Local Conditional High-Level Programs

Sebastian Sardiña
Cognitive Robotics Group
Department of Computer Science
University of Toronto
Toronto, CANADA

ssardina@cs.toronto.edu
High-Level Programs for Agents

Alternative to first principle planning (e.g., Golog, ConGolog)

A nondeterministic program stands for a scheme of the solution whose gaps have to be filled in.

- domain-dependent actions: $\text{goto}(A)$, $\text{openDoor}$, $\text{board\_plane}$, ...
- tests with domain-dependent fluents: if $\text{gate} = 90$ then ... else ....
- nondeterministic points: $\text{buy}(coffee) | \text{buy}(magazine)$
- semantics in the situation calculus
Airport Example (Golog)

(Lakemeyer 1998)

\textbf{proc} catch\_plane

\[(\pi a.a)^*; \ \text{at(airport)}?;\]
\[(\text{goto(terminal1)} \mid \text{goto(terminal2)})\];
\texttt{look\_at\_panel}; \quad /* \text{Sensing Action} */
\[(\text{buy(magazine)} \mid \text{buy(paper)})\];
\texttt{if} gate \geq 90 \texttt{then} \{ \text{goto(gate)}; \text{buy(coffee)} \} \texttt{else}
\quad \{ \text{buy(coffee)}; \text{goto(gate)} \}
\texttt{board\_plane};

\textbf{end\_proc}

Golog and ConGolog do not deal with sensing
Incremental Execution of Programs

(Single step semantics)

$Trans(\delta, s, \delta', s')$: program $\delta$ in situation $s$ may legally execute one step, ending in situation $s'$ with program $\delta'$ remaining.

$Final(\delta, s)$: program $\delta$ may legally terminate in situation $s$.

- An (online) execution is a sequence of $Trans'$ followed by a $Final$;
- After each step, sensing information may be collected;
- Each step is executed in the real world $\rightarrow$ no backtracking.

\[ e.g., \ Trans(a, s, \delta', s') \equiv Poss(a, s) \land \delta' = nil \land s' = do(a, s) \]
Local Offline Verification with Search $\Sigma$

(De Giacomo & Levesque 1998)

select the next action that will guarantee
the existence of some successful execution

$$Final(\Sigma\delta, s) \equiv Final(\delta, s)$$

$$Trans(\Sigma\delta, s, \delta', s') \equiv \exists \gamma, \gamma'. \delta' = \Sigma\gamma \land Trans(\delta, s, \gamma, s') \land Trans*(\gamma, s', \gamma', s'') \land Final(\gamma', s'')$$

$Trans^*$: transitive reflexive closure of $Trans$
Drawbacks of $\Sigma$

- $\Sigma$ does not calculate complete plans
- $\Sigma$ does not distinguish between $\Sigma \delta$ and $\Sigma \delta'$:

  $\delta = A; \text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2$

  $\delta' = A; \text{Sense}_\phi; \text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2$

  where $\phi$ is unknown initially, and both $\delta_1$ and $\delta_2$ are executable.

Both $\Sigma \delta$ and $\Sigma \delta'$ will first execute action $A$. 
Conditional Planning and sGolog

Contingency plans to tackle incomplete knowledge
e.g., CNLP, X11, Cassandra, C-BURIDIAN, etc.

sGolog → conditional version of Golog
→ compute conditional action trees (CATs)
→ semantics via macro expansion in the situation calculus
Developing a Conditional Search I

A Golog program $\delta_{CPP}$ is a *conditional program plan* (CPP) if

- $\delta_{CPP} = \text{nil}$ or $\delta_{CPP} = A$;
- $\delta_{CPP} = (A; \delta_1)$, and $\delta_1$ is a CPP;
- $\delta_{CPP} = \text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2$, $\phi$ is a fluent formula and $\delta_1, \delta_2$ are CPPs.

$\text{condPlan}(\delta)$: $\delta$ is a CPP.
Developing a Conditional Search II

\[ \text{run}(\delta_{\text{CPP}}, s) : \text{situation representing the execution of } \delta_{\text{CPP}} \text{ from } s. \]

\[
\begin{align*}
\text{run}(a, s) & = \text{do}(a, s) \\
\text{run}((a; \delta), s) & = \text{run}(\delta, \text{do}(a, s)) \\
\text{run}(\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2, s) & = \text{if } \phi(s) \text{ then } \text{run}(\delta_1, s) \\
& \quad \text{else } \text{run}(\delta_2, s)
\end{align*}
\]

\[ \text{knowHow}(\delta_{\text{CPP}}, s) : \text{is the agent “able” to execute } \delta_{\text{CPP}}? \]

\[
\begin{align*}
\text{knowHow}((a; \delta), s) & \equiv \text{knowHow}(\delta, \text{do}(a, s)) \\
\text{knowHow}(\text{if } \phi(s) \text{ then } \delta_1 \text{ else } \delta_2, s) & \equiv K\text{whether}(\phi, s) \land \\
& \quad \phi(s) \sqsupset \text{knowHow}(\delta_1, s) \land \\
& \quad \neg \phi(s) \sqsupset \text{knowHow}(\delta_2, s)
\end{align*}
\]
Definition of $\Sigma_c$

