On Ability to Autonomously Execute Agent Programs with Sensing

Sebastian Sardiña
Dept. of Computer Science
University of Toronto
ssardina@cs.toronto.edu

Giuseppe De Giacomo
Dip. Informatica e Sistemistica
Univer. di Roma “La Sapienza”
degiacomo@dis.uniroma1.it

Yves Lespérance
Dept. of Computer Science
York University
lesperan@cs.yorku.ca

Hector J. Levesque
Dept. of Computer Science
University of Toronto
hector@cs.toronto.edu

http://www.cs.toronto.edu/~cogrobo
Motivation

Agent programming languages should support planning/deliberation under *incomplete information* with *sensing actions*.

Need a spec. for this.

Most existing formal models of deliberation are epistemic, while agent programming languages have transition system semantics. Hard to relate.

Here, develop new non-epistemic formal model of deliberation.

Set within situation calculus and IndiGolog.

But important lessons for agent programming languages in general.
Situation Calculus

- A dialect of the predicate calculus with sorts for actions and situations

- **Actions:** open, look, dial(14), go(Airport), check_departures, ...

- **Situations:**
  - $S_0$ the initial situation
  - $do(a, s)$ where $a$ is an action term and $s$ is a situation

- **Fluents:** predicates/functions whose value changes from situation to situation:

  $$Locked(S_0) \land \neg Locked(do(dial(2), S_0))$$

- Use a distinguished predicate $Poss(a, s)$ to assert that it is possible to execute action $a$ in situation $s$
High-Level Deterministic Programs

\( a, \delta_1; \delta_2, \)
\textbf{if} \( \phi \) \textbf{then} \( \delta_1 \) \textbf{else} \( \delta_2 \) \textbf{endIf},
\textbf{while} \( \phi \) \textbf{do} \( \delta \) \textbf{endWhile},

primitive action
sequence
conditional
while loop
High-Level Deterministic Programs

\( a, \delta_1; \delta_2, \)
\( \text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2 \text{ endIf}, \)
\( \text{while } \phi \text{ do } \delta \text{ endwhile}, \)

\[ \text{GetOnFlight}_{\text{Detailed}} \overset{\text{def}}{=} \]

\[ \text{go(Airport)}; \]
\[ \text{check departures}; \quad \% \text{sensing action} \]
\[ \text{if Parked(Flight123, GateA) then} \]
\[ \quad \text{go(GateA); board plane(Flight123)} \]
\[ \text{else} \]
\[ \quad \text{go(GateB); board plane(Flight123)} \]
\[ \text{endIf} \]

Without sensing, goal cannot be achieved!
High-Level Programs and Histories

A **history** \( \sigma = (a_1, \mu_1), (a_2, \mu_2), \ldots, (a_n, \mu_n) \) describes a run with actions and their sensing results; e.g.:

\[
(go(Airport), 1) \cdot (check\_departures, 0) \cdot (go(GateB), 1).
\]

A **configuration** is a pair \( (\delta, \sigma) \) with a program \( \delta \) and a history \( \sigma \) specifying the actions performed so far and the sensing results obtained.
A **history** $\sigma = (a_1, \mu_1), (a_2, \mu_2), \ldots, (a_n, \mu_n)$ describes a run with actions and their sensing results; e.g.:

\[
(go(Airport), 1) \cdot (check\_departures, 0) \cdot (go(GateB), 1).
\]

A **configuration** is a pair $(\delta, \sigma)$ with a program $\delta$ and a history $\sigma$ specifying the actions performed so far and the sensing results obtained.

$Sensed[\sigma]$ stands for formula specifying the *sensing results* of history $\sigma$, e.g.:

\[
SF(go(Airport), S_0) \land \\
\neg SF(check\_departures, do(go(Airport), S_0)) \land \\
SF(go(GateB), do(check\_departures, do(go(Airport), S_0)))).
\]

$end[\sigma]$ stands for the situation term of history, $\sigma$, e.g.:

\[
end[\epsilon] = S_0 \text{ and } end[(go(Airport), 1)] = do(go(Airport), S_0)
\]
Two relations in the object language:

\(\text{Trans}(\delta, s, \delta', s')\): means that it can make transition \((\delta, s) \rightarrow (\delta', s')\) by executing a single primitive action or test.

