Towards model checking interpreted systems

A. Lomuscio, F. Raimondi and M. Sergot
Department of Computing
Imperial College
London SW7 2BZ, UK
{alessio,frrai,mjs}@doc.ic.ac.uk

Abstract.
We show how it is possible to pair the NuSMV model checker with Akka, a software platform used to check validity of propositional modal formulas, to verify properties of multi-agent systems formalised on interpreted systems semantics. We demonstrate this by analysing three variants of the bit-transmission problem.

1 INTRODUCTION

Artificial Intelligence and Model Checking are areas of research with traditionally little or no interaction with one another. Artificial Intelligence is concerned with the study and implementation of systems exhibiting complex, human-like behaviour, while Model Checking studies the development and practice of formal techniques leading to the automatic verification of properties of software.

It is only recently, partly thanks to the emphasis on distributed multi-agent systems, that the AI arena has become more interested in the issue of formally verifying properties of a computational system. Indeed, over the past 15 years a shift seems to have taken place among researchers interested in Logic for AI from advocating the use of formal logic as a *computational mechanism*—as it is used notably in logic programming—to the idea of using mathematical logic as a formal language *to specify and reason about* complex distributed systems, perhaps intelligent agents. This shift of objectives has been matched by a slow but persistent change in the techniques that have been developed. One of the areas in which this shift is perhaps most evident is that of multi-agent systems. Much of the formal work in the area has focused on investigations of the suitability of particular logics, often modal logics, to the task of modelling complex attributes of intelligent agents, such as their knowledge, beliefs, desires, and their temporal evolution. The rationale is to use such logics as specifications for intelligent agents.

The field of intelligent agents has made considerable progress in the past few years, and applications have recently been deployed in areas such as agent-mediated e-commerce. It is therefore natural that researchers have recently become interested in *verifying* that agents comply with their specifications.

A converse discovery path seems to have occurred in the more formal areas of software engineering, where AI concepts—such as those of defaults and non-monotonicity—have increasingly been receiving attention. We do not discuss this here; rather, we report on possible uses in AI of model checking technology.

The rest of the paper is organised as follows. In the next section, we motivate and introduce the idea of model checking interpreted systems. In Section 3 we present the formal apparatus that we shall be using throughout the paper. Section 4 presents the main results in this line of investigation. We conclude in Section 5 by evaluating the results and comparing our work to recent developments.

2 MODEL CHECKING INTERPRETED SYSTEMS

Researchers involved in the formalisation of intelligent agents have investigated temporal extensions of particular modal logics used to model intensional notions such as knowledge or belief. Several modal logics have been proposed to deal with these concepts, and properties such as completeness, decidability, and computational complexity have been explored [4]. In this line of work the question of verifying whether a system complies with a specification is expressed by the formula:

$$\Gamma \vdash \varphi$$

where Γ is a logical theory or a set of formulas that represent the specification in the language of the logic, φ is a logical formula encoding the property that one would want to check, and $\vdash$ is the derivability relation of the logic.

The above is a syntactical, or proof-theoretical, check. If semantical considerations apply [16], the check above can be re-phrased in semantical terms by evaluating the validity of the formula:

$$\mathcal{M} \models \varphi$$

where $\mathcal{M}$ is a class of models representing the specifications, φ a logical formula representing the property to be checked, and $\models$ the symbol of modal validity on a class of models. If the formalism under investigation enjoys strong completeness the two properties are equivalent. This allows for the possibility of using different tools (theorem provers, or semantical checkers, etc) to check the validity of the relations above in an automatic way.

This paradigm is certainly sound, but if we intend to use it to analyse concrete scenarios, we may find that it offers too coarse a level of detail, even in principle. Indeed, in the traditional approach, there is little way of expressing that a particular functioning protocol of a multi-agent system, or a program, *generates* a semantical class $\mathcal{M}$, or that it can be represented in a complete way by a set of formulas Γ. On the contrary, in the literature either $\mathcal{M}$ or Γ are taken as the starting points in the analysis. But by doing so we fail to represent the protocols these agents are running, the actions they perform, etc.

One way to address the problem of representing both levels—the protocols, or programs, and the semantical structures that these generate— is through the techniques of model checking. While this has been done successfully in a wide range of cases, it is not a truly satisfying solution for the AI researcher. The problem is that the language of temporal logic is too weak to represent typical AI properties, such as knowledge, beliefs, desires, goals, etc. On the one hand we have fully-fledged model checkers only able to represent a temporal language; on the other, we have complex and refined multi-modal logics to represent intensional aspects of agency that lack verification tools. It seems to us that any attempt of integration of these two areas is worth pursuing. This observation is not new by any means: research along these lines has already appeared for instance in [1, 7, 14, 13].

In this paper we attempt to contribute to this line of research by trying to bring together interpreted systems and model checking on a concrete and well-defined scenario—a variant of the bit transmission problem.

The formalism of interpreted systems [4] is powerful enough to be able to handle, at least in principle, both the temporal epistemic evolution of a system by means of a modal language, and a low level description of the protocols the agents are running. Interpreted systems have been successfully explored by Fagin, Halpern, Moses, and Vardi and colleagues in more than 15 years of work. Crucially, in this line of research not only has the temporal evolution of epistemic and doxastic properties been explored, but, more importantly, complete axiomatisations of these have been proven to correspond to concrete classes of MAS—such as synchronous systems, asynchronous systems [5], perfect recall [6], broadcasting [9], etc.

In this paper we are only concerned with checking epistemic and deontic properties of MAS. In particular, as is known, for a rather large class of problems a static analysis is sufficient to represent key properties of the system. We shall see later that this is the case for example in a variation of the bit transmission problem. In this case, even though we do not use the full power of model checking, it can still be used to help prove epistemic properties of a formalisation. The way we tackle the problem in this paper is as follows (see Figure 1).

1. We study our scenario formally using the formalism of deontic interpreted systems, an extension of interpreted systems designed to represent correct functioning behaviour of the agents as well as epistemic states.
2. We code this representation in NuSMV and feed it to a NuSMV checker [3].
3. We use the NuSMV checker to produce the set of runs of the system, and deduce from there the set of reachable states of the system (the idea that it suffices to consider the set of reachable states when model checking a finite state system has already been made in [15]).
4. We feed these into the modal checker Akka (after some simple cosmetic operations on the format of the states)
5. We use the Akka front-end to check the epistemic properties of the scenario.

