A Multi-Agent Framework Based on Communication and Concurrency

Jamshid Bagherzadeh M. and S. Arun-Kumar

Indian Institute of Technology Delhi, 110016, New Delhi, India
{jamshid, sak}@cse.iitd.ernet.in

Abstract. Multi-agent systems are receiving attention nowadays, as they can be applied in a variety of disciplines like industry, e-commerce, control systems, etc. As the areas of usage are more crucial the matter of assurance of correctness of programs is important. However, verification of these systems has received attention only very recently. Hitherto such systems have been programmed without a clearly defined formal semantics and with very little idea about formally expressed verifiable properties to be satisfied by such systems. In this paper a new multi-agent programming language, called ECCS is defined. Its operational semantics is defined and a model checking algorithm is described which verifies properties expressed in the logic ACTL.
Keywords: Multi-agent systems, Verification, Model checking, CCS, CTL.

1 Introduction

A multi-agent system consists of various agents, which communicate with each other through an agent communication language (ACL), to fulfill their individual goals (intentions). Each agent may also have its own set of beliefs, about other agents and the environment. However each agent is *rational* and *logically consistent* with regard to its beliefs, goals and uncertainties and is capable of deriving the logical consequences of its attitudes. The problem of agent communication verification is to check whether agents that use ACL, conform to its semantics, i.e. when an agent sends a message (communicative act) to other agents, whether it complies with the semantics of the message or not. For example when agent a sends a message $inform(\omega)$ to agent b, it has to conform with the semantics of $inform$ or simply satisfy the conditions of the message.

In this document we define a framework for agent programming and communication. We use CCS [12] as an agent programming language, and extend it with some primitive communicative acts (message types) from FIPA-ACL [8]. We define a structural operational semantics [14] of the language. We then introduce a variant of the logic CTL [2, 3] called ACTL[6], to define the properties of agent systems. Finally we use a model checking algorithm to verify whether programs comply with the defined specifications.

This document is organized as follows. In section 2 we show the structure of the information store. Section 3 defines the syntax of programming language

ECCS. We define the semantics of ECCS in section 4 and the syntax and semantics of the logic ACTL in section 5. In section 6 we present our model checking algorithm. In section 7 we give a small example, and section 8 is the conclusion.

2 Information Store (IS)

Let $Ag = \{1, ..., n\}$ be a set of agents. We use three modal operators B, I and U standing for belief, intention and uncertainty respectively, which represent the attitudes of the individual agents. We assume the formal semantics of beliefs and intentions are defined in a Kripke structure $M = (S, B_1, \ldots, B_n, I_1, \ldots, I_n, L)$ where S is the set of states and L is a labeling of states. B_i and I_i are belief and intention accessibility relations for agent i defined as $B_i \subseteq S \times S$ and $I_i \subseteq S \times S$. Agent i, in state s, believes ϕ, i.e. $s \models B_i\phi$, iff for all s' s.t. $(s, s') \in B_i$, $s' \models \phi$. The semantics for intention modal operator is defined in a similar fashion. As usual, we assume B_i is serial, transitive and euclidean, and I_i is serial for any $i \in Ag$. U_i is not defined as a separate relation, but is interpreted in terms of B_i. In FIPA, $U_i\phi$ is defined as,"i is uncertain about ϕ, but thinks ϕ is more likely than $\neg\phi$". Assuming this and discussions in [9], we formally interpret uncertainty in the Kripke structure based on belief relations. Consider $S' = \{t | (s, t) \in B_i\}$. We define $s \models U_i\phi$, if $s' \models \phi$ for a majority of states $s' \in S'$, and at least one state $s' \in S'$, $s' \not\models \phi$. Here **majority** means, more than half of S' if S' is finite, and all but a finite number of S' if S' is infinite. The belief relations satisfy the axioms of the logic KD45 and intention relations satisfy those of the logic KD. These axioms for an attitude O_i are:

K: $\vdash O_i(F \Rightarrow G) \Rightarrow (O_iF \Rightarrow O_iG)$ **4:** $\vdash O_iF \Rightarrow O_iO_iF$
D: $\vdash O_iF \Rightarrow \neg O_i\neg F$ **5:** $\vdash \neg O_i\neg F \Rightarrow O_i\neg O_i\neg F$.

When $O_i = B_i$ all axioms **K, D, 4** and **5** hold. For $O_i = I_i$ only axioms **K** and **D** hold. Based on the above definition we can see that when $O_i = U_i$ only axiom **D** will hold. Also based on the definition of belief relations, $\vdash B_iB_iF \Rightarrow B_iF$ and $\vdash B_i\neg B_i\neg F \Rightarrow \neg B_i\neg F$ hold for any $i \in Ag$.