\(\text{Final}(\delta, s)\): means that computation can be terminated in \((\delta, s)\)
**IndiGoLog** *Semantics for Online Executions*

Two relations in the object language:

*Trans*(δ, s, δ', s')*: means that it can make transition (δ, s) → (δ', s') by executing a single primitive action or test.

*Final*(δ, s): means that computation can be terminated in (δ, s)

Semantics based on two notions (relative to an underlying theory of action *D*):

- Configuration (δ, σ) *may evolve* to (δ', σ') *w.r.t. a model* *M*.

- A configuration (δ, σ) is *final*, i.e. may legally terminate.
**IndiGolog** Semantics for Online Executions

Two relations in the object language:

\[ \text{Trans}(\delta, s, \delta', s') : \text{means that it can make transition } (\delta, s) \rightarrow (\delta', s') \text{ by executing a single primitive action or test.} \]

\[ \text{Final}(\delta, s) : \text{means that computation can be terminated in } (\delta, s) \]

Semantics based on two notions (relative to an underlying theory of action \( \mathcal{D} \)):

- Configuration \((\delta, \sigma)\) may evolve to \((\delta', \sigma')\) \textit{w.r.t. a model } \( \mathcal{M} \).

- A configuration \((\delta, \sigma)\) is final, i.e. may legally terminate.

The theory \( \mathcal{D} \), augmented with the sensing results in \( \sigma \), must entail that the transition is possible.

Model \( \mathcal{M} \) above represents a possible environment (generate sensing results.)
Configuration \((\delta, \sigma)\) *may evolve* to \((\delta', \sigma')\) *w.r.t. a model* \(M\) (relative to an underlying theory of action \(\mathcal{D}\)) iff

(i) \(M\) is a model of \(\mathcal{D} \cup \{Sensed[\sigma]\}\), and (ii)

\[
\mathcal{D} \cup \mathcal{T} \cup \{Sensed[\sigma]\} \models \text{Trans}(\delta, s, \delta', s'),
\]

and (iii)

\[
\sigma' = \begin{cases} 
\sigma \cdot (a, 1) & \text{if } s' = \text{do}(a, s) \text{ and } M \models SF(a, s), \\
\sigma \cdot (a, 0) & \text{if } s' = \text{do}(a, s) \text{ and } M \not\models SF(a, s), \\
\sigma & \text{if } s' = s,
\end{cases}
\]

where \(s = \text{end}[\sigma]\) and \(s' = \text{end}[\sigma']\).

Configuration \((\delta, \sigma)\) is *final* when: \(\mathcal{D} \cup \mathcal{T} \cup \{Sensed[\sigma]\} \models \text{Final}(\delta, \text{end}[\sigma])\)
Deliberation: An EC-based Account

The idea:

An agent can/knows how to execute program $\delta$ in history $\sigma$ iff he can always choose a next action/transition and eventually reach a terminating configuration no matter what sensing results are obtained.

Agent must know that the chosen action/transition is allowed by the program and is executable.

In what follows, we will define $K_{EC}(\delta, \sigma)$ using both entailment and consistency.
We define $K_{How_{EC}}(\delta, \sigma)$ to be the smallest relation $\mathcal{R}(\delta, \sigma)$ such that:
We define $KHow_{EC}(\delta, \sigma)$ to be the smallest relation $\mathcal{R}(\delta, \sigma)$ such that:

(E1) if $(\delta, \sigma)$ is final, then $\mathcal{R}(\delta, \sigma)$;
We define $KHow_{EC}(\delta, \sigma)$ to be the smallest relation $\mathcal{R}(\delta, \sigma)$ such that:

(E1) if $(\delta, \sigma)$ is final, then $\mathcal{R}(\delta, \sigma)$;

(E2) if $(\delta, \sigma)$ may evolve to configurations $(\delta', \sigma \cdot (a, \mu_i))$ w.r.t. some models $M_i$ with $i = 1..k$ for some $k \geq 1$, and $\mathcal{R}(\delta', \sigma \cdot (a, \mu_i))$ holds for all $i = 1..k$, then $\mathcal{R}(\delta, \sigma)$.

