

Modularity and Openness in Modeling Multi-Agent Systems

Wojciech Jamroga

Computer Science and Communication, and
Interdisciplinary Centre on Security, Reliability and Trust
University of Luxembourg

wojtek.jamroga@uni.lu

Artur Męski

Institute of Computer Science
Polish Academy of Sciences, Warsaw, Poland,
and FMCS, University of Łódź, Poland

meski@ipipan.waw.pl

Maciej Szreter

Institute of Computer Science
Polish Academy of Sciences, Warsaw, Poland

mszreter@ipipan.waw.pl

We revisit the formalism of modular interpreted systems (MIS) which encourages modular and open modeling of synchronous multi-agent systems. The original formulation of MIS did not live entirely up to its promise. In this paper, we propose how to improve modularity and openness of MIS by changing the structure of interference functions. These relatively small changes allow for surprisingly high flexibility when modeling actual multi-agent systems. We demonstrate this on two well-known examples, namely the trains, tunnel and controller, and the dining cryptographers.

Perhaps more importantly, we propose how the notions of multi-agency and openness, crucial for multi-agent systems, can be precisely defined based on their MIS representations.

1 Introduction

The paradigm of multi-agent systems (MAS) focuses on systems consisting of autonomous entities acting in a common environment. Regardless of whether we deem the entities to be intelligent or not, proactive or reactive, etc., there are two design-level properties that a multi-agent system should satisfy. First, it should be *modular* in the sense that it is inhabited by *loosely coupled* components. That is, interaction between agents is crucial for the system, but it should be relatively scarce compared to the intensity of local computation within agents (otherwise the system is in fact a single-agent system in disguise). Secondly, it should be *open* in the sense that an agent should be able to join or leave the system without changing the design of the other components.

Models and representations of MAS can be roughly divided into two classes. On one hand, there are models of various agent logics, most notably modal logics of knowledge, action, time, and strategic ability [7, 8, 2]. These models are well suited for theoretical analysis of general properties of agent systems. However, they are too abstract in the sense that: (a) they are based on abstract notions of global state and global transition so the structure of a model does not reflect the structure of a MAS at all, and (b) they come with neither explicit nor implicit methodology for design and analysis of actual agent systems. At the other extreme there are practical-purpose high-level representation languages like Promela [11], Estelle [6], and Reactive Modules (RM) [1]. They are application-oriented, and usually include too many features to be convenient for theoretical analysis. The middle ground consists of formalisms that originate from abstract logical models but try to encapsulate a particular modeling methodology. For instance, interpreted systems [8] support local design of the state space; however, transitions are still global, i.e., they are defined between global rather than local states. Synchronous automata networks [10] and ISPL specifications [17, 19] push the idea further: they are based on local states and semi-local transitions,

i.e., the outcome of a transition is local, but its domain global. This makes agents hard to separate from one another in a model, which hampers its modularity. On the other hand, concurrent programs [16] and asynchronous automata networks [10] are fairly modular but they support only systems whose execution can be appropriately modeled by interleaving of local actions and/or events.

Modular Interpreted Systems (MIS) are a class of models proposed in [13] to achieve separation of the interference between agents from the local processing within agents. The main idea behind MIS was to encapsulate the way agents' actions interfere by so called *interaction tokens* from a given alphabet $\mathcal{I}n$, together with functions out_i, in_i that define the *interface* of agent i . That is, out_i specifies how i 's actions influence the evolution of the other agents, whereas in_i specifies how what rest of the world influences the local transition of i . Modular interpreted systems received relatively little attention, though some work was done on studying computational properties of the related verification problem [12], facilitating verification by abstraction [14], and using MIS to analyze homogeneous multi-agent systems [4]. This possibly stems from the fact that, in their original incarnation, MIS are not as modular and open as one would expect. More precisely, the types of functions used to define interference fix the number of agents in the MIS. Moreover, the assumption that all the functions used in a model are deterministic limit the practical applicability, as modeling of many natural scenarios becomes cumbersome.

In this paper, we try to revive MIS as an interesting formalism for modeling multi-agent systems. We propose how to improve modularity and openness of the original class by changing the structure of interference functions out, in . The idea is to use multisets of interference tokens instead of k -tuples. This way, we do not need to “hardwire” information about other modules inside a module. Additionally, we assume that the “manifestation” function out can be nondeterministic. These relatively small changes allow for surprisingly high flexibility when modeling MAS. We demonstrate that on two well-known benchmark examples: trains, tunnel and controller, and the dining cryptographers.

Perhaps more importantly, we propose how two important features of multi-agent systems can be formally defined, based on MIS representations. First, we show how to decide if a system is designed in a proper multi-agent way by looking at the relation between the complexity of its interference layer to the complexity of its global unfolding. Moreover, we define the degree of openness of a MIS as the complexity of the minimal transformation that the model must undergo in order to add a new agent to the system, or remove an existing one. We apply the definitions to our benchmark models, and show that different variants of cryptographers grossly differ in the amount of openness that they offer.

The paper has the following structure. In Section 2, the new variant of MIS is defined, along with its execution semantics. Section 3 presents MIS representations for two benchmarks: Tunnel, Trains and Controller (TTC) and Dining Cryptographers (DC). A graphical notation is provided to make the examples easier to read. In Sections 4 and 5, we propose formal definitions of multi-agency and openness, respectively, and apply them to several variants of the benchmarks. Section 6 concludes the paper.

1.1 Related Work

The modeling structures discussed in this paper share many similarities with existing modeling frameworks, in particular with Reactive Modules [1]. Still, MIS and RMs have different perspectives: Reactive Modules is an application-oriented language, while the focus of modular interpreted systems is more theoretic. This results in a higher abstraction level of MIS which are based on abstract states and interaction tokens. MIS aim at separating internal activities of modules and interactions between modules, what is not (explicitly) featured in RM.

Modularity in models and model checking has been the focus of many papers. Most notably, Hierarchical State Machines of Alur et al. [3, 20] and the approach of hierarchical module checking by

Murano et al. [18] feature both “horizontal” and “vertical” modularity, i.e., a system can be constructed by means of parallel composition as well as nesting of modules. Similarly, dynamic modifications and “true openness” of models has been advocated in [9]. In that paper, Dynamic Reactive Modules (DRM) were proposed, which allow for dynamic reconfiguration of the structure of the system (including adding and removing modules). Our approach differs from the ones cited above in two ways. On one hand, we focus on an abstract formulation of the *separation of concerns* between modules (and agents), rather than providing concrete mechanisms that implement the separation. On the other, we define indicators that show *how good the resulting models is*. That is, our measures of agentivity and openness are meant to assess the model “from the outside”. In particular, the focus of the DRM is on providing a mechanism for adding and removing agents in the RM representation. We implement these operations on the meta-level, as a basis of the mathematical measure of openness. Our work could in principle be applied to DRMs and other formalisms, but it would require defining the appropriate multi-agent mechanisms which are already present in Interpreted Systems.

2 Modular Interpreted Systems Revisited

Modular interpreted systems were proposed in [13] to encourage modular and open design of synchronous agent systems. Below, we present an update on the formalism. The new version of MIS differs from the original one [13] as follows. First, a single agent can be now modeled by more than one module to allow for compact design of agents’ local state spaces and transition functions. Secondly, the type of function in_i is now independent from the structure and cardinality of the set of agents, thus removing the main obstacle to modularity and openness of representation in the previous version. Thirdly, the interaction functions in_i, out_i are nondeterministic in order to enable nondeterministic choice and randomization (needed, e.g., to obtain fair scheduling or secure exchange of information). Fourthly, we separate agents from their names. This way, agents that are not present in the “current” MIS can be referenced in order to facilitate possible future expansion of the MIS.

