Multiagent Systems Verification via Model Checking

Massimo Benerecetti[1], Fausto Giunchiglia[1,2] Luciano Serafini[2]
[1] DISA - Universitá degli Studi di Trento,
Via Inama 5, 38100 Trento, Italy
[2] IRST - Istituto Trentino di Cultura,
38050 Povo, Trento, Italy

Abstract

Model checking is a very successful technique which has been applied in the design and verification of finite state concurrent reactive processes. In this paper we show how this technique can be lifted to be applicable to multiagent systems. Our approach allows us to reuse the technology and tools developed in model checking, to design and verify multiagent systems in a modular and incremental way, and also to have a very efficient model checking algorithm.

1 Introduction

Model checking is a very successful automatic technique which has been devised for the design and verification of finite state reactive systems, e.g., sequential circuit designs, communication protocols, and safety critical control systems (see, e.g., [1]). There is evidence that model checking, when applicable, is far more successful than the other approaches to formal methods and verification (e.g., first order or inductive theorem proving, tableau based reasoning about modal satisfiability).

In this paper we show how model checking can be "lifted" to become applicable to multiagent systems in a way to *(i)* reuse with almost no variations all the technology and tools developed in model checking; and *(ii)* allow for a modular design and verification of multiagent systems. The first feature allows us to exploit the huge amount of expertise, technology and tools developed in model checking. The second feature is necessary in order to deal with real world complex systems (see [3] for a discussion on this topic).

Model checking allows us to model concurrent reactive finite state *processes*. We model *agents* as concurrent reactive non-terminating finite state processes able to have what we call BDI attitudes, i.e., beliefs, desires and intentions. The specification of an agent has therefore two orthogonal aspects: a temporal aspect and a "mental attitudes" aspect. The key idea underlying our approach is to keep these two aspects separated. In practice things work as follows:

- when we consider the temporal evolution of an agent we treat BDI atoms (i.e. atomic formulas expressing belief, desire, or intention) as atomic propositions. The fact that these formulas talk about BDI attitudes is not taken into consideration.
- We deal with BDI attitudes as follows. The fact that an agent a_1 has BDI attitudes about another agent a_2 is modeled as the fact that a_1 has access to a representation of a_2 as a process (one representation for each BDI attitude). Then, any time it needs to verify the truth value of some BDI atom about a_2, e.g., $B_2\mathsf{AF}\,\phi$, a_1 simply tests whether, e.g., $\mathsf{AF}\,\phi$ holds in its (appropriate) representation of a_2. BDI attitudes are essentially used to control the "jumping" among processes. This operation is iterated in the obvious way in case of nested BDI attitudes.

```
SENDER:
initial state
    all propositions are false
loop
    read($p$)
    if $p \wedge \neg B_r p$ then
        put-msg(inform($s, r, p$))
    if $\neg p \wedge \neg B_r \neg p$ then
        put-msg(inform($s, r, \neg p$))
    get-msg($m$)
    if $m$ =inform($r, s, B_r p$) then
        $B_r p := True$; $B_r \neg p := False$;
    if $m$ =inform($r, s, B_r \neg p$) then
        $B_r \neg p := True$; $B_r p := False$;
endloop
```

```
RECEIVER:
Initial state
    $p = False$;  $m = \langle null \rangle$
loop
    get-msg($m$)
    if $m$ =inform($s, r, p$) then
        $p := True$;
        put-msg(inform($r, s, B_r p$))
    if $m$ =inform($s, r, \neg p$) then
        $p := False$;
        put-msg(inform($r, s, B_r \neg p$))
endloop
```

Figure 1: The s and r's algorithms.

The paper is structured as follows. In Section 2 we describe a motivating example which we then formalize and study throughout the rest of the paper. The basic ingredients of model checking are: (i) a propositional temporal logic used to write specifications; (ii) a language for describing the system (i.e., the set of processes) to be verified as a finite state automaton; and (iii) a model checking procedure which efficiently and automatically determines whether the specifications are satisfied by the state-transition graph generated by the system automaton. Sections 3, 4 and 5 describe these three ingredients for multiagent model checking. The description is given incrementally over the standard model checking notions. In particular, we adopt CTL [1] as the propositional temporal logic used to state specifications. We conclude with a discussion of an achievement and the related work (Section 6).

2 A motivating example

Let us consider the following scenario involving two agents: a receiver r and a sender s. s continuously reads news on a certain subject from its sensors (e.g., the standard input). Once read the news, s informs r only if it believes that r does not have the correct knowledge about that subject (this in order to minimize the traffic over the network). Once received the news, r acknowledges this fact back to s.

