Interpreted systems semantics for process algebra with identity annotations

Francien Dechesne[1] and Mohammad Reza Mousavi[2]

[1] Philosophy Section, Faculty of Technology, Policy and Management
Delft University of Technology, The Netherlands
[2] Department of Computer Science
Eindhoven University of Technology, The Netherlands

Abstract. Process algebras have been developed as formalisms for specifying the behavioral aspects of protocols. Interpreted systems have been proposed as a semantic model for multi-agent communication. In this paper, we connect these two formalisms by defining an interpreted systems semantics for a generic process algebraic formalism. This allows us to translate and compare the vast body of knowledge and results for each of the two formalisms to the other and perform epistemic reasoning, e.g., using model-checking tools for interpreted systems, on process algebraic specifications. Based on our translation we formulate and prove some results about the interpreted systems generated by process algebraic specifications.

1 Introduction

1.1 Motivation

Process algebras [29, 19, 3] have evolved in the past three decades into a rich theory of behavioral specification for concurrent and distributed systems. Various process algebras come equipped with rich syntax, rigorous semantics and strong equational and operational reasoning techniques.

Interpreted systems (ISs) [15, 30, 17, 14] have started around the same time as process algebras and have gained popularity as semantic models to include epistemic aspects into multi-agent systems. Since then much research has been devoted into both theory and implementation of interpreted systems.

In this paper, we propose an interpreted systems semantics for a generic process algebra, thereby establishing a link between these two worlds. This link allows one to translate the vast body of knowledge in each of the two realms to the other and benefit from the tools available for both formalisms when dealing with a multi-agent system (e.g., by using model-checkers for interpreted systems for protocols specified in process algebra). Also, algebraic structures of processes and their equational theory can be used to compositionally reason about logical properties, see, e.g., [1].

1.2 Related Work

In this paper we aim to show how Interpreted Systems can provide semantics for a generic process algebraic formalism. This paves the way for using model-checking tools based on interpreted systems, e.g., MCMAS [27] and MCK [28], for verifying epistemic properties of process algebraic specifications. This also relates to the line of work in translating operational specification languages to the input languages of the above-mentioned model checkers, see, e.g., [5]. A subsequent goal for the research initiated by this paper is to characterize the class of IS models for different process algebras.

The benefits of combining behavioral (e.g., process algebraic) and epistemic formalisms (e.g., epistemic logic) have been noted by several authors, starting from Halpern and Moses in the seminal [15]. There already exists a rich literature on such combinational frameworks, especially in the application area of security protocols [6, 25, 7, 36, 11]. Our work in [9] contributed to this body of knowledge by providing a combinational framework for verifying a rich epistemic temporal language on a process algebraic formalism. Our study of interpreted systems for process algebras is based on the process algebraic framework proposed in [9]. This framework bears relation to the epistemic systems of [34] by the concept of an *appearance map* (renaming function in our framework). Recently, the process-algebraic framework of [9] has been extended in [25] to support probabilistic constructs. In [6], an epistemic temporal logic for the applied pi-calculus is presented (although [6] only allows for single-agent knowledge).

We mention [26], which aims at axiomatizations for interesting classes of ISs (notably *hypercubes*), through a characterization of the epistemic dimension of those ISs as a subclass of S5 Kripke models. This work however disregards the temporal dimension of the ISs.

The intention of our paper relates to the work of Van Benthem et al. on exploring the interface between the Dynamic Epistemic- (DEL) and Epistemic Temporal- (ETL) frameworks [4, 21]. Their line of work compares and merges the two, and characterizes the classes of models for ETL generated from DEL models.

There have been several attempts to define a knowledge-based semantics for programming languages [24, 20, 23, 12, 37, 13]. The closest to our work are [20, 23], where a knowledge-based semantics is given to a CSP-like process algebra with local states and assignments. The fundamental difference between the approach of [20, 23] and that of the present paper is that there, each agent is supposed to be represented by a sequential process, while in our approach agents may have different observations and perceptions of process algebraic actions and they need not be (although can be) incarnated in a particular process. In other words, in our approach there needs not be a one-to-one mapping between agents and processes.

1.3 Structure of the Paper

In Section 2, we present our generic process algebraic formalism called *CCSi*. The basic definitions of interpreted systems are recalled in Section 3. Then, the

semantics of *CCSi* in terms of an interpreted system is presented in 4. Some formal results about the semantic framework are presented in Section 5 and the paper is concluded in Section 6.

2 *CCSi*: A Process Algebraic Formalism

2.1 Syntax

The basic building blocks of a behavioral specification in a process algebra are *atomic actions*. They represent sending, receiving or communicating (synchronizing on) messages as well as internal computations that may or may not be visible to the observers. Atomic actions are composed using various composition operators, leading to myriad process algebras. Here, we confine ourselves to a simple process algebra, inspired by the well-known Calculus of Communicating Systems (CCS) [29]. The formalism studied in this paper (and also in the earlier work) is called *CCSi*, for CCS with identities. We slightly extend our earlier definition of *CCSi* with termination constant ϵ and unbounded choice to allow for infinite branching. We have intentionally chosen for a process algebra with few composition operators to illustrate the ideas. The constructions presented here can be extended in a straight-forward manner to various other process algebras such as those introduced in [19, 3].

Let $\mathcal{A}ct$ be a finite set of *action names* ($a, b, a_0, \ldots$ and $!a, ?a, \ldots$), and let $\mathcal{I}d$ be a finite set of *identities* (of the participating principals or agents) typically denoted by $i, j, \ldots i_1, i_2, \ldots$. Action letters preceded by a question mark or exclamation mark $?a$ and $!a$ represent the receiving and the sending parts of a communication, respectively, which result through synchronization in the communication a.[3] We let Greek letters $\alpha, \beta, \ldots$ range over the complete set of actions $\mathcal{A}ct$ (including the sending- and receiving parts), while letters $a, b, \ldots$ only range over actions without question- and exclamation marks.

Processes in *CCSi* are specified using *decorated actions* $d \in D ::= \{(\mathtt{J})\alpha | \mathtt{J} \subseteq \mathcal{I}d, \alpha \in \mathcal{A}ct\}$, and a global renaming function $\rho : \mathcal{A}ct \to \mathcal{A}ct \cup \{\tau\}$. The intuitive meaning of a decorated action $(\mathtt{J})\alpha$ is that action α is taken visibly to the principals in $\mathtt{J}$ (the intended audience of this α). Principals not in $\mathtt{J}$ will observe the so-called *public appearance* $\rho(\alpha)$ of α. In the signature of ρ, τ denotes the "silent appearance" of an action; it is assumed that $\tau \notin \mathcal{A}ct$ and for any other action α, if $\rho(\alpha) = \tau$, then $(\mathtt{J})\alpha$ becomes unobservable to the principals not in $\mathtt{J}$. We abbreviate $(\mathcal{I}\mathtt{d})\alpha$, i.e., an action visible to everyone, by α. *CCSi*-Processes are then specified as follows, together with a renaming function ρ:

$$Proc \ ::= \ \epsilon \mid D \mid Proc ; Proc \mid Proc \,||\, Proc \mid \Sigma_{i \in I} Proc_i$$

[3] Here we take a variation on standard CCS, where successful synchronization of a send- and receive action results in a silent action τ. We take the synchronization a in our framework to be the successful communication of a message (which is not a silent action).

where termination process ϵ cannot make any transition, but terminates, $Proc\,;Proc$ denotes sequential composition, $Proc\,||\,Proc$ denotes parallel composition, and $\Sigma_{i\in I}Proc_i$ denotes nondeterministic choice among processes $Proc_i$, for each i in the non-empty (and possibly infinite) index set I. We will write $p_1+\ldots+p_m$ to denote the finite nondeterministic choice $\Sigma_{\{1,\ldots,m\}}p_i$.)

The combination of identity annotations on actions and the action renaming provides different views on the behavior of the system, according to different principals.[4] Modeling passive observation of a system by hiding parts of it to specific principals is already done in the literature (e.g., in [35]), but we will generate the views for all principals simultaneously. This enables talking about properties such as "i knows that j knows that k has communicated message a".

Note that in this formalism there is no direct correspondence between the processes ($Proc$) and agents ($\mathcal{I}d$). We are not so much interested in how the principals behave (what their actions are), but we are interested in what they get to know from what happens. This is in line with the approach taken in the seminal [15], where agents are process*ors*, not process*es* (their framework focuses on knowledge and does not specify the behavior of the system explicitly). But it differs with some earlier work which uses process algebra for the specification of multi-agent systems such as [20, 23, 12, 37, 13], where agents *are* modeled as processes. In our approach, an agent may take part in several processes and a process may comprise actions that are visible (communicated by/to) many different principals. This is useful in the behavioral specification of protocols, where an agent may participate in different threads of communications and a single thread may be involved in several synchronisations with different agents.

Example 1. **(Toy Example: Syntax)** Let $\mathcal{A}ct = \{a, a_0, b, c\}$. We elaborate the definitions throughout the paper for the following simple *CCSi*-process p and renaming function ρ.

$$p \doteq ((\mathsf{1})?a\,||\,(\mathsf{2})!a) + ((\mathsf{3})b\,;c)$$

$$\rho(a) = \rho(a_0) \doteq a_0, \rho(b) \doteq \tau, \rho(c) \doteq c$$

Process p features a non-deterministic choice between the following two options:

1. synchronizing on action a; the result of synchronization is directly visible to principals 1 and 2, while the rest precieve this as action a_0, or
2. performing an action b followed by c. Action b is only visible to principal 3, and the rest of the principals do not even notice that an action has taken place. Action c is visible to all principals. Note that as defined earlier, action c abbreviates the decorated action $(\mathcal{I}\mathtt{d})c$ (everyone sees c as it happens).

[4] We will see that the send- ($!a$) and receive ($?a$) parts of an action will not be explicit in the semantics, but only the result a of their successful communication will be. This means $\rho(?a)$ and $\rho(!a)$ can be defined arbitrarily, or be left undefined.

$$
\begin{array}{lcl}
Crypt(\mathrm{i}) & = & \sum_{b:Bool} \left(\ (\mathrm{i})?pay(\mathrm{i},b); CryptFlip(\mathrm{i},b) \ \right) \\
\\
CryptFlip(\mathrm{i},b) & = & \sum_{c:Bool} \left(\ (\mathrm{i})flip(\mathrm{i},c); CryptShare(\mathrm{i},b,c) \ \right) \\
\\
CryptShare(\mathrm{i},b,c) & = & \sum_{d:Bool} \left(\ ((\mathrm{i})!share((\mathrm{i} \mod 2)+1,c) \,||\, (\mathrm{i})?share(\mathrm{i},d)) \, ; \right. \\
 & & \qquad \left. CryptBcast(\mathrm{i},b,c,d) \ \right) \\
\\
CryptBcast(\mathrm{i},b,c,d) & = & ((\mathrm{i})!bcast(\mathrm{i}, b \oplus c \oplus d) \\
 & & \quad || \sum_{x:Bool} ((\mathrm{i})?bcast((\mathrm{i} \mod 2)+1,x))) \, ; \\
 & & \qquad paid(\mathrm{i}, b \oplus c \oplus d \oplus x) \\
\\
Master & = & \quad (\mathtt{M})!pay(1,\top); (\mathtt{M})!pay(2,\bot) \\
 & & + \ (\mathtt{M})!pay(1,\bot); (\mathtt{M})!pay(2,\top) \\
 & & + \ (\mathtt{M})!pay(1,\bot); (\mathtt{M})!pay(2,\bot)
\end{array}
$$

Fig. 1. A *CCSi* model of the Dining Cryptographers protocol for 2 cryptographers.