The scenario examined here was investigated (together with a more complex variation) in [12] without model checking techniques.

3 PRELIMINARIES

We present here the main definitions for the notation we are going to use in this paper, as from [4, 12]. Due to space consideration we are forced to assume working knowledge with some of the technical machinery presented there.

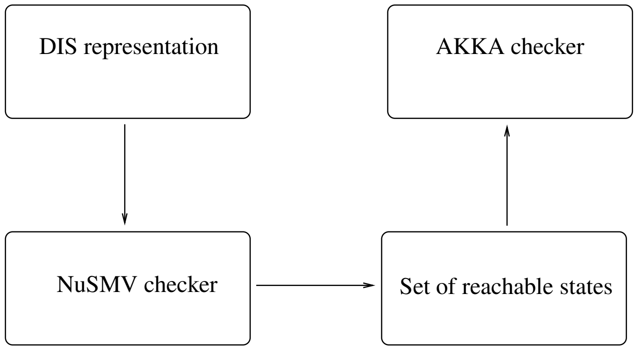

Figure 1. The methodology employed.

3.1 Interpreted Systems

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 $L_1 \times \cdots \times L_n \times L_E$. 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.

An *interpreted* system of global states is a pair $IS = (S, h)$ where S is a system of global states and $h : S \to 2^P$ is an interpretation function for a set of propositional variables P.

Systems of global states can be used to interpret epistemic modalities K_i, one for each agent.

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

Alternatively one can consider generated models $(S, \sim_1, \ldots, \sim_n, h)$ of the standard (Kripke) form, where the equivalence relations $\sim_i$ are defined on equivalence of local states, and then interpret modalities in the standard modal tradition (e.g. [2, 8]). The resulting logic for modalities K_i is $S5_n$; this models agents with complete introspection capabilities and veridical knowledge.

3.2 Deontic interpreted systems

The notion of interpreted systems can be extended to incorporate the idea of correct functioning behaviour of some or all of the components [11].

Given n agents and $n+1$ non-empty sets $G_E, G_1, \ldots, G_n$, a *deontic system of global states* is any system of global states defined on $L_E \supseteq G_E, \ldots, L_n \supseteq G_n$. G_E is called the *set of green states for the environment*, and for any agent i, G_i is called *the set of green states for agent i*. The complement of G_E with respect to L_E (respectively G_i with respect to L_i) is called the *set of red states for the environment (respectively for agent i)*.

The terms ‘green’ and ‘red’ are chosen as neutral terms, to avoid overloading them with unintended readings and connotations. The term ‘green’ can be read as ‘legal’, ‘acceptable’, ‘desirable’, ‘correct’, depending on the context of a given application.

Deontic systems of global states are used to interpret modalities such as the following

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

$O_i\, \varphi$ is used to represent that φ holds in all (global) states in which agent i is functioning correctly. Again, one can consider generated models $(S, \sim_1, \ldots, \sim_n, R_1^O, \ldots, R_n^O, h)$, where the equivalence relations are defined as above and the relations R_i^O are defined by $g\, R_i^O\, g'$ if $l_i(g') \in G_i$, with a standard modal logic interpretation for the operators O_i.

Knowledge can be modelled on deontic interpreted systems in the same way as on interpreted systems, and one can study various combinations of the modalities such as $K_i\, O_j$, $O_j\, K_i$, and others. Another concept of particular interest is knowledge that an agent i has *on the assumption that the system (the environment, agent j, group of agents X) is functioning correctly*. We employ the (doubly relativised) modal operator $\widehat{K}_i^j$ for this notion, interpreted as follows:

$$(IS, g) \models \widehat{K}_i^j\, \varphi \quad \text{if for all } g' \text{ such that } l_i(g) = l_i(g') \text{ and } l_j(g') \in G_j \text{ we have that } (IS, g') \models \varphi.$$

Uses of this modal operator will be illustrated in the examples to follow.

3.3 Protocols

Interpreted systems can be extended to deal with temporal evolution: consider a set of runs over global states $R = \{r : N \to S\}$, where a run r is defined as a function from time to global states and time ranges over natural numbers [4].

We assume that for every agent of the system and for the environment there is a set Act_i and Act_E of actions. For example, in a message passing system, `send(x,i,j)` could mean that agent i sends a message x to agent j; this action would then be in Act_i.

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, considering *sets of actions*, we allow nondeterminism in the protocol):

$$P_i : L_i \to 2^{Act_i}$$

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

$$\pi : S \times Act \to S$$

where $S = L_1 \times \cdots \times L_n \times L_E$ and $Act = Act_1 \times \cdots \times Act_n \times Act_E$.

4 THE BIT TRANSMISSION PROBLEM

In the rest of the paper we test the methodology of Figure 1 on three variants of the bit transmission problem.

4.1 The basic version

The bit-transmission problem [4] involves two agents, a *sender* S, and a *receiver* R, communicating over a faulty communication channel. The channel may drop messages but will not flip the value of a bit being sent. S wants to communicate some information—the value of a bit for the sake of the example—to R. We would like to design a protocol that accomplishes this objective while minimising the use of the communication channel.

One protocol for achieving this is as follows. S immediately starts sending the bit to R, and continues to do so until it receives an acknowledgement from R. R does nothing until it receives the bit; from then on it sends acknowledgements of receipt to S. S stops sending the bit to R when it receives an acknowledgement. Note that R will continue sending acknowledgments even after S has received its acknowledgement. Intuitively S will know for sure that the bit has been received by R when it gets an acknowledgement from R. R, on the other hand, will never be able to know whether its acknowledgement has been received since S does not answer the acknowledgement.

We assume *fairness* ([4], p.164) for the communication channel: every message that is repeatedly sent in the run is eventually delivered.

