic restriction on strategies that exactly captures the syntactic restrictions developed by Chatzikokolakis and Palamidessi [CP07]. It should be noted that this correspondence is significant since it only works with one precise restriction on the strategies,

[1] They do not explicitly talk about schedulers in their paper but the import is the same.

which characterizes the knowledge of the schedulers. This restriction is an important achievement because although Chatzikokolakis and Palamidessi showed that these schedulers solve certain security problems, this is the first time that the epistemic qualities of these schedulers have been made explicit. In their paper certain equations are shown to hold and it is informally argued that these equations suggest that the desired anonymity properties hold.

The advantage to thinking in terms of strategies is that it is quite easy to capture restrictions on the knowledge of the agents as restrictions on the allowed strategies. For example, if one were to try to introduce some entirely new restriction on what schedulers "know" one would have to rethink the syntax and the operational semantics of the process calculus with schedulers and work to convince oneself that the correct concept was being captured. With strategies, one can easily add such restrictions and it is clear that the restrictions capture the intended epistemic concept. For instance, our notion of introspection makes completely manifest what the agents know since it is couched as an explicit statement of what the moves can depend on. Indeed, previously one only had an intuitive notion of what the schedulers of [CP07] "knew" and it required some careful design to come up with the right rules to capture this in the operational semantics. Thus, strategies and restrictions are a beneficial way to model interaction and independence in process algebra.

Related work

There are many kinds of games used in mathematics, logic and computer science. Games are also used widely in economics, although these are quite different from the games that we consider. Even within logic there is a remarkable variety of games. The logical games most related to our games are Lorenzen games. Lorenzen games are *dialogues* that follow certain rules about the patterns of questions and answers. There is a notion of winning and the main results concern the correspondence between winning strategies and the existence of constructive proofs. The idea of dialogue games

appears in programming language semantics culminating with the deep and fundamental results of Abramsky, Jagadeesan, Malacaria [AJM00] and Hyland and Ong [HO00] on full abstraction for PCF. These games do not have a notion of winning. Rather the games simply delineate sets of possible plays and *strategies* are used to model programs. This has been a fruitful paradigm to which many researchers - far too many to enumerate - have contributed. It has emerged that games of this kind form a semantic universe where many kinds of language features coexist. Different features are simply modelled by different conditions on the strategies.

The games that we describe are most similar to these kinds of games in spirit, but there are crucial differences. Our games are not dialogue games and there is no notion of question and answer, as a result, conditions like bracketing have no meaning in our setting. There is no notion of winning in our games either. Our games are specifically intended to model multiple agents working in a concurrent language. While there have been some connections drawn between concurrent languages like the π-calculus and dialogue games [HO00] these are results that say that π-calculus can be used to describe dialogue games, not that dialogue games can be used to model π-calculus. The latter remains a fundamental challenge and one that promises to lead to a semantic understanding of mobility.

"Innocence" is an important concept pervading game semantics [HO00, DH01]. This is a very particular restriction on what the players know. In order to define innocence much more complex structures come into play; one needs special indicators of dependence (called "justification pointers") that are used to formalize a concept called the "view" of each process. In the end innocence, like our introspection concept, is a statement about what knowledge the agents have. Our games have much less complicated structure because there are no issues with higher types and the introspection notion is relatively simple to define.

Eleven

Background

We begin by introducing a process calculus with actions labelled by an additional token and a protection operator. The labels on actions allow us to control what is visible about an action; if two actions have the same label then they are indistinguishable to an agent controlling the execution of the process. The protection operator, represented by curly brackets, indicates that the choice of the top-level action in the protected subprocess must be made independently from the choices concerning unprotected actions in the process. This idea is explained in more detail below.

We let l, j, and k represent labels, a and b actions, $\bar{a}$ and $\bar{b}$ co-actions, τ the silent action, and α and β generic actions, co-actions, or silent action. The syntax for a process is as follows:

$$P, Q ::= l : \alpha.P \mid P|Q \mid P + Q \mid (\nu a)P \mid l : \{P\} \mid 0$$

The operational semantics for this process calculus is shown in Fig. 11.1. The transition relation in the operational semantics includes both the action and the label for the action. In the case of synchronization, the labels for both synchronizing actions are included in the transition, and for the SWITCH rule, two labels are also included, one representing the fact that the protected process was chosen and one representing the action taken within the protected process. All the labels have an X or Y subscripted to them, denoting whether the label was part of a protected choice (Y) or not (X). There are corresponding right rules for $+$ and $|$; these operators are

both associative and commutative. All of the rules are analogous to those of traditional process algebra, except for the rule SWITCH, which requires that protected processes do a silent action. The reason for this restriction on the SWITCH operator is that this operator is intended to represent choices made independently from the other choices in the process. For example in the process $(l_1 : a \ + \ l_2 : b) \mid l_3 : \{k_1 : \tau \,.\, l_4 : a \ + \ k_2 : \tau \,.\, l_4 : b\}$, the left and right choices are represented as independent. This means that whatever agent controls whether the left part of the process performs an a or b action does not control how the choice on the right side of the process is resolved. This choice is resolved by an entity independent from the traditional scheduler. Therefore, we require that the protected subprocess do a silent action, because any other action would be observable to the outside world, and therefore observable to the scheduler, allowing it to base its decisions on the outcome of the protected choice, which would make this choice dependent on other choices. This independence is not a part of the operational semantics; rather, it represents the idea that the protected subprocess makes decisions independently from the main process. Furthermore, requiring the protected subprocess to do a silent action prevents synchronization between protected and unprotected parts of the process, since these two parts of the process should be independent.

$$\text{ACT}\ \frac{}{l : \alpha \,.\, P \xrightarrow[l_X]{\alpha} P} \qquad \text{RES}\ \frac{P \xrightarrow[s]{\alpha} P' \quad \alpha \neq a, \bar{a}}{(\nu a)P \xrightarrow[s]{\alpha} (\nu a)P'} \qquad \text{SUM1}\ \frac{P \xrightarrow[s]{\alpha} P'}{P + Q \xrightarrow[s]{\alpha} P'}$$

$$\text{PAR1}\ \frac{P \xrightarrow[s]{\alpha} P'}{P|Q \xrightarrow[s]{\alpha} P'|Q} \qquad \text{COM}\ \frac{P \xrightarrow[l_X]{a} P' \quad Q \xrightarrow[j_X]{\bar{a}} Q'}{P|Q \xrightarrow[(l,j)_X]{\tau} P'|Q'} \qquad \text{SWITCH}\ \frac{P \xrightarrow[j_X]{\tau} P'}{l : \{P\} \xrightarrow[l_X . j_Y]{\tau} P'}$$

Figure 11.1: Operational semantics

From now on, we will only consider *deterministically labelled* processes: processes where there can never be more than one action available with the same label.

Definition 11.0.21 (Deterministically Labelled)**.** *P is* deterministically labelled *if the following conditions hold:*

1. *It is impossible for P to make two different transitions with the same labels: for all strings s, if $P \xrightarrow[s]{\alpha} P'$ and $P \xrightarrow[s]{\beta} P''$ then $\alpha = \beta$ and $P' = P''$.*

2. *If $P \xrightarrow[l_X . j_Y]{\tau} P'$ then there is no transition $P \xrightarrow[l_X]{\alpha} P''$ for any α or P''.*

3. *If $P \xrightarrow[s]{\alpha} P'$ then P' is deterministically labelled.*

Note that any blocked[1] process is deterministically labelled, so since we only consider finite processes without recursion, this concept is well defined.

Roughly, this means that two enabled actions never have the same label. For example, $P = l : a + l : b$ is not deterministically labelled because $P \xrightarrow[l_X]{a} 0$ and $P \xrightarrow[l_X]{b} 0$ but $a \neq b$, violating the first condition. Also, $P = l_1 : a \ + \ l_1 : \{l_2 : \tau\}$ is not deterministically labelled since $P \xrightarrow[l_{1X} . l_{2Y}]{\tau} 0$ and $P \xrightarrow[l_1]{a} 0$, violating the second condition. Further, no process with this as a (reachable) subprocess is deterministically labelled. However, $l_1 : a \,.\, l_3 : b + l_2 : c \,.\, l_3 : d$ is deterministically labelled even though l_3 occurs twice, since there is no series of transitions that will result in both l_3's being available simultaneously.

Also, $l_1 : a \mid l_2 : b \,.\, l_1 : c$ is not deterministically labelled because it can transition to $l_1 : a \mid l_1 : c$ which is not deterministically labelled.

Note, however, that $l : a \,.\, P + l : a \,.\, P$ is deterministically labelled. Even though l is available twice, $l : a \,.\, P + l : a \,.\, P \xrightarrow[l]{a} P$ is the only transition available labelled with l, so P is deterministically labelled.

[1] A process is blocked if it cannot make any transition.

Twelve

Games and Strategies

In this chapter we define two-player games on deterministically labelled processes. One game is defined for each deterministically labelled process. The two players are called X and Y. The moves in the game are labels and pairs of labels. Moves represent an action being taken by the process. The player X controls all the unprotected actions, and the player Y is in charge of all the top level actions within the protected subprocesses. This makes it possible to represent the independent resolution of the two kinds of choice, by carefully defining the appropriate strategies for these games. A strategy is for one player and determines the moves the player will choose within the game. Games and strategies are both made up of *valid positions*, discussed in the next section.

12.1 Valid Positions

Valid positions are defined on a process and represent valid plays or executions for that process, with player X moving first. Every valid position is a string of moves (labels or pairs of labels from the process), each of which is assigned to a player X or Y, with player X moving first. The set of all valid positions for a process represents all possible executions of the process, including partial, unfinished executions.

Definition 12.1.1 (Move)**.** *A* move *is anything of the form* l_X, l_Y, $(l,j)_X$, *or* $(l,j)_Y$ *where* l, *and* j *are labels.* l_X *and* $(l,j)_X$ *are called* X-moves *and* l_Y *and* $(l,j)_Y$ *are called* Y-moves.

To define valid positions, we must define an extension of the transition relation.

Definition 12.1.2. *This extends the transition relation to multiple transitions, ignoring the actions for the transitions but keeping track of the labels.*

1. *For any process* P, $P \xrightarrow[\varepsilon]{} P$.
2. *If* $P \xrightarrow[s]{\alpha} P'$ *and* $P' \xrightarrow[s']{} P''$ *then* $P \xrightarrow[s.s']{} P''$.

Now we define valid positions.

Definition 12.1.3 (Valid position)**.** *If* $P \xrightarrow[s]{} P'$ *then every prefix of* s *(including* s*) is a* valid position *for* P.

In order for the set of valid positions to be prefix closed, we must explicitly include prefixes in the definition because of the SWITCH rule. For example, for the process $l:\{j:\tau\}$, the set of valid positions is $\{\varepsilon,\ l_X,\ l_X.j_Y\}$, but if the condition about prefixes were not included in the definition of valid positions, l_X would not be a valid position, because the process does not have any transition with this label alone.

Example 12.1.4. *Consider the process*

$$P = (\nu b)\left(l_1:\{k_1:\tau\,.\,l_2:a\,.\,l_3:b + k_2:\tau\,.\,l_2:c\,.\,l_3:b\} \mid l_4:\bar{b}\,.\,(l_5:d + l_6:e)\right).$$

Here are some of the valid positions for P*:*

$$l_{1X}.k_{1Y}.l_{2X}.(l_3,l_4)_X.l_{5X}$$
$$l_{1X}.k_{1Y}.l_{2X}.(l_3,l_4)_X.l_{6X}$$
$$l_{1X}.k_{2Y}.l_{2X}.(l_3,l_4)_X.l_{5X}$$
$$l_{1X}.k_{2Y}.l_{2X}.(l_3,l_4)_X.l_{6X}$$

The prefixes of these valid positions are also valid positions.

It is easy to see that the valid positions form a tree structure. The tree of valid positions will be our game tree, on which we will eventually define strategies and plays of the game.

Definition 12.1.5 (Game tree)**.** *Let V be the set of valid positions for process P. The* game tree *for P is a tree where the nodes are the valid positions for P and the edges are moves. Specifically, the root of the game tree is ε, and for a node s, the children of s are all valid positions of the form $s.m$.*

Now, for notational convenience, we define the set of children of a valid position.

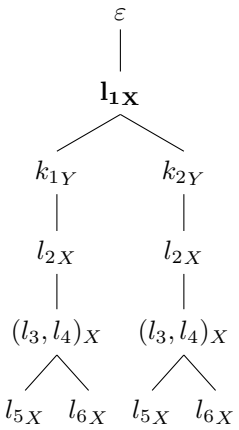
Definition 12.1.6. *Let V be the set of valid positions for a process. For $s \in V$, we define $Ch_V(s) = \{s' \in V \mid s' = s.m \text{ for some move } m\}$. If the set V is clear, we will use the notation $Ch(s)$.*

We also define a partial function $Pl : V \to \{X, Y\}$, the player whose turn it is at V.

Definition 12.1.7. *Let V be the set of valid positions for a process. For $s \in V$, $Z \in \{X, Y\}$, $Pl(s) = Z$ if and only if there is some $s' \in Ch(s)$ such that $s' = s.l_Z$. If $Pl(s) = Z$, we say that s belongs to Z.*

Note that a position can belong to at most one player, since a process never has both X and Y moves enabled at the same time. Furthermore, the leaves of the tree, where the process is blocked, do not belong to either player.

