An Incremental Interpreter for High-Level Programs with Sensing

Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
degiacomo@dis.uniroma1.it

Hector Levesque
Department of Computer Science
University of Toronto
Toronto, Canada M5S 3H5
hector@cs.toronto.edu

AAAI 1998 Fall Symposium on Cognitive Robotics, Orlando, USA, 1998.

Situation calculus

Situation calculus: Second order language for representing dynamically changing worlds [McCarthy63,McCarthy&Hayes69,Reiter91].

Language includes:

- Special symbols S_0, $do(a,s)$, $Poss(a,s)$
- Primitive actions $\mathtt{move}(b_1,b_2)$, $\mathtt{moveToTable}(b)$
- Fluents $\mathtt{On}(b_1,b_2,s)$, $\mathtt{Clear}(b,s)$ $\mathtt{OnTable}(b,s)$
- Situation independent predicates $\mathtt{Block}(b)$, $\mathtt{Red}(a)$

Basic action theory [Reiter91,Reiter98]:

- Axioms describing initial situation S_0. *The description can be incomplete.*
- Action precondition axioms (one per action).

$$Poss(\mathtt{moveToTable}(b),s) \equiv \mathtt{Clear}(b,s) \wedge \neg\mathtt{OnTable}(b,s)$$

- Successor state axioms (one per fluent). *These solve the frame problem.*

$$\begin{array}{l}\mathtt{OnTable}(b,do(a,s)) \equiv \\ \quad a = \mathtt{moveToTable}(b,s) \;\vee\; \mathtt{OnTable}(b,s) \wedge \neg\exists b'.a = \mathtt{move}(b,b')\end{array}$$

- Unique name assumption for actions.
- Some foundational domain independent axioms.

Golog/ConGolog

Golog [Levesque et al.97]: logic-based programming language. Primitive actions formalized in action theory. *GOLOG = alGOL in lOGic*

It includes following constructs:

- nil — empty program
- a — primitive action
- $\phi?$ — test condition
- $(\delta_1; \delta_2)$ — sequence
- $(\delta_1 \mid \delta_2)$ — nondeterministic choice of action
- $\pi v.\delta$ — nondeterministic choice of value
- δ^* — nondeterministic iteration
- $\mathbf{proc}\ P(\vec{v})\ \ \delta_P$ — procedure definition

ConGolog *(Concurrent GOLOG)* [IJCAI97] additionally includes:

- $\mathbf{if}\,\phi\,\mathbf{then}\,\delta_1\,\mathbf{else}\,\delta_2$ — sync. if-then-else
- $\mathbf{while}\,\phi\,\mathbf{do}\,\delta$ — sync. while-loop
- $(\delta_1 \parallel \delta_2)$ — concurrent execution
- $(\delta_1 \rangle\!\rangle\, \delta_2)$ — concurrency with different priorities
- $\delta^{\parallel}$ — concurrent iteration
- $<\phi \rightarrow \delta>$ — interrupt

In ϕ above the situations argument is suppressed.

Transition relation

An general way to formulate semantics of Golog/ConGolog programs is in terms of **single steps** (either actions or tests). Captured by two relations [IJCAI97]:

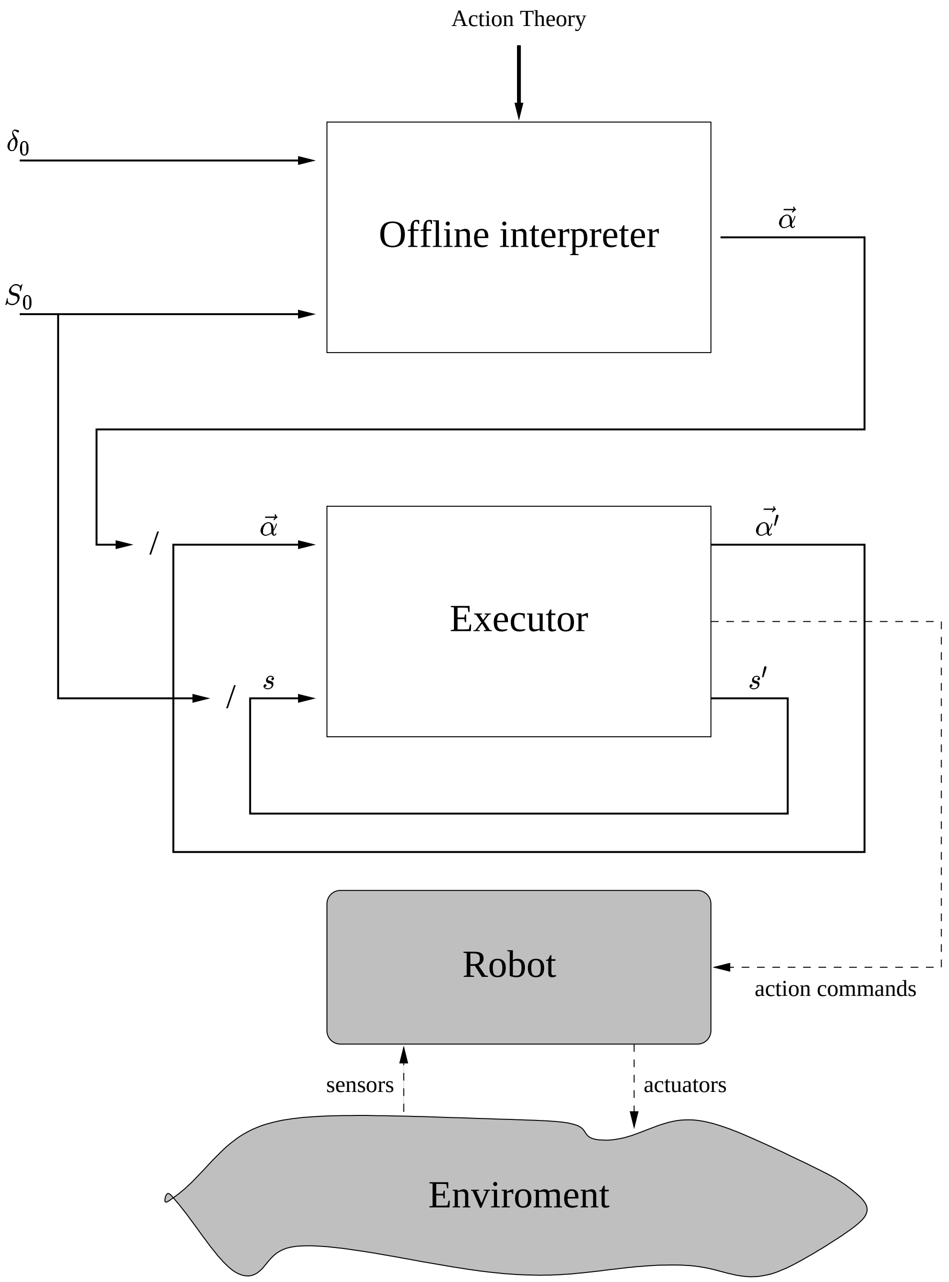
- $Trans(\delta, s, \delta', s')$ – program δ executed in situation s may perform an action resulting in situation s' with program δ' remaining to be executed.
- $Final(\delta, s)$ – program δ may be considered successfully terminated in situation s.

The relation $Do(\delta, s, s')$, which, selects all situations s' resulting from a **successfully terminating execution** of program δ starting from s, can be expressed using reflexive-transitive closure of $Trans$ – "single step the program until you can successfully terminate".