2.1 New Definition of MIS

Let a bag (multiset) over set X be any function $X \rightarrow \mathbb{N}$. The set of all bags over X will be denoted by $\mathcal{B}(X)$, and the union of bags by \uplus .

Definition 1 (Modular interpreted system) We define a modular interpreted system (MIS) as a tuple

$$S = (\text{Agtnames}, \text{Act}, \mathcal{I}n, \mathbb{A}gt),$$

where Agtnames is a finite set of agent names, Act is a finite set of action names, $\mathcal{I}n$ is a finite interaction alphabet, and $\mathbb{A}gt = \{a_1, \dots, a_k\}$ is a finite set of agents (whose structure is defined in the following paragraph). A set of directed tokens, used to specify the recipients of interactions, is defined as $\mathcal{T}ok = \mathcal{I}n \times (\text{Agtnames} \cup \{\varepsilon\})$, where ε denotes that the interaction needs to be broadcasted to all the agents in the system.

Each agent $a_j = (id, \{m_1, \dots, m_n\})$ consists of a unique name $id \in \text{Agtnames}$ (also denoted with $\text{name}(a_j)$), and one or more modules $m_j = (St_j, \text{Init}_j, d_j, out_j, in_j, o_j, \Pi_j, \pi_j)$, where:

- St_j is a set of local states,
- $\text{Init}_j \subseteq St_j$ is the set of initial states,
- $d_j : St_j \rightarrow \mathcal{P}(\text{Act})$ defines local availability of actions; for convenience of the notation, we additionally define the set of situated actions as $D_j = \{(q_j, \alpha) \mid q_j \in St_j, \alpha \in d_j(q_j)\}$,

- out_j, in_j are interaction (or interference) functions:
 - $out_j : D_j \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{T}ok))$ refers to the set of influences (chosen nondeterministically) that a given situated action (of module m_j) may possibly have on the recipients of the embedded interaction symbols, and
 - $in_j : St_j \times \mathcal{B}(\mathcal{I}n) \rightarrow \mathcal{P}(\mathcal{I}n)$ translates external manifestations from the other modules into the (nondeterministically chosen) “impression” that they make on module m_j depending on the local state of m_j ; we assume $in_j(\cdot) \neq \emptyset$;
- $o_j : D_j \times \mathcal{I}n \rightarrow \mathcal{P}(St_j)$ is a local transition function (possibly nondeterministic),
- Π_j is a set of local propositions of module m_j (we require that Π_j and Π_m are disjoint when $j \neq m$),
- $\pi_j : \Pi_j \rightarrow \mathcal{P}(St_j)$ is a valuation of these propositions.

Additionally, we define the cardinality of S (denoted $card(S)$) as the number of agents in S .

Typically, each agent in a MIS consists of exactly one module, and we will use the terms interchangeably. Also, we will omit $Init_j$ from the description of a module whenever $Init_j = St_j$.

Note that function in_j is in general infinite. For practical purposes, finite representation of in_j is needed. We use decision lists similarly to [15, 19]. Thus, in_i will be described as an ordered list of pairs of the form *condition* \mapsto *value*. The first pair on the list with a matching condition decides on the value of the function. The conditions are boolean combinations of membership and cardinality tests, and are defined over the variable s for the conditions defined on states, and over H for the conditions on multisets of received interferences. We require that the last condition on the list is \top , so that the function is total. Several examples of MIS’s are presented in Sections 3 and 5.

2.2 Execution Semantics for MIS

Definition 2 (Explicit models) A nondeterministic concurrent epistemic game structure (NCEGS) is a tuple $C = (\mathcal{A}, St, St_0, PV, \mathcal{V}, Act, \mathfrak{d}, t, \sim_1, \dots, \sim_k)$, where: $\mathcal{A} = \{1, \dots, k\}$ is a nonempty set of agents, St is a nonempty set of states, $St_0 \subseteq St$ is the set of initial states, PV is a set of atomic propositions, $\mathcal{V} : PV \rightarrow \mathcal{P}(St)$ is a valuation function, $\mathfrak{d} : \mathcal{A} \times St \rightarrow \mathcal{P}(Act)$ assigns nonempty sets of actions available at each state, and t is a (nondeterministic) transition function that assigns a nonempty set $Q = t(q, \alpha_1, \dots, \alpha_k)$ of outcome states to a state q , and a tuple of actions $(\alpha_1, \dots, \alpha_k)$ that can be executed in q .

We define the semantics of MIS through an unfolding to NCEGS.

Definition 3 (Unfolding of MIS) Unfolding of the modular interpreted system S from Definition 1 to a nondeterministic concurrent epistemic game structure $NCEGS(S) = (\mathcal{A}', St', St'_0, PV', \mathcal{V}', Act', \mathfrak{d}', t')$ is defined as follows:

- $\mathcal{A}' = \{1, \dots, k\}$, and $Act' = Act$,
- $St' = \prod_{i=1}^k St_i$,
- $St'_0 = \{(q_1, \dots, q_k) \mid (\forall i \in \{1, \dots, k\}) q_i \in Init_i\}$,
- $PV' = \bigcup_{i=1}^k \Pi_i$, and $\mathcal{V}'(p) = \pi_i(p)$ when $p \in \Pi_i$,
- $\mathfrak{d}'(i, q) = d_i(q_i)$ for global state $q = (q_1, \dots, q_k)$, and $i \in \mathcal{A}'$,
- The transition function t' is constructed as follows. Let $q = (q_1, \dots, q_k)$ be a state, and $\alpha = (\alpha_1, \dots, \alpha_k)$ be a joint action. We define an auxiliary function $oi_i(q_i, \alpha_i)$ of all the possible interferences of agent i , for q_i , and α_i : $\gamma' \in oi_i(q_i, \alpha_i)$ iff there exist T_1, \dots, T_k such that $T_j \in out_j((q_j, \alpha_j))$, and $\gamma' \in in_i(q_i, \mathcal{S}_1 \uplus \dots \uplus \mathcal{S}_k)$, where $\mathcal{S}_j = \{\gamma_j \mid (\exists r \in \{name(a_j), \varepsilon\}) (\gamma_j, r) \in T_j\}$ for all $j \in \mathcal{A}'$. Then $(q'_1, \dots, q'_k) \in t'(q, \alpha_1, \dots, \alpha_k)$ iff $q'_i \in oi_i((q_i, \alpha_i), \gamma)$, where $\gamma \in oi_i(q_i, \alpha_i)$;

- $q \sim_i q'$ iff q and q' agree on the local states of all the modules in agent a_i .

Definition 3 immediately provides some important logics (such as CTL, LTL, ATL, epistemic logic, and their combinations) with semantics over modular interpreted systems. By the same token, the model checking and satisfiability problems for those logics are well defined in MIS.

3 Modeling with MIS

We argue that the revised definition of MIS achieves a high level of separation between components in a model. The interaction between an agent and the rest of the world is encapsulated in the agent's interference functions out_i, in_i . Of course, the design of the agent must take into account the tokens that can be sent from modules with which the agent is supposed to interact. For instance, the out, in functions of two communicating agents must be prepared to receive communication tokens from the other party. However, the interference functions can be oblivious to the modules with which the agent does not interact. In this section, we demonstrate the advantages on two benchmark scenarios: Trains, Tunnel, and Controller (TTC), and Dining Cryptographers (DC).

