# Verification in multi-agent systems

Franco Raimondi
Department of Computer Science
King's College London
London
franco@dcs.kcl.ac.uk

**Abstract**

Multi-agent systems are often taken as a paradigm in the specification of complex systems because of their power in the abstraction from implementation details. Different kinds of modal logics have been used to model agents' knowledge - beliefs - desires, and their evolution over time. In this report I investigate how model checking techniques can be applied to some problems of verification in multi-agent systems. I review briefly th literature for multi-agent systems logics, and for model checking. Also, I present results achieved in the verification of non-temporal epistemic properties of two examples, encoded in the formalism of interpreted systems: the bit transmission problem and the protocol of the dining cryptographers.

# Contents

# 1 Introduction

The concept of *rational agent* has been adopted by many disciplines, including economy, philosophy, computer science and mathematics. One of the reason for this is that we naturally ascribe "mental qualities" to complex systems. Quoting from [28] (cited in [41, 20]):

> "To ascribe certain *beliefs, knowledge, free will, intentions, consciousness, abilities* or *wants* to a machine or computer program is legitimate when such an ascription expresses the same information about the machine that it expresses about a person. It is useful when the ascription helps us understand the structure of the machine, its past or future behaviour, or how to repair or improve it. [...] Ascription of mental qualities is most straightforward for machines of known structure such as thermostats and computer operating systems, but is most useful when applied to entities whose structure is incompletely known".

In the past twenty years, various logical theories have been developed to formalise *knowledge*, *beliefs*, *desires*, *intentions* and other *intentional attitudes* (in the sense of [11]) of rational agents. Such logics include Cohen and Levesque's theory of intention [10], Rao and Georgeff BDI architecture [16], Wooldridge's Logic for Rational Agents (LORA, [38]), and interpreted systems by Fagin, Halpern, Moses and Vardi [12].

In my research I am interested mainly in the formalisation of the *knowledge* of an agent, and its evolution over time. Indeed, epistemic logics can be used to capture properties of distributed systems and, in particular, to express desired behaviour of protocols [15]. In this report I shall use the formalism of interpreted systems [12] to represent multi-agent systems (MAS) and to reason about their epistemic properties. Interpreted systems provide a *computationally grounded* theory of agency. The notion of *computationally grounded* theory of agency was introduced by M. Wooldridge in [39] to denote a theory that can be interpreted in terms of some concrete computational model.

Though J. Halpern and M. Vardi suggested the use of model checking techniques for the *verification* of multi-agent systems in 1991 [14], it is only recently that results along these lines have been achieved [1, 22, 32, 40, 3, 31, 21, 36]. This research aims at making a contribution towards solving the problem of the verification of MAS. I argue that existing model checking techniques and tools for temporal logics can be extended to take into account other modalities that are commonly used to model MAS.

The rest of the report is organised as follows. I begin with some mathematical preliminaries (Section 2) and with the objectives of my research (Section 5. In Section 3 I review various theories for the formalisation of MAS, with a detailed introduction to interpreted systems. In Section 4 I first present model checking techniques, and then I analyse different approaches to the problem of model checking MAS. In Section 6 I introduce two examples of multi-agent

systems: the bit transmission problem and the protocol of the dining cryptographers. Section 7 lists the results obtained so far. I conclude in Section 8 by evaluating the results and by sketching possible evolutions of my the research.

# 2 Mathematical preliminaries

In this section I present a brief overview of the modal logic formalism that I am going to use in the report. This overview is taken mainly from [13] and [7], where all the proofs that are not reported here can be found.

## 2.1 Syntax

Let $P$ be a countable set of atomic formulae, usually denoted $p, q, \ldots$ The language of propositional modal logic $\mathcal{L}$ is defined by the the set of formulae $\varphi \in \mathcal{L}$:

$$\varphi ::= p | \neg\varphi | \varphi_1 \wedge \varphi_2 | \Box\varphi$$

Other connectives are introduced as usual. In particular, $\Diamond\varphi = \neg\Box\neg\varphi$, $\bot = p \wedge \neg p$, and $\top = \neg\bot$. Possible readings of $\Box\varphi$ are "It is necessarily true that $\varphi$", "It will always be true that $\varphi$", "It ought to be that $\varphi$", "It is known that $\varphi$". In the following I shall use other symbols for the modal operator $\Box$, including $K$ (to be read "it is known that") and $O$ (to be read "it ought to be that").

## 2.2 Logics

By *schema* I mean a set of formulae all having the same syntactic form. For example, the schema $\Box A \rightarrow A$ is the set of formulae
$\{\Box B \rightarrow B : B \in \mathcal{L}\}$. A *logic* is any set $\mathbf{L} \in \mathcal{L}$ such that

- $\mathbf{L}$ includes all tautologies
- $\mathbf{L}$ is closed under *Modus Ponens*, i.e.
  if $A, A \rightarrow B \in \mathbf{L}$, then $B \in \mathbf{L}$.

Members of $\mathbf{L}$ are called *theorems* and I write $\vdash_{\mathbf{L}} \varphi$ if $\varphi \in \mathbf{L}$.

A logic is *normal* if it contains the schema

$$\mathbf{K} : \Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$$

and is closed under necessitation, i.e.

$$\text{if } \vdash_{\mathbf{L}} A, \text{then } \vdash_{\mathbf{L}} \Box A$$

The smallest normal logic is denoted with $\mathbf{K}$[1]. Following standard conventions, I denote with $\mathbf{K}\mathbf{X_1}\ldots\mathbf{X_n}$ the smallest normal logic containing the schemata $X_1 \ldots X_n$; these schemata are also called the *axioms* of the logic. Table 1 lists the names of some commonly used axioms. Table 2 lists the names of some commonly used logics.

[1] Notice that I use here the same symbol to denote a schema and a logic. It should be clear from the context which is the intended meaning.

| | |
|---|---|
| D: | $\Box A \to \Diamond A$ |
| T: | $\Box A \to A$ |
| 4: | $\Box A \to \Box\Box A$ |
| 5: | $\neg\Box A \to \Box\neg\Box A$ |

Table 1: Some common names for axioms.

| Name | Axioms |
|---|---|
| **S4** | KT4 |
| **S5** | KT5 |

Table 2: Some common names for logics.

## 2.3 Possible worlds semantics

A *frame* $F$ is a pair $F = (W, R)$ where $W$ is a non-empty set of *worlds*, or *possible states*, and $R$ is a binary relation on $W$, $R \subseteq W \times W$. A *model* M is a pair $M = (F, V)$ where $F$ is a frame and $V$ is a function $V : P \to 2^W$ ($P$ is the set of atomic formulae, see above). $V$ assigns to each atomic formula a subset of $W$; $V(p)$ is the set of worlds in which $p$ has the value "true".

I write

$$M \models_w \varphi$$

to denote that $\varphi$ is *true* (or, equivalently, $\varphi$ *holds*) in model $M$ at point $w \in W$. $\models$ is defined inductively as follows:

$M \models_w p$ iff $w \in V(p)$
$M \models_w \top$
$M \models_w (\varphi \wedge \psi)$ iff $M \models_w \varphi$ and $M \models_w \psi$
$M \models_w \Box\varphi$ iff for all $w' \in W$, $wRw'$ implies $M \models_{w'} \varphi$

I write $M \not\models_w \varphi$ if it is not the case that $M \models_w \varphi$.

A formula $\varphi$ is *true in a model* $M$, denoted with $M \models \varphi$, if $M \models_w \varphi$ for every $w \in W$. A formula is *valid in a frame* $F$, denoted with $F \models \varphi$, if $M \models \varphi$ for all models $F = (M, V)$ based on $F$.

Let $\mathcal{C}$ be a class of frames. A logic $\mathbf{L}$ is *sound* with respect to $\mathcal{C}$ if, for every formula $\varphi$, $\vdash_{\mathbf{L}} \varphi$ implies $\mathcal{C} \models \varphi$. $\mathbf{L}$ is *complete* with respect to $\mathcal{C}$ if $\mathcal{C} \models \varphi$ implies $\vdash_{\mathbf{L}} \varphi$. $\mathbf{L}$ is *determined by* $\mathcal{C}$ if it is sound and complete with respect to $\mathcal{C}$

## 2.4 Standard completeness proofs

Possible world semantics is particularly attractive because many modal logics are determined by simple classes of frames (see Table 3). Generally, the proof of soundness of a logic with respect to some class of frames is straightforward. The proof of completeness is usually a bit more complicated and requires some standard machinery, presented below.

