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

(Lakemeyer 1998)

\textbf{proc} catch\_plane
\begin{align*}
(\pi a.a)^*; & \text{at(airport)?;} \\
(\text{goto(terminal1)} & | \text{goto(terminal2)}) \\
\text{look\_at\_panel;} & \quad /* \text{Sensing Action} */ \\
(\text{buy(magazine)} & | \text{buy(paper)}) \\
\text{if} & \text{gate} \geq 90 \text{ then } \{ \text{goto(gate)}; \text{buy(coffee)} \} \quad \text{else} \\
& \quad \{ \text{buy(coffee)}; \text{goto(gate)} \} \\
\text{board\_plane} \\
\text{end\_proc}
\end{align*}

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

(Single step semantics)

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

\( \text{Final}(\delta, s) \): program \( \delta \) may legally terminate in situation \( s \).

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

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

(De Giacomo & Levesque 1998)

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

\[ Final(\Sigma \delta, s) \equiv Final(\delta, s) \]

\[ Trans(\Sigma \delta, s, \delta', s') \equiv \exists \gamma, \gamma', s''. \delta = \Sigma \gamma \wedge Trans(\delta, s, \gamma, s') \wedge Trans^*(\gamma, s', \gamma', s'') \wedge Final(\gamma', s'') \]

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

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

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

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

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

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

\[
\text{run}(a, s) = \text{do}(a, s)
\]
\[
\text{run}((a; \delta), s) = \text{run}(\delta, \text{do}(a, s))
\]
\[
\text{run}(\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2, s) = \text{if } \phi(s) \text{ then } \text{run}(\delta_1, s) \text{ else } \text{run}(\delta_2, s)
\]

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

\[
\text{knowHow}((a; \delta), s) \equiv \text{knowHow}(\delta, \text{do}(a, s))
\]
\[
\text{knowHow(\text{if } \phi(s) \text{ then } \delta_1 \text{ else } \delta_2, s) \equiv } \text{Kwhether}(\phi, s) \land \phi(s) \supset \text{knowHow}(\delta_1, s) \land \neg\phi(s) \supset \text{knowHow}(\delta_2, s)
\]
Definition of $\Sigma_C$

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

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

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

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

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

- $\delta'$ 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 $\text{branch}(\phi)$

```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 $branch(\phi)$ action is encountered.

Only two modifications are required:

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

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

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

- $k\text{whether}(P,S)$: 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:** $\text{trans}(\text{searchcb}(\delta), s, \text{CPP}, S)$

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

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

(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 $\Sigma_c(catch\_plane)$

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

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

\[ \delta' = goto(airport); goto(terminal2); look\_at\_panel; buy(paper); \]
\[ \quad \text{if } gate \geq 90 \text{ then } \{ goto(gate); buy(coffee); board\_plane \} \]
\[ \quad \text{else } [ \text{if } (p \lor \neg p) \text{ then } \]
\[ \quad \quad \{ buy(coffee); goto(gate); board\_plane \} \]
\[ \quad \text{else } \{ buy(coffee) \} ] \]
Incremental 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 $\Sigma_{cb}$

\[
\begin{align*}
\text{trans}(\text{searchcb}(E),S,\text{CPP},S) &: \text{ build_cpp}(E,S,\text{CPP}). \\
\text{build_cpp}(E,S,[[]]) & : \text{ final}(E,S). \\
\text{build_cpp}([E_1|E_2],S,C) &: !, \text{ build_cpp}(E_1,S,C_1), \\
& \quad \text{ext_cpp}(E_2,S,C_1,C). \\
\text{build_cpp}(\text{branch}(F),S,\text{if}(F,[[],[]])) &: !, \text{ kwhether}(F,S). \\
\text{build_cpp}(E,S,C) & : \text{ trans}(E,S,E_1,[\text{branch}(P)|S]), \\
& \quad \text{build_cpp}([\text{branch}(P)|E_1],S,C). \\
\text{build_cpp}(E,S,C) & : \text{ trans}(E,S,E_1,S), \text{ do}(E_1,S,C). \\
\text{build_cpp}(E,S,[A|C_1]) & : \text{ trans}(E,S,E_1,[A|S]), A \neq \text{ branch}(P), \\
& \quad \text{build_cpp}(E_1,S_1,C_1).
\end{align*}
\]

\[
\begin{align*}
\text{ext_cpp}(E,S,[A|C],[A|C_2]) &: \text{ action}(A), \text{ ext_cpp}(E,[A|S],C,C_2). \\
\text{ext_cpp}(E,S,\text{if}(P,C_1,C_2),\text{if}(P,C_3,C_4)) &: - \\
& \quad \text{ext_cpp}(E,\text{asm}(P,\text{true})|S],C_1,C_3), \\
& \quad \text{ext_cpp}(E,\text{asm}(P,\text{false})|S],C_2,C_4). \\
\text{ext+cpp}(E,S,[[],C]) &: \text{ build_cpp}(E,S,C). \text{ /* leaf of CPP */}
\end{align*}
\]
Programs with Concurrency (ConGolog)

proc catch_plane
    (have_drink ∧ thirsty → drink) ⟩⟩
    [⟨πa.a⟩* ; at(airport)?;
    (goto(terminal1) | goto(terminal2));
    look_at_panel; /* Sensing Action */
    (buy(magazine) | buy(paper));
    if gate ≥ 90 then { goto(gate); buy(coffee) } else
    { buy(coffee); goto(gate) }
    board_plane;]
end_proc

δ_{CPP} = goto(airport); goto(terminal2); look_at_panel; buy(paper);
    if gate ≥ 90 then { goto(gate); buy(coffee); drink; board_plane}
    else { buy(coffee); goto(gate); drink; board_plane}
Epistemic Search: $\Sigma_e$

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

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

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

$LEFDP(dp, s)$: agent knows $dp$ is final or knows a transition