Example 12.1.8. *Here is the game tree for*
$P = (\nu b)\left(l_1 : \{k_1 : \tau \,.\, l_2 : a \,.\, l_3 : b + k_2 : \tau \,.\, l_2 : c \,.\, l_3 : b\} \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right)$.



The node in bold belongs to Y; all the other nodes except the leaves, which belong to neither player, belong to X. At each level, we write only the last move in the valid position to save space. For example, the bottom left node actually represents the valid position $l_{1X}.k_{1Y}.l_{2X}.(l_3, l_4)_X.l_{5X}$.

12.2 Strategies

A strategy for a certain player is a special subtree of the game tree. The idea behind a strategy is that it tells a player what move to make whenever it is his turn. We will only consider deterministic, complete strategies (also called functional strategies): strategies that tell the player of the strategy exactly one move to make at any possible execution of the game.

From now on, when we use m without a subscript to denote a move, it will mean a move including its player: a move of the form l_X, $(l_1, l_2)_X$, l_Y, or $(l_1, l_2)_Y$. When we use m_X, m_Y, or m_Z to denote a move, it means a move with the specified subscript, where Z represents X or Y.

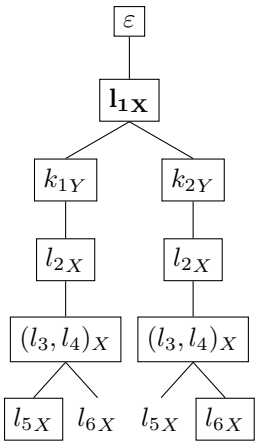
Definition 12.2.1 (Strategy)**.** *Let Z stand for either X or Y, and let $\bar{Z}$ stand for the opposite player. In the game for P, a* strategy for Z *is a subtree T of the game tree for P meeting the following three conditions:*

1. $\varepsilon \in T$
2. *If $s \in T$ and $Pl(s) = Z$, then exactly one of the children of s is in T.*

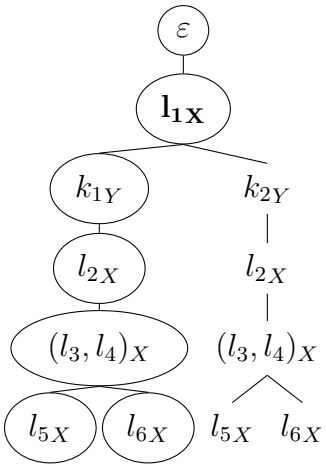
3. If $s \in T$ and $Pl(s) = \bar{Z}$, then $Ch(s) \subseteq T$.

So, a strategy for player Z is a tree where whenever it is Z's turn, all but one of the children has been pruned, but whenever it is the other player's turn all continuations are included. Thus, Z can respond to any possible move of $\bar{Z}$, and Z will always have exactly one move available when it is his turn.

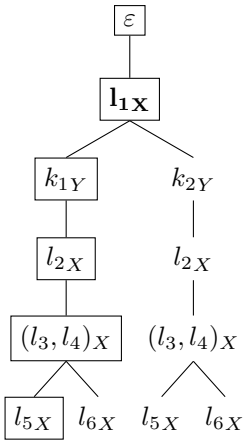
Example 12.2.2. *For*
$P = (\nu b)\left(l_1 : \{k_1 : \tau \,.\, l_2 : a \,.\, l_3 : b + k_2 : \tau \,.\, l_2 : c \,.\, l_3 : b\} \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right)$*, the boxed nodes show a subtree which is a strategy for X:*



Here, the circled nodes show a strategy for Y:

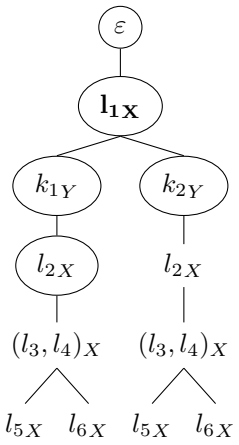


Here is a non-strategy for X:



This is not a strategy for X because it contains l_{1X} and $Pl(l_{1X}) = Y$ but it does not contain all the children of this position.

Here is an example of something that is not a strategy for Y:



This is not a Y-strategy for two reasons. First, since $Pl(l_{1X}) = Y$, this node must have exactly one child. Second, no strategy can ever exclude all the children of any node; at least one child of every node that is not a leaf must be included in the strategy.

12.3 Execution of Processes According to Strategies

In this section we define the execution of a process with two strategies- one for each player.

Proposition 12.3.1. *In the game for some process P, if S_1 is a strategy for X and S_2 is a strategy for Y, then $S_1 \cap S_2 = \{\varepsilon, m_1, m_1.m_2, ...m_1.m_2...m_k\}$ for some moves $m_1, ..., m_k$, and the valid position $m_1.m_2...m_k$ is a leaf in the game tree for P.*

Proof. First, $\varepsilon \in S_1 \cap S_2$ because every strategy contains ε.

Now we will show that for every valid position $t \in S_1 \cap S_2$, either t is a leaf in the game tree for P or there is exactly one move m such that $t.m \in S_1 \cap S_2$. This is true because if t is not a leaf, then t belongs either to X or to Y. If $Pl(t) = X$, then by definition of X-strategy, exactly one of the children of t is in S_1, and by definition of Y-strategy, all of the children of t are in S_2, so t has exactly one child in $S_1 \cap S_2$. Similarly, if $Pl(t) = Y$, then all the children of t are in S_1 and t has exactly one child in S_2, so t has exactly one child in $S_1 \cap S_2$.

Since $\varepsilon \in S_1 \cap S_2$, and every non-leaf element of $S_1 \cap S_2$ has exactly one child in $S_1 \cap S_2$ and the game tree for P is finite, $S_1 \cap S_2$ must be of the form
$\{\varepsilon, m_1, m_1.m_2, ...m_1.m_2...m_k\}$, and $m_1.m_2...m_k$ must be a leaf in the game tree for $S_1 \cap S_2$, since it has no child in $S_1 \cap S_2$. ∎

Definition 12.3.2 (Execution)**.** *Define the execution of a process P with X-strategy S_1 and Y-strategy S_2 as follows: Let s be the deepest (leaf) element in the subtree $S_1 \cap S_2$. The execution of P according to S_1 and S_2 is the sequence of processes $P, P_1, ..., P_n$ such that $s = s_1 s_2 ... s_n$ where each s_i is either a single X move of an X move followed by a Y move, and*

$$P \xrightarrow[s_1]{\alpha_1} P_1 \xrightarrow[s_2]{\alpha_2} P_2 \xrightarrow[s_3]{\alpha_3} ... \xrightarrow[s_{n-1}]{\alpha_{n-1}} P_{n-1} \xrightarrow[s_n]{\alpha_n} P_n$$

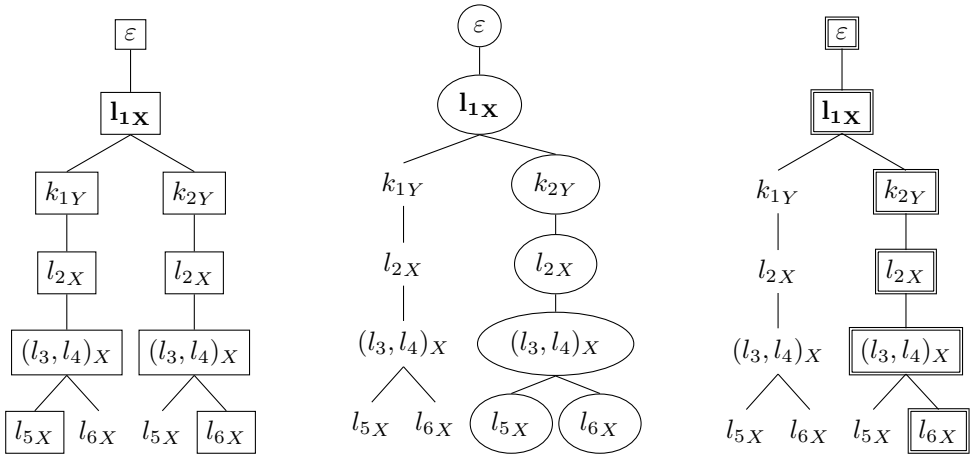
for some $\alpha_1, ..., \alpha_n$. This represents the sequence of moves that will be chosen and processes that will be reached if labels are chosen according to the strategies S_1 and S_2.

We already proved that $S_1 \cap S_2$ is of the form $\{\varepsilon, m_1, m_1.m_2, ..., m_1.m_2...m_k\}$: exactly one entire branch in the game tree. Thus, there is a unique maximal element, and it defines the execution of P with S_1 and S_2.

Example 12.3.3. *For the process discussed above,*

$$P = (\nu b)\left(l_1 : \{k_1 : \tau \,.\, l_2 : a \,.\, l_3 : b + k_2 : \tau \,.\, l_2 : c \,.\, l_3 : b\} \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right)$$

we will show the execution corresponding to the following pair of strategies, S_1 the X-strategy on the left, S_2 the Y-strategy in the middle, and the intersection on the right:



The maximal element of $S_1 \cap S_2$ is the position $l_{1X}.k_{2Y}.l_{2X}.(l_3, l_4)_X.l_{6X}$. This gives the execution

$$(\nu b)\left(l_1 : \{k_1 : \tau \,.\, l_2 : a \,.\, l_3 : b + k_2 : \tau \,.\, l_2 : c \,.\, l_3 : b\} \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right) \xrightarrow[l_{1X}.k_{2Y}]{\tau}$$
$$(\nu b)\left((l_2 : c \,.\, l_3 : b) \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right) \xrightarrow[l_{2X}]{c} (\nu b)\left(l_3 : b \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right)$$
$$\xrightarrow[(l_3, l_4)_X]{\tau} (\nu b)\,(l_5 : d + l_6 : e) \xrightarrow[l_{6X}]{e} 0$$

This example shows why, in the definition of the execution, we set $s = s_1 s_2 ... s_n$ where each s_i is either a single X move of an X move followed by

a Y move. In the first step of the execution, l_{1X} and k_{2Y} together define one transition for the process. Neither a switch move nor a Y-move alone gives a process transition according to the operational semantics; the two must be combined to produce a single transition.

12.4 Epistemic Restrictions on Strategies

Now that we have shown how properly specified strategies determine the execution of a process, we can consider epistemic restrictions on strategies, representing agents' actions when their knowledge is limited. In general, we impose epistemic conditions on strategies first by determining what knowledge is appropriate for each agent, that is, which sets of executions should be indistinguishable for him, in the form of an equivalence relation on valid positions. Once the correct notion of the agent's knowledge is determined, we can define strategies that respect that condition.

Definition 12.4.1. *Given an equivalence relation $E \subseteq V \times V$, we say that a strategy T* respects *E* for player *Z if for all $s_1, s_2 \in T$, if $(s_1, s_2) \in E$ and $Pl(s_1) = Pl(s_2) = Z$, then for every move m, $s_1.m \in T$ if and only if $s_2.m \in T$. We call this an* epistemic restriction.

In other words, Z must choose the same move whether s_1 or s_2 describes the execution of the process so far, because it does not know whether s_1 or s_2 has occurred- they are indistinguishable for him. Note that we quantify only over the player's own positions; all children of the other player's positions must be in the strategy, as always.

For example, we could require that an agent only have knowledge of his own past moves, or only know what moves are currently available to him, or only remember his past three moves. In order to formalize these epistemic restrictions on strategies, we need the following subsidiary definitions:

Definition 12.4.2. *Let V denote the set of valid positions for a process P. If s is a valid position for P, $enabled(s)$ represents the set of moves available after s: define $enabled(s) = \{m \mid s.m \in V\}$. Also define the X and Y*

moves available after s as, respectively, $enabled_X(s) = \{m_X \mid s.m_X \in V\}$ and $enabled_Y(s) = \{m_Y \mid s.m_Y \in V\}$.

Definition 12.4.3. *If s is a valid position for P and Z is a player, let $\bar{Z}$ denote the other player. We define $Z(s)$, the string of Z moves in s, inductively as follows:*

1. $Z(\varepsilon) = \varepsilon$.
2. $Z(s.m_Z) = Z(s).m_Z$.
3. $Z(s.m_{\bar{Z}}) = Z(s)$.

Now we can formally define the epistemic restriction for an agent only remembering his own past moves. In this case, it is useful to define an equivalence relation for each agent.

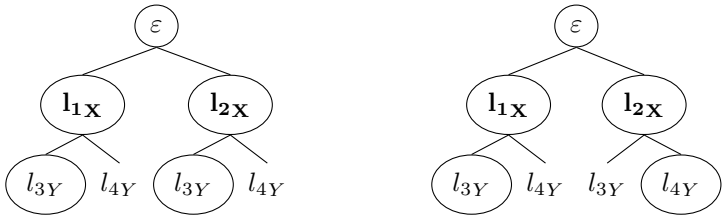
Definition 12.4.4. *We will define the equivalence relation H_Z as $H_Z = \{(s_1, s_2) \mid Z(s_1) = Z(s_2)\}$.*

In a strategy that respects this condition for its player, the player responds the same way no matter what the other player does, because it does not have knowledge of the other player's actions.

Example 12.4.5. *In the following process, for readability, we replace labels with superscript numbers preceding actions: ${}^1a.P$ represents $l_1 : a \,.\, P$. As a simple example, consider the process*

$$P = {}^1\{\, {}^3\tau + {}^4\tau\} + {}^2\{\, {}^3\tau + {}^4\tau\}$$

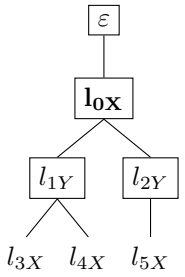
the Y-strategy on the left respects H_Y, but the Y-strategy on the right does not:



The second strategy does not respect H_Y because $Y(l_{1X}) = Y(l_{2X}) = \varepsilon$, so $(l_{1X}, l_{2X}) \in H_Y$, and both these positions are in the strategy and belong to Y, so they should be indistinguishable to Y and have the same continuation, but they do not.

