Cognitive Robotics: Agent Programming Languages

- Cognitive Robotics
- GOLOG and the Situation Calculus
- An Elevator Controller
- A Prolog Implementation
- Literature
Cognitive Robotics

“We believe that human intelligence depends essentially on the fact that we can represent in language facts about our situation, our goals, and the effects of the various actions we can perform. Moreover, we can draw conclusions from the facts to the effects that certain sequences of actions are likely to achieve our goals.” (John McCarthy 1963)
What’s the Goal?

► We want agents to decide what to do in order to achieve their goals:
  ▶ causality, ability, knowledge and believe
  ▶ causes, can, knows, believes
  ▶ knowledge representation and reasoning, strategy, search and control

► Applications:
  ▶ high level control of robots and industrial processes
  ▶ intelligent software agents
  ▶ discrete event simulations
  ▶ etc.
The Framework due to McCarthy (1963)

▶ “General properties of causality, and certain obvious but until now unformalized facts about the possibility and results of actions, are given as axioms.”

▶ “It is a logical consequence of the facts of a situation and the general axioms that certain persons can achieve certain goals by taking certain actions.”

▶ “The formal descriptions of situations should correspond as closely as possible to what people may reasonably be presumed to know about them when deciding what to do.”
States, Actions and Causality

- A state is the complete state of affairs of the universe at an instant of time.
- Given a state, laws of motion (or actions) determine all future states.
- Example: blocksworld.
- Neither states nor actions can be completely described.
  - inherent partiality
  - only facts about situations and actions can be specified
  - fluents
- Language: first order logic plus some extensions.
Situations and Fluents

- A **situation** is a term denoting a state. It records the history of how a state has evolved.
  - $s_0$ is called the **initial situation** and denotes the initial state.
  - $\text{do}(\text{move}(X, Y), S)$ denotes the situation obtained by executing the action $\text{move}(X, Y)$ in situation $S$.

- A **fluent** is a term denoting a fact about a situation that may change when actions are executed.
  - $\text{at}(P, X)$ denotes the fact that agent $P$ is at location $X$.
  - $\text{raining}(X)$ denotes the fact that it is raining at location $X$.

- The binary predicate **holds** is used to denote that a certain fluent holds in a particular situation.
  - $\text{holds}(\text{at}(P, X), s_0)$ denotes that agent $P$ is at location $X$ in the initial situation $s_0$. 
Frame Problem

- **Common Assumption**: Unless an action explicitly causes a fact to hold or not to hold, the facts are preserved by the action.
  - philosophical and technical aspects!
  - representational and computational aspects!
GOLOG

- Agent programming language for reasoning about the situations of the world and considering the effects of various possible plans.
- Maintains an explicit representation of the world.
- Very high level programming language.
- Planning vs. finding a sequence of actions that constitutes a legal execution of some high level non–deterministic program.
  - Reason about preconditions and effects.
  - If program is almost deterministic then there is little search.
  - As more non–determinism is included, the search resembles more and more traditional planning.
  - User controls the search effort.
Primitive Actions, Test Actions and Sequence

► $\text{do}(\delta, S, S')$ holds, whenever $S'$ is a terminating situation of an execution of a complex action $\delta$ starting in situation $S$.

► Primitive actions:

$$\text{do}(A, S, S') \overset{\text{def}}{=} \text{poss}(A, S) \land S' = \text{do}(A, S).$$

► Test actions:

$$\text{do}(\Phi?, S, S') \overset{\text{def}}{=} \text{holds}(\Phi, S) \land S = S'.$$

► Sequence:

$$\text{do}([\delta_1; \delta_2], S, S') \overset{\text{def}}{=} (\exists S^*) (\text{do}(\delta_1, S, S^*) \land \text{do}(\delta_2, S^*, S')).$$
Complex Actions: Nondeterministic Choice

- Nondeterministic choice of two actions:

\[
do([\delta_1 | \delta_2], S, S') \overset{def}{=} \do(\delta_1, S, S') \lor \do(\delta_2, S, S')
\]

- Conditionals:

\[
\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2 \text{ endIf } \overset{def}{=} [\phi?; \delta_1] \parallel [\neg \phi?; \delta_2]
\]

\[
\text{if } \text{car_in_driveway} \text{ then } \text{drive} \text{ else } \text{walk} \text{ endIf}
\]

- Nondeterministic choice of action arguments:

\[
do((\pi X) \delta(X), S, S') \overset{def}{=} (\exists X) \do(\delta(X), S, S')
\]

\[
(\pi X) \text{ remove}(X)
\]
Complex Actions: Nondeterministic Iteration

- **Nondeterministic Iteration**: Execute $\delta$ zero or more times.

  $$do(\delta^*, S, S')$$

- The macro is defined by a second order formula (see [Levesque et al., 1997]).

