Epistemic Strategies and Games on Concurrent Processes

KONSTANTINOS CHATZIKOKOLAKIS[1] and SOPHIA KNIGHT[1]
and
CATUSCIA PALAMIDESSI[1] and PRAKASH PANANGADEN[2]
1: INRIA and LIX, Ecole Polytechnique, France.
2: School of Computer Science, McGill University, Montréal, Québec

We develop a game semantics for process algebra with two interacting agents. The purpose of our semantics is to make manifest the role of knowledge and information flow in the interactions between agents and to control the information available to interacting agents. We define games and strategies on process algebras, 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 strategies represent different amounts of information being available to a scheduler. We also show that a certain class of strategies corresponds to the syntactic schedulers of Chatzikokolakis and Palamidessi, which were developed to overcome problems with traditional schedulers modelling interaction. The restrictions on these strategies have an explicit epistemic flavour.

Categories and Subject Descriptors: F.1.2 [**Theory of Computation**]: Modes of Computation—*parallelism*; *concurrency*; F.3.2 [**Theory of Computation**]: Logics and Meanings of Programs—*operational semantics*; I.2.4 [**Computing Methodologies**]: Knowledge representation formalisms and methods—*Modal logic*

General Terms: Security, Verification

Additional Key Words and Phrases: Game semantics, schedulers, epistemic logic, concurrency, process algebra, probability

1. INTRODUCTION

Concurrency theory is fundamentally about the interaction of autonomous agents, usually assumed to be computing agents. The most widely studied framework for the study of concurrent agents is process algebra. In this approach processes are given an explicit syntax and their dynamics are given by operational semantics. The structure obtained is a state transition system called a labelled transition system. The main mathematical tools that are brought to bear are binary relations that characterize behavioral equivalence of states and which lead to an equational theory (hence the name "process algebra") and modal logics that characterize the

This research was funded in part by NSERC and by an INRIA-Canada collaboration grant.

equivalences. The most commonly used equivalence relation is *bisimulation* and the logic that characterizes it is Hennessy-Milner logic [Hennessy and Milner 1980; 1985] independently discovered by van Benthem [van Benthem 1976; 1983]. 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 [Halpern and Moses 1984; Fagin et al. 1995]. Strangely, it has taken a long time for these ideas to surface in the concurrency theory 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 resolves all the choices, based on its own *knowledge*. 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 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 hold 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 adversarial or demonic scheduler all the time. For example, Schneider and Sidiropoulos [1996] 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 Beauxis and Palamidessi[2009]. These issues manifest themselves particularly sharply in the issue of anonymity.

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 is *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 Chatzikokolakis and Palamidessi [2010]: 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. This problem is discussed at length in [Chatzikokolakis et al. 2009]. The authors also propose a solution by introducing the concept of *demonic bisimulation*, which essentially requires the transitions for s and t to be chosen *by the same scheduler*.

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 paper 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 representing 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 [Abramsky and Jagadeesan 1994; Hyland and Ong 2000; Abramsky et al. 2000] 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 [2010]. It should be noted that this correspondence is significant since it only works with one precise restriction on the strategies, 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 se-

mantics 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 Chatzikokolakis and Palamidessi [2010] "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 et al. [2000] and Hyland and Ong [2000] 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 [Hyland and Ong 2000] 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 [Hyland and Ong 2000; Danos and Harmer 2001]. 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.

The complementary nature of the process algebraic and epistemic approaches to security, and the benefits to combine them, have been already recognized by several authors (e.g. [Hughes and Shmatikov 2004; Dechesne et al. 2007; Kramer et al. 2009; Chadha et al. 2009]). The approach of Hughes and Shmatikov [2004] uses function views to represent partial information and make the interface between protocol and properties. The approach of Dechesne et al. [2007] bridges the gap more directly, by proposing a combined framework in which the intruders epistemic knowledge is modeled by the set of traces generated by a process. Kramer et al. [2009] have extended the above work to probabilistic processes, and have also taken into account the presence of the scheduler, although (in contrast to the present paper) not as an active component in possible collusion with the adversary, but rather as part of the lack of knowledge of the agents. Chadha et al. [2009] have also proposed an epistemic logic to describe what an intruder can learn from the traces of a process. They have considered the applied π-calculus [Abadi and Fournet 2001], which is an extension of the pi calculus designed to manipulate complex data and functions, such as cryptographic primitives (e.g. encryption, signature, ...), instead of just names. Consequently, the approach of [Chadha et al. 2009] is much richer than the above approaches (and than the present work) from the point of view of reasoning about the specific properties of cryptographic protocols. The specificity of the present paper is that the scheduler is an integral part of the system, and that we use a game-theoretic approach to represent the interplay between the scheduler and the process, which contributes to construct (dynamically) the knowledge of the adversary.