Consider a logic $\mathbf{L}$. Given a formula $\varphi$ and a set $\Gamma \subseteq \mathcal{L}$ of formulae, $\varphi$ is *deducible* from $\Gamma$, denoted with $\Gamma \vdash \varphi$, if there exist $\psi_1, \ldots, \psi_n \in \Gamma$ such that

| Logic | Class of frames |
|---|---|
| **K** | All frames |
| **KD** | Serial frames |
| **KT** | Reflexive frames |
| **S4 = KT4** | Transitive and reflexive frames |
| **S5 = KT5** | Transitive, reflexive and symmetric (i.e equivalence) frames |

Table 3: Classes of frames.

$\vdash \psi_1 \to (\psi_2 \to (\ldots \to (\psi_n \to \varphi)\ldots))$. A set $\Gamma \in \mathcal{L}$ is *consistent* if $\Gamma \nvdash \bot$. A set $\Gamma$ is *maximal* if, for any $\varphi \in \mathcal{L}$, either $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$.

**Theorem 2.1 (Lindebaum's Lemma)** *Every consistent set of formulae can be extended to a maximal consistent set*

The *canonical model* of a logic $\mathbf{L}$ is a structure $M_{\mathbf{L}} = (W_{\mathbf{L}}, R_{\mathbf{L}}, V_{\mathbf{L}})$ where
$W_{\mathbf{L}} = \{w \subseteq \mathcal{L} : w \text{ is maximal}\}$,
$wR_{\mathbf{L}}w'$ iff $\{\varphi \in \mathcal{L} : \Box\varphi \in w\} \subseteq w'$,
$V_{\mathbf{L}}(p) = \{w \in W_{\mathbf{L}} : p \in w\}$.

**Theorem 2.2** *For all formulae $\varphi$,*

$$M_{\mathbf{L}} \models \varphi \quad \text{iff} \quad \vdash_{\mathbf{L}} \varphi$$

To show that a logic $\mathbf{L}$ is complete with respect to some class of frames $\mathcal{C}$, it is enough to prove that the frame $F_{\mathbf{L}}$ underlying the canonical model $M_{\mathbf{L}}$ belongs to $\mathcal{C}$. Indeed, if $\nvdash_{\mathbf{L}} \varphi$, then $\varphi$ is false in the canonical model $M_{\mathbf{L}}$ by Theorem 2.2, and hence is false in the canonical frame $F_{\mathbf{L}}$. This shows that $\mathcal{C} \models \varphi$ implies $\vdash_{\mathbf{L}} \varphi$. Determination of logics in Table 3 is proved using this technique.

## 2.5 Multimodal logics

The syntax presented above can be extended to take into account more than one modal operator. Namely, consider a countable set of propositional variables $P$ as above, consider $i = 1, \ldots, n$ modal operators, and define formulae $\varphi$ of this multimodal language $\mathbf{L}$ as follows:

$$\varphi ::= p|\neg\varphi|\varphi_1 \wedge \varphi_2|\Box_i\varphi$$

Possible world semantics is easily extended for this multimodal language. A frame $F = (W, R_i)$, $i = 1, \ldots, n$ is a structure where $W$ is a set of worlds and $R_i \subseteq W$ are binary relations on $W$. A model is a pair $M = (F, V)$ where $V$ is a valuation function as above. The definitions consistency and maximality are unchanged, as well as the Lindebaum's lemma. Canonical models can be defined in a similar manner to prove the completeness of various logics.

## 2.6 Temporal logics

### 2.6.1 LTL

The language $\mathbf{L}_{LTL}$ of Linear-Time Logic (LTL, [19]) is defined over a set of atomic formulae $p, q, \ldots \in P$ as follows:

$$\varphi ::= p | \neg\varphi | \varphi_1 \wedge \varphi_2 | X\varphi | U(\varphi, \psi) | F(\varphi) | G(\varphi).$$

LTL formulae are interpreted over *Kripke structures*. A Kripke structure $(KS)$ is a triple $(S, R, I)$ where $S$ is a set of states, $R \subseteq S \times S$ is a transition relation and $I : S \to 2^P$ is an interpretation function. A *path* is a sequence of states in $S$ $\pi =< \pi_0, \pi_1, \pi_2, \ldots >$ such that $(\pi_i, \pi_{i+1}) \in R$ for all $i \geq 0$. $\pi_i$ denotes the $i$-th state in path $\pi$, and $pi^i =< \pi_i, \pi_{i+1}, \ldots >$ is the "tail" of path $\pi$ starting at $\pi_i$.

*Satisfiability* of a formula $\varphi$ with respect to a path $\pi$ is defined inductively as follows:

| | | |
|---|---|---|
| $\pi \models p$ | iff | $p \in I(\pi_0)$, |
| $\pi \models \neg\varphi$ | iff | $\pi \not\models \varphi$, |
| $\pi \models \varphi_1 \vee \varphi_2$ | iff | $\pi \models \varphi_1$ or $\pi \models \varphi_2$, |
| $\pi \models X(\varphi)$ | iff | $\pi^1 \models \varphi$, |
| $\pi \models (\varphi U \psi)$ | iff | $\pi^i \models \varphi$ until $\pi^k \psi$, for all $i < k$, |
| $\pi \models F(\varphi)$ | iff | there is $k$ such that $\pi^k \models \varphi$, |
| $\pi \models G(\varphi)$ | iff | $\pi^k \models \varphi$ for all $k \geq 0$. |

Notice that $F(\varphi) = \top U \varphi$, $F(\varphi) = \neg G \neg(\varphi)$, and $G(\varphi) = \neg F \neg(\varphi)$.

### 2.6.2 CTL

The language $\mathbf{L}_{CTL}$ of Computational Tree Logic (CTL, [29, 9]) is defined over a set of atomic formulae $p, q, \ldots \in P$ as follows:

$$\varphi ::= p | \neg\varphi | \varphi_1 \wedge \varphi_2 | EX(\varphi) | E(\varphi U \psi) | EG(\varphi).$$

CTL formulae are interpreted over *branching-time* structures, using the concepts of path and state introduced for LTL.

*Satisfiability* of a formula $\varphi$ in a state $s$ is defined inductively as follows:

| | | |
|---|---|---|
| $s \models p$ | iff | $p \in I(s)$, |
| $s \models \neg\varphi$ | iff | $s \not\models \varphi$, |
| $s \models \varphi_1 \vee \varphi_2$ | iff | $s \models \varphi_1$ or $s \models \varphi_2$, |
| $s \models EX(\varphi)$ | iff | There exists a path $\pi$ such that $\pi_0 = s$ and $\pi_1 \models \varphi$, |
| $s \models E(\varphi U \psi)$ | iff | There exists a path $\pi$ such that $\pi_0 = s$ and a $k \geq 0$ such that $\pi_k \models \psi$ and $\pi_i \models \varphi$ for all $0 \leq i < k$, |
| $s \models EG(\varphi)$ | iff | There exists a path $\pi$ such that $\pi_0 = s$ and $\pi_i \models \varphi$ for all $i \geq 0$. |

Other CTL operators can be derived, for example: $EF\varphi = E(\top U \varphi)$, $AX\varphi = \neg EX \neg\varphi$, and $AG\varphi = \neg EF \neg\varphi$.

# 3 Review of multi-agent systems theories

Many different formalisms are available for reasoning about MAS; a detailed review can be found in [41, 16]. Below I list the main characteristics of some formalisms.

## 3.1 Cohen and Levesque's intention logic

The key assumption of Cohen and Levesque is that intelligent agents must achieve a *rational balance* between beliefs, goals, and intentions([10], p.214). To this end, they introduce a first order multi-modal logic with four primary operators: BEL, GOAL, HAPPENS and DONE ([10], p.222). The semantics of BEL and GOAL is the usual Kripke semantics; the accessibility relation for BEL is euclidean, transitive and serial; the accessibility relation for GOAL is serial. Moreover, the GOAL relation is a subset of the BEL relation. Worlds in the formalism are an infinite sequence of events.

Beside the temporal operators HAPPENS and DONE, there are other constructs similar to dynamic logic, such as ";" to denote a sequence of events and "?" to denote a test action.

The standard temporal operators $\Box$ ("always") and $\Diamond$ ("at some time") are defined as abbreviations:

$$\Diamond\varphi = \exists x(\text{Happens } x; \varphi?); \quad \Box\varphi = \neg\Diamond\neg\varphi$$

Other constructs are derived from the basic operators; the most important is *persistent* goal[2]:

$$\begin{aligned}(\text{P-GOAL } i\ p) \quad = \quad & (\text{GOAL } i\ (\text{LATER } p)) \wedge \\ & (\text{BEL } i\ \neg p) \wedge \\ & [\text{BEFORE}((\text{BEL } i\ p) \vee (\text{BEL } i\ \Box\neg p)) \\ & \neg(\text{GOAL } i\ (\text{LATER } p))]\end{aligned}$$

which means that an agent $i$ has $p$ as a persistent goal if: $i$ has a goal that $p$ becomes true at some point in the future, and $i$ believes that $p$ is currently false, and $i$ drops his goal only if $i$ believes that the goal has been satisfied, or $i$ believes that the goal will never be satisfied.