Let $\mathcal{O} = \{B, U, I, \neg B, \neg U, \neg I\}$ be a set of symbols. Let $\mathbf{V}$ be the set $(\mathcal{O} \times Ag)^*$, i.e., the set of finite strings of the form $M_{1\,i_1} \ldots M_{n i_n}$ with $M_k \in \mathcal{O}$ and $i_k \in Ag$. We call any $v \in \mathbf{V}$, a view. Intuitively, each view in $\mathbf{V}$ represents a possible nesting of mental attitudes. For example, the view B_i contains beliefs of agent i, B_iB_j contains beliefs of i about beliefs of j, and so on. Considering this definition, $B_iB_jI_k\omega$ shows that agent i believes that agent j believes that agent k intends to make ω true. The Information store of each agent is like a tree, and any view is a node of the tree. The IS of a multi-agent system is a collection of trees (Figure 1). We define **AtomProp** as a set of atomic propositions. A **literal** is an atomic proposition or its negation. The set **ModFor** of *modal formulas* is defined by the following grammar:

$$\omega ::= p \mid O_i\omega \mid \neg p \mid \neg O_i\omega$$

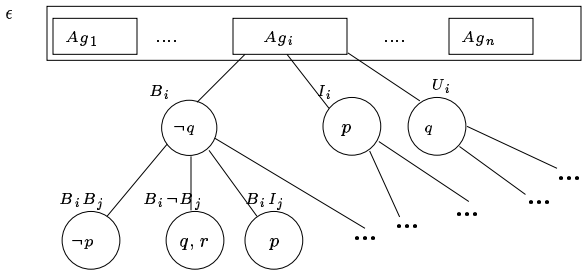

Fig. 1. Information store as a tree, with nodes representing views.

where $O \in \{B, I, U\}$. We associate a subset of **ModFor** to each view $v \in \mathbf{V}$. Note that literals are stored explicitly in any view of the agent's information tree except the view $v = \epsilon$. $O_i\omega$ ($\neg O_i\omega$) is an implicit formula of view v if ω is a formula of view vo_i ($v\neg o_i$). We may imagine the information store of a multi-agent system as a function Ψ, where

$$\Psi \ : \ \mathbf{V} \longrightarrow \ \wp(\mathbf{ModFor})$$

Some of the notations which are used in the paper are: Ψ denotes IS of multi-agent system, $\Psi_i = \Psi_i(\epsilon)$ denotes IS of agent i, $\Psi_{O_i} = \Psi_i(O_i) = \Psi(O_i)$ is the subtree of agent i, rooted at O_i. We assume ψ is any subtree of Ψ, and $\psi_{O_i}(v) = \psi(O_i v)$ (for example $\Psi_{B_i}(B_j)$ and $\Psi(B_iB_j)$ denote the same subtree).

We have views $\neg O_i$ as well as views O_i. It must be noted that at the top level (Ψ_i), we have assumed a closed world and only three nodes B_i, U_i and I_i are present, and we don't have $\neg B_i$, $\neg U_i$, and $\neg I_i$. At this level, we assume $\Psi_i \models \neg O_i\omega$ iff $\Psi_i \not\models O_i\omega$. At the lower levels of tree however, we have $\neg O$ nodes also. Thus for example $\Psi_{B_i} \models \neg O_j\omega$ iff $\Psi_{B_i\neg O_j} \models \omega$.

2.1 The Satisfaction Relation

Based on the preceeding discussion, we may derive the following relationships between the various modalities. These relationships merely emphasize the rationality and the logical consistency of the mental attitudes of each agent.

1. $B_i\omega \Rightarrow B_i\omega'$ if $\omega \Rightarrow \omega'$
2. $I_i\omega \Rightarrow I_i\omega'$ if $\omega \Rightarrow \omega'$
3. $B_i\omega \Rightarrow \neg B_i\neg\omega$
4. $I_i\omega \Rightarrow \neg I_i\neg\omega$
5. $U_i\omega \Rightarrow \neg U_i\neg\omega$
6. $B_i\omega \Rightarrow \neg U_i\omega$
7. $U_i\omega \Rightarrow \neg B_i\omega$
8. $B_i\omega \Rightarrow \neg U_i\neg\omega$
9. $U_i\omega \Rightarrow \neg B_i\neg\omega$
10. $B_i\omega \Leftrightarrow B_iB_i\omega$
11. $\neg B_i\omega \Leftrightarrow B_i\neg B_i\omega$

Rules 1 to 4 and 10 and 11 follow from **K, D, 4** and **5**. Rules 5-9 follow from the definition of U_i modality. The IS of a multi-agent system should not violate the above rules. So when updating the information store, its consistency must be maintained somehow.

Definition 1 *Each node of the IS represents a view $v \in \mathbf{V}$. A node or view is said to be of* **uncertain parity** *if some U or $\neg U$ ocuurs in v. All other nodes*

are of **certain parity**. *Further a node of certain parity is* **even** *if there are an even number of* $\neg B$ *and* $\neg I$ *modalities in* v, *and is* **odd** *otherwise.*

Theorem 2. *If* $F \Rightarrow G$. *Then for any view* $v = O_1 \ldots O_k \in \mathbf{V}$, *of certain parity,* $vF \Rightarrow vG$ *if* v *is even, and* $vG \Rightarrow vF$ *if* v *is odd.* □

Let **For** (which we call *local formulas*) be the set of formulas ϕ defined as:

$$\phi ::= O_i\omega \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \neg\phi \qquad \textit{where } \omega \in \mathbf{ModFor}$$

Definition 3 *A formula* ω *is* **accessible** *from information tree* Ψ_i, *i.e.,* $\Psi_i \models_{acc} \omega$, *if it is stored explicitly in* Ψ_i. *This relation is formally defined as:*

$\Psi_i \models_{acc} \omega$ *iff* $\Psi_i, \epsilon \models_{acc} \omega$
$\Psi_i, v \models_{acc} l$ *iff* $l \in \Psi_i(v)$ *where* l *is a literal*
$\Psi_i, v \models_{acc} O_i\omega$ *iff* $\Psi_i, vO_i \models_{acc} \omega$ *where* $O \in \{B, I, U, \neg B, \neg I, \neg U\}$