Example 2. **(Dining Cryptographers: Syntax)** In this example, we give a formal specification of the Dining Cryptographers protocol [8], which has been extensively studied in the literature (e.g., in [35, 2, 22, 16, 33]). For reasons of presentational simplicity, we give a version with two cryptographers and an external observer.

In general, the scene is about a number of cryptographers (2 in our case) having dinner together. At the end, they learn that the bill has been paid by one of them, or by their master. They do not want to compromise each other's right to anonymity, but they wish to make it known to the public whether the payer was the master or not. (The usual presentation of the setting includes at least three cryptographers, in which case the paying cryptographer -if any- will remain anonymous not just to the public, but to the other cryptographers as well.) To this end, they come up with the following protocol: each neighboring cryptographer generates a shared bit, by flipping a coin; then each cryptographer computes the exclusive or (XOR) of the bits she sees (one in our case) with her own bit, and announces the result — or the flipped result, if she was herself the payer. The XOR of the publicly announced results indicates whether the payer was an insider or the master.

We specify the protocol for an external observer (O), two cryptographers (1 and 2), and the master (M). The observer is assumed to perfectly know the protocol; it tries to comprise the anonymity of the cryptographers and learn about the identity of the payer by looking at the trace of the protocol which has taken place, and comparing it with the possible traces with different payers.

A model of this protocol in our process language is shown in Figure 1.

The model is adopted and adapted from our earlier publication [9] and is close to the CSP description presented in [35], the only significant difference being that the actions are annotated with identities from the set $\mathcal{I}d = \{\mathrm{O}, 1, 2, \mathrm{M}\}$.

Note that we use parameters in the basic actions and process definitions only to provide a notational shorthand for the concrete actions and processes resulting from instantiating them. For example, $?pay(\mathrm{i}, b)$ is not defined in our process language but rather it stands for a number of instances such as $?pay(1, \top)$, $?pay(2, \bot)$ each of which are basic actions (obtained by globally replacing i and b with a member of $\{1, 2\}$and $\{\bot, \top\}$ in the process definition each time). In the description of the protocol $\oplus$ denotes exclusive or. Also the process names are syntactic sugar for the processes they define. The behavior of the ith cryptographer is specified by the process $Crypt(\mathrm{i})$ and the behavior of the whole DC system as a parallel composition of $Crypt(\mathrm{i})$'s and the $Master$ process:

$$DC_2 = Crypt(1) \,||\, Crypt(2) \,||\, Master$$

Note that the observer principal is not mentioned anywhere in the specification and will only be represented in the semantic model of the protocol.

A cryptographer process executes a series of actions corresponding to the three big steps of the protocol: decide whether to pay or not, flip the coins together with the neighbors, and announce the result of XOR-ing the two coins and her own paying bit. The first step is modeled as a statement $pay(\mathrm{i}, b)$, which is in fact a communication step with the $Master$.

The second step is modeled by the processes $CryptFlip(\mathrm{i})$ and $CryptShare(\mathrm{i})$. Process $Crypt(\mathrm{i})$ executes a *flip* action and then shares the result with the neighbor, by executing an action $!share$ which will synchronize with the $?share$ from the neighboring cryptographer. $CryptBcast$ models the last phase, announcing the result of one's computation ($!bcast$), receiving the results from all the others ($?bcast$) and concluding for itself that a cryptographer has paid or not ($paid(\mathrm{i}, \top)$, or $paid(\mathrm{i}, \bot)$, respectively).

The renaming function ρ specifies how much of a cryptographers' actions is visible for observing parties. For any $\mathrm{i} \in \{1, 2\}$ and $b \in \{\top, \bot\}$, we define

$$\begin{array}{lll} \rho(pay(\mathrm{i}, b)) = pay(\mathrm{i}) & \rho(bcast(\mathrm{i}, b)) = bcast(\mathrm{i}, b) & \rho(share(\mathrm{i}, c)) = share(\mathrm{i}) \\ \rho(flip(\mathrm{i}, b)) = flip(\mathrm{i}) & \rho(paid(\mathrm{i}, b)) = paid(\mathrm{i}, b) & \end{array}$$

where $pay(1)$, $bcast(1, \top)$, ... are basic actions.

2.2 Transition Systems Semantics

The operational semantics of *CCSi* (from [9]) is given in Figure 2 in terms of Structural Operational Semantics (SOS) rules [32]. The operational state of *CCSi* is a pair (p, π), where p is a process in the syntax given in Section 2.1 and π is a sequence of decorated actions (a *history*).

We include the history in the operational state of our semantics in order to capture the epistemic aspect. Such a sequence of decorated actions together with the renaming function allow us to construct in each state how each principal perceives what has happened so far. This allows us to evaluate epistemic statements on the semantics. (Note that just process terms as states only would only

code the possible future actions, and contain no information about the past - let alone code individual perceptions of the past; the semantics for process $a; p + b; p$ would, for example, after initial branching meet in a single state coded with p.) Using histories and principals' perception, we recover the notions of knowledge and knowledge update in the operational semantics of protocols. If a particular proposition is true in all operational states that are perceived the same for a particular principal, then the principal knows the proposition. The knowledge of principals is updated by appending new perceived actions to the histories.

The operational semantics of a process p is its associated labeled transitions system (with (p, ϵ) as the starting state, where ϵ denotes the empty history of decorated actions) defined by the deduction rules of Figure 2. The transitions in this LTS are of the form $\xrightarrow{a}$, which is, in turn, defined in terms of the auxiliary transition relation $\overset{(\mathrm{J})a}{\Rightarrow}$, by stripping off the decorations and blocking non-synchronized sends and receives.

Process ϵ cannot make any transition, but terminates immediately; this is denoted by the termination predicate $\surd$ in the deduction rule (axiom) (ϵ). The semantics of a decorated action is defined by the deduction rule (**a**): the process d can perform the action d (which is concatenated to the history) and then turn into the terminated process ϵ. The operational behavior of nondeterministic choice is defined by the choice in the behavior of its arguments. This is captured by the deduction rule scheme (**ni**) (for each index set I, $i \in I$). Transition semantics of sequential composition is defined by either taking an action from the first component, or termination of the first component followed by an action from the second one, as specified by deduction rules (**s0**) and (**s1**) respectively. The semantics of parallel composition is defined by the interleaving (deduction rules (**p0**)-(**p1**)) and the synchronization (deduction rules (**p2**)-(**p3**)) of the actions of its arguments (we here omit deduction rules (**p1**) and (**p3**), which are, respectively, symmetric copies of (**p0**) and (**p2**)). Termination of nondeterministic choice, sequential composition and parallel composition is specified, respectively, by (**nti**), (**st**), and (**pt**).

Deduction rule (**strip**) strips down the decorated action into plain actions (by removing the intended audience) and ignores send- and receive actions (hence, one could say it enforces synchronization among communicating processes in the trace semantics).

In addition to the operational semantics, we define an epistemic semantics for the process calculus using the indistinguishability relation $\overset{i}{\cdots}$, which is defined in terms of the indistinguishability relation $\overset{i}{=}$ (see also [9]). Considering the deduction rules for $\overset{i}{=}$, reflexivity is captured in deduction rule (**refl**); rule ($= \rho\mathbf{0}$) defines the case for a visible action to principle i; rule ($= \rho\mathbf{1}$) concerns when two invisible actions which have the same public appearance for i; rule ($= \rho\mathbf{2}$) defines the case for an action which is invisible to i but appears to i as another visible action; rule ($= \tau\mathbf{0}$) is about an invisible action which appears as τ to i (i.e., is absolutely unobservable for i). (Again for the sake of brevity, we have omitted symmetric rules for ($= \rho\mathbf{2}$) and ($= \tau\mathbf{0}$).) Deduction rule (**I**) lifts

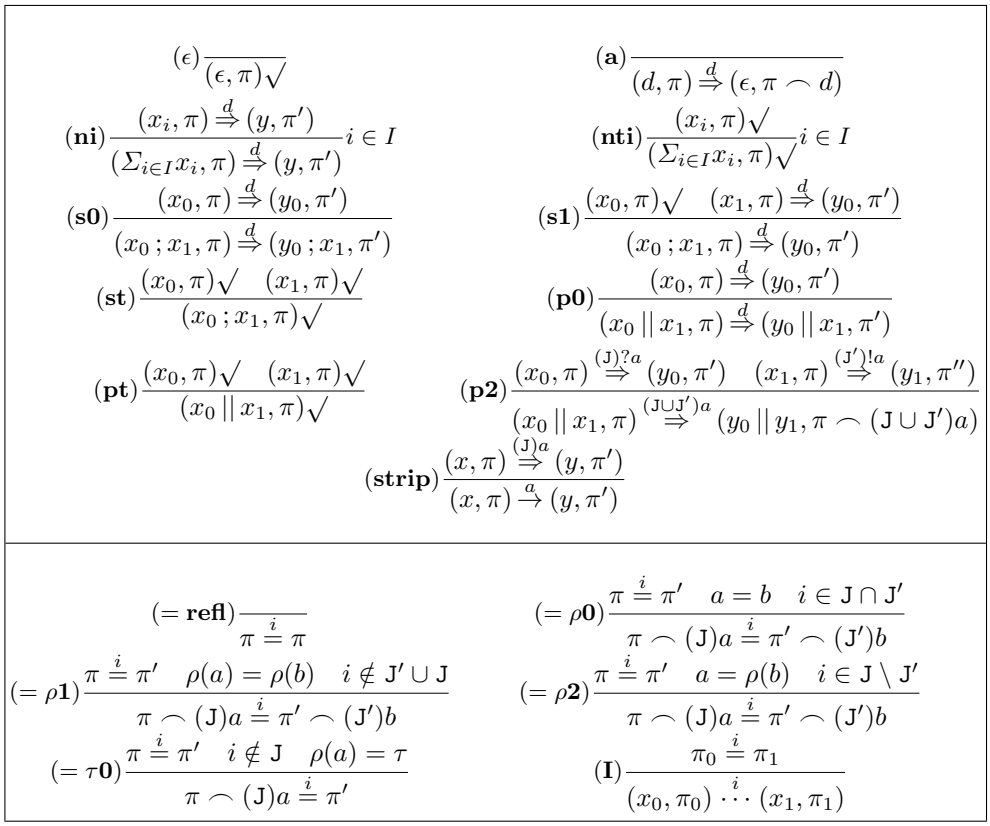

Fig. 2. SOS of *CCSi* (cf. [9])

the indistinguishability relation $\overset{i}{=}$ from sequences of decorated actions to the indistinguishability relation $\overset{i}{\cdots}$ on the operational state of *CCSi*. As shown in [9], both $\overset{i}{=}$ and $\overset{i}{\cdots}$ are equivalence relations.

This semantics is introduced here to present the original semantics given in [9] and to compare it with the interpreted systems semantics presented in the subsequent sections. For these and the following definitions, we now provide a running example for clarification.