*Intentions to act* are defined as follows[3]:

$$\begin{aligned}(\text{INTEND } i\ \alpha) \quad = \quad & (\text{P-GOAL } i \\ & [\text{DONE } i\ (\text{BEL } i\ (\text{HAPPENS } \alpha))?; \alpha])\end{aligned}$$

Notice that an agent drops an intention of doing an action only if the agent believes that the action has been performed, or the agent believes that the action cannot be performed.

[2] The definition of LATER and BEFORE is straightforward and can be found in the paper

[3] The similar definition for "intending that something becomes true" can be found in [10]

| Condition | Axiom |
|---|---|
| $\mathcal{B} \subseteq_{sup} \mathcal{D} \subseteq_{sup} \mathcal{I}$ | (INTEND $i$ $E(\varphi)$) $\rightarrow$ (DES $i$ $E(\varphi)$)(BEL $i$ $E(\varphi)$) |
| $\mathcal{B} \subseteq_{sub} \mathcal{D} \subseteq_{sub} \mathcal{I}$ | (INTEND $i$ $A(\varphi)$) $\rightarrow$ (DES $i$ $A(\varphi)$)(BEL $i$ $A(\varphi)$) |
| $\mathcal{B} \subseteq \mathcal{D} \subseteq \mathcal{I}$ | (INTEND $i$ $\varphi$) $\rightarrow$ (DES $i$ $\varphi$)(BEL $i$ $\varphi$) |
| $\mathcal{B} \cap \mathcal{D} \neq \emptyset$ | (BEL $i$ $\varphi$) $\rightarrow \neg$(DES $i$ $\neg\varphi$) |
| $\mathcal{D} \cap \mathcal{I} \neq \emptyset$ | (DES $i$ $\varphi$) $\rightarrow \neg$(INTEND $i$ $\neg\varphi$) |
| $\mathcal{B} \cap \mathcal{I} \neq \emptyset$ | (BEL $i$ $\varphi$) $\rightarrow \neg$(INTEND $i$ $\neg\varphi$) |

Table 4: Some Interaction conditions and corresponding axioms

## 3.2 Rao and Georgeff's BDI logic

My presentation here follows the lines of [16]. Rao and Georgeff's BDI propose a family of logics based on branching time temporal logics CTL. Their logics include the modal operators BEL, DES and INTEND for expressing beliefs, desires and intentions. Beliefs correspond to information that an agent has about the world. Desires correspond to states of affairs that an agent would like to achieve. Intentions correspond to desires that an agent is *committed* to achieve.

The semantics of BDI modalities is based on the standard Kripke semantics. However, each world is itself a Kripke structure for CTL logic. Hence, a world is a structure $w =< T, R >$ where $T$ is a non-empty set of time points and $R$ is a branching time structure on $T$. A *situation* is a pair $< w, t >$ composed by a world and a time point. The accessibility relations $\mathcal{B}, \mathcal{D}, \mathcal{I}$ for BEL, DES and INTEND are defined on situations. The logics proposed by Rao and Georgeff differ on the the interactions between modalities. Interaction between relations correspond to axioms in the logic. For example, if $\mathcal{D} \subseteq \mathcal{I}$, then for every agent, $i$ INTEND $i$ $\varphi \rightarrow$ DES $i$ $\varphi$.

But worlds are themselves structures, so one can also reason about interactions on the structure of worlds. If $w$ and $w'$ are worlds, $w \sqsubseteq w'$ means that $w$ has the same structure of $w'$, but fewer paths. Consider now two accessibility relation $R$ and $R'$. $R$ is a *structural subset* of $R'$, denoted with $R \subseteq_{sub} R'$, if for every $R$-accessible world $w$, there is an $R'$-accessible world $w'$ such that $w \sqsubseteq w'$. Similarly, $R$ is a *structural superset* of $R'$, denoted with $R \subseteq_{sup} R'$, if $w' \sqsubseteq w$.

Various BDI logical systems can be obtained from the interactions between relations. See Table 4 for an example.

## 3.3 Benerecetti, Giunchiglia and Serafini's MATL

MultiAgent Temporal Logic [1] is the composition of the temporal logic CTL and the logic HML (Hierarchical MetaLogic) to represent beliefs, desires and intentions.

HML is defined as follows. Let $I$ be a set of agent, and $O = \{B, D, I\}$ be a set of symbols, one for each attitude. Let $OI^* = (O \times I)^*$, i.e. each $\alpha \in OI^*$ is a string representing a possible nesting of attitudes. Each $\alpha \in OI^*$ is called a *view*, including the empty string $\epsilon$ representing the view of an "external observer".

An agent "is a tree rooted in the view that the external observer has of it" (notice that the view that an agent has of another agent can be different from the agent itself). A logical language $L_\alpha$ is associated to each view $\alpha$. Each language is used to express what is true in the representation corresponding to $\alpha$. It is imposed that $O_i\varphi$ is a formula of $L_\alpha$ iff $\varphi$ is a formula of $L_{O_i\alpha}$.

The semantics of $\{L_\alpha\}_{\alpha \in OI^*}$ is given by means of the concept of *tree*. A tree is a subset of the set of possible interpretation of a language $L_\alpha$, denoted with $M_\alpha$. Namely, each interpretation is denoted with $t_\alpha \in M_\alpha$, and a tree is a set $\{t_\alpha\}_{\alpha \in OI^*}$. A *compatibility relation* $T$ is a set of trees. A tree satisfies a formula at a view iff the formula is satisfied by all the elements that the tree associates to the view.

A *Hierarchical MetaStructure* (HM Structure) is a set of trees $T$ on $L_\alpha$ closed under containment such that there is a $t \in T$ with $t_\epsilon \neq \emptyset$, if $t_\alpha$ satisfies $O_i\varphi$, then $t_{O_i\alpha}$ satisfies $\varphi$, and if for all $t' \in T$, $t'_\alpha \in t_\alpha$ implies that $t_{\alpha O_i}$ satisfies $\varphi$ , then $t_\alpha$ satisfies $O_i\varphi$.

MATL structures (i.e. models) are a particular kind of HM structures: each language $L_\alpha$ is a CTL language. This allows for the interpretation of formulae of a language that includes BDI and temporal (CTL) operators.

## 3.4 Wooldridge's LORA

LORA [39] can be viewed as an extension of the temporal logic CTL. LORA has four main components: a classical first-order component, a BDI component, a temporal component and an action component (in a dynamic logic style).

The BDI component is very similar to the Rao and Georgeff's formalism presented in Section 3.2. The state of an agent is defined by its beliefs, desires and intentions, whose semantics is given via standard Kripke semantics, and worlds are themselves branching time structures. LORA also contains terms to reason about *groups* of agents.

The semantics of LORA is defined by means of models. A model for LORA is a structure

$$M =< T, R, W, D, Act, Agt, \mathcal{B}, \mathcal{D}, \mathcal{I}, C, \Phi >$$

where $T$ is the set of all time points, $R \subseteq T \times T$ is a branching time relation over $T$, $W$ is a set of worlds over $T$ (see Section 3.2), $D =< D_{Ag}, D_{Ac}, D_{Gr}, D_U >$ is a domain, $Act : R \rightarrow D_{Ac}$ associates an action with every relation in $R$, $Agt : D_{Ac} \rightarrow D_{Agt}$ associates an agent with every action, $\mathcal{B}, \mathcal{D}, \mathcal{I}$ are the accessibility relations, $C$ is an interpretation function for constants and $\Phi$ is an interpretation function for predicates. $D_{Ag} = \{1, \ldots, n\}$ is a set of agents, $D_{Ac} = \{\alpha, \alpha', \ldots\}$ is a set of actions, $D_{Gr}$ is a set of non-empty subsets of $D_{Ag}$, i.e. groups of agents, $D_U$ is a set of other individuals. Semantics for state and path formulae is given using LORA models.

## 3.5 Interpreted systems

This section introduces the formalism of interpreted systems in some detail, as presented in [25].

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. It is often convenient to represent the Environment in which the agents "live" as an agent; under this assumption, local states for the Environment will be denoted by $L_E$. 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 $l_E, l_E', \ldots$.

For every agent of 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 called 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*, the protocol is allowed to be non-deterministic):

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

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

The evolution of the system can be modelled by means of a transition function $\pi$ from global states and joint actions to global states:

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

where $Act = Act_1 \times \ldots \times Act_n \times Act_E$ is the set of joint actions for the system.

Intuitively this defines temporal flows on the set of global states. A *run* $r$ is a function from time to global states: $r : I\!N \to S$. Hence, a run is a sequence of global states obtained by applying the function $\pi$ to global states and joint actions. $\mathcal{R}$ denotes a set of runs; $(r, m)$ denotes the global state of run $r$ at time $m$. Equivalently, I will write $r(m)$ for $(r, m)$. $r_i(m)$ denotes the local state of agent $i$ in the global state $(r, m) = r(m)$.

