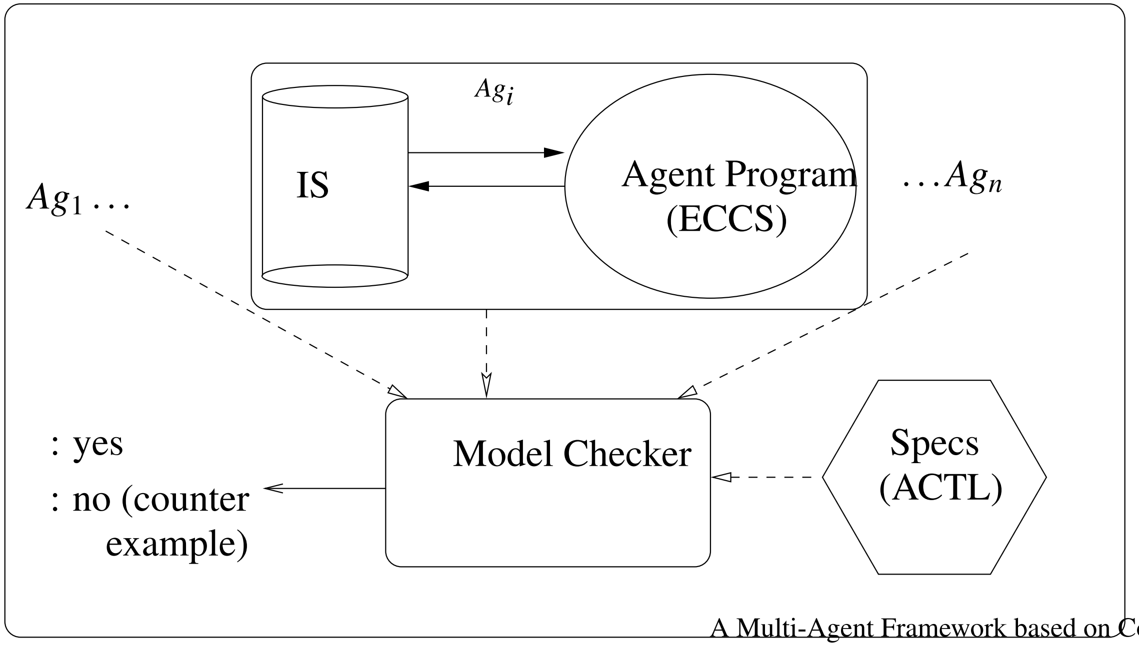
A Multi-Agent Framework based on Communication and Concurrency

Jamshid Bagherzadeh M. , S. Arun-Kumar

Indian Institute of Technology Delhi

Contents

- Mental state of the agents (information store)
- ECCS: Syntax and semantics of agent programming language
- ACTL: Logic for expressing properties
- Model Checking



Introduction

- In AI community: “Agent is an entity that functions continuously and autonomously in an environment, in which other processes take place and other agents exist”
- Other: An agent is an entity whose state is viewed as consisting of mental components such as, knowledge, beliefs, desires, capabilities, Intentions, plans...
- Systems whose structures are incompletely known (like robots, people,...)
- Modal logic is used for the formal representation of attitudes.
- Formal semantics is defined in terms of Kripke structures (possible worlds semantics)

Introduction: Multiagent systems

- Collection of agents which can cooperate and communicate with each other, through an agent communication language (ACL).
- ACL examples: KQML, FIPA-ACL
- Agents communicate via message passing in the form of Speech act theory (Performatives), to fulfill their individual goals (intentions).