3.1 Tunnel, Trains, and Controller (TTC)

TTC is a variant of classical mutual exclusion, and models n trains moving over cyclic tracks sharing a single tunnel. Because only one train can be in the tunnel at a time, trains need to get a permission from the controller before entering the tunnel. We model the scenario by MIS $TTC_n = (Agt, Act, In)$, where:

- $Agtnames = \{tr_1, \dots, tr_n, ctrl\}$,
- $Act = \{nop, approach, request, enter, leave\}$,
- $\mathcal{S}_n = \{idle, appr, try_1, \dots, try_n, retry, granted, left, enter, aw_reqs, grant, grant_1, \dots, grant_n, no_reqs, infd, ack_release, aw_leave\}$.
- $\mathbb{A}gt = \{\mathbf{tr}_1, \dots, \mathbf{tr}_n, \mathbf{ctrl}\}$,

The system includes n trains $\mathbf{tr}_i = (tr_i, \{(St_i, Init_i, d_i, out_i, in_i, o_i, \Pi_i, \pi_i)\})$ for $i \in \{0, \dots, n\}$ such that:

$St_i = \{out, tun_needed, granted, in\}$, and $Init_i = \{out\}$. d_i is defined as:

- $out \mapsto \{nop, approach\}$,
- $tun_needed \mapsto \{request\}$,
- $granted \mapsto \{enter\}$,
- $in \mapsto \{nop, leave\}$

o_i is defined as:

- $((out, nop), idle) \mapsto \{out\}$
- $((out, approach), appr) \mapsto \{tun_needed\}$,
- $((tun_needed, request), retry) \mapsto \{tun_needed\}$,
- $((tun_needed, request), granted) \mapsto \{granted\}$,
- $((granted, enter), enter) \mapsto \{in\}$,
- $((in, nop), idle) \mapsto \{in\}$,
- $((in, leave), leave) \mapsto \{out\}$

$\Pi_i = \{in_tunnel\}$

- $(out, nop) \mapsto \{\{(idle, tr_i)\}\}$,
- $(out, approach) \mapsto \{\{(appr, tr_i)\}\}$,
- $(tun_needed, request) \mapsto \{\{(try_i, ctrl)\}\}$,
- $(granted, enter) \mapsto \{\{(enter, tr_i)\}\}$,
- $(in, nop) \mapsto \{\{(idle, tr_i)\}\}$,
- $(in, leave) \mapsto \{\{(left, ctrl), (left, tr_i)\}\}$

in_i is defined as:

- $s = out \wedge appr \in H \mapsto \{appr\}$,
- $s = tun_needed \wedge grant \in H \mapsto \{granted\}$,
- $s = tun_needed \mapsto \{retry\}$,
- $s = granted \wedge enter \in H \mapsto \{granted\}$,
- $s = in \wedge left \in H \mapsto \{left\}$,
- $\top \mapsto \{idle\}$

$\pi_i = \{in \mapsto in_tunnel\}$

Moreover, the agent **ctrl** = $(ctrl, \{(St_c, Init_c, d_c, out_c, in_c, o_c, \Pi_c, \pi_c)\})$ modeling the controller is defined as follows:

$St_c = \{tun_free, infd, tr_1granted, \dots, tr_ngranted\}$, and
 $Init_c = \{tun_free\}$.

d_c is defined as:

- $tun_free \mapsto \{accepting\}$,
- $infd \mapsto \{waiting\}$,
- $tr_1granted \mapsto \{inform\}$,
- ...
- $tr_ngranted \mapsto \{inform\}$

o_c is defined as:

- $((tun_free, accepting), no_reqs) \mapsto \{tun_free\}$,
- $((infd, waiting), aw_leave) \mapsto \{infd\}$,
- $((infd, waiting), ack_release) \mapsto \{tun_free\}$,
- $((tun_free, accepting), grant_1) \mapsto \{tr_1granted\}$,
- ...
- $((tun_free, accepting), grant_n) \mapsto \{tr_ngranted\}$,
- $((tr_1granted, inform), infd) \mapsto \{infd\}$,
- ...
- $((tr_ngranted, inform), infd) \mapsto \{infd\}$

out_c is defined as:

- $(tun_free, accepting) \mapsto \{\{(aw_reqs, \epsilon)\}\}$,
- $(infd, waiting) \mapsto \{\{(aw_leave, ctrl)\}\}$,
- $(tr_1granted, inform) \mapsto \{\{(grant, tr_1)\}\}$,
- ...
- $(tr_ngranted, inform) \mapsto \{\{(grant, tr_n)\}\}$,

in_c is defined as:

- $s = tun_free \wedge try_1 \in H \mapsto \{grant_1\}$,
- ...
- $s = tun_free \wedge try_n \in H \mapsto \{grant_n\}$,
- $s = tun_free \mapsto \{no_reqs\}$,
- $s = tr_1granted \vee \dots \vee s = tr_ngranted \mapsto \{infd\}$,
- $s = infd \wedge left \in H \mapsto \{ack_release\}$,
- $s = infd \mapsto \{aw_leave\}$,
- $\top \mapsto \{idle\}$

$\Pi_c = \{tunnel_busy\}$

$\pi_c = \{infd \mapsto tunnel_busy\}$

The model is illustrated in Figure 1 using the notation introduced in Section 3.2. The protocol focuses on the procedure of gaining a permission to access the tunnel. Before requesting the permission, a train approaches the tunnel, and its state changes to *tun_needed*. In this state it requests the permission from the controller. When the controller grants the permission to one of the nondeterministically chosen trains (*tr_igranted*) it informs the train that got access to the tunnel about this fact, and moves to the state *infd*. The train enters the tunnel in the next step of the protocol, and changes its state to *in*, whereas the remaining trains may continue requesting the access (they remain in *tun_needed*). When the train leaves the tunnel, it changes its state to *tun_free*.

3.2 Graphical Representation

As the definitions of MIS tend to be verbose, we introduce a simple graphical notation, based on networks of communicating automata. Let us explain it, based on Figure 1, which is a graphical representation of the tunnel, trains, and controller model from Section 3.1:

- Modules defining different **agents** and belonging to the same agent are separated by solid and dashed lines, respectively,
- Circles correspond to **local states**. An arrow with loose end pointing into a circle denotes an initial state,
- Boxes define **local actions** associated with a state,
- For a local action, dashed lines going out of it define **emitted influences**, specified with the receiver and the influence at the left and right side of an harpoon arrow pointed left, respectively. When no receiver is specified, the influence is broadcasted,

