Electronic Notes in Theoretical Computer Science 85 No. 2 (2004)
URL: http://www.elsevier.nl/locate/entcs/volume85.html 16 pages

# A Tool For Specification And Verification Of Epistemic Properties In Interpreted Systems

Franco Raimondi and Alessio Lomuscio [1]

*Department of Computer Science*
*King's College London*
*London WC2R 2LS, UK*

**Abstract**

We present a compiler that translates a multi-agent systems specification given in the formalism of Interpreted Systems into an SMV program. We show how an SMV model checker can be coupled with a Kripke model editor (Akka) to allow for the mechanical verification of epistemic properties of multi-agent systems. We apply this methodology to the verification of a communication protocol — the dining cryptographers.

## 1 Introduction

Formal logic is traditionally seen as a powerful tool in the analysis, representation, and interpretation of communication. With the advent of distributed systems, logic, and formal methods, have provided two concrete tools to researchers involved with issues relating to communication: a specification language, and a verification mechanism.

Logic is used as a *specification language* for communication when analysing protocols and meaning of utterances of artificial languages, such as in the recent application of speech-act theory to communication in multi-agent systems. Formal methods based on formal logic are used as a *verification mechanism* in the analysis of properties of communication protocol for distributed systems. This paper concerns the use of machinery based on logic for the latter.

Verification of communication protocols is generally performed either by theorem provers or by model checkers. While theorem provers are established technology, *model checking* [8] is a relatively recent technique for the verification of distributed systems, allowing for the mechanical verification of prop-

[1] Email: {franco,alessio}@dcs.kcl.ac.uk.
This work was supported by the Nuffield Foundation grant NAL/00690/G.

erties expressed by means of temporal formulae. Temporal logic is a powerful formalism, but it is not expressive enough to represent the typical properties we are typically concerned with in a multi-agent system: notably the knowledge, and other attitudes such as desires and intentions, of the agents. J. Halpern and M. Vardi suggested the use of model checking techniques in the verification of multi-agent systems in 1991 ([10]) by means of richer languages including not only temporal operators but also epistemic, but it is only recently that results along these lines have been achieved ([1,15,18,22,2,17,14,11,12]).

Irrespective of recent research, verifying a concrete communication protocol remains a non-trivial task. First one needs to give a concrete computational model of the system — either by means of Petri nets, timed automata, etc. Once this is given, one needs a tool that automatically builds the semantics model for the system. This semantical model, typically a temporal model, must then be used to interpret the language that is used to specify and verify properties about the system. While some tools are available, to our knowledge there is currently no unified platform available, that can assist the designer in the process from concrete specification of the different automata for the agents to the verification of properties by means of a model checker able to check logics richer than plain temporal logic. The difficulty with providing an all-encompassing platform is that several issues are intertwined:

- What formalism — automata, Petri nets — is to be used to represent the transitions in the components resulting from a communication protocol?
- What temporal model — Interpreted Systems, plain Kripke semantics — is to be employed to represent the computation paths defined by the low-level description of the system?
- What logical language — temporal, epistemic, deontic — is to be employed to represent crucial properties of the protocol under consideration?
- What particular symbolic representation — OBDD's, SAT-based, etc — is to be used for the model checking task?
- What specific model checker - NuSMV, Spin, etc — should be employed to assist in the task?

Many competing options are enumerated above, and there is currently no "correct way to proceed", but rather, it seems to us, a spectrum of options are available to investigate further for parties interested in these issues. In view of making a contribution on the issues outlined above, in this paper we present a tool that integrates traditional model checking techniques with Interpreted Systems semantics [9]. Interpreted Systems are a powerful formalism to reason about epistemic and temporal properties of a MAS. The tool presented here allows for the verification of static epistemic properties of an Interpreted System (i.e. properties involving epistemic operators only). We argue that this is sufficient in a number of cases; to support this claim, we apply the tool to the verification of a communication protocol—the protocol of the dining

cryptographers [6].

The rest of the paper is organised as follows. In Section 2 we review the main technical constructions used in the rest of the paper. In Section 3 we present and discuss a methodology for checking epistemic properties of a MAS. In Section 4 we present in some detail the tool that allows us to do so. In Section 5 we show how this can be put to work on a widely discussed example — the dining cryptographers.