In recent interesting work by Pacuit and Simon [Pacuit and Simon 2010] they discuss reasoning about protocols under uncertainty using dynamic epistemic logic. These ideas are close in spirit but the formal development is entirely different. The main difference is that programs are part of the syntax of formulas in dynamic logics whereas in process algebras the modal formulas and the transition systems are kept apart. Nevertheless it is also very much in the same general area as this paper though of course the particular issues of schedulers and their ability to leak knowledge are not part of their discussion.

2. BACKGROUND

We begin by introducing a process calculus with two distinctive features: labels associated with action prefixes and a protection operator represented by curly brackets. The execution of a process is controlled by two agents, X and Y. The first is the traditional scheduler controlling the execution of the global process. A sub-process enclosed in the protected operator, however, is controlled by the second agent Y, modelling the fact that this sub-process needs to hide information from the scheduler. The labels on actions are the only information that the agents have access to about the process. This allows an agent to hide information from the other agent; for example, assume that Y has two possible executions to choose from, leading to two different processes which, however, have the same labels. Since X can only see the labels, it will be unaware of the exact choice of Y. The interaction between the agents is made explicit in the next sections, where the agents are viewed as players

$$\text{ACT}\ \frac{}{l:\alpha\,.\,P \xrightarrow[l_X]{\alpha} P} \qquad \text{RES}\ \frac{P \xrightarrow[s]{\alpha} P' \quad \alpha \neq a,\bar{a}}{(\nu a)P \xrightarrow[s]{\alpha} (\nu a)P'} \qquad \text{SUM1}\ \frac{P \xrightarrow[s]{\alpha} P'}{P+Q \xrightarrow[s]{\alpha} P'}$$

$$\text{PAR1}\ \frac{P \xrightarrow[s]{\alpha} P'}{P|Q \xrightarrow[s]{\alpha} P'|Q} \qquad \text{COM}\ \frac{P \xrightarrow[l_X]{a} P' \quad Q \xrightarrow[j_X]{\bar{a}} Q'}{P|Q \xrightarrow[(l,j)_X]{\tau} P'|Q'} \qquad \text{SWITCH}\ \frac{P \xrightarrow[j_X]{\tau} P'}{l:\{P\} \xrightarrow[l_X.j_Y]{\tau} P'}$$

Fig. 1. Operational semantics

in a game.

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. 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. Simiarly, for the SWITCH rule, two labels are also included, one representing the fact that the protected process was chosen (by X) and one representing the action taken within the protected process (by Y). All the labels have an X or Y subscripted to them, denoting the agent that controls the corresponding action. 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, the novelties being the labels and the introduction of the SWITCH rule. This rule defines executions of protected processes, controlled by an independent agent Y. The main agent X (the traditional scheduler) selects the label of the protected sub-process and the second agent Y controls the execution of that sub-process. For example consider the process

$$(l_1 : a \ + \ l_2 : b) \mid l_3 : \{k_1 : \tau \,.\, l_4 : a \ + \ k_2 : \tau \,.\, l_4 : b\}$$

The main agent X controls whether the left part of the process performs an a or b action, but does not control how the choice on the right side of the process is resolved. This choice is instead controlled by an agent Y that acts independently.

Note that the SWITCH rule requires that the protected subprocess does a silent action τ. This ensures that the choice of Y is performed independently from any choice of X. Without this restriction, X could force Y to choose a specific action a by enforcing a synchronization to some external process performing an $\bar{a}$. A silent action, on the other hand, is independent from any external behaviour. Note also that the protection operator only protects the top level choices, since the SWITCH rule removes the protection. If the inner choices should be also controlled by Y then a nested protection operator can be used. This provides fine-grained control over which choices are independent and allows arbitrary alternations between the two agents.

Finally note that a deterministic choice is resolved by selecting a label at the left-hand or right-hand side of the choice operator; the operator itself has no labels. This might be counterintuitive at first, but leads to simpler semantics. It allows any transition to be determined by at most 2 labels. Labelling each side of the plus operator would require more, in the case of nested choices.

Because of our restriction to deterministically labelled processes, the evolution of a process is completely determined by the actions of the two agents. Besides the two agents' choices, there is no other source of nondeterminism.

Deterministically labelled processes. The idea behind this calculus is that the choices of the two agents, represented as the selection of a label, should completely determine the evolution of the process. In other words, there should be no other source of nondeterminism, besides the two agents' choices. However, this is not necessarily the case. Consider for example the process $l:a+l:b$, containing a choice of two actions, both labelled by l. The agent X (the global scheduler) is supposed to control the execution of this process, however selecting the label l (the only one available) does not uniquely determine a transition, there is still nondeterminism left. To avoid this problem, from now on we restrict ourselves to *deterministically labelled* processes, i.e. those where there can never be more than one action available with the same label.

We write $P \longrightarrow P'$ if $P \xrightarrow[s]{\alpha} P'$ for some α, s and $\longrightarrow_*$ for the reflexive and transitive closure of $\longrightarrow$.