Definition 4 *A formula* ω *is* **derivable** *from* Ψ_i, *i.e.,* $\Psi_i \vdash \omega$, *iff there is a formula* ϕ *s.t.* $\Psi_i \models_{acc} \phi$, *and* $\phi \Rightarrow \omega$ *using rules 1-11 and theorem 1. Note that Any accessible formula is also derivable.*

Definition 5 (Satisfaction) *A formula* ϕ *is satisfiable in* Ψ_i, *iff* ϕ *is derivable by* Ψ_i. *Satisfaction relation* $\models$ *for different formulas is defined as:*

- $\Psi_i \models O_j\omega$ *iff* $\Psi_i \vdash O_j\omega$
- $\Psi_i \models \phi_1 \wedge \phi_2$ *iff* $\Psi_i \models \phi_1$ *and* $\Psi_i \models \phi_2$
- $\Psi_i \models \phi_1 \vee \phi_2$ *iff* $\Psi_i \models \phi_1$ *or* $\Psi_i \models \phi_2$
- $\Psi_i \models \neg\phi$ $\Psi_i \not\models \phi$

2.2 Information Revision

Belief and goal revision is an important subject in light of non-monotonicity in the behaviour of agents in their interaction with other agents. Agents need to add new information and remove the old information and they like to have consistent information.

Any agent is capable of updating its own local information but not that of any other agent. However, an agent may seek to influence the attitudes of other agents by communication. As discussed earlier, we like properties 1-9 to be satisfied in information store. So when updating the information store, we will regard them as consistency rules.

Definition 6 *The information store of agent* i *(*Ψ_i*) is consistent if the following holds*

$$\mathbf{r}: \quad If\ \Psi_i \models O_i\omega\ then\ \Psi_i \not\models O_i\neg\omega \quad (O \in \{B, I, U\})$$

Information store of the multi-agent system is consistent if the information store of individual agents are consistent. To update the information store, we define function $upd(\Psi_i, v, \omega)$, where Ψ_i is the information store of agent i, v is a view of

Ψ_i and $\omega \in$ **ModFor** is a formula. This function is given in table 1. We assume $O \in \{B, U, I\}$, l is a literal, $cond \equiv$ *"if v contains only B and I modalities"*, and $M, N \in \{B, U\}$ s.t. in any formula including M and N either $M = B$ and $N = U$, or $M = U$ and $N = B$. The table includes two parts, first part shows adding of a formula and the second part shows deletion of a formula from IS. We prefix a formula with the symbol $\sim$ to denote its deletion. It is supposed

Table 1. IS revision

command	primary update	if cond Secondary update
$upd(\Psi_i, v, l)$	$= \Psi_i(v) \cup \{l\}$;	if $cond$ then $\Psi_i(v) - \{\neg l\}$.
$upd(\Psi_i, v, B_j B_j \omega)$	$= upd(\Psi_i, v, B_j \omega)$.	
$upd(\Psi_i, v, B_j \neg B_j \omega)$	$= upd(\Psi_i, v, \neg B_j \omega)$.	
$upd(\Psi_i, v, M_j \omega)$	$= upd(\Psi_i, v M_j, \omega)$;	if $cond$ then [$upd(\Psi_i, v, \sim \neg M_j \omega)$; $upd(\Psi_i, v, \sim N_j \omega)$; $upd(\Psi_i, v, \sim N_j \neg \omega)$]
$upd(\Psi_i, v, I_j \omega)$	$= upd(\Psi_i, v I_j, \omega)$;	if $cond$ then $upd(\Psi_i, v, \sim \neg I_j \omega)$.
$upd(\Psi_i, v, \neg O_j \omega)$	$= upd(\Psi_i, v \neg O_j, \omega)$;	if $cond$ then $upd(\Psi_i, v, \sim O_j \omega)$.
$upd(\Psi_i, v, \sim l)$	$= \Psi_i(v) - \{l\}$.	
$upd(\Psi_i, v, \sim M_j \omega)$	$= upd(\Psi_i, v M_j, \sim \omega)$;	if odd then [$upd(\Psi_i, v, \sim \neg M_j \neg \omega)$; $upd(\Psi_i, v, \sim \neg N_j \omega)$; $upd(\Psi_i, v, \sim \neg N_j \neg \omega)$]
$upd(\Psi_i, v, \sim I_j \omega)$	$= upd(\Psi_i, v I_j, \sim \omega)$;	if odd then $upd(\Psi_i, v, \sim \neg I_j \neg \omega)$.
$upd(\Psi_i, v, \sim \neg M_j \omega)$	$= upd(\Psi_i, v \neg M_j, \sim \omega)$;	if $even$ then [$upd(\Psi_i, v, \sim M_j \neg \omega)$; $upd(\Psi_i, v, \sim N_j \omega)$; $upd(\Psi_i, v, \sim N_j \neg \omega)$]
$upd(\Psi_i, v, \sim \neg I_j \omega)$	$= upd(\Psi_i, v \neg I_j, \sim \omega)$;	if $even$ then $upd(\Psi_i, v, \sim I_j \neg \omega)$.