## 2 Review of concepts and notation

### *2.1 Model Checking techniques*

Given a program $P$, and a property that can be represented as a logical formula $\varphi$ in some logic, model checking techniques allow for the automatic verification of whether or not a model $M_P$, representing the program $P$, satisfies the formula $\varphi$.

In the last two decades there have been great advances in the effectiveness of this approach thanks to sophisticate data manipulation techniques. Techniques based on Binary Decision Diagrams (BDDs, [3]) have been used to develop model checkers that are able to check large number of states ([4]). Alternative approaches using automata have also been developed [20].

Software tools originated from these lines of research. SPIN (see [13]) exploits automata theory and related algorithms, while SMV [16] uses BDDs to represent states and transitions. In this paper we will use NuSMV, a novel implementation of SMV ([7]).

The input language of NuSMV allows for the specification of a finite system with different levels of abstraction. In the simplest case, the input language requires three main sections:

1. A section for variables declaration,
2. A section for variable initialisation,
3. A section for the description of the transition relation.

The following is an example of a NuSMV program. [2] :

```
MODULE main
VAR
  request : boolean;
  state   : {ready, busy};
ASSIGN
  init(state) := ready;
  next(state) :=
    case
      state = ready & request = 1 : busy;
      1     : {ready, busy};
    esac;
```

[2] From NuSMV tutorial, available at `http://nusmv.irst.itc.it`

(the line `"1 : {ready, busy};"` is equivalent to an `else` condition in traditional programming languages). Given the program above, NuSMV can then be used to create a model associated with it, and then to model check temporal formulae. For example, if we were to feed a NuSMV checker with the CTL formula
`AG((request=0) -> AF(state=ready))` [3]
NuSMV would produce a counterexample.

Following this approach a large number of systems ranging from communication protocols to hardware components have been verified by means of temporal languages. These are not agent systems, but standard distributed processes. If we are to investigate whether we can apply this methodology to agent verification, we need to incorporate this technique with an agent based semantics, like Interpreted Systems.

### *2.2 Interpreted Systems*

Interpreted Systems [9] are a computationally grounded semantics in the sense of [21], aimed at representing agents in a distributed setting. We present the main definitions here, but refer to the literature for more details.

Consider $n$ agents in a system and $n$ non-empty sets $L_1, \ldots, L_n$ of local states, one for every agent of the system, and a set of states for the environment $L_E$. Elements of $L_i$ will be denoted by $l_1, l_1', l_2, l_2', \ldots$. Elements of $L_E$ will be denoted by $l_E, l_E', \ldots$.

A *system of global states for n agents* $S$ is a non-empty subset of a Cartesian product $L_1 \times \ldots \times L_n \times L_E$. When $g = (l_1, \ldots, l_n, l_E)$ is a global state of a system $S$, $l_i(g)$ denotes the local state of agent $i$ in global state $g$. $l_E(g)$ denotes the local state of the environment in global state $g$.

We assume that, for every agent in the system and for the environment, there is a set $Act_i$ and $Act_E$ of actions that the agents and the environment can perform. Actions are not executed randomly, but following particular specifications that we call protocols. A protocol $P_i$ for agent $i$ is a function from the set $L_i$ of local states to a non-empty set of actions $Act_i$ (notice that, by considering *sets of actions*, we allow non-determinism in the protocol):

$$P_i : L_i \rightarrow 2^{Act_i}.$$

We can then model the evolution of the system by means of a transition function $\pi$ from global states and joint actions to global states:

$$\pi : S \times Act \rightarrow S$$

where $S = L_1 \times \ldots \times L_n \times L_E$ and $Act = Act_1 \times \ldots \times Act_n \times Act_E$ is the set of joint actions for the system.

Intuitively this defines temporal flows on the set of global states. Specifically, we consider a set of runs over global states $R = \{r : \mathbb{N} \rightarrow S\}$, where

[3] In the formula AG is the modal operator for "forever in the future in all branches", and the propositions are to be interpreted in the intuitive way.

a run $r$ is defined as a function from time to global states, and time ranges over natural numbers. A run is a sequence of global states that are obtained by applying the function $\pi$ to global states and joint actions.

Interpreted Systems can be used to model time and knowledge. To do that, consider a pair $IS = (S, h)$ where $S$ is a set of global states and $h : S \to 2^P$ is an interpretation function for a set of propositional variables $P$.