Note that in for some equivalence relations, for certain processes there are no strategies respecting the equivalence relation. This occurs if there are two indistinguishable positions that do not have any enabled moves in common. Here is a simple example of a process where no X-strategy respects H_X, the equivalence based on X's past actions.

Example 12.4.6. *For the process ${}^0\{\ {}^1\tau\,.\,({}^3a +^4 b) +^2 \tau\,.\,{}^5a\}$, with the game tree below, there is no X-strategy respecting H_X. Any X-strategy must contain the boxed nodes by definition, since it must contain exactly one child of every X position and all children of every Y position. But $X(l_{0X}.l_{1Y}) = X(l_{0X}.l_{2Y}) = l_{0X}$, so $(l_{0X}.l_{1Y}, l_{0X}.l_{2Y}) \in H_X$ and these two positions must contain the same continuations in the strategy. However, $enabled(l_{0X}.l_{1Y}) \cap enabled(l_{0X}.l_{2Y}) = \emptyset$, so there is no possible strategy respecting this epistemic restriction.*



Although some epistemic restrictions cannot be respected on certain processes, some epistemic restrictions can be respected on any process. For equivalence relation E, if $(s_1, s_2) \in E \Rightarrow enabled(s_1) = enabled(s_2)$, then it is evident that for any process there is a strategy respecting E.

Example 12.4.7. *We can require that an agent only know what moves are currently available to him. We will call this equivalence relation Av_Z: $(s_1, s_2) \in Av_Z \Leftrightarrow enabled_Z(s_1) = enabled_Z(s_2)$. As discussed above, for any process it will be possible to find a strategy that respects this condition.*

We now single out a very important epistemic restriction, called introspection. An introspective strategy allows a player to "remember" not only his own history of moves, but also the moves that were available to him at every point in the past, including the current step. Introspective strategies are important because they exactly capture the intended independence requirement for the protection operator.

Definition 12.4.8. *For player Z, positions s_1 and s_2 are called* introspectively Z-equivalent*, denoted $(s_1, s_2) \in I_Z$, if they satisfy the following conditions:*

1. $Pl(s_1) = Pl(s_2) = Z$
2. $Z(s_1) = Z(s_2)$
3. $enabled_Z(s_1) = enabled_Z(s_2)$.
4. *For all prefixes s_1' of s_1 and s_2' of s_2, if $Pl(s_1') = Pl(s_2') = Z$ and $Z(s_1') = Z(s_2')$, then $enabled_Z(s_1') = enabled_Z(s_2')$.*

In this definition, two positions are indistinguishable if the player made the same series of moves to arrive at both positions, and at any point in the past where it had made a certain series of moves in both positions and had moves available, it had the same set of moves available in both positions.

The introspection condition corresponds to perfect recall of the moves that an agent made as well as the moves that it could have made but did not. However, it is not aware of opponent moves except insofar as such moves determine its own choices. One can imagine restrictions where an agent has only the ability to recall a bounded amount of its past history, but these type of restrictions are not relevant to the particular situation in which we are interested.

For the rest of this section, we will only discuss the introspective equivalence condition, so when we say that two positions are indistinguishable for Z, we mean that the are introspectively Z-equivalent.

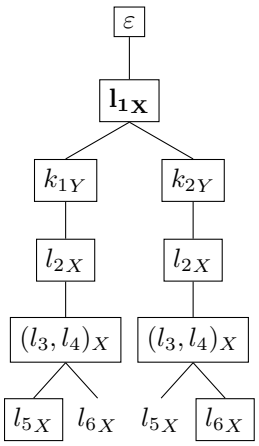
Definition 12.4.9 (Introspection)**.** *Given a process P, and S a strategy for player Z on P, S is* introspective *if it respects the introspection equivalence relation for Z.*

In other words, the player chooses the move it makes at each step based on his past moves, the moves that are available to him, and the moves that were available to him at each point in the past. If these conditions are all the same at two positions, the player cannot distinguish them, so it makes the same move at both positions.

Example 12.4.10. *For*

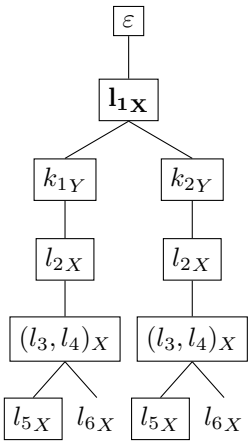
$$P = (\nu b)\left(l_1 : \{k_1 : \tau \,.\, l_2 : a \,.\, l_3 : b + k_2 : \tau \,.\, l_2 : c \,.\, l_3 : b\} \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right)$$

the strategy given above for X,



is not introspective. This is because in order to satisfy the introspection condition, $l_{1X}.k_{1Y}.l_{2X}.(l_3, l_4)_X$ and $l_{1X}.k_{2Y}.l_{2X}.(l_3, l_4)_X$ should have the same moves appended to them in S, since they are X indistinguishable. However,
$l_{1X}.k_{1Y}.l_{2X}.(l_3, l_4)_X.l_{5X} \in S$ and $l_{1X}.k_{2Y}.l_{2X}.(l_3, l_4)_X.l_{5X} \notin S$, and similarly, $l_{1X}.k_{2Y}.l_{2X}.(l_3, l_4)_X.l_{6X} \in S$ and $l_{1X}.k_{2Y}.l_{2X}.(l_3, l_4)_X.l_{5X} \notin S$.

An example of an introspective strategy for X is this:

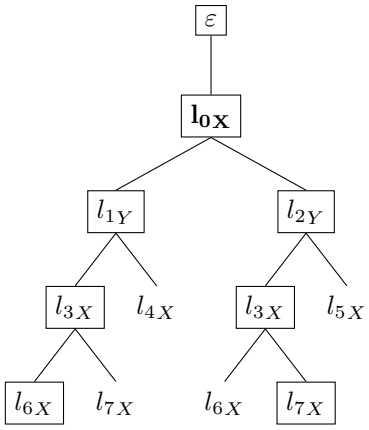


Here is an example showing why the prefixes of the valid positions are discussed in the definition of introspective. For readability, labels are replaced with superscript numbers preceding actions: ${}^1a.P$ represents $l_1 : a \,.\, P$.

Example 12.4.11. *Consider*

$$P = {}^0\{\, {}^1\tau \,.(\, {}^3c \,.(\, {}^6f + {}^7g\,) + {}^4d\,) + {}^2\tau \,.(\, {}^3c \,.(\, {}^6f + {}^7g\,) + {}^5e\,)\}.$$

Let X's strategy be the boxed nodes:



This strategy is introspective. Even though $X(l_{0X}.l_{1Y}.l_{3X}) = X(l_{0X}.l_{2Y}.l_{3X})$ *and*
$enabled_X(l_{0X}.l_{1Y}.l_{3X}) = enabled_X(l_{0X}.l_{2Y}.l_{3X})$*, it is acceptable that the two strings have different moves appended to them, because* $enabled_X(l_{0X}.l_{1Y}) = \{l_{3X}, l_{4X}\}$ *and* $enabled_X(l_{0X}.l_{2Y}) = \{l_{3X}, l_{5X}\}$*. This can be thought of*

as X being able to distinguish between the two positions $l_{0X}.l_{1Y}.l_{3X}$ and $l_{0X}.l_{2Y}.l_{3X}$ because it remembers what moves were available to him earlier and is able to use this information to tell apart the two positions.

The essence of the introspection condition is that a player knows what moves it has made in the past and knows what moves, if any, were available to it at each point in the past, but cannot see any moves that its opponent has made. Thus, each player must choose its moves based solely on its own past moves, the past moves that were available to it, and the moves available to it now.

Thirteen

Correspondence between Strategies and Schedulers

In this chapter, we first review the syntactic schedulers defined in [CP10a] and then prove that introspective strategies correspond exactly to these schedulers. This result is important because these schedulers are defined purely syntactically, without any explicit reference to knowledge or equivalence between executions. Since the players' knowledge is explicit in the definition of introspective strategies, this equivalence explains the knowledge requirements underlying the syntactic schedulers, which had not been discussed before.

13.1 Background on Schedulers

The process calculus with schedulers uses the syntax for processes discussed above, with the protection operator, but also adds a new ingredient: explicit syntax for a pair of independent schedulers. The schedulers use labels, rather than actions, to interact with a process, making it possible to use labels to control a scheduler's "view" of a process. The schedulers choose a sequence of labels, to execute actions, or pairs of labels, to synchronize processes, and also can check whether a label or synchronization is available, using an if... then... else... construct. The two schedulers operate inde-

pendently and do not communicate with one another, and each scheduler controls certain choices in the process. This makes it possible to represent independent choices in the process calculus. A *complete process* is an ordinary process augmented with a pair of schedulers. In this section, we also add the notion of *general labels*, either a single ordinary label or a pair of ordinary labels. This convention is useful because an ordinary label and a pair of synchronizing ordinary labels both represent a single action by a scheduler. We let l and k represent ordinary labels and L and K represent general labels. The notations $\sigma(L)$, $\sigma(l)$, and $\sigma(l,k)$ are used to designate a choice made by a scheduler: $\sigma(l)$ means a single action will be executed, $\sigma(l,k)$ means that the scheduler will synchronize two actions, and $\sigma(L)$ can represent either of these cases. We let a and b represent actions, $\bar{a}$ and $\bar{b}$ co-actions, τ the silent action, α and β generic actions, co-actions, or silent action, P and Q processes, and ρ and η schedulers. The syntax for a complete process is as follows:

$$\begin{aligned} P,Q &::= l:\alpha.P \mid P|Q \mid P+Q \mid (\nu a)P \mid l:\{P\} \mid 0 \\ L &::= l \mid (l,k) \\ \rho,\eta &::= \sigma(L).\rho \mid \textbf{if } L \textbf{ then } \rho \textbf{ else } \eta \mid 0 \\ CP &::= P \parallel \rho,\eta \end{aligned}$$

The first scheduler is called the primary scheduler and the second scheduler is the secondary scheduler.

The rules for the operational semantics of the process calculus with schedulers are in Fig. 13.1. Using the **if then else** construct (rules IF1, IF2), the scheduler can check whether a move is available and choose what to do based on that information. The SWITCH rule says that the curly brackets indicate a point where the secondary scheduler makes the next choice. After making this choice, control reverts to the primary scheduler. The choice made by the secondary scheduler must result in a τ observation because the process is encapsulated and cannot interact with the environment at this point. Of course, once control reverts to the primary scheduler, interactions with the external environment can indeed take place. The order

$$\text{ACT}\ \frac{}{l : \alpha.P \parallel \sigma(l).\rho, \eta \xrightarrow[l_X]{\alpha} P \parallel \rho, \eta}$$

$$\text{RES}\ \frac{P \parallel \rho, \eta \xrightarrow[s]{\alpha} P' \parallel \rho', \eta' \quad \alpha \neq a, \bar{a}}{(\nu a)P \parallel \rho, \eta \xrightarrow[s]{\alpha} (\nu a)P' \parallel \rho', \eta'}$$

$$\text{SUM1}\ \frac{P \parallel \rho, \eta \xrightarrow[s]{\alpha} P' \parallel \rho', \eta' \qquad \rho \neq \textbf{if } L \textbf{ then } \rho_1 \textbf{ else } \rho_2}{P + Q \parallel \rho, \eta \xrightarrow[s]{\alpha} P' \parallel \rho', \eta'}$$

$$\text{PAR1}\ \frac{P \parallel \rho, \eta \xrightarrow[s]{\alpha} P' \parallel \rho', \eta' \qquad \rho \neq \textbf{if } L \textbf{ then } \rho_1 \textbf{ else } \rho_2}{P|Q \parallel \rho, \eta \xrightarrow[s]{\alpha} P'|Q \parallel \rho', \eta'}$$

$$\text{SWITCH}\ \frac{P \parallel \eta, 0 \xrightarrow[j_X]{\tau} P'||\eta', 0}{l : \{P\} \parallel \sigma(l).\rho, \eta \xrightarrow[l_X.j_Y]{\tau} P' \parallel \rho, \eta'}$$

$$\text{COM}\ \frac{P \parallel \sigma(l).0, 0 \xrightarrow[l_X]{a} P' \parallel 0, 0 \qquad Q \parallel \sigma(j).0, 0 \xrightarrow[j_X]{\bar{a}} Q' \parallel 0, 0}{P|Q \parallel \sigma(l, j).\rho, \eta \xrightarrow[(l,j)_X]{\tau} P'|Q' \parallel \rho, \eta}$$

$$\text{IF1}\ \frac{P \parallel \rho_1, \eta \xrightarrow[s]{\alpha} P' \parallel \rho_1', \eta' \quad P \parallel \sigma(L).0, \theta \xrightarrow[s']{\beta} P'' \parallel 0, \theta' \quad \text{for some scheduler } \theta}{P \parallel \textbf{if } L \textbf{ then } \rho_1 \textbf{ else } \rho_2, \eta \xrightarrow[s]{\alpha} P' \parallel \rho_1', \eta'}$$

$$\text{IF2}\ \frac{P \parallel \rho_2, \eta \xrightarrow[s]{\alpha} P' \parallel \rho_2', \eta' \quad P \parallel \sigma(L).0, \theta \not\rightarrow \quad \text{for all schedulers } \theta}{P \parallel \textbf{if } L \textbf{ then } \rho_1 \textbf{ else } \rho_2, \eta \xrightarrow[s]{\alpha} P' \parallel \rho_2', \eta'}$$

Figure 13.1: Operational semantics for processes with schedulers

in which the schedulers are written indicates which one is to be regarded as primary. In the rules SUM1 and PAR1, we require that the primary scheduler not be of the form **if** L **then** ρ_1 **else** ρ_2 because the **if then else** construct allows a scheduler to check whether a label is available. Thus, the behaviour of a process P with primary scheduler **if** L **then** ρ_1 **else** ρ_2 may be different than the behaviour of process $P + Q$ with the same scheduler if the label L is available in process Q. The same condition applies to PAR1. The rules IF1 and IF2 check whether a process can execute any transition with the one step primary scheduler $\sigma(L)$ and any secondary scheduler. If there is any transition that can occur for this complete process, then the first branch of the primary scheduler is activated, otherwise, the second

branch occurs.