Definition 2.1. *P is* deterministically labelled *iff for all Q s.t. $P \longrightarrow_* Q$ the following conditions hold:*

(1) *If $Q \xrightarrow[s]{\alpha} Q'$ and $Q \xrightarrow[s]{\beta} Q''$ then $\alpha = \beta$ and $Q' = Q''$.*

(2) *If $Q \xrightarrow[l_X . j_Y]{\tau} Q'$ then there is no transition $Q \xrightarrow[l_X]{\alpha} Q''$ for any α or Q''.*

The first condition ensures 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.[1] The second condition ensures that if a label for X leads to a transition of a protected sub-processes (involving a label of Y), there is no transition with the same label involving only X. For example, $P = l_1:a \ + \ l_1:\{l_2:\tau\}$ is not deterministically labelled since $P \xrightarrow[l_{1X} . l_{2Y}]{\tau} 0$ and $P \xrightarrow[l_1]{a} 0$.

On the other hand, $l_1:a \,.\, l_3:b \ + \ l_2:c \,.\, l_3:d$ 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.

[1]Note, however, that $l:a \,.\, P \ + \ l:a \,.\, P$ is deterministically labelled. Even though l is available twice, $l:a \,.\, P \ + \ l:a \,.\, P \xrightarrow[l_X]{a} P$ is the only transition available, so P is deterministically labelled.

Note that determining whether a labelling is deterministic is decidable, since we only consider finite processes, but there is no simple syntactic restriction to characterize such labellings. Note also that the calculus could be extended with replication to describe infinite behaviour. This, however, requires the use of a relabelling operator to ensure that a labelling remains deterministic, which adds technical complications (we refer to [Chatzikokolakis and Palamidessi 2010] for details). Such an extension is orthogonal to our goals so we chose to keep our model simple. We believe that our results extend naturally to the infinite setting, we plan to investigate this in the future.

3. GAMES AND STRATEGIES

In this section 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.

3.1 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 3.1. *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 3.2. *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 3.3. *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 3.4. *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 3.5. *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.

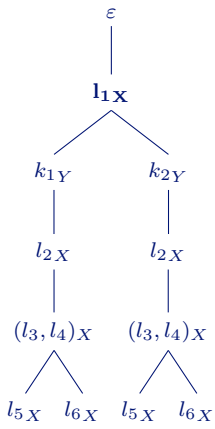
DEFINITION 3.6. *Let V be the set of valid positions for a process. For $s \in V$, we define $Ch_V(s) = \{s' \in V \mid s' = s.m \text{ for some move } m\}$. If the set V is clear, we will use the notation $Ch(s)$.*

We also define a partial function $Pl : V \to \{X, Y\}$, the player whose turn it is at V.

DEFINITION 3.7. *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 3.8. *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 $l_{1X}.k_{1Y}.l_{2X}.(l_3, l_4)_X.l_{5X}$.

3.2 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 3.9. *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:*

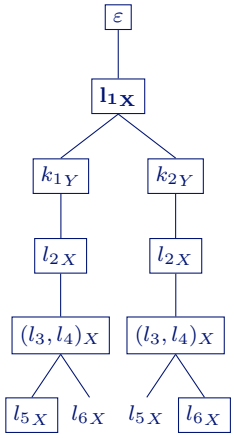
(1) $\varepsilon \in T$

(2) If $s \in T$ and $Pl(s) = Z$, then exactly one of the children of s is in T.

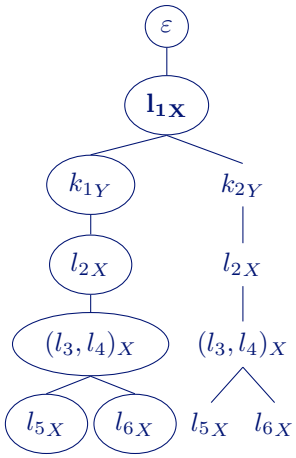
(3) If $s \in T$ and $Pl(s) = \bar{Z}$, then $Ch(s) \subseteq T$.

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.

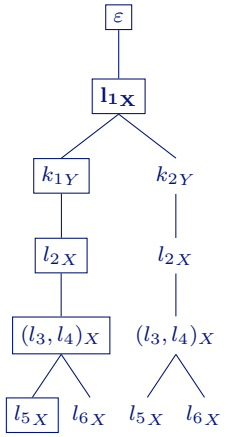
EXAMPLE 3.10. *For*
$P = (\nu b)\left(l_1 : \{k_1 : \tau \,.\, l_2 : a \,.\, l_3 : b + k_2 : \tau \,.\, l_2 : c \,.\, l_3 : b\} \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right)$*, the boxed nodes show a subtree which is a strategy for X:*



Here, the circled nodes show a strategy for Y:

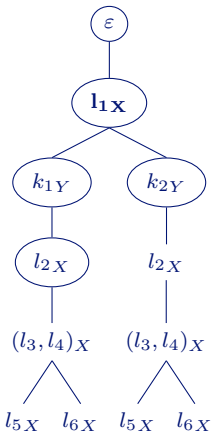


Here is a non-strategy for X:



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.