Temporal connectives of the type of CTL [16] can then be evaluated on Interpreted Systems. For the purposes of this paper we are concerned with epistemic operators. These can be interpreted by means of epistemic modalities $K_i$, one for each agent, as follows [9]:

$$(IS, g) \models K_i \varphi \quad \text{if for all } g' \text{ we have that } l_i(g) = l_i(g') \text{ implies } (IS, g') \models \varphi.$$

The resulting logic for the modalities $K_i$ is $S5_n$; this models agents with complete introspection capabilities and veridical knowledge.

We shall use Interpreted Systems as a semantic basis to specify a MAS. They will also be represented in NuSMV in the verification process.

## 3 A methodology for model checking epistemic properties in Interpreted Systems

While MAS theories encompass a variety of attitudes, in this paper we focus on knowledge. Being able to verify temporal epistemic properties of a system would allow us to reason in terms of temporal evolution of knowledge, and knowledge about a changing world. We argue that, in particular cases, verification of static properties is adequate. This is so in all circumstances in which preconditions and postconditions can be stated in terms of logical propositions. We give an example of this in Section 5.

In order to specify and verify the epistemic properties of a MAS, we identify the following procedure:

(i) **Specify an Interpreted System in terms of local states, protocols, and transitions**. We give a concrete example of this in Section 5, for the protocol of the dining cryptographers.

(ii) **Translate the specification of step 1 into an SMV program**. This can be done automatically; a Java program (presented below) can be used to perform this translation.

(iii) **Use a model checker to compute the set of reachable states**. Given a symbolic representation for states and transitions, as the one obtained in the previous step, the set of reachable states can be computed as a fixed-point operator [16]. NuSMV provides this facility from version 2.1. Notice that temporal properties of the MAS can be checked at this stage.

(iv) **Build an epistemic model from the set of reachable states**. The

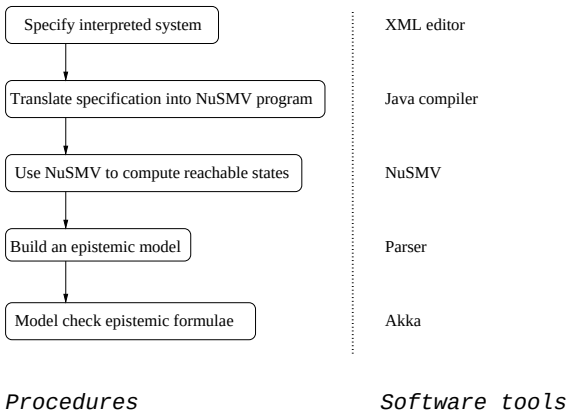


Fig. 1. Methodology

output of the model checker is used as a starting point for the definition of the epistemic model of the original Interpreted System. The epistemic relations are built automatically, parsing the output of NuSMV by means of a software we wrote.

(v) **Model checking epistemic formulae**. In the present paper we use Akka [4], a Kripke model editor that supports model testing by Lex Hendriks. Akka accepts the description of a model and an evaluation function and allows formulae to be checked against this input. Akka poses no restriction on the syntax of the formulae, so that formulae can involve more than one modal operator, and modalities can be nested. The model and the evaluation function (in the syntax of Akka) are provided by the parser in the previous step.

The methodology is summarised in Figure 1.

## 4 Translating Interpreted Systems into SMV code

In this section we present the tool that we use to translate Interpreted Systems into SMV. We first state a number of assumptions we make on the specification, and then briefly describe the tool.

### *4.1 Assumptions on the Interpreted System*

We restrict our attention to the class of Interpreted Systems with the following properties:

- **Finite systems**: we consider systems with a finite number of local states and actions. This is a limitation, but adequate for many examples where the set of state is finite.
- **Initial configuration**: we are required to specify the number of agents, local states and actions when setting up the model. Hence, the maximum

[4] `http://turing.wins.uva.nl/~lhendrik/`

number of local states cannot change at run-time.