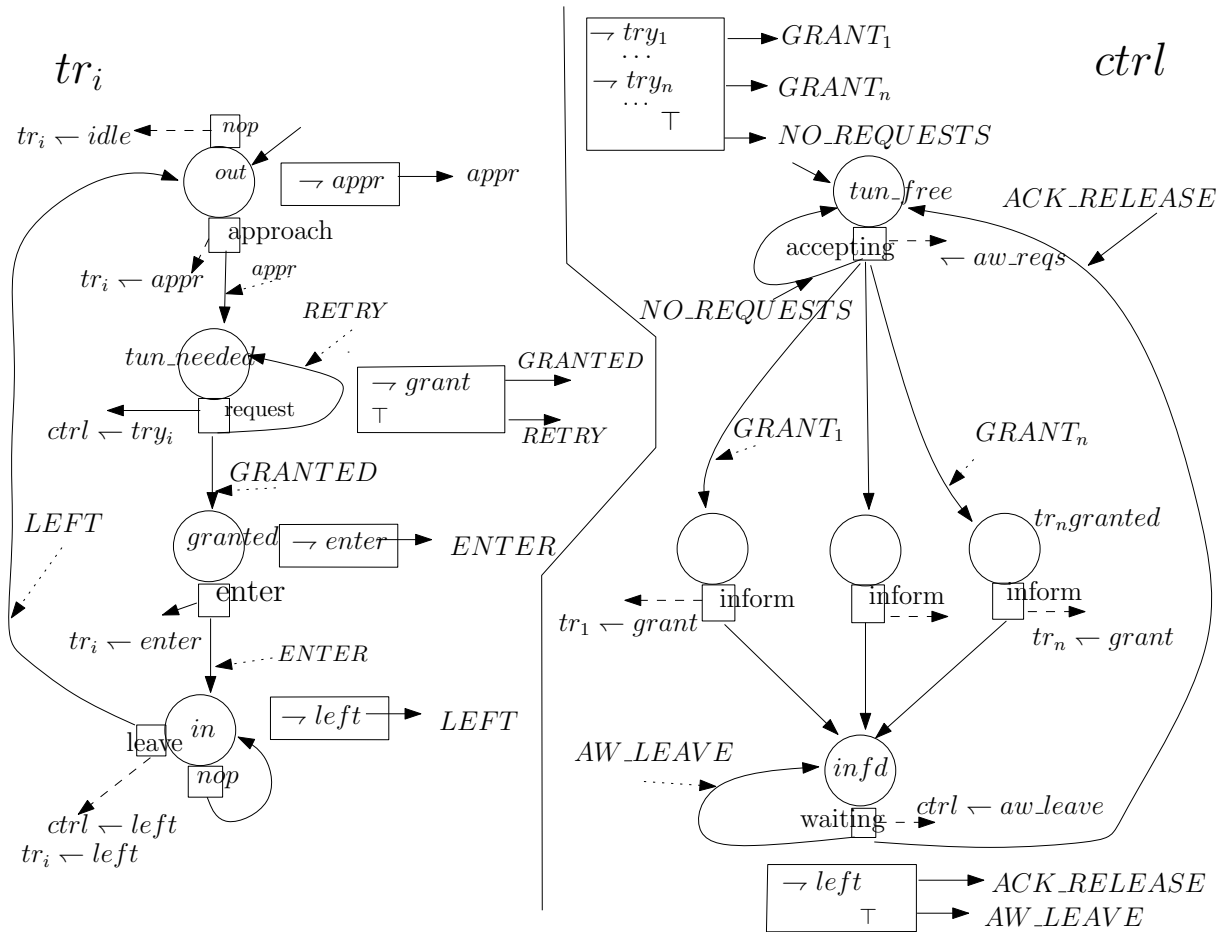


Figure 1: Tunnel, trains and controller (TTC) in the graphical representation

- Solid lines with arrows, connecting an action with a local state, correspond to a **local transition function**,
- For a local state, guarded commands (possibly in a box) define the translation of **external manifestations** received by an agent into **local impressions**. A harpoon arrow pointed right corresponds to a sender at the left side and the message at the right side, and if the sender is not specified it means receiving from anyone. For a transition, dotted arrows pointing at it correspond to application of those impressions.

The number of interactions x received by an agent is denoted with $n(x)$. The notation $*$ labeling a transition means that it is executed when none of the remaining transitions are enabled. For example, it could be used instead of directly specifying the generation and application of *aw_leave* manifestation in the controller.

Some parts can be skipped or abstracted away if it does not lead to confusion. For example, if no influence is emitted and only one transition is associated with an action, this action needs not be directly specified. In Figure 1, the self-loop from the *in* state is not accompanied by the associated local impression nor the impression. Similarly, a single influence addressed to the very module that issued it can be omitted. For example, we do not show the manifestation *idle* in the graph. Valuations of

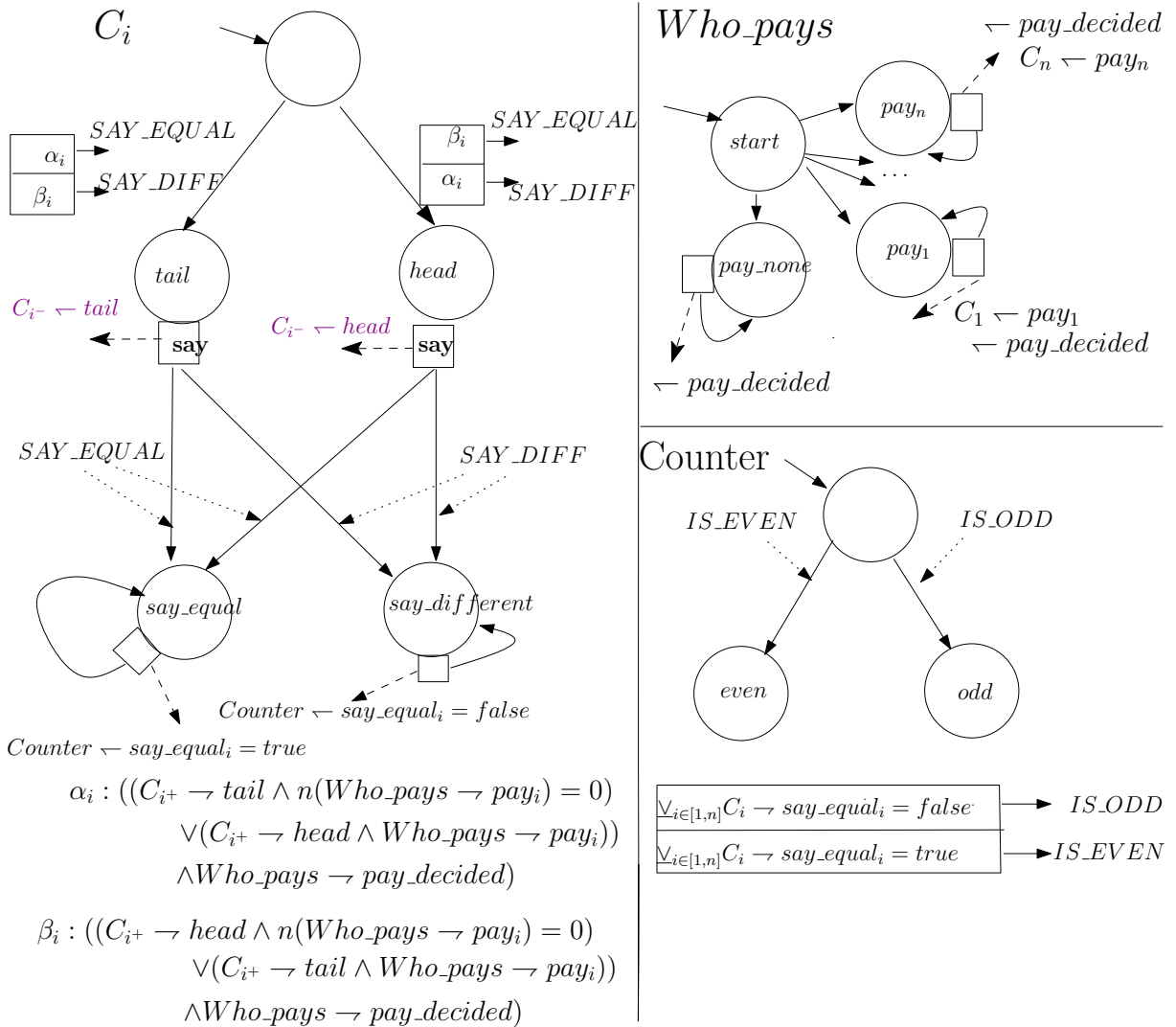


Figure 2: MIS for dining cryptographers (DC1)

propositions can be depicted in a similar way as for networks of automata. We omit them in our examples throughout, as they do not play a role in this paper.