A single transition of $\Sigma_c(\delta)$ returns a conditional plan compatible with $\delta$

\[
\begin{align*}
\text{Final}(\Sigma_c \delta, s) & \equiv \text{Final}(\delta, s) \\
\text{Trans}(\Sigma_c \delta, s, \delta', s') & \equiv s' = s \land \text{condPlan}(\delta') \land \text{knowHow}(\delta', s) \land \\
& \quad \exists \delta''. \text{Trans}^*(\delta, s, \delta'', \text{run}(\delta', s)) \land \text{Final}(\delta'', \text{run}(\delta', s))
\end{align*}
\]

The returned $\delta'$ ...

- is always a CPP and one that is possible to execute;
- represents the original program $\delta$ faithfully;
- is deterministic, has no search, and no concurrency.
Solutions for $\Sigma_c (catch\_plane)$

$Axioms \models Trans(\Sigma_c catch\_plane, S_0, \delta', S_0)$

$\delta' = goto(airport); goto(terminal2); look\_at\_panel; buy(paper);
if gate \geq 90 then \{goto(gate); buy(coffee); board\_plane\}
else \{buy(coffee); goto(gate); board\_plane\}$

- $\delta'$ is similar to what sGolog would return;
- there are many other solutions w.r.t. $\Sigma_c$
sGolog and $\Sigma_c$

**Theorem:** All solutions of sGolog are solutions of $\Sigma_c$

**PLUS**

- $\Sigma_c$ solves programs with concurrency;
- $\Sigma_c$ fits in an interleaved account of execution;
- $\Sigma_c$ branches “automatically”;
- no need for a new class of terms: CPP are regular Golog programs.
Implementing $\Sigma_c$

**Problem:** how and where should we split?

**Solution:** rely on the programmer (as in [Lakemeyer 1998])

**How:** a restrictive $\Sigma_{cb}$ that splits only w.r.t. special action $branch(\phi)$

```plaintext
proc catch_plane2
  (πa.a)*; at(airport)?;
  (goto(terminal1) | goto(terminal2));
  look_at_panel;        /* Sensing Action! */
  (buy(magazine) | buy(paper));
  branch(gate ≥ 90);
  if gate ≥ 90 then { goto(gate); buy(coffee) } else 
    { buy(coffee); goto(gate) }
  board_plane;
end_proc
```
Definition of $\Sigma_{cb}$

$\Sigma_{cb}$ splits **only** when a $\text{branch}(\phi)$ action is encountered.

Only two modifications are required:

- Special action $\text{branch}(\phi)$ is treated as a regular primitive action;
- Modify last axiom for $\text{run}(\delta, s)$:

  $$\text{run}(\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2) = \text{if } \phi(s) \text{ then } \text{do}(\text{branch}(\phi), \text{run}(\delta_1, s))$$
  $$\text{else } \text{do}(\text{branch}(\phi), \text{run}(\delta_2, s))$$

**Theorem:** sGolog and $\Sigma_{cb}$ compute the **same solutions** for Golog programs.
Implementing $\Sigma_{cb}$

- \textbf{kw}ether($P, S$): is fluent $P$ known at $S$?
- \textbf{trans}/4 and \textbf{final}/2 for all ConGolog constructs
- $\text{branch}(\phi)$ is restricted to relational fluents, i.e. $\phi = F$
- on a $\text{branch}(F)$ action, both truth values are conceivable

\textbf{Goal:} $\text{trans}(\text{search}cb(\delta), s, \text{CPP}, S)$

(1) If $G$ succeeds with answer $\text{CPP} = p', S = s'$, then $p'$ is a CPP, $s' = s$,

\begin{align*}
\text{Axioms} & \models \ Trans(\Sigma_c p, s, p', s') \\
\text{Axioms} & \models \ Trans(\Sigma_{cb} p, s, p', s')
\end{align*}

(2) If $G$ finitely fails, then

$\text{Axioms} \models \forall p', s'. \neg Trans(\Sigma_{cb} p, s, p', s')$
Conclusions

A new construct for ConGolog that ...

- provides conditional offline planning to incremental executions;
- is very simple: only two new axioms for Trans and Final;
- handles all ConGolog: solves non-determinism and concurrency;
- calculates deterministic ready-to-execute plans;
- deals with knowledge producing actions.

