

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: *goto(A), openDoor, board_plane, ...*
- tests with domain-dependent fluents: **if** *gate = 90* **then ... else ...**
- nondeterministic points: *buy(coffee)|buy(magazine)*
- semantics in the situation calculus

Airport Example (Golog)

(Lakemeyer 1998)

```
proc catch_plane
  ( $\pi a.a$ )*; at(airport)?;
  (goto(terminal1) | goto(terminal2));
  look_at_panel;          /* Sensing Action */
  (buy(magazine) | buy(paper));
  if gate  $\geq$  90 then { goto(gate); buy(coffee) } else
                        { buy(coffee); goto(gate) }
  board_plane;
end_proc
```

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 \rightarrow **no backtracking**.

$$\text{e.g., } Trans(a, s, \delta', s') \equiv Poss(a, s) \wedge \delta' = nil \wedge s' = do(a, s)$$

Local Offline Verification with Search Σ

(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 Σ

- Σ does not calculate complete plans
- Σ 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 ϕ is **unknown** initially, and both δ_1 and δ_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 δ_{CPP} is a *conditional program plan* (CPP) if

- $\delta_{CPP} = nil$ or $\delta_{CPP} = A$;
- $\delta_{CPP} = (A; \delta_1)$, and δ_1 is a CPP;
- $\delta_{CPP} = \mathbf{if } \phi \mathbf{ then } \delta_1 \mathbf{ else } \delta_2$, ϕ is a fluent formula and δ_1, δ_2 are CPPs.

condPlan(δ): δ is a CPP.

Developing a Conditional Search II

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

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

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

$$\begin{aligned} knowHow((a; \delta), s) &\equiv knowHow(\delta, do(a, s)) \\ knowHow(\text{if } \phi(s) \text{ then } \delta_1 \text{ else } \delta_2, s) &\equiv \textit{Kwhether}(\phi, s) \wedge \\ &\quad \phi(s) \supset knowHow(\delta_1, s) \wedge \\ &\quad \neg\phi(s) \supset knowHow(\delta_2, s) \end{aligned}$$

Definition of Σ_c

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

$$\begin{aligned}
 \textit{Final}(\Sigma_c\delta, s) &\equiv \textit{Final}(\delta, s) \\
 \textit{Trans}(\Sigma_c\delta, s, \delta', s') &\equiv s' = s \wedge \textit{condPlan}(\delta') \wedge \textit{knowHow}(\delta', s) \wedge \\
 &\quad \exists \delta''. \textit{Trans}^*(\delta, s, \delta'', \textit{run}(\delta', s)) \wedge \textit{Final}(\delta'', \textit{run}(\delta', s))
 \end{aligned}$$

The returned δ' ...

- is always a CPP and one that is possible to execute;
- represents the original program δ faithfully;
- is deterministic, has no search, and no concurrency.

Solutions for $\Sigma_c(\textit{catch_plane})$

$$\textit{Axioms} \models \textit{Trans}(\Sigma_c \textit{catch_plane}, S_0, \delta', S_0)$$

$\delta' = \textit{goto}(\textit{airport}); \textit{goto}(\textit{terminal2}); \textit{look_at_panel}; \textit{buy}(\textit{paper});$
if $\textit{gate} \geq 90$ then $\{\textit{goto}(\textit{gate}); \textit{buy}(\textit{coffee}); \textit{board_plane}\}$
else $\{\textit{buy}(\textit{coffee}); \textit{goto}(\textit{gate}); \textit{board_plane}\}$

- δ' is similar to what sGolog would return;
- there are many other solutions w.r.t. Σ_c

sGolog and Σ_c

Theorem: All solutions of sGolog are solutions of Σ_c

PLUS

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

Implementing Σ_c

Problem: how and where should we split?

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

How: a restrictive Σ_{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 Σ_{cb}

Σ_{cb} splits **only** when a *branch*(ϕ) action is encountered.

Only two modifications are required:

- Special action *branch*(ϕ) is treated as a regular primitive action;
- Modify last axiom for *run*(δ, s):

$$\begin{aligned} \textit{run}(\textit{if } \phi \textit{ then } \delta_1 \textit{ else } \delta_2) = & \textit{if } \phi(s) \textit{ then } \textit{do}(\textit{branch}(\phi), \textit{run}(\delta_1, s)) \\ & \textit{else } \textit{do}(\textit{branch}(\phi), \textit{run}(\delta_2, s)) \end{aligned}$$