In this paper:

- Agent programming and communication: We use CCS
- Extended with: Primitive Communicative Acts from FIPA-ACL [1]
- Attitudes Belief(B), Intention(I), Uncertainty(U)
- Information Store(IS) of each agent includes formulas involving B, I, U and other logical operators

Contents

- Mental state of the agents (information store)
- ECCS: Syntax and semantics of agent programming language
- ACTL: Logic for expressing properties
- Model Checking

Information Store: Logic for B, I, U

- Belief modality (B_i) : The logic $\mathbf{KD45}$,
- Intention relations (I_i): The logic $\mathbf{KD}$.
- Uncertainty (U_i): Only axiom $\mathbf{D}$.
- $\mathbf{K:} \vdash O_i(F \Rightarrow G) \Rightarrow (O_iF \Rightarrow O_iG)$

  $\mathbf{D:} \vdash O_iF \Rightarrow \neg O_i \neg F$

  $\mathbf{4:} \vdash O_iF \Rightarrow O_iO_iF$

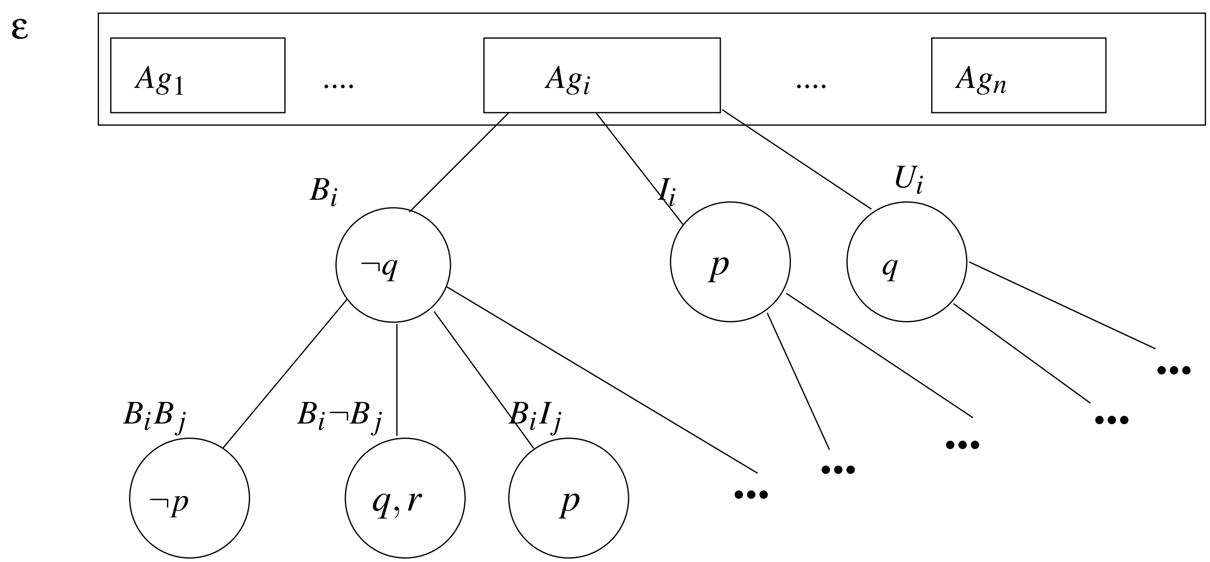
  $\mathbf{5:} \ \vdash \neg O_iF \Rightarrow O_i \neg O_iF$.

Information Store

- $Ag = \{1, \ldots, n\}$: Set of agents with typical elements like $i \; and \; j$

Information Store

- $Ag = \{1, \ldots, n\}$: Set of agents with typical elements like $i \; and \; j$



- For example ε, B_i, B_iB_j and $B_iB_jI_k$ are some views.