Given a set of propositional variables $P$, an *interpreted system* of global states is a pair $IS = (\mathcal{R}, h)$ where $\mathcal{R}$ is a set of runs and $h : S \to 2^P$ is an interpretation function; intuitively, $h$ returns the propositional variables true in a global state.

It is possible to associate a Kripke model $M_{IS}$ to each interpreted system $IS = (\mathcal{R}, h)$, as follows ([12], p.111): $M_{IS} = (S, h, \sim_1, \ldots, \sim_n)$, where $S$ is the set of global states that are reachable in some run $r \in \mathcal{R}$, $h$ is the same evaluation function than above, and $\sim_1, \ldots, \sim_n$ are binary relations on $S$ defined by $s \sim_i s'$ if $l_i(s) = l_i(s')$ ($s, s'$ are global states of the form $(r, m)$ and $(r', m')$). From this correspondence it is possible to define the following:

$$(IS, r, m) \models \varphi \quad \text{if} \quad (M_{IS}, s) \models \varphi$$

where $(r, m) = s$. That is: $\varphi$ holds in the interpreted system $IS$ in run $r$ at time point $m$ if $\varphi$ holds in the equivalent Kripke model. Hence, all the propositional operators are easily defined; Epistemic modalities $K_i$ (one for each agent) are interpreted as follows:

$$(IS, r, m) \models K_i\,\varphi \quad \text{if for all } (r', m')\ (r, m) \sim_i (r', m') \text{ implies } (IS, r', m') \models \varphi.$$

I say that $\varphi$ is valid in the interpreted system $IS$ and I write $IS \models \varphi$ if $(IS, r, m) \models \varphi$ for every $(r, m)$ in $IS$. The resulting logic for modalities $K_i$ is $S5_n$; this models agents with complete introspection capabilities and veridical knowledge [18].

Temporal operators can be added to this formalism. LTL-like operators are defined in [12], p.115. In [21] the logic CTLK is built introducing CTL-like operators in the formalism of interpreted systems.

### 3.6 Extending interpreted systems with deontic operators

The notion of interpreted systems can be extended to incorporate the idea of correct functioning behaviour of some or all of the components [24].

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. 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

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

$O_i \varphi$ is used to represent that $\varphi$ 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.* The (doubly relativised) modal operator $\widehat{K}_i^j$ is used for this notion, interpreted as follows:

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

$\widehat{K}_i^j \varphi$ may be read as expressing that agent $i$ knows $\varphi$ (in the same information-theoretic sense as is captured by $K_i \varphi$) when agent $j$ is functioning correctly. Note that $O_j K_i \varphi \rightarrow \widehat{K}_i^j \varphi$ is valid, but not the converse.

# 4 Verification in MAS

## 4.1 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, [4]) have been used to develop model checkers that are able to check large number of states ([5]). Alternative approaches using automata have also been developed [37]. In the following sections I present the two approaches and I introduce two widely used model checkers: NuSMV and SPIN. The former is based on BDDs', the latter on automata.

### 4.1.1 Model checking with SMV

Bryant [4] introduced techniques and algorithms to manipulate boolean functions. The idea is to represent boolean functions of the form $f(x_1, \ldots, x_n)$ as a rooted, directed and a-cyclic graph, called Ordered Binary Decision Diagram (OBBD). A function graph can often be "reduced" by eliminating redundant vertices ([4], p.5). Given a fixed ordering of the variables $x_1, \ldots, x_n$, it can be shown that the reduced function graphs form a *canonical* representation for Boolean functions[4].

Algorithms for the manipulation of function graphs are given in the paper, and it is shown that these algorithms have a time complexity proportional to the size of graphs. Although in the worst case a boolean function could require a graph whose size is exponential in the number of arguments, it is shown that many functions have a more compact representation.

To see how these ideas can be applied in the verification of CTL models, consider a model $(S, R, L)$ for the logic CTL, where $S$ is a set of states, $R$ is the transition relation and $L$ is the valuation. Every CTL formula $f$ is identified with the set of states in which $f$ is true, $\{s \in S : s \models f\}$. It can be shown that CTL operators can be characterised using fixed point operators ([5, 29]). For instance,

$$EFp = \mu y.(p \vee EXy)$$

$$EGp = \nu y.(p \wedge EXy)$$

That is: the set of states in which $EFp$ is true is the least fixed point $y$ of the operator $(p \vee EXy)$, where each formula is identified with the corresponding set of states. Analogously, the set of states satisfying $EGp$ is the greatest fixed point of the operator $(p \wedge EXy)$.

[4] In the sense that the reduced function graph is unique (up to isomorphism) for each boolean function $f$.

Provided that the set of states $S$ is finite ([29], p.21), the fixed points above can be obtained as the limit of the following (finite) series:

$$EFp = \cup_i (\lambda y.(p \vee EXy))^i(\text{false})$$

$$EGp = \cap_i (\lambda y.(p \wedge EXy))^i(\text{true})$$

OBDD's and the fixed point characterisation of CTL operators are the building blocks of symbolic model checking. The idea is to represent the set of states $S$ as a set of boolean vectors $\{0,1\}^n$, and the transition relation $R$ as a boolean function on states ([29], p.27). It is easy to derive an algorithm to compute the set of states satisfying temporal operators, iterating the relevant operators, as presented above (see [29, 5, 9] for details). Encoding states and transitions with OBDD's (i.e. *symbolically*) gives an effective solution to the so called *state explosion problem*. This problem arises when the states in $S$ are dealt with explicitly; indeed, the size of $S$ is exponential in the number of (boolean) variables.

SMV (Symbolic Model Verifier) is a tool developed from 1993 at Carnegie Mellon University by Clarke, McMillan, and others. It is a software for checking properties of finite state systems expressed in CTL temporal logic. SMV has an input language to describe the temporal model, and uses OBDDs to represent states and transitions.

For the purposes of my research, I will consider NuSMV [8], an implementation of SMV. 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 variables initialisation,
3. A section for the description of the transition relation.

The following is an example of a NuSMV program.[5]:

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

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))`
NuSMV would produce a counterexample.

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

### 4.1.2 Model checking with SPIN

Another approach to model checking involves the use of automata. Formally [30, 33], a Büchi automaton $B = (Q, I, \delta, F)$ over an alphabet $\Sigma$ is given by a finite set $Q$ of locations, a non-empty set $I \subseteq Q$ of initial locations, a transition relation $\delta \subseteq Q \times \Sigma \times Q$, and a set $F \subseteq Q$ of accepting locations. A run of $B$ over an infinite $\omega$-word $w = a_0 a_1 \ldots \in \Sigma^\omega$ is an infinite sequence $\rho = q_0 q_1 \ldots$ of locations $q_i \in Q$ such that $q_0 \in I$ and $(q_i, a_i, q_{i+1}) \in \delta$. The run is *accepting* if there exists some $q \in F$ such that $q_i = q$ holds for infinitely many $i$. The language $\mathcal{L}(B) \subseteq \Sigma^\omega$ it the set of $\omega$-words for which there exists some accepting run $\rho$ of $B$. A language $L \in \Sigma^\omega$ is called $\omega$-regular iff $L = \mathcal{L}(B)$ for some automaton $B$.

It is possible to prove [37] that, given an LTL formula $\varphi$ built over a finite set $P$ of atomic propositions, one can build a Büchi automaton $B_\varphi = (Q, I, \delta, F)$ over the alphabet $\Sigma$, where $\Sigma = 2^P$ and $|S| \leq 2^{O(|\varphi|)}$, such that $\mathcal{L}(B_\varphi)$ is exactly the set of computations satisfying the formula $\varphi$.

Similarly ([33], p.18), given a state $s$ of an LTL model, it is possible to construct a Büchi automaton $B_s$, accepting just the words $\omega_\pi$ corresponding to valid computation paths $\pi$ starting at $s$.

Checking whether $s \models \varphi$ amounts to checking whether $\mathcal{L}(B_s) \subseteq \mathcal{L}(B_\varphi)$. Equivalently, one can check whether $\mathcal{L}(B_s) \cap \overline{\mathcal{L}(B_\varphi)} = \emptyset$. Three important results from [37] and [30] are summarised below:

1. The emptiness problem for Büchi automata (i.e., the problem of determining for a given automaton $B$ whether $B$ accepts some word) is solvable in linear time.

2. For a Büchi automaton $B$ with $n$ locations over alphabet $\Sigma$, there is a Büchi automaton $\overline{B}$ with $2^{O(nlog(n))}$ locations such that $\mathcal{L}(\overline{B}) = \Sigma^\omega \setminus \mathcal{L}(B)$.