Obs.: always consider the underlying theory plus the observations in history $\sigma$. 
E.g. Tree Chopping Problem

Agent wants to cut down a tree.

Possible actions are:

- **chop**, take one chop at the tree, and
- **look**, which tells the agent whether the tree has fallen (i.e., senses fluent $Down$).

It is known that tree will eventually fall, but number of *chops* needed is not bounded.
E.g. Tree Chopping Problem

Agent wants to cut down a tree.

Possible actions are:

- *chop*, take one chop at the tree, and
- *look*, which tells the agent whether the tree has fallen (i.e., senses fluent *Down*).

It is known that tree will eventually fall, but number of *chops* needed is not bounded.

Intuitively, the following program $\delta_{tc}$:

```
while $\neg Down$ do chop; look endWhile
```

solves the problem. But . . .
Formalizing the Tree Chopping E.g.: $D_{tc}$

- $D_{ss}$ contains the following successor state axiom:

\[ \text{RemainingChops}(\text{do}(a, s)) = n \equiv \]
\[ a = \text{chop} \land \text{RemainingChops}(s) = n + 1 \lor \]
\[ a \neq \text{chop} \land \text{RemainingChops}(s) = n. \]

- $D_{ap}$ contains the following 2 precondition axioms:

\[ \text{Poss}(\text{chop}, s) \equiv (\text{RemainingChops}(s) > 0), \quad \text{Poss}(\text{look}, s) \equiv \text{True}. \]

- $D_{sf}$ contains the following sensing axiom:

\[ \text{SF}(\text{look}, s) \equiv (\text{RemainingChops}(s) = 0). \]

- $D_{S_0} = \{\text{RemainingChops}(S_0) > 0\}$ (initial situation).

\[ \text{Down}(s) \stackrel{\text{def}}{=} (\text{RemainingChops}(s) = 0). \]
Tree Chopping E.g. (cont.)

Intuitively, the following program $\delta_{tc}$:

\[
\textbf{while } \neg\text{Down } \textbf{do chop; look endWhile}
\]

solves the problem. But . . .
Tree Chopping E.g. (cont.)

Intuitively, the following program $\delta_{tc}$:

\[
\text{while } \neg \text{Down do chop; look endWhile}
\]

solves the problem. But . . .

**Theorem 1.** For all $k \in \mathbb{N}$, $K\text{How}_{EC}(\delta_{tc}, [(\text{chop}, 1) \cdot (\text{look}, 0)]^k)$ does not hold. In particular, when $k = 0$, $K\text{How}_{EC}(\delta_{tc}, \epsilon)$ does not hold.

In fact, . . .
Intuitively, the following program $\delta_{tc}$:

\begin{verbatim}
while $\neg$Down do chop; look endWhile
\end{verbatim}

solves the problem. But . . .

**Theorem 1.** For all $k \in \mathbb{N}$, $KHow_{EC}(\delta_{tc}, [(chop, 1) \cdot (look, 0)]^k)$ does not hold. In particular, when $k = 0$, $KHow_{EC}(\delta_{tc}, \epsilon)$ does not hold.

In fact, ...

**Theorem 2.** Whenever $KHow_{EC}(\delta, \sigma)$ holds, there is simple kind of conditional plan, a TREE program, that can be followed to execute $\delta$ in $\sigma$.

which means that ...
Intuitively, the following program $\delta_{tc}$:

$$\text{while } \neg \text{Down do } \text{chop}; \text{look } \text{endWhile}$$

solves the problem. But . . .

**Theorem 1.** For all $k \in \mathbb{N}$, $K\text{How}_{EC}(\delta_{tc}, [(\text{chop}, 1) \cdot (\text{look}, 0)]^k)$ does not hold. In particular, when $k = 0$, $K\text{How}_{EC}(\delta_{tc}, \epsilon)$ does not hold.