What we would like to do is to check mechanically that the protocol above guarantees that when sender receives the acknowledgement it then knows (in the information-theoretic sense defined in Section 3.1) that the receiver knows the value of the bit. In order to do this, first we model the scenario in the interpreted systems paradigm.

4.1.1 *Interpreted Systems analysis*

There are three active components in the scenario: a sender, a receiver, and a communication channel. In line with the spirit of the formalism of interpreted systems, it is convenient to see sender and receiver as agents, and the communication channel as the environment. Each of these can be modelled by considering their local states. For the sender S, it is enough to consider four possible local states. They represent the value of the bit S is attempting to transmit, and whether or not S has received an acknowledgement from R. Three different local states are enough to capture the state of R: the value of the received bit, and ϵ representing a circumstance under which no bit has been received yet. So we have

$$L_S = \{0, 1, (0, ack), (1, ack)\}, \quad L_R = \{0, 1, \epsilon\}.$$

To model the environment we consider four different local states, representing the possible combinations of messages that have been sent in the current round, by S and R, respectively. The four local states are:

$$L_E = \{(.,.), (sendbit, .), (., sendack), (sendbit, sendack)\},$$

where '.' represents configurations in which no message has been sent by the corresponding agent. These four local states are not strictly necessary for the examples analysed in this paper but they are used in other variants and so we retain them.

Global states for the system G are defined as $G \subseteq L_S \times L_R \times L_E$. A global state $g = (l_S, l_R, l_E)$ gives a snapshot of the system at a given time. Note that not all triples of the product are admissible in principle, but only those that can be reached in a run of the protocol, as will be explained below.

Consider then a set of actions Act_i for every agent in the system and the environment.

$$Act_S = \{sendbit(0), sendbit(1), \lambda\}, \quad Act_R = \{sendack, \lambda\}.$$

Here λ stands for no action ('no-op').

The actions Act_E for the environment correspond to the transmission of messages between S and R on the unreliable communication channel. To make the example sufficiently rich, we will assume that the communication channel can transmit messages in both directions simultaneously, and that a message travelling in one direction can get through while a message travelling in the opposite direction is lost.

(Alternatively, think of a pair of unidirectional communication channels whose faults are independent of one other.) The set of actions for the environment is

$$Act_E = \{S{-}R,\ S{\rightarrow},\ {\leftarrow}R,\ -\}$$

$S{-}R$ represents the action in which the channel transmits any message successfully in both directions, $S{\rightarrow}$ that it transmits successfully from S to R but loses any message from R to S, ${\leftarrow}R$ that it transmits successfully from R to S but loses any message from S to R, and $-$ that it loses any messages sent in either direction.

We can model the evolution of the system by means of a transition function $\pi : G \times Act \to G$, where $Act = Act_S \times Act_R \times Act_E$ is the set of joint actions for the system. π codes the fact that the state of the communication channel determines whether the actions performed by the agents (i.e., the messages they send on the channel) get through or not, and what their effects are. For example, the definition of π contains the following:

$$\begin{aligned}
\pi(\,(0,\epsilon,(.,.)),\ (sendbit(0),\lambda,S{-}R)\,) &= (0,0,(sendbit(0),\lambda)),\\
\pi(\,(0,\epsilon,(.,.)),\ (sendbit(0),\lambda,S{\rightarrow})\,) &= (0,0,(sendbit(0),\lambda)),\\
\pi(\,(0,\epsilon,(.,.)),\ (sendbit(0),\lambda,{\leftarrow}R)\,) &= (0,\epsilon,(sendbit(0),\lambda)),\\
\pi(\,(0,\epsilon,(.,.)),\ (sendbit(0),\lambda,-)\,) &= (0,\epsilon,(sendbit(0),\lambda)),
\end{aligned}$$

to capture that when the channel works properly the message $sendbit(0)$ from S gets through and gets processed accordingly by R.

Other cases can be similarly expressed. (They can be reconstructed from the NuSMV code given later.)

For compliance with a given protocol, only certain actions are performable at a given time for an agent. For example if S has not yet received an acknowledgement from R and is in the local state 0, then according to the simple protocol under consideration, S should perform the action $sendbit(0)$. A *protocol* for agent i is a function $P_i : L_i \to 2^{Act_i}$ mapping sets of actions from a local state. $P_i(l_i)$ is the set of actions performable according to the protocol by agent i when its local state is l_i. For the example under consideration the protocol can be defined as follows:

$$\begin{aligned}
&P_S(0) = sendbit(0), \quad P_S(1) = sendbit(1),\\
&P_S((0,ack)) = P_S((1,ack)) = \lambda,\\
&P_R(\epsilon) = \lambda, \quad P_R(0) = P_R(1) = sendack.
\end{aligned}$$

We omit brackets when writing singleton sets.

For the environment, we use the constant function:

$$P_E(l_E) = Act_E = \{S{-}R,\ S{\rightarrow},\ {\leftarrow}R,\ -\}, \quad \text{for all } l_E \in L_E$$

We assume that the system starts from a state $g_0 = (0,\epsilon,(.,.))$ or $g_0 = (1,\epsilon,(.,.))$. We are interested in the set of global states reachable from these initial configurations as defined by the transition function π and the protocol functions P_S, P_R and P_E.

4.1.2 Implementation

Given the description as above we can implement the scenario in NuSMV by representing the local states as NuSMV variables and translating the protocol functions and system evolution function π into the syntax of NuSMV. For example, the local states for the sender S are coded in NuSMV as:

```
LS       : {LS0, LS1, LS2, LS3};
-- Local states for the sender:
--            LS0 = 0
--            LS1 = 1
--            LS2 = (0,ack)
--            LS3 = (1,ack)
```