Clearly, if a process is blocked, then no transition is possible with any schedulers. On the other hand, it is possible for a process that is not blocked to have no transitions available with certain schedulers. For example, the process $l\!:\!a$ is not blocked, but no transitions are available for the complete process $l : a \parallel \sigma(j), 0$. Thus, it is useful to define the notion of a pair of schedulers being nonblocking for a certain process.

Definition 13.1.1 (Nonblocking Schedulers)**.** *For a process P which is not blocked, a pair of schedulers ρ, η are inductively defined as* nonblocking *if $P \parallel \rho, \eta \xrightarrow{\alpha} P' \parallel \rho', \eta'$ for some α, P', ρ', and η', and if P is not blocked, then ρ' and η' are non-blocking for P'.*

Since we consider only finite processes, this inductive definition characterizes all nonblocking scheduler pairs for processes that are not blocked.

We have defined a nonblocking scheduler pair as, essentially, a pair of schedulers that choose a move for the process whenever one is available. Now we define the concept of a single scheduler being nonblocking. We would like to say that a single primary or secondary scheduler for a process is nonblocking if it can be paired with any nonblocking secondary or primary scheduler (respectively) for the process and not cause the process to be blocked. Obviously, this would be a circular definition, so we define nonblocking first inductively for a secondary scheduler, and then for a primary scheduler, with reference to nonblocking secondary schedulers.

Definition 13.1.2. *If P is a deterministically labelled process and is not blocked, then a scheduler η is a* nonblocking secondary *scheduler for P if for every general label L such that for some η_1,*

$$P \parallel \sigma(L), \eta_1 \xrightarrow[s]{\alpha} P' \parallel 0, \eta_1'$$

(for some α, s, P', and η_1'), then

$$P \parallel \sigma(L), \eta \xrightarrow[s']{\beta} P'' \parallel 0, \eta'$$

(for some β, s', P'' and η'), and if P'' is not blocked, η' is a nonblocking secondary scheduler for P''.

If P is blocked, then any secondary scheduler is defined to be nonblocking.

First, note that this is a complete inductive definition because we only consider finite processes, so any process will be blocked after some finite number of steps. The meaning of this definition is the following: if there is a label that can be chosen by the primary scheduler and execute an action in conjunction with some arbitrary secondary scheduler, then a nonblocking secondary scheduler must also be able to execute an action in conjunction with the primary scheduler that chooses this label.

For a blocked process, all schedulers are considered to be nonblocking because it is not the scheduler that is preventing an action from occurring, but the process itself, so the scheduler is nonblocking.

Definition 13.1.3. *If P is a deterministically labelled process that is not blocked, then primary scheduler ρ is* primary nonblocking *if for any nonblocking secondary scheduler η,*

$$P \parallel \rho, \eta \xrightarrow[s]{\alpha} P' \parallel \rho', \eta'$$

(for some α, s, P', ρ', η') and if P' is not blocked, then ρ' is a nonblocking primary scheduler for P'.

In other words, a primary scheduler is one that will schedule an action for the process no matter what nonblocking secondary scheduler it is paired with.

13.2 Correspondence Theorem

The main correspondence theorem can now be stated.

Theorem 13.2.1. *Given a deterministically labelled process P, a nonblocking primary scheduler ρ for P, and a nonblocking secondary scheduler η for P, there is an introspective X strategy S depending only on P and ρ, and an introspective Y strategy T depending only on P and η, such that the execution of $P \parallel \rho, \eta$ is identical to the execution of P with S and T.*

Furthermore, given a deterministically labelled process P, an introspective X strategy S for P, and an introspective Y strategy T for P, there is a nonblocking primary scheduler ρ depending only on S and P and a nonblocking secondary scheduler η depending only on T and P such that the execution of P with S and T is identical to the execution of $P \parallel \rho, \eta$.

Before we discuss the proof we make some observations on the quantifier structure of the statement of the theorem. One could imagine stating the first part as follows:

$$\forall P, \rho \exists S \text{ s.t. } \forall \eta \exists T \ldots$$

This is *apparently* stronger and certainly clearer than the original version which uses the clumsy phrase "depending only on..." However, this is not the case; it is actually weaker. The "new improved" version allows T to depend on ρ, which the version stated in the theorem does not allow. There is in fact a formal logic called "Independence Friendly" (IF) logic which allows quantifiers to be introduced with independence statements; this is just what the version in the statement of the theorem does, without, of course, dragging in all the formal apparatus of IF logic. In fact, it can be proved that there are statements of IF logic than cannot be rendered in ordinary first-order logic; the statement of the theorem is an example.

In order to prove the theorem we need this definition:

Definition 13.2.2. *A move l in process P is called a* switch *move if it chooses a label of the form $l : \{P'\}$ in P. Otherwise, it is called an* ordinary *move.*

Proof. There are several steps involved in the proof, so we begin by providing an outline.

1. We prove that every scheduler has an equivalent introspective strategy, in the following way
   a) We provide a translation from a scheduler to a strategy
   b) We prove that the translation does indeed yield a strategy

c) We prove that the strategy is equivalent to the scheduler

d) We prove that the strategy is introspective

2. We prove that every introspective strategy has an equivalent scheduler in the following steps

a) We provide a translation from a strategy to a scheduler

b) We prove that the scheduler is equivalent to the strategy

c) We prove that the translation yields a nonblocking scheduler

Translation from a scheduler to a strategy

We will give a procedure that takes a scheduler and returns a strategy. It is an inductive procedure so it also has an argument keeping track of where it is in the tree of valid positions. Thus, for scheduler ρ, $Strat(\rho, \varepsilon)$ is the corresponding strategy.

Note that the translation only works with respect to a specific process. It must take the tree of valid positions into consideration. Z stands for X if it is a primary scheduler and Y if it is a secondary scheduler. Let s_Z denote the position s where $Pl(s) = Z$, and let $s_{\bar{Z}}$ be the position s where $Pl(s) = \bar{Z}$, and s_l denote the position s where s is a leaf.

$$
\begin{aligned}
Strat(\sigma(l).\rho, s_Z) &= \{s_Z\} \cup Strat(\rho, s_Z.l_Z) \\
Strat(\textbf{if } l \textbf{ then } \rho_1 \textbf{ else } \rho_2, s_Z) &= \begin{cases} Strat(\rho_1, s_Z) & \text{if } s_Z.l_Z \in Ch(s_Z) \\ Strat(\rho_2, s_Z) & \text{otherwise} \end{cases} \\
Strat(\rho, s_{\bar{Z}}) &= \{s_{\bar{Z}}\} \cup \bigcup_{s' \in Ch(s_{\bar{Z}})} Strat(\rho, s') \\
Strat(\rho, s_l) &= \{s_l\}
\end{aligned}
$$

The case for $Strat(0, s_Z)$ is not defined because we assume nonblocking schedulers, so they will always schedule an action when it is Z's turn, and therefore the scheduler 0 cannot occur at a position belonging to Z.

Now, note that $s \in Strat(\rho, s)$, for any ρ and any s. This is true because the only case where s is not specifically added to the strategy is

$Strat($**if** l **then** ρ_1 **else** $\rho_2, s_Z)$. But this is equal to either $Strat(\rho_1, s_Z)$ or $Strat(\rho_2, s_Z)$, so eventually s_Z will be added to the strategy.

Proof that the translation yields a strategy

In order to prove that this translation yields a strategy, we must check that for any nonblocking scheduler ρ, $Strat(\rho, \varepsilon)$ contains ε, contains exactly one child of every Z position in $Strat(\rho, \varepsilon)$ and contains every child of any $\bar{Z}$ position in $Strat(\rho, \varepsilon)$. We already showed that $Strat(\rho, \varepsilon)$ contains ε. And every time the algorithm encounters a $\bar{Z}$ position, it adds all its children to the strategy, since it adds $Strat(\rho, s')$ to the strategy for each child s', and $s' \in Strat(\rho, s')$. Finally, every time the translation encounters a Z position, it adds the strategy for exactly one child of this position and the corresponding subscheduler. Thus, this child will be added to the strategy, and there is no way for any other child of this position to be added to the strategy.

Proof that the strategy is equivalent to the scheduler

Now we show that the strategy given by the translation is equivalent to the scheduler, in the sense that given process P, if S is the translation of ρ and T is the translation of η, then the execution of P with S and T is identical to the execution of $P \parallel \rho, \eta$. Since we have shown that the procedure does indeed produce a strategy, it is straightforward to see that it is correct. At any position where it is Z's turn, the function has two choices: first, it can go to the child in the game tree which is required by the scheduler, meaning that this position will be added to the strategy at the next step. The other option is testing an if statement and applying the proper sub scheduler at the current position in the game tree. Since the schedulers and game trees are finite, it is clear that this gives the correct strategy in the end.

Proof that the strategy is introspective

Assume that $(s_1, s_2) \in I_Z$, and s_1 and s_2 are in $Strat(\rho, \varepsilon)$. We will prove that $s_1.m \in Strat(\rho, \varepsilon)$ if and only if $s_2.m \in Strat(\rho, \varepsilon)$. We must also prove by induction on the number of Z-moves in s_1 that in calculating

$Strat(\rho, \varepsilon)$, for all schedulers ρ', $Strat(\rho', s_1)$ will be reached as a subcase of the recursive definition of $Strat(\rho, \varepsilon)$ iff $Strat(\rho', s_2)$ be reached as a subcase of the recursive definition of $Strat(\rho, \varepsilon)$.

Base Case: s_1 has 0 Z-moves. So s_1 is a string of 0 or more $\bar{Z}$-moves, and s_2 must also be a string of 0 or more $\bar{Z}$-moves. It is easy to see that $Strat(\rho, s_1)$ and $Strat(\rho, s_2)$ will both be called, since $Strat(\rho, s_{\bar{Z}})$ just calls $Strat(\rho, s')$ for children of $s_{\bar{Z}}$, without changing ρ, until $Strat(\rho, s_1)$ and $Strat(\rho, s_2)$ are both added to the strategy. At this point, if ρ is of the form $\sigma(l).\rho'$, then $Strat(\rho', s_1.l_Z)$ and $Strat(\rho', s_2.l_Z)$ will be called. On the other hand, ρ could be of the form **if** l **then** ρ_1 **else** ρ_2. But we know that $(s_1, s_2) \in I_Z$, so $s_1.l_z \in Ch(s_1)$ iff $s_2.l_Z \in Ch(s_2)$. Thus, for ρ_i either ρ_1 or ρ_2, $Strat(\rho_i, s_1)$ will be called iff $Strat(\rho_i, s_2)$ is called. Furthermore, this will be repeated until the function has gone through all the "**if** ... **then** ... **else** ..." statements, and reached a scheduler of the form $\sigma(l).\rho'$, and the same scheduler will always be called for both s_1 and s_2.

Induction Step: s_1 has n Z-moves, and therefore s_2 also has n Z-moves. Thus, $s_1 = s_1'.t_1$, and $s_2 = s_2'.t_2$, where $(s_1', s_2') \in I_Z$ and t_1 and t_2 are both strings of 0 or more $\bar{Z}$ moves. So, by the induction hypothesis, s_1' and s_2' were added to the strategy by the recursive definition $Strat$ eventually reaching two subcases of the form $Strat(\rho', s_1')$ and $Strat(\rho', s_2')$ for the same sub scheduler ρ'. After this point, the same thing occurs as in the induction hypothesis when the positions belong to $\bar{Z}$, and the recursive definition eventually reaches the point $Strat(\rho', s_1)$ and $Strat(\rho', s_2)$ and as in the base case, the same move must be added to the strategy as a continuation of both s_1 and s_2. Thus, after any two introspectively equivalent positions, the same move is added, so the strategy is introspective.

Translation from a strategy to a scheduler

Now we give a procedure to get a scheduler corresponding to an introspective strategy. Let P be a deterministically labelled process, S a strategy for player Z, and V the set of valid positions for P.

First we introduce a new piece notation in schedulers which is an encoding of a more complicated scheduler term.

Consider the set of all labels in process P, $l_1, ..., l_k$. We want to encode an "if" statement that checks whether exactly a certain subset of moves is enabled, and no others. Logically, we want to encode a statement along the lines of "If $(\bigwedge_{i\in I} l_i \wedge \bigwedge_{i\notin I} \neg l_i)$ then ρ_1 else ρ_2."