In fact, ...

**Theorem 2.** Whenever $K\text{How}_{EC}(\delta, \sigma)$ holds, there is simple kind of conditional plan, a TREE program, that can be followed to execute $\delta$ in $\sigma$.

which means that ...

**Corollary 1.** $K\text{How}_{EC}$ is only correct when problem has a bounded solution
Why does $K\text{How}_{EC}$ Fail?

$\delta_{tc} \overset{\text{def}}{=} \text{while } \neg \text{Down do chop; look endWhile}$
Why does $KHow_{EC}$ Fail?

\[
\begin{align*}
\delta_{tc} & \\
\downarrow & \text{chop} \\
\text{look}; \delta_{tc} &
\end{align*}
\]
Why does $KHow_{EC}$ Fail?

\[ \delta_{tc} \]
\[ \downarrow \text{chop} \]
\[ \text{look;} \delta_{tc} \rightarrow \delta_{tc} \]
\[ \downarrow \text{look}, 0 \quad \text{look}, 1 \]
Why does $K\text{How}_{EC}$ Fail?

$\delta_{tc}$

$\downarrow$ chop

$\text{look;} \delta_{tc}$ $\rightarrow$ $\delta_{tc}$ $\checkmark$

$\downarrow$ look, 0 look, 1

$\delta_{tc}$

$\downarrow$ chop
Why does $K\text{How}_{EC}$ Fail?

\[ \delta_{tc} \]
\[ \downarrow \text{chop} \]
\[ \text{look;} \delta_{tc} \quad \rightarrow \quad \delta_{tc} \checkmark \]
\[ \downarrow \text{look}, 0 \quad \text{look}, 1 \]

\[ \delta_{tc} \]
\[ \downarrow \text{chop} \]
\[ \text{look;} \delta_{tc} \quad \rightarrow \quad \delta_{tc} \checkmark \]
\[ \downarrow \text{look}, 0 \quad \text{look}, 1 \]

\[ \ldots \]
Why does $KHow_{EC}$ Fail?

If $KHow_{EC} (\delta_{tc}, \epsilon)$, then for all $j \in \mathbb{N}$:

$KHow_{EC} (\delta_{tc}, ((\text{chop}, 1) \cdot (\text{look}, 0))^j)$ and $KHow_{EC} (\text{look}; \delta_{tc}, ((\text{chop}, 1) \cdot (\text{look}, 0))^j \cdot (\text{chop}, 1)).$

$KHow_{EC}$ is “taking into account” the execution where three never comes down!

Every finite prefix of the execution is consistent with $D_{tc}$.

But, the set of all of them together is not!
Deliberation: ET-based Account

To solve the problem, we consider each environment/model individually.

We define $K\text{HowIn}M(\delta, \sigma, M)$ to be the smallest relation $\mathcal{R}(\delta, \sigma)$ such that:
Deliberation: ET-based Account

To solve the problem, we consider each environment/model individually.

We define $KHowInM(\delta, \sigma, M)$ to be the smallest relation $\mathcal{R}(\delta, \sigma)$ such that:

(T1) if $(\delta, \sigma)$ is final, then $\mathcal{R}(\delta, \sigma)$;
Deliberation: ET-based Account

To solve the problem, we consider each environment/model individually.

We define $K_{HowInM}(\delta, \sigma, M)$ to be the smallest relation $\mathcal{R}(\delta, \sigma)$ such that:

1. **(T1)** if $(\delta, \sigma)$ is final, then $\mathcal{R}(\delta, \sigma)$;

2. **(T2)** if $(\delta, \sigma)$ may evolve to $(\delta', \sigma \cdot (a, \mu))$ w.r.t. $M$ and $\mathcal{R}(\delta', \sigma \cdot (a, \mu))$, then $\mathcal{R}(\delta, \sigma)$.

*Uses truth in a model!*
Deliberation: ET-based Account (cont.)

$KHow_{ET}(\delta, \sigma)$: iff it knows how to execute it in every model of the history (i.e., in every possible environment).