3.3 Execution of Processes According to Strategies

In this section we define the execution of a process with two strategies- one for each player.

PROPOSITION 3.11. *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 3.12. *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*

$$P \xrightarrow[s_1]{\alpha_1} P_1 \xrightarrow[s_2]{\alpha_2} P_2 \xrightarrow[s_3]{\alpha_3} ... \xrightarrow[s_{n-1}]{\alpha_{n-1}} P_{n-1} \xrightarrow[s_n]{\alpha_n} P_n$$

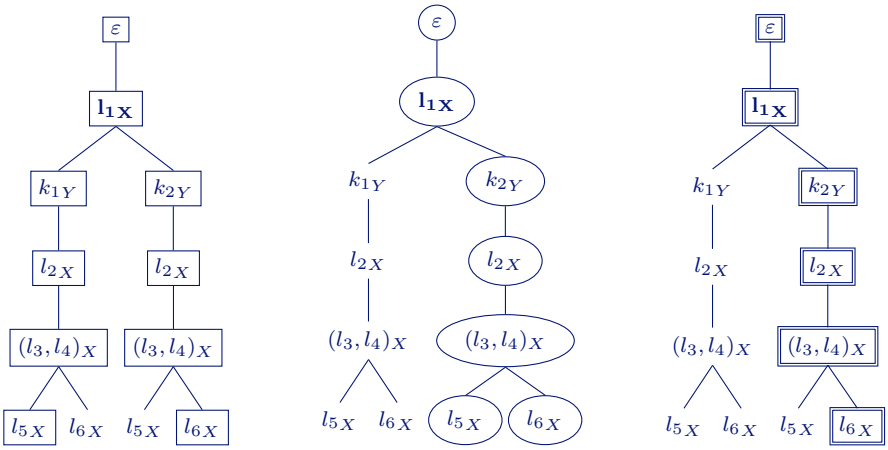
for some $\alpha_1, ..., \alpha_n$. This represents the sequence of moves that will be chosen and processes that will be reached if labels are chosen according to the strategies S_1 and S_2.

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 3.13. *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

$$(\nu b)\left(l_1 : \{k_1 : \tau \,.\, l_2 : a \,.\, l_3 : b + k_2 : \tau \,.\, l_2 : c \,.\, l_3 : b\} \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right) \xrightarrow[l_{1X}.k_{2Y}]{\tau}$$

$$(\nu b)\left((l_2 : c \,.\, l_3 : b) \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right) \xrightarrow[l_{2X}]{c} (\nu b)\left(l_3 : b \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right)$$

$$\xrightarrow[(l_3,l_4)_X]{\tau} (\nu b)\left(l_5 : d + l_6 : e\right) \xrightarrow[l_{6X}]{e} 0$$

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.

3.4 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 3.14. *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 3.15. *Let V denote the set of valid positions for a process P. If s is a valid position for P, $enabled(s, V)$ represents the set of moves available after s: define $enabled(s, V) = \{m \mid s.m \in V\}$. We use $enabled(s)$ where V is clear from the context. 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 3.16. *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.

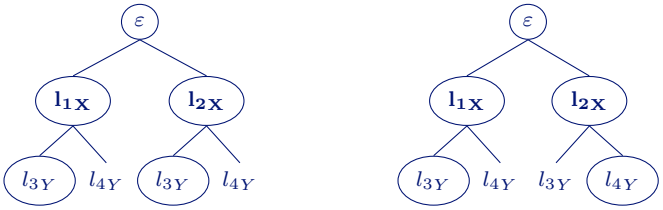
DEFINITION 3.17. *We will define the equivalence relation H_Z as $H_Z = \{(s_1, s_2) \mid Z(s_1) = Z(s_2)\}$.*

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 3.18. *In the following process, for readability, we replace labels with superscript numbers preceding actions:* ${}^{1}a.P$ *represents* $l_1 : a \,.\, P$*. As a simple example, consider the process*

$$P = {}^{1}\{\, {}^{3}\tau + {}^{4}\tau\} + {}^{2}\{\, {}^{3}\tau + {}^{4}\tau\}$$

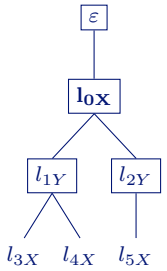
the Y*-strategy on the left respects* H_Y*, but the* Y*-strategy on the right does not:*



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 3.19. *For the process* ${}^{0}\{\, {}^{1}\tau \,.\, ({}^{3}a + {}^{4}\, b) + {}^{2}\, \tau \,.\, {}^{5}a\}$*, 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 $(s_1, s_2) \in E \longrightarrow enabled(s_1) = enabled(s_2)$, then it is evident that for any process there is a strategy respecting E.

EXAMPLE 3.20. *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 3.21. *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.