- **Local states**: we assume that local states can be represented as a list of variables, each having a finite range of values. More in detail, consider an agent $i$: the local state $L_i$ is a tuple $L_i = \langle v_{1,i}, \ldots, v_{n,i} \rangle$ where each $v_{n,i}$ ranges over a finite set of values (see Section 5 for an example).
- **Evolution function**: In the description below we use a slightly modified and simpler syntax for $\pi$ (see Section 2.2); the idea is to decompose the final global states of the function $\pi$. We consider $n$ evolution functions, one for each agent, $\pi_i : S \times Act \rightarrow L_i$ $(i = 1, \ldots, n)$ from global states and actions to local states of agent $i$. In the tool, we shall list only the global states and actions that cause a change in the local state of agent $i$, and assume that, if a global state is not listed in the definition of some $\pi_i$, then this global state is not relevant in the evolution of $L_i$.

### *4.2 Input and Output of the Java Translator*

The specification of an Interpreted System is required as an input for the Java translator. This specification must contain at least the following informations:
1. Number of agents.
2. Number of local states and actions for each agent.
3. Number of variables in each local state, for each agent; values of each variable in the local state.
4. Protocol as a function from local states (i.e. set of variables) to actions, one for each agent.
5. Initial state(s).
6. Transition functions from local states and actions to a single local state (see previous Section).

These parameters are read from an XML file. The following is a schematic representation of the specification of an Interpreted System, as it is read by the translator [5]:

```
<is>
  <agent name="Agt1">
    <localstates nvar="1">
      [...]
   </localstates>
   <actions number="3">
      [...]
   </actions>
   <protocol>
     [...]
   </protocol>
  </agent>
```

[5] A DTD for the specification of Interpreted Systems can be found at:
`http://www.dcs.kcl.ac.uk/pg/franco/is/is.dtd`.

```
  [...]
  <evFunct>
    <agtEvFunct agtname="Agt1">
      <transition>
        [...]
      </transition>
    </agtEvFunct>
  </evFunct>
</is>
```

The protocol and the evolution functions are specified as sets of pairs (local state, actions) and (global state + actions, local state). For the evolution functions, we assume that if a global state and/or action is not listed, then it does not affect the change of the local state of the agent.

We chose to use XML to specify Interpreted Systems for the following reasons:

- The description of an Interpreted System requires the specification of simple data structures such as lists and maps, and XML allows for the description of this kind of semi-structured data.
- Parsers and checkers are freely available for most of the programming languages, thus enabling an easier integration with existing tools.
- The proposed DTD can be extended following new requirements.

Editing an XML file can be cumbersome, and we are currently developing a graphical interface to make the input of the parameters easier.

Given the input above, an SMV program is generated by the translator; each agent has two variables, one for a list of local states, and one for a list of actions. Local states are computed automatically from the list of local variables.

Essentially, the protocol in the Interpreted System gives the rules to compute the evolution of actions in the SMV code. The evolution function from the specification of the Interpreted System is used to create the block of SMV code needed for the transition functions between the variables representing local states.

The Java software performing the translation can be obtained from the authors of this paper.

## 5 A communication example: The Dining Cryptographer

It is known that Interpreted Systems provide a good abstraction model to specify and verify the behaviour of systems. The tool presented in Section 4, apart from being an exercise in compilation of specifications, allows us to go from an abstract description of an Interpreted System to the execution traces of it in a format that is compatible to one of the leading model checkers.

We are interested in specifying systems via Interpreted Systems because we regard them as promising in the verification of communication protocols, as demonstrated in [9,19]. We test this belief by using the scenario of the Dining Cryptographers, provided by Chaum [6].

In his paper, Chaum shows how messages can be broadcasted anonymously. In particular, he shows that protocols exist that allow for the change in the knowledge of the participants about some global property of the system, without them being able to detect the source of this information.

### *5.1 Statement of the Problem*

The Dining Cryptographers scenario is introduced in [6] as follows:

"*Three cryptographers are sitting down to dinner at their favourite three-star restaurant. Their waiter informs them that arrangements have been made with the maitre d'hotel for the bill to be paid anonymously. One of the cryptographers might be paying for the dinner, or it might have been NSA (U.S. National Security Agency). The three cryptographers respect each other's right to make an anonymous payment, but they wonder if NSA is paying. They resolve their uncertainty fairly by carrying out the following protocol:*

*Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer on his right, so that only the two of them can see the outcome. Each cryptographer then states aloud whether the two coins he can see – the one he flipped and the one his left-hand neighbour flipped – fell on the same side or on different sides. If one of the cryptographers is the payer, he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number indicates that NSA is paying (assuming that the dinner was paid for only once). Yet if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is.*"[6]

Notice that the same protocol works for any number of cryptographers either greater or equal to three (see [6]).