$upd(\Psi_i, \omega)$ denotes $upd(\Psi_i, \epsilon, \omega)$. This function satisfies consistency rule r, when it updates the IS. We can extend the function upd to update a sequence of modal formulas as $upd(\Psi_i : \omega_1, \omega_2, ..., \omega_n) = upd(...upd(upd(\Psi_i, \omega_1), \omega_2)..., \omega_n)$. Finally we have the following theorem. Its proof is omitted because of space limit.

Theorem 7. *The function upd preserves property r.*

3 Syntax of Agent Programming Language

We use CCS [12] as the basis of our agent language and extend it with some new features, to obtain ECCS. The formal syntax of ECCS (Extended CCS) is as follows:

$$P = 0 \mid \bar{c}(\alpha).P \mid c(\alpha).P \mid update(O_i\omega).P \mid query(\phi).P \mid observe(p).P \mid P_1 + P_2 \mid A$$

The intuitive meaning of different constructs are as in CCS [12]. In the above definition, P, P_1 *and* P_2 are ECCS agents. Operators $\bar{c}(\alpha)$ and $c(\alpha)$ are used for sending and receiving information between agents respectively, where c is an unidirectional communication channel between two agents (with an input port c and output port $\bar{c}$) and α is one of $\{$ $inform(\omega)$, $confirm(\omega)$, $disconfirm(\omega)$, $request(a)\}$. These performatives will be explained in the sequel. Operators

$update(O_i\omega)$ and $query(\phi)$ are used for updating and querying $O_i\omega$ and ϕ in the information store, where $\omega \in$ **ModFor** and $\phi \in$ **For**. $observe(p)$ is used to observe some information from the environment and p is an atomic proposition.

The meaning of the composition operators is as usual. $a.P$ is an agent that can perform action a and then become the agent P, $P_1 + P_2$ is choice, which means either P_1 or P_2 will be executed. Finally we assume $Labels$ is a set of labels and $A \in Labels$ is a name which stands for an ECCS program. Labels are used for defining recursive procedures.

A multi-agent system is defined as a parallel composition of various agents which can communicate with each other. In the following $P_i (i \in Ag)$ are agents, and Ψ_i are information stores.

$$M = < P_1, \Psi_1 > \ | \ < P_2, \Psi_2 > | \ ... \ | < P_n, \Psi_n >$$

3.1 FIPA's performatives and axioms

We consider four FIPA [8] primitive performatives.The syntax of each is of the form $< s, act(r, \alpha) >$, where s is the sender, r is the receiver, act is the type of action, and α is the content of the message, which can be for example an action a asking r to do something specific, or a proposition ω. The semantics of each communicative act consists of Feasibility Preconditions (FP), which need to be satisfied before the act is performed, and Rational Effect (RE), which is the effect expected after the act is performed. In this definition $Bif_r(\omega) = B_r\omega \vee B_r\neg\omega$, and $Uif_r(\omega) = U_r\omega \vee U_r\neg\omega$. $FP(a)[s\backslash r]$ denotes the part of the FPs of a which are mental attitudes of s. $Agent(r, a)$ means that agent r can perform action a, and $Done(a)$ means that action a is done (if $B_i Done(a)$) or intended to be done (if $I_i Done(a)$). These terms will be discussed later in section 4.1.

In addition to the performatives, in FIPA some axioms have been defined too. We don't discuss these axioms here, but as discussed in [13] there are some problems using only the set of performatives and axioms defined by FIPA. We don't explain these problems here, but the interested reader may refer to [13].

perf.	**syntax**	**semantics**
inform	$\langle s, inform(r, \omega)\rangle$	FP : $B_s\omega \wedge \neg B_s Bif_r\ \omega \wedge \neg B_s Uif_r\ \omega$ RE: $B_r\omega$
confirm	$\langle s, confirm(r, \omega)\rangle$	FP : $B_s\omega \wedge B_s U_r\omega$ RE: $B_r\omega$
disconf	$\langle s, disconf(r, \omega)\rangle$	FP : $B_s\neg\omega \wedge (B_s B_r\omega \vee B_s U_r\omega)$ RE: $B_r\neg\omega$
request	$\langle s, request(r, a)\rangle$	FP : $(FP(a)[s\backslash r]) \wedge B_s Agent(r, a) \wedge \neg B_s I_r Done(a)$ RE : $Done(a)$

4 Semantics of ECCS

The semantics of ECCS is defined by Structural Operational Semantics (SOS) [14]. Many of the semantic rules are extensions of those for CCS [12]. We assume every agent of the system is honest and reliable. Let

$$Act_\tau = \{\tau\} \cup \{c(\alpha), \bar{c}(\alpha) \mid \alpha \ is\ a\ communicative\ act\} \cup \{p,\ \neg p \mid p \in AtomProp\}$$

In addition let $\Psi_i = (\Psi_{B_i},\ \Psi_{U_i},\ \Psi_{I_i})$ be the complete information store of i. The operational semantics of *update*, *query* and *observe* operators are as:

$$\frac{\Psi_i' = upd(\Psi_i, O_i\omega)}{\langle update(O_i\omega).P_i, \Psi_i\rangle \xrightarrow{\tau} \langle P_i, \Psi_i'\rangle} \qquad \frac{\Psi_i \models \phi}{\langle query(\phi).P_i, \Psi_i\rangle \xrightarrow{\tau} \langle P_i, \Psi_i\rangle}\ ,\ \phi \in \mathbf{For}$$

$$\frac{\Psi_i \not\models (B_ip \vee B_i\neg p),\ \Psi_i \models (I_iB_ip \vee I_iB_i\neg p),\ Env \models l}{\langle observe(p).P_i, \Psi_i\rangle \xrightarrow{l} \langle P_i, \Psi_i'\rangle}$$

where $l \in \{p, \neg p\}, p \in AtomProp$, and $\Psi_i' = upd(\Psi_i : B_il, \sim I_iB_il, \sim I_il, \sim I_iB_i\neg l, \sim I_i\neg l)$.