In a similar way we can code the local states for R and for the environment, and specify their actions. The NuSMV code, together with code for assigning values to the initial states, is shown in Figure 2. This part of the code is the same for all versions of the bit transmission problem discussed in this paper.

```
MODULE main
VAR

  LS       : {LS0, LS1, LS2, LS3};
-- Local states for the sender:
--            LS0 = 0
--            LS1 = 1
--            LS2 = (0,ack)
--            LS3 = (1,ack)

  LR       : {LR0, LR1, LR2};
-- Local states for the receiver:
--            LR0 = 0
--            LR1 = 1
--            LR2 = e [nothing received]
--            LR3 = (0,f)
--            LR4 = (1,f)
--            LR5 = (e,f)

  LE       : {LE0, LE1, LE2, LE3};
-- Local states for the environment:
--            LE0 = (.,.)
--            LE1 = (sendbit,.)
--            LE2 = (.,sendack)
--            LE3 = (sendbit,sendack)

  ActE     : {s->,<-r,s-r,x-x};
-- Actions of the environment:
--            s-> = transmit: receiver receives;
--            <-r = transmit: sender receives;
--            s-r = transmit: both receive;
--            x-x = transmit: lose both;

  ActS     : {sb0,sb1,snull};
-- Actions for the Sender:
--            sb0 = send bit 0
--            sb1 = send bit 1
--            snull  = do nothing

  ActR     : {rnull,sendack};
-- Actions for the Receiver:
--            rnull = do nothing
--            sendack = obvious


ASSIGN
-- Initial local states
  init(LS)  := {LS0,LS1};
  init(LR)  := LR2;
  init(LE)  := LE0;
  init(ActE)  := {s->,<-r,s-r,x-x};
  init(ActS)  := snull;
  init(ActR)  := {rnull};
```

Figure 2. NuSMV code for the bit transmission problem: local states and actions.

The protocol functions and the evolution function (called `next` in the code) are coded in terms of local states and actions. For example,

```
next(LS) := case
   ( LS = LS0 ) & ( ActR = sendack )
    & ( ActE = <-r | ActE = s-r ) : LS2;
   ( LS = LS1 ) & ( ActR = sendack )
    & ( ActE = <-r | ActE = s-r ) : LS3;
   1 : LS;
esac;
```

The rest of the code for the basic version of the bit transmission problem is shown in Figure 3.

We are now able to use NuSMV to check various temporal properties of this system in the standard manner. However, it is the analysis of static epistemic properties of the system that is the focus of attention in this paper, and for that we pass output from NuSMV to a different model checker, Akka.

```
--     MODULE main continued (basic version)

-- Here we implement the protocol functions

  next(ActE) := {s->,<-r,s-r,x-x};

  next(ActS) := case
     ( LS = LS0 )   : sb0;
     ( LS = LS1 )   : sb1;
     ( LS = LS2 ) | ( LS = LS3) : snull;
     1 : ActS;
  esac;

  next(ActR) := case
     ( LR = LR0 ) | ( LR = LR1 ) : {sendack};
     ( LR = LR2 )                : {rnull};
  esac;

-- Here we compute the next local state, as a
-- function of actions and local states

  next(LS) := case
     ( LS = LS0 ) & ( ActR = sendack ) &
              ( ActE = <-r | ActE = s-r ) : LS2;
     ( LS = LS1 ) & ( ActR = sendack ) &
              ( ActE = <-r | ActE = s-r ) : LS3;
     1 : LS;
  esac;

  next(LR) := case
     ( ActS = sb0 ) & ( LR = LR2 ) &
              ( ActE = s-> | ActE = s-r ) : LR0;
     ( ActS = sb1 ) & ( LR = LR2 ) &
              ( ActE = s-> | ActE = s-r ) : LR1;
     1 : LR;
  esac;

  next(LE) := case
     ( ActS = sb0 | ActS = sb1 ) &
                    ( ActR = rnull ) : LE1;
     ( ActS = sb0 | ActS = sb1 ) &
                  ( ActR = sendack ) : LE3;
     ( ActS = snull ) & ( ActR = sendack ) : LE2;
     1 : LE;
  esac;

FAIRNESS
  AF(ActE != x-x ) & running
```

Figure 3. NuSMV code for the basic version of the bit transmission problem.

4.1.3 *Interface between NuSMV and Akka*

The key properties we would like to verify are properties involving epistemic modalities, whose interpretation depends on the internal structure of the global states.

We modified the NuSMV code to generate the reachable global states of the system. The output is a file with the list of all the reachable states:

```
     ⋮
-----------
LS3,LR5,LE3
-----------
LS3,LR5,LE1
-----------
LS3,LR5,LE2
-----------
     ⋮
```

We parse this file by means of a simple C++ program, producing as output a Kripke model representing the epistemic relations generated by the scenario. The format of the output file is tailored for Akka, a software package that allows checking validity in a model, as discussed below.

4.1.4 *Verification*

Akka[1] offers a Kripke Model Editor and supports model testing. Figure 4 shows a screen-shot of the model obtained as depicted by Akka. The nodes represent the global states of the model and the arcs represent the epistemic (equivalence) relations between the states.

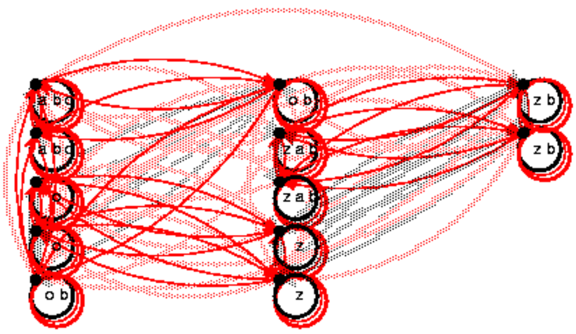

Figure 4. Screenshot in Akka of the model IS_{b} for the basic version of the bit transmission problem.

We are now in a position to check any epistemic property of the system that can be written as a modal formula of arbitrary complexity (as is known, the computational complexity of the check is linear in the size of the formula, and of the number of states of the model). To this end, let us name IS_{b} the model obtained by following the process described above, on which an appropriate set of propositional variables is interpreted in a natural way[2].