First note that we can encode "If $(l_1 \wedge l_2)$ then ρ_1 else ρ_2" as **If** l_1 **then** (**If** l_2 **then** ρ_1 **else** ρ_2) **else** ρ_2. It is easy to see that the second scheduler is equivalent to the intuitive meaning of the first one.

Similarly, we can encode "If $\neg l$ then ρ_1 else ρ_2" as **If** l **then** ρ_2 **else** ρ_1.

Finally, we can encode "If $l_1 \wedge \neg l_2$ then ρ_1 else ρ_2" as **if** l_1 **then** (**if** l_2 **then** ρ_2 **else** ρ_1) **else** ρ_2. We can combine an arbitrary number of conjunctions of labels and negations of labels in the same way.

If the set of labels for a process is L, we will use the notation **if** $= L_1$ **then** ρ_1 **else** ρ_2 for the scheduler that executes ρ_1 if exactly the set of moves L_1 is enabled, and none of the moves in $L \backslash L_1$ are enabled, and executes ρ_2 otherwise.

Now we can give the procedure for translating a strategy to a scheduler. The idea is, roughly, that for strategy S, we have a recursive function ρ_S that takes a set of introspectively equivalent valid positions as its input and gives the scheduler corresponding to the strategy's behavior on that set of valid positions. Then $\rho_S(\{\varepsilon\}$ will be the scheduler corresponding to the strategy's behavior starting from beginning of the process. We need several subsidiary definitions in order to give the function.

Definition 13.2.3. *For $R \subseteq V$, define*

$$ext_Z(R) = \{r.s \in V \mid r \in R, Z(s) = \varepsilon \text{ and } Pl(r.s) = Z\}.$$

This is the set of descendants of elements of R that are the first descendants where it is Z's turn. This function is useful because the scheduler only acts when it is Z's turn, so it allows us to skip forward to the next part of the strategy where we will have to define the corresponding scheduler.

Definition 13.2.4. *$ext_Z(R)/I_Z$ is the quotient of $ext_Z(R)$ by the introspective equivalence relation.*

R will be a set of introspectively equivalent positions, but $ext_Z(R)$ may extend elements of R to positions that are in different equivalence classes. The scheduler can distinguish between these classes and can act differently on each class, corresponding to the strategy.

Definition 13.2.5. *If R is a set of introspectively equivalent valid positions, define $en(R)$ as enabled(s) where $s \in R$. Since all the positions in R are introspectively equivalent, they all have the same set of enabled moves, so this definition is consistent.*

This definition will be used to allow the scheduler to distinguish between different equivalence classes of valid positions at a certain point in the execution, using the scheduler construction discussed above.

Definition 13.2.6. *Let S be an introspective strategy for Z and let A be a set of introspectively equivalent valid positions. If $S \cap A \neq \emptyset$, define $mv_S(A)$ as the move m such that $s \in A$ and $s.m \in S$. This is a consistent definition since all introspectively equivalent positions must be followed by the same move in an introspective strategy.*

We use this definition to define the move that the scheduler schedules for a given equivalence class.

We need one more piece of notation.

Definition 13.2.7. *If R is a set of introspectively equivalent positions and $m \in en(R)$, then define $R \odot m$ as $\{r.m \mid r \in R\}$. Note that if $R \subseteq V$ and $m \in en(R)$ then $R \odot m \subseteq V$.*

Finally, here is the recursive function Sch_S that turns a strategy S into a scheduler, $Sch_S(\{\varepsilon\})$.

$$Sch_S(R) = \begin{cases} 0 & \text{If } ext_z(R) = \emptyset \\ \textbf{if } = en(R_1) \textbf{ then } \sigma(mv(R_1)).Sch_S(R_1 \odot mv(R_1)) \textbf{ else} & \\ \quad \textbf{if } = en(R_2) \textbf{ then } \sigma(mv(R_2)).Sch_S(R_2 \odot mv(R_2)) \textbf{ else} & \\ \quad\quad ... & \\ \quad\quad\quad \textbf{if } = en(R_{k-1}) \textbf{ then } \sigma(mv(R_{k-1})).Sch_S(R_{k-1} \odot mv(R_{k-1})) & \\ \quad\quad\quad\quad \textbf{else } \sigma(mv(R_k)).Sch_S(R_k \odot mv(R_k)) & \text{Otherwise} \end{cases}$$

where $ext_Z(R)/I_Z = \{R_1, R_2, ..., R_k\}$.

Proof that the scheduler is equivalent to the strategy

A formal proof of the correctness would be tedious, so we just provide an argument in words. We must show that the execution of the process P with any X-strategy S and any Y-strategy T is the same as the execution of $P \parallel Sch_S(\{\varepsilon\}), Sch_T(\{\varepsilon\})$.

First, note that when we start out with $Sch_S(\{\varepsilon\})$, any time there is a recursive call to the function $Sch_S(R)$, R will be a set of introspectively equivalent valid positions. This would be easy to prove by induction, since in the case where there are recursive calls to the Sch_S function, it is always after quotienting the set $ext_Z(R)$ by I_Z, the introspective equivalence relation, and the argument to the function is an equivalence class.

The scheduler is correct because at each step, the function takes all the continuations of all the elements of the equivalence class where it was last Z's turn. This set is divided into equivalence classes based on the introspective equivalence relation. For each equivalence class R, we add an if clause to the scheduler, so that this clause will only be true in the equivalence class R and not in any other equivalence class. Inside each if clause, the correct move according to the strategy is scheduled ($\sigma(mv(R))$) and then the correct scheduler is recursively computed as the continuation after this move. On the other hand, if $ext_Z(R) = \emptyset$, then the corresponding scheduler is 0, because this means there are no continuations of any position in R where it is Z's turn again. Thus, the scheduler should not schedule any further actions.

Proof that the scheduler is nonblocking

Since we showed that the scheduler is equivalent to the strategy that it translates, and we know that by definition the strategy provides a move in every possible situation, the scheduler must in fact be nonblocking.

∎

Fourteen

Games for Processes with Probabilistic Choice

In this chapter, we discuss labelled processes equipped with a probabilistic choice operator and a single scheduler or player that resolves all nonprobabilistic choices. In some ways, this situation is similar to the two-agent situation; the single nondeterministic agent interacts with the outcomes of probabilistic choices in much the same way as it interacts with the outcome of choices made by the other player in the two-player situation. On the other hand, the probabilistic choice cannot be said to be resolved according to a strategy since it is, of course, resolved completely probabilistically, according to the distributions built into the process definition.

We begin by giving background on probabilistic processes. Next, we discuss games, strategies and epistemic restrictions for these processes. Finally, we prove that these introspective strategies for processes with probabilistic choice are equivalent to the schedulers for processes with probabilistic choice defined in [CP10b].

14.1 Syntax and Semantics

The syntax of these processes is almost the same as the syntax of processes with an independence operator. The only difference is that the brackets

signifying an independent choice are replaced with a labelled probabilistic choice operator.

$$P, Q ::= 0 \mid l : \alpha.P \mid P + Q \mid l : \sum_i l_i : p_i P_i \mid P|Q \mid (\nu a)P$$

For a process of the form $l : \sum_i l_i : p_i P_i$, we also require that $\sum_i p_i = 1$.

The operational semantics for labelled processes with probabilistic choice, shown in Fig. 14.1, is generally similar to the operational semantics without probability, but with two significant changes. First, each transition between two processes now has a probability assigned to it, in addition to an action and string of labels like in the other operational semantics. Second, the SWITCH rule is replaced with the PROB rule, representing probabilistic choice; the choice is resolved by the process doing a silent transition to one of the subprocesses, with the probability indicated in the original process. The other rules are straightforward analogues of the traditional process algebra rules. Note that only a τ transition can have a probability other than one. This is why in the COM rule we require that the transitions taken by P and Q have probability one; in fact, this is the only possibility for these transitions. In the strings of labels, a label can either have a subscript X, if it is not a label on a branch of a probabilistic choice, or no added subscript, if it is a label on a branch of a probabilistic choice.

$$\text{ACT}\ \frac{}{l : \alpha\,.\,P \xrightarrow[l_X]{\alpha}_1 P} \qquad \text{PROB}\ \frac{}{l : \sum_i l_i : p_i P_i \xrightarrow[l_X.l_i]{\tau}_{p_i} P_i}$$

$$\text{SUM1}\ \frac{P \xrightarrow[\lambda]{\alpha}_p P'}{P + Q \xrightarrow[\lambda]{\alpha}_p P'} \qquad \text{PAR1}\ \frac{P \xrightarrow[\lambda]{\alpha}_p P'}{P|Q \xrightarrow[\lambda]{\alpha}_p P'|Q}$$

$$\text{COM}\ \frac{P \xrightarrow[l_X]{a}_1 P' \quad Q \xrightarrow[j_X]{\bar{a}}_1 Q'}{P|Q \xrightarrow[(l,j)_X]{\tau}_1 P'|Q'} \qquad \text{RES}\ \frac{P \xrightarrow[\lambda]{\alpha}_p P' \quad \alpha \neq a, \bar{a}}{(\nu a)P \xrightarrow[\lambda]{\alpha}_p (\nu a)P'}$$

Figure 14.1: Operational semantics for processes with probabilistic choice

We will only consider deterministically labelled processes: processes where every transition has a unique string of labels.

Definition 14.1.1 (Deterministically Labelled)**.** *A probabilistic process P is* deterministically labelled *if the following conditions hold:*

1. *It is impossible for P to make two different transitions with the same labels: if $P \xrightarrow[s]{\alpha}_{p_1} P'$ and $P \xrightarrow[s]{\beta}_{p_2} P''$ then $\alpha = \beta$, $p_1 = p_2$, and $P' = P''$.*

2. *If $P \xrightarrow[l_X.l']{\tau}_{p} P'$ then there is no transition $P \xrightarrow[l_X]{\alpha} P''$ for any α or p or P''.*

3. *Whenever $P \xrightarrow[s]{\alpha}_{p} P'$ then P' is deterministically labelled.*

Finally, since we are considering probabilities, we must discuss how they are composed in transition sequences of process. To construct transition sequences, we assume that the probabilities at every step are independent from one another. Thus, the probability of a sequence of transitions is just the product of the probabilities of each transition in the sequence. This is formalized below.

14.2 Games, Valid Positions and Strategies

In this section, we define games and strategies on probabilistic labelled processes. The construction of games and strategies is similar to the two player construction, since the player interacts with the probabilistic choices in a way similar to the way the two players interact in the nonprobabilistic case.

14.2.1 Valid Positions

First we define the extension of the transition relation to allow sequences of transitions, by concatenating the label strings and multiplying the probabilities.

Definition 14.2.1. *For any process P, $P \xrightarrow[\varepsilon]{}_{1} P$, and if $P \xrightarrow[s]{\alpha}_{p_1} P'$ and $P' \xrightarrow[s']{}_{p_2} P''$, then $P \xrightarrow[s.s']{}_{p_1 \cdot p_2} P''$.*

Now we define valid positions.

Definition 14.2.2 (Valid Position)**.** *If* $P \xrightarrow[s]{}_{p} P'$ *then every prefix of* s*, including* s*, is a* valid position *for* P*.*

Now we define the game tree for P. Because of the combination of nondeterministic and probabilistic choice in the tree, we do not define a probability measure on the game tree. Instead, the game tree represents all possible executions, without taking the probability of each execution into account. The probability measure on valid positions is defined later with respect to a strategy that resolves the nondeterministic choices.

Definition 14.2.3. *Let* V *be the set of valid positions for probabilistic process* P*. The game tree for* P *is a tree where the root is epsilon and the other nodes are the other valid positions for* P*. For a node* s*, the children of* s *are all the positions of the form* $s.m$*.*

As in the nondeterministic case, we define the set of children of a valid position.

Definition 14.2.4. *Let* V *be the set of valid positions for a process. For* $s \in V$*, we define* $Ch(s) = \{s' \in V \mid s' = s.m \text{ for some move } m\}$*.*

We define the partial function $Pl : V \to \{X, prob\}$, the function that says whether at a valid position it is the player's turn or a probabilistic choice point.

Definition 14.2.5. *Let* V *be the set of valid positions for a process. For* $s \in V$*,* $Pl(s) = X$ *if and only if there is some* $s' \in Ch(s)$ *such that* $s' = s.l_X$*.* $Pl(s) = prob$ *if and only if there is some* $s' \in Ch(s)$ *and* $Pl(s) \neq X$*. If* $Pl(s) = X$*, we say that* s *belongs to the player or is a player position, and if* $Pl(s) = prob$ *we say that* s *is a probabilistic position. The leaves in the game tree are neither player positions nor probabilistic positions.*

14.2.2 Strategies

Besides there only being one player, the definition of a strategy and the restrictions on strategies are quite similar to the two player case. We recall all the definitions here only for convenience.

We start by defining player moves and probabilistic moves.

Definition 14.2.6. *If $s.m_X$ is a valid position for P, then m_X is a* player move *in this valid position. If $s.l$ is a valid position for P, then l is a* probabilistic move *in this valid position.*

Now we can define strategies.

Definition 14.2.7 (Strategy)**.** *In the game for a process P, a* strategy *S is a subtree T of the game tree for P meeting the following three conditions:*

1. $\varepsilon \in T$
2. *If $s \in T$ and $Pl(s) = X$ then exactly one of the children of s is in T.*
3. *If $s \in T$ and $Pl(s) = prob$ then $Ch(s) \subseteq T$.*

14.2.3 Execution of a probabilistic process with a strategy

Since a strategy resolves all the nonprobabilistic choices in a probabilistic process, a process paired with a strategy gives a normalized distribution on possible executions of the process.