It is assumed $\omega \in \mathbf{ModFor}$ and $\Psi_i' = (\Psi_{B_i}',\ \Psi_{U_i}',\ \Psi_{I_i}')$ is the information store of agent i after performing an *update* or *observe*. Observe(p) is used for updating the information of an agent by observing some proposition from the environment. As *observe* is a communicative action (communication with environment), we define $FP(observe(p)) = \neg B_ip \wedge \neg B_i\neg p$ and $RE(observe(p)) = B_ip \vee B_i\neg p$. We assume *Env* (environment) contains the truth value of a set of related atomic propositions. After observe we will remove I_iB_il or I_il or $I_iB_i\neg l$ or $I_i\neg l$ as the intended proposition is believed. Query is used to ask ϕ from the information store and it is executed if ϕ is satisfied by information store of agent i.

4.1 Communication Operators

As mentioned before, we have two operators for communication of agents: $c(\alpha)$ and $\bar{c}(\alpha)$ for receiving and sending information respectively. We use four primitive communicative acts: $inform(\omega)$, $confirm(\omega)$, $disconfirm(\omega)$ and $request(a)$. We define the semantics of $inform$ and $request$ here. The semantics of $confirm$ and $disconfirm$ are similar to that of $inform$ (only FP and RE is different as shown in section 3.1). Semantics of sending messages is shown in table 2. We

Table 2. Semantics of sending a message

$\dfrac{\Psi_i \models B_i\omega \wedge \neg B_iBif_j\omega \wedge \neg B_iUif_j\omega,\ \ \Psi_i \models I_iB_j\omega \vee I_iDone(a)}{\langle \bar{c}(inform(\omega)).P_i, \Psi_i\rangle \xrightarrow{\bar{c}(inform(\omega))} \langle P_i, \Psi_i'\rangle}$ where $a \in \{\bar{c}(inform(\omega)),\ \bar{c}(inform\_if(\omega))\}$ and $\Psi_i' = upd(\Psi_i, B_iB_j\omega, \sim I_iB_j\omega, \sim I_iDone(a))$.
$\dfrac{\Psi_i \models FP(a)[i\backslash j] \wedge B_iAgent(j,a) \wedge \neg B_iI_jDone(a),\ \ \Psi_i \models I_iDone(a) \vee I_i\omega}{\langle \bar{c}(request(a)).P_i, \Psi_i\rangle \xrightarrow{\bar{c}(request(a))} \langle P_i, \Psi_i'\rangle}$ where $RE(a) = \omega$ and $\Psi_i' = upd(\Psi_i, B_iI_jDone(a),\ \sim I_iDone(a),\ \sim I_i\omega)$

suppose i is the sender and j is the recipient. Both the transitions have three parts. Firstly, FP of performative must be satisfied. This is known from section 3.1. Second is the intention which are the reason to send the message, and

the third is updating of IS to obtain Ψ_i'. Note that in FIPA, $inform\_if(\omega) \equiv inform(\omega) \mid inform(\neg\omega)$, where $\mid$ is the non-deterministic choice operator. In the semantics of inform we have deleted $I_iB_j\omega$ and $I_iDone(a)$, because the intended proposition is likely to be satisfied with the execution of the action. In the semantics of $request$, $Agent(j,a)$ is a function $Agent : Ag \times Act \rightarrow \{true, false\}$, which given an agent and an action, specifies whether the agent can do the action or not. $FP(a)[i\backslash j]$ is not defined clearly in the FIPA. It is defined in FIPA as the feasibility preconditions of the action a relating only to the sender i. We suppose a in the $\bar{c}(request(a))$, is either an internal action or one of $S = \{\bar{d}(inform(\omega)), observe(p), \bar{d}(confirm(\omega)), \bar{d}(disconfirm(\omega)), \bar{d}(inform\_if(\omega))\ \}$ where d is a channel name. Finally If c is a channel from i to j and d from j to i, Then $FP(a)[i\backslash j]$ for actions of S in order is: $\neg Bif_i\omega \wedge \neg Uif_i\omega$, $\{\}$, $U_i\omega$, $B_i\omega \vee U_i\omega$, and $\neg Bif_i\omega \wedge \neg Uif_i\omega$. If d is not a channel from j to i then $FP(a)[i\backslash j] = \{\}$.