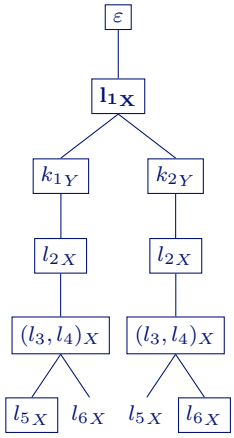
DEFINITION 3.22. *Given a process* P*, and* S *a strategy for player* Z *on* P*,* S *is* introspective *if it respects the introspection equivalence relation for* Z*.*

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 3.23. *For*

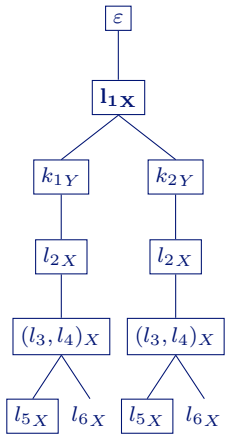
$$P = (\nu b)\left(l_1 : \{k_1 : \tau \,.\, l_2 : a \,.\, l_3 : b + k_2 : \tau \,.\, l_2 : c \,.\, l_3 : b\} \mid l_4 : \bar{b} \,.\, (l_5 : d + l_6 : e)\right)$$

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:

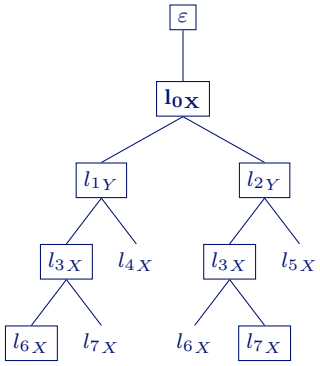


Here is an example showing why the prefixes of the valid positions are discussed in the definition of introspective. For readability, labels are replaced with superscript numbers preceding actions: ${}^1a.P$ represents $l_1 : a \,.\, P$.

EXAMPLE 3.24. *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.

4. CORRESPONDENCE BETWEEN STRATEGIES AND SCHEDULERS

In this section, we first review the syntactic schedulers defined in [Chatzikokolakis and Palamidessi 2010] 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.

4.1 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 independently 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

$$\text{ACT}\ \frac{}{l:\alpha.P \parallel \sigma(l).\rho,\eta \xrightarrow[l_X]{\alpha} P \parallel \rho,\eta}$$

$$\text{RES}\ \frac{P \parallel \rho,\eta \xrightarrow[s]{\alpha} P' \parallel \rho',\eta' \quad \alpha \neq a,\bar{a}}{(\nu a)P \parallel \rho,\eta \xrightarrow[s]{\alpha} (\nu a)P' \parallel \rho',\eta'}$$

$$\text{SUM1}\ \frac{P \parallel \rho,\eta \xrightarrow[s]{\alpha} P' \parallel \rho',\eta' \quad \rho \neq \textbf{if } L \textbf{ then } \rho_1 \textbf{ else } \rho_2}{P+Q \parallel \rho,\eta \xrightarrow[s]{\alpha} P' \parallel \rho',\eta'}$$

$$\text{PAR1}\ \frac{P \parallel \rho,\eta \xrightarrow[s]{\alpha} P' \parallel \rho',\eta' \quad \rho \neq \textbf{if } L \textbf{ then } \rho_1 \textbf{ else } \rho_2}{P|Q \parallel \rho,\eta \xrightarrow[s]{\alpha} P'|Q \parallel \rho',\eta'}$$

$$\text{SWITCH}\ \frac{P \parallel \eta,0 \xrightarrow[j_X]{\tau} P'||\eta',0}{l:\{P\} \parallel \sigma(l).\rho,\eta \xrightarrow[l_X.j_Y]{\tau} P' \parallel \rho,\eta'}$$

$$\text{COM}\ \frac{P \parallel \sigma(l).0,0 \xrightarrow[l_X]{a} P' \parallel 0,0 \qquad Q \parallel \sigma(j).0,0 \xrightarrow[j_X]{\bar{a}} Q' \parallel 0,0}{P|Q \parallel \sigma(l,j).\rho,\eta \xrightarrow[(l,j)_X]{\tau} P'|Q' \parallel \rho,\eta}$$

$$\text{IF1}\ \frac{P \parallel \rho_1,\eta \xrightarrow[s]{\alpha} P' \parallel \rho_1',\eta' \quad P \parallel \sigma(L).0,\theta \xrightarrow[s']{\beta} P'' \parallel 0,\theta' \quad \text{for some scheduler } \theta}{P \parallel \textbf{if } L \textbf{ then } \rho_1 \textbf{ else } \rho_2,\eta \xrightarrow[s]{\alpha} P' \parallel \rho_1',\eta'}$$

$$\text{IF2}\ \frac{P \parallel \rho_2,\eta \xrightarrow[s]{\alpha} P' \parallel \rho_2',\eta' \quad P \parallel \sigma(L).0,\theta \not\rightarrow \quad \text{for all schedulers } \theta}{P \parallel \textbf{if } L \textbf{ then } \rho_1 \textbf{ else } \rho_2,\eta \xrightarrow[s]{\alpha} P' \parallel \rho_2',\eta'}$$

Fig. 2. Operational semantics for processes with schedulers

with schedulers are in Fig. 2. 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 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 4.1. *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 4.2. *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 4.3. *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.

