

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"

degiasimo@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.

High-Level Programs and Histories

Very simple *deterministic* language:

$a,$	primitive action
$\delta_1; \delta_2,$	sequence
if ϕ then δ_1 else δ_2 endif,	conditional
while ϕ do δ endWhile,	while loop

A **history** $\sigma = (a_1, \mu_1), (a_2, \mu_2), \dots, (a_n, \mu_n)$ 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 (δ, σ) with a program δ and a history σ specifying the actions performed so far and the sensing results obtained.

INDIGOLOG **Semantics for Online Executions**

Based on two notions:

- Configuration (δ, σ) may evolve to (δ', σ') w.r.t. a model M (relative to an underlying theory of action \mathcal{D}).
- A configuration is *final*, i.e. may legally terminate.

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

Model M above represents a possible environment and is just used to generate the sensing results.

Deliberation: An EC-based Account

The idea:

An agent *can/knows how* to execute program δ in history σ 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 we will define $KHow_{EC}(\delta, \sigma)$ using both **entailment** and **consistency**.

Deliberation: An EC-based Account (cont.)

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

Deliberation: An EC-based Account (cont.)

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

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

Deliberation: An EC-based Account (cont.)

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

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

(E2) if (δ, σ) *may evolve* to configurations $(\delta', \sigma \cdot (a, \mu_i))$ w.r.t. some models M_i with $k = 1..k$ 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 σ .

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 δ_{tc} :

while \neg *Down* **do** *chop*; *look* **endWhile**

solves the problem. But . . .

Formalizing the Tree Chopping E.g.: \mathcal{D}_{tc}

- \mathcal{D}_{ss} contains the following successor state axiom:

$$\begin{aligned} \text{RemainingChops}(\text{do}(a, s)) = n &\equiv \\ a = \text{chop} \wedge \text{RemainingChops}(s) = n + 1 \vee \\ a \neq \text{chop} \wedge \text{RemainingChops}(s) = n. \end{aligned}$$

- \mathcal{D}_{ap} contains the following 2 precondition axioms:

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

- \mathcal{D}_{sf} contains the following sensing axiom:

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

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

$$\text{Down}(s) \stackrel{\text{def}}{=} (\text{RemainingChops}(s) = 0)$$

Tree Chopping E.g. (cont.)

Intuitively, the following program δ_{tc} :

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

solves the problem. But . . .

Tree Chopping E.g. (cont.)

Intuitively, the following program δ_{tc} :

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

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, ...

Tree Chopping E.g. (cont.)

Intuitively, the following program δ_{tc} :

while \neg *Down* **do** *chop*; *look* **endWhile**

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 δ in σ .*

which means that ...

Tree Chopping E.g. (cont.)

Intuitively, the following program δ_{tc} :