We cannot define a probability measure on the set of all valid positions for several reasons. First, the probability assigned to a valid position must be based on the probability of that execution of the process occurring, but not all valid positions actually represent possible executions. For example, for the process

$$l : (l_1 : \tfrac{1}{2}(l' : a) + l_2 : \tfrac{1}{2}(l'' : b))$$

l_X is a valid position, but there is no reasonable way to assign a probability to this valid position because alone, it does not represent a partial execution

of the process. Furthermore, the fact that some valid positions represent partial executions and the combination of probabilistic and nonprobabilistic choice means that the sum of the probabilities of all the valid positions will usually be more than one. Thus, we will only define the probability measure on a special, restricted set of valid positions.

First, we define the notion of a *final* valid position: a valid position with no possible continuations.

Definition 14.2.8. *Let V be the set of all valid positions for a process. Define the set of* final *valid positions as $V_f = \{s \mid s \in V \text{ and } Ch(s) = \emptyset\}$. s is a* final valid position *if $s \in V_f$.*

Next we will consider the set of final valid positions in a strategy S.

Definition 14.2.9. *Let V be the set of valid positions for a process P and let S be a strategy for P. Define*

$$final(S) = \{s \in V_f \mid s \in S\}.$$

Since a strategy resolves all nonprobabilistic nondeterminism, and taking only the final valid positions removes all partial executions, this definition gives us a set on which a probability measure can be defined.

Definition 14.2.10. *If S is a strategy for process P, define $\mu_P : final(S) \longrightarrow [0,1]$ as follows: for $s \in final(S)$, if $P \underset{s}{\overset{p}{\longrightarrow}} P'$, then $\mu_P(s) = p$.*

We will prove that μ_P is indeed a probability measure, but first we need an auxiliary definition.

Definition 14.2.11. *For S a strategy, define*

$$S/s = \{s' \mid s.s' \in S\}.$$

Theorem 14.2.12. *If S is a strategy for P, then $\mu_P : final(S) \longrightarrow [0,1]$ is a probability measure.*

Proof. Since μ_P is defined on singletons and then extended in the evident way to arbitrary sets and the overall space is finite it is clear that μ_P is additive. Thus, all we have to show is that

$$\mu_P(final(S)) = \sum_{s \in final(S)} \mu_P(s) = 1$$

This will be proved by induction on the length of the maximal element in $final(S)$.

Base Case : P is blocked. Then ε is the only valid position for P, so $\varepsilon \in V_f$ and $S = \{\varepsilon\}$ by definition of strategy, so $final(S) = \{\varepsilon\}$. And for any process P, $P \underset{\varepsilon}{\longrightarrow}_1 P$, so $\mu_P(\varepsilon) = 1$.

Case : S starts by choosing a move m that does not label a probabilistic choice, resulting in P going to P'. Then it is easy to see that S/m is a strategy for P', so by the induction hypothesis, $\mu_{P'}(final(S/m)) = 1$. Note, that every element of $final(S)$ is of the form $m.s$ where $s \in final(S/m)$, since from the definition of strategy, S can only contain one child of m. Furthermore, since $P \underset{m}{\longrightarrow}_1 P'$, we see from the definition of μ_P that if $m.s \in final(S)$ then $\mu_P(m.s) = \mu_{P'}(s)$. Therefore, $\mu_P(final(S)) = \mu_{P'}(final(S/m)) = 1$.

Case : S starts by choosing a label l of a probabilistic move of the form $l : \sum_{i=1}^{n} l_i : p_i P_i$. For $i = 1$ to n, let

$$S_i = \begin{cases} S/(l.l_i) & \text{if } P_i \text{ is not blocked} \\ \{\varepsilon\} & \text{otherwise} \end{cases}$$

Then since S must by definition of strategy contain all children of l, it is easy to see that for each i, S_i must be a strategy for P_i. Now, for a string s and a set S', let $s \odot S' = \{s.s' | s' \in S'\}$. Then it can be shown that

$$final(S) = \bigcup_{i=1}^{n} l.l_i \odot final(S_i).$$

Furthermore,

$$\mu_P(l.l_i \odot final(S_i)) = \sum_{s' \in final(S_i)} \mu_P(l.l_i.s'),$$

but since $P \xrightarrow[l.l_i]{p_i} P_i$, by definition 14.2.1, we have that $\mu_P(l.l_i.s') = p_i \cdot \mu_{P_i}(s')$. So altogether,

$$\begin{array}{lcll} \sum_{s \in final(S)} \mu_P(s) & = & \sum_{i=1}^{n} \mu_P(l.l_i \odot final(S_i)) & \\ & = & \sum_{i=1}^{n} p_i \cdot \mu_{P_i}(final(S_i)) & \\ & = & \sum_{i=1}^{n} p_i & \text{by induction hypothesis} \\ & = & 1 & \text{by definition} \end{array}$$

∎

Finally, we would like to point out that epistemic restrictions on strategies are defined in the probabilistic case just exactly as they are in the nondeterministic, two-player case. For example, a player strategy that respects the introspective equivalence relation would correspond to a player or scheduler that does not see the outcomes of probabilistic choices, but has all the information about the moves he has made and the moves that have been available to it.

Fifteen

A Modal Logic for Strategies

In this section we present a modal logic intended to reason about games on processes, particularly knowledge, information flow, and the effects of actions on knowledge. This is not intended to be the final word on the subject; this is a version developed for this particular game-semantics application. One of the advantages of this logic is that it allows us to characterize certain useful equivalences on positions using classes of formulas. This characterization is intended to be in the spirit of the Hennessy-Milner-van Benthem theorem which gives a modal characterization of bisimulation. Of course, our characterization result is much less general than this theorem, because the equivalences we are characterizing are less general than bisimulation, and because our relations are characterized only by specific classes of formulas, rather than by all formulas in the logic, as in the Hennessy-Milner-van Benthem theorem.

We consider two-player processes with a switch operator rather than probabilistic processes because we wish to avoid probabilistic logic, the subtleties of which are largely orthogonal to our present considerations. We take the tree of valid positions for a process as our set of states. Our logic will allow us to discuss several aspects of any given valid position. These aspects are intended to be natural possibilities for a player's perceptions of what is occurring in the execution of the game.

- Which player made the last move and what the last move was,

- What moves are available and what player they belong to,
- What formulas are satisfied by specific continuations of the current valid position,
- What formulas are satisfied by specific prefixes of the current valid position,
- The knowledge of each player in the current state, according to an equivalence relation on the set of states, independent from the logic, and
- What formulas were satisfied by the state immediately after either player's last move.

15.1 Syntax and Semantics

As mentioned above, we take the tree of valid positions for a certain process as our model, and a specific valid position as our state. For V the tree of valid positions for a process, a valid position $s \in V$ and a formula ϕ, we say that $(V, s) \models \phi$ if ϕ is true at s in the game tree V. When it is unambiguous from the context what the model is, we omit the V and write $s \models \phi$.

Let L represent a general label (a single label or a synchronizing pair of labels), m a move (a general label together with a player), let X and Y be the two players, and let Z represent either X or Y.

$$\phi ::= C_Z(L) \mid A_Z(L) \mid \bigcirc_m \phi \mid \ominus\phi \mid K_Z\phi \mid @_Z\phi \mid \phi \wedge \phi \mid \neg\phi \mid \top.$$

We give the semantics for the operators first and explain them afterwards.

1. $(V, s.L_Z) \models C_Z(L)$.
2. $(V, s) \models \ominus\phi$ if for some position s', $s \in Ch_V(s')$t and $(V, s') \models \phi$.
3. $(V, s) \models @_Z\phi$ if $s = s'.L_Z$ and $(V, s) \models \phi$ or $s = s'.L_Z.L_{\bar{Z}}^1.L_{\bar{Z}}^2...L_{\bar{Z}}^n$ and $(V, s'.L_Z) \models \phi$.

4. $(V, s) \models K_Z\phi$ if for all $s' \sim_Z s$, $(V, s') \models \phi$.

5. $(V, s) \models \phi_1 \wedge \phi_2$ if $(V, s) \models \phi_1$ and $(V, s) \models \phi_2$.

6. $(V, s) \models \neg\phi$ if it is not the case that $(V, s) \models \phi$.

7. $(V, s) \models \top$ for all s and all V.

8. $(V, s) \models A_Z(L)$ if $s.L_Z \in Ch_V(s)$.

9. $(V, s) \models \bigcirc_m\phi$ if $(V, s.m) \in Ch_V(s)$ and $(V, s.m) \models \phi$.

Some of these operators require discussion. The first three deal with the history of the current position, and the last two deal with possible continuations of the current position. $C_Z(L)$ just says that the last move chosen was L, and it was chosen by player Z. Similarly, $\ominus\phi$ removes the last move from the current position and checks whether ϕ held at that point.

$@_Z\phi$ is more complicated. According to the formal definition, it holds when ϕ holds at the *most recent position where it was Z's turn* before the current position. This operator appears contrived at first glance, but in the setting of agents who may have limited knowledge, it has significance beyond just being used to characterize introspection. After an agent moves, it may not know what the other agent has done, and indeed whether the other agent has done anything at all, until it is again the original agent's turn. Thus, it may know what the conditions were in the game at the last time that it was its turn, without knowing what they are now, and this kind of information is exactly what the $@_Z$ operator captures. The fact that this operator is reasonable and natural in the setting of agents interacting with limited knowledge of the overall execution of the process, will be made clearer when we show that it turns out to be useful in discussing other reasonable limitations of agents' knowledge in different settings.

The knowledge operator is standard from epistemic logic. Its semantics requires the definition of the equivalence relation $\sim_Z$, which is given as part of the model. The idea behind this operator is that an agent considers several states possible when it is in a certain state. This is the agent's

uncertainty about what state the system is in. The agent only knows a fact if it is true in all the states that it considers possible from the current state.

$A_Z(L)$ means that from the current position, it is agent Z's turn and it has the option to choose move L. $\bigcirc_m\phi$ is similar to the familiar $\langle a\rangle\phi$ operator in Hennessy-Milner Logic, or the $X\phi$ operator in Linear Temporal Logic. It means that move m is available and if it occurs next, then ϕ will be true. Since we require our processes to be deterministically labelled, if ϕ may hold after m and m is available, then ϕ will certainly hold after m. The move can only lead to one state, because of deterministic labelling.

Finally, note that in the syntax and semantics we only discuss the traditional logical connectives $\wedge$ and $\neg$, so that the notation is concise. However, from now on we will use $\phi_1 \vee \phi_2$ as shorthand for $\neg(\neg\phi_1 \wedge \neg\phi_2)$, $\phi_1 \Rightarrow \phi_2$ for $\neg\phi_1 \vee \phi_2$, and $\phi_1 \Leftrightarrow \phi_2$ for $(\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1)$. On the other hand, we do not actually need the operator $A_Z(L)$ since it is equivalent to $\bigcirc_{L_Z}\top$ but we leave it in our syntax and semantics anyway, to make the explanations simpler.

15.2 Basic Properties Captured in Modal Logic

This section discusses formulas that capture some basic properties. Many of them hold in most modal logics while some others are specific to our case. These kinds of formulas often arise in the course of giving a complete axiomatization for a modal logic.

1. $\bigcirc_m\phi \Rightarrow \neg \bigcirc_m \neg\phi$.

   This formula is true because we require our processes to be deterministically labelled. Thus, there is at most one state that any valid position can transfer to for any given move m, and any formula that can possibly hold after m therefore must hold after m.

2. $\phi \Rightarrow \neg \bigcirc_m \neg\ominus\phi$.

This formula is true because our states have a tree structure: there is at most one immediate previous state for any valid position.

3. $C_Z(L) \Rightarrow \ominus A_Z(L)$.

   This formula says that if a move was chosen in the previous state, it must have been available there.

4. $A_Z(L) \Rightarrow \bigcirc_{L_Z} C_Z(L)$.

   This formula says that if a move is enabled, then there is a next state where that move was chosen. The last two formulas seem obvious, but formal expressions of the relationships between the operators are often useful, and are necessary to give a complete axiomatization for the logic.

Since we define knowledge using an equivalence class on states in the normal Kripke way, we automatically know that the knowledge axioms as discussed, for example, in [Kri63], are true:

1. $K_Z\phi \Rightarrow \phi$.

   This can be interpreted as saying that knowledge is true.

2. $K_Z\phi \Rightarrow K_Z K_Z \phi$.

   This means that the agents are aware that they know what they know.

3. $(K_Z(\phi \Rightarrow \psi) \wedge K_Z\phi) \Rightarrow K_Z\psi$.

   Agents can reason and form new knowledge from what they know.

4. $\neg K_Z\phi \Rightarrow K_Z \neg K_Z \phi$.

   If an agent does not know something, it is aware of this fact.

15.3 Logical Characterization of Indistinguishability Relations

In the section about epistemic restrictions on strategies, we discussed several possible indistinguishability (uncertainty) relations on valid positions. We

will show that this logic can be used to characterize all of the equivalences we discussed. That is, for each equivalence relation E we discussed, we will show that there is a class of formulas Φ_E such that for valid positions s and t, sEt if and only if for all $\phi \in \Phi_E$, $s \models \phi \Leftrightarrow t \models \phi$. This kind of result could because, for example, it means that given a logically definable equivalence relation or a definition of an agent's perception, it means that anytime an agent can distinguish two states, we can come up with a specific formula that the agent knows to be true at one state and false at the other state. Furthermore, in many situations it may be more convenient or intuitive to describe an agent's equivalence relation by giving a class of formulas that equivalent states agree on. This class of formulas can be thought of as the class of formulas that the agent is aware of: at any state, the agent knows whether any formula in this class is true or false. The following examples will make this discussion clearer.