Definition of *Trans* & *Final*:

$$
\begin{array}{lcl}
Trans(nil, s, \delta', s') & \equiv & False \\
Trans(a, s, \delta', s') & \equiv & Poss(a[s], s) \wedge \delta' = nil \wedge s' = do(a[s], s) \\
Trans(\phi?, s, \delta', s') & \equiv & \phi[s] \wedge \delta' = nil \wedge s' = s \\
Trans(\delta_1; \delta_2, s, \delta', s') & \equiv & \exists \gamma. Trans(\delta_1, s, \gamma, s') \wedge \delta' = \gamma; \delta_2 \ \vee \\
 & & \quad Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta', s') \\
Trans(\pi v.\delta, s, \delta', s') & \equiv & \exists x. Trans(\delta^v_x, s, \delta', s') \\
\ldots & & \\
Trans(\delta_1 \parallel \delta_2, s, \delta', s') & \equiv & \exists \gamma. \delta' = (\gamma \parallel \delta_2) \wedge Trans(\delta_1, s, \gamma, s') \vee \\
 & & \exists \gamma. \delta' = (\delta_1 \parallel \gamma) \wedge Trans(\delta_2, s, \gamma, s') \\
\ldots & & \\
Final(nil, s) & \equiv & True \\
Final(a, s) & \equiv & False \\
Final(\phi?, s) & \equiv & False \\
Final(\delta_1; \delta_2, s) & \equiv & Final(\delta_1, s) \wedge Final(\delta_2, s) \\
\ldots & & \\
Final(\delta^*, s) & \equiv & True \\
\ldots & &
\end{array}
$$

Offline interpreter architecture



Offline interpreter (no sensing)

Phase 1:

Compute $\vec{\alpha}$ s.t. $Axioms \models Do(\delta, S_0, do(\vec{\alpha}, S_0))$.

Note: Phase 1 allows for backtracking when computing $\vec{\alpha}$!

Phase 2: (given a pre-computed $\vec{\alpha}$)

Repeat

$$(\vec{\alpha}, s) \text{ steps to } \underset{a \text{ is executed}}{(\vec{\alpha'}, do(a, s))} \quad \text{where } \vec{\alpha} = a; \vec{\alpha'}$$

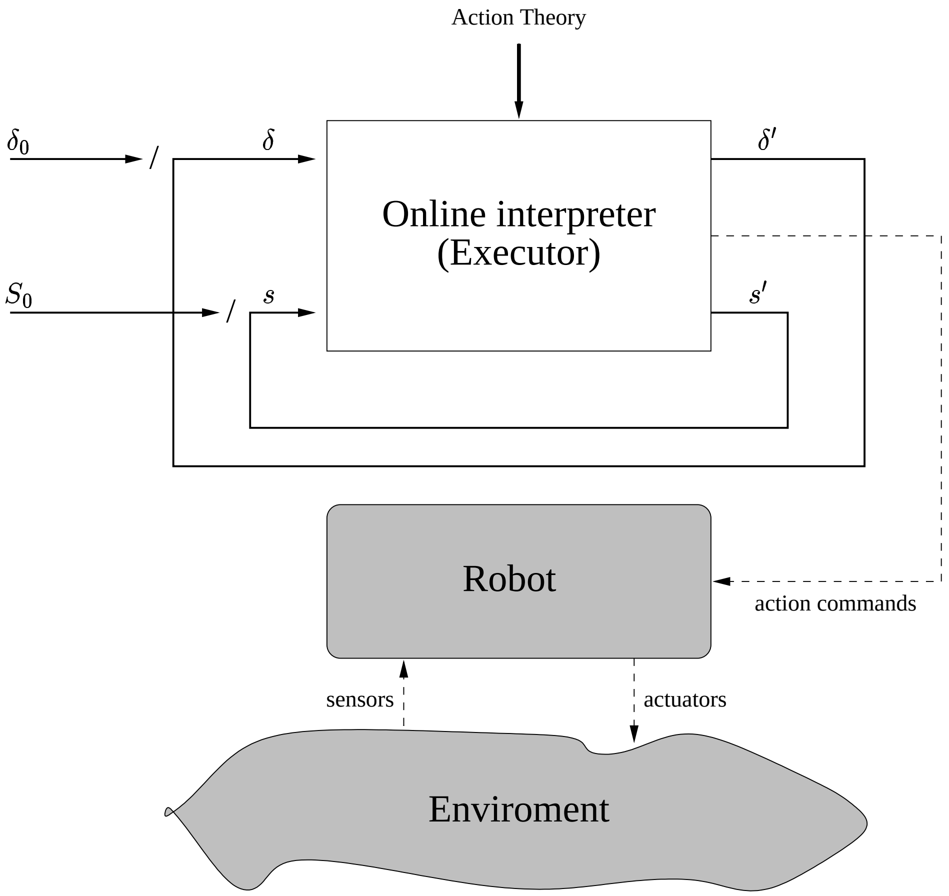
Until $\vec{\alpha} = \epsilon$.