Formally:

$$KHow_{ET}(\delta, \sigma)$$

iff

for every model $M$ such that $M \models \mathcal{D} \cup \{Sensed[\sigma]\}$,

$KHowInM(\delta, \sigma, M)$ holds.
Deliberation: ET-based Account (cont.)

\( KHow_{ET}(\delta, \sigma) \): iff it knows how to execute it in every model of the history (i.e., in every possible environment).

Formally:

\[
KHow_{ET}(\delta, \sigma) \iff \\
\text{for every model } M \text{ such that } M \models D \cup \{Sensed[\sigma]\}, \\
KHowInM(\delta, \sigma, M) \text{ holds.}
\]

\( KHow_{ET} \) handles the tree chopping example: \( KHow_{ET}(\delta_{tc}, \epsilon) \) holds!
Generalizing to Non-deterministic Programs

Two possible approaches:

- Angelic: choices are under the control of the agent (deliberation);
- Demoniac: choices are **not** under the control of the agent (execution).

\[
K_{How}^{Ang}(\delta, \sigma) \quad \text{iff} \quad \text{there is a deterministic program } \delta^d \text{ such that } K_{How}^{ET}(\delta^d, \sigma) \text{ holds, and} \]

\[
\mathcal{D} \cup \mathcal{T} \cup \{\text{Sensed}[\sigma]\} \models \exists s. Do(\delta^d, \text{end}[\sigma], s) \land Do(\delta, \text{end}[\sigma], s).\]


Further Issues

- $KHow_X$ can be used to define agent ability $\text{Can}_X(\phi, \sigma)$.

- Angelic version for nondeterministic programs can be used to define a meta-theoretic account of deliberation (i.e., search construct in INDIGOLOG).

- Relation $KHow_{ET}$ corresponds exactly to effective controllers and robot programs as described in [Lin and Levesque 1998]. Hence, it is both sound and complete w.r.t. robot achievability.
Lessons for Agent Programming Languages

Most agent programming languages have transition semantics and use *entailment* to evaluate tests and action preconditions (e.g., ConGolog, 3APL, FLUX, etc.).

Most agent programming languages only do on-line reactive execution; no deliberation/lookahead.

Deliberation is only a different control regime involving search over the agent program’s transition tree.

With sensing, need to find more than just a path to a final configuration, need a plan/subtree with branches for all possible sensing results.

Can determine possible sensing results by checking consistency with KB.

Essentially an EC-based approach.
Lessons for Agent Programming Languages (cont.)

As methods for implementing deliberation in restricted cases, EC-based approaches are fine.

But as a semantics or specification, we claim they are wrong.
Lessons for Agent Programming Languages (cont.)

As methods for implementing deliberation in restricted cases, EC-based approaches are fine.

But as a semantics or specification, we claim they are wrong.

\[ K_{How_{EC}} \] gives the wrong results on problems that can’t be solved in a bounded number of actions.

What’s required is something like our ET-based account.
Summary

Agent programming languages should support planning/deliberation under *incomplete information* with *sensing actions*.

We develop some non-epistemic formal models of deliberation.

The obvious model is one where agent makes decisions about what to do in terms of what is *entailed by* or *consistent with* his KB.

This model *doesn’t work* properly for problems that have no bounded solution and require iteration.

We propose an alternative *entailment and truth-based model* that works.

There are important lessons for agent programming languages in general.
Further Research

Relate this metatheoretic account of deliberation to epistemic ones.

Re-state everything in terms of other agent formalisms: 3APL, AgentSpeak, FLUX, etc.

Implementation of search/planning with non-binary sensing actions.

More general accounts of sensing and knowledge change.

Multiagent planning.

Etc.
Related Work

- **[Moore 1985]:** knowledge and plans.

- **[Davis 1994]:** Knowledge precondition for plans (executable plans).

- **[Davis & Morgenstern 2004]:** A First-Order Theory of Communication and Multi-Agent Plans.

- **[Mc Illraith & Son 2002]:** self-sufficient programs.

- **Lin & Levesque 1998]:** robot programs and effective controllers.