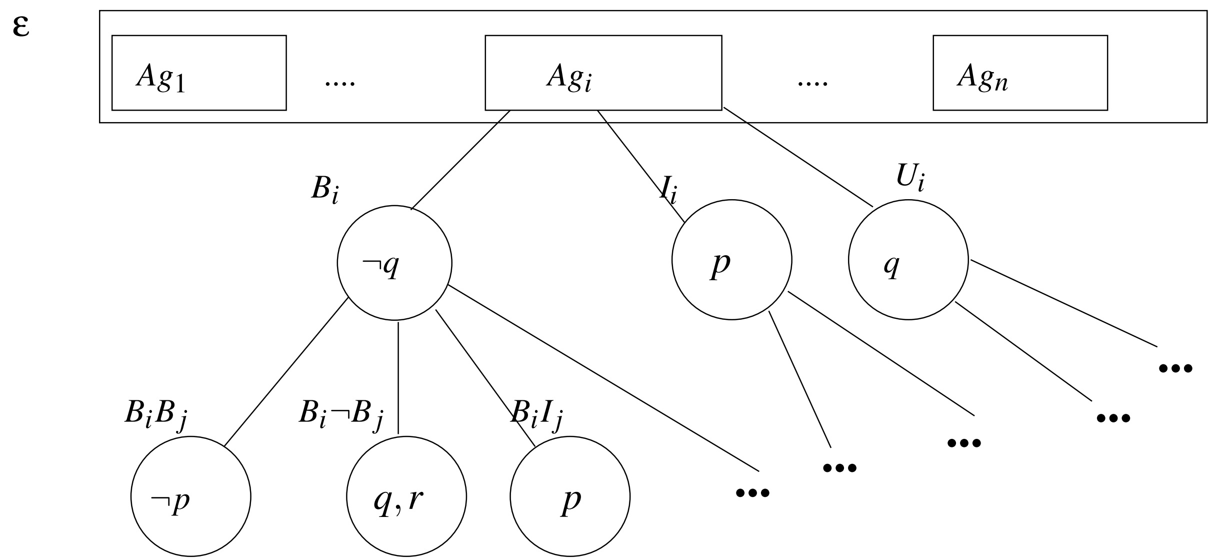
Information Store

- The set $\mathbf{ModFor}$ of *modal formulas* is defined as:
  $\omega ::= p \mid \neg p \mid O_i\omega \mid \neg O_i\omega$ where $O \in \{B, I, U\}$.

Information Store

- The set $\mathbf{ModFor}$ of *modal formulas* is defined as:
  $\omega ::= p \mid \neg p \mid O_i\omega \mid \neg O_i\omega$ where $O \in \{B, I, U\}$.
- Information store : A function Ψ, where

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



Properties of the modalities

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. $\neg B_i\omega \Leftrightarrow B_i\neg B_i\omega$

Satisfaction relation

- A formula ω is **accessible** from IS tree Ψ_i, $(\Psi_i \models_{acc} \omega)$

$$
\begin{array}{lll}
\Psi_i, v \models_{acc} l & \text{iff } \; l \in \Psi_i(v) & \text{where } l \text{ is a literal} \\
\Psi_i, v \models_{acc} O_i\omega & \text{iff } \; \Psi_i, vO_i \models_{acc} \omega & \\
 & \text{where } O \in \{B, I, U, \neg B, \neg I, \neg U\} & \\
\Psi_i \models_{acc} \omega & \text{iff } \; \Psi_i, \varepsilon \models_{acc} \omega &
\end{array}
$$

Satisfaction relation

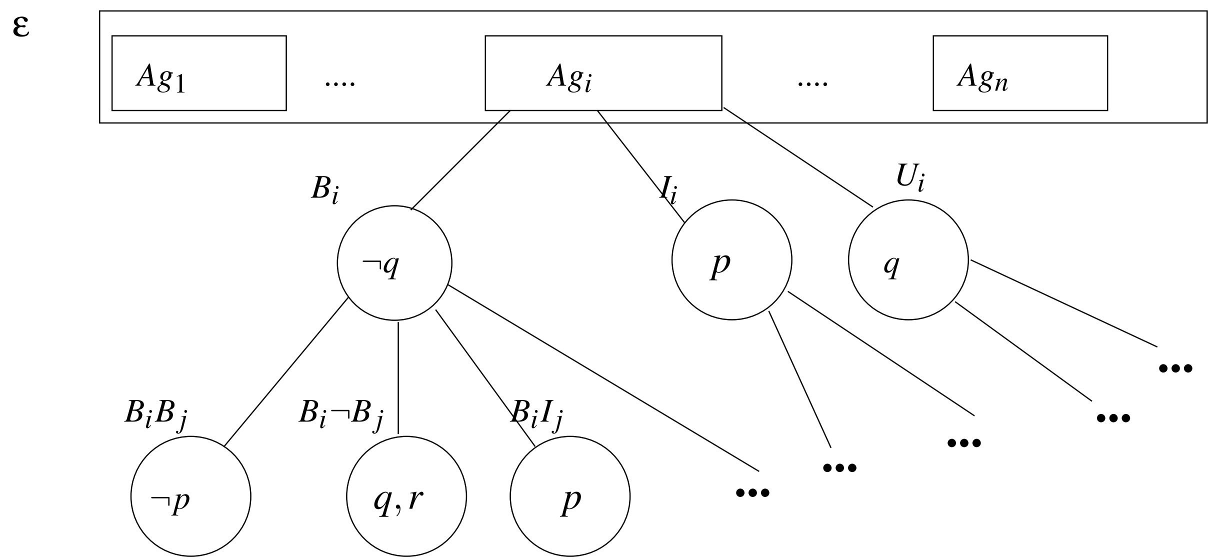
- A formula ω is **accessible** from IS tree Ψ_i, ($\Psi_i \models_{acc} \omega$)