Example 15.3.1. *Recall from Definition 12.4.4 that $s_1 H_Z s_2$ iff $Z(s_1) = Z(s_2)$, that is, each player only remembers his own moves. Let Φ be the class of all formulas of the form $(@_Z \ominus)^n @_Z C_Z(L)$, for $n \geq 0$. Then $s_1 H_Z s_2$ if and only if for any $\phi \in \Phi$, $s_1 \models \phi \Leftrightarrow s_2 \models \phi$. This is because $s \models @_Z C_Z(L)$ if and only if L is the last Z move in s, and $s \models @_Z \ominus @_Z C_Z(L')$ if and only if L' is the second to last Z move in s, and so on. So if two valid positions agree on all such formulas, they must have the same Z moves in the same order.*

The above example also serves as justification for the @ operator. Even though this operator may seem strange, it is natural from the point of view of a player, who may only be aware of what happens when it is his turn to move, but cannot distinguish between the other player not moving at all and it being the first agent's turn again immediately, or the other player making one move before it is the first player's turn again, or the other player making many moves before it is again the first player's turn.

Example 15.3.2. *Recall from Example 12.4.7 that $s_1 Av_Z s_2$ iff $Av_Z(s_1) = Av_Z(s_2)$. Clearly, the set of formulas that characterizes this equivalence relation is the set of all formulas of the form $A_Z(L)$.*

We will also give a few new examples of equivalences that were not discussed earlier as well.

Example 15.3.3. *Consider the equivalence relation n where $(s_1, s_2) \in n$ iff the last n moves in s_1 are the same as the last n moves in s_2. This relation is the same for either player. It describes agents who see all the moves that occur but only have finite memory. The class of formulas characterizing this equivalence relation is the class $\{\ominus^k(C_Z(L)) \mid k < n,\ Z \in \{X, Y\},$ and L is any move$\}$.*

Example 15.3.4. *Similarly, we could say that two positions are indistinguishable for player Z if Z made the same last n moves in both positions. We call this equivalence n_Z, and the class of formulas characterizing it is $\{(@_Z\ominus)^k @_Z C_Z(L) \mid k < n\}$.*

Finally, we can characterize the introspective indistinguishability relation we discussed above. Recall from Definition 12.4.8 that $s_1 I_Z s_2$ if all of the following conditions hold:

1. $Z(s_1) = Z(s_2)$
2. $enabled_Z(s_1) = enabled_Z(s_2)$
3. For all $s_1' \leq s_1$, $s_2 \leq s_2'$, if $Z(s_1') = Z(s_2')$ then $enabled_Z(s_1') = enabled_Z(s_2')$ or $enabled_Z(s_1') = \emptyset$ or $enabled_Z(s_2') = \emptyset$.

Proposition 15.3.5. *sI_Zt if and only if s and t agree on all formulas of the form*

$$(@_Z\ominus)^n @_Z C_Z(L)$$

for $n \geq 0$, and for any L, and also agree on all formulas of the form

$$(@_Z\ominus)^n A_Z(L)$$

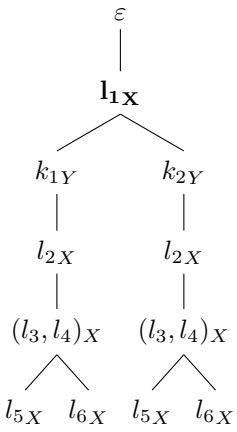
for $n \geq 0$ and for any L.

Proof. First, as discussed above, s and t agreeing on all formulas of the form $(@_Z\ominus)^n @_Z C_Z(L)$ is equivalent to $Z(s) = Z(t)$. Similarly, s and t agreeing on all formulas of the form $A_Z(L)$ (i.e. $(@_Z\ominus)^0 A_Z(L)$) means

that $enabled_Z(s) = enabled_Z(t)$. Finally s and t agreeing on all formulas of the form $(@_Z\ominus)^n A_Z(L)$ is equivalent to the third condition in the definition of the introspective relation. This is because we have already ensured that $Z(s) = Z(t)$ so $(@_Z\ominus)^n$ means counting backwards n Z moves and n contiguous series of $\bar{Z}$ moves, and then checking that $enabled_Z$ is the same in the two strings. This shows that two valid positions agree on all formulas of the specified forms if and only if they are Z-indistinguishable. ∎

Example 15.3.6. *To make this idea clearer, we show how the logic works with one of the processes discussed earlier. For*

$$P = (\nu b)\left(l_1 : \{k_1 : \tau \,.\, l_2 : a \,.\, l_3 : b + k_2 : \tau \,.\, l_2 : c \,.\, l_3 : b\} \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right)$$



$(l_{1X}.k_{1Y}.l_{2X},\ l_{1X}.k_{1Y}.l_{2X}) \in I_X$ *and therefore, these two positions agree on all formulas of the form* $(@_X\ominus)^n @_X C_X(L)$ *and* $(@_X\ominus)^n A_X(L)$*. For example we will unfold one such formula with the semantics,*

$l_{1X}.k_{1Y}.l_{2X}$	$\models$	$@_X\ominus A_X(l_2)$	*because*
$l_{1X}.k_{1Y}.l_{2X}$	$\models$	$\ominus A_X(l_2)$	*because*
$l_{1X}.k_{1Y}$	$\models$	$A_X(l_2)$	*because* $l_{1X}.k_{1Y}.l_{2X} \in Ch(l_{1X}.k_{1Y})$

Similarly, $l_{1X}.k_{2Y}.l_{2X} \models @_X\ominus A_X(l_2)$*. Furthermore, these two positions agree on all other formulas in the characterizing class.*

As another example, in the same process,

$$(l_{1X}.k_{1Y}.l_{2X}.(l_3, l_4)_X,\ l_{1X}.k_{2Y}.l_{2X}.(l_3, l_4)_X) \in I_X$$

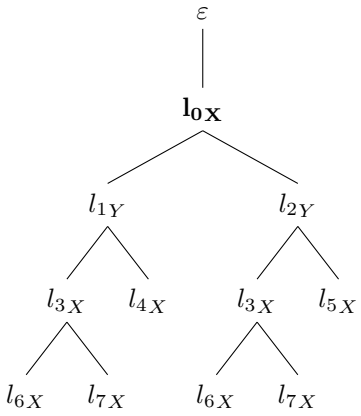
Both of these positions model the following formulas in the characterizing class:

$A_X(l_5)$	$A_X(l_6)$
$@_X C_X((l_3, l_4))$	$@_X \ominus A_X((l_3, l_4))$
$@_X \ominus @_X C_X(l_2)$	$@_X \ominus @_X \ominus A_X(l_2)$
$@_X \ominus @_X \ominus @_X C_X(l_1)$	$@_X \ominus @_X \ominus @_X \ominus A_X(l_1)$

and neither of them models any other formula in the characterizing class.

Example 15.3.7. *Consider the process*

$$P = {}^0\{\ {}^1\tau\ .(\ {}^3c\ .(\ {}^6f + {}^7g\ ) + {}^4d\ ) + {}^2\tau\ .(\ {}^3c\ .(\ {}^6f + {}^7g\ ) + {}^5e\ )\}.$$



The positions $l_{0X}.l_{1Y}.l_{3X}$ and $l_{0X}.l_{2Y}.l_{3X}$ are not introspectively equivalent for X. $l_{0X}.l_{1Y}.l_{3X} \models @_X \ominus A_X(l_4)$ but $l_{0X}.l_{2Y}.l_{3X} \not\models @_X \ominus A_X(l_4)$. Furthermore, $l_{0X}.l_{1Y}.l_{3X} \not\models @_X \ominus A_X(l_5)$ whereas $l_{0X}.l_{2Y}.l_{3X} \not\models @_X \ominus A_X(l_4)$.

15.4 Properties Following from Logical Characterizations of Equivalence Relations

When we are in a setting where we have a logical characterization of the desired indistinguishability relation for agents, we can conclude that certain logical formulas about their knowledge hold universally in the system. This result has interesting implications for our logic. Let $\sim_Z$ be the indistinguishability relation for Z.

Theorem 15.4.1. *If Φ characterizes $\sim_Z$, that is, if $s_1 \sim_Z s_2$ if and only if s_1 and s_2 agree on all formulas in ϕ, then for any $\phi \in \Phi$, $\phi \rightarrow K_Z\phi$, and $\neg\phi \rightarrow K_Z\neg\phi$. Furthermore, for any formula $\phi \in \Phi$, every state satisfies $K_Z\phi \vee K_Z\neg\phi$.*

Proof. Assume V is the set of valid positions and Φ characterizes $\sim_Z$. For any position s, if $s \models \phi$, then for all $t \sim_Z s$, $t \models \phi$. So, by the semantics of K_Z, this means that $s \models K_Z\phi$. Similarly, if $s \models \neg\phi$, then for all $t \sim_Z s$, $t \models \neg\phi$, so $s \models K_Z(\neg\phi)$. Thus, at all states, for any formula $\phi \in \Phi$, $\phi \rightarrow K_Z\phi$ and $\neg\phi \rightarrow K_Z(\neg\phi)$. Finally, since $\phi \vee \neg\phi$ holds at any state, we can conclude that $K_Z\phi \vee K_Z\neg\phi$ holds at any state. ∎

Conclusions and Related Work

Now we present the conclusions and related work for this part of the thesis, and in the next chapter we present the conclusions for the whole thesis.

In this part of the thesis we have given a semantic treatment of a process algebra with two kinds of choice in terms of games and strategies. This gives a semantic understanding of the "knowledge" possessed by schedulers when they resolve choices. This epistemic aspect is captured by restrictions on what the schedulers can see when they execute their strategies. We have also introduced a modal logic with dynamic and epistemic modalities to capture more precisely what agents know.

This work is a first step toward a systematic game semantic exploration of concurrency. We plan to continue this line of research in several directions. First of all, we would like to develop a process algebra which is more naturally adapted to games and perhaps also to multi-agent games. This will lead to richer notions of interactions between agents than synchronization and value or name passing.

In an interesting paper published in 2003 [MW03], Mohalik and Walukiewicz explored distributed games from the viewpoint of distributed controller synthesis. In that work the goal is to *synthesize* a finite-state controller that will model a finite set of independent concurrent agents interacting with an adversarial environment. The question addressed there, the synthesis problem, has a long history in both concurrency theory and control theory. In the work just cited, there is also a restriction of agents' strategies to what they can see *locally*. Though not expressed as epistemic restrictions that is clearly what is intended and the paper even cites the distributed systems model of Halpern and Moses [HM84] as an explicit acknowledgment of the

epistemic aspects of their work. It is a very suggestive connection and we look forward to exploring this in future work.

Second, we would like to enrich further the epistemic aspects of the subject. In particular, we would like to move toward an explicit combination of modal process logic and epistemic logic so that we can describe in a compositional process-algebraic way how agents learn and exchange knowledge. The idea is to move towards a more general logic that would capture how agents learn as transitions occur in a labelled transition system equipped with additional equivalence relations. This goal is closely related to the second part of this thesis, and we would like to make stronger connections between the logic developed there and a specific process calculus which is more naturally adapted to include local information specific to certain agents, as well as game-like interaction.

Third, we would like to explore more subtle notions of transfer of control between the agents. Thus, for example, there could be a protracted dialogue between the agents before they decide on a process move. This could conceivably be fruitful for incorporating higher-order or mobile processes. Of course, the theory of higher-order processes is much more complicated and game semantics for it will involve the complexities that are needed for models of the λ-calculus [HO00]. However, it might be illuminating to understand restrictions on strategies, such as innocence, explicitly in epistemic terms. Of course, many of the restrictions will not be epistemic, for example, well-bracketing.

Finally, we would like to combine the epistemic and probabilistic notions using ideas from information theory [Sha48]. These information theoretic ideas have been used for an analysis of anonymity [CPP08], indeed it was that investigation that sparked the research reported in [CP10b] and which ultimately led to the present work. As far as we know, the only paper looking at epistemic logic and information theory is by [KNP90] where the amount of information shared when agents possess common knowledge is quantified. Of course, these ideas are speculative at this point.

Sixteen

Conclusion

In this thesis, we presented three new ways of analyzing epistemic information in concurrent settings. In Part I of the thesis, we focused on developing modalities as programming constructs in Concurrent Constraint Programming. We added epistemic and spatial modalities as new combinators, making it possible to view multi-agent modal logic as a programming language. In Part II, we developed a dynamic epistemic logic of multi-agent labelled transition systems with fact changing actions. This allowed us to analyze the effects of actions on agents' knowledge. In Part III, we introduced a process calculus with independent agents taking actions, and a game semantics to replace the traditional scheduler. The advantage of a game semantics for a process calculus is that it provides an elegant way to model the effect of agents' limited knowledge on the actions they can take.

A great deal of future work remains to be done on these issues. Currently, our ideas for future work are mainly focused on SCCP. We are particularly interested in extensions of SCCP that would make it more applicable to solving real-world problems in distributed systems. SCCP is particularly adapted to resolving issues of unreliable agents or agent failure, hierarchies of agents communicating with one another, large systems of agents acting in parallel, and problems of information flow, information change, and security threats. Extending this work to include a temporal modality and more general mobility would make it even more effective for modelling these

systems.