3.3 Dining Cryptographers: Standard Version (DC1)

Dining Cryptographers is a well-known benchmark proposed by Chaum [5]. n cryptographers are having dinner, and the bill is to be paid anonymously, either by one of them or by their employer. In order to learn which option is the case without disclosing which cryptographer is paying (if any), they run a two-stage protocol. First, every cryptographer is paired with precisely one other participant (they sit around the table), thus forming a cycle. Every pair shares a one-bit secret, say by tossing a coin behind a menu. In the second stage, each cryptographer publicly announces whether he sees an odd or an even number of coin heads, saying the opposite if being the payer.

In the simplest case (DC1) the number of cryptographers is fixed, and each cryptographer is directly

bound with its neighbours. Cryptographers announce their utterances by broadcasting them. A modular interpreted system modeling this setting is presented in Figure 2. For n cryptographers, the i th cryptographer is modeled by agent C_i ($0 \leq i \leq n$). We introduce notation i^+ and i^- to refer to the right and the left neighbour of cryptographer i , respectively. The system includes also two additional agents. *Who_pays* initializes the system by determining who is the payer, and communicating it to the cryptographers. According to the protocol definition, either one of the cryptographers is chosen, or none of the participants pays. Agent *Counter* counts the utterances of the cryptographers, computes the XOR operation (denoted by $\underline{\vee}$ and assuming that utterances *different* and *equal* correspond to true and false values, respectively, thus the result is true iff the number of *different* utterances is odd), and determines the outcome of the protocol. Figure 2 shows the modular interpreted system for DC1.

4 How to Measure Multi-Agency

In this section, we present our preliminary attempt at defining what it means for a design to be multi-agent. Intuitively, separate agents should have only limited coordination and/or communication capabilities. Otherwise, the whole system can be seen as a single agent in disguise. The idea is to measure the complexity of interference between different agents, and relate it to the complexity of the system. The former factor will be captured by the number of directed interaction tokens that a given agent can generate; the latter by the number of global transitions that can occur. We say that the agent is well designed if its interference complexity is reasonably smaller than overall complexity of the system.

Definition 4 (Interaction complexity) *The interaction complexity of agent i in modular interpreted system M , denoted $IC(i)$, is defined as follows. Let $\#out_i(q_i)$ be the maximal number of directed tokens generated by function out_i to modules of other agents in state q_i . Furthermore, let $\#in_i(q_i)$ be the maximal number of tokens admitted by function in_i from modules of other agents in state q_i . Now, $IC(i) = \sum_{q_i \in St_i} (\#out_i(q_i) + \#in_i(q_i))$.*

The interaction complexity of M is defined as $IC(M) = \sum_{i \in \mathbb{A}_{gt}} IC(i)$.

Definition 5 (Global complexity) *The global complexity of MIS M , denoted $GC(M)$, is the number of transitions in the NCEGS unfolding of M .*

How can we express that $IC(M)$ is “reasonably smaller” than $GC(M)$? Such a requirement is relatively easy to specify for *classes* of models, parameterized with values of some parameter (for instance, the number of identical trains in the tunnel-controller scenario).

Definition 6 (\mathcal{C} -sparse interaction, multi-agent design) *Let \mathcal{M} be a class of MIS and \mathcal{C} a class of complexity functions $f : \mathbb{N} \rightarrow \mathbb{R}_+ \cup \{0\}$. We say that \mathcal{M} is characterized by \mathcal{C} -sparse interaction iff there is a function $f \in \mathcal{C}$ such that $IC(M) \leq f(GC(M))$ for every $M \in \mathcal{M}$.*

Furthermore, we say that \mathcal{M} has multi-agent design iff \mathcal{M} has LOGTIME-sparse interaction, and $card(M) \geq 2$ for every $M \in \mathcal{M}$.

Proposition 1 *Classes TTC and DC1 have multi-agent design.*

The proof is straightforward. It is easy to see that the other variants of Dining Cryptographers, discussed in Section 5, also have multi-agent design.

5 How Open is an Open System?

The idea of open systems is important for several communities: not only MAS, but also verification, software engineering, etc. It is becoming even more important now, with modern technologies enabling dynamic networks of devices, users and services whose nodes can be created and removed according to current needs. Traditionally, the term *open system* is understood as a process coupled with the environment, which is rather disappointing given the highly distributed nature of MAS nowadays. One would rather like “openness” to mean that components (agents in our case) can freely join and leave the system without the need to redesign the rest of it.

Perfectly open systems are seldom in practice; usually, adding/removing components requires some transformation of the remaining part (for instance, if a server is to send personalized information to an arbitrary number of clients then it must add the name of each new client to the appropriate distribution lists). So, it is rather the degree of openness that should be captured. We try to answer the question *How open is the system?* (or, to be more precise, its model) in the next subsection.

5.1 A Measure of Openness

We base the measure on the following intuition: openness of a system is *simplicity of adding and removing agents to and from the model*. That is, we consider two natural transformations of models: expansion (adding agents) and reduction (removing agents). We note that the simplicity of a transformation is best measured by its algorithmic complexity, i.e., the number of steps needed to complete the transformation. A perfectly open system requires no transformation at all (0 steps) to accommodate new components, whereas at the other extreme we have systems that require redesigning of the model from scratch whenever a new agent arrives.

Note that the openness of a model depends on *which* agents want to join or leave. For instance, the system with trains and controllers should be able to easily accommodate additional trains, but not necessarily additional controllers. Likewise, departure of a train should be straightforward, but not necessarily that of the controller. No less importantly, the context matters. We are usually not interested in an arbitrary expansion or reduction (which are obviously trivial). We want to add or remove agents while keeping the “essence” of the system’s behavior intact. The following definitions formalize the idea.

Definition 7 (Expansion and reduction of a MIS) *Let $M = (Agtnames, Act, \mathcal{I}n, \mathbb{A}gt)$ be a MIS, and \mathbf{a} an agent (in the sense of Definition 1). By $agt(\mathbf{a})$ (resp. $act(\mathbf{a})$, $in(\mathbf{a})$) we denote the set of agent names (resp. action symbols, interaction symbols) occurring in \mathbf{a} . Moreover, $ns(\mathbf{a}, M)$ will denote the set of \mathbf{a} ’s namesakes in M .¹ Note that $ns(\mathbf{a}, M)$ can contain at most 1 agent.*

The expansion of M by \mathbf{a} is defined as the modular interpreted system $M \oplus \mathbf{a} = (Agtnames', Act', \mathcal{I}n', \mathbb{A}gt')$ where: $Agtnames' = Agtnames \cup agt(\mathbf{a})$, $Act' = Act \cup act(\mathbf{a})$, $\mathcal{I}n' = \mathcal{I}n \cup in(\mathbf{a})$, and $\mathbb{A}gt' = \mathbb{A}gt \setminus ns(\mathbf{a}, M) \cup \{\mathbf{a}\}$. The reduction of M by \mathbf{a} is defined as $M \ominus \mathbf{a} = (Agtnames, Act, \mathcal{I}n, \mathbb{A}gt')$ where $\mathbb{A}gt' = \mathbb{A}gt \setminus \{\mathbf{a}\}$.

Thus, expansion corresponds to “dumb” pasting an agent into a MIS, and reduction corresponds to simple removal of the agent. The operations are well defined in the following sense.

Proposition 2 *Expansion/reduction of a MIS is always a MIS.²*

It is easy to see that removing an agent and pasting it in again does not change the MIS. The reverse sequence of operations does change the MIS. However, both structures have the same unfoldings:

¹ That is, agents in M that have the same id as \mathbf{a} .