Example 3. **(Toy Example: Semantics)** Consider process p specified in Example 1. The traces of p can be generated by the SOS-rules as follows:

1. $((1)?a,\langle\rangle)\overset{(1)?a}{\Rightarrow}(\epsilon,\langle(1)?a\rangle)$ – rule (**a**)
2. $((2)!a,\langle\rangle)\overset{(2)!a}{\Rightarrow}(\epsilon,\langle(2)!a\rangle)$ – rule (**a**)
3. $((1)?a\,||\,(2)!a,\langle\rangle)\overset{(1,2)a}{\Rightarrow}(\epsilon\,||\,\epsilon,\langle(1,2)a\rangle)$ – rule (**p2**), using 1,2
4. $(\epsilon\,||\,\epsilon,\langle(1,2)a\rangle)\surd$ – rules (ϵ), (**pt**)
5. $((3)b,\langle\rangle)\overset{(3)b}{\Rightarrow}(\epsilon,\langle(3)b\rangle)$ – rule (**a**)

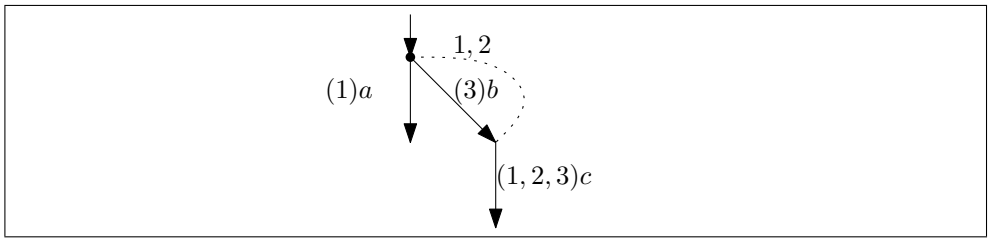

Fig. 3. Operational Semantics of the Toy Example

6. $((\mathsf{3})b\,;c,\langle\rangle) \overset{(\mathsf{3})b}{\Rightarrow} (\epsilon\,;c,\langle(\mathsf{3})b\rangle)$ – rule (**s0**), using 5
7. $(c,\langle(\mathsf{3})b\rangle) \overset{c}{\Rightarrow} (\epsilon,\langle(\mathsf{3})b,c\rangle)$ – rule (**a**)
8. $(\epsilon,\langle(\mathsf{3})b,c\rangle)\surd$ – rule (ϵ)
9. $(\epsilon\,;c,\langle(\mathsf{3})b\rangle) \overset{c}{\Rightarrow} (\epsilon,\langle(\mathsf{3})b,c\rangle)$ – rule (**s1**), using 7,8
10. $((\mathsf{1})?a\,||\,(\mathsf{2})!a,\langle\rangle) \overset{(\mathsf{1})?a}{\Rightarrow} (\epsilon\,||\,(\mathsf{2})!a,\langle(\mathsf{1})?a\rangle) \overset{(\mathsf{2})!a}{\Rightarrow} (\epsilon\,||(\epsilon,\langle(\mathsf{1})?a,(\mathsf{2})!a\rangle)$ – rules (**p0**), (**p1**), using 1,2
11. $((\mathsf{1})?a\,||\,(\mathsf{2})!a,\langle\rangle) \overset{(\mathsf{2})!a}{\Rightarrow} ((\mathsf{1})?a\,||\,\epsilon,\langle(\mathsf{2})!a\rangle) \overset{(\mathsf{1})?a}{\Rightarrow} (\epsilon\,||\,\epsilon,\langle(\mathsf{2})!a,(\mathsf{1})?a\rangle)$ – rule (**p1**), (**p0**), using 2,1
12. $(p,\langle\rangle) \overset{(\mathsf{1},\mathsf{2})a}{\Rightarrow} (\epsilon\,||\,\epsilon,\langle(\mathsf{1},\mathsf{2})a\rangle)$ – rule (**ni**), using 3
13. $(p,\langle\rangle) \overset{(\mathsf{1})?a}{\Rightarrow} (\epsilon\,||\,(\mathsf{2})!a,\langle(\mathsf{1})?a\rangle) \overset{(\mathsf{2})!a}{\Rightarrow} (\epsilon\,||\,\epsilon,\langle(\mathsf{1})?a,(\mathsf{2})!a\rangle)$ – rule (**ni**) using 10
14. $(p,\langle\rangle) \overset{(\mathsf{2})!a}{\Rightarrow} ((\mathsf{1})?a\,||\,\epsilon,\langle(\mathsf{2})!a\rangle) \overset{(\mathsf{1})?a}{\Rightarrow} (\epsilon\,||\,\epsilon,\langle(\mathsf{2})!a,(\mathsf{1})?a\rangle)$ – rule (**ni**) using 11
15. $(p,\langle\rangle) \overset{(\mathsf{3})b}{\Rightarrow} (\epsilon\,;c,\langle(\mathsf{3})b\rangle) \overset{c}{\Rightarrow} (\epsilon,\langle(\mathsf{3})b,c\rangle)$ – rule (**n1**) using 6, followed by 9
16. $(p,\langle\rangle) \overset{a}{\rightarrow} (\epsilon\,||\,\epsilon,\langle(\mathsf{1},\mathsf{2})a\rangle)$ – rule (**strip**), using 12
17. $(p,\langle\rangle) \overset{b}{\rightarrow} (\epsilon\,;c,(\mathsf{3})b) \overset{c}{\rightarrow} (\epsilon,\langle(\mathsf{3})b,c\rangle)$ – rule (**strip**) using 15

Note again that rule (**strip**) is restricted to 'closed' actions, i.e., excluding $!a, ?a$, so it does not apply to lines 12 and 13. The set of traces of p is therefore (lines 15 and 16): $\{\langle a\rangle,\langle b,c\rangle\}$.

Now, we also generate the indistinguishability relation through the SOS-rules:

1. For all $i \in \mathcal{I}d$ and for all π: $\pi \overset{i}{=} \pi$ – rule (= **refl**)
2. $\langle(1,2)a\rangle \overset{3}{=} \langle a_0\rangle$ – rule (= ρ**2**)
3. $\langle(\mathsf{3})b,c\rangle \overset{1}{=} \langle c\rangle$ – rule (= τ**0**)
4. $\langle(\mathsf{3})b,c\rangle \overset{2}{=} \langle c\rangle$ – rule (= τ**0**)

The state space of this example is depicted in Figure 3. In this figure, the initial state is designated with a small incoming arrow. The transitions derived from the operational semantics are drawn as solid arrows and the indistinguishability relation is drawn as a dotted line labeled with the principal identities. (In order not to clutter the figure, we dispensed with the self-loops denoting the reflexivity of the indistinguishability relation.)

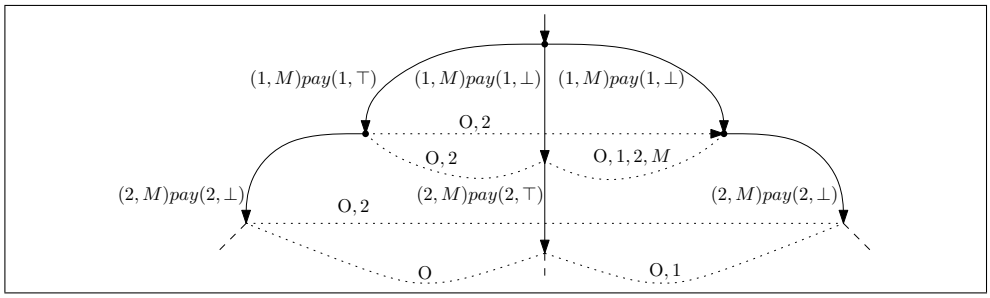

Fig. 4. Operational Semantics of the Dining Cryptographers Protocol

Example 4. **(Dining Cryptographers: Semantics)** Consider the specification of dining cryptographers given in Example 2. The complete state space of the protocol is too large to be studied manually (see [10] for a prototype implementation of a model-checker for a process algebra, using which we performed a mechanized analysis of this protocol). The initial steps of the protocol are depicted in Figure 4. Consider, for example, the leftmost and the middle traces in Figure 4. After the first step of the protocol, principals 1 and 2 observe action $pay(1, \top)$ in the leftmost trace and action $pay(1, \bot)$ in the middle trace and hence, can distinguish the target states of these two actions. Principals O and 2, however, observe $pay(1)$ in both cases and hence the resulting states are indistinguishable to them. After the second step, principal 2 can also distinguish between the two traces, because it can observe $pay(2, \bot)$ as the second action of the leftmost trace, while it can observe $pay(2, \top)$ as the second action of the middle trace. Principal O still cannot distinguish the two traces because the second action appears in both cases as $pay(2)$ to it. Note that modeling this aspect of knowledge about the actions that have taken place and revisions in the knowledge is made possible thanks to the notion of history (of past actions) that is included in the operational state.

Below, we give three completed traces of the protocol, which are continuations of the three initial branches depicted in Figure 4. The protocol starts with the $Master$ synchronising on pay-actions with each cryptographer. These traces do show the essence of the protocol (i.e., the three possible cases for payment) with some choice of coin flips. Our choices for coin flips may seem arbitrary at the first glance, but actually, a particular choice is made to demonstrate how the protocol guarantees anonymity:

1. The first trace, given below, is a continuation of the leftmost trace, in which the first cryptographer has paid; the result of both coin flips in this particular trace is a head ($\top$). For the sake of brevity, in the description of the traces, we only mention the histories and the transitions and omit the process expressions:

$$(DC_2, \langle\rangle) \xrightarrow{pay(1,\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\top)\rangle) \xrightarrow{pay(2,\bot)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\top), (\mathtt{2},\mathtt{M})pay(2,\bot)\rangle) \xrightarrow{flip(1,\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\top), (\mathtt{2},\mathtt{M})pay(2,\bot), (\mathtt{1})flip(1,\top)\rangle) \xrightarrow{flip(2,\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\top), (\mathtt{2},\mathtt{M})pay(2,\bot), (\mathtt{1})flip(1,\top), (\mathtt{2})flip(2,\top)\rangle) \xrightarrow{share(1,\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\top), (\mathtt{2},\mathtt{M})pay(2,\bot), (\mathtt{1})flip(1,\top), (\mathtt{2})flip(2,\top), ((\mathtt{1},\mathtt{2}))share(1,\top)\rangle) \xrightarrow{share(2,\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\top), (\mathtt{2},\mathtt{M})pay(2,\bot), (\mathtt{1})flip(1,\top), (\mathtt{2})flip(2,\top), ((\mathtt{1},\mathtt{2}))share(1,\top), ((\mathtt{1},\mathtt{2}))share(2,\top)\rangle) \xrightarrow{bcast(1,\top\oplus\top\oplus\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\top), (\mathtt{2},\mathtt{M})pay(2,\bot), (\mathtt{1})flip(1,\top), (\mathtt{2})flip(2,\top), ((\mathtt{1},\mathtt{2}))share(1,\top), ((\mathtt{1},\mathtt{2}))share(2,\top), ((\mathtt{1},\mathtt{2}))bcast(1,\top)\rangle) \xrightarrow{bcast(2,\bot\oplus\top\oplus\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\top), (\mathtt{2},\mathtt{M})pay(2,\bot), (\mathtt{1})flip(1,\top), (\mathtt{2})flip(2,\top), ((\mathtt{1},\mathtt{2}))share(1,\top), ((\mathtt{1},\mathtt{2}))share(2,\top), ((\mathtt{1},\mathtt{2}))bcast(1,\top), ((\mathtt{1},\mathtt{2}))bcast(2,\bot)\rangle) \xrightarrow{paid(1,\top\oplus\top\oplus\top\oplus\bot)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\top), (\mathtt{2},\mathtt{M})pay(2,\bot), (\mathtt{1})flip(1,\top), (\mathtt{2})flip(2,\top), ((\mathtt{1},\mathtt{2}))share(1,\top), ((\mathtt{1},\mathtt{2}))share(2,\top), ((\mathtt{1},\mathtt{2}))bcast(1,\top), ((\mathtt{1},\mathtt{2}))bcast(2,\bot), ((\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{M}))paid(1,\top)\rangle) \xrightarrow{paid(2,\bot\oplus\top\oplus\top\oplus\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\top), (\mathtt{2},\mathtt{M})pay(2,\bot), (\mathtt{1})flip(1,\top), (\mathtt{2})flip(2,\top), ((\mathtt{1},\mathtt{2}))share(1,\top), ((\mathtt{1},\mathtt{2}))share(2,\top), ((\mathtt{1},\mathtt{2}))bcast(1,\top), ((\mathtt{1},\mathtt{2}))bcast(2,\bot), ((\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{M}))paid(1,\top), ((\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{M}))paid(2,\top)\rangle) \quad \surd$$

As it can be seen, upon termination, the history indicates that both principals 1 and 2 have announced that a cryptographer has paid and this announcement can be observed by each and every principal.