We implement this scenario using a FIPA compliant [2] architecture. We have therefore three agents: s, r, and a network (communication protocol) which allows them to interact. Figures 1, 2 give the algorithmic descriptions of s, r and the communication protocol, respectively in a `Promela`-like language [4] [1]. In these algorithms, the news subject of the information exchange is the truth value of the propositional atom p. inform(s,r, p) returns a message with sender s, receiver r, and content p (inform is a FIPA primitive). put-msg and get-msg are the primitives for putting and getting (from the communication channel) a message. read allows for reading from the standard input. B_r is the operator used to represent the beliefs of r as perceived by the other agents, and dually for B_s. Notice that the communication protocol has beliefs about r and s and therefore must have a representation of how they behave. We suppose that this representation coincides with what r and s actually are, as described in Figure 1. This allows us to model the fact that the communication protocol behaves correctly following what s and r do. There is no nesting of belief operators and, therefore, there is no need of further

[1] `Promela` is the input language of SPIN and, indirectly, also of other model checkers.

```
PROTOCOL:
initial state
    all propositions are false
loop
    set all propositions to false
    do-one-of
        begin
            $B_s \mathsf{AF}\, Do(\text{put-msg}(\text{inform}(s, r, p))) = True$;
            $B_r \mathsf{AF}\, Do(\text{get-msg}(\text{inform}(s, r, p))) = True$;
        end
        begin
            $B_s \mathsf{AF}\, Do(\text{put-msg}(\text{inform}(s, r, \neg p))) = True$;
            $B_r \mathsf{AF}\, Do(\text{get-msg}(\text{inform}(s, r, \neg p))) = True$;
        end
    end
    set all propositions to false
    do-one-of
        begin
            $B_r \mathsf{AF}\, Do(\text{put-msg}(\text{inform}(r, s, B_r p))) = True$;
            $B_s \mathsf{AF}\, Do(\text{get-msg}(\text{inform}(r, s, B_r p))) = True$;
        end
        begin
            $B_r \mathsf{AF}\, Do(\text{put-msg}(\text{inform}(r, s, B_r \neg p))) = True$;
            $B_s \mathsf{AF}\, Do(\text{get-msg}(\text{inform}(r, s, B_r \neg p))) = True$;
        end
endloop
```

Figure 2: The communication protocol algorithm.

representations. s also has beliefs about r. We suppose that s (which in principle does not know anything about how r works) only knows that r can be in one of two states, with p being either true or false. In Figure 2, $B_s \mathsf{AF}\, Do($`<statement>`$)$ ($B_r \mathsf{AF}\, Do($`<statement>`$)$) intuitively means that s (r) will necessarily reach a state in which it will have just performed the action corresponding to `<statement>`. The algorithm in Figure 2 codifies the fact that the protocol implements the information flow between s and r, and the fact that it always delivers the messages it is asked to deliver.

3 The basic idea

Finite state processes can be modeled as finite state machines. In order to model processes we will employ the logic CTL, a propositional branching-time temporal logic which has been widely used in modeling finite state processes [1]. Let us consider in turn the language and semantics. Given a set P of propositional atoms, the set of CTL formulas ϕ is defined inductively as follows:

$$\phi, \psi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \mathsf{EX}\,\phi \mid \mathsf{A}\,(\phi\, \mathcal{U}\, \psi) \mid \mathsf{E}\,(\phi\, \mathcal{U}\, \psi)$$

where $p \in P$. $\mathsf{EX}\,\phi$ intuitively means that there is a path such that ϕ will be true in the next step; $\mathsf{A}\,(\phi\, \mathcal{U}\, \psi)$ means that ψ will be true in a state in the future and that ϕ will be true in all the states before, for all paths;

$\mathsf{E}\,(\phi\,\mathcal{U}\,\psi)$ means that there exists a path such that ψ will be true in a state in the future and that ϕ will be true in all the states before. The following abbreviations are used:

$$
\begin{array}{ll}
\bot \overset{\text{def}}{=} p \wedge \neg p & \top \overset{\text{def}}{=} \neg\bot \\
\phi \supset \psi \overset{\text{def}}{=} \neg(\phi \wedge \neg\psi) & \mathsf{AF}\,\phi \overset{\text{def}}{=} \mathsf{A}\,(\top\,\mathcal{U}\,\phi) \\
\mathsf{EF}\,\phi \overset{\text{def}}{=} \mathsf{E}\,(\top\,\mathcal{U}\,\phi) & \mathsf{AG}\,\phi \overset{\text{def}}{=} \neg\mathsf{E}\,(\top\,\mathcal{U}\,\neg\phi) \\
\mathsf{EG}\,\phi \overset{\text{def}}{=} \neg\mathsf{A}\,(\top\,\mathcal{U}\,\neg\phi) & \mathsf{AX}\,\phi \overset{\text{def}}{=} \neg\mathsf{EX}\,\neg\phi
\end{array}
$$

The semantics for CTL formulas is the standard branching-time temporal semantics based on Kripke-structures. A CTL structure is a tuple $m = \langle S, s_0, R, L\rangle$, where S is a set states, $s_0 \in S$ is the *initial state*, R is a total binary relation on S, and $L : S \to \mathcal{P}(P)$ is a *labeling function*, which associates to each state $s \in S$ the set $L(s)$ of propositional atoms true at s. A *path* x in m is an infinite sequence of states $s_1, s_2, \cdots$ such that for every $i \geq 1$, $s_i R s_{i+1}$. Satisfiability of a formula ϕ in a CTL structure m at a state s is defined as follows:

- $m, s \models p$ iff $p \in L(s)$;
- $m, s \models \neg\phi$ iff $m, s \not\models \phi$;
- $m, s \models \phi \wedge \psi$ iff $m, s \models \phi$ and $m, s \models \psi$;
- $m, s \models \mathsf{EX}\,\phi$ iff there's a s' with sRs', such that $m, s' \models \phi$;
- $m, s \models \mathsf{A}\,(\phi\,\mathcal{U}\,\psi)$ iff for every path $x = (s = s_1, s_2, \cdots)$ there's a $k \geq 1$ such that $m, s_k \models \psi$ and, for every $1 \leq j < k$, $m, s_j \models \phi$;
- $m, s \models \mathsf{E}\,(\phi\,\mathcal{U}\,\psi)$ iff there's a path $x = (s = s_1, s_2, \cdots)$ and a $k \geq 1$ such that $m, s_k \models \psi$ and for every $1 \leq j < k$, $m, s_j \models \phi$.
- $m \models \phi$ iff $m, s_0 \models \phi$.

We build the notion of agent incrementally over the notion of process. Suppose that we have a set I of agents. Each agent has its own beliefs, desires, and intentions about itself and the other agents. We adopt the usual syntax for propositional attitudes: $B_i\phi$, $D_i\phi$ and $I_i\phi$ mean that agent i believes, desires and intends (to bring about) ϕ, respectively. B_i, D_i and I_i are called BDI operators for agent i (or simply BDI operators). O_i denotes any BDI operator for agent i. The idea is to model each nesting of BDI operators as a different process evolving over time. For example, the beliefs of agent s evolving over time in our example can be modeled by the process whose algorithm *SENDER* is given in Figure 1.

Formally, let $O = \{\textsc{b}, \textsc{d}, \textsc{i}\}$ be a set of symbols, one for each BDI attitude. OI^* denotes the set $(O \times I)^*$, i.e., the set of finite (possibly empty) strings of the form $\mathrm{o}_1 i_1 \ldots \mathrm{o}_n i_n$ with $\mathrm{o}_k \in O$ and $i_k \in I$. We call any $\alpha \in OI^*$, a *view*. Intuitively, each view in OI^* represents a possible nesting of BDI attitudes. We also allow for the empty string, ϵ. The intuition is that ϵ represents the view of an external observer which, from the outside, "sees" the behavior of the overall multiagent system. Depending on the goals, the external observer can represent the person designing the system, or a selected process of the multiagent system which is given this privileged status.

Consider the example in Section 2. We take the external observer to be the point of view of the communication protocol. Therefore, there is a view ϵ corresponding to the beliefs of the communication protocol, and also two views $\textsc{b}s$, $\textsc{b}r$ corresponding to the beliefs of the two agents s and r. Finally s has beliefs about the beliefs of r. Figure 3 graphically represents this situation. The dark circles represent the views which exist in principle, the white ones represent those which are actually needed.

An agent, e.g., s, is thus a tree of views rooted in the view that the external observer has of it, e.g., $\textsc{b}s$. Notice also that the view that an agent has of another agent is in general different from the agent itself. This allows us for instance, in the example of Section 2, to model the fact that s might have false beliefs about r.

The final step is to associate a logical language $\mathcal{L}_\alpha$ to each view $\alpha \in OI^*$. Intuitively, each $\mathcal{L}_\alpha$ is the language used to express what is true (and false) in the representation corresponding to α. Let $\{P_\alpha\}$ be a family of sets of propositional atoms. Each P_α allows for the definition of a different language (also called an MATL language

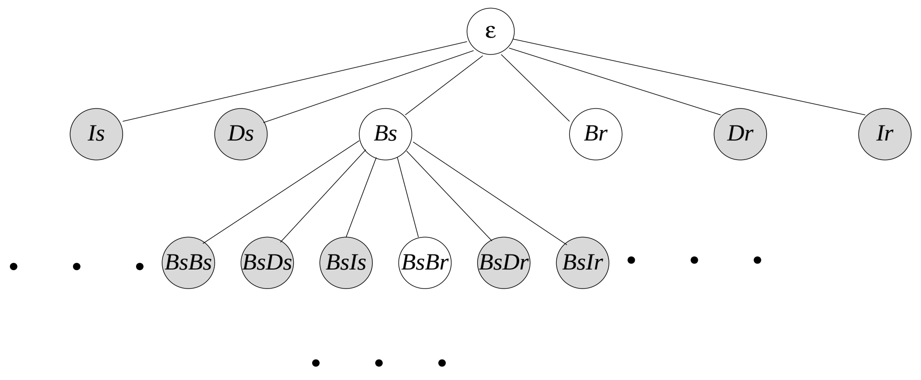

Figure 3: The set of views for the example of Section 2.