---

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

[2] We assume the following:

$$
\begin{array}{lll}
(IS_{\mathrm{b}}, g) \models \mathbf{bit = 0} & \text{if} & l_S(g) = 0, \text{ or } l_S(g) = (0, ack) \\
(IS_{\mathrm{b}}, g) \models \mathbf{bit = 1} & \text{if} & l_S(g) = 1, \text{ or } l_S(g) = (1, ack) \\
(IS_{\mathrm{b}}, g) \models \mathbf{recbit} & \text{if} & l_R(g) = 1, \text{ or } l_R(g) = 0 \\
(IS_{\mathrm{b}}, g) \models \mathbf{recack} & \text{if} & l_S(g) = (1, ack), \text{ or } l_S(g) = (0, ack).
\end{array}
$$

For the example under consideration we shall want to check whether, when the sender S receives an acknowledgement message, S knows that the receiver R knows the value of the bit.

$$IS_{\mathrm{b}} \models \mathbf{recack} \rightarrow K_S(K_R((\mathbf{bit} = \mathbf{0}) \vee (\mathbf{bit} = \mathbf{1}))$$

This proposition is easily translated into the syntax of Akka and we can check it against the output of the parser. As expected, the proposition is valid on IS_{b}.

In the same way, the following can be checked and confirmed to be valid in the basic version of the problem:

$$\begin{aligned} &IS_{\mathrm{b}} \models \mathbf{recbit} \rightarrow \big(K_R\,(\mathbf{bit} = \mathbf{0}) \vee K_R\,(\mathbf{bit} = \mathbf{1})\big) \\ &IS_{\mathrm{b}} \models \mathbf{recack} \rightarrow K_S\big(K_R\,(\mathbf{bit} = \mathbf{0}) \vee K_R\,(\mathbf{bit} = \mathbf{1})\big) \\ &IS_{\mathrm{b}} \models \mathbf{recack} \wedge (\mathbf{bit} = \mathbf{0}) \rightarrow K_S\,K_R\,(\mathbf{bit} = \mathbf{0}) \end{aligned}$$

4.2 Faulty Receiver —1

An interesting scenario arises when we assume that the agents may not behave as they are supposed to. For example, the receiver may not send an acknowledgement message when it receives a bit ([12]). We deal with this case by considering a new protocol which extends the original one.

4.2.1 *Interpreted Systems Analysis*

We introduce two new local states for the receiver R. The local states for the receiver R are now:

$$L'_R = \{0, 1, \epsilon, (0, f), (1, f)\}$$

The states (i, f) $(i = \{0, 1\})$ are *faulty* states in which R received a bit but failed to send an acknowledgement.

We consequently modify the protocol for the receiver R:

$$P'_R(0) = P'_R(1) = \{sendack, \lambda\}$$

by extending the definition to cover also the faulty local states $(0, f)$ and $(1, f)$:

$$P'_R((0, f)) = P'_R((1, f)) = \{sendack, \lambda\}$$

If the receiver enters one of its faulty states, it can nevertheless recover to a non-faulty state [11], by sending an acknowledgement this time round.

For this version of the problem, we want the system evolution function π' to be the same as in the basic version for actions conforming to protocols in non-faulty states. For the other cases, π' needs to cover the conditions under which we move to a faulty local state for agent R, and then the outcome of transitions originating from faulty local states for agent R. For example, the definition of π' contains, for all $\gamma_E \in Act_E$, unless stated otherwise (the definitions for $\mathbf{bit} = \mathbf{1}$ are analogous):

$$\begin{aligned} &\pi'(\,(0, 0, (.,.)), (sendbit(0), \lambda, \gamma_E)\,) = \\ &\qquad\qquad (0, (0, f), (sendbit(0), \lambda)) \\ &\pi'(\,((0, ack), 0, (.,.)), (\lambda, \lambda, \gamma_E)\,) = \\ &\qquad\qquad ((0, ack), (0, f), (\lambda, \lambda)) \\ &\pi'(\,(0, (0, f), (.,.)), (sendbit(0), \lambda, \gamma_E)\,) = \\ &\qquad\qquad (0, (0, f), (sendbit(0), \lambda)) \\ &\pi'(\,(0, (0, f), (.,.)), (sendbit(0), sendack, \gamma_E)\,) = \\ &\qquad ((0, ack), 0, (sendbit(0), sendack)) \qquad (\gamma_E \in \{S{-}R, {\leftarrow}R\}) \\ &\pi'(\,(0, (0, f), (.,.)), (sendbit(0), sendack, \gamma_E)\,) = \\ &\qquad (0, 0, (sendbit(0), sendack)) \qquad (\gamma_E \in \{S{\rightarrow}, -\}) \end{aligned}$$

Note that in the last two cases we have chosen to say that sending an acknowledgement in a faulty local state puts R back into a fault-free local state—the record of the protocol-violating fault is wiped out. Others cases for the definition of π' are similarly expressed (cf. the NuSMV code in Figure 5).

4.2.2 *Implementation*

The NuSMV implementation of this version of the bit transmission problem is shown in Figure 5. It is a straightforward extension of the code for the basic version presented above. We need to introduce a fairness constraint besides the one used in the basic version. Declaration of local states and actions, and initial values, are exactly as for the basic case and so are omitted. See Figure 2.

4.2.3 *Verification*

As in the previous case, we use (the modified version of) NuSMV to generate the reachable global states, and then parse the output to build a representation of the global states and epistemic relations for input to Akka. Akka's depiction of the model IS'_{b} obtained is shown in Figure 6.

It is easy to check that the epistemic properties that held in the basic version also hold in this case[3]. Namely:

$$\begin{aligned} &IS'_{\mathrm{b}} \models \mathbf{recbit} \rightarrow \big(K_R\,(\mathbf{bit} = \mathbf{0}) \vee K_R\,(\mathbf{bit} = \mathbf{1})\big) \\ &IS'_{\mathrm{b}} \models \mathbf{recack} \rightarrow K_S\big(K_R\,(\mathbf{bit} = \mathbf{0}) \vee K_R\,(\mathbf{bit} = \mathbf{1})\big) \\ &IS'_{\mathrm{b}} \models \mathbf{recack} \rightarrow K_S(K_R((\mathbf{bit} = \mathbf{0}) \vee (\mathbf{bit} = \mathbf{1})) \\ &IS'_{\mathrm{b}} \models \mathbf{recack} \wedge (\mathbf{bit} = \mathbf{0}) \rightarrow K_S\,K_R\,(\mathbf{bit} = \mathbf{0}) \end{aligned}$$

4.3 Faulty Receiver —2

We now consider a slightly more complicated scenario. We assume that the receiver R may send acknowledgements even when it is not supposed to.