$$
\begin{array}{lll}
\Psi_i, v \models_{acc} l & \text{iff } \; l \in \Psi_i(v) & \text{where } l \text{ is a literal} \\
\Psi_i, v \models_{acc} O_i\omega & \text{iff } \; \Psi_i, vO_i \models_{acc} \omega & \\
 & \text{where } O \in \{B, I, U, \neg B, \neg I, \neg U\} & \\
\Psi_i \models_{acc} \omega & \text{iff } \; \Psi_i, \varepsilon \models_{acc} \omega &
\end{array}
$$

- A formula ω is **derivable** from Ψ_i, ($\Psi_i \vdash \omega$), iff there is a formula ϕ s.t. $\Psi_i \models_{acc} \phi$, and $\phi \Rightarrow \omega$ using rules 1-11.

  Any accessible formula is also derivable.

Satisfaction relation



Example: $I_i p$, $B_iB_j\neg p$ are accessible

$\neg I_i \neg p$, $B_i \neg B_j p$ are derivable

Information Store: Satisfaction relation

Let $\mathbf{For}$ (*local formulas*) be the set of formulas ϕ defined as:

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

Information Store: Satisfaction relation

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

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

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* $\Psi_i \models \phi_2$
- $\Psi_i \models \neg\phi$ iff $\Psi_i \not\models \phi$

Information Store: Revision

- Information store of the multi-agent system is consistent if the information store of individual agents are consistent.
- If we apply the properties 1-11 when updating the IS consistency will hold.
- To update the information store, we define function upd:

$$upd(\Psi_i, v, \omega).$$

Information Store: Revision Function

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

$cond \equiv$ "*if v contains only B and I modalities*"

Information Store: Contraction Function

$upd(\Psi_i, v, \sim l) \quad = \Psi_i(v) - \{l\}$.	
$upd(\Psi_i, v, \sim M_j\omega) \quad = upd(\Psi_i, vM_j, \sim \omega)$;	if odd then [$upd(\Psi_i, v, \sim \neg M_j \neg\omega)$; $upd(\Psi_i, v, \sim \neg N_j\omega)$; $upd(\Psi_i, v, \sim \neg N_j \neg\omega)$]
$upd(\Psi_i, v, \sim I_j\omega) \quad = upd(\Psi_i, vI_j, \sim \omega)$;	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 [$upd(\Psi_i, v, \sim M_j \neg\omega)$; $upd(\Psi_i, v, \sim N_j\omega)$; $upd(\Psi_i, v, \sim N_j \neg\omega)$]
$upd(\Psi_i, v, \sim \neg I_j\omega) \quad = upd(\Psi_i, v\neg I_j, \sim \omega)$;	if $even$ then $upd(\Psi_i, v, \sim I_j \neg\omega)$.

even $\equiv$ *if there are an even number of* $\neg B$ *and* $\neg I$ *modalities in v.*

odd $\equiv$ *if there are an odd number of* $\neg B$ *and* $\neg I$ *modalities in v.*

Contents

- Mental state of the agents (information store)
- ECCS: Syntax and semantics of agent programming language
- ACTL: Logic for expressing properties
- Model Checking

Syntax of agent programming language

Formal syntax of ECCS:

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

P, P_1 and P_2 are ECCS agents
$A = P$ is a procees name.
$\alpha \ \in \{\ inform(\omega), request(a)\}$
ω: information to be updated
ϕ: information to be queried
p: a proposition

Syntax of agent programming language

Formal syntax of ECCS:

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

The multiagent system is defined as:

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

FIPA Performatives

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 : $B_r\omega$
request	$\langle s, request(r, a)\rangle$	FP : $(FP(a)[s\backslash r]) \quad \wedge \quad B_s Agent(r, a) \quad \wedge$ $\neg B_s I_r Done(a)$ RE : $Done(a)$

$Bif_r(\omega) = B_r\omega \vee B_r\neg\omega$, $Uif_r(\omega) = U_r\omega \vee U_r\neg\omega$.