2. The second trace, given below, is a continuation of the trace in the middle of Figure 4, in which the second cryptographer has paid, the result of the coin flip by crytographer 1 is a head ($\top$) and that of the cryptographer 2 is a tail ($\bot$):

$(DC_2, \langle\rangle)$ $\xrightarrow{pay(1,\bot)}$

$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot)\rangle)$ $\xrightarrow{pay(2,\bot)}$

$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot)\rangle)$ $\xrightarrow{flip(1,\top)}$

$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top)\rangle)$ $\xrightarrow{flip(2,\bot)}$

$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\bot)\rangle)$ $\xrightarrow{share(1,\top)}$

$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\bot),$
$((\mathtt{1},\mathtt{2}))share(1,\top)\rangle)$ $\xrightarrow{share(2,\top)}$

$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\bot),$
$((\mathtt{1},\mathtt{2}))share(1,\top),((\mathtt{1},\mathtt{2}))share(2,\top)\rangle)$ $\xrightarrow{bcast(1,\bot\oplus\top\oplus\bot)}$

$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\bot),$
$((\mathtt{1},\mathtt{2}))share(1,\top),((\mathtt{1},\mathtt{2}))share(2,\top),((\mathtt{1},\mathtt{2}))bcast(1,\top)\rangle)$ $\xrightarrow{bcast(2,\top\oplus\top\oplus\top)}$

$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\bot),$
$((\mathtt{1},\mathtt{2}))share(1,\top),((\mathtt{1},\mathtt{2}))share(2,\top),$
$((\mathtt{1},\mathtt{2}))bcast(1,\bot),((\mathtt{1},\mathtt{2}))bcast(2,\bot)\rangle)$ $\xrightarrow{paid(1,\bot\oplus\top\oplus\top\oplus\top)}$

$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\bot),$
$((\mathtt{1},\mathtt{2}))share(1,\top),((\mathtt{1},\mathtt{2}))share(2,\top),$
$((\mathtt{1},\mathtt{2}))bcast(1,\bot),((\mathtt{1},\mathtt{2}))bcast(2,\bot),$
$((\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{M}))paid(1,\top)\rangle)$ $\xrightarrow{paid(2,\top\oplus\top\oplus\top\oplus\bot)}$

$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\bot),$
$((\mathtt{1},\mathtt{2}))share(1,\top),((\mathtt{1},\mathtt{2}))share(2,\top),$
$((\mathtt{1},\mathtt{2}))bcast(1,\bot),((\mathtt{1},\mathtt{2}))bcast(2,\bot),$
$((\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{M}))paid(1,\top),((\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{M}))paid(2,\top)\rangle)$ $\surd$

Similar to the previous trace, at the end of the protocol, it has been announced, by both cryptographers, that a cryptographer has paid the bill. The observer, however, cannot distinguish this trace from the first one, and hence, at the end of either of the two traces, cannot establish which cryptographer has paid. Note that the two cryptographers do know (both in the first and the second trace) who has paid the bill: for example, at the end of the above-given trace, cryptographer 2 knows that it has paid the bill, because this trace is distinguishable (by observing $paid(2, \top)$) from any other trace in which it has not paid the bill. Cryptographer 1 also knows that 2 has paid the bill, because it knows after the first step that it has not paid the bill,

and after the last step knows that a cryptographer, hence 2, has paid the bill; in other words, this trace is distinguishable from other traces in which 2 has not paid in the observable *pay* and *paid* actions.

3. The last trace is a continuation of the rightmost trace in Figure 4, in which the master has taken care of the bill and no cryptographer has paid. Both coin flips in this trace result in a head ($\top$):

$$(DC_2, \langle\rangle) \xrightarrow{pay(1,\bot)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot)\rangle) \xrightarrow{pay(2,\bot)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot)\rangle) \xrightarrow{flip(1,\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top)\rangle) \xrightarrow{flip(2,\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\top)\rangle) \xrightarrow{share(1,\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\top),$$
$$((\mathtt{1},\mathtt{2}))share(1,\top)\rangle) \xrightarrow{share(2,\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\top),$$
$$((\mathtt{1},\mathtt{2}))share(1,\top),((\mathtt{1},\mathtt{2}))share(2,\top)\rangle) \xrightarrow{bcast(1,\bot\oplus\top\oplus\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\top),$$
$$((\mathtt{1},\mathtt{2}))share(1,\top),((\mathtt{1},\mathtt{2}))share(2,\top),((\mathtt{1},\mathtt{2}))bcast(1,\bot)\rangle) \xrightarrow{bcast(2,\bot\oplus\top\oplus\top)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\top),$$
$$((\mathtt{1},\mathtt{2}))share(1,\top),((\mathtt{1},\mathtt{2}))share(2,\top),$$
$$((\mathtt{1},\mathtt{2}))bcast(1,\bot),((\mathtt{1},\mathtt{2}))bcast(2,\bot)\rangle) \xrightarrow{paid(1,\bot\oplus\top\oplus\top\oplus\bot)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\top),$$
$$((\mathtt{1},\mathtt{2}))share(1,\top),((\mathtt{1},\mathtt{2}))share(2,\top),$$
$$((\mathtt{1},\mathtt{2}))bcast(1,\bot),((\mathtt{1},\mathtt{2}))bcast(2,\bot),$$
$$((\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{M}))paid(1,\bot)\rangle) \xrightarrow{paid(2,\bot\oplus\top\oplus\top\oplus\bot)}$$

$$(-, \langle(\mathtt{1},\mathtt{M})pay(1,\bot),(\mathtt{2},\mathtt{M})pay(2,\bot),(\mathtt{1})flip(1,\top),(\mathtt{2})flip(2,\top),$$
$$((\mathtt{1},\mathtt{2}))share(1,\top),((\mathtt{1},\mathtt{2}))share(2,\top),$$
$$((\mathtt{1},\mathtt{2}))bcast(1,\bot),((\mathtt{1},\mathtt{2}))bcast(2,\bot),$$
$$((\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{M}))paid(1,\bot),((\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{M}))paid(2,\bot)\rangle)$$

After observing this trace, all principals know that the master has taken care of the bill, because they all observe both $paid(2,\bot)$ and $paid(2,\bot)$ and in all traces that are indistinguishable from the present trace (i.e., contain the same observable *paid* actions in the end) no cryptographer has paid. The latter claim can be checked formally, using an exhaustive search of the state space of the procotol; we refer to [10] for the details.

3 Interpreted Systems

In this section, we recall from [17] formal definitions, terminology and notational conventions regarding interpreted systems. Note however that we deviate from the original definition of [17], by restricting to finite runs. We do this in the context of our process language, because our process terms only afford finite behavior. In order to have infinite runs corresponding to our porcesses, we could have every run have an infinite 'stuttering' tail (as is suggested in [15] as well). We will discuss in Section 5 why we have chosen not to do so for the context of this paper.

Definition 1 (Interpreted Systems (finite depth)) *Given a set of $n > 0$ agents with identifiers in $\mathcal{I}d = \{1, \ldots, n\}$, and for each agent $i \in \mathcal{I}d$ a set of local states L_i, a global state $\bar{l}$ is an n-tuple $(l_1, \ldots, l_n)$ with $l_i \in L_i$. Let $L = \prod_{i=1}^{n} L_i$ denote the set of global states. A run r is a finite sequence of global states $r(0), r(1), \ldots, r(m)$ for some $m \in \mathbb{N}$. A protocol R is a non-empty set of runs.*

Given a set Φ of atomic logical formulae, a valuation is a function $\nu : L \to \Phi$.

An interpreted system is then a pair (R, ν), where R is a protocol and ν is a valuation.

Two global states are taken to be indistinguishable for an agent if their local states are equal. This defines an equivalence relation for the evaluation of epistemic formulas:

Definition 2 (Indistinguishability relations in ISs [17]) *Given an interpreted system with set of global states L, for each agent $i \in \mathcal{I}d$ the relation $\overset{i}{\approx} \subseteq L \times L$ is defined by: $\bar{l} \overset{i}{\approx} \bar{l}'$ iff $l_i = l'_i$.*

The valuation part ν of an Interpreted System (which is taken to be defined on the full L) can be specified independently of the protocol part R. As it turns out, linking our framework and Interpreted Systems is essentially about linking our operational semantics to the protocol part of ISs and linking the respective indistinguishability relations. Within this paper we do not yet explore with which logical language, including a meaningful choice for the atoms, it is best to talk about our epistemic-operational models for processes. This will be part of our future work, and for now, we therefore do not specify the ν-part, only the protocol part of our ISs.

4 Interpreted Systems Semantics for *CCSi*

In this section, we define an interpreted systems semantics for the process algebra *CCSi*. We do so by defining the influence of each operational step on the local state of each principal and then aggregating these influences into the definition of a run. The development of this section is only dependent on the definition of an operational semantics, as defined , and hence the same schema can be

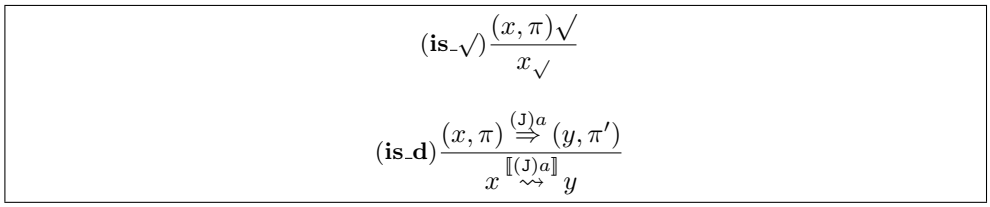

Fig. 5. Influence of a decorated action on the local states

used for any other process algebra (process algebraic formalism) as long as the visibility range (the set of principals to which the action is visible) and the public appearance of each action is provided in the operational semantics.

Before we define the interpreted systems semantics for a *CCSi* process, we first describe what we will call the local and global state of such a semantics: For a process $p \in CCSi$ with renaming function ρ, a local state $l_i \in L_i \subseteq \mathcal{A}ct^*$ is a sequence of actions in p as they appear to agent $i \in \mathcal{I}d$ under ρ. The set of global states L is defined to be $L = \prod_{i=1}^{n} L_i$. The protocol corresponding to p is the set of all runs of p, which are the sequences of global states corresponding to the traces of p. Note that unlike the histories of *CCSi*, which are sequences of *decorated* actions, the local states of *CCSi* are sequences of *actions* (without any decoration).