² The proofs of results in Section 5 are straightforward from the construction of MIS, and we leave them to the reader.

Proposition 3 *Let \mathbf{a} be an agent in M . Then, $(M \ominus \mathbf{a}) \oplus \mathbf{a} = M$. Moreover, let \mathbf{a} be an agent with no namesake in M . Then, $NCEGS((M \oplus \mathbf{a}) \ominus \mathbf{a}) = NCEGS(M)$.*

Now we can make our first attempt at a measure of openness.

Definition 8 (Degree of openness) *Let θ be a property of models,³ M a modular interpreted system, and \mathbf{a} an agent. The degree of openness of M wrt expansion by \mathbf{a} under constraint θ is defined as the minimal number of steps that transform $M \oplus \mathbf{a}$ into a MIS M' such that $\text{card}(M') = \text{card}(M \oplus \mathbf{a})$ and M' satisfies θ .*

Likewise, the degree of openness of modular interpreted system M wrt reduction by agent \mathbf{a} under constraint θ is the minimal number of steps that transform $M \ominus \mathbf{a}$ into an M' such that $\text{card}(M') = \text{card}(M \ominus \mathbf{a})$ and M' satisfies θ .

The constraint θ can for example refer to liveness of the system or some of its components, fairness in access to some resources, and/or safety of critical sections. Note that the cardinality check is essential in the definition – otherwise, a possible transformation would be to simply delete the newly added agent from $M \oplus \mathbf{a}$ (respectively, to restore \mathbf{a} in $M \ominus \mathbf{a}$).

Definition 9 (Openness of a class of models) *Let \mathcal{M} be a class of MIS, \mathbf{a} an agent, and θ a property of models. Moreover, let \mathcal{C} be a class of complexity functions $f : \mathbb{N} \rightarrow \mathbb{R}_+ \cup \{0\}$. \mathcal{M} is \mathcal{C} -open wrt expansion (resp. reduction) by \mathbf{a} under constraint θ iff there is a complexity function $f \in \mathcal{C}$ such that for every $M \in \mathcal{M}$ the degree of openness of M wrt expansion (resp. reduction) by \mathbf{a} under θ is no greater than $f(|M|)$.*

The most cumbersome part of the above definitions is the constraint θ . How can one capture the “essence” of acceptable expansions and reductions? Note that, semantically, θ can be seen as a subclass of models. We postulate that in most scenarios the class that defines acceptable expansions/reductions is the very class whose openness we want to measure. This leads to the following refinement of the previous definitions.

Definition 10 (Openness in a class) *The degree of openness of M wrt expansion (resp. reduction) by \mathbf{a} in class \mathcal{M} is the minimal number of steps that transform $M \oplus \mathbf{a}$ (resp. $M \ominus \mathbf{a}$) into a MIS $M' \in \mathcal{M}$ such that $\text{card}(M') = \text{card}(M \oplus \mathbf{a})$.*

Moreover, \mathcal{M} is \mathcal{C} -open wrt expansion (resp. reduction) by \mathbf{a} iff there is a complexity function $f \in \mathcal{C}$ such that for every $M \in \mathcal{M}$ the degree of openness of M wrt expansion (resp. reduction) by \mathbf{a} in \mathcal{M} is no greater than $f(|M|)$.

We explain the measure in greater detail in the remainder of Section 5. It is important to note that (in contrast to the measure of multi-agentivity proposed in Section 4) our measure of openness is not specific to MIS, and can be applied to other modeling frameworks.

Remark 4 *Alternatively, we could define the openness of M wrt \mathbf{a} and θ by the Kolmogorov complexity of an appropriate expansion/reduction, i.e., by the size of the shortest algorithm that transforms M in an appropriate way. We chose time complexity instead, for two reasons. First, Kolmogorov complexity often obscures the level of difficulty of a process (e.g., a two-line algorithm with an infinite **while** loop can implement infinitely many changes, which gives the same complexity as changing the names of two communication channels for a controller). Secondly, computing Kolmogorov complexity can be cumbersome as it is Turing-equivalent to answering the halting problem.*

³ We do not restrict the language in which θ is specified. It can be propositional logic, first-order temporal logic, or even the general language of mathematics. The only requirement is that, for every MIS M , the truth of θ in M is well defined.

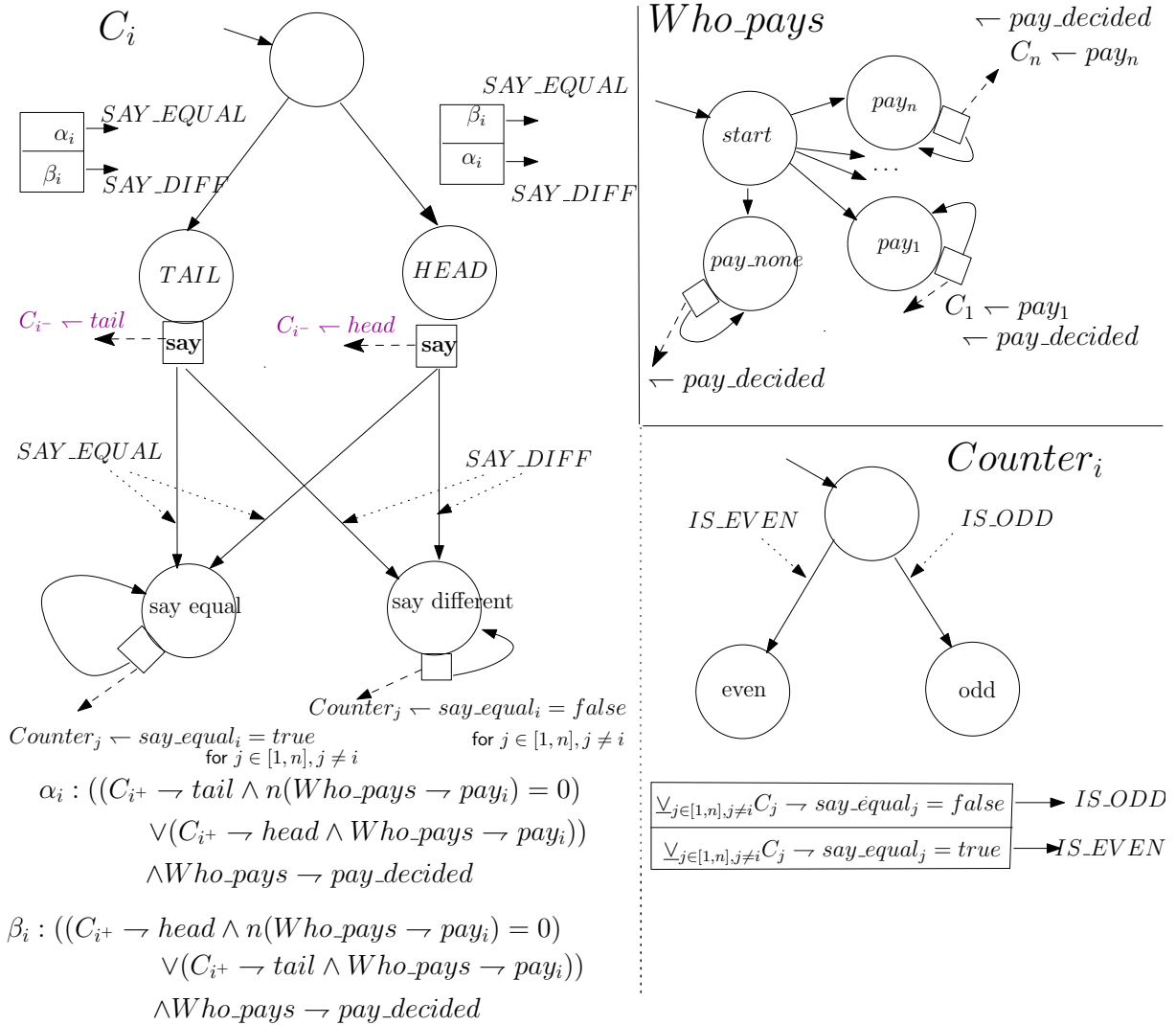


Figure 3: Dining cryptographers version DC2: direct channels instead of broadcast

We observe, however, that a Kolmogorov-style measure of openness can be a good alternative for infinite models, especially ones that require infinitely many steps to accommodate changes in the configuration of components.