4.2 Correspondence Theorem

The main correspondence theorem can now be stated.

THEOREM 4.4. *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 4.5. *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 $Strat(\textbf{if } l \textbf{ then } \rho_1 \textbf{ else } \rho_2, s_Z)$. 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 4.6. *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 4.7. *$ext_Z(R)/I_Z$ 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 4.8. *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 4.9. *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 4.10. *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, $Sch_S(\{\varepsilon\})$.

$$Sch_S(R) = \begin{cases} 0 & \text{If } ext_z(R) = \emptyset \\ \textbf{if } = en(R_1) \textbf{ then } \sigma(mv(R_1)).Sch_S(R_1 \odot mv(R_1)) \textbf{ else} & \\ \quad \textbf{if } = en(R_2) \textbf{ then } \sigma(mv(R_2)).Sch_S(R_2 \odot mv(R_2)) \textbf{ else} & \\ \quad ... & \\ \quad\quad \textbf{if } = en(R_{k-1}) \textbf{ then } \sigma(mv(R_{k-1})).Sch_S(R_{k-1} \odot mv(R_{k-1})) & \\ \quad\quad \textbf{else } \sigma(mv(R_k)).Sch_S(R_k \odot mv(R_k)) & \text{Otherwise} \end{cases}$$

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.

■

5. GAMES FOR PROCESSES WITH PROBABILISTIC CHOICE

In this section, 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 Chatzikokolakis and Palamidessi [2010].

5.1 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. 3, 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.

$$\text{ACT}\ \frac{}{l : \alpha\,.\,P \xrightarrow[l_X]{\alpha}_1 P} \qquad \text{PROB}\ \frac{}{l : \sum_i l_i : p_i P_i \xrightarrow[l_X.l_i]{\tau}_{p_i} P_i}$$

$$\text{SUM1}\ \frac{P \xrightarrow[\lambda]{\alpha}_p P'}{P + Q \xrightarrow[\lambda]{\alpha}_p P'} \qquad \text{PAR1}\ \frac{P \xrightarrow[\lambda]{\alpha}_p P'}{P|Q \xrightarrow[\lambda]{\alpha}_p P'|Q}$$

$$\text{COM}\ \frac{P \xrightarrow[l_X]{a}_1 P' \quad Q \xrightarrow[jX]{\bar{a}}_1 Q'}{P|Q \xrightarrow[(l,j)X]{\tau}_1 P'|Q'} \qquad \text{RES}\ \frac{P \xrightarrow[\lambda]{\alpha}_p P' \quad \alpha \neq a, \bar{a}}{(\nu a)P \xrightarrow[\lambda]{\alpha}_p (\nu a)P'}$$

Fig. 3. 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 5.1. *A probabilistic process P is* deterministically labelled *if for all Q s.t. $P \longrightarrow_* Q$ the following conditions hold:*

(1) *if* $Q \xrightarrow[s]{\alpha}_{p_1} Q'$ *and* $Q \xrightarrow[s]{\beta}_{p_2} Q''$ *then* $\alpha = \beta$, $p_1 = p_2$, *and* $Q' = Q''$.

(2) *If* $Q \xrightarrow[l_X.l']{\tau}_{p} Q'$ *then there is no transition* $Q \xrightarrow[l_X]{\alpha} Q''$ *for any* α, p, Q''.

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.

5.2 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.

5.2.1 *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 5.2. *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 5.3. *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 5.4. *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 5.5. *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 5.6. *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.*

5.2.2 *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 5.7. *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 5.8. *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 $Ch(s) \subseteq T$.

5.2.3 *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 5.9. *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 5.10. *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 5.11. *If S is a strategy for process P, define $\mu_P : final(S) \longrightarrow [0,1]$ as follows: for $s \in final(S)$, if $P \underset{s}{\overset{}{\longrightarrow}}_{p} P'$, then $\mu_P(s) = p$.*

We will prove that μ_P is indeed a probability measure, but first we need an auxiliary definition.

DEFINITION 5.12. *For S a strategy, define*

$$S/s = \{s' \mid s.s' \in S\}.$$

THEOREM 5.13. *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, $P \underset{\varepsilon}{\overset{}{\longrightarrow}}_{1} P$, 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 $P \underset{m}{\overset{}{\longrightarrow}}_{1} P'$, 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 $P \xrightarrow[p_i]{l.l_i} P_i$, by definition 5.2, we have that $\mu_P(l.l_i.s') = p_i \cdot \mu_{P_i}(s')$. So altogether,

$$\begin{array}{lll} \sum_{s \in final(S)} \mu_P(s) & = & \sum_{i=1}^{n} \mu_P(l.l_i \odot final(S_i)) \\ & = & \sum_{i=1}^{n} p_i \cdot \mu_{P_i}(final(S_i)) \\ & = & \sum_{i=1}^{n} p_i \qquad \text{by induction hypothesis} \\ & = & 1 \qquad \text{by definition} \end{array}$$

∎

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.