Semantics: Operators

- The semantics of ECCS is defined by Structural Operational Semantics (SOS).

Semantics: Operators

- The semantics of ECCS is defined by Structural Operational Semantics (SOS).
- $Act_\tau = \{\tau\} \cup \{c(\alpha), \bar{c}(\alpha) \mid \alpha \; is\ a\ communicative\ act\} \cup \{p,\ \neg p \mid p \in AtomProp\}$

Semantics: Operators

- The semantics of ECCS is defined by Structural Operational Semantics (SOS).
- $Act_\tau = \{\tau\} \cup \{c(\alpha), \bar{c}(\alpha) \mid \alpha \; is \; a \; communicative \; act\} \cup \{p, \; \neg p \mid p \in AtomProp\}$
- UPDATE

$$\frac{\Psi_i' = upd(\Psi_i, O_i\omega)}{\langle update(O_i\omega).P_i, \Psi_i\rangle \xrightarrow{\tau} \langle P_i, \Psi_i'\rangle} \quad \omega \in \mathbf{ModFor}.$$

Semantics: Operators

- The semantics of ECCS is defined by Structural Operational Semantics (SOS).
- $Act_\tau = \{\tau\} \cup \{c(\alpha), \bar{c}(\alpha) \mid \alpha \text{ } \textit{is a communicative act}\} \cup \{p,\ \neg p \mid p \in AtomProp\}$
- UPDATE

$$\frac{\Psi_i' = upd(\Psi_i, O_i\omega)}{\langle update(O_i\omega).P_i, \Psi_i\rangle \xrightarrow{\tau} \langle P_i, \Psi_i'\rangle} \quad \omega \in \mathbf{ModFor}.$$

- QUERY

$$\frac{\Psi_i \models \phi}{\langle query(\phi).P_i, \Psi_i\rangle \xrightarrow{\tau} \langle P_i, \Psi_i\rangle}\ ,\ \phi \in \mathbf{For}$$

Semantics: Operators

OBSERVE

As *observe* is a communicative action (communication with environment), We define $FP(observe(p)) = \neg B_i p \wedge \neg B_i \neg p$ and

$$RE(observe(p)) = B_i p \vee B_i \neg p.$$

Observe(p) is used for updating the information of an agent by observing some proposition from the environment. After observe we will remove $I_i B_i l$ or $I_i l$ or $I_i B_i \neg l$ or $I_i \neg l$ as the intended proposition is believed.

$$\boxed{\begin{array}{c}\dfrac{\Psi_i \not\models (B_i p \vee B_i \neg p),\ \Psi_i \models (I_i B_i p \vee I_i B_i \neg p),\ Env \models l}{\langle observe(p).P_i, \Psi_i \rangle \xrightarrow{l} \langle P_i, \Psi_i' \rangle} \\ \Psi_i' = upd(\Psi_i : B_i l, \sim I_i B_i l, \sim I_i l, \sim I_i B_i \neg l, \sim I_i \neg l).\end{array}}$$

Semantics: Sending a message

- We use four primitive communicative acts: $inform(\omega)$, $confirm(\omega)$, $disconfirm(\omega)$ and $request(a)$.
- Semantics of sending $inform$
  Agent must be sincere. So it must satisfy sincerity conditions. In the semantics of inform we have deleted $I_i B_j \omega$ and $I_i Done(a)$, because the intended proposition is likely to be satisfied with the execution of the action.
- $$\boxed{\begin{array}{c} \dfrac{\Psi_i \models B_i\omega \wedge \neg B_i Bif_j \omega \wedge \neg B_i Uif_j \omega, \quad \Psi_i \models I_i B_j \omega \vee I_i Done(a)}{\langle \bar{c}(inform(\omega)).P_i, \Psi_i\rangle \xrightarrow{\bar{c}(inform(\omega))} \langle P_i, \Psi'_i\rangle} \\ \text{where } a \in \{\bar{c}(inform(\omega)),\ \bar{c}(inform\_if(\omega))\} \text{ and} \\ \Psi'_i = upd(\Psi_i, B_i B_j \omega, \sim I_i B_j \omega, \sim I_i Done(a)). \end{array}}$$