Definition 3 (Concatenation of actions-tuples to global states) *Consider a global state $\overline{l} = (l_1, \ldots, l_n)$ and an n-tuple $\overline{a} = (a_1, \ldots, a_n)$ of actions in $\mathcal{A}ct \cup \{\tau\}$. Then $\overline{l} \frown \overline{a} = (l'_1, \ldots, l'_n)$ where $l'_i = l_i$ if $a_i = \tau$ and $l'_i = l_i \cdot a_i$ otherwise.*

Definition 4 (Decorated action tuple) *For any decorated action $d = (\mathtt{J})a$ we define $[\![d]\!]$ to be the n-tuple $(a_1, \ldots, a_n)$ with $a_i = a$ for $i \in \mathtt{J}$, and $a_i = \rho(a)$ for $i \notin \mathtt{J}$.*

We are now ready to define the interpreted system semantics of processes by defining their associated protocols. Definition 5 defines the protocol associated with the process ϵ to be the singleton set comprising empty local sequences. The protocol associated with a decorated action, is defined by the tuple of local appearances of each action to each agent. Finally, the notion of protocol is lifted in the expected way from decorated actions to processes.

Definition 5 (Interpreted System protocols for *CCSi*) *The influence of a decorated action (or termination) on the local state of each principal is defined in Figure 5.*

We write $p_{\downarrow}$ in the remainder of this definition, to denote that process p can terminate according to the deduction rules of Figure 5, i.e., $p_{\surd}$, or cannot take any transition, i.e., there are no process p' and n-tuple of actions $\overline{a}$ s.t. $p \overset{\overline{a}}{\rightsquigarrow} p'$.

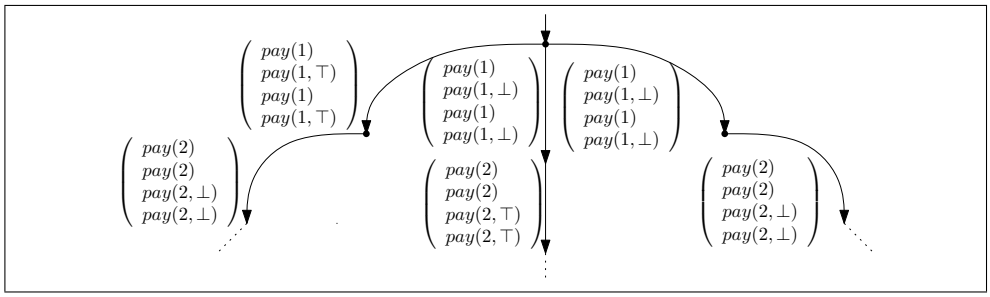

Fig. 6. Interpreted Systems Semantics of the Dining Cryptographers Protocol

A run of a process p is a sequence $(\bar{l}_0, \bar{l}_1, \ldots, \bar{l}_m)$ of global states, such that there exist processes $p_0, p_1, \ldots, p_m$ with $\bar{l}_0 = (\langle\rangle, \ldots, \langle\rangle)$, $p_0 = p$, $p_{m\downarrow}$, and (if $m > 0$) for each $k < m$, it holds that $\bar{l}_{k+1} = \bar{l}_k^{\frown} \overline{a}$ and $p_k \overset{\overline{a}}{\rightsquigarrow} p_{k+1}$.

The protocol associated with a process p, denoted as $[\![p]\!]$, is the set of all runs of p.

Example 5. **(Toy Example: Interpreted Systems)** The IS semantics for the proces $p = ((1)?a \,||\, (2)!a) + ((3)b\,;c)$ consists of the following runs:

$$(\langle\rangle, \langle\rangle, \langle\rangle), (\langle a\rangle, \langle a\rangle, \langle a_0\rangle)$$

and $(\langle\rangle, \langle\rangle, \langle\rangle), (\langle\rangle, \langle\rangle, \langle b\rangle), (\langle c\rangle, \langle c\rangle, \langle b, c\rangle)$;

We see that the process histories of decorated actions are unfolded, through the annotations and the renaming functions, into the local perspectives of the principals in the IS semantics: local states are the sequences of (undecorated) actions from the history as perceived by the corresponding principal.

Example 6. **(Dining Cryptographers: Interpreted Systems)** In Figure 6, the initial steps of the runs of dining cryptographers protocol are depicted. Each tuple depicted in Figure 6, represents the view of principals O, 1, 2, and M, respectively, of the action that has take place.

The runs of the interpreted systems semantics, corresponding to the traces given in Example 4, are given below:

1. The first run corresponds to the leftmost trace in Figures 4 and 6; the global state of the run is a 4-tuple comprising the local states of O, 1, 2, and M, respectively (to save space, we have abbreviated the action names $flip$, $share$ and $bcast$ into fl, sh and bc, respectively):

$$
\begin{pmatrix}
\langle\rangle,\\
\langle\rangle,\\
\langle\rangle,\\
\langle\rangle
\end{pmatrix},
$$

$$
\begin{pmatrix}
\langle pay(1)\rangle,\\
\langle pay(1,\top)\rangle,\\
\langle pay(1)\rangle,\\
\langle pay(1,\top)\rangle
\end{pmatrix},
$$

$$
\begin{pmatrix}
\langle pay(1), & pay(2)\rangle,\\
\langle pay(1,\top), & pay(2)\rangle,\\
\langle pay(1), & pay(2,\top)\rangle,\\
\langle pay(1,\top), & pay(2,\top)\rangle
\end{pmatrix},
$$

$$
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1)\rangle,\\
\langle pay(1,\top), & pay(2), & fl(1,\top)\rangle,\\
\langle pay(1), & pay(2,\top), & fl(1)\rangle,\\
\langle pay(1,\top), & pay(2,\top), & fl(1)\rangle
\end{pmatrix},
$$

$$
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2)\rangle,\\
\langle pay(1,\top), & pay(2), & fl(1,\top), & fl(2)\rangle,\\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot)\rangle,\\
\langle pay(1,\top), & pay(2,\top), & fl(1), & fl(2)\rangle
\end{pmatrix},
$$

$$
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1)\rangle,\\
\langle pay(1,\top), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top)\rangle,\\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top)\rangle,\\
\langle pay(1,\top), & pay(2,\top), & fl(1), & fl(2), & sh(1)\rangle
\end{pmatrix},
$$

$$
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1), & sh(2)\rangle,\\
\langle pay(1,\top), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top), & sh(2,\bot)\rangle,\\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top), & sh(2,\bot)\rangle,\\
\langle pay(1,\top), & pay(2,\top), & fl(1), & fl(2), & sh(1), & sh(2)\rangle
\end{pmatrix},
$$

$$
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top)\rangle,\\
\langle pay(1,\top), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top), & sh(2,\bot), & bc(1,\top)\rangle,\\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top), & sh(2,\bot), & bc(1,\top)\rangle,\\
\langle pay(1,\top), & pay(2,\top), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top)\rangle
\end{pmatrix},
$$

$$
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot)\rangle,\\
\langle pay(1,\top), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot)\rangle,\\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot)\rangle,\\
\langle pay(1,\top), & pay(2,\top), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot)\rangle
\end{pmatrix},
$$

$$
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot), & paid(1,\top)\rangle,\\
\langle pay(1,\top), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot), & paid(1,\top)\rangle,\\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot), & paid(1,\top)\rangle,\\
\langle pay(1,\top), & pay(2,\top), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot), & paid(1,\top)\rangle
\end{pmatrix},
$$

$$
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot), & paid(1,\top), & paid(2,\top)\rangle\\
\langle pay(1,\top), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot), & paid(1,\top), & paid(2,\top)\rangle\\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot), & paid(1,\top), & paid(2,\top)\rangle\\
\langle pay(1,\top), & pay(2,\top), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot), & paid(1,\top), & paid(2,\top)\rangle
\end{pmatrix}
$$

There are a couple of interesting observations to be made about the above-given run: firstly, each operational step of the protocol results in appending one action to the local state of each and every principal. This phenomenon, called synchronicity, is because of the particular definition of ρ, which does not map any action to the invisible action τ. Secondly, no principal can observe all actions as they actually happen in the protocol: a cryptographer cannot observe the content of the commuinication between the master and the other cryptographer and the result of its coin flip, the master cannot observe the result of the coin flip for any of the two cryptographer and the communication between the two cryptographers for sharing them, and the observer cannot observe any of the aforementioned information.

2. The second run corresponds to the middle trace in Figures 4 and 6 using the same abbreviations as in the first run:

$$
\begin{pmatrix}
\langle\rangle, \\
\langle\rangle, \\
\langle\rangle, \\
\langle\rangle
\end{pmatrix},
\begin{pmatrix}
\langle pay(1)\rangle, \\
\langle pay(1,\bot)\rangle, \\
\langle pay(1)\rangle, \\
\langle pay(1,\bot)\rangle
\end{pmatrix},
\begin{pmatrix}
\langle pay(1), & pay(2)\rangle, \\
\langle pay(1,\bot), & pay(2)\rangle, \\
\langle pay(1), & pay(2,\top)\rangle, \\
\langle pay(1,\bot), & pay(2,\top)\rangle
\end{pmatrix},
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1)\rangle, \\
\langle pay(1,\bot), & pay(2), & fl(1,\top)\rangle, \\
\langle pay(1), & pay(2,\top), & fl(1)\rangle, \\
\langle pay(1,\bot), & pay(2,\top), & fl(1)\rangle
\end{pmatrix},
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2)\rangle, \\
\langle pay(1,\bot), & pay(2), & fl(1,\top), & fl(2)\rangle, \\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot)\rangle, \\
\langle pay(1,\bot), & pay(2,\top), & fl(1), & fl(2)\rangle
\end{pmatrix},
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1)\rangle, \\
\langle pay(1,\bot), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top)\rangle, \\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top)\rangle, \\
\langle pay(1,\bot), & pay(2,\top), & fl(1), & fl(2), & sh(1)\rangle
\end{pmatrix},
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1), & sh(2)\rangle, \\
\langle pay(1,\bot), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top), & sh(2,\bot)\rangle, \\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top), & sh(2,\bot)\rangle, \\
\langle pay(1,\bot), & pay(2,\top), & fl(1), & fl(2), & sh(1), & sh(2)\rangle
\end{pmatrix},
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top)\rangle, \\
\langle pay(1,\bot), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top), & sh(2,\bot), & bc(1,\top)\rangle, \\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top), & sh(2,\bot), & bc(1,\top)\rangle, \\
\langle pay(1,\bot), & pay(2,\top), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top)\rangle
\end{pmatrix},
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot)\rangle, \\
\langle pay(1,\bot), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot)\rangle, \\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot)\rangle, \\
\langle pay(1,\bot), & pay(2,\top), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot)\rangle
\end{pmatrix},
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot), & paid(1,\top)\rangle, \\
\langle pay(1,\bot), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot), & paid(1,\top)\rangle, \\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot), & paid(1,\top)\rangle, \\
\langle pay(1,\bot), & pay(2,\top), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot), & paid(1,\top)\rangle
\end{pmatrix},
\begin{pmatrix}
\langle pay(1), & pay(2), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot), & paid(1,\top), & paid(2,\top)\rangle \\
\langle pay(1,\bot), & pay(2), & fl(1,\top), & fl(2), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot), & paid(1,\top), & paid(2,\top)\rangle \\
\langle pay(1), & pay(2,\top), & fl(1), & fl(2,\bot), & sh(1,\top), & sh(2,\bot), & bc(1,\top), & bc(2,\bot), & paid(1,\top), & paid(2,\top)\rangle \\
\langle pay(1,\bot), & pay(2,\top), & fl(1), & fl(2), & sh(1), & sh(2), & bc(1,\top), & bc(2,\bot), & paid(1,\top), & paid(2,\top)\rangle
\end{pmatrix}
$$

Consider the first and the second run presented above and consider the local state corresponding to the view of the observer (viz. the first element in the global state); for convenience, we quote the local state of the observer in the final global state of both runs below:

$$
\begin{array}{l}
\langle pay(1),\ pay(2),\ fl(1),\ fl(2),\ sh(1),\ sh(2),\ bc(1,\top),\ bc(2,\bot),\ paid(1,\top),\ paid(2,\top)\rangle \\
\langle pay(1),\ pay(2),\ fl(1),\ fl(2),\ sh(1),\ sh(2),\ bc(1,\top),\ bc(2,\bot),\ paid(1,\top),\ paid(2,\top)\rangle
\end{array}
$$

As it can be seen, the local views of the observer in the two runs coincide and following Definition 2, these two runs are indistinguishable for the observer. However, the local states of the other principals differ in one or more actions (namely, $flip$ and pay). These results have also been established in the operational semantic model of the protocol and they hint at the correspondence between the two semantic models. We formalize and prove this correspondence in the remainder of this paper.