- While statements:

  $$\text{while } \Phi \text{ do } \delta \text{ endwhile } \overset{\text{def}}{=} \left[ [\Phi?; \delta]^*; \neg \Phi? \right]$$

  $$\text{while } (\exists B) \text{ ontable}(B) \text{ do } (\pi X) \text{ remove}(X) \text{ endwhile}$$
Complex Actions: Procedures

▶ Procedure calls:

\[ \text{do}(p(t_1, \ldots, t_n), S, S') \overset{\text{def}}{=} p(t_1[S], \ldots, t_n[S], S, S') \]

▷ Call by value: the \( t_i \) are evaluated wrt \( S \) before calling \( p \).

▶ GOLOG programs:

\[ \text{do}(\{ \text{proc } p_1(V_1)\delta_1 \text{ endProc; } \ldots; \text{proc } p_n(V_n)\delta_n \text{ endProc; } \delta_0 \}, S, S') \]

where \( p_i(V_i) \) is a procedure declaration with formal parameters \( V_i \) and \( \delta_i \) is its body, \( 1 \leq i \leq n \), and \( \delta_0 \) is the main program, ie. a complex action.

▷ The macro is defined by a second order formula (see [Levesque et al, 1997]).
Example Procedures

- **Move an elevator down** $N$ **floors:**

  \[
  \text{proc } d(N) \ [(N = 0)?[d(N - 1); \textit{down}]] \text{ endProc,}
  \]

  where \textit{down} moves an elevator down one floor.

- **Park an elevator on the ground floor:**

  \[
  \text{proc } \textit{park} (\pi m)[\textit{atfloor}(m)?; \textit{down}(m)] \text{ endProc.}
  \]

- **Define test action** \textit{above} **as the transitive closure of** \textit{on}.

  \[
  \text{proc } \textit{above}(x, y) \ [(x = y)?(\pi z)[\textit{on}(x, z)?; \textit{above}(z, y)]] \text{ endProc.}
  \]
What can be done and what cannot be done?

Let $\mathcal{F}$ be a GOLOG specification.

What can be done? Proving properties of a given $\delta$:

- **Correctness:**
  \[ \mathcal{F} \models (\forall S) \ (\text{do}(\delta, s_0, S) \rightarrow p(S)) \]
  or, even stronger
  \[ \mathcal{F} \models (\forall S_0, S) \ (\text{do}(\delta, S_0, S) \rightarrow p(S)). \]

- **Termination:**
  \[ \mathcal{F} \models (\exists S) \ \text{do}(\delta, s_0, S) \]
  or, even stronger
  \[ \mathcal{F} \models (\forall S_0) \ (\exists S) \ \text{do}(\delta, S_0, S). \]

What cannot be done? Synthesizing complex actions:

\[ \mathcal{F} \models (\exists \delta, S') \ (\text{do}(\delta, s_0, S') \land \text{Goal}(S')) \]

is not even a well–formed expression!
An Elevator Controller: Primitive Actions and Fluents

► **Primitive actions:**

- $up(N)$ denotes the movement of the elevator up to floor $N$.
- $down(N)$ denotes the movement of the elevator down to floor $N$.
- $turnoff(N)$ denotes the turning off of the call button $N$.
- $open$ denotes the opening of the elevator door.
- $close$ denotes the closing of the elevator door.

► **Fluents:**

- $current\_floor(N)$ denotes that the elevator is at floor $N$.
- $on(N)$ denotes that call button $N$ is on.
- $next\_floor(N)$ denotes that the next floor to be served is $N$.

► **Primitive action preconditions:**

- $poss(up(N), S) \iff (\exists M) \ (holds(current\_floor(M), S) \land M < N)$.
- $poss(down(N), S) \iff (\exists M) \ (holds(current\_floor(M), S) \land M > N)$.
- $poss(open, S) \iff \langle \rangle$.
- $poss(close, S) \iff \langle \rangle$.
- $poss(turnoff(N), S) \iff holds(on(N), S)$. 