Semantics: Sending a message

- Semantics of sending *request*.

$$\frac{\Psi_i \models FP(a)[i\backslash j] \wedge B_i Agent(j,a) \wedge \neg B_i I_j Done(a), \quad \Psi_i \models I_i Done(a) \vee I_i \omega}{\langle \bar{c}(request(a)).P_i, \Psi_i \rangle \xrightarrow{\bar{c}(request(a))} \langle P_i, \Psi'_i \rangle}$$

where $RE(a) = \omega$ and
$\Psi'_i = upd(\Psi_i, B_i I_j Done(a), \sim I_i Done(a), \sim I_i \omega)$

Semantics: Receiving a message

Agents trust each other.

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

where $a \in \{c(inform(\omega)),\ c(inform\_if(\omega))\}$.

Semantics: Receiving a message

Agents trust each other.

$$\frac{\Psi'_j = upd(\Psi_j,\ B_j\omega,\ \sim B_j I_i Done(a),\ \sim I_j B_j \omega,\ \sim I_j Done(a))}{\langle c(inform(\omega)).P_j, \Psi_j\rangle \xrightarrow{c(inform(\omega))} \langle P_j, \Psi'_j\rangle}$$

where $a \in \{c(inform(\omega)),\ c(inform\_if(\omega))\}$.

$$\frac{\Psi'_j = upd(\Psi_j, I_j Done(a))}{\langle c(request(a)).P_j, \Psi_j\rangle \xrightarrow{c(request(a))} \langle P_j, \Psi'_j\rangle}$$

Semantics of Choice

Nondeterministic choice $\mathbf{P_1 + P_2}$:

$$\frac{\langle P_{i_1}, \Psi_i \rangle \xrightarrow{a} \langle P'_{i_1}, \Psi'_i \rangle}{\langle P_{i_1} + P_{i_2}, \Psi_i \rangle \xrightarrow{a} \langle P'_{i_1}, \Psi'_i \rangle}$$

$$\frac{\langle P_{i_1}, \Psi_i \rangle \xrightarrow{a} \langle P'_{i_1}, \Psi'_i \rangle}{\langle P_{i_2} + P_{i_1}, \Psi_i \rangle \xrightarrow{a} \langle P'_{i_1}, \Psi'_i \rangle}$$

Semantics: *Parallel composition* $\mathbf{P_1} \mid \mathbf{P_2} | \ldots | \mathbf{P_n}$

- 1: First for any $l \in \{\tau\} \cup \ literals$

$$\frac{\langle P_i, \Psi_i \rangle \xrightarrow{l} \langle P_i', \Psi_i' \rangle}{[\ldots|\langle P_i, \Psi_i \rangle|\ldots] \xrightarrow{l} [\ldots|\langle P_i', \Psi_i' \rangle|\ldots]}$$

Semantics: *Parallel composition* $\mathbf{P_1} \mid \mathbf{P_2} | \ldots |\mathbf{P_n}$

- 1: First for any $l \in \{\tau\} \cup \ literals$

$$\frac{\langle P_i, \Psi_i \rangle \xrightarrow{l} \langle P_i', \Psi_i' \rangle}{[\ldots|\langle P_i, \Psi_i \rangle|\ldots] \xrightarrow{l} [\ldots|\langle P_i', \Psi_i' \rangle|\ldots]}$$

- 2: Second for any communication γ,

$$\frac{\langle P_i, \Psi_i \rangle \xrightarrow{\bar{\gamma}} \langle P_i', \Psi_i' \rangle \ , \ \langle P_j, \Psi_j \rangle \xrightarrow{\gamma} \langle P_j', \Psi_j' \rangle}{\begin{array}{r}[\ldots|\langle P_i, \Psi_i \rangle|\ldots|\langle P_j, \Psi_j \rangle|\ldots] \xrightarrow{\tau[\gamma,\bar{\gamma}]} \\ [\ldots|\langle P_i', \Psi_i' \rangle|\ldots|\langle P_j', \Psi_j' \rangle|\ldots]\end{array}}$$

– In the second rule $\tau[\gamma,\bar{\gamma}]$ means the action is silent action but γ and $\bar{\gamma}$ have produced it.