Semantics of receiving a message is defined in table 3. Intuitively j after

Table 3. Semantics of receiving a message

$$\frac{\Psi_j' = upd(\Psi_j,\ B_j\omega,\ \sim B_jI_iDone(a),\ \sim I_jB_j\omega,\ \sim I_jDone(a))}{\langle c(inform(\omega)).P_j, \Psi_j\rangle \xrightarrow{c(inform(\omega))} \langle P_j, \Psi_j'\rangle}$$ where $a \in \{c(inform(\omega)),\ c(inform\_if(\omega))\}$.
$$\frac{\Psi_j' = upd(\Psi_j, I_jDone(a))}{\langle c(request(a)).P_j, \Psi_j\rangle \xrightarrow{c(request(a))} \langle P_j, \Psi_j'\rangle}$$

receiving the $inform(\omega)$ which is sent by i, would believe the contents of inform, because agents trust each other. The second argument removes $B_jI_iDone(a)$, as the action is beleived to be done by i. Actually, $B_jI_iDone(a)$ might be added after a request and we know it is satisfied if $c(inform(\omega))$ is done. $I_jB_j\omega$ and $I_jDone(a)$ will be removed because of the same reasons.

4.2 Summation and Parallel composition

Semantics of summation $P_1 + P_2$ is defined as usual. Let $a \in Act_\tau$ then:

$$\frac{\langle P_{i_1}, \Psi_i\rangle \xrightarrow{a} \langle P_{i_1}', \Psi_i'\rangle}{\langle P_{i_1} + P_{i_2}, \Psi_i\rangle \xrightarrow{a} \langle P_{i_1}', \Psi_i'\rangle} \qquad \frac{\langle P_{i_1}, \Psi_i\rangle \xrightarrow{a} \langle P_{i_1}', \Psi_i'\rangle}{\langle P_{i_2} + P_{i_1}, \Psi_i\rangle \xrightarrow{a} \langle P_{i_1}', \Psi_i'\rangle}$$

In the semantics of parallel composition there are two cases: First for any $l \in \{\tau\} \cup\ literals$ and the second for any communication γ,

$$\frac{\langle P_i, \Psi_i\rangle \xrightarrow{l} \langle P_i', \Psi_i'\rangle}{[...|\langle P_i, \Psi_i\rangle|...] \xrightarrow{l} [...|\langle P_i', \Psi_i'\rangle|...]} \qquad \frac{\langle P_i, \Psi_i\rangle \xrightarrow{\bar{\gamma}} \langle P_i', \Psi_i'\rangle\ ,\ \langle P_j, \Psi_j\rangle \xrightarrow{\gamma} \langle P_j', \Psi_j'\rangle}{[...|\langle P_i, \Psi_i\rangle|...|\langle P_j, \Psi_j\rangle|...] \xrightarrow{\tau[\gamma,\bar{\gamma}]} [...|\langle P_i', \Psi_i'\rangle|...|\langle P_j', \Psi_j'\rangle|...]}$$

In the second rule $\tau[\gamma, \bar{\gamma}]$ means the action is silent action but γ and $\bar{\gamma}$ have produced it. This will be useful in model checking when we use action expressions in the formulas.

5 ACTL Logic

In this section we define a logic which is like the branching time temporal logic CTL, but it has action operators for handling actions [6]. In addition the logic has operators for expressing mental properties like beliefs, uncertainties and intentions. The formal syntax of ACTL is defined as:

$$\mu ::= true \mid \phi \mid \neg\mu \mid \mu \wedge \mu \mid EX_x\mu \mid EG_x\mu \mid E\mu_x U_{x'}\mu' \\ \mid E\mu_x U\mu' \mid AX_x\mu \mid AG_x\mu \mid A\mu_x U_{x'}\mu' \mid A\mu_x U\mu'$$

Where $\phi \in$ **For** and $x, x' \subseteq Act_\tau$ are sets of actions. When x or x' include only one action, we omit the set brackets. The other operators can be defined using those of above. For $P \in \{E, A\}$, $PG\mu_x = \mu \wedge PG_x\mu$, $PF_x\mu = P\ true\ U_x\mu$. Many of the operators have their traditional meaning as in CTL. Semantics of ACTL is defined in terms of Labelled Transition Systems (LTS) with labelling in the transitions as well as states. Let **IS**= $\{\Psi \mid \Psi$ *is information store of multi-agent system*$\}$, and LTS structure M be a tuple of four elements as $< S, R, L, s_0 >$ where S is the set of states, $R \subseteq S \times Act_\tau \times S$ is the transition relation , $L : S \rightarrow$ **IS** is the state labelling function which assigns an information store Ψ for any state s, and $s_0 \in S$ is the initial state. We suppose $L(s_0) = \{\ \}$, or information store initially is empty. Formal semantics of ACTL is defined in table 4. Satisfaction of local formulas in each state ($L(S) \models \phi$) has been defined in section 2. Note that ACTL extends CTL and we can express CTL formulas by using Act_τ to express action expression x in any formula. For example $EX_{Act_\tau}\mu \in ACTL$ is like $EX\mu \in CTL$, because $a \in Act_\tau$ for any action a.