### *5.2 The Interpreted System of the Dining Cryptographers*

We analyse the scenario above by means of Interpreted Systems semantics. We introduce three agents $C_i$, $i = \{1, 2, 3\}$, to model the three cryptographers, and one agent $E$ for the environment. In our representation the environment is used to (non-deterministically) select the initial configuration of the payer and the results of coin tosses.

We represent the local state $LC_i$ for each cryptographer $C_i$ with a tuple

$LC_i = \langle v_1, v_2, v_3 \rangle$ where [6] :

$$v_1 = \begin{cases} \lambda & \text{the initial state} \\ \text{NotPaid} & \text{if the agent did not pay for the dinner} \\ \text{Paid} & \text{if the agent paid for the dinner} \end{cases}$$

$$v_2 = \begin{cases} \lambda & \text{initial state} \\ \text{Different} & \text{if the left coin is different from the} \\ & \text{right coin for } C_i \\ \text{Equal} & \text{if the left coin is equal to the right coin} \end{cases}$$

$$v_3 = \begin{cases} \lambda & \text{initial state} \\ \text{Odd} & \text{odd number of differences uttered} \\ \text{Even} & \text{even number of differences uttered} \end{cases}$$

Local states for the environment are tuples $LE$ of the form $LE = \langle ChA, ChB, ChC, payer \rangle$ where $ChA, ChB, ChC$ are the "channels" between the Cryptographers, with value randomly selected at the beginning of the run being Head or Tail (the outcome of the coin toss), and

$$payer = \begin{cases} 1 \text{ if } C_1 \text{ paid for the dinner} \\ 2 \text{ if } C_2 \text{ paid for the dinner} \\ 3 \text{ if } C_3 \text{ paid for the dinner} \\ 4 \text{ if the NSA paid for the dinner} \end{cases}$$

The actions for the cryptographers are:

$$ActC_1 = ActC_2 = ActC_3 = \{\lambda, \text{say(equal)}, \text{say(not equal)}\}$$

where $\lambda$ denotes a null action.

We assume that the environment is not performing any action: $ActE = \lambda$. Hence, there is no protocol for the environment [7] .

[6] From now on we will denote an empty or undefined state by $\lambda$.

[7] Equivalently one can think of a protocol mapping every local state for the environment to the null action $\lambda$.

The protocol $PC_i$ for the cryptographers is:

$$PC_i(LC_i) = \begin{cases} \text{say(equal)} & \text{if } LC_i \text{ is of the form} \\ & \langle \text{ NotPaid,Equal,* } \rangle \text{ or} \\ & \langle \text{ Paid,NotEqual,* } \rangle \\ \text{say(not equal)} & \text{if } LC_i \text{ is of the form} \\ & \langle \text{ NotPaid,NotEqual,* } \rangle \text{ or} \\ & \langle \text{ Paid,Equal,* } \rangle \\ \lambda & \text{all the remaining cases} \end{cases}$$

We now define the initial state for the system. We take the following initial state for the agents representing the cryptographers:

$$\text{init}(LC_1) = \text{init}(LC_2) = \text{init}(LC_3) = \langle \lambda, \lambda, \lambda \rangle$$

The initial state for the environment is randomly selected from the set of possible combinations of values for Channels (Head or Tail) and payer (one of the cryptographers or the NSA).

The evolution of the system is modelled by the transition function $\pi : G \times Act \to G$, where $G = LC_1 \times LC_2 \times LC_3 \times LE$ is the set of global states, and $Act = ActC_1 \times ActC_2 \times ActC_3 \times ActE$. Notice that we can skip the evolution of $LE$ and the dependences from $ActE$ in the definition of $\pi$, thanks to our assumptions on the environment (no actions, and local state fixed at the beginning of the run).