Contents

- Mental state of the agents (information store)
- ECCS: Syntax and semantics of agent programming language
- ACTL: Logic for expressing properties
- Model Checking

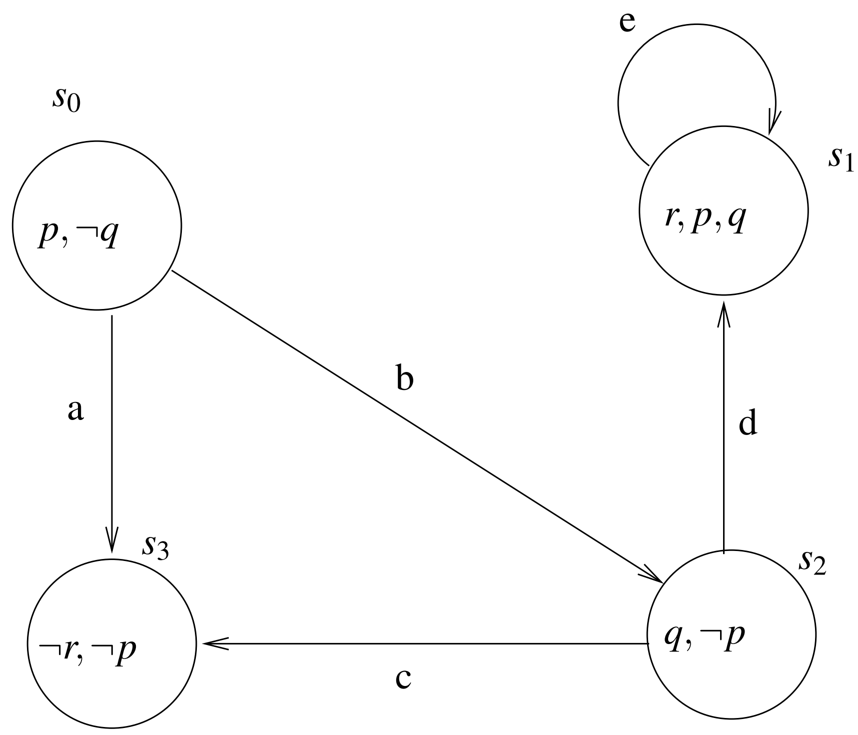
ACTL logic: Syntax

- $\mu ::= true \mid \phi \mid \neg\mu \mid \mu \wedge \mu \mid EX_x\mu \mid EG_x\mu \mid E\mu_x U_{x'}\mu'$
  $\mid E\mu_x U\mu' \mid AX_x\mu \mid AG_x\mu \mid A\mu_x U_{x'}\mu' \mid A\mu_x U\mu'$
- $\phi ::= O_i\omega \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \neg\phi$
- $\omega ::= p \mid O_i\omega \mid \neg p \mid \neg O_i\omega$
- $x ::= a \mid \neg x \mid x \vee x' \mid true$ where $a \in Act_\tau$

Semantics of ACTL

Semantics of ACTL is defined in terms of Labelled Transition Systems(LTS).