Cognitive Robotics: Action Programming Languages
An Elevator Controller: Successor State Axioms

**Concerning current_floor:**

\[
\text{(poss}(A, S) \\
\rightarrow \\
\text{(holds(current_floor}(M), \text{do}(A, S)) \\
\leftrightarrow \\
\text{(A = up}(M) \lor A = \text{down}(M) \\
\lor \\
\text{holds(current_floor}(M), S) \land \neg(\exists N) A = \text{up}(N) \land \neg(\exists N) A = \text{down}(N))))).
\]

**Concerning on:**

\[
\text{(poss}(A, S) \\
\rightarrow \\
\text{(holds(on}(M), \text{do}(A, S)) \leftrightarrow \text{holds(on}(M), S) \land A \neq \text{turnoff}(M))).
\]
An Elevator Controller: A Defined Fluent

- Selecting the next floor to be served:

\[
(\text{holds}(\text{next\_floor}(N), S) \leftrightarrow (\text{holds}(\text{on}(N), S) \land (\forall M, L) (\text{holds}(\text{on}(M), S) \land \text{holds}(\text{current\_floor}(L), S) \rightarrow |M - L| \geq |N - L|))).
\]
An Elevator Controller: The Procedures

- The procedures:

  proc $\text{serve}(N)$ [\text{go\_floor}(N); \text{turnoff}(N); \text{open}; \text{close}] endProc.
  proc $\text{go\_floor}(N)$ [\text{current\_floor}(N)?|\text{up}(N)|\text{down}(N)] endProc.
  proc $\text{serve\_a\_floor}(\pi N)[\text{next\_floor}(N)?; \text{serve}(N)]$ endProc.
  proc $\text{control}$ [\text{while} (\exists N) \text{on}(N) \text{do} \text{serve\_a\_floor} \text{endWhile}; \text{park}] endProc.
  proc $\text{park}$ if $\text{current\_floor}(0)$ then \text{open} else \text{down}(0); \text{open} endIf endProc.

- Initial situation:

  $\text{holds}(\text{current\_floor}(4), s_0) \land \text{holds}(\text{on}(5), s_0) \land \text{holds}(\text{on}(3), s_0)$
Reasoning about the Elevator Controller

- Let $\mathcal{F}$ be the corresponding GOLOG specification, then e.g.

$$\mathcal{F} \models (\exists S) \ do(\Pi; \ control, s_0, S),$$

where $\Pi$ is the sequence of procedure definitions.

- A successful proof might return the substitution

$$S = \ do(\ open, \ do(\down (0), \ do(\ close, \ do(\ open, \ do(\ turnoff(5),
\ do(\ up(5), \ do(\ close, \ do(\ open, \ do(\ turnoff(3), \ do(\ down(3), s_0))))))))))).$$

- $\ do(\Pi; \ control, s_0, S)$ is a second–order formula!

- Interpreter implemented in first–order Prolog.

What precisely is the relation between $do$ and its counterpart $\text{do}$ in Prolog?
Applications

- Elevator controller.
- Serving coffee in an office environment.
- An autonomous museum tour guide.
- Personal home banking assistant (Lespérance et al: 1997).
GOLEX

- Bridging the gap between cognitive robotics and real robots, in particular, between GOLOG and RHINO.
- Decompose primitive actions specified in GOLOG into a sequence of directives for the low-level robot control system.

\[
\text{exec(go}(L)\text{)} \leftarrow \text{position}(L, (X, Y)), \\
\text{pan_tilt_set_track_point}((X, Y)), \\
\text{target_message}(L, M), \\
\text{speech_talk_text}([\text{"please follow me to"}, L]), \\
\text{sound_play}(\text{horn}), \\
\text{robot_drive_path}([X, Y]), \\
\text{robot_turn_to_point}((X, Y)).
\]
GOLEX – Continued

- **Execution monitoring:**
  - stops the execution of an action if timed out,
  - verifies that primitive actions have been successively carried out,
  - ensures that the world is consistent with GOLOG’s model
  - etc.

- **Simple forms of sensing and acting:**
  - simple forms of speech,
  - accepts confirmations,
  - accepts simple forms of user input
  - etc.

- **Applications:**
  - Museum tour guide,
  - Coffee delivery agent.
Literature