6. 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.

6.1 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 $s = s'.L_Z.L_Z^1.L_Z^2...L_Z^n$ 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 $\neg(\neg\phi_1 \wedge \neg\phi_2)$, $\phi_1 \longrightarrow \phi_2$ for $\neg\phi_1 \vee \phi_2$, and $\phi_1 \Leftrightarrow \phi_2$ for $(\phi_1 \longrightarrow \phi_2) \wedge (\phi_2 \longrightarrow \phi_1)$. 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.

6.2 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) $\bigcirc_m\phi \longrightarrow \neg \bigcirc_m \neg\phi$.

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) $\phi \longrightarrow \neg \bigcirc_m \neg \ominus \phi$.

This formula is true because our states have a tree structure: there is at most one immediate previous state for any valid position.

(3) $C_Z(L) \longrightarrow \ominus A_Z(L)$.

This formula says that if a move was chosen in the previous state, it must have been available there.

(4) $A_Z(L) \longrightarrow \bigcirc_{L_Z} C_Z(L)$.

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 Kripke [1963], are true:

(1) $K_Z \phi \longrightarrow \phi$.

This can be interpreted as saying that knowledge is true.

(2) $K_Z \phi \longrightarrow K_Z K_Z \phi$.

This means that the agents are aware that they know what they know.

(3) $(K_Z(\phi \longrightarrow \psi) \wedge K_Z \phi) \longrightarrow K_Z \psi$.

Agents can reason and form new knowledge from what they know.

(4) $\neg K_Z \phi \longrightarrow K_Z \neg K_Z \phi$.

If an agent does not know something, it is aware of this fact.

6.3 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 6.1. *Recall from Definition 3.17 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 6.2. *Recall from Example 3.20 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 6.3. *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 $\{\ominus^k(C_Z(L)) \mid k < n,\ Z \in \{X, Y\},\ \text{and } L \text{ is any move}\}$.*

EXAMPLE 6.4. *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 3.21 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 6.5. *$s I_Z t$ 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

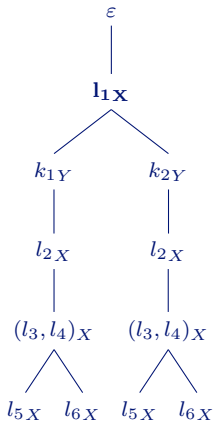
$$(@_Z \ominus)^n A_Z(L)$$

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 6.6. *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,

$$\begin{array}{lll} l_{1X}.k_{1Y}.l_{2X} & \models @_X \ominus A_X(l_2) & \textit{because} \\ l_{1X}.k_{1Y}.l_{2X} & \models \ominus A_X(l_2) & \textit{because} \\ l_{1X}.k_{1Y} & \models A_X(l_2) & \textit{because } l_{1X}.k_{1Y}.l_{2X} \in Ch(l_{1X}.k_{1Y}) \end{array}$$

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,

$$(l_{1X}.k_{1Y}.l_{2X}.(l_3, l_4)_X,\ l_{1X}.k_{2Y}.l_{2X}.(l_3, l_4)_X) \in I_X$$

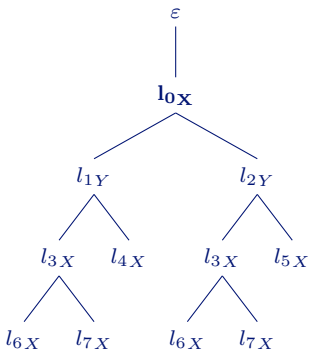
Both of these positions model the following formulas in the characterizing class:

$$\begin{array}{ll} A_X(l_5) & A_X(l_6) \\ @_X C_X((l_3,l_4)) & @_X \ominus A_X((l_3,l_4)) \\ @_X \ominus @_X C_X(l_2) & @_X \ominus @_X \ominus A_X(l_2) \\ @_X \ominus @_X \ominus @_X C_X(l_1) & @_X \ominus @_X \ominus @_X \ominus A_X(l_1) \end{array}$$

and neither of them models any other formula in the characterizing class.

EXAMPLE 6.7. *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 $l_{0X}.l_{2Y}.l_{3X} \not\models @_X \ominus A_X(l_4)$.

6.4 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 6.8. *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 $\phi \in \Phi$, $\phi \longrightarrow K_Z\phi$, and $\neg\phi \longrightarrow K_Z\neg\phi$. 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 $\phi \in \Phi$, $\phi \longrightarrow K_Z\phi$ and $\neg\phi \longrightarrow K_Z(\neg\phi)$. 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. ∎

7. CONCLUSIONS AND FUTURE WORK

In this paper 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 [Mohalik and Walukiewicz 2003], 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 allow 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 [Halpern and Moses 1984] as an explicit acknolwedgment 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 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. In work underway the second and fourth authors have developed such a logic and are working on a completeness proof. This is a rather more substantial undertaking than the fairly elementary treatment of the modal logic given here.