LTS structure M is a tuple of four elements as $< S, R, L, s_0 >$ where S is the set of states,



ACTL logic: Formal semantics

$s \models true$	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* $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \rightarrow \ldots$ *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* $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \rightarrow \ldots$ *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* $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \rightarrow \ldots$ *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* $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \rightarrow \ldots$ *such that* $s_{i+1} \models \mu$ *and* $a_i \in x$, *for all* $i \geq 0$.
$s_0 \models A\ldots$	*iff for all paths* $\ldots$ *(same as existential counterpart)* $\ldots$

ACTL logic: action expression satisfaction

Satisfaction of action expressions by actions is defined as:

- $a \models true$ always
- $a \models b$ iff $a = b$
- $a \models \neg x$ iff $a \not\models x$
- $a \models x \vee x'$ iff $a \models x$ or $a \models x'$

Contents

- Mental state of the agents (information store)
- ECCS: Syntax and semantics of agent programming language
- ACTL: Logic for expressing properties
- Model Checking

Model checking

- Procedure Check-simple takes an information store and a particular view of that, and checks a local formula ϕ in that view.
- Check-sub-prop takes a local formula and checks all the states of LTS to see whether the formula is satisfied in each state or not.
- Check-EGx(f, x) for checking $EG\ f_x$,
- Check-ExUx'(f, x, x', g) for E($f_x U_{x'} g$),
- Check-ExU(f, x, g) for E($f_x U g$),
- Check-EXx(f) for $EX_x f$ are defined here and procedures
- Check-disjunct($f_1,\ f_2$) for $f_1 \vee f_2$,
- Check-conjunct(f, g) for $f \wedge g$, and
- Check-neg(f) for $\neg f$.

Model checking

The main procedure to check formula f in LTS structure M is defined below:

Procedure Check-ACTL(ListOfSub , M)

Begin

for each subformula f_i in ListOfSub do

if f_i is of form {p, $\phi_1 \vee_o \phi_2$, $\phi_1 \wedge_o \phi_2$, $\neg_o \phi$, $O_i \phi$} then

Call Check-sub-prop(f_i);

if $f_i = f_1 \vee f_2$ then

Call Check-disjunct($f_1,\ f_2$);

if $f_i = f_1 \wedge f_2$ then

Call Check-conjunct($f_1,\ f_2$);

if $f_i = \neg f_j$ then

Call Check-neg(f_j);

Model checking

The main procedure to check formula f in LTS structure M is defined below:

if $f_i = EGg_x$ then

Call Check-EGx($g,\ x$);

if $f_i = E(f_x U_{x'} g)$ then

Call Check-ExUx'($f,\ x,\ x',\ g$);

if $f_i = E(f_x U g)$ then

Call Check-ExU($f,\ x,\ g$);

if $f_i = EX_x f_j$ then

Call Check-EXx($x,\ f_j$);

end for;

if $label(s_0)$ contains f then "f is satisfied"

else "f is not satisfied";

end;

end Check-ACTL.

Example: ***Provider and Salesman***

– Assume there are two agents: $Provider$, $Salesman$.

– Provider provides items, Salesman finds customer.

– If there is no item, Provider will provide a new item from environment and salesman can sell it.

– When item is sold then provider will provide a new item.

– $inform\_if(s, \phi)$ is feasible if either $inform(s, \phi)$ or $inform(s, \neg\phi)$ is feasible.

Example: Provider Agent

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($I_pBif_pc\_r$).
$\bar{p}\bar{s}$⟨query_if(c_r)⟩.
{ sp⟨inform(c_r)⟩. Loop } +
{ sp⟨ inform($\neg$ c_r)⟩.
update($I_pBif_pc\_r$).
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_pB_pi_r). Loop }

Example: Salesman Program

Slman = 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).

$s\bar{p}\langle$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).

$s\bar{p}\langle$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 }

Example: Some system properties in ACTL

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

Conclusion and future work

Summary:

- The language ECCS (Syntax and Semantics), Information store, The logic ACTL (syntax and Semantics), A Model checking algorithm.
- Motivation: Using of abstract functional language CCS, Suitable for Concurrent Processes, with well defined semantics.

Conclusion and future work

Summary:

- Literature: Frameworks by Wooldridge , Bordini, Fisher which attempt model checking on languages like Mable and Agentspeak.
  Method: Translation of the programs to another language which is input language of a model checker: like Promela in SPIN model checker.
- Add disjunction to IS, Define an abstract rules based language based on CCS, Efficient model checking algorithms, Considering more performatives from FIPA-ACL and protocols of FIPA.

End

Thank You

References

[1] Foundation for Intelligent Physical Agents(FIPA). Fipa97 agent specification.