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: $\text{if } gate = 90 \text{ then ... else }$....
- nondeterministic points: $\text{buy(}coffee\text{)}|\text{buy(}magazine\text{)}$
- semantics in the situation calculus
Airport Example (Golog)

(Lakemeyer 1998)

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

\textendproc

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

(Single step semantics)

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

Final(δ, s): program δ 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 → no backtracking.

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

(De Giacomo & Levesque 1998)

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

$$\text{Final}(\Sigma\delta, s) \equiv \text{Final}(\delta, s)$$

$$\text{Trans}(\Sigma\delta, s, \delta', s') \equiv \exists \gamma, \gamma', s''. \delta' = \Sigma\gamma \land \text{Trans}(\delta, s, \gamma, s') \land$$

$$\text{Trans}^*(\gamma, s', \gamma, s'') \land \text{Final}(\gamma', s'')$$

$\text{Trans}^*$: transitive reflexive closure of $\text{Trans}$
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

- $\delta_{CPP} = nil$ 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

\textit{run}(\delta_{\text{CPP}}, s): \textit{situation representing the execution of } \delta_{\text{CPP}} \textit{ 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*}

\textit{knowHow}(\delta_{\text{CPP}}, s): \textit{is the agent “able” to execute } \delta_{\text{CPP}}?\n
\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) \supset \text{knowHow}(\delta_1, s) \land \\
& \quad \neg \phi(s) \supset \text{knowHow}(\delta_2, s)
\end{align*}
Definition of $\Sigma_c$

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

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

$$Trans(\Sigma_c \delta, s, \delta', s') \equiv s' = s \land \text{condPlan}(\delta') \land \text{knowHow}(\delta', s) \land \exists \delta''. \text{Trans}^*(\delta, s, \delta'', \text{run}(\delta', s)) \land Final(\delta'', \text{run}(\delta', s))$$

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 $\Sigma_c(\text{catch\_plane})$

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

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

- $\delta'$ 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 the programmer (as in [Lakemeyer 1998])

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

**proc** catch\_plane2

$\pi a.a)^*); at(airport)?;

(goto(terminal1) | goto(terminal2));

look\_at\_panel; /* Sensing Action! */

(buy(magazine) | buy(paper));

branch(gate \geq 90);

if gate \geq 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 $\Sigma_{cb}$

- $\text{whether}(P,S)$: is fluent $P$ known at $S$?
- $\text{trans}/4$ and $\text{final}/2$ 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

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

1. If $G$ succeeds with answer $CPP = p', S = s'$, then $p'$ is a CPP, $s' = s$,
   
   \[ \text{Axioms} \models \text{Trans}(\Sigma_{cp} p, s, p', s') \]
   \[ \text{Axioms} \models \text{Trans}(\Sigma_{cb} p, s, p', s') \]

2. If $G$ finitely fails, then
   
   \[ \text{Axioms} \models \forall p', s'. \neg \text{Trans}(\Sigma_{cb} p, s, p', s') \]
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 \( \Sigma_c \)**

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

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

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

\text{board\_plane;}]
end\_proc

\[
\delta_{CPP} = \text{goto(airport)}; \text{goto(terminal2)}; \text{look\_at\_panel}; \text{buy(paper)};
\]
if \text{gate} \geq 90 then \{ \text{goto(gate)}; \text{buy(coffee); drink; board\_plane} \}
else \{ \text{buy(coffee); goto(gate); drink; board\_plane} \}