5 Some formal results

In this section, we present three types of formal results regarding our interpreted systems semantics for *CCSi*. The first type establishes a correspondence between the operational and the interpreted systems semantics of *CCSi*. The second type of results determine the expressiveness of process algebraic specifications in generating interpreted systems. Finally, we give the third type of results about the characterization of the interpreted systems generated by process algebraic specifications.

Correspondence The first result, formulated below, relates our interpreted systems semantics with the operational semantics originally defined for *CCSi*. It states that each component of a protocol in the interpreted systems semantics is a local projection on a trace obtained from the operational semantics.

Theorem 6 *For each CCSi process p, the following two statements hold:*

- $r = r(0), \ldots, r(m) \in [\![p]\!] \Rightarrow \exists_{p',\pi}(p, \langle\rangle) \rightarrow^* (p', \pi) \wedge p'_{\downarrow} \wedge \forall i \leq n, \pi \stackrel{i}{=} \lfloor r(m)_i \rfloor$,
- $\forall_{\pi}(p, \langle\rangle) \rightarrow^* (p', \pi) \wedge p'_{\downarrow} \Rightarrow \exists_{r \in [\![p]\!], m \in \mathbb{N}} r = r(0), ..., r(m) \wedge \forall i \leq n, \pi \stackrel{i}{=} \lfloor r(m)_i \rfloor$,

where $\rightarrow^*$ denotes the reflexive transitive closure of the union of transition relations $\stackrel{a}{\rightarrow}$, and $\lfloor l_i \rfloor$ is the sequence of actions in l_i lifted to form a history, by reading the actions as decorated implicitly with $(\mathcal{I}d)$, i.e. as publicly visible (cf. the convention on p. 3).

The first item states that for each run in the IS semantics of p, there is a pair (p', π) with process p' terminating, and such that for each i, how i perceives the history π is equal to his local state $r(m)_i$. The proof goes by an induction on the length of the run. The second item states that conversely, for each process p that can terminate after history π, there is a run in the IS semantics of p in which for each i, i's local state at the end of the run captures how i perceives π. The proof of the second item is by an induction on the number of the transitions leading to (p', π).

Expressiveness Before we study the expressiveness of process algebras in generating interpreted systems, we confine our attention to the set of interpreted systems of which the local states are initialized with the empty history and are updated at each step by at most one action; this is the idea behind the notion of *initialized* and *prefix-closed* interpreted systems defined below.

Definition 7 *Consider an interpreted system (R, ν) with sets L_i of local states such that $L_i \subseteq \mathcal{A}ct^*$ for each $i \in \mathcal{I}d$. A run $r = r(0), \ldots, r(m) \in R$ is* prefix closed *if for each two consecutive global states $r(k) = (l_1, \ldots, l_n)$ and $r(k+1) = (l'_1, \ldots, l'_n)$ with $k < m$, and each $i \in \mathcal{I}d$ it holds that either $l_i = l'_i$ or $l_i \frown \alpha = l'_i$ for some $\alpha \in \mathcal{A}ct$. The run r is* initialized *if $r(0) = (\langle\rangle, \ldots, \langle\rangle)$. Interpreted system (R, ν) is* initialized *and* prefix closed*, if each and every run $r \in R$ is initialized, and prefix closed.*

It trivially holds that the interpreted system semantics for our processes are initialized, and prefix-closed. The following theorems show the (lack of) expressiveness of process algebras in generating interpreted systems. The first two theorems show that all initialized and prefix-closed interpreted systems with 1 action or at most 2 agents can be specified by a process algebraic description. The third theorem shows that in the setting with more than 1 action and more than 2 agents, not all initialized and prefix closed interpreted systems can be captured by process algebraic specifications.

Theorem 8 *For an action set $\mathcal{A}ct$ with $\mid \mathcal{A}ct \mid \leq 1$ (i.e., with cardinality at most 1), for each finite initialized, and prefix-closed interpreted system (R, ν), there exists a process algebraic description p and renaming ρ such that $[\![p]\!] = R$.*

Theorem 9 *Assume that the system comprises at most 2 agents; for each finite initialized and prefix-closed interpreted system (R, ν), there exists a process algebraic description p and a renaming ρ such that $[\![p]\!] = R$.*

Theorem 10 *For an action set of cardinality at least 2 and more than 2 agents, there exist finite initialized and prefix-closed interpreted systems that cannot be generated by any process algebraic specification.*

Proof of Theorem 10. Consider the singleton protocol $\{((\langle\rangle, \langle\rangle, \langle\rangle), (\alpha, \beta, \gamma))\}$, where α, β and γ denote three distinct actions. We claim that this protocol cannot be generated by any process algebraic specification p with the given signature for ρ. Assume towards contradiction, that such a p exists; p cannot have non-ϵ summands or parallel components, since otherwise the protocol cannot be singleton. Hence, p should be an action prefixing followed by ϵ, i.e., is of the form $d; \epsilon$. It follows from Definition 5 that $[\![p]\!] = (\langle\rangle, \langle\rangle, \langle\rangle)^\frown[\![d]\!]$. Without loss of generality assume that $d = (J)\alpha$; then since $\beta \neq \langle\rangle$ and $\gamma \neq \langle\rangle$, it should hold that $\beta = \rho(\alpha)$ and $\gamma = \rho(\alpha)$, which contradicts the assumption that β and γ are distinct. ■

Theorem 10 points out a gap in the expressiveness of our process algebraic specification language. In the proof of the theorem, this shortcoming is traced back to the restrictive nature of our global renaming function: it presumes a dichotomy of actions and their public appearances while interpreted systems allow for several (more than 2) different appearances of actions. This expressiveness gap can be filled in various ways, e.g., by adding the principal identities as a parameter to the signature of the renaming function, thereby allowing for different appearance for different principals. This will be an important next step towards making our framework more general and increasing its expressiveness, especially for application in communication protocols.

Towards characterization The properties of the epistemic relations in the IS semantics for the *CCSi* process algebra derive from the signature and the properties of the renaming function. For example, if $\rho(a) = a$ for all a, then the

equivalence classes of the $\overset{i}{\approx}$ are trivial (all singletons). The one action with a special interpretation, the silent action τ, plays a special role. For this paper, we have excluded τ as member of $\mathcal{A}ct$, but allowed it to be in the range of ρ. If $\rho(a) = \tau$ for some $a \neq \tau$, the renaming function enables modeling that some agents do not notice anything happening, when a actually happens.

If we would have allowed τ to be in $\mathcal{A}ct$ (and thereby in the domain of ρ), for a sensible interpretation of the intuition behind the renaming function, we should probably fix $\rho(\tau) = \tau$, although one could model the "illusion" that something happens when it actually doesn't, by allowing $\rho(\tau) = a$ for some $a \neq \tau$.

The three characterizing properties from [4] can be translated into our formalism as follows:

Synchronicity: if $r, r' \in [\![p]\!]$ and $r(m) \overset{i}{\approx} r'(m')$, then $m = m'$.

This property relates directly to a simple characteristic of the renaming function: we have synchronicity for process p with renaming ρ iff $[\rho(a) \neq \tau$ for all $a \neq \tau$ for which $(\mathtt{J})a \in p$ with $\mathtt{J} \neq \mathcal{I}d]$. If $(\mathtt{J})\tau$ were allowed to occur as action with $\mathtt{J} \neq \mathcal{I}d$, then the extra clause $\rho(\tau) = \tau$ needs to hold.

Perfect Recall: For all $r(m)^\frown[d], r'(m')^\frown[d'] \in [\![p]\!]$: $r(m)^\frown[d] \overset{i}{\approx} r'(m')^\frown[d']$ implies $r(m) \overset{i}{\approx} r'(m')$.

This property was proven to hold for the process algebra with identities in [9]. It holds for synchronous combinations of processes and renaming functions, as shown in the proof sketch of Appendix C.

Uniform No Miracles: if $r(m) \overset{i}{\approx} r'(m')$ and there are $r''(m''), r'''(m''')$ and d, d' such that $r''(m'')^\frown[d] \overset{i}{\approx} r'''(m''')^\frown[d']$, then $r(m)^\frown[d] \overset{i}{\approx} r'(m')^\frown[d']$.

The investigation of the conditions under which this property holds for the ISs generated from processes and renaming functions is left for future work.

Discussion: finite vs infinite runs As we indicated before defining Interpreted Systems for our processes in Definition 1, an important difference with the standard account is the fact that we take runs to be finite sequences of global states (corresponding to the finite behavior of our processes) rather than infinite sequences. For future work, we consider generalizing our process language by including recursion, which would incorporate infinitely running processes. If we then generate the corresponding ISs in the way we do in this paper, these would contain both infinite and finite runs. This is still deviating from the standard notion of IS.

In order to turn finite runs (corresponding to finite behavior) into infinite runs could be to add an infinitely stuttering tail: after termination (or the inability to proceed) at time M, generate an infinite tail with $r(m) = r(m')$ for all $m, m' > M$. However, in our current framework, this would make us lose the property of *synchronicity*: for terminating processes then $r(m) \overset{i}{=} r(m')$ for all $m \neq m' > M$. In [15] problems of synchronization in distributed systems are solved by assuming hardware clocks within the processes. For us, implementing such assumption

however, would have to be done beyond our process language (remember we take the agents, or principals, in $\mathcal{I}d$ as different entities than the processes).

6 Conclusions and future work

In this paper, we defined an interpreted systems semantics for a CCS-based process algebra. The defined semantics can be adopted for any other process-algebraic formalism as long as the visibility range and the public appearance of each atomic action in the process algebra is defined (either by using a richer syntax for atomic actions, or by providing this information as an addendum to the process-algebraic specification). We formally compared the interpreted systems semantics with the original operational semantics of the process algebra and provided a few semantic properties of the generated models by imposing restrictions on the public appearance function and the syntax of the process description.

There are two immediate next steps. The first is to include infinite behavior into the process language, but also to adapt the construction of ISs to generate more standard ISs, i.e. consisting of infinite runs. The second is to develop an epistemic temporal logical language to reason about the processes, by determining which set of propositions will be natural given the process language. Here it is relevant to keep in mind the potential application area of security protocols and the kind of properties relevant there.