Example:

$$(\texttt{goodAct}|\texttt{badAct}); \texttt{Good}?; bigDetProg; \texttt{Goal}?$$

Phase 1 chooses `goodAct`.

Online interpreter architecture



Online interpreter (no sensing)

Repeat

$$(\delta, s) \text{ steps to } (\delta', s')$$

if $s' = do(a, s)$, then a is executed

provided that

$$Axioms \models Trans(\delta, s, \delta', s')$$

(if more steps possible a random one is chosen)

Until $Axioms \models Final(\delta, s)$.

Note: interpreter **cannot backtrack**, because it does actions in real world.

Example:

$$(\texttt{goodAct}|\texttt{badAct}); \texttt{Good}?; bigDetProg; \texttt{Goal}?$$

Interpreter may choose `badAct`, since selecting `goodAct` may require look ahead!

Search operator

To recover the possibility of making reasoned choices in an online interpreter we introduce an explicit **search operator** within the programs.

$$\begin{array}{lcl} Trans(\Sigma(\delta), s, \delta', s') & \equiv & \exists \gamma'.\ \delta' = \Sigma(\gamma') \wedge Trans(\delta, s, \gamma', s') \wedge \\ & & \quad \exists \gamma'', s''.\ Trans^*(\gamma', s', \gamma'', s'') \wedge Final(\gamma'', s'') \\ Final(\Sigma(\delta), s) & \equiv & Final(\delta, s) \end{array}$$

The search operator essentially forces to find offline a successful computation of the piece of the program it is applied to.

It allows for great flexibility in controlling the search. We can search through the all program (as an offline computation) or we can search very locally.

Example:

$$\Sigma((\mathtt{goodAct}|\mathtt{badAct}); \mathtt{Good}?; bigDetProg; \mathtt{Goal}?)$$

Interpreter chooses `goodAct`.

$$\Sigma((\mathtt{goodAct}|\mathtt{badAct}); \mathtt{Good}?); bigDetProg; \mathtt{Goal}?$$

Interpreter chooses `goodAct`!!!.

Sensing & histories

As in [AAAI96], we introduce *sensed fluent axioms* one for each primitive action a of the form $SF(a, s) \equiv \phi_a(s)$ characterizing SF. (For ordinary action a, $SF(a, s) \equiv True$.)