3. Given a Büchi automaton $A$ with $n$ locations and a Büchi automaton $B$ with $m$ locations, one can construct a Büchi automaton with $2nm$ states that accepts $\mathcal{L}(A) \cap \mathcal{L}(B)$.

From these results it is possible to derive that the time complexity for model checking using automata is linear in the size of the model and exponential in the size of the formula $\varphi$ to be checked (see [30], p.18).

SPIN [17] is a model checker based on the theory of automata presented above. The specification of the model is given via a graphical front-end called XSPIN, using the PROMELA language. LTL formulae are typed in XSPIN. The PROMELA code and the LTL formula are converted mechanically to Büchi automata, and the verification of the formula is performed following the procedure presented above.

### 4.1.3 Bounded Model Checking

A different approach to symbolic model checking has been introduced recently in [2]. This approach is based on SAT procedures (i.e. propositional decision

procedures): the idea is to reduce model checking for LTL to propositional satisfiability.

Specifically, given an LTL temporal model $\mathcal{M}$ and a positive integer $k$, the *bounded semantics* for a model $\mathcal{M}$ is defined considering computational paths of length $k$; in symbols, $(\mathcal{M}, \pi) \models_k \varphi$ means that the formula $\varphi$ is valid in the model $\mathcal{M}$ along the path $\pi$ with bound $k$. It is possible to prove ([2]) that, given an LTL model $\mathcal{M}$ and an LTL formula $\varphi$, $(\mathcal{M}, \pi) \models \varphi$ iff there exists an integer $k$ and a path $\pi$ with $(\mathcal{M}, \pi) \models_k \varphi$.

Bounded model checking can then be reduced to propositional satisfiability, as follows. A propositional formula $[\mathcal{M}, \varphi]_k$ is built such that $[\mathcal{M}, \varphi]_k$ is satisfiable iff $\varphi$ is valid along some path $\pi$. To construct $[\mathcal{M}, \varphi]_k$, a propositional formula $[\mathcal{M}]_k$ is first defined to enforce a valid path. Then, the LTL formula $\varphi$ is translated into a propositional formula $[\varphi]_k$, and $[\mathcal{M}, \varphi]_k$ is obtained as the conjunction of $[\mathcal{M}]_k$ and $[\varphi]_k$: $[\mathcal{M}, \varphi]_k = [\mathcal{M}]_k \wedge [\varphi]_k$.

It is possible to prove that $[\mathcal{M}, \varphi]_k$ is satisfiable iff $(\mathcal{M}, \pi) \models_k \varphi$ for some $\pi$ and some $k$ ([2]).

Bounded model checking has been extended to CTL in [34].

## 4.2 Model checking MAS: the state of the art

The methodologies presented in Section 4.1 have been widely and successfully used in the verification of temporal properties of hardware and software components.

As the objective of my research is to verify properties of multi-agent systems, the methodologies presented above can be taken as a starting point, but need to be extended. Specifically, they are not designed to represent multi-agent systems, and they allow for the verification of a single modality, the temporal modality.

Different approaches have been proposed for the problem of verification in MAS. In this section I summarise some of them.

1. In [40], M. Wooldrige et al. introduce the MABLE language to specify a multi-agent system. Properties of the systems (i.e. logical formulae) are expressed using a simplified version of LORA (see Section 3). Beliefs, desires and intentions of the agents are modelled using nested data structures, following the ideas of Benerecetti, Giunchiglia and Serafini (see Section 3). The MABLE compiler produces a PROMELA code from the MABLE description, and logical formulae are translated into LTL formulae suitable for SPIN. Similarly, in [3] Bordini et al. translate a multi-agent system specification given in a particular language, AgentSpeak(F), into PROMELA code. In general, these two works are concerned with the verification of BDI attitudes and their evolution with time.

2. The problem of model checking knowledge and time is explored in [31] by R. van der Meyden and N. Shilov. They consider a particular class of interpreted systems: *synchronous with perfect recall* interpreted systems. *Perfect recall* means that each agent keeps a complete record of all the

(local) states he passes through; formally, the local state $r_i(m)$ of agent $i$ at time $m$ is a sequence of the form $r_i(m) = r_i(0) \ldots r_i(m)$. Here *synchronous* means that if $(r, u) \sim_i (r', v)$, then $u = v$. Besides complexity results for the problem, an algorithm is given for model checking these class of systems using Büchi automata.

3. In [32], R. van der Meyden and K. Su describe how OBDD's and symbolic model checking techniques can be used for the verification of *synchronous with perfect recall* interpreted systems by means of an example, the protocol of the dining Cryptographers (see Section 6). This approach is different from the one at the previous point, as OBDD's are used in place of Büchi automata. Though limited to a particular class of interpreted systems, this approach tries to use traditional model checking techniques in the verification of multi-agent systems.

4. In [15] W. van der Hoek and M. Wooldridge present a methodology to model check knowledge and time. Their approach is based on interpreted system and *local propositions*. Formally, given an interpreted system $I = (R, \pi)$ (see Section 3.5), a proposition $\varphi$ is said to be *local to Agent i* if $\varphi$ is propositional (i.e. no modal operators appear in $\varphi$) and for all points $(r, u)$, $(r', v)$ in $I$, if $(r, u) \sim_i (r', v)$, then $(I, (r, u)) \models \varphi$ iff $(I, (r', v)) \models \varphi$. W. van der Hoek and M. Wooldridge prove that it is possible to translate a modal formula involving knowledge operators into an LTL formula involving local propositions. This last formula is checked using SPIN. The main problem with this approach is that the local propositions needed for the translation into LTL cannot be computed mechanically, and must be provided by the user. Nevertheless, several properties of the bit transmission problem (see Section 6) can be checked automatically ([15], p.11).

5. More recently, A. Lomuscio and W. Penczek [21] extended an algorithm for bounded model checking CTL formulae to include epistemic operators in a logic called CTLK, based on interpreted systems semantics. The idea is very similar in spirit to [2]: the model checking problem is reduced to a SAT problem for a propositional formula $[\mathcal{M}, \varphi]_k$, where $\mathcal{M}$ is a model of the logic CTLK, $\varphi$ is a formula and $k$ is a bound. The propositional formula is the conjunction of two formulae $[\mathcal{M}, \varphi]_k = [\mathcal{M}]_k \wedge [\varphi]_k$: the first formula constrains the model to be a correct model, while the second formula translates the CTLK operators into propositional operators.

# 5 Research plan

Multi-agent systems have long been recognised as an important conceptualisation in the design and analysis of complex, distributed systems. These systems include, for example, communication and security protocols. However, if the multi-agent systems paradigm is to be used, being able to *verify formally* properties of multi-agent systems is a crucial element.

My research is concerned with the problem of verification in multi-agent systems. I chose to concentrate on the formalism of interpreted systems to describe multi-agent systems: this formalism focuses mainly on epistemic and temporal operators and allows to reason about knowledge of an agent, common and distributed knowledge of a group of agents, and their evolution with time [12]. Sergot and Lomuscio [23] extended interpreted systems by means of deontic operators (deontic interpreted systems), thus allowing to reason about the correct behaviour of single components. In the framework of interpreted systems extended with deontic operators, I have two kind of results that I hope to achieve by the end of my PhD:

- To provide a methodology to verify automatically properties of interpreted systems,
- To extend the theoretical results of [23] and [20]

The works presented in the literature overview of Section 4.2 deal with various techniques for the formal verification of *attitudes* in multi-agent systems. However, verification of *epistemic* properties is considered only in [31] and [32], and for a limited class of interpreted systems. My aim is to consider a more general class of deontic interpreted systems, and to allow for the verification of epistemic and deontic modalities.

I have already obtained some results in the verification of non-temporal epistemic and deontic properties of interpreted systems, and these results are presented in Section 7. In the work carried out so far, my idea has been to translate a specification of an interpreted system into an SMV program, so that the set of reachable global states can be computed. From the set of reachable global states it is possible to build the epistemic relations. All this process is performed automatically. This allowed for the automatic verification of two examples from the literature: the bit transmission problem and the protocol of the dining cryptographers. These results have been presented in details in the two papers in the Appendix.

My work-plan for the future is:

- To modify and extend the compilation process from an interpreted system specification into SMV code. Ideally, this new compilation technique would allow to verify epistemic and temporal operators using existing tools, such as NuSMV. I plan to explore the feasibility of this line of research in the next **3 months**.
- To explore the translation from interpreted systems into timed automata. Indeed, there is a tool to verify temporal and epistemic modalities for timed automata [27] using *bounded model checking techniques*. I plan to explore this line of research in parallel with the compilation into SMV code in the next **4 months**.
- To explore the use of OBBD's (or other model checking techniques) for the verification of interpreted systems. This line of research is different

from the ones above, because it does not involve existing tools. My idea is to follow the steps of [29]: translate global states and transition functions into OBBD's, and then verify properties by means of algorithms based on OBBD's manipulation. This line of research involves a theoretical investigation that could take up to **6 months**, possibly followed by a software implementation.