(on $\{P_\alpha\}$)). The family of MATL languages on $\{P_\alpha\}$ is the family of CTL languages $\{\mathcal{L}_\alpha\}$ where $\mathcal{L}_\alpha$ is the smallest CTL language containing the set of propositional atoms P_α and the BDI atoms $\mathrm{O}_i\phi$ for any formula ϕ of $\mathcal{L}_{\alpha \mathrm{O}i}$. In particular, $\mathcal{L}_\epsilon$ is used to speak about the whole multiagent system. Thus, intuitively, a formula $p \wedge B_i \mathsf{AG}\neg p \in \mathcal{L}_\epsilon$, (denoted by $\epsilon : p \wedge B_i \mathsf{AG}\neg p$) means that p is true and that i believes that in every future state p will be false. The languages $\mathcal{L}_{Bi}$ $\mathcal{L}_{Di}$, and $\mathcal{L}_{Ii}$ are the languages that i adopts to represent its beliefs, desires and intentions, respectively. The language $\mathcal{L}_{BiIj}$ is used to specify i's beliefs about j's intentions, and so on. Note that the only restriction on the languages is that $\mathrm{O}_i\phi$ must be an atomic formula of $\mathcal{L}_\alpha$ if and only if ϕ is a formula of $\mathcal{L}_{\alpha \mathrm{O}i}$ (see [3] for a study of how this condition can be modified in order to capture various interesting properties). We allow also for empty languages. However $\mathcal{L}_\epsilon$ cannot be empty as we need to be able to talk about the whole multiagent system.

4 Multiagent finite state machines

We are interested in extending model checking to multiagent model checking. In model checking we deal with *finite CTL structures*, i.e., those CTL structures which have a finite set of states, and also a labeling function mapping to a finite number of atoms. The crucial observation is that finite CTL structures can be seen as *finite state machines* (FSMs), an FSM being an object $f = \langle S, s_0, R, L\rangle$ (with everything finite). Our solution is to extend the notion of FSM to that of *MultiAgent Finite State Machine (MAFSM)*, where, roughly speaking, a MAFSM is a finite set of FSMs.

A first step in this direction is to restrict ourselves to a finite number of views α. However this is not enough as also a finite set of views allows for an infinite number of BDI atoms. Even if we have a finite number of processes we cannot model them as FSMs. We solve this problem by introducing the notion of explicit BDI atom. Formally if $\{\mathcal{L}_\alpha\}$ is a family of MATL languages, then $Expl(\textsc{o}i, \alpha)$ is a (possibly empty) *finite* subset of the BDI atoms of $\mathcal{L}_\alpha$. The elements of $Expl(\textsc{o}i, \alpha)$ are called *explicit BDI atoms.* We have the following.

Definition 4.1 *Let $\{\mathcal{L}_\alpha\}$ be a family of MATL languages on $\{P_\alpha\}$. A* MultiAgent Finite State Machine *(MAFSM) $F = \{F_\alpha\}$ for $\{\mathcal{L}_\alpha\}$ is a recursive total function such that:*

1. *$F_\epsilon \neq \emptyset$;*
2. *for all views $\alpha \in OI^n \subset OI^*$ with OI^n finite, it associates a finite set F_α of FSMs on the MATL language on the following atoms: P_α, $Expl(\textsc{b}i, \alpha)$, $Expl(\textsc{d}i, \alpha)$ and $Expl(\textsc{i}i, \alpha)$, for all $i \in I$;*
3. *for all the views $\alpha \in OI^* \setminus OI^n$, $F_\alpha = \emptyset$.*

The first condition is needed as otherwise there is nothing we can reason about; the second allows us to deal with finite views, and the third allows us to deal with finite sets of atoms.

Example 4.1 Let us construct the MAFSM of the example in Section 2. Let us concentrate on a process and the corresponding view. The propositional atoms are all the propositional atoms which appear in the `Promela`-like specification. $\mathrm{O}_i\phi$ is a BDI atom of $\mathcal{L}_\alpha$ if and only if ϕ is a formula of $\mathcal{L}_{\alpha \mathrm{O}i}$. The set of explicit BDI atoms contains all the BDI atoms which are set in the `Promela`-like specification. The intuition is that explicit BDI atoms can change independently, while the value of implicit BDI atoms can only change as a consequence of changes of value of explicit BDI atoms. This identifies the set of state variables and therefore the space of possible states. The value of the state variables in the initial state can be extracted from the top part of the `Promela`-like specification.

Let us detail the FSMs corresponding to the processes of our example. Any `Promela`-like construct can be directly translated, with a one-to-one mapping, into transitions of a finite state automaton. We leave to the reader the definition of the state transitions of the FSMs in the views ϵ, $\textsc{Bs}$, $\textsc{Br}$. Being the initial state completely defined there is only one FSM per view. The propositional atoms of all the views can be easily extracted from the algorithms given in Section 2. We concentrate therefore on the explicit BDI atoms.

FSMs in ϵ: F_ϵ contains only one FSM generated from *PROTOCOL*.[2]

$$Expl(\textsc{Br}, \epsilon) = \left\{ \begin{array}{l} B_r \mathsf{AF}\ Do(\text{put-msg}(\text{inform}(s, r, B_r p))), \\ B_r \mathsf{AF}\ Do(\text{put-msg}(\text{inform}(s, r, B_r \neg p))), \\ B_r \mathsf{AF}\ Do(\text{get-msg}(\text{inform}(r, s, p))), \\ B_r \mathsf{AF}\ Do(\text{get-msg}(\text{inform}(r, s, \neg p))) \end{array} \right\}$$

$$Expl(\textsc{Bs}, \epsilon) = \left\{ \begin{array}{l} B_s \mathsf{AF}\ Do(\text{get-msg}(\text{inform}(r, s, B_r \neg p))), \\ B_s \mathsf{AF}\ Do(\text{put-msg}(\text{inform}(s, r, p))), \\ B_s \mathsf{AF}\ Do(\text{put-msg}(\text{inform}(s, r, \neg p))), \\ B_s \mathsf{AF}\ Do(\text{get-msg}(\text{inform}(r, s, B_r p))) \end{array} \right\}$$

and $Expl(\textsc{Ds}, \epsilon) = Expl(\textsc{Dr}, \epsilon) = Expl(\textsc{Is}, \epsilon) = Expl(\textsc{Ir}, \epsilon) = \emptyset$.