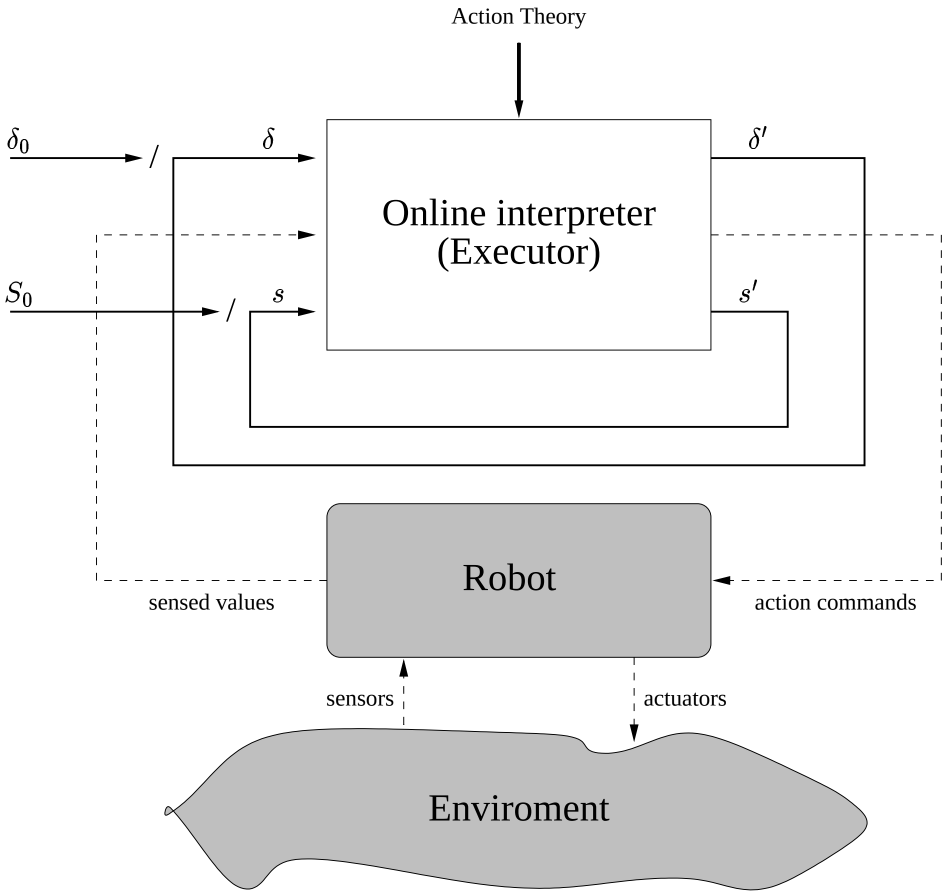
To describe a run which includes both actions and their sensing results, we use the notion of a **history**. (Histories are not terms of the situation calculus.)

By a **history** we mean a sequence of pairs (a, x) where a is a primitive action and x is 1 or 0, a sensing result. Intuitively, the history $(a_1, x_1) \cdot \ldots \cdot (a_n, x_n)$ is one where actions $a_1, \ldots, a_n$ happen starting in some initial situation, and each action a_i returns sensing value x_i.

We denote by $end[\sigma, s]$ the **end situation** of history (the situation resulting from performing the actions in the history).

We denote by $Sensed[\sigma, s]$ the **sensing results** of a history, defined inductively as: $Sensed[\epsilon, s] = True$; a $Sensed[\sigma \cdot (a, 1), s] = Sensed[\sigma, s] \wedge SF(a, end[\sigma, s])$, and $Sensed[\sigma \cdot (a, 0), s] = Sensed[\sigma, s] \wedge \neg SF(a, end[\sigma, s])$. This formula uses SF to tell us what must be true for the sensing to come out as specified by σ starting in s.

Online interpreter architecture with sensing



Online interpreter with sensing

Repeat

$$(\delta, \sigma) \text{ steps to } (\delta', \sigma')$$

if $\sigma' = (a, x) \cdot \sigma$, then a is executed and x is the sensed value returned

provided that

$$Axioms \cup Sensed[\sigma, S_0] \models Trans(\delta, end[\sigma, S_0], \delta', s')$$

$$\sigma' = \begin{cases} (a, x) \cdot \sigma & \text{if } s' = do(a, end[\sigma, S_0]) \\ \sigma & \text{if } s' = s \end{cases}$$

(if more steps possible a random one is chosen)

Until $Axioms \cup Sensed[\sigma, S_0] \models Final(\delta, end[\sigma, S_0])$.

Example:

$$\Sigma((\texttt{goodAct}|\texttt{badAct}); \texttt{Good}?); \texttt{readQ}; (\textbf{while}\ \texttt{Q}\ \textbf{do}\ \delta); \texttt{Goal}?$$

$$\text{where } SF(\texttt{readQ}, s) \equiv \texttt{Q}(s)$$

Online interpreter OK. Offline interpreter???