4.3.1 *Deontic Interpreted Systems Analysis*

For this version of the problem we introduce a new local state for the receiver R, namely (ϵ, f). This is the local state in which R did not receive a bit but nevertheless R sent an acknowledgement.

For the analysis of this scenario we use the deontic machinery presented in Section 3.2, as well as the epistemic approach of the previous section. For S, since we are not admitting (for the purposes

For example $\mathbf{bit} = \mathbf{0}$ is true at a global state of the model if the local state of the sender is either 0, or $(0, ack)$.

[3] The interpretation of the variables is similar and not repeated here.

```
--     MODULE main continued (receiver may fail to ack)

-- Here we implement the protocol functions

  next(ActE) := {s->,<-r,s-r,x-x};

  next(ActS) := case
    ( LS = LS0 )   : sb0;
    ( LS = LS1 )   : sb1;
    ( LS = LS2 ) | ( LS = LS3) : snull;
    1 : ActS;
  esac;

  next(ActR) := case
    ( LR = LR0 ) | ( LR = LR1 ) | ( LR = LR3 )
                 | ( LR = LR4 )   : {sendack, rnull};
    ( LR = LR2 )                  : {rnull};
  esac;

-- Here we compute the next local state, as a
-- function of actions and local states

  next(LS) := case
    ( LS = LS0 ) & ( ActR = sendack ) &
                            ( ActE = <-r ) : LS2;
    ( LS = LS1 ) & ( ActR = sendack ) &
                            ( ActE = <-r ) : LS3;
    1 : LS;
  esac;

  next(LR) := case
    ( ActS = sb0 ) & ( LR = LR2 ) & ( ActR = sendack )
                   & ( ActE = s-> | ActE = s-r )  : LR0;
    ( ActS = sb1 ) & ( LR = LR2 ) & ( ActR = sendack )
                   & ( ActE = s-> | ActE = s-r )  : LR1;
    ( ActS = sb0 ) & ( LR = LR2 ) & ( ActR = rnull )
                   & ( ActE = s-> | ActE = s-r )  : LR3;
    ( ActS = sb1 ) & ( LR = LR2 ) & ( ActR = rnull )
                   & ( ActE = s-> | ActE = s-r )  : LR4;
    ( LR = LR0 ) & ( ActR = rnull )                : LR3;
    ( LR = LR1 ) & ( ActR = rnull )                : LR4;
    ( LR = LR3 ) & ( ActR = sendack )              : LR0;
    ( LR = LR4 ) & ( ActR = sendack )              : LR1;
    1 : LR;
  esac;

  next(LE) := case
    ( ActS = sb0 | ActS = sb1 ) & ( ActR = rnull ) : LE1;
    ( ActS = sb0 | ActS = sb1 ) &
                            ( ActR = sendack ) : LE3;
    ( ActS = snull ) & ( ActR = sendack )          : LE2;
    1 : LE;
  esac;

FAIRNESS
  AF(ActE != x-x ) &
   (((LR=LR0)|(LR=LR1))->AF(ActR=sendack)) &
   running
```

Figure 5. NuSMV code for the case where the receiver may fail to send acknowledgements.

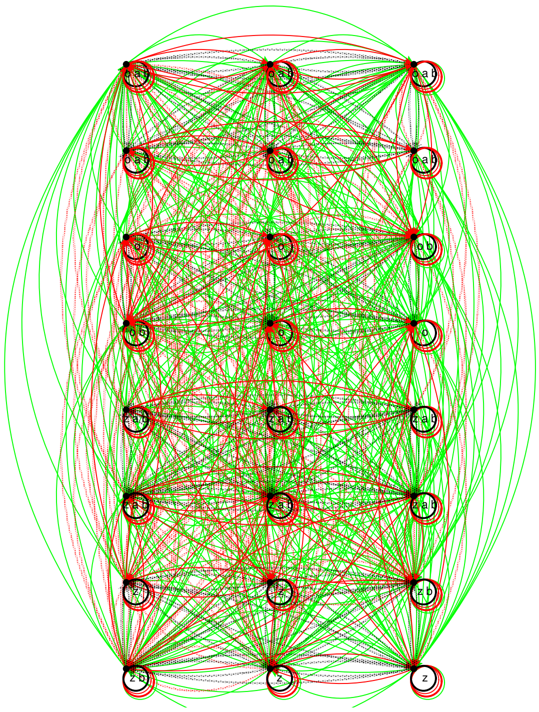

Figure 6. Screenshot in Akka of the model IS'_{b} of the bit transmission problem where receiver may fail to send acknowledgements.

of the example) the possibility of faults, its local states are all green. We thus have:

$$L''_S = G''_S = \{0, 1, (0, ack), (1, ack)\}, \quad R''_S = \emptyset.$$

For the case of the environment, we have admitted the possibility of faulty, or unreliable, behaviour but these 'faults' are not violations of the protocol under examination. Accordingly, all local states of the environment are also green; $R''_E = \emptyset$, $L''_E = G''_E$, and we have:

$$G''_E = \{(.,.), (sendbit, .), (., sendack), (sendbit, sendack)\}.$$

For R, we define the local states as follows:

$$G''_R = \{0, 1, \epsilon\},\ R''_R = \{(0, f), (1, f), (\epsilon, f)\},\ L''_R = G''_R \cup R''_R.$$

Now we define the protocol functions of this deontic interpreted system. Given that the two sets of local states for S and E have not changed we can keep the functions P_S and P_E as for the basic version. We need to extend P_R so that it is defined also on the red local states of R.

$$\begin{aligned} &P''_R(\epsilon) = P_R(\epsilon) = \lambda, \\ &P''_R(0) = P''_R(1) = P_R(0) = P_R(1) = sendack \end{aligned}$$

For the red local states $R''_R = \{(0, f), (1, f), (\epsilon, f)\}$ we shall define

$$P''_R((0, f)) = P''_R((1, f)) = P''_R((\epsilon, f)) = Act_R = \{sendack, \lambda\}$$