FSMs in $\textsc{Bs}$: F_{B_s} contains only one FSM generated from *SENDER*. $Expl(\textsc{Br}, \textsc{Bs}) = \{B_r p, B_r \neg p\}$. All the other sets of explicit BDI atoms are empty.

FSMs in $\textsc{Br}$: F_{B_r} contains only one FSM generated from *RECEIVER*. $Expl(\mathrm{O}i, \textsc{Br})$ is the empty set.

FSMs in $\textsc{BsBr}$: Section 2 does not give an algorithmic specification of this view. It only says that "... s ... only knows that r can be in one of two states, with p being either true or false". This tells us that there is only one state variable p, and that $F_{B_s B_r}$ will contain all the FSMs with only one state variable (which are sixteen). This formalizes the fact that s knows nothing of the initial state and state transitions of r. □

Given the notion of MAFSM, the next step is give a notion of satisfiability in a MAFSM. We start from the notion of satisfiability of CTL formulas in an FSM at a state. This notion is defined as in CTL structures. This allows us to determine the satisfiability of all the propositional and explicit BDI atoms (and all the formulas belonging to the corresponding MATL language). For these formulas we do not need to use the machinery associated to BDI attitudes. However, this machinery is needed in order to deal with the (infinite) number of BDI atoms which are not memorized anywhere in MAFSM.

Let the set of *implicit BDI atoms* of a view α, written $Impl(\mathrm{O}i, \alpha)$, be defined as the (infinite) subset of all BDI atoms of $\mathcal{L}_\alpha$ which are not explicit BDI atoms, i.e. $Impl(\mathrm{O}i, \alpha) = \{\mathrm{O}_i\phi \in \mathcal{L}_\alpha \setminus Expl(\mathrm{O}i, \alpha)\}$. Let $ArgExpl(\mathrm{O}i, \alpha, s)$ be defined as follows.

$$ArgExpl(\mathrm{O}i, \alpha, s) = \{\phi \in \mathcal{L}_{\alpha \mathrm{O}i} \mid \mathrm{O}_i\phi \in L(s)\ \cap\ Expl(\mathrm{O}i, \alpha)\}$$

Intuitively, $ArgExpl(\mathrm{O}i, \alpha, s)$ consists of all the formulas $\phi \in \mathcal{L}_{\alpha \mathrm{O}i}$ such that the explicit BDI atom $\mathrm{O}_i\phi$ is true in s. At this point, to define the satisfiability in a MAFSM, it is sufficient to use the fact that we know how to compute $ArgExpl(\mathrm{O}i, \alpha, s)$ (it is sufficient to use CTL satisfiability and then to compare the results of the relevant CTL structures) and exploit $ArgExpl(\mathrm{O}i, \alpha, s)$ to compute the implicit BDI atoms which satisfy an appropriate correctness and completeness condition.

[2]Note that $Do(\text{put-msg}(\text{inform}(s, r, p)))$ is a propositional atom of the language of s which is true only if s is in a state in which it has just performed the action "put-msg(inform(s, r, p))". The other similar atoms have a similar intuitive meaning.

Definition 4.2 (Satisfiability in a MAFSM) *Let F be a MAFSM, α a view, $f = \langle S, s_0, R, L\rangle \in F_\alpha$ an FSM, and $s \in S$ a state. Then, for any formula ϕ of $\mathcal{L}_\alpha$, the satisfiability relation $F, \alpha, f, s \models \phi$ is defined as follows:*

1. *$F, \alpha, f, s \models p$, where p is a propositional atom or an explicit BDI atom: the same as FSM satisfiability;*
2. *satisfiability of propositional connectives and CTL operators: the same as FSM satisfiability;*
3. *$F, \alpha, f, s \models \mathrm{O}_i\phi$, where $\mathrm{O}_i\phi$ is an implicit BDI atom, iff for all $f' \in F_{\alpha \mathrm{O}i}$ and s' state of f', $F, \alpha\mathrm{o}i, f', s' \models \bigwedge \mathit{ArgExpl}(\mathrm{o}i, \alpha, s) \supset \phi$*

We have furthermore:

4. *$F, \alpha, f \models \phi$ iff $F, \alpha, f, s_0 \models \phi$;*
5. *$F, \alpha \models \phi$ iff for all $f \in F_\alpha$, $F, \alpha, f \models \phi$;*
6. *$F \models \alpha : \phi$ iff $F, \alpha \models \phi$.*