Table 4. satisfaction of ACTL formulas in labelled transition systems

$s \models true$	for any state s holds.
$s \models \phi$	*iff* $L(s) \models \phi$ *which means information store in state s must satisfy local formula* ϕ.
$s \models \neg\mu$	*iff* $s \not\models \mu$
$s \models \mu_1 \wedge \mu_2$	*iff* $s \models \mu_1$ *and* $s \models \mu_2$
$s_0 \models EX_x\mu$	*iff there is a path* $\pi = s_0 \stackrel{a_1}{\rightarrow} s_1 \stackrel{a_2}{\rightarrow} s_2 \rightarrow ...$ *such that* $s_1 \models \mu$ *and* $a_1 \in x$.
$s_0 \models E(\mu_x U_{x'}\mu')$	*iff* $s_0 \models \mu$ *and there is a path* $\pi = s_0 \stackrel{a_1}{\rightarrow} s_1 \stackrel{a_2}{\rightarrow} s_2 \rightarrow ...$ *and* $\exists j \geq 1$, $s_j \models \mu'$ *and* $a_j \in x'$ *and for all* $1 \leq i < j$, $s_i \models \mu$ *and* $a_i \in x$.
$s_0 \models E(\mu_x U\mu')$	*iff there is a path* $\pi = s_0 \stackrel{a_1}{\rightarrow} s_1 \stackrel{a_2}{\rightarrow} s_2 \rightarrow ...$ *and* $\exists j \geq 0$ *such that* $s_j \models \mu'$ *and for all* $0 \leq i < j$, $s_i \models \mu$ *and* $a_{i+1} \in x$.
$s_0 \models EG_x\mu$	*iff there is a path* $\pi = s_0 \stackrel{a_1}{\rightarrow} s_1 \stackrel{a_2}{\rightarrow} s_2 \rightarrow ...$ *such that* $s_{i+1} \models \mu$ *and* $a_i \in x$, *for all* $i \geq 0$.
$s_0 \models A...$	*iff for all paths ... (same as existential counterpart) ...*

6 Model Checking

Model checking of multi agent systems is a process of verifying correctness of agent systems. In these systems we not only deal with traditional concepts of concurrent systems like safety and liveness properties [19] but the mental properties of the system must be satisfied as well [22].

As we saw before, ECCS programs can be modeled as LTSs (as defined by the operational semantics). We may transform LTSs to Kripke models [11] and ACTL formulas to CTL formulas, then we may use existing model checkers. The other alternative is to define algorithms to check properties expressed in ACTL, directly on LTS. Algorithms for direct model checking of ACTL is omitted here because of space limit.

Our algorithm is similar to that of checking CTL formulas in the Kripke structures [3]. The algorithm contains various procedures to deal with different kinds of ACTL formulas. We assume some global data structures such as: M=(S, R, L, s_0) is the LTS to be checked, $labels = \{label(s_i) \mid label(s_i)$ is set of labels of s_i $\}$. We will store any subformula which is satisfied in a state s_i in the variable $label(s_i)$. In addition we assume $ListOfSub(\mu)$ is an ordered list of subformulas of a formula $\mu \in ACTL$ from shortest to longest length.

Our main procedure is Check-ACTL which, for any subformula $f \in ListOfSub(\mu)$, calls the appropriate model checking procedure based on the structure of the subformula. The list of subformulas are ordered from shortest to longest, and it guarantees whenever we check a subformula (like $f_1 \wedge f_2$), all of its subformulas (like f_1 and f_2) already have been checked. After checking all subformulas of the set, labels of any state shows the set of subformulas, satisfied by that state. If a state contains μ, it satisfies μ. If labels of initial state $label(s_0)$ contains μ then we say μ is satisfied in LTS structure M.

7 An example

We implement a very small example using ECCS and define its properties with the ACTL logic. In this example we assume there are two agents, one is *Provider* agent and the other is *Salesman* agent. Provider is responsible to provide an item and salesman is responsible for finding customer. Every time, if provider believes that there is no item, it will provide a new item from the environment and salesman can sell the item. We assume every time there is an item in the environment. When item is sold then provider will provide a new item. In the following example we have assumed *ps* is a channel from *Provider* to *Salesman* and *sp* from *Salesman* to *Provider*, $inform\_if(s, \phi)$ is feasible if either *inform(s,* ϕ*)* or *inform(s,* $\neg\phi$*)* is feasible, *c_r = customer is ready* and *i_r = item is ready.* Also we have assumed $\langle s, query\_if(r, \phi)\rangle \equiv \langle s, request(r, \langle r, inform\_if(s, \phi)\rangle)\rangle$, This operator allows s to query p from r.

Prvdr = update(I_pB_p i_r). Loop
Loop = {query(I_pB_p i_r). observe(i_r). update($I_pBif_pc\_r$). Loop}

$+$ {query($I_p Bif_p c\_r$). $\bar{ps}\langle$query_if(c_r)$\rangle$.
{$sp\langle$inform(c_r)$\rangle$. Loop } + {$sp\langle$ inform($\neg$ c_r)$\rangle$. update($I_p Bif_p c\_r$).
update($\neg B_p \neg$ c_r). Loop} }
$+$ {query(B_p c_r). update($\neg B_p$ i_r). update($\neg B_p$c_r). update($I_p B_p$i_r). Loop}

SIman = update($I_s(Bif_s c\_r)$). Loops
Loops = query($I_s Bif_s c\_r$). $ps\langle$query_if(c_r)$\rangle$.
{observe(c_r). $\bar{sp}\langle$inform(c_r)$\rangle$. update($\neg B_s$c_r).
update($\neg B_s B_p$c_r). update($I_s Bif_s c\_r$). Loops }
+{observe($\neg$c_r). $\bar{sp}\langle$inform($\neg$c_r)$\rangle$. update($\neg B_s \neg$c_r).
update($\neg B_s B_p \neg$c_r). update($I_s B_s c\_r$). Loops }

Here are some properties of the system, written in ACTL. For reasons of simplicity we don't write action expression indexes in formulas.