Even so, the definition of $\pi$ is too long to report; we will give here only two examples:

$$\pi(\langle \lambda, \lambda, \lambda \rangle, \langle \lambda, \lambda, \lambda \rangle, \langle \lambda, \lambda, \lambda \rangle, \langle \text{head,tail,head}, 1 \rangle, \lambda, \lambda, \lambda, \lambda)$$

$$=$$

$$(\langle \text{Paid,Different}, \lambda \rangle, \langle \text{NotPaid,Different}, \lambda \rangle,$$

$$\langle \text{NotPaid,Different}, \lambda \rangle,$$

$$\langle \text{Head,Tail,Head}, 1 \rangle)$$

The above represents the fact that in the initial state in which the results of coin tosses are Head, Tail, Head for $ChA$, $ChB$, and $ChC$ respectively, and in which the first cryptographer paid for the dinner, there exists a transition to a state where $C_1$ has value $Paid$ for the local variable $v_1$, while the others cryptographers have $NotPaid$.

At the next time step the cryptographers utter the appropriate sentence (*equal* or *not equal*), following their protocol. This enables the transitions for

the evaluation of the last variable, $v_3$:

$$\pi(\langle \text{Paid, NotEqual}, \lambda\rangle, \langle, \text{NotPaid, NotEqual}, \lambda\rangle,$$
$$\langle \text{NotPaid, NotEqual}, \lambda\rangle,$$
$$\langle \text{Head,Tail,Head}, 1\rangle,$$
$$\text{say(equal)}, \text{say(not equal)}, \text{say(not equal)}, \lambda)$$
$$=$$
$$(\langle \text{Paid,NotEqual,Odd}\rangle,$$
$$\langle \text{NotPaid,NotEqual,Odd}\rangle,$$
$$\langle \text{NotPaid,NotEqual,Odd}\rangle,$$
$$\langle \text{Head,Tail,Head}, 1\rangle)$$

This is the final state of the system. A similar analysis can be carried out for all the other remaining cases.

### *5.3 The Methodology in Practice*

Following the considerations above, we encoded the Interpreted System for the dining Cryptographers as an XML file. Specifically, this contains four agents, three variables for the local states of the Cryptographers, four variables for the environment, two actions for the Cryptographers.

The definition of the evolution function is the most cumbersome step. However, thanks to our assumptions of Section 4.1, we can specify only the global states and actions that actually cause a change on local states.

For example, under this assumption, the first cryptographer can be modelled with transitions of the form:

$$\langle \text{Paid, Equal}, \lambda\rangle \text{ if } (\langle \lambda, \lambda, \lambda\rangle, *, \langle 1, \text{Head, Head}, *\rangle, ), (*)$$
$$\text{or } (\langle \lambda, \lambda, \lambda\rangle, *, \langle 1, \text{Tail, Tail}, *\rangle, ), (*)$$

This represents the fact that the first Cryptographer would change his local state to $\langle \text{Paid, Equal}, \lambda\rangle$ only if he was in the local state $\langle \lambda, \lambda, \lambda\rangle$ and the environment was $\langle 1, \text{Head, Head}, *\rangle$ or $\langle 1, \text{Tail, Tail}, *\rangle$.

Similarly, it is possible to define all the remaining conditions causing a transition for the first cryptographer; these, together with the transitions for the other cryptographers and the environment, are encoded in XML for the Java translator.

One can feed this specification into the translator and produce the SMV code for the example[8]. NuSMV can then be used to generate the set of reachable states. For this example, these are 96 out of 629856 possible combinations of local states and actions, as they are represented in NuSMV. Both

[8] The code is available at: `http://www.dcs.kcl.ac.uk/pg/franco/is/dincry2.smv`.

the translation of the specification into SMV code and the computation of the set of reachable states require less than one second on a 500 MHz PC with 256 Mbytes of RAM. The reachable states are stored in a text file that can be processed by the parser to produce the epistemic model $IS_{\mathrm{d}}$ in Akka's format.