We intend to extend this research and study the characteristics of interpreted systems models generated by different process algebras. Furthermore, we would like to mechanize our semantics in a tool in order to be able to verify epistemic properties of process-algebraic descriptions using the tools developed for interpreted systems.

Acknowledgements We are grateful to the anonymous reviewers for their useful comments, suggestions, and pointers for future work.

References

1. L. Aceto, A. Birgisson, A. Ingolfsdottir, and M.R. Mousavi. Decompositional Reasoning about the History of Parallel Processes. *Proceedings of FSEN'11*, volume 7141 of *LNCS*, Springer-Verlag, 2011.
2. M. Bhargava and C. Palamidessi. Probabilistic anonymity. *Proceedings of CONCUR'05*, volume 3653 of *LNCS*, Springer-Verlag, 2011.
3. J.C.M. Baeten, T. Basten, and M.A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press, 2009.
4. J. van Benthem, J. Gerbrandy, T. Hoshi, and E. Pacuit. Merging frameworks for interaction. *Journal of Philosophical Logic*, 38(5):491–526, 2009.
5. I. Boureanu, M. Cohen, and A. Lomuscio. A Compilation Method for the Verification of Temporal-Epistemic Properties of Cryptographic Protocols. *Journal of Applied Non-Classical Logics*, 19)4):463–487, 2009.

6. R. Chadha, S. Delaune, and S. Kremer. Epistemic logic for the applied pi calculus. In *Proceedings of FMOODS - FORTE '09*, volume 5522 of *LNCS*, pages 182–197. Springer, 2009.
7. K. Chatzikokolakis, S. Knight and P. Panangaden. Epistemic Strategies and Games on Concurrent Processes In *SOFSEM'09*, volume 5404 of *LNCS*, pages 153-166, Springer, 2009.
8. D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology, 1:65-75, 1988.
9. F. Dechesne, M.R. Mousavi, and S. Orzan. Operational and epistemic approaches to protocol analysis: bridging the gap. In *Proceedings of LPAR'07*, volume 4790 of *LNCS*, pages 226–241, Springer, 2007.
10. F. Dechesne, M.R. Mousavi, and S. Orzan. Operational and epistemic approaches to protocol analysis: bridging the gap. *Technical Report CSR-07-15*, Department of Computer Science, Eindhoven University of Technology, 2007.
11. F. Dechesne and Y. Wang. To know or not to know: epistemic approaches to security protocol verification. *Synthese*, 177:51–76, 2010.
12. R. van Eijk, F. de Boer, W. van der Hoek and J.-J. Ch. Meyer. Operational Semantics for Agent Communication Languages. In *Issues in Agent Communication*, volume 1916 of LNCS, pages 80–95, Springer, 2000.
13. R. van Eijk, F. de Boer, W. van der Hoek and J.-J. Ch. Meyer. Process Algebra for Agent Communication: A General Semantic Approach. In *Communication in Multiagent Systems - Agent Communication Languages and Conversation Policies*, volume 2650 of LNCS, pages 113–128, Springer, 2003.
14. R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.
15. J.Y. Halpern and Y. Moses. *Knowledge and Common Knowledge in a distributed environment*. Proceedings of the 3rd ACM Conference on Principles of Distributed Computing, pages 50-61, ACM, 1984, Reprinted in the Journal of the Association for Computing Machinery, Vol. 37, No. 3, pages 549-587, 1990.
16. J.Y. Halpern and K.R. O'Neill. Anonymity and information hiding in multiagent systems. *Journal of Computer Security*, pages 483-514, 2005.
17. J. Halpern and M.Y. Vardi. Reasoning about knowledge and time in asynchronous systems. In *Proceedings of STOC'88*, pages 53–65, ACM Press, 1988.
18. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.
19. T. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
20. W. van der Hoek, M. van Hulst, and J.-J. Ch. Meyer. Towards an Epistemic Approach to Reasoning about Concurrent Programs. In Proceedings of the REX Workshop'92, volume 666 of LNCS, pages 261–287, Springer, 1992.
21. T. Hoshi. Merging DEL and ETL. *J. of Logic, Lang. and Inf.*, 19:413–430, 2010.
22. D. Hughes and V. Shmatikov. Information hiding, anonymity and privacy: A modular approach. *Journal of Computer Security*, 12(1):3-36, 2004.
23. M. van Hulst, and J.-J. Ch. Meyer. An epistemic proof system for parallel processes: extended abstract. In *Proceeding of TARK'94*, pages 243–254, ACM Press, 1994
24. S. Katz, and G. Taubenfeld. What Processes Know: Definitions and Proof Methods. In *Proceedings of PODC'86*, pages 249–262, ACM Press, 1986.
25. S. Kramer, C. Palamidessi, R. Segala, A. Turrini, and C. Braun. A quantitative doxastic logic for probabilistic processes and applications to information-hiding. *The Journal of Applied Non-Classical Logic*, 19/4:489–516, 2009.
26. A. Lomuscio and M. Ryan. Ideal agents sharing (some!) knowledge. In *Proceedings of ECAI'98*, pages 557–561, John Wiley and Sons, 1998.

27. A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In *Proceedings of CAV'09*, volume 5643 of LNCS, pages 682–688, Springer, 2009.
28. R. van der Meyden, and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proceedings of CSFW'04*, pages 280–291, IEEE, 2004
29. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.
30. R. Parikh and R. Ramanujam. Distributed processes and the logic of knowledge. In *Proceedings of CLP'85*, volume 193 of *LNCS*, pages 256–268. Springer, 1985.
31. D.M.R. Park. Concurrency and Automata on Infinite Sequences. IN *Proceedings of the 5th GI Conference*, volume 104 of LNCS, pages 167–183, Springer, 1981.
32. G.D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60:17–139, 2004.
33. F. Raimondi and A. Lomuscio. A tool for specification and verification of epistemic properties in interpreted systems. *Electronic Notes in Theoretical Computer Science*, 85(4), 2004.
34. S. Richards and M. Sadrzadeh. Aximo: Automated axiomatic reasoning for information update. In *Proceedings of M4M5'07*, volume 231 of *ENTCS*, pages 211–225, Elsevier, 2009.
35. S. Schneider and A. Sidiropoulos. CSP and anonymity. In *Proceedings of ESORICS'96*, volume 1146 of *LNCS*, pages 198–218, Springer, 1996.
36. B. Toninho and L. Caires. A spatial-epistemic logic for reasoning about security protocols. In *Proceedings of SecCo'10*, volume 51 of EPTCS, pages 1–15, 2010.
37. W. de Vries, F.S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. A Truly Concurrent Model for Interacting Agents. In *Proceedings of PRIMA'01*, volume 2132 of LNCS, pages 16–30, Springer, 2001.

A Proof of Correspondence Results

Before we proceed with the proof of the theorem, we state and prove some auxiliary lemmata about our SOS semantics of Figure 2 as well the interpreted systems semantics of Definition 5. These lemmata will come in handy in the proof of Theorem 6.

Lemma 11 *(Arbitrary Past) For each two CCSi processes p, p', each sequence of decorated actions π and and each decorated action d, if $(p,\pi) \stackrel{d}{\Rightarrow} (p',\pi')$, then $\pi' = \pi^\frown d$. Moreover, for each π'', $(p,\pi'') \stackrel{d}{\Rightarrow} (p',\pi''^\frown d)$.*

Proof. By induction on the proof structure for $(p,\pi) \stackrel{d}{\Rightarrow} (p',\pi')$. The base case follows immediately from deduction rule (**a**) in Figure 2. The induction step follows by considering the last deduction rule used in the proof of $(p,\pi) \stackrel{d}{\Rightarrow} (p',\pi')$ applying the induction hypothesis on the premise(s) and using the result again as the premises of the same deduction rule. ■

Lemma 12 *(Termination trace) For each CCSi process p, $p_\downarrow$ if and only if $(\langle\rangle,\ldots,\langle\rangle) \in [\![p]\!]$.*

Proof. The only-if implication follows immediately from Definition 5 of run: since $p_\downarrow$, we have that $(\langle\rangle,\ldots,\langle\rangle)$ is among the runs of $[\![p]\!]$.

For the if side, assume that $(\langle\rangle,\ldots,\langle\rangle) \in [\![p]\!]$. Then according to Definition 5 it should hold for $p_0 = p = p_m$ that $p_{m\downarrow}$. ■

Lemma 13 *For all CCSi processes p, p', p'', sequences of decorated actions π, π', π'', and each decorated action $(\mathtt{J})a$, if $(p,\pi) \stackrel{d}{\Rightarrow} (p',\pi') \Rightarrow^* (p'',\pi'')$, and $p''_\downarrow$ then there exist some $r \in [\![p]\!]$ and $r' \in [\![p']\!]$ such that $r = r(0),\ldots,r(m)$, for some $m \in \mathbb{N}$, $r' = r'(0),\ldots,r'(m-1)$, and for each $i < m$, $r(i+1) = [\![d]\!]^\frown r'(i)$.*

Proof. By induction on the number of transitions leading to $\Rightarrow^*$. For the base case, if the number of transitions $\Rightarrow^*$ is zero, then $(p,\pi) \stackrel{d}{\Rightarrow} (p',\pi')$ and $p'_\downarrow$. It follows from Definition 5 that $(\langle\rangle,\ldots,\langle\rangle) \in [\![p']\!]$ and $(\langle\rangle,\ldots,\langle\rangle) \in [\![p']\!]$ and $((\langle\rangle,\ldots,\langle\rangle),[\![d]\!]) \in [\![p']\!]$. Hence, the lemma follows.

Similarly, for the induction step, assume that $\Rightarrow^*$ is due to a step labeled d', followed by a number (zero or more) subsequent steps, i.e., $(p,\pi) \stackrel{d}{\Rightarrow} (p',\pi') \stackrel{d}{\Rightarrow} (q,\pi_0) \Rightarrow^* (p'',\pi'')$, for some q and π_0. The induction hypothesis applies to the trace originating from p' and hence, there exist some $r' \in [\![p']\!]$ such that $r' = r'(0),\ldots,r'(m)$, for some $m \in \mathbb{N}$. It follows from $r' \in [\![p']\!]$ and Definition 5 that there exists a sequence of processes $p_0,\ldots,p_m$ such that $p_0 = p'$ and for each $k < m$, it holds that $\bar{l}_{k+1} = \bar{l}_k^\frown \overline{a_k}$ and $p_k \stackrel{\overline{a_k}}{\rightsquigarrow} p_{k+1}$. Consider the sequence, $p, p', p_1,\ldots,m$. We have that $p \stackrel{[\![d]\!]}{\rightsquigarrow} p'$, due to deduction rule **(is_d)** of Figure 5. Moreover, for each $k < m$, it holds that $p_k \stackrel{\overline{a_k}}{\rightsquigarrow} p_{k+1}$. Define the run r to be of the form $r(0),\ldots,r(m+1)$ such that $r(0) = (\langle\rangle,\ldots,\langle\rangle)$, and for each

$i \leq m$, $r(i+1) = [\![d]\!]^\frown r(i)$, which implies, according to the definition of r', that $r(i+1) = [\![d]\!]^\frown(r'(i-1)^\frown\overline{a_i}) = ([\![d]\!]^\frown r'(i-1))^\frown\overline{a_i} = r(i)^\frown\overline{a_i}$. It then follows from Definition 5 that $r \in [\![p]\!]$, which was to be shown. ■

Proof of Theorem 6. The if-side is proven by an induction on the number of transitions comprising $\rightarrow^*$.