Quantitative reasoning Currently, SCCP does not include any notion of quantitative information or weighted belief. An extension with this feature is obviously crucial for modelling real-world systems with epistemic information. We plan to generalize the SCCP notion of knowledge to a form of probabilistic measure on epistemic statements to express notions of confidence, uncertainty, or weighted belief, enabling the formalism to express more meaningful and faithful epistemic information, as well as more subtle changes in an agent's epistemic state.

Scalability SCCP assumes a fixed number of agents. This is a typical restriction in epistemic formalisms [FHMV95] but it limits our model considerably, since we aim to model systems with an unbounded, continually growing number of users. We plan to introduce an operator for dynamically creating new agents.

Spatial mobility The SCCP model only allows a very restricted notion of mobility: the ability of agents and programs to change from one space to another. This is a common feature in the distributed systems under consideration. In mobile systems, users and programs can change their communication structure, move around, and exchange information. Using ideas from dynamic epistemic logic [KMP12], temporal logic [Pnu77], and process calculi such as Ambients [CG00], we expect to be able to express spatial changes over the temporal evolution of distributed systems.

Temporal extension Timed CCP (TCCP) [SJG94] is a well-established extension of CCP which follows the paradigms of synchronous languages such as ESTEREL [BMT92]. Thus, TCCP is useful for modelling real-time reactive systems that maintain a permanent interaction with their environment. TCCP is used to model real-time controllers, communication, and other reactive phenomena. On the other hand, temporal logic is a well-established and practical subfield of modal logic, e.g. [Pnu77], showing that time can effectively be

treated as a modality. This suggests that we should be able to add a temporal modality to SCCP in a similar way to adding the spatial and epistemic modalities. By adding a temporal modality to SCCP in the spirit of TCCP, we can make it more suitable for formal modelling of wireless sensor networks, since these networks have temporal aspects such as constant communication and waiting for one event to trigger or suppress another event.

Coalgebraic approach The coalgebraic setting for modal logic provides a general framework allowing one to reason about a variety of different modal logics in a uniform way [Mos99]. Using coalgebraic methods may allow us to extend our existing results in several ways. This research goal should tie together several of our other research goals, since we hope that it will make it possible to use unified reasoning techniques to deal with quantitative reasoning and spatial mobility as natural extensions of the modalities we already have in our calculus.

Bibliography

[AG99] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.

[AJ94a] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *J. Symbolic Logic*, 59(2):543–574, 1994.

[AJ94b] S. Abramsky and A. Jung. Domain theory. In T. S. E. Maibaum S. Abramsky, D. M. Gabbay, editor, *Handbook of Logic in Computer Science, vol. III*. Oxford University Press, 1994.

[AJM00] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 2000.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Number 53 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001.

[BJPV09] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *LICS*, 2009.

[BM07] Maria Grazia Buscemi and Ugo Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In *ESOP*, pages 18–32, 2007.

[BMT92] Dave Berry, Robin Milner, and David N. Turner. A semantics for ml concurrency primitives. In *Proceedings Of The 19th Annual ACM Symposium On Principles Of Programming Languages*, pages 119–129, 1992.

[BP05] Mohit Bhargava and Catuscia Palamidessi. Probabilistic anonymity. In *Proc. of CONCUR*, volume 3653 of *LNCS*, pages 171–185. Springer, 2005.

[CC03] Luís Caires and Luca Cardelli. A spatial logic for concurrency (part i). *Information and Computation*, 186(2):194–235, 2003.

[CC04] Luís Caires and Luca Cardelli. A spatial logic for concurrency - ii. *Theor. Comput. Sci.*, 322(3):517–565, 2004.

[CDK09] Rohit Chadha, Stéphanie Delaune, and Steve Kremer. Epistemic logic for the applied pi calculus. In David Lee, Antónia Lopes, and Arnd Poetzsch-Heffter, editors, *Proceedings of IFIP International Conference on Formal Techniques for Distributed Systems (FMOODS/FORTE'09)*, Lecture Notes in Computer Science, pages 182–197. Springer, June 2009.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.

[CKPP12] Konstantinos Chatzikokolakis, Sophia Knight, Catuscia Palamidessi, and Prakash Panangaden. Epistemic strategies and games on concurrent processes. *ACM Trans. Comput. Log.*, 13(4):28, 2012.

[CP07] Konstantinos Chatzikokolakis and Catuscia Palamidessi. Making random choices invisible to the scheduler. In *International Conference on Concurrency Theory, CONCUR*, Lecture Notes In Computer Science 4703, pages 42–58, 2007.

[CP10a] Konstantinos Chatzikokolakis and Catuscia Palamidessi. Making random choices invisible to the scheduler. *Information and Computation*, 208(6):694–715, 2010.

[CP10b] Konstantinos Chatzikokolakis and Catuscia Palamidessi. Making random choices invisible to the scheduler. *Information and Computation*, 208(6):694–715, 2010.

[CPP08] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. *Inf. and Comp.*, 206(2–4):378–401, 2008.

[Den74] JackB. Dennis. First version of a data flow procedure language. In B. Robinet, editor, *Programming Symposium*, volume 19 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, 1974.

[DH01] Vincent Danos and Russell Harmer. The anatomy of innocence. *Lecture Notes in Computer Science*, 2142:188–202, 2001.

[DMO07] F. Deschesne, M.R. Mousavi, and S. Orzan. Operational and epistemic approaches to protocol analysis: Bridging the gap. In *14th International Conference on Logic for Programming Artificial Intelligence and Reasoning: LPAR'07*, Springer-Verlag Lecture Notes in Computer Science, pages 226–241, 2007.

[FGMP97] Moreno Falaschi, Maurizio Gabbrielli, Kim Marriott, and Catuscia Palamidessi. Confluence in concurrent constraint programming. *Theor. Comput. Sci.*, 183(2):281–315, 1997.

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[FL79] Michael Fischer and Richard Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 1979.

[FRS01] François Fages, Paul Ruet, and Sylvain Soliman. Linear concurrent constraint programming: Operational and phase semantics. *Inf. Comput.*, 165(1):14–41, 2001.

[GKK+03] G.Gierz, K.H.Hoffman, K.Keimel, J.D.Lawson, M.Mislove, and D.S.Scott. *Continuous lattices and domains*. Number 93 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2003.

[Gol03] Robert Goldblatt. Mathematical modal logic: A view of its evolution. *J. Applied Logic*, 1(5-6):309–392, 2003.

[Hin62] J. Hintikka. *Knowledge and Belief*. Cornell University Press, 1962.

[HM80] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In J. W. de Bakker and Jan van Leeuwen, editors, *ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer, 1980.

[HM84] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. In *Proceedings of the Third A.C.M. Symposium on Principles of Distributed Computing*, pages 50–61, 1984. A revised version appears as IBM Research Report RJ 4421, Aug., 1987.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–162, 1985.

[HM90] J. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *JACM*, 37(3):549–587, 1990.

[HMT88] Joseph Y. Halpern, Yoram Moses, and Mark R. Tuttle. A knowledge-based analysis of zero knowledge. In *Proceedings of the 20th ACM Symposium on Theory of Computing*, pages 132–147, 1988.

[HO00] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for pcf: I. models, observables and the full abstraction problem, ii. dialogue games and innocent strategies, iii. a fully abstract and universal game model. *Information and Computation*, 163:285–408, 2000.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HS04] Dominic Hughes and Vitaly Shmatikov. Information hiding, anonymity and privacy: a modular approach. *Journal of Computer Security*, 12(1):3–36, 2004.

[HT89] Joseph Y. Halpern and Mark R. Tuttle. Knowledge, probability and adversaries. In *Proceedings Of The Eighth Annual ACM Symposium On Principles of Distributed Computing*, pages 103–118. ACM, 1989.

[Joh02] Peter Johnstone. *Sketches of an Elephant: A Topos Theory Compendium I,II.* Number 43,44 in Oxford Logic Guides. Oxford University Press, 2002.

[KMP12] Sophia Knight, Radu Mardare, and Prakash Panangaden. Combining epistemic logic and hennessy-milner logic. In Robert L. Constable and Alexandra Silva, editors, *Logic and Program Semantics*, volume 7230 of *Lecture Notes in Computer Science*, pages 219–243. Springer, 2012.

[KNP90] Paul Krasucki, Gilbert Ndjatou, and Rohit Parikh. Probabilistic knowledge and probabilistic common knowledge. In *ISMIS 90*, pages 1–8. North Holland, 1990.

[KPPV12] Sophia Knight, Catuscia Palamidessi, Prakash Panangaden, and Frank D. Valencia. Spatial and epistemic modalities in constraint-based process calculi. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 317–332. Springer, 2012.

[Kri63] S. Kripke. Semantical analysis of modal logic. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[Lyn96] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[Mos99] Larry S. Moss. Coalgebraic logic. *Annals of Pure and Applied Logic*, 96:277–317, 1999.

[MPSS95] N. P. Mendler, P. Panangaden, P. J. Scott, and R. A. G. Seely. A logical view of concurrent constraint programming. *Nordic Journal of Computing*, 2:182–221, 1995.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes i and ii. *Information and Computation*, 100:1–77, 1992.

[MT44] J. C. C. McKinsey and Alfred Tarski. The algebra of topology. *The Annals of Mathematics, second series*, 1944.

[MW03] Swarup Mohalik and Igor Walukiewicz. Distributed games. In Paritosh Pandya and Jaikumar Radhakrishnan, editors, *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of *Lecture Notes in Computer Science*, pages 338–351. Springer Berlin, Heidelberg, 2003.

[NT87] G. Neiger and S. Toueg. Substituting for real time and common knowledge in asynchronous distributed systems. In *Proceedings of the Sixth A.C.M. Symposium on Principles of Distributed Computing*, pages 281–293, 1987.

[NT90] Gil Neiger and Mark R. Tuttle. Common knowledge and consistent simultaneous coordination. In J. van Leeuwen and N. Santoro, editors, *Proceedings of the Fourth International Workshop on Distributed Algorithms*, volume 486 of *Lecture Notes In Computer Science*, pages 334–352. Springer-Verlag, 1990.

[Pet63] Carl Adam Petri. Fundamentals of a theory of asynchronous information flow. In *Proc. of IFIP Congress 62*, pages 386–390, Amsterdam, 1963. North Holland Publ. Comp.

[Pnu77] A. Pnueli. The temporal logic of programs. In *The Eighteenth Annual IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[Pop94] Sally Popkorn. *First Steps in Modal Logic*. Cambridge University Press, 1994.

[Pra76] V. Pratt. Semantical considerations on floyd-hoare logic. In *Proceedings of the Seventeenth Annual IEEE Symposium on Computer Science*, pages 109–121. IEEE Press, 1976.

[PS11] Eric Pacuit and Sunil Simon. Reasoning with protocols under imperfect information. *The Review of Symbolic Logic*, 4(3):412–444, September 2011.

[PSSS93] P. Panangaden, V. Saraswat, P.J. Scott, and R.A.G. Seely. A hyperdoctrinal view of concurrent constraint programming. In J.W. de Bakker et al, editor, *Semantics: Foundations and Applications; Proceedings of REX Workshop*, number 666 in Lecture Notes In Computer Science, pages 457–476, 1993.

[PT88] P. Panangaden and K. E. Taylor. Concurrent common knowledge. In *Proceedings of the Seventh Annual ACM Symposium on Principles f Distributed Computing*, pages 197–209, 1988.

[PT92] P. Panangaden and K. E. Taylor. Concurrent common knowledge. *Distributed Computing*, 6:73–93, 1992.

[Rét98] Jean-Hugues Réty. Distributed concurrent constraint programming. *Fundam. Inform.*, 1998.

[Sar89] Vijay A. Saraswat. *Concurrent Constraint Programming Languages*. PhD thesis, Carnegie-Mellon University, January 1989.

[Sar93] Vijay A. Saraswat. *Concurrent constraint programming*. ACM Doctoral dissertation awards. MIT Press, 1993.

[Sco82] D. S. Scott. Domains for denotational semantics. In *Ninth International Colloquium On Automata Languages And Programming*. Springer-Verlag, 1982. Lecture Notes In Computer Science 140.

[Sha48] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656, July and October 1948.

[SJG94] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of timed concurrent constraint programming. In *Proceedings of the Ninth Annual IEEE Symposium On Logic In Computer Science, Paris, 1994*, pages 71–80. IEEE Press, 1994.

[SJPR91] V. A. Saraswat, R. Jagadeesan, P. Panangaden, and M. Rinard. Semantic foundations of concurrent constraint programming. In L. Augstsson et. al., editor, *Proceedings of the Baastad 91 workshop on Concurrency*, number 63 in Programming Methodology Group Chalmers University of Technology and Goteborg University, pages 385–427, 1991.

[SRP91] V. A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, 1991.

[SS96] Steve Schneider and Abraham Sidiropoulos. CSP and anonymity. In *Proc. of ESORICS*, volume 1146 of *LNCS*, pages 198–218. Springer, 1996.

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(2):285–309, 1955.

[Tur37] Alan M. Turing. Computability and lambda-definability. *J. Symb. Log.*, 2(4):153–163, 1937.

[vDvdHK08] Hans van Ditmarsch, Wiebe van der Hoek, and Barteld Kooi. *Dynamic Epistemic Logic*. Number 337 in Synthese Library. Springer-Verlag, 2008.

[vW51] G.H. von Wright. *An essay in modal logic*. Studies in logic and the foundations of mathemathics. North-Holland Publishing Company, 1951.

Index