- For all the lines of research presented above, I plan to investigate the complexity of the problem of verification.
- In parallel with the previous points, and up to the end of my PhD, I plan to include more examples of multi-agent systems verification.

Also, there are some theoretical issues related to the formalism of interpreted systems that I would like to investigate:

- The completeness of $S5_n + \Box_1\Diamond_2\varphi \rightarrow \Diamond_1\Box_2\varphi$ [20]. The standard completeness proof using canonical models failed for this system, and I plan to explore different completeness proof techniques for this issue.
- An axiomatisation for the $\widehat{K}_i^j$ operator [23].

I will investigate these issues in parallel with the research for the verification problem in multi-agent systems, and I hope to obtain some results in the next **6 to 9 months**.

# 6 Examples

In my research I would like to develop methodologies to model check epistemic properties of multi-agent systems. A lot of research in the area of multi-agent systems is theoretical, with just few examples. I think that it is important to ground theory into examples, and this Section presents two scenarios: the bit transmission problem and the protocol of the dining cryptographers, and how these can be encoded in the formalism of interpreted systems.

## 6.1 The bit transmission problem

The bit-transmission problem [12] 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) to $R$. 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 acknowledgements 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. There are three active components in the scenario: a sender, a receiver, and a communication channel. It is convenient to see sender and receiver as agents, and the communication channel as the environment. 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 $\epsilon$ representing a circumstance under which no bit has been received yet:

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

The environment requires 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. Global states for the system $G$ are defined as $G \subseteq L_S \times L_R \times L_E$.

The set of actions $Act_i$ for every agent in the system can be modelled as follows:

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

Here $\lambda$ stands for no action.

The actions $Act_E$ for the environment correspond to the transmission of messages between $S$ and $R$ on the unreliable communication channel. It is assumed 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, one can 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.

The protocol $P_S : L_S \rightarrow 2^{Act_S}$ for $S$ can be defined as follows:

$$P_S(0) = sendbit(0), \quad P_S(1) = sendbit(1),$$
$$P_S((0, ack)) = P_S((1, ack)) = \lambda,$$

| **Final state** | **Transition condition** |
|---|---|
| $(0, ack)$ | $L_S = 0$ and $Act_R = sendack$ and $Act_E = S{-}R$ or<br>$L_S = 0$ and $Act_R = sendack$ and $Act_E = {\leftarrow}R$ |
| $(1, ack)$ | $L_S = 1$ and $Act_R = sendack$ and $Act_E = S{-}R$ or<br>$L_S = 1$ and $Act_R = sendack$ and $Act_E = {\leftarrow}R$ |

Table 5: Transition conditions for $S$

| **Final state** | **Transition condition** |
|---|---|
| 0 | $Act_S = sendbit(0)$ and $L_R = \epsilon$ and $Act_E = S{-}R$ or<br>$Act_S = sendbit(0)$ and $L_R = \epsilon$ and $Act_E = S{\rightarrow}$ |
| 1 | $Act_S = sendbit(1)$ and $L_R = \epsilon$ and $Act_E = S{-}R$ or<br>$Act_S = sendbit(1)$ and $L_R = \epsilon$ and $Act_E = S{\rightarrow}$ |

Table 6: Transition conditions for $R$

(I omit brackets when writing singleton sets).

Similarly, for $R$:

$$P_R(\epsilon) = \lambda, \quad P_R(0) = P_R(1) = sendack.$$

For the environment, a constant function can be used:

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

The evolution of the system can be modelled by means of the evolution functions for each agent. In [12] the evolution function is a function $\pi : G \times Act \rightarrow G$, where $Act = Act_S \times Act_R \times Act_E$ is the set of joint actions for the system. For example, the definition of $\pi$ contains the following:

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

Equivalently, in Table 5, 6, and 7 I list the conditions actually causing a transition. Let $IS_{\rm b}$ be the model obtained by following the process described above, on which an appropriate set of propositional variables is interpreted in a natural way[6]. For the example under consideration I shall want to check the following

[6] I assume the following:

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

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

| Final state | Transition condition |
|---|---|
| $(.,.)$ | $Act_S = \lambda$ and $Act_R = \lambda$ |
| $(sendbit,.)$ | $Act_S = sendbit(0)$ and $Act_R = \lambda$ or<br>$Act_S = sendbit(1)$ and $Act_R = \lambda$ |
| $(.,sendack)$ | $Act_R = \lambda$ and $Act_R = sendack$ |
| $(sendbit,sendack)$ | $Act_S = sendbit(0)$ and $Act_R = sendack$ or<br>$Act_S = sendbit(1)$ and $Act_R = sendack$ |

Table 7: Transition conditions for $E$

formulae:

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

$$IS_{\mathrm{b}} \models \mathbf{recbit} \rightarrow \big(K_R\,(\mathbf{bit} = \mathbf{0}) \vee K_R\,(\mathbf{bit} = \mathbf{1})\big) \tag{2}$$

$$IS_{\mathrm{b}} \models \mathbf{recack} \wedge (\mathbf{bit} = \mathbf{0}) \rightarrow K_S\,K_R\,(\mathbf{bit} = \mathbf{0}) \tag{3}$$

## 6.2 The bit transmission problem with faults

I analyse here the case in which agents do not behave as they are supposed to. Two concrete examples are considered. In the first, the receiver may fail to send an acknowledgement message when it receives a bit [26]. In the second, the receiver $R$ may send acknowledgements even when it is not supposed to, i.e., when it has not yet received the value of the bit.

### 6.2.1 Faulty receiver – 1

This case is dealt with by considering a new protocol which extends the original one. Two new local states are introduced 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. If the receiver enters one of its faulty states, it can nevertheless recover to a non-faulty state [24], by sending an acknowledgement the next time round.

The protocol for the receiver $R$ is modified as follows:

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

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

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

For this version of the problem, the system evolution function $\pi'$ is required to be the same as in the basic version for actions conforming to protocols in non-faulty states. For the other cases, $\pi'$ 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 $\pi'$ contains, for all $\gamma_E \in Act_E$, unless stated otherwise (the definitions for $\mathbf{bit} = \mathbf{1}$ are analogous):

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

Note that in the last two cases 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. For completeness:

$$\begin{aligned}
\pi'(\,((0,ack),(0,f),(.,.)),(\lambda,\lambda,\gamma_E)\,) &= ((0,ack),(0,f),(\lambda,\lambda)) \\
\pi'(\,((0,ack),(0,f),(.,.)),(\lambda,sendack,\gamma_E)\,) &= ((0,ack),0,(\lambda,sendack))
\end{aligned}$$

Others cases for the definition of $\pi'$ are similarly expressed.

Let $IS'_{\mathrm{b}}$ be the model obtained by following the process described above[7]. In the next Section it will be verified that:

$$IS'_{\mathrm{b}} \models \mathbf{recbit} \rightarrow \big(K_R\,(\mathbf{bit} = \mathbf{0}) \vee K_R\,(\mathbf{bit} = \mathbf{1})\big) \tag{4}$$

$$IS'_{\mathrm{b}} \models \mathbf{recack} \rightarrow K_S\big(K_R\,(\mathbf{bit} = \mathbf{0}) \vee K_R\,(\mathbf{bit} = \mathbf{1})\big) \tag{5}$$

$$IS'_{\mathrm{b}} \models \mathbf{recack} \wedge (\mathbf{bit} = \mathbf{0}) \rightarrow K_S\,K_R\,(\mathbf{bit} = \mathbf{0}) \tag{6}$$

### 6.2.2 Faulty receiver − 2

For this version of the problem a new local state for the receiver $R$ is introduced, namely $(\epsilon, f)$, and deontic concepts introduced in Section 3.6 are used. This is the local state in which $R$ did not receive a bit but nevertheless $R$ sent an acknowledgement. The local states $(0,f)$ and $(1,f)$ of $R$ represent the case where $R$ has received the value of the bit and has sent an erroneous acknowledgement at some time in the past. For $S$, since I am not admitting (for the purposes of the example) the possibility of faults, its local states are all green. Thus:

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

For the case of the environment, the possibility of faulty, or unreliable, behaviour of the channel is already represented by the four kinds of transmit actions.

[7] The interpretation of the variables is similar.