**BUT**

- what about generation of more general plans (e.g., loops)?
- can we develop better ways of splitting?
- how to use search in programs?
Transforming a Complex Program

Want an operator $\Sigma_c(\delta)$ that can transform an arbitrary nondeterministic concurrent program $\delta$ into a simple conditional program.
Solutions for $\Sigma_c(catch\_plane)$

$$Axioms \models Trans(\Sigma_c catch\_plane, S_0, \delta', S_0)$$

$$\delta' = goto(airport); goto(terminal2); look\_at\_panel; buy(paper);$$

if $gate \geq 90$ then $\{goto(gate); buy(coffee); board\_plane\}$
else $\{buy(coffee); goto(gate); board\_plane\}$

$$\delta' = goto(airport); goto(terminal2); look\_at\_panel; buy(paper);$$

if $gate \geq 90$ then $\{goto(gate); buy(coffee); board\_plane\}$
else [ if $(p \lor \neg p)$ then
$\{buy(coffee); goto(gate); board\_plane\}$
else $\{buy(coffee)\} \]
**Incremetal Execution and $\Sigma_C$**

\[\begin{align*}
Axioms \cup Sensed[s_0] & \models Trans(\delta_0, s_0, \delta_1, s_1) \\
Axioms \cup Sensed[s_1] & \models Trans(\delta_1, s_1, \delta_2, s_2) \\
\vdots \\
\Rightarrow Axioms \cup Sensed[s_k] & \models Trans((\Sigma_c \delta_k); \delta', s_k, (\delta_{CPP}; \delta'), s_k) \\
\vdots \\
\text{deterministic} \\
\text{execution of } \delta_{CPP} \\
\vdots \\
Axioms \cup Sensed[s_j] & \models Trans(\delta', s_j, \delta'', s_{j+1}) \\
\vdots
\end{align*}\]
Prolog Code for $\Sigma_{cb}$

```prolog
trans(searchcb(E),S,CPP,S):- build_cpp(E,S,CPP).

build_cpp(E,S,[]) :- final(E,S).
build_cpp([E1|E2],S,C):- !, build_cpp(E1,S,C1),
                     ext_cpp(E2,S,C1,C).
build_cpp(branch(F),S,if(F,[],[],[])):- !, kwhether(F,S).
build_cpp(E,S,C) :- trans(E,S,E1,[branch(P)|S]),
                  build_cpp([branch(P)|E1],S,C).
build_cpp(E,S,C) :- trans(E,S,E1,S), do(E1,S,C).
build_cpp(E,S,[A|C1]) :- trans(E,S,E1,[A|S]), A \= branch(P),
                        build_cpp(E1,S1,C1).

ext_cpp(E,S,[A|C],[A|C2]):- action(A),ext_cpp(E,[A|S],C,C2).
ext_cpp(E,S,if(P,C1,C2),if(P,C3,C4)):-
                      ext_cpp(E,[asm(P,true)|S],C1,C3),
                      ext_cpp(E,[asm(P,false)|S],C2,C4).
ext+cpp(E,S,[]),C):- build_cpp(E,S,C). /* leaf of CPP */
```
Programs with Concurrency (ConGolog)

```plaintext
proc catch_plane
(have_drink \land thirsty \rightarrow drink) \}
\[ (\pi a.a)^* ; \text{at(airport)}? ;
\text{goto(terminal1)} | \text{goto(terminal2)});
look\_at\_panel; /* Sensing Action */
\text{(buy(magazine)} | \text{buy(paper)}) ;
\text{if \_gate} \geq 90 \text{ then } \{ \text{goto(gate)} ; \text{buy(coffee)} \} \text{ else } \{ \text{buy(coffee)} ; \text{goto(gate)} \}
\text{board\_plane;]}
end_proc

\delta_{CPP} = \text{goto(airport)} ; \text{goto(terminal2)} ; \text{look\_at\_panel} ; \text{buy(paper)} ;
\text{if \_gate} \geq 90 \text{ then } \{ \text{goto(gate)} ; \text{buy(coffee)} ; \text{drink} ; \text{board\_plane} \}
\text{ else } \{ \text{buy(coffee)} ; \text{goto(gate)} ; \text{drink} ; \text{board\_plane} \}
```
Epistemic Search: $\Sigma_e$

$\Sigma_e(\delta)$ computes a deterministic epistemically feasible strategy $dp$ compatible with $\delta$.

$$Trans(\Sigma_e(\delta), s, dp', s') \equiv$$
$$\exists dp. EFDP(dp, s) \land$$
$$\exists s_f. Trans(dp, s, dp', s') \land Do(dp', s', s_f) \land Do(\delta, s, s_f)$$

$$EFDP(dp, s) \overset{\text{def}}{=} \forall dp', s'. Trans^*(dp, s, dp', s') \supset LEFDP(dp', s')$$

$$LEFDP(dp, s): \text{agent knows } dp \text{ is final or knows a transition}$$