### *5.4 Model Checking the Formulae*

We define a set of atomic propositions $\{\mathbf{paid_1}, \mathbf{paid_2}, \mathbf{paid_3}, \mathbf{even}, \mathbf{odd}\}$ that we can interpret in a natural way in the model $IS_{\mathrm{d}}$ obtained by following the process described above. Notice that $\{\mathbf{even}, \mathbf{odd}\}$ are true upon termination of the protocol, thus giving the required postconditions for the evaluation of epistemic formulae [9]:

$$(IS_{\mathrm{d}}, g) \models \mathbf{paid_1} \text{ if } l_{C_1}(g) = \langle \mathrm{Paid}, *, * \rangle$$
$$(IS_{\mathrm{d}}, g) \models \mathbf{paid_2} \text{ if } l_{C_2}(g) = \langle \mathrm{Paid}, *, * \rangle$$
$$(IS_{\mathrm{d}}, g) \models \mathbf{paid_3} \text{ if } l_{C_2}(g) = \langle \mathrm{Paid}, *, * \rangle$$
$$(IS_{\mathrm{d}}, g) \models \mathbf{even} \quad \text{if } l_{C_i}(g) = \langle *, *, \mathrm{Even} \rangle \text{ for every } i$$
$$(IS_{\mathrm{d}}, g) \models \mathbf{odd} \quad \text{if } l_{C_i}(g) = \langle *, *, \mathrm{Odd} \rangle \text{ for every } i$$

With Akka we can easily check the following propositions:

$$IS_{\mathrm{d}} \models \mathbf{odd} \rightarrow (\neg \mathbf{paid_1} \rightarrow (K_{C_1}(\mathbf{paid_2} \vee \mathbf{paid_3}) \wedge \neg K_{C_1}(\mathbf{paid_2}) \wedge \neg K_{C_1}(\mathbf{paid_3})))$$

$$IS_{\mathrm{d}} \models \mathbf{even} \rightarrow K_{C_1}(\neg \mathbf{paid_1} \wedge \neg \mathbf{paid_2} \wedge \neg \mathbf{paid_3})$$

These two formulae confirm the correctness of the statement of section 5.1: if the first cryptographer did not pay for the dinner and there is an odd number of differences in the utterances, then the first cryptographer knows that either the second or the third cryptographer paid for the dinner; moreover, in this case, the first cryptographer does not know which one of the remaining cryptographers is the payer.

Conversely, if the number of differences in the utterances is even, then the first cryptographer knows that nobody paid for the dinner.

Interestingly, in our model the following is not valid:

$$IS_{\mathrm{d}} \not\models \neg \mathbf{paid_1} \rightarrow (K_{C_1}(\neg \mathbf{paid_1} \wedge \neg \mathbf{paid_2} \wedge \neg \mathbf{paid_3}) \vee (K_{C_1}(\mathbf{paid_2} \vee \mathbf{paid_3}) \wedge \neg K_{C_1}(\mathbf{paid_2}) \wedge \neg K_{C_1}(\mathbf{paid_2})))$$

[9] In the following, $g$ will denote a global state; $l_{C_i}(g)$ will denote the local state for Cryptographer $i$ in global state $g$; $\langle Paid, *, * \rangle$ will be a local state in which the first variable is $Paid$ and all the other variables are allowed to have any value.

Also, we have:

$$IS_{\mathrm{d}} \not\models \neg\mathbf{paid_1} \rightarrow (K_{C_1}(\neg\mathbf{paid_1} \wedge \neg\mathbf{paid_2} \wedge \neg\mathbf{paid_3}) \vee K_{C_1}(\mathbf{paid_2} \vee \mathbf{paid_3}))$$

Indeed, consider a global state in which the local state for $C_1$ is $\langle$NotPaid,Different,$\lambda\rangle$ (such a global state exists in the set of reachable global states). In this state $\mathbf{paid_1}$ does not hold; also, in this local states there is no information about the parity of the utterances and $C_1$ considers possible global states in which parity is *Odd*, and others in which parity is *Even*. In the first case, $\neg\mathbf{paid_1} \wedge \neg\mathbf{paid_2} \wedge \neg\mathbf{paid_3}$ does not hold in a global state that $C_1$ considers possible. In the second case $\mathbf{paid_2} \vee \mathbf{paid_3}$ is false, thus invalidating $K_{C_1}(\mathbf{paid_2} \vee \mathbf{paid_3})$.

## 6 Conclusions

Logic has always been of use in the analysis of communication in multi-agent systems, both for the case of humans and computers. To date, verification of communication protocols has been limited to the use of theorem provers, and model checkers limited to temporal languages. While this is appropriate for the low-level communication protocols used in networking, complex multi-agent systems following in spirit the intentional stance need richer languages. The problem with using richer languages to verify these protocols is that current provers and checkers are not suited to represent other modalities such as knowledge. In this paper we have attempted to take a step in this direction, by providing a path from a concrete specification of a multi-agent system to the construction of execution traces, and checking of properties.