'Faults' of the communication channel 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:

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

For $R$, local states are defined as follows:

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

The protocol functions of this deontic interpreted system are now defined. Given that the two sets of local states for $S$ and $E$ have not changed, the functions $P_S$ and $P_E$ can be kept as for the basic version. $P_R$ ha to be extended, 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

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

It remains to define the evolution function $\pi''$. Essentially the definition of $\pi$ is extended 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 $\pi$, i.e., $R$ will correctly store the bit if it has received it and remain in the state $\epsilon$ otherwise. First, the effects of protocol-violating actions in green $R$ states are specified. For the case where the bit is 0 (the other can be done similarly) it is imposed that:

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

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 the definition of $\pi$ is extended so that $\pi''$ is defined also on red local states for $R$. Once $R$ is in a red state it is imposed that it will remain in a red

state, although it will correctly store messages, if received.

$$\pi''(\,(0,(\epsilon,f)),(.,.)),(sendbit(0),sendack,S{-}R)\,) = \\ (\,(0,ack),(0,f),(sendbit,sendack)\,)$$

$$\pi''(\,(0,(\epsilon,f)),(.,.)),(sendbit(0),sendack,S{\rightarrow})\,) = \\ (\,0,(0,f),(sendbit,sendack)\,)$$

$$\pi''(\,(0,(\epsilon,f)),(.,.)),(sendbit(0),sendack,{\leftarrow}R)\,) = \\ (\,(0,ack),(\epsilon,f),(sendbit,sendack)\,)$$

$$\pi''(\,(0,(\epsilon,f)),(.,.)),(sendbit(0),sendack,-)\,) = \\ (\,0,(\epsilon,f),(sendbit,sendack)\,)$$

The other cases are omitted.

Let $IS''_{\mathrm{b}}$ be the model obtained in this example. Considering green and red states the doubly relativised operator $\widehat{K}_S^R$ introduced in section 3.5 can be interpreted. I want to verify that none of the epistemic formulae presented in earlier sections hold in this version. In particular:

$$IS''_{\mathrm{b}} \not\models \mathbf{recack} \rightarrow K_S\big(K_R\,(\mathbf{bit}=\mathbf{0}) \vee K_R\,(\mathbf{bit}=\mathbf{1})\big) \tag{7}$$

However, it is possible to verify that a particular form of knowledge still holds. If $S$ makes the assumption of $R$'s correct functioning behaviour, then, upon receipt of an acknowledgement, it makes sense for $S$ to assume that $R$ does know the value of the bit; this is exactly what is captured by $\widehat{K}_S^R$[8].

$$IS''_{\mathrm{b}} \models \mathbf{recack} \rightarrow \widehat{K}_S^R\big(K_R\,(\mathbf{bit}=\mathbf{0}) \vee K_R\,(\mathbf{bit}=\mathbf{1})\big) \tag{8}$$

$$IS''_{\mathrm{b}} \models \mathbf{recack} \wedge (\mathbf{bit}=\mathbf{0}) \rightarrow \widehat{K}_S^R\, K_R\,(\mathbf{bit}=\mathbf{0}) \tag{9}$$

## 6.3 The protocol of the dining cryptographers

This protocol is presented in [6] and [32]. The general aim of the protocol is to allow an untraceable broadcasting of messages in multi-agent systems. The protocol is originally introduced with the following example:

"*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*

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

*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]

Three agents $C_i$ ($i = \{1, 2, 3\}$) model the three cryptographers and one agent $E$ models the Environment. The Environment is used to (non-deterministically) select the initial configuration of the payer and the results of coin tosses.

Local states $LC_i$ for each cryptographer $C_i$ are represented with a tuple

$$LC_i = \langle v_1, v_2, v_3 \rangle$$

where[9]:

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

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

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

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:

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

The environment is not performing any action:

$$ActE = \lambda$$

[9] $\lambda$ denotes an empty or undefined state or action.

| **Final state** | **Transition condition** |
|---|---|
| $\langle$Paid, Equal, $\lambda\rangle$ | $(\langle\lambda,\lambda,\lambda\rangle, *, \langle 1,$Head, Head, $*\rangle, ), (*)$ or<br>$(\langle\lambda,\lambda,\lambda\rangle, *, \langle 1,$Tail, Tail, $*\rangle, ), (*)$ |

Table 8: Transition conditions for $C_1$

Hence, there is no protocol for the Environment[10].
The protocol $PC_i$ for the cryptographers is as follows:

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

The definition of initial states for agents representing the cryptographers is:

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

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 set of global states is $G = LC_1 \times LC_2 \times LC_3 \times LE$; $Act = ActC_1 \times ActC_2 \times ActC_3 \times ActE$ is the set of joint actions. Following what has been done for the bit transmission problem in the previous section, the evolution function for the system is built using the definition of the evolution functions for each agent. Notice that the evolution of $LE$ and the dependences from $ActE$ in the remaining evolution functions are not needed, thanks to the assumptions on the environment (no actions, and local state fixed at the beginning of the run). As an example, the transition conditions causing a transition to local state $\langle$Paid, Equal, $\lambda\rangle$ for the first Cryptographer $C_1$ are listed in Table 8.

Let $IS_{\rm d}$ be the model obtained by following the process described above. A set of atomic propositions {**paid$_1$**, **paid$_2$**, **paid$_3$**, **even**, **odd**} is defined, that can be interpreted in a natural way in $IS_{\rm d}$:

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

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

I will check the correctness of:

$$IS_{\mathrm{d}} \models \mathbf{odd} \rightarrow (\neg \mathbf{paid_1} \rightarrow (K_{C_1}(\mathbf{paid_2} \vee \mathbf{paid_3}) \wedge \neg K_{C_1}(\mathbf{paid_2}) \wedge \neg K_{C_1}(\mathbf{paid_2})))) \tag{10}$$

$$IS_{\mathrm{d}} \models \mathbf{even} \rightarrow K_{C_1}(\neg \mathbf{paid_1} \wedge \neg \mathbf{paid_2} \wedge \neg \mathbf{paid_3}) \tag{11}$$

# 7 Results obtained so far

The available model checkers (e.g. SMV and SPIN) do not allow for the verification of epistemic operators, but still they can be used in the verification of interpreted systems.

The methodology that I present here has been developed to automatically verify non-temporal epistemic properties of interpreted systems. Given an interpreted system $IS$, it is possible to build an SMV program $P_{IS}$ such that the set of states in the temporal model generated by $P_{IS}$ has a one-to-one correspondence with global states of the interpreted system. The SMV program is constructed as follows[11]:

1. Declare an SMV-variable for each agent, where the set of local states is stored.
2. Declare an SMV-variable for the actions, one for each agent.
3. Actions (in SMV) take a value that is based on the protocol of the interpreted system.
4. The conditions for the evolution of the local states variable are obtained from the evolution function of the corresponding interpreted system.
5. Propositions are defined using the function $h$ of the interpreted system.

The set of "reachable states" is needed to build epistemic and deontic relations between states to evaluate formulae involving epistemic and deontic operators. This set can be obtained automatically from NuSMV, as explained below.

Then, I have built a parser that takes the set of reachable states as input and produces a model with epistemic and deontic relations in the format of Akka, a Kripke model editor[12]. This methodology is summarised in Fig. 1

In the following subsections I apply this methodology to the mechanical verification of formulae (1–11) of Section 6.

[11] The methodology presented here is a revised version of the one in [22].

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

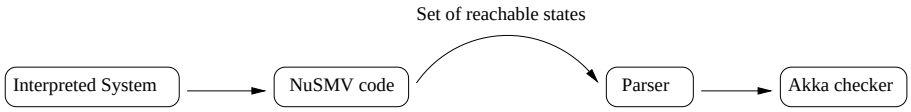


Figure 1: The methodology for non-temporal model checking.

## 7.1 Verification of the bit transmission problem

This example is analysed in detail in the paper [22]. I report here the main results.