In the definition of $F, \alpha, f, s \models \phi$, item 3 is the crucial step. $\bigwedge \mathit{ArgExpl}(\mathrm{o}i, \alpha, s)$ is the conjunction of all the elements of $\mathit{ArgExpl}(\mathrm{o}i, \alpha, s)$. We need to use $\mathit{ArgExpl}(\mathrm{o}i, \alpha, s)$ in order to compute the formulas ϕ such that $\mathrm{O}_i\phi$ is an implicit BDI atom. Notice that item 3 gives to BDI operators the same strength as modal $K(3m)$, where m is the number of agents. In particular, we have that if $\Gamma \supset \phi$ is a theorem in a view then $\mathrm{O}_i\Gamma \supset \mathrm{O}_i\phi$ is a theorem in the (appropriate) view above, where $\mathrm{O}_i\Gamma$ is the set $\{\mathrm{O}_i\phi \mid \phi \in \Gamma\}$.

Item 4 states that a FSM satisfies a formula if the formula is satisfied in its initial state. Item 5 states that a formula is satisfied in a view if it is satisfied by all the FSMs of that view. Finally item 6 states that a labeled formula $\alpha : \phi$ is satisfied if ϕ is satisfied in the view corresponding to the label.

5 Model Checking a MAFSM

The basic operation of a standard CTL model checking algorithm is to extend the labeling function of an FSM (which considers only propositional atoms) to all the (atomic and not atomic) subformulas of the formula being model checked. Let us call Extended FSM (or, simply, FSM when the context makes clear what we mean) the result of this operation. The generation of an extended FSM relies on the fact that the labeling function explicitly defines the truth value of all the atoms. The problem is that in the FSMs of a MAFSM the labeling function is not defined on implicit BDI atoms, whose truth value is therefore left undefined; and that we need to know the truth values of the implicit BDI atoms occurring in the formula to be model checked. The definition of satisfiability in a MAFSM (item 3 in Definition 4.2) tells us how to fix this problem. That is, if $\mathrm{O}_i\psi$ is an implicit BDI atom, then $F, \alpha, f, s \models \mathrm{O}_i\psi$ if and only if for every $f' \in F_{\alpha\mathrm{O}i}$ and every state s' of f', we have $F, \alpha\mathrm{o}i, f', s' \models \bigwedge \mathit{ArgExpl}(\mathrm{o}i, \alpha, s) \supset \psi$.

The crucial observation is that $\mathit{ArgExpl}(\mathrm{o}i, \alpha, s)$ is generated from $\mathit{Expl}(\mathrm{o}i, \alpha)$ and the labeling functions of the FSMs; that it is a finite set; and that it is a property of the MAFSM (and thus independent of the formula to be model checked). The idea, therefore, is to precompute, once for all, and store in an appropriate data structure, this information. In particular, for each BDI operator O_i, let $C_{\mathrm{O}i}$, called the *(MAFSM) compatibility relation* of O_i, be a relation defined as follows. Let $ex \subseteq \mathit{Expl}(\mathrm{o}i, \alpha)$ be a subset of the explicit BDI atoms of a view α. Then:

$$C_{\mathrm{O}i}(\alpha, ex) = \{\langle f', s'\rangle \mid f' \in F_{\alpha\mathrm{O}i}, s' \text{ a state of } f' \text{ and } F, \alpha\mathrm{o}i, f', s' \models \{\phi \mid \mathrm{O}_i\phi \in ex\}\}$$

Starting from a view α and a set of explicit BDI atoms ex of α, $C_{\mathrm{O}i}(\alpha, ex)$ collects all the FSMs f' and states s' of f' (in the view $\alpha\mathrm{o}i$ below) which satisfy the arguments of the chosen explicit BDI atoms. We need to consider all the subsets ex of $\mathit{Expl}(\mathrm{o}i, \alpha)$ as a priori we do not know which explicit BDI atoms are relevant for

Global Variables

$\mathit{Expl}(\text{O}i,\alpha) = \{\text{O}_i\phi\}$	with $\text{O}_i\phi$ an explicit BDI atom of α;
$F = \{F_\alpha\}$	with F_α a set of FSMs on $\mathcal{L}_\alpha$;
$C_{\text{O}i} = \{\langle \alpha, ex, f', s'\rangle\}$	with $ex \subseteq \mathit{Expl}(\text{O}i,\alpha)$, f' an FSM in $F_{\alpha\text{O}i}$ and s' a state of f'.

```
Algorithm  MAMC($\alpha, \phi$)

    MAMC-View($\alpha, \{\phi\}$)
    for each $f \in F_\alpha$ do
        if $\phi \notin L(s_0)$ then return($False$)
    end
    return($True$)
end
```

```
Algorithm  MAMC-View($\alpha, \Gamma$)

    $Sub := \bigcup\{sub(\phi) \mid \phi \in \Gamma\}$
    for each $\text{O}i$ do
        $ArgImpl(\text{O}i, \alpha, Sub) := \{\phi \mid \text{O}_i\phi \in Sub \setminus Expl(\text{O}i, \alpha)\}$
        if $ArgImpl(\text{O}i, \alpha, Sub) \neq \emptyset$ then
            MAMC-View($\alpha\text{O}i, ArgImpl(\text{O}i, \alpha, Sub)$)
        endif
    end
    for each $f \in F_\alpha$ do                                    /* $f = \langle S, s_0, R, L\rangle$ */
        if $\langle Sub$ contains implicit BDI atoms$\rangle$ then
            for each $s \in S$ do
                for each $\text{O}i$ do
                    $ArgImpl(\text{O}i, \alpha, Sub) := \{\phi \mid \text{O}_i\phi \in Sub \setminus Expl(\text{O}i, \alpha)\}$
                    for each $\langle f', s'\rangle \in C_{\text{O}i}(\alpha, L(s) \cap Expl(\text{O}i, \alpha))$ do /* $f' = \langle S', s'_0, R', L'\rangle$ */
                        $ArgImpl(\text{O}i, \alpha, Sub) := ArgImpl(\text{O}i, \alpha, Sub) \cap L'(s')$
                    end
                    $L(s) := L(s) \cup \text{O}_i ArgImpl(\text{O}i, \alpha, Sub)$
                end
            end
        endif
        CTLMC($f, \Gamma$)
    end
end
```