Theorem: sGolog and Σ_{cb} compute the **same solutions** for Golog programs

Implementing Σ_{cb}

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

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

$$\text{Axioms} \models \text{Trans}(\Sigma_c p, s, p', s')$$

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

(2) If G finitely fails, then

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

Conclusions

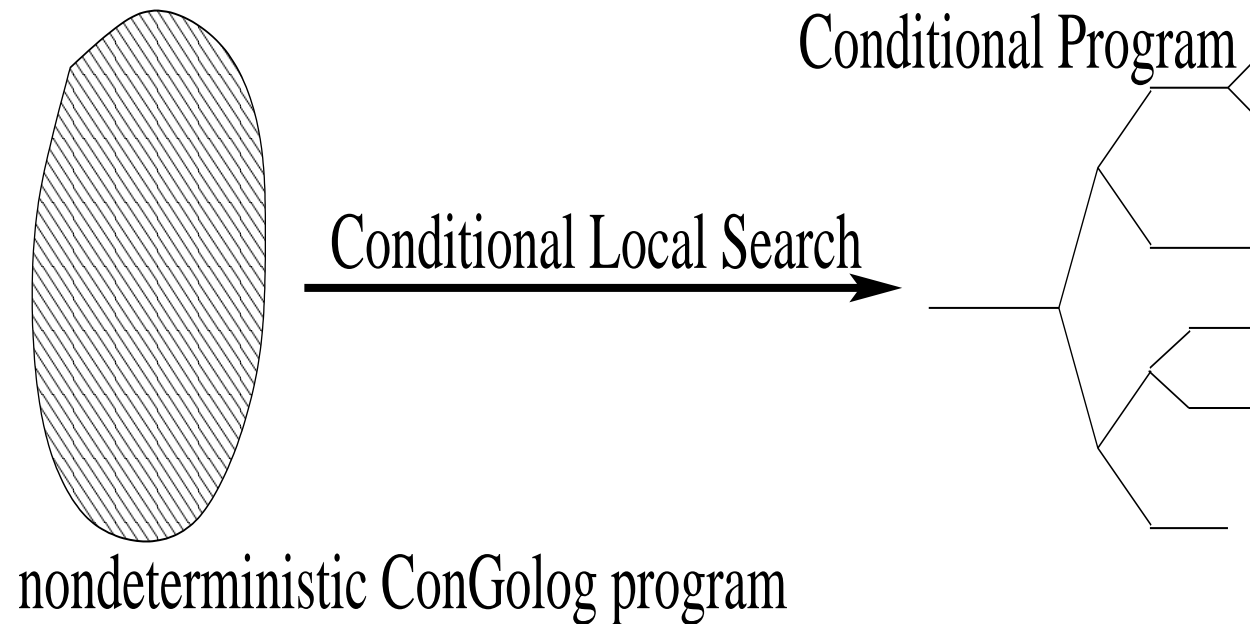
A new construct for ConGolog that ...

- 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 δ into a simple conditional program

Solutions for $\Sigma_c(\textit{catch_plane})$

$$\textit{Axioms} \models \textit{Trans}(\Sigma_c \textit{catch_plane}, S_0, \delta', S_0)$$

$$\begin{aligned} \delta' = & \textit{goto}(\textit{airport}); \textit{goto}(\textit{terminal2}); \textit{look_at_panel}; \textit{buy}(\textit{paper}); \\ & \textit{if } gate \geq 90 \textit{ then } \{ \textit{goto}(\textit{gate}); \textit{buy}(\textit{coffee}); \textit{board_plane} \} \\ & \textit{else } \{ \textit{buy}(\textit{coffee}); \textit{goto}(\textit{gate}); \textit{board_plane} \} \end{aligned}$$

$$\begin{aligned} \delta' = & \textit{goto}(\textit{airport}); \textit{goto}(\textit{terminal2}); \textit{look_at_panel}; \textit{buy}(\textit{paper}); \\ & \textit{if } gate \geq 90 \textit{ then } \{ \textit{goto}(\textit{gate}); \textit{buy}(\textit{coffee}); \textit{board_plane} \} \\ & \textit{else [if } (p \vee \neg p) \textit{ then} \\ & \quad \{ \textit{buy}(\textit{coffee}); \textit{goto}(\textit{gate}); \textit{board_plane} \} \\ & \quad \textit{else } \{ \textit{buy}(\textit{coffee}) \} \textit{]} \end{aligned}$$

Incremental Execution and Σ_C

$$\begin{array}{ccc}
 \text{Axioms} \cup \text{Sensed}[s_0] & \models & \text{Trans}(\delta_0, s_0, \delta_1, s_1) \\
 \text{Axioms} \cup \text{Sensed}[s_1] & \models & \text{Trans}(\delta_1, s_1, \delta_2, s_2) \\
 & \vdots & \\
 \Rightarrow \text{Axioms} \cup \text{Sensed}[s_k] & \models & \text{Trans}((\Sigma_C \delta_k); \delta', s_k, (\delta_{CPP}; \delta'), s_k) \\
 & \vdots & \\
 & \text{deterministic} & \\
 & \text{execution of } \delta_{CPP} & \\
 & \vdots & \\
 \text{Axioms} \cup \text{Sensed}[s_j] & \models & \text{Trans}(\delta', s_j, \delta'', s_{j+1}) \\
 & \vdots &
 \end{array}$$

Prolog Code for Σ_{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,[],C):- build_cpp(E,S,C). /* leaf of CPP */

```

Programs with Concurrency (ConGolog)

```

proc catch_plane
  (have_drink  $\wedge$  thirsty  $\rightarrow$  drink)  $\rangle\rangle$ 
    [(( $\pi a.a$ )*; at(airport)?;
      (goto(terminal1) | goto(terminal2));
      look_at_panel;          /* Sensing Action */
      (buy(magazine) | buy(paper));
      if gate  $\geq$  90 then { goto(gate); buy(coffee) } else
                          { buy(coffee); goto(gate) }

      board_plane;]
end_proc

```

$$\delta_{CPP} = \textit{goto}(\textit{airport}); \textit{goto}(\textit{terminal2}); \textit{look_at_panel}; \textit{buy}(\textit{paper});$$

$$\text{if } \textit{gate} \geq 90 \text{ then } \{ \textit{goto}(\textit{gate}); \textit{buy}(\textit{coffee}); \textit{drink}; \textit{board_plane} \}$$

$$\text{else } \{ \textit{buy}(\textit{coffee}); \textit{goto}(\textit{gate}); \textit{drink}; \textit{board_plane} \}$$

Epistemic Search: Σ_e

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

$$\begin{aligned} Trans(\Sigma_e(\delta), s, dp', s') &\equiv \\ &\exists dp. EFDP(dp, s) \wedge \\ &\exists s_f. Trans(dp, s, dp', s') \wedge Do(dp', s', s_f) \wedge Do(\delta, s, s_f) \end{aligned}$$

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

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