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 [Abramsky et al. 2000; Hyland and Ong 2000]. However, it might be illuminating to understand restrictions on strategies like innocence in epistemic terms explicitly. 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 [Shannon 1948]. We have used these information theoretic ideas for an analysis of anonymity [Chatzikokolakis et al. 2008], indeed it was that investigation that sparked the research reported in Chatzikokolakis and Palamidessi [2010] 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 Krasucki et al. [1990] where they quantify the amount of information shared when agents possess common knowledge. Of course, this is very speculative at this point.

ACKNOWLEDGMENTS

This research was supported by a grant from NSERC and by an INRIA-McGill grant.

REFERENCES

Abadi, M. and Fournet, C. 2001. Mobile values, new names, and secure communication. In *28th Annual Symposium on Principles of Programming Languages (POPL)*. ACM, 104–115.

Abramsky, S. and Jagadeesan, R. 1994. Games and full completeness for multiplicative linear logic. *J. Symbolic Logic 59,* 2, 543–574.

Abramsky, S., Jagadeesan, R., and Malacaria, P. 2000. Full abstraction for PCF. *Information and Computation 163*, 409–470.

Beauxis, R. and Palamidessi, C. 2009. Probabilistic and nondeterministic aspects of anonymity. *Theoretical Computer Science 410,* 41, 4006–4025.

Chadha, R., Delaune, S., and Kremer, S. 2009. Epistemic logic for the applied pi calculus. In *Proceedings of FMOODS/FORTE 2009*. 182–197.

Chatzikokolakis, K., Norman, G., and Parker, D. 2009. Bisimulation for demonic schedulers. In *Proc. of the Twelfth International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2009)*, L. de Alfaro, Ed. Lecture Notes in Computer Science, vol. 5504. Springer, York, UK, 318–332.

Chatzikokolakis, K. and Palamidessi, C. 2010. Making random choices invisible to the scheduler. *Information and Computation 208,* 6, 694–715.

Chatzikokolakis, K., Palamidessi, C., and Panangaden, P. 2008. Anonymity protocols as noisy channels. *Inf. and Comp. 206,* 2–4, 378–401.

Danos, V. and Harmer, R. 2001. The anatomy of innocence. *Lecture Notes in Computer Science 2142*, 188–202.

Dechesne, F., Mousavi, M., and Orzan, S. 2007. Operational and epistemic approaches to protocol analysis: Bridging the gap. In *Proceedings of the 14th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'07)*, N. Dershowitz and A. Voronkov, Eds. Lecture Notes in Computer Science, vol. 4790. Springer, 226–241.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. 1995. *Reasoning About Knowledge.* MIT Press.

Halpern, J. Y. and Moses, Y. 1984. Knowledge and common knowledge in a distributed environment. In *Proc. of Principles of Distributed Computing*. 50–61.

Hennessy, M. and Milner, R. 1980. On observing nondeterminism and concurrency. In *Automata, Languages and Programming*, J. de Bakker and J. van Leeuwen, Eds. Lecture Notes in Computer Science, vol. 85. Springer Berlin / Heidelberg, 299–309.

Hennessy, M. and Milner, R. 1985. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM 32,* 1, 137–162.

Hughes, D. and Shmatikov, V. 2004. Information hiding, anonymity and privacy: a modular approach. *Journal of Computer Security 12,* 1, 3–36.

Hyland, J. M. E. and Ong, C.-H. L. 2000. On full abstraction for PCF. *Information and Computation 163*, 285–408.

Kramer, S., Palamidessi, C., Segala, R., Turrini, A., and Braun, C. 2009. A quantitative doxastic logic for probabilistic processes and applications to information-hiding. *The Journal of Applied Non-Classical Logics 19,* 4, 489–516.

Krasucki, P., Ndjatou, G., and Parikh, R. 1990. Probabilistic knowledge and probabilistic common knowledge. In *ISMIS 90*. North Holland, 1–8.

Kripke, S. 1963. Semantical analysis of modal logic. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik 9*, 67–96.

Mohalik, S. and Walukiewicz, I. 2003. Distributed games. In *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science*, P. Pandya and J. Radhakrishnan, Eds. Lecture Notes in Computer Science, vol. 2914. Springer Berlin, Heidelberg, 338–351.

Pacuit, E. and Simon, S. 2010. Reasoning with protocols under imperfect information. In *Advances in Modal Logic*.

Schneider, S. and Sidiropoulos, A. 1996. CSP and anonymity. In *Proc. of ESORICS*. LNCS, vol. 1146. Springer, 198–218.

Shannon, C. 1948. A mathematical theory of communication. *Bell System Technical Journal 27*, 379–423,623–656.

van Benthem, J. 1976. Modal correspondence theory. Ph.D. thesis, University of Amsterdam.

van Benthem, J. 1983. *Modal Logic and Classical Logic*. Bibliopolis.

Received June 10, 2010. Revised version: August 5, 2011.