Figure 4: The multiagent model checking algorithm.

the computation of the truth value of an implicit BDI atom. Implicit BDI atoms evaluated at different states will need different ex's.

It can be easily seen that

$$\langle f', s'\rangle \in C_{\text{O}i}(\alpha, L(s) \cap \mathit{Expl}(\text{O}i, \alpha)) iff F, \alpha\text{O}i, f', s' \models \bigwedge \mathit{ArgExpl}(\text{O}i, \alpha, s)$$

and, therefore, that

$$F, \alpha, f, s \models \text{O}_i\phi \text{ iff for all } \langle f', s'\rangle \in C_{\text{O}i}(\alpha, L(s) \cap \mathit{Expl}(\text{O}i, \alpha)),\ F, \alpha\text{O}i, f', s' \models \phi \qquad (1)$$

where $\mathrm{O}_i\phi$ is an implicit BDI atom and $L(s) \cap \mathit{Expl}(\mathrm{O}i, \alpha)$ is the set of explicit BDI atoms satisfied by state s of $f \in F_\alpha$.

The model checking algorithm relies on the following global data structures: a data structure F which contains, for each view, a set of (extended) FSMs F_α; a data structure $\mathit{Expl}(\mathrm{O}i, \alpha)$ which contains for each operator and view, the set of explicit BDI atoms $\mathrm{O}_i\phi$; a data structure C which contains for each modal operator O_i, a compatibility relation $C_{\mathrm{O}i} = \langle \alpha, ex, f', s' \rangle$. The MultiAgent Model Checking algorithm MAMC(α, ϕ) takes two arguments, namely a view α and the MATL formula $\phi \in \mathcal{L}_\alpha$ that we want to model check. MAMC(α, ϕ) returns true if $F \models \alpha : \phi$, false otherwise. Notice that we can model check any view (and therefore any subpart of the multiagent system). Thus, if we take α to be ϵ we model check the overall multiagent system; if we take α to be of length 1 we model check a single agent; if we take α to be of length 2 we model check the view that an agent has of another agent, and so on.

The algorithm of MAMC(α, ϕ) is shown in Figure 4. As a first step, MAMC(α, ϕ) calls the algorithm MAMC-View on view α and the set of formulas $\{\phi\}$. MAMC-View(α, Γ) takes in input a view α and a set of formulas $\Gamma \in \mathcal{L}_\alpha$, and labels the MAFSM with all the subformulas of the formulas in Γ. As a result, after this step, MAMC can return the appropriate truth value simply by testing whether ϕ is contained in the label set of the initial state s_0 of all the FSMs $f \in F_\alpha$ (remember, from Section 3 that an FSM satisfies a formula if and only if its initial state satisfies it).

Notationally, let $\mathit{sub}(\phi)$ denote the set of subformulas of ϕ (remember that $\mathrm{O}_i\phi$ is atomic and that, therefore, it is the only subformula of itself). Then, inside MAMC-View, we can distinguish three main phases.

Phase 1: Initialization. This phase, corresponding to the first line of the algorithm, collects in Sub all the subformulas of the formulas in Γ.

Phase 2: Model Checking implicit BDI atoms. This phase corresponds to the first loop. This loop considers in turn all the pairs $\mathrm{O}i$. For each of them, the first step is to compute the set $\mathit{ArgImpl}(\mathrm{O}i, \alpha, \mathit{Sub})$, i.e., the set of all the formulas ϕ which are arguments of the implicit BDI atoms $O_i\phi$ which are subformulas of Γ. Notice that this step is performed using the set of explicit BDI atoms $\mathit{Expl}(\mathrm{O}i, \alpha)$ and not the set of implicit BDI atoms $\mathit{Impl}(\mathrm{O}i, \alpha)$, the latter set being infinite. Notice also that the knowledge of the formulas to be model checked allows us to restrict ourselves to the *finite* set of implicit BDI atoms $\mathit{Sub} \setminus \mathit{Expl}(\mathrm{O}i, \alpha)$ which are relevant to the satisfiability of these formulas.