while \neg *Down* **do** *chop*; *look* **endWhile**

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.*

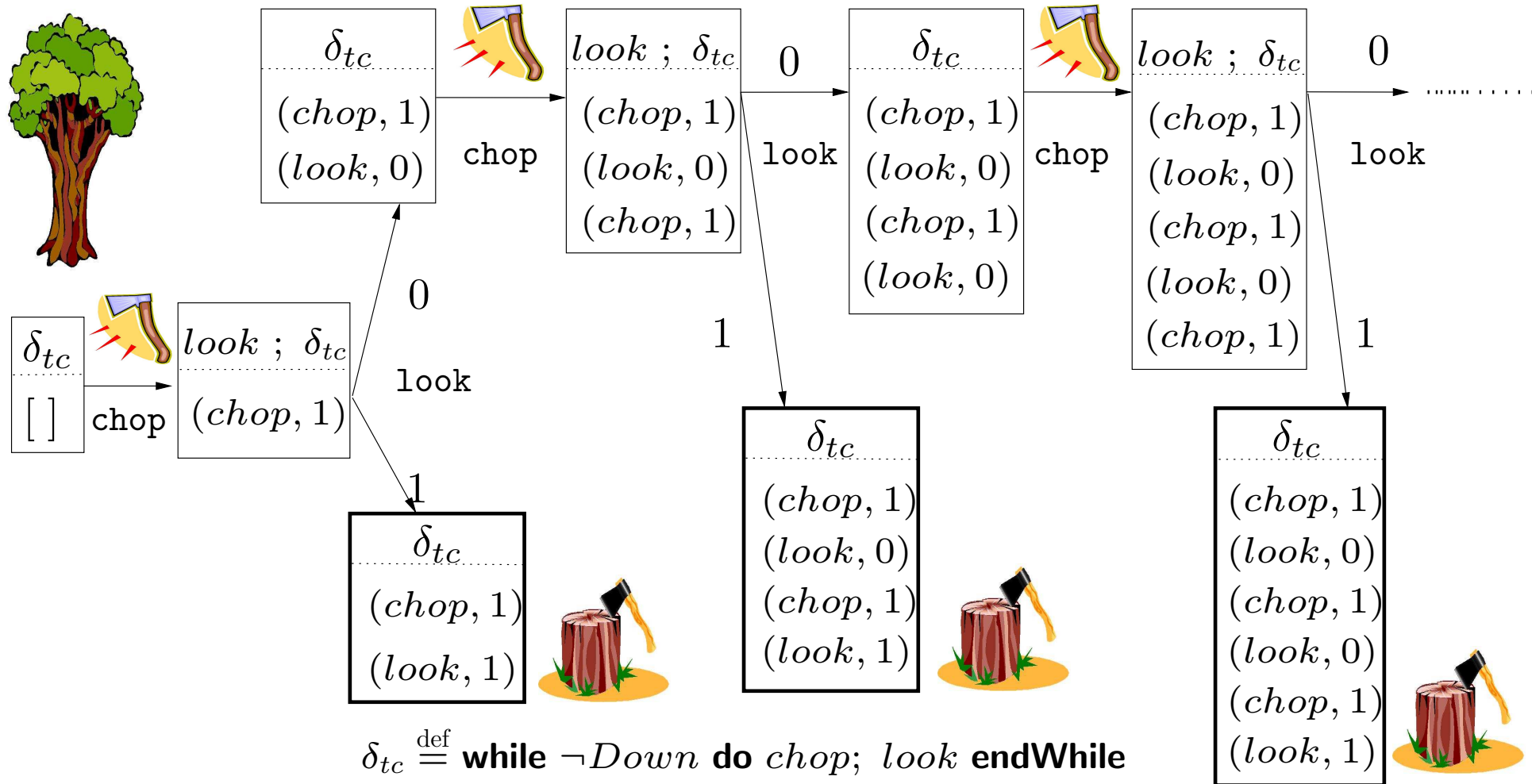
In fact, ...

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

which means that ...

Corollary 1. *$KHow_{EC}$ is only correct when problem has a bounded solution*

Why does $KHow_{EC}$ Fail?



Why does $KHow_{EC}$ Fail? (cont.)

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

$KHow_{EC}(\delta_{tc}, [(chop, 1) \cdot (look, 0)]^j)$ and

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

Why does $KHow_{EC}$ Fail? (cont.)

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

$KHow_{EC}(\delta_{tc}, [(chop, 1) \cdot (look, 0)]^j)$ and

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

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

Every finite prefix of the execution is consistent with \mathcal{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 $KHowInM(\delta, \sigma, M)$ 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 (δ, σ) is *final*, then $\mathcal{R}(\delta, \sigma)$;

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 (δ, σ) is *final*, then $\mathcal{R}(\delta, \sigma)$;

(T2) if (δ, σ) 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: iff $KHowInM(\delta, \sigma, M)$ holds for every model M of the theory + sensing at σ .

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: iff $KHowInM(\delta, \sigma, M)$ holds for every model M of the theory + sensing at σ .

$KHow_{ET}$ handles the tree chopping example.

It can be generalized to deal with non-deterministic programs.

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*.



$KHow_{EC}$ 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.

INDIGOLOG Semantics for Online Executions — Formal Details

Configuration (δ, σ) may evolve to (δ', σ') w.r.t. a model M (relative to an underlying theory of action \mathcal{D}) iff

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

$$\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma_i]\} \models Trans(\delta, end[\sigma], \delta', end[\sigma'])$$

and (iii)

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