5.2 How to Open Up Cryptographers

In Section 3.3 we modeled the standard version of the Dining Cryptographers protocol as a modular interpreted system (class DC1). In this section, we will determine the openness of DC1, plus two other classes of MIS modeling other versions of the protocol. To comply with classical rules of composition, we begin with the least open variant.

5.2.1 DC-Net, Direct Channels, No Broadcasting (DC2)

Let us assume that no broadcast channel is available, or it is too faulty (or insecure) to be of use in multi-party computation. In such case, every pair of cryptographers must use a direct secured channel for communicating the final utterance. The result of the computation is calculated independently by every cryptographer. We denote this class of models by DC2, and construct it as follows. Each cryptographer i is modeled by agent C_i , similar to the cryptographer agents in DC1. Instead of a single global counter of utterances, there is one counter per every cryptographer ($Counter_i$). The final utterance is sent by direct point-to-point channels to the counters of all other participants. The resulting MIS is shown in Figure 3.

Adding a new cryptographer C_i to $DC1_n$ requires the following changes. First, modifying links among the new neighbours of C_i yields 10 changes. Secondly, every agent C_j in $DC1_n$ must be modified in order to establish a communication channel with C_i . This requires $2 \cdot 5$ changes per cryptographer, thus $10n$ changes are needed. Thirdly, for the agent Who_pays , we add the state pay_i with corresponding transitions: a single non-deterministic transition from $start$ to pay_i (17 steps: 2 for d_i + 4 for out_i + 8 for in_i + 3 for o_i), and the loop sending payment information (19 steps: 4 for d_i + 4 for out_i + 4 for in_i + 3 for o_i). Finally, $Counter$ needs to be updated to take into account the new participant. A XOR argument is added with receiving a manifestation, yielding $2 \cdot 4 = 8$ changes. Thus, the overall openness complexity for $DC2_n$ is $10n + 54$.

Proposition 5 *Class DC2 is $O(n)$ -open wrt expansion by a cryptographer.*

5.2.2 Dining Cryptographers: Standard Version with Broadcast (DC1)

Let us now go back to the standard version of the protocol, presented in Section 3.3 Adding a new cryptographer C_i requires the following changes. First, modifying links for the new neighbors of C_i requires 10 changes. Secondly, changes in Who_pays and $Counter$ are the same as for $DC2_n$, yielding 44 steps. Thus, 54 changes are needed to accommodate the new cryptographer, regardless of the number of agents already present in the system.

Proposition 6 *Class DC1 is $O(1)$ -open wrt expansion by a cryptographer.*

5.2.3 Fully Open System, Cryptographers without Identifiers (DC0)

In our most radical variant, cryptographers are not arbitrarily assigned as neighbors. Instead, they establish their neighborhood relation on their own before starting the protocol. Every cryptographer is modeled by two modules C_i and Pay_i , and there are two additional agents $Oracle$ and $Counter$, cf. Figure 4. The system proceeds as follows:

Setting up the payer. Every cryptographer sends the oracle his declaration whether he is going to pay or not (chosen nondeterministically). This is performed by module Pay_i . If $Oracle$ receives at most one statement $want_pay$, it confirms to all cryptographers. If more than one statements $want_pay$ is sent, the round is repeated until the payment issue becomes resolved.

Establishing the neighbourhood relation and tossing coins. Each cryptographer either nondeterministically tosses a coin and announces the outcome, or listens to such announcements from the other agents. If there is exactly one cryptographer announcing and one listening, they become paired. They register the value of the announcement, and proceed further. A cryptographer who started with announcing will now listen, and vice versa. This takes several rounds, and completes when every cryptographer has been paired with two neighbors (one to whom he listened, and one to whom he announced).

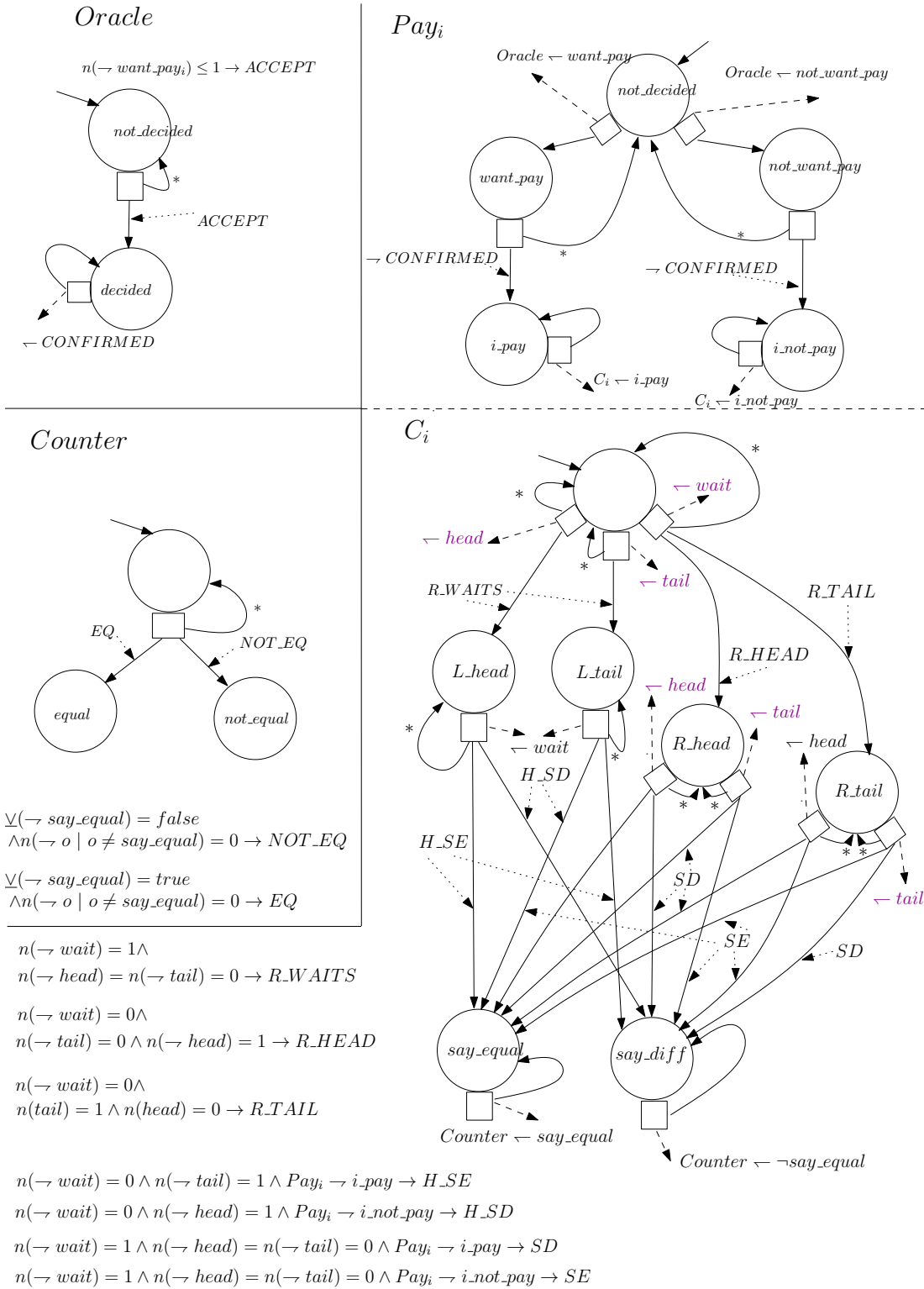


Figure 4: Cryptographers without identifiers (DC0)

Computation. A broadcast channel is used for sending around the utterances (*say_equal* or \neg *say_equal*). *Counter* counts the utterances and computes their XOR on the spot, in the way described before.