**Basic case**: 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 the local states for $R$ and for the environment can be encoded, and the actions specified. The protocol functions and the evolution function $\pi$ (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;
```

I modified the NuSMV code to generate the reachable global states of the system[13]. The output is a file with the list of all the reachable states. This file is parsed by means of a C++ program that I wrote, 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. Figure 2 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. 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. Specifically, we can check mechanically that Formulae (1–3) hold in this model.

**Faulty Receiver − 1**: The NuSMV implementation of this version of the bit transmission problem is a straightforward extension of the code for the basic version presented above (details can be found in [22]). Formulae (4–6) can be checked with Akka. We see that the faults by $R$ considered in this version of the

[13]Thanks to Marco Pistore, who helped in the modification of NuSMV v2.0. Thanks to this work, printing reachable states is a feature available from NuSMV v2.1.

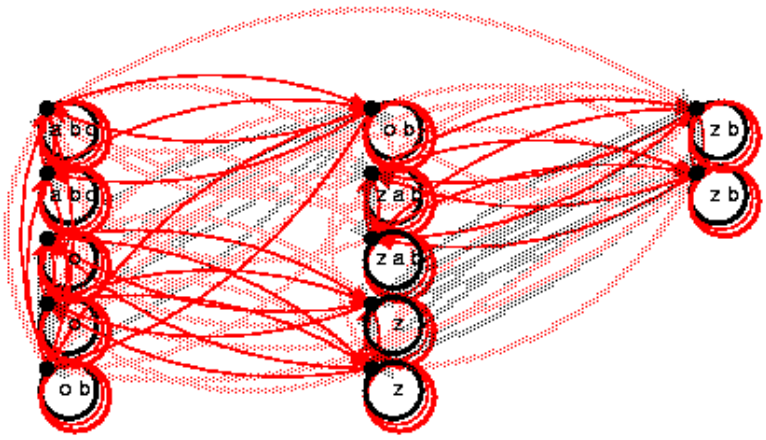


Figure 2: Screenshot in Akka of the model $IS_{\mathrm{b}}$ for the basic version of the bit transmission problem.

example are recoverable, in the sense that they do not necessarily compromise communication.

**Faulty Receiver – 2**: The NuSMV code for this case is an extension of the basic code. The evolution function for the receiver is a bit more complicated, to cover all possible cases of faulty behaviour. Formulae (7–9) can be checked mechanically, as shown in [22].

## 7.2 Verification of the protocol of the dining cryptographers

In the interpreted system for the protocol of the dining cryptographers four agents are involved: three cryptographers and the environment. I decided to extend the methodology presented above, by developing a Java compiler to translate the specification of an interpreted system given in XML into SMV code. Preliminary results are reported in [36, 35]. The revised methodology proceeds as follows:

- **Local states**: I 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 $i$: a local state $l_i$ for agent $i$ 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.

- **Evolution function**: In the description below I use a slightly modified and simpler syntax for $\pi$: the idea is to decompose final global states of the function $\pi$. Consider $n$ evolution functions, one for each agent, $\pi_i : S \times Act \to L_i$ $(i = 1, \ldots, n)$ from global states and actions to local

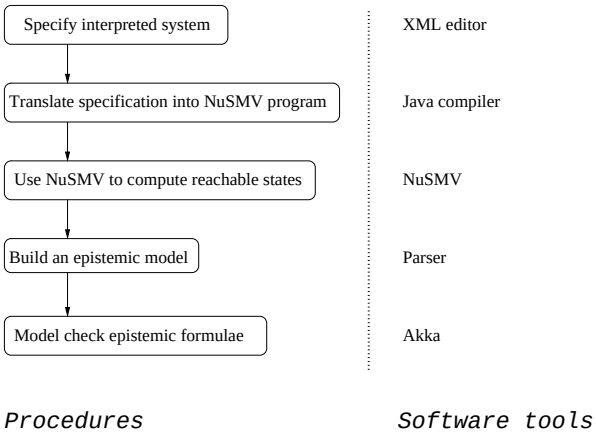


Figure 3: The revised methodology for non-temporal model checking.

states of agent $i$. In the following, I 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 $L_i$.

The specification of an interpreted system is required as an input for the Java translator. This specification must contain at least the following information:
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.

These parameters are read from an XML file, whose DTD is available at the following link:
`http://www.dcs.kcl.ac.uk/pg/franco/is/is.dtd`.

This revised methodology is illustrated in Fig. 3. I did not include deontic operators in the Java translator and in the XML specification, but nevertheless we have been able to check the protocol of the dining cryptographers.

Specifically, Formulae (10) and (11) are true in the model $IS_{\mathrm{d}}$. These two formulae confirm the correctness of the statement: 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 odd, then the first cryptographer knows that nobody paid for the dinner.

# 8 Conclusion

In this report I have presented my research, that deals mainly with the issue of verification in MAS. After a detailed introduction on multi-agent system theories and model checking, I presented results obtained in the verification of epistemic properties in interpreted systems. Also, a review of the (recent) literature in MAS verification is reported in Section 4.2.

Verification of MAS is a relatively unexplored field of research, and this research aims at making a contribution in this field. Specifically, in the work carried out so far I have been able to model check automatically non-temporal epistemic properties of interpreted systems in two well known examples: the bit transmission problem and the protocol of the dining cryptographers. Other examples could be checked using the methodology presented in this work, and this is one of my objectives for the future.

However , the methodologies that I have presented in Section 7 have some drawbacks:

- The representation of global states in Akka is not symbolic. It is true that in most cases the set of reachable states is orders of magnitude smaller than the full cartesian product of local states, and reachable states are computed symbolically by NuSMV. Nevertheless, scalability can be an issue for problems with a large set of reachable states.
- Model checking is limited to epistemic or deontic operators. Though non-temporal model checking makes sense in many cases (see [36]), sometimes we want to reason about the evolution of knowledge with time.

To overcome these issues, different lines of research are currently under investigation. I plan to obtain a methodology to symbolically model check a richer language, including temporal and epistemic operators, by the end of this research.

In parallel, from a more theoretical point of view, I am currently studying the completeness of the $\widehat{K}_j^i$ operator and some other theoretical issues.

# 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] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proceedings of Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99), number 1579 in LNCS*, 1999.

[3] 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.

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

[5] 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.

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

[7] B. F. Chellas. *Modal Logic, an Introduction*. Cambridge University Press, Cambridge, 1980.

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

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

[10] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence, AI*, 42(2–3):213–261, March 1990.

[11] D. C. Dennett. *The intentional stance*. The MIT Press, Massachusetts, 1987. 388 pages, 1987.

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

[13] R. Goldblatt. *Logics of Time and Computation*. Center for the Study of Language and Information, Stanford, California, 2 edition, 1992.

[14] 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.

[15] W. van der Hoek and M. Wooldridge. Model checking knowledge and time. *Lecture Notes in Computer Science*, 2318, 2002.

[16] W. van der Hoek and M. Wooldridge. Towards a logic of rational agency. *Logic Journal of the IGPL*, 11(2):133–157, March 2003.

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

[18] G. E. Hughes and M. J. Cresswell. *A New Introduction to Modal Logic*. Routledge, New York, 1996.

[19] M. R. A. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, England, 2000.

[20] A. Lomuscio. *Information Sharing among Ideal Agents*. PhD thesis, School of Computer Science, University of Birmingham, February 1999.

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

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

[23] A. Lomuscio and M. Sergot. Extending interpreting systems with some deontic concepts. In J. van Benthem, editor, *Proceedings of TARK 2001*, pages 207—218, San Francisco, CA, July 2001. Morgan Kauffman.

[24] A. Lomuscio and M. Sergot. On multi-agent systems specification via deontic logic. In J.-J Meyer, editor, *Proceedings of ATAL 2001*. Springer Verlag, July 2001. To Appear.

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

[26] A. Lomuscio and M. Sergot. Violation, error recovery, and enforcement in the bit transmission problem. In *Proceedings of DEON'02*, London, May 2002.

[27] A. Lomuscio and W Penczek T. Lasica. Bounded model checking for interpreted systems: preliminary experimental results. In *Proceedings of the Second NASA Workshop on Formal Approaches to Agent-Based Systems FAABS II*, Greenbelt, MD, USA, October 2002.

[28] J. McCarthy. Ascribing mental qualities to machines. In Martin Ringle, editor, *Philosophical Perspectives in Artificial Intelligence*, pages 161–195. Humanities Press, Atlantic Highlands, New Jersey, 1979.

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

[30] S. Merz. Model checking: A tutorial overview. *Lecture Notes in Computer Science*, 2067:3–??, 2001.

[31] 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.

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

[33] M. Müller-Olm, D. Schmidt, and B. Steffen. Model-checking: A tutorial introduction. In Agostino Cortesi and Gilberto Filé, editors, *Static Analysis*, volume 1694 of *Lecture Notes in Computer Science*, pages 330–354. Springer, 1999.

[34] W. Penczek, B. Wozna, and A. Zbrzezny. Bounded model checking the universal fragment of CTL. *Fundamenta Informaticae*, 50:1–22, 2002.

[35] F. Raimondi. Model checking epistemic properties of interpreted systems. In *Proceedings of ESSLLI03 - Student Session*, August 2003.

[36] F. Raimondi and A. Lomuscio. A tool for specification and verification of epistemic and temporal properties of multi-agent system. In *Proceedings of LCMAS, Workshop on Logic and Communication in Multi-Agent Systems*, Eindhoven, June 2003.

[37] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logic in Computer Science (LICS'86)*, pages 332–345, Washington, D.C., USA, June 1986. IEEE Computer Society Press.

[38] M. Wooldridge. Computationally grounded theories of agency. In E. Durfee, editor, *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS 2000)*. IEEE Press, July 2000.

[39] M. Wooldridge. *Reasoning about Rational Agents*. Intelligent Robots and Autonomous Agents. The MIT Press, Cambridge, Massachusetts, 2000.

[40] M. Wooldridge, M. 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.

[41] M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995.