The second step is to call recursively MAMC-View on the view below and on the set $\Gamma = \mathit{ArgImpl}(\mathrm{O}i, \alpha, \mathit{Sub})$. In this phase MAMC-View visits the tree structure which needs to be model checked, extending the labeling functions of the visited FSMs. The leaves of this tree are the views for which there is no need to model check implicit BDI atoms, due to the fact that no more implicit BDI atoms occur in the set of formulas Γ in input.

Notice that the tree which is visited is usually a subtree of the tree constituent the MAFSM, in particular it is the subtree which contains all and only the views which are necessary to compute the implicit BDI atoms mentioned in the top level goal formula given in input to MAMC.

Phase 3: This phase is a loop over all the FSMs f of the current view α. This loop iteratively performs the following two phases:

Phase 3.1: Labeling f with the implicit BDI atoms. This phase corresponds to the second level loop. It is entered only if we are at a view for which some implicit BDI atom occurs in the input formulas Γ. Here the algorithm extends the labeling function of f with the true implicit BDI atoms $\mathrm{O}_i\phi$ occurring in Γ. This step is computed according to Definition (1) of satisfiability of implicit BDI atoms. Thus $L(s) \cap \mathit{Expl}(\mathrm{O}i, \alpha)$ is the set of true explicit BDI atoms in a state s of the FSM f of view α. $\langle f', s' \rangle$ is any FSM f' and state s', where f' in view $\alpha\mathrm{O}i$ is compatible with the true explicit BDI atoms computed before. The innermost loop computes and stores in $\mathit{ArgImpl}(\mathrm{O}i, \alpha, \mathit{Sub})$ the arguments of the implicit BDI atoms which occur in Sub and which are true in all the pairs $\langle f', s' \rangle$. $\mathrm{O}_i\mathit{ArgImpl}(\mathrm{O}i, \alpha, \mathit{Sub})$ at the end of the loop is the set of implicit BDI atoms true in the current state s. This set is therefore used to extend the labeling of s.

Phase 3.2: Model checking f. At this point every state s in the current f has been labeled with all the atoms (i.e, propositional atoms, explicit and implicit BDI atoms) occurring in Γ. Therefore, it is sufficient to apply the usual CTL model checking algorithm CTLMC(f, Γ). CTLMC(f, Γ) takes in input an (Extended) FSM f and a set of formulas Γ and extends the labeling function of f to all the subformulas of Γ (see for instance [1]). Notice that we can call any state-of-the-art model checker as a black box.

The following result states that MAMC-View actually solves the model checking problem for MATL.

Theorem 5.1 (Correctness of MAMC-View) *Let $f = \langle S, s_0, R, L \rangle \in F_\alpha$, with α any view, and s a state in S. $F, \alpha, f, s \models \phi$ iff MAMC-View($\alpha, \{\phi\}$) applied to F constructs a new F such that $\phi \in L(s)$, with s state of f.*

6 Conclusion

In this paper we have defined a model-checking based decision procedure for multiagent systems. Our approach allows us to reuse the technology and tools (!) developed in model checking and to specify multiagent systems incrementally.

The closest work to ours is the work by Rao & Georgeff [5]. Similarly to us, they employ a class of logics obtained by combining branching-time temporal logics (e.g., CTL), with logics for BDI attitudes. The resulting logics are relatively similar to ours. The main (essential) difference is that, in their approach, a multiagent system is specified by using a unique language. Similarly, their semantics consists of a unique Kripke structure with a temporal accessibility relation and one accessibility relation for each BDI attitude. This "having everything in a single pot" makes them lose some of our properties, in particular: the modularity and incrementality of the specification, but also the structural correspondence we have between the agents' specification and the structure of the model. Notice that, together being important per se, these two features give us important advantages in the definition of the model checking algorithm. In particular: we can deal very naturally with the case of bounded nesting; we can implement multiagent model checking by directly calling, as a subroutine, the standard model checking algorithm; and we can implement an algorithm which visits the smallest possible submodel (this last property does not seem to be possessed by the algorithm in [5] — as they need to label all the worlds in the model). Finally, we also improve on their work as they don't face the problem of the automatic generation of a model starting from the specification of a multiagent system. Notice that, at the current state-of-the-art, we can take a standard model checking language (e.g., `Promela`) and compiler, and extend them so that it becomes possible to generate models of multiagent systems.

References

[1] E. Clarke, O. Grumberg, and D. Long. Model Checking. In *Proceedings of the International Summer School on Deductive Program Design*, Marktoberdorf, Germany, 1994.

[2] FIPA Foundation for Intelligent Physical Agents. Fipa '97 draft specification, 1997. Revision 2.0 available at `http://drogo.cselt.stet.it/fipa/`.

[3] E. Giunchiglia and F. Giunchiglia. Ideal and Real Belief about Belief. In *Practical Reasoning, International Conference on Formal and Applied Practical Reasoning, FAPR'96*, number 1085 in Lecture Notes in Artificial Intelligence, pages 261–275. Springer Verlag, 1996.

[4] G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1991.

[5] A. S. Rao and M. P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 318–324, Chambéry, France, 1993.