DC0 is fully open, as adding a new cryptographer requires no adaptation of $DC0_n$.

Proposition 7 *Class DC0 is $O(0)$ -open wrt expansion by a cryptographer.*

By comparing their classes of openness, it is clear that DC1 is significantly more open wrt expansion than DC2 (constant vs. linear openness). On the other hand, it seems that the gap between DC1 and DC0 is rather slight ($O(1)$ vs. $O(0)$). Is that really the case? We believe that the difference between $O(1)$ -openness and $O(0)$ -openness is larger than one is used to in complexity of algorithms. First, constant openness means that, when expanding the MIS by a *set* of new agents, the required transformation can be linear in the size of the set. More importantly, non-zero openness signifies the need to come up with a *correct* procedure of expansion. In contrast, zero openness means zero hassle: the new agents can join the system as they come. There is no need for “maintenance” of the system so that it stays compliant with its (usually implicit) specification.

6 Conclusions

In this paper, we propose a new version of modular interpreted systems. The aim is to let modeling and analysis of multi-agent systems benefit from true separation of interference between agents and the “internals” of their processes that go on in a system. Thanks to that, one can strive for a more modular and open design. Even more importantly, one can use the MIS representation of a system to assess its agentivity and openness through application of simple mathematical measures.

We emphasize that it was *not* our aim to create yet another agent programming language or representations that will be used as input to cutting-edge model checkers. Instead, we propose a class of models which enables to expose the internal structure of a multi-agent system, and to define the concepts of openness and multi-agentivity in a precise mathematical sense. While our definition of multi-agentivity is specific to MIS, the measure of openness is in fact generic, and can be applied to models defined in other formalisms (such as Reactive Modules). We plan to look closer at the degree of openness provided by different representation frameworks in the future.

We would also like to stress that the focus of this paper regarding the measures of agentivity and openness is on formalizing the concepts and showing how they work on benchmarks. An formal study of the measures and their properties is a matter of future work.

Acknowledgements. The authors thank Andrzej Tarlecki for his suggestion to improve modularity of MIS by using multisets, and Thomas Agotnes for discussions. Wojciech Jamroga acknowledges the support of the FNR (National Research Fund) Luxembourg under project GALOT – INTER/DFG/12/06. Artur Męski acknowledges the support of the European Union, European Social Fund. Project PO KL “Information technologies: Research and their interdisciplinary applications” (UDA-POKL.04.01.01-00-051/10-00).

References

- [1] R. Alur & T. A. Henzinger (1999): *Reactive Modules*. *Formal Methods in System Design* 15(1), pp. 7–48, doi:10.1023/A:1008739929481.

- [2] R. Alur, T. A. Henzinger & O. Kupferman (2002): *Alternating-Time Temporal Logic*. *Journal of the ACM* 49, pp. 672–713, doi:10.1145/585265.585270.
- [3] R. Alur, S. Kannan & M. Yannakakis (1999): *Communicating Hierarchical State Machines*. In: *Proceedings of ICALP*, pp. 169–178, doi:10.1007/3-540-48523-6.14.
- [4] J. Calta (2012): *Synthesis of Strategies for Multi-Agent Systems*. Ph.D. thesis, Humboldt University Berlin.
- [5] D. Chaum (1988): *The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability*. *Journal of Cryptology* 1(1), pp. 65–75, doi:10.1007/BF00206326.
- [6] P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Pórola, M. Szreter, B. Woźna & A. Zbrzezny (2003): *Verics: A Tool for Verifying Timed Automata and Estelle Specifications*. In: *Proceedings of the of the 9th Int. Conf. on Tools and Algorithms for Construction and Analysis of Systems (TACAS'03)*, LNCS 2619, Springer, pp. 278–283, doi:10.1007/3-540-36577-X_20.
- [7] E. A. Emerson (1990): *Temporal and Modal Logic*. In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science*, B, Elsevier Science Publishers, pp. 995–1072.
- [8] R. Fagin, J. Y. Halpern, Y. Moses & M. Y. Vardi (1995): *Reasoning about Knowledge*. MIT Press.
- [9] J. Fisher, T. A. Henzinger, D. Nickovic, N. Piterman, A. V. Singh & M. Y. Vardi (2011): *Dynamic Reactive Modules*. In: *Proceedings of CONCUR*, pp. 404–418, doi:10.1007/978-3-642-23217-6_27.
- [10] F. Gezeg (1986): *Products of Automata*. EATCS Monographs on Theor. Comput. Sci., Springer, doi:10.1007/978-3-642-61611-2.
- [11] G. J. Holzmann (1997): *The Model Checker SPIN*. *IEEE Transactions on Software Engineering* 23(5), pp. 279–295, doi:10.1109/32.588521.
- [12] W. Jamroga & T. Ågotnes (2006): *Modular Interpreted Systems: A Preliminary Report*. Technical Report IfI-06-15, Clausthal University of Technology.
- [13] W. Jamroga & T. Ågotnes (2007): *Modular Interpreted Systems*. In: *Proceedings of AAMAS'07*, pp. 892–899, doi:10.1145/1329125.1329286.
- [14] M. Köster & P. Lohmann (2011): *Abstraction for model checking modular interpreted systems over ATL*. In: *Proceedings of AAMAS*, pp. 1129–1130.
- [15] F. Laroussinie, N. Markey & G. Oreiby (2008): *On the Expressiveness and Complexity of ATL*. *Logical Methods in Computer Science* 4, p. 7, doi:10.2168/LMCS-4(2:7)2008.
- [16] O. Lichtenstein & A. Pnueli (1985): *Checking that finite state concurrent programs satisfy their linear specification*. In: *POPL '85: Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, ACM, New York, NY, USA, pp. 97–107, doi:10.1145/318593.318622.
- [17] A. Lomuscio & F. Raimondi (2006): *MCMAS : A Model Checker for Multi-agent Systems*. In: *Proceedings of TACAS, Lecture Notes in Computer Science* 4314, pp. 450–454, doi:10.1007/11691372_31.
- [18] A. Murano, M. Napoli & M. Parente (2008): *Program Complexity in Hierarchical Module Checking*. In: *Proceedings of LPAR*, pp. 318–332, doi:10.1007/978-3-540-89439-1_23.
- [19] F. Raimondi (2006): *Model Checking Multi-Agent Systems*. Ph.D. thesis, University College London.
- [20] S. La Torre, M. Napoli, M. Parente & G. Parlato (2008): *Verification of scope-dependent hierarchical state machines*. *Information and Computation* 206(9-10), pp. 1161–1177, doi:10.1016/j.ic.2008.03.017.