For the computation of reachable global states for this version, it remains to define the evolution function π''. Essentially we want to extend the definition of π by insisting that R's local states will be red if R has sent an acknowledgement, either in the current round or in the past, without having received the bit first, and otherwise copy R's transitions in π, i.e., R will correctly store the bit if it has received it and remain in the state ϵ otherwise. First, we specify the effects of protocol-violating actions in green R states. For the case where the bit is 0 (the other can be done similarly) we shall impose:

$$\begin{aligned} &\pi''(\,(0, \epsilon, (.,.)), (sendbit(0), sendack, S{-}R)\,) = \\ &\qquad (\,(0, ack), (0, f), (sendbit(0), sendack)\,) \\ &\pi''(\,(0, \epsilon, (.,.)), (sendbit(0), sendack, S{\rightarrow})\,) = \\ &\qquad (\,0, (0, f), (sendbit(0), sendack)\,) \\ &\pi''(\,(0, \epsilon, (.,.)), (sendbit(0), sendack, {\leftarrow}R)\,) = \\ &\qquad (\,(0, ack), (\epsilon, f), (sendbit(0), sendack)\,) \\ &\pi''(\,(0, \epsilon, (.,.)), (sendbit(0), sendack, -)\,) = \\ &\qquad (\,0, (\epsilon, f), (sendbit(0), sendack)\,) \end{aligned}$$

Note that in the first case above the result state is a faulty (red) state even though communication has taken place, and that in the second and fourth cases the result state is a faulty (red) state even though the erroneous acknowledgement was not received by S.

Now we extend the definition of π so that π'' is defined also on red local states for R. Once R is in a red state we will impose that it will remain in a red state, although it will correctly store messages, if received.

$$\begin{aligned} &\pi''(\,(0, (\epsilon, f)), (.,.)), (sendbit(0), sendack, S{-}R)\,) = \\ &\qquad (\,(0, ack), (0, f), (sendbit(0), sendack)\,) \\ &\pi''(\,(0, (\epsilon, f)), (.,.)), (sendbit(0), sendack, S{\rightarrow})\,) = \\ &\qquad (\,0, (0, f), (sendbit(0), sendack)\,) \\ &\pi''(\,(0, (\epsilon, f)), (.,.)), (sendbit(0), sendack, {\leftarrow}R)\,) = \\ &\qquad (\,(0, ack), (\epsilon, f), (sendbit(0), sendack)\,) \\ &\pi''(\,(0, (\epsilon, f)), (.,.)), (sendbit(0), sendack, -)\,) = \\ &\qquad (\,0, (\epsilon, f), (sendbit(0), sendack)\,) \end{aligned}$$

We omit the other cases. They are straightforwardly expressed and can be reconstructed from the NuSMV code shown for this variant of the example in Figure 5.

4.3.2 Implementation

The NuSMV implementation of this version is shown in Figure 7. It is an extension of the basic code of Figure 3; the transition function for the receiver is a bit more complicated, to cover all possible cases of faulty behaviour.

```
--      MODULE main continued (receiver may incorrectly ack)

-- Here we implement the protocol functions

  next(ActE) := {s->,<-r,s-r,x-x};

  next(ActS) := case
    ( LS = LS0 )   : sb0;
    ( LS = LS1 )   : sb1;
    ( LS = LS2 ) | ( LS = LS3) : snull;
    1 : ActS;
  esac;

  next(ActR) := case
    ( LR = LR0 ) | ( LR = LR1 )          : {sendack};
    ( LR = LR2 ) | ( LR = LR3 ) |
          ( LR = LR4 ) | ( LR = LR5 ) : {sendack,rnull};
  esac;

-- Here we compute the next local state, as a
-- function of actions and local states

  next(LS) := case
    ( LS = LS0 ) & ( ActR = sendack ) &
                          ( ActE = <-r ) : LS2;
    ( LS = LS1 ) & ( ActR = sendack ) &
                          ( ActE = <-r ) : LS3;
    1 : LS;
  esac;

  next(LR) := case
    ( ActS = sb0 ) & ( LR = LR2 ) &
                    ( ActE = s-> | ActE = s-r ) : LR0;
    ( ActS = sb1 ) & ( LR = LR2 ) &
                    ( ActE = s-> | ActE = s-r ) : LR1;
    ( LR = LR2 ) & ( ActR = sendack ) &
                   ( ActE != s-> & ActE != s-r) : LR5;
    ( LR = LR2 ) & ( ActR = sendack ) &
      ( ActE = s-> | ActE = s-r ) & ( ActS = sb0 ): LR3;
    ( LR = LR2 ) & ( ActR = sendack ) &
       (ActE = s-> | ActE = s-r) & ( ActS = sb1 ) : LR4;
    ( LR = LR5 ) & ( ActS = sb0 ) &
                      (ActE = s-> | ActE = s-r) : LR3;
    ( LR = LR5 ) & ( ActS = sb1 ) &
                      (ActE = s-> | ActE = s-r) : LR4;
    1 : LR;
  esac;

  next(LE) := case
   ( ActS = sb0 | ActS = sb1 ) & ( ActR = rnull )   : LE1;
   ( ActS = sb0 | ActS = sb1 ) & ( ActR = sendack ): LE3;
   ( ActS = snull ) & ( ActR = sendack )           : LE2;
   1 : LE;
  esac;

FAIRNESS
  AF(ActE != x-x ) & running
```

Figure 7. NuSMV code for the case where receiver may send incorrect acknowledgements.

4.3.3 Verification

As in the previous cases, NuSMV outputs the reachable global states and, using these states, we can create a model with epistemic relations for K_S and K_R. If we consider red and green states for R we can parse the global states and create a model that includes also "deontic" relations for $\widehat{K}^R_S$, the operator introduced in section 3.2. This all-inclusive model, IS''_b, is shown in Figure 8.