The base case, i.e., the case for immediate termination of p, follows from Lemma 12. The induction step follows from Lemma 13.

Similarly, the only-if side is proven by an induction on the length of run r. For a singleton run, i.e., a run comprising of a single global state with empty sequences as local states, the Theorem follows from Lemma 12. The induction step follows from Lemma 13. ■

B Proof of Expressiveness Results

Proof of Theorem 8. Without loss of generality, we assume that the set of actions $\mathcal{A}ct = \{\alpha\}$ (otherwise, if $\mathcal{A}ct = \emptyset$, the only possible run in the interpreted system is the one with the empty sequence of actions, which can be generated by the *CCSi* process 0).

We prove the theorem by constructing a (sequential) process p_r for each run r of a given protocol R. The process specification corresponding to the protocol is then $\sum_{r\in R} p_r$. We construct the p_r by an induction on the size of the run r and show that for each run r there exists a process p_r such that under the renaming function ρ with $\rho(\alpha) = \tau$, it holds that $[\![p_r]\!] = r$.

For a run of size 1, we have that $r = (\langle\rangle, \ldots, \langle\rangle)$. It clearly holds that $[\![\delta]\!] = r$ and hence r can be generated by the *CCSi* process δ.

Assume that each initialized prefix-closed run of size m or less can be generated by a *CCSi* process and consider an initialized prefix-closed run $r = r(0), \ldots, r(m)$, where $m \geq 1$. Then $r(0) = (\langle\rangle, \ldots, \langle\rangle)$ (because r is initialized). Write $r(1) = (l_0, \ldots, l_n)$, then for each $i \leq n$ one of the following cases holds:

- $l_i = \langle\rangle$, or
- $l_i = \langle a\rangle$, or

Define the set $\mathtt{J} \subseteq \mathcal{I}d$ as $\mathtt{J} \doteq \{i \mid l_i = \langle a\rangle\}$. We have already defined $\rho(\alpha) = \tau$ and thus, we have that $[\![(\mathtt{J})\alpha]\!] = (l_1, \ldots, l_n)$. Define r' to be the run $r'(0), r'(1), \ldots, r'(m-1)$, where $r'(0) = (\langle\rangle, \ldots, \langle\rangle)$ and for each $j \leq n$ and $i < m$, $r'(i)_j$ is obtained by removing l_j from the head of $r(i+1)_j$, if $l_j \neq \langle\rangle$, or $r(i+1)_j$, otherwise. It clearly holds that r' is initialized and r' is of size m, and hence, the induction hypothesis applies to r'. Thus, there exists a process p such that $[\![p]\!] = r'(0), \ldots, r'(m-1)$. It then follows that $[\![J(\alpha); p]\!] = r(0)$, $r(1)$, $\ldots$, $r(m)$, which was to be shown.

■

Proof of Theorem 9. Without loss of generality we assume that $\mathcal{A}ct \neq \emptyset$, where $\mathcal{A}ct$ is the set of actions appearing in the interpreted system.

Define the set $\mathcal{A}ct_P \doteq \mathcal{A}ct \cup \{\alpha_\beta, \alpha_\tau \mid \alpha, \beta \in \mathcal{A}ct\}$. Also define the function $\rho : \mathcal{A}ct_P \to \mathcal{A}ct_P \cup \{\tau\}$ such that $\rho(\alpha) \doteq \alpha$ $\rho(\alpha_\beta) \doteq b$ and $\rho(\alpha_\tau) = \tau$, for each $\alpha, \beta \mathcal{A}ct$. Similar to the proof of Theorem 8, we construct for each run r of a protocol, a process p such that $[\![p]\!] = r$, under the renaming function ρ.

For a run of size 1, we have that $r = r(0) = (\langle\rangle, \langle\rangle)$. It clearly holds that $[\![\delta]\!] = r$ and hence r can be generated by the *CCSi* process 0.

Assume that each prefix-closed run of size m or less can be generated by a *CCSi* process and consider a prefix-closed run $r = r(0), \ldots, r(m)$, where $m \geq 1$. Assume that $r(0) = (a_0, a_1)$ and $r(1) = (a_0 \frown b_0, a_1 \frown b_1)$. Define r' to be the run $r'(0), r'(1), \ldots, r'(m-1)$, where $r'(0) = (\langle\rangle, \ldots, \langle\rangle)$ and for each $j \in \{0, 1\}$ and $i < m$, $r'(i)_j$ is obtained by removing a_j from the head of $r(i+1)_j$, if $b_j \neq \langle\rangle$, or $r(i+1)_j$, otherwise. Because r' is of size m, the induction hypothesis applies and there exists a process p such that $[\![p]\!] = r'(0), \ldots, r'(m-1)$. Moreover, because r is initialized, and prefix closed, and we have that $a_0 = a_1 = \langle\rangle$ and one of the following four cases holds:

1. $a_0 = b_0$, and $a_1 = b_1$, or
2. $a_0 = b_0$, and $b_1 = \alpha$, for some $\alpha \in \mathcal{A}ct$, or
3. $b_0 = \alpha$, and $a_1 = b_1$, for some $\alpha \in \mathcal{A}ct$, or
4. $b_0 = \alpha$, and $b_1 = \beta$, for some $\alpha, \beta \in \mathcal{A}ct$.

Based on each of the above-given four cases, define d as follows:

1. $d \doteq (\emptyset)\alpha$ for some $\alpha \in \mathcal{A}ct$,
2. $d \doteq (\{0\})\alpha_\tau$,
3. $d \doteq (\{1\})\alpha_\tau$,
4. $d \doteq (\{0\})\alpha_\beta$.

Following the above-given definition of d the definition of ρ given before, we have that $[\![d]\!] = (b_0, b_1)$. Then it follows from Definition that $[\![d; p]\!] = r(0)$, $r(1)$, $\ldots$, $r(m)$, which was to be shown. ■

C Towards characterization

We discuss the three properties that were proven to be characterizing the ISs resulting from the translation of DEL into ETL in [4].

In this discussion, assume a process p in *CCSi*, and a renaming function $\rho : \mathcal{A}ct \to \mathcal{A}ct$ have been specified. Let $\mathcal{I}d = \{1, \ldots, m\}$ be the identities occurring in p. Recall that the elements of $[\![p]\!]$ are (finite) sequences r of global states, and that global states $r(n)$ in $[\![p]\!]$ (i.e. run r at 'time' n) are m-tuples $(l_1, \ldots, l_m)$ of local states. Local states are the (finite) sequences of the actions that happened according to p as observed by agent i in a global state $r(n)$.

Synchronicity : if $r, r' \in [\![p]\!]$ and for some m, m': $r(m) \overset{i}{\approx} r'(m')$, then $m = m'$.

Lemma 14 *Synchronicity is satisfied for process p with renaming ρ unless:*

1. $(\mathtt{J})\tau$ *occurs in* p *with* $\mathtt{J} \neq \mathcal{I}d$ *and* $\rho(\tau) \neq \tau$*, or:*
2. $(\mathtt{J})\alpha$ *occurs in* p *with* $\mathtt{J} \neq \mathcal{I}d$ *and* $\rho(\alpha) = \tau$

Proof sketch. Synchronicity states that agents cannot confuse the number of actions that have happened according to p so far. There are only two ways by which such confusion would occur, so we have synchronicity unless:

1. if agents could think that something actually happened when nothing did: $(\mathtt{J})\tau$ occurs in p with $\mathtt{J} \neq \mathcal{I}d$ and $\rho(\tau) = \alpha$ for some $\alpha \neq \tau$. (In the *CCSi*-version we presented in this paper, τ does not occur as action.)
2. Conversely, if agents can think that nothing happened when something actually did: $(\mathtt{J})\alpha$ occurs in p with $\mathtt{J} \neq \mathcal{I}d$ and $\rho(\alpha) = \tau$.

Excluding the two situations above ensures synchronicity: they will not occur if and only if $\rho(\tau) = \tau$, and $\rho(\alpha) \neq \tau$ for all $\alpha \neq \tau$ for which $(\mathtt{J})\alpha \in p$ with $\mathtt{J} \neq \mathcal{I}d$ ■

Perfect Recall : For all $r(m)^\frown[d], r'(m')^\frown[d'] \in [\![p]\!]$: $r(m)^\frown[d] \overset{i}{\approx} r'(m')^\frown[d']$ implies $r(m) \overset{i}{\approx} r'(m')$

A formulation of Perfect Recall for the process algebra with identities has been proven to hold in [9]. It turns out that the combination p, ρ has perfect recall for i iff $\rho(\alpha) \neq \tau$ for all occurring $(\mathtt{J})\alpha$ where $i \notin J$ (cf. the second clause in Lemma 14). In particular, Perfect Recall holds for synchronous combinations of processes and renamings.

Proof sketch. Assume $r(m)^\frown[(\mathtt{J})\alpha] \overset{i}{\approx} r'(m')^\frown[(\mathtt{J}')\alpha']$ in $[\![p]\!]$. Then we have four cases: (1) $i \in \mathtt{J} \cap \mathtt{J}'$, (2) $i \in \mathtt{J} \setminus \mathtt{J}'$, (3) (symmetrically) $i \in \mathtt{J}' \setminus \mathtt{J}$, or (4) $i \notin \mathtt{J} \cup \mathtt{J}'$.
Case (1): Then $r(m)^\frown[(\mathtt{J})\alpha] \overset{i}{\approx} r'(m')^\frown[(\mathtt{J}')\alpha']$ means that $r(m)_i^\frown\alpha = r'(m')_i^\frown\alpha'$. It follows that $\alpha = \alpha'$ and $r(m)_i = r'(m')_i$, hence $r(m) \overset{i}{\approx} r'(m')$.
Case (2): Then EITHER $\rho(\alpha') = \tau$ and $r(m)_i^\frown\alpha = r'(m')_i$, so $r(m)_i \neq r'(m')_i$, OR $\rho(\alpha') \neq \tau$ and $r(m)_i^\frown\alpha = r'(m')_i^\frown\rho(\alpha')$, so $\alpha = \rho(\alpha')$ and $r(m)_i = r'(m')_i$, hence $r(m) \overset{i}{\approx} r'(m')$.
Case (3) is symmetrical to case (2).
Case (4): Here again, we have 4 cases $\rho(\alpha) = \tau = \rho(\alpha')$, $\rho(\alpha) \neq \tau \neq \rho(\alpha')$, $\rho(\alpha) \neq \tau = \rho(\alpha')$ and $\rho(\alpha) = \rho \neq \rho(\alpha')$. It is not hard to see that from the first two cases it follows that $r(m)_i = r'(m')_i$, and from the latter two: $r(m)_i \neq r'(m')_i$.

So, the combination p, ρ is has perfect recall for i iff $\rho(\alpha) \neq \tau$ for all occurring $(\mathtt{J})\alpha$ where $i \notin J$. ■

Uniform No Miracles : if in $[\![p]\!]$, $r(m) \overset{i}{\approx} r'(m')$ and there are $r''(m''), r'''(m''')$ and d, d' such that $r''(m'')^\frown[d] \overset{i}{\approx} r'''(m''')^\frown[d']$, then $r(m)^\frown[d] \overset{i}{\approx} r'(m')^\frown[d]$.

This property is somewhat hard to read. It expresses that if two actions somewhere lead to indistinguishabie states, the agent will never be able to distinguish the states resulting from these actions happening *from* indistinguishable states.

We leave the analysis of this property for future work.