$AG[(B_p i\_r \wedge B_p c\_r) \Rightarrow AFI_p B_p i\_r]$. $\quad AG[B_s c\_r \Rightarrow AFI_s Bif_s c\_r]$.
$AG[B_s \neg c\_r \Rightarrow EFB_s c\_r]$. $\quad AG[I_p Bif_p c\_r \Rightarrow AFB_p c\_r]$.

8 Conclusion and Future Work

In this paper we have defined a language ECCS, with its syntax and semantics. We defined the structure of information store, which is a tree like structure with multiple views. A logic ACTL was defined for defining specifications of the system and a model checking algorithm was proposed for verification of agents.

Most of the work in the verification of multi-agent systems try to define a theory of rational agency [4, 16], but they suffer from lack of computationally grounded semantics [21]. There are some frameworks [23, 1] which attempt model checking on agent programs. One of the most relevant works to ours is Meyer's et.al. [20] work on defining a multi-agent framework using CSP [10] and CCP [17]. Authors have given a proof method for proving the properties of the system as well. In addition there are some other agent programming languages which are inspired by logic programming languages [5, 18, 15], and for few of them verification issues are specified.

Our work uses an abstract functional language CCS with a well defined operational semantics and we can simply check FIPA compliance properties in the semantics of the language (as we have done). Also, the model checking algorithm is easy and straightforward to implement in this framework.

The reason we have omitted disjunction in our information store is due to problems that crop up with disjunction. One of these problems is inconsistency checking, which is NP-complete.

One possible extension which we plan to try, is to add disjunction to our framework and use clausal resolution for inferring formulas from information store [7]. Also we plan to extend our framework to include some more performatives from FIPA-ACL. Finally we will use these ideas in an abstract rules based agent programming language.

References

1. R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In *AAMAS 2003*, pages 409–416, 2003.
2. E. M. Clark, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans Programming Lang Syst*, 8(2):244–263, 1986.
3. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
4. P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990.
5. M. Dastani, B. van Riemsdijk, F. Dignum, and J.-J. Ch. Meyer. A programming language for cognitive agents: Goal Directed 3APL. In M. Dastani and et.al., editors, *Proc. of the First Workshop on Programming Multi-agent Systems: Lang., Frameworks, Techs. and Tools (ProMAS03)*, pages 9–15, Melbourne, 2003.
6. R. De Nicola, A. Fantechi, S. Gnesi, and G. Ristori. An action-based framework for verifying logical and behavioral properties of concurrent systems. *Computer Networks and ISDN Systems*, 25(7):761–778, 1993.
7. C. Dixon, M. Fisher, and A. Bolotov. Clausal resolution in a logic of rational agency. *Artif. Intell.*, 139(1):47–89, 2002.
8. Foundation for Intelligent Physical Agents(FIPA). Fipa97 agent specification, http://www.fipa.org.
9. J. Y. Halpern. A logical approach to reasoning about uncertainty: a tutorial. In X. Arrazola, K. Korta, and F. J. Pelletier, editors, *Discourse, Interaction, and Communication*, pages 141–155. Kluwer, 1998.
10. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
11. S. A. Kripke. Semantical considerations on modal logic. In *A Colloquium on Modal and Many-Valued Logics*, Helsinki, 1962.
12. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
13. J. Pitt and A. Mamdani. Some remarks on the semantics of FIPA's agent communication language. *Aut. Agents and Multi-Agent Systems*, 4:333–356, 1999.
14. G. D. Plotkin. A structural approach to operational semantics. Technical report, DAIMI, FN 19, Department of Comp. Sci., University of Aarhus, Denmark, 1981.
15. A. S. Rao. AgentSpeak(L): BDI Agents speak out in a logical computable language. In W. Van de Velde and J. Perram, editors, *Proc. of MAAMAW'96*, number 1038 in LNAI, pages 42–55, The Netherlands, 1996. Springer-Verlag.
16. A. S. Rao and M. P. Georgeff. BDI-agents: From theory to practice. In *Proceedings of the First Intl. Conf. on Multiagent Systems(ICMAS-95)*, San Francisco, 1995.
17. Vijay A. Saraswat. *Concurrent Constraint Programming*. The MIT Press, 1993.
18. Y. Shoham. Agent-oriented programming. *Artif. Intell.*, 60(1):51–92, March 1993.
19. C. Stirling. *Modal and Temporal Properties of Processes*. Springer-Verlag, 2001.
20. R. M. Van Eijk, F. S. De Boer, W. Van Der Hoek, and J.-J. Ch. Meyer. A verification framework for agent communication. *Autonomous Agents and Multi-Agent Systems*, 6(2):185–219, 2003.
21. M. Wooldridge. Computationally grounded theories of agency. In E. Durfee, editor, *Proc. of the 4th Intrl. Conf. on Multi-Agent Systems (ICMAS 2000)*. IEEE Press.
22. M. Wooldridge. Verifiable semantics for agent communication languages. In Y. Demazeau, editor, *Proceedings of the 3rd International Conference on Multi Agent Systems*, pages 349–356. IEEE Computer Society, 1998.
23. M. Wooldridge, M. Fisher, M.-Ph. Huget, and S. Parsons. Model checking multi-agent systems with MABLE. In *Proc. of the first intrl. joint conf. on Autonomous agents and multiagent systems*, pages 952–959. ACM Press, 2002.