First of all, it is possible to check that none of the epistemic formulas presented above hold in this version. However, a particular form of knowledge still holds. Intuitively if S could make the assumption of R's correct functioning behaviour, then, upon receipt of an acknowledgement, it would then make sense for it to assume that R does know the value of the bit; this is exactly the meaning of $\widehat{K}_S^R$. And indeed, using Akka we are able to check the validity[4] of the following formulas in the model:

$$IS''_{\mathrm{b}} \models \mathbf{recack} \rightarrow \widehat{K}_S^R \left(K_R \left(\mathbf{bit} = \mathbf{0} \right) \vee K_R \left(\mathbf{bit} = \mathbf{1} \right) \right)$$

$$IS''_{\mathrm{b}} \models \mathbf{recack} \wedge \left(\mathbf{bit} = \mathbf{0} \right) \rightarrow \widehat{K}_S^R \, K_R \left(\mathbf{bit} = \mathbf{0} \right)$$

We refer to [12] for more details, but note that unlike the usual epistemic operators associated with interpreted systems, $\widehat{K}_i^j$ is not an S5 operator, and in particular the axiom T does not hold for it, i.e., knowledge under assumptions of correct functioning behaviour does not imply truth. This operator is of particular interest in this circumstance because it captures precisely our intuition about the example.

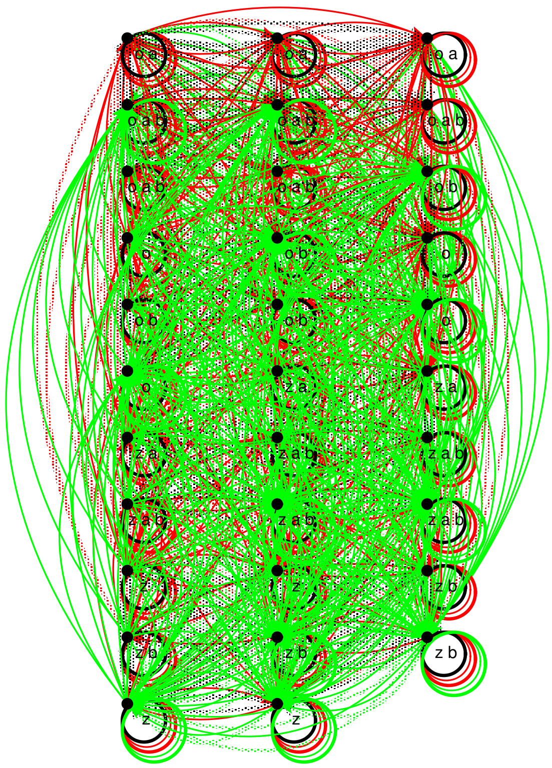

Figure 8. Screenshot in Akka of the model IS''_{b} for the bit transmission problem where receiver may send incorrect acknowledgements.

5 CONCLUSIONS

In this paper we have made a first attempt at combining an established model checking tool such as NuSMV with Akka, a traditional tool used for testing validity of modal formulas.

We see the contribution of this paper as being twofold. Firstly, we provide a simple methodology for checking static epistemic properties in interpreted systems. Although much research has gone into developing epistemic logic in AI, both at proof-theoretical and at semantical level, comparatively less attention has been given to verifying automatically that particular functioning protocols exhibit particular epistemic properties. In other words no method had been developed in which one can implement a functioning protocol and test properties of the resulting execution. We believe that the time is ripe for further combinations of model checking tools with well-explored formalisms in AI.

Secondly, we find the technical results on the violations of the bit-transmission problem interesting on their own merits. The bit-transmission problem is well-studied in the literature and is one of the key examples that show the capabilities of the interpreted systems paradigm. Indeed an in-depth analysis of this scenario was carried out in [7] by means of the Spin model checker. We differ from that paper in two respects. First our methodology of pairings Akka with NuSMV is different from what advocated there. Second, by means of deontic interpreted systems, we analyse variations of the scenario in which violations occur.

We have tried to show that in some examples verifying *static* epistemic and deontic properties is sufficient to establish basic properties of the system. Still, we would like to extend the methodology above to deal with the full *dynamic* case, i.e., to move to a system in which we can check temporal deontic and epistemic formulas. A promising avenue seems to be to implement the Andersonian reductions explored in [10].

[4] The interpretation for the atoms is a straightforward extension of what reported previously and not repeated here.

ACKNOWLEDGEMENTS

The authors are grateful to Marco Pistore of NuSMV development team for providing important directions to modify the source code of NuSMV to deal with the collection of global states from the NuSMV software. We also thank Lex Hendriks for providing us with a new release of Akka.

The authors acknowledge support from EU project ALFEBIITE, IST-1999-20298.

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] B. Chellas, *Modal Logic — An Introduction*, Cambridge University Press, Cambridge, 1980.

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

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

[5] R. Fagin, J. Y. Halpern, and M. Y. Vardi, 'What can machines know? On the properties of knowledge in distributed systems', *Journal of the ACM*, **39**(2), 328–376, (April 1992).

[6] J. Halpern, R. van der Meyden, and M. Y. Vardi, 'Complete axiomatisations for reasoning about knowledge and time'. Submitted, 1997.

[7] 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).

[8] A. Lomuscio, R. van der Meyden, and M. Ryan, 'Knowledge in multiagent systems: Initial configurations and broadcast.', *ACM Transactions of Computational Logic*, **1**(2), (October 2000).

[9] A. Lomuscio and M. Sergot, 'On multi-agent systems specification via deontic logic', in *Proceedings of ATAL 2001*, ed., J.-J Meyer. Springer Verlag, (July 2001). To Appear.

[10] A. Lomuscio and M. Sergot, 'The bit transmission problem revisited', Technical Report 4/2002, Department of Computing, Imperial College, London SW7 2BZ, UK, (2002).

[11] Ron van der Meyden, 'Common knowledge and update in finite environments', *Information and Computation*, **140**(2), 115–157, (1 February 1998).

[12] Ron 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).

[13] Moshe Y. Vardi, 'Implementing knowledge-based programs', in *Theoretical Aspects of Rationality and Knowledge: Proceedings of the Sixth Conference (TARK 1996)*, ed., Yoav Shoham, 15–30, Morgan Kaufmann, San Francisco, (1996).

[14] M. Wooldridge, 'Computationally grounded theories of agency', in *Proceedings of ICMAS, International Conference of Multi-Agent Systems*, ed., E. Durfee, IEEE Press, (2000).