Specifically, we have here presented a tool for model checking epistemic formulae in multi-agent systems. We have used Interpreted Systems as a framework for the specification of MAS and we have suggested how a model checker for temporal models (NuSMV) may be used in the verification of epistemic properties. A software tool to provide the necessary translation was discussed.

The tool provided has been tested on a well known scenario in communication: the protocol of the dining cryptographers. In the future we would like to test other scenarios, particularly from the security literature. In that exercise it would be instructive to check whether a static analysis is sufficient (as it is claimed by [5] in their influential paper on BAN logic), or whether a move to a temporal epistemic is required. While this analysis is in progress we are currently planning to add a graphical interface to the tool so that a specification of Interpreted Systems can be given graphically.

The issue of scalability of this approach is also one that we would like to investigate further. Preliminary results seem to indicate that the phases of

compilation into SMV, the construction of the set of global states, and the testing of epistemic formulae all scale up fairly well. Still, we would not expect this approach to be compared in speed with the fastest methodologies available. What we do find of interest here is that a bridge was made between specification of a protocol and model checking, by means of automatic compilation of one specification into another, thereby allowing for epistemic properties to be verified.

## References

[1] M. Benerecetti, F. Giunchiglia, and L. Serafini. Model checking multiagent systems. *Journal of Logic and Computation*, 8(3):401–423, June 1998.

[2] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, July 2003.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, pages 677–691, August 1986.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: $10^{20}$ states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[5] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[6] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.

[7] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model verifier. *Lecture Notes in Computer Science*, 1633, 1999.

[8] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[9] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.

[10] J. Halpern and M. Y. Vardi. Model checking vs. theorem proving: A manifesto. In J. Allen, R. E. Fikes, and E. Sandewall, editors, *Proceedings 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning, KR'91*, Morgan Kaufmann Series in Knowledge Representation and Reasoning, pages 325–334. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[11] W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In *SPIN 2002 — Proceedings of the Ninth International SPIN Workshop on Model Checking of Software*, Grenoble, France, April 2002.

[12] W. van der Hoek and M. Wooldridge. Tractable multiagent planning for epistemic goals. In M. Gini, T. Ishida, C. Castelfranchi, and W. L. Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 1167–1174. ACM Press, July 2002.

[13] G. J. Holzmann. The model checker spin. *IEEE transaction on software engineering*, 23(5), May 1997.

[14] A. Lomuscio and W. Penczek. Bounded model checking for interpreted systems. Technical report, Institute of Computer Science of the Polish Academy of Sciences, 2002.

[15] A. Lomuscio, F. Raimondi, and M. Sergot. Towards model checking interpreted systems. In *Proceedings of MoChArt*, Lyon, France, August 2002.

[16] K. McMillan. *Symbolic model checking: An approach to the state explosion problem*. Kluwer Academic Publishers, 1993.

[17] R. van der Meyden and N.V. Shilov. Model checking knowledge and time in systems with perfect recall. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 19, 1999.

[18] R. van der Meyden and Kaile Su. Symbolic model checking the knowledge of the dining cryptographers. Submitted, 2002.

[19] Freek Stulp and Rineke Verbrugge. A knowledge-based algorithm for the internet transmission control protocol (TCP) (extended version). *Bulletin of Economic Research*, 54(1):69–94, January 2002. Blackwell Publishers Ltd, Oxford, UK and Boston, USA.

[20] M. Y. Vardi. An automata-theoretic approach to protocol verification (abstract). In *International Conference on Concurrency (CONCURRENCY '88)*, pages 73–73, Berlin - Heidelberg - New York, October 1988. Springer.

[21] M. Wooldridge. Computationally grounded theories of agency. In E. Durfee, editor, *Proceedings of ICMAS, International Conference of Multi-Agent Systems*. IEEE Press, 2000.

[22] Michael Wooldridge, Michael Fisher, Marc-Philippe Huget, and Simon Parsons. Model checking multi-agent systems with MABLE. In Maria Gini, Toru Ishida, Cristiano Castelfranchi, and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 952–959. ACM Press, July 2002.