Golog semantics

Golog/ConGolog programs are syntactic objects.

How do we assign a formal semantics to them?

Let us first consider Golog only.

For simplicity we will not consider procedures, but see [DLL-AIJ00,LRLLS97].

Golog semantics (cont.)

We start by considering a single model of the SitCalc action theory. (That is we start by assuming complete information, just as in normal computer programs)

Any idea of what the semantics should talk about?

Evaluation semantics: intro

Idea: describe the overall result of the evaluation of the Golog program.

Given a Golog program δ and a situation s **compute the situation s' obtained by executing δ in s**.

More formally: Define the **relation**:

$$(\delta, s) \longrightarrow s'$$

where δ is a program, s is the situation in which the program is evaluated, and s' is the situation obtained by the evaluation.

Such a relation can be defined inductively in a standard way using the so called **evaluation (structural) rules**

Evaluation semantics: references

The general approach we follows is is the *structural operational semantics* approach[Plotkin81, Nielson&Nielson99].

This whole-computation semantics is often call: *evaluation semantics* or *natural semantics* or *computation semantic*.

Evaluation rules for Golog: deterministic constructs

$$Act: \quad \frac{(a,s) \longrightarrow do(a[s],s)}{true} \quad \text{if } Poss(a[s],s)$$

$$Test: \quad \frac{(\phi?,s) \longrightarrow s}{true} \quad \text{if } \phi[s]$$

$$Seq: \quad \frac{(\delta_1;\delta_2,\ s) \longrightarrow s'}{(\delta_1,s) \longrightarrow s'' \ \wedge\ (\delta_2,s') \longrightarrow s'}$$

$$if: \quad \frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2, s) \longrightarrow s'}{(\delta_1,s) \longrightarrow s'} \quad \text{if } \phi[s] \qquad \frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2, s) \longrightarrow s'}{(\delta_2,s) \longrightarrow s'} \quad \text{if } \neg\phi[s]$$

$$while: \quad \frac{(\textbf{while } \phi \textbf{ do } \delta, s) \longrightarrow s}{true} \quad \text{if } \neg\phi[s] \qquad \frac{(\textbf{while } \phi \textbf{ do } \delta, s) \longrightarrow s'}{(\delta,s) \longrightarrow s'' \ \wedge\ (\textbf{while } \phi \textbf{ do } \delta\ s'') \longrightarrow s'} \quad \text{if } \phi[s]$$

Evaluation rules: nondeterministic constructs

$$Nondetbranch: \quad \frac{(\delta_1 \mid \delta_2,\ s) \longrightarrow s'}{(\delta_1,s) \longrightarrow s'} \qquad \frac{(\delta_1 \mid \delta_2,\ s) \longrightarrow s'}{(\delta_2,s) \longrightarrow s'}$$

$$Nondetchoice: \quad \frac{(\pi\, x.\ \delta(x),s) \longrightarrow s'}{(\delta(t),s) \longrightarrow s'} \quad \textit{(for any t)}$$

$$Nondetiter: \quad \frac{(\delta^*,s) \longrightarrow s}{true} \qquad \frac{(\delta^*,s) \longrightarrow s'}{(\delta,s) \longrightarrow s'' \wedge (\delta^*,s'') \longrightarrow s'}$$

Structural rules

The structural rules have the following schema:

$$\frac{\text{CONSEQUENT}}{\text{ANTECEDENT}} \text{ if SIDE-CONDITION}$$

which is to be interpreted logically as:

$$\forall(\text{ANTECEDENT} \wedge \text{SIDE-CONDITION} \supset \text{CONSEQUENT})$$

where $\forall Q$ stands for the universal closure of all free variables occurring in Q, and, typically, ANTECEDENT, SIDE-CONDITION and CONSEQUENT share free variables.

Given a model of the SitCalc action theory, the structural rules define inductively a relation, namely: **the smallest relation satisfying the rules**.

Examples

Compute the following assuming actions are always possible:

- $(a; b, S_0) \longrightarrow s_f$
- $((a \mid b); c, S_0) \longrightarrow s_f$
- $((a \mid b); c; P?, S_0) \longrightarrow s_f$ where P true iff a is not performed yet.

Getting logical

Till now we have defined the relation $(\delta, s) \longrightarrow s'$ in a single model of the SitCalc action theory of interest.

But what about if the action theory has incomplete information and hence admits several models?

Idea: *Define a logical predicate* $Do(\delta, s, s')$ *starting from the definition of the relation* $(\delta, s) \longrightarrow s'$.

Definition of Do: intro

How: *do we define a logical predicate* $Do(\delta, s, s')$ *starting from the definition of the relation* $(\delta, s) \longrightarrow s'$?

- Rules correspond to logical conditions;

- The minimal predicate satisfying the rules is expressible in 2nd-order logic by using the formulas of the following form:

$\forall D.\{$

logical formulas corresponding to the rules

that use the **predicate variable** D in place of the relation

$\} \supset D(\delta, s, s').$

Definition of Do

$$
\begin{array}{l}
Do(\delta, s, s') \;\equiv \\
\quad \forall D.\{ \\
\qquad \forall[\, Poss(a[s], s) \;\supset\; D(a, s, do(a[s], s))\,] \;\wedge \\
\qquad \forall[\, \phi[s] \;\supset\; D(\phi?, s, s)\,] \;\wedge \\
\qquad \forall[\, D(\delta_1, s, s'') \wedge D(\delta_2, s'', s') \;\supset\; D(\delta_1; \delta_2,\, s, s')]\,] \;\wedge \\
\qquad \forall[\, \phi[s] \wedge D(\delta_1, s, s') \;\vee\; \neg\phi[s] \wedge D(\delta_2, s, s')] \;\supset\; D(\textbf{if}\ \phi\ \textbf{then}\ \delta_1 \textbf{else}\ \delta_2, s, s')\,] \;\wedge \\
\qquad \forall[\, \neg\phi[s] \wedge s' = s \;\vee\; \phi[s] \wedge D(\delta_2, s, s') \wedge D(\textbf{while}\ \phi\ \textbf{do}\ \delta, s, s') \;\supset\; D(\textbf{while}\ \phi\ \textbf{do}\ \delta, s, s')\,] \;\wedge \\
\qquad \forall[\, D(\delta_1, s, s') \;\vee\; D(\delta_2, s'', s') \;\supset\; D(\delta_1 \mid \delta_2,\, s, s')\,] \;\wedge \\
\qquad \forall[\, D(\delta(t), s, s') \;\supset\; D(\pi\, x.\, \delta(x), s, s')\,] \;\wedge \\
\qquad \forall[\, s' = s \;\vee\; D(\delta, s, s'') \wedge D(\delta^*, s'', s') \;\supset\; D(\delta^*, s, s')\,] \;\wedge \\
\quad \} \;\supset\; D(\delta, s, s').
\end{array}
$$

Examples

Assuming the action theory Γ does not logically implies $Poss(a, S_0)$, but all other actions are possible, find all s_f that constitute (certain) executions of the programs seen before, i.e., such that the following logical implication holds:

- $\Gamma \models Do(a; c,\ S_0,\ s_f)$
- $\Gamma \models Do((a \mid b); c,\ S_0,\ s_f)$
- $\Gamma \models Do((a \mid b); c; P?,\ S_0,\ s_f)$ where P holds iff a is not performed yet.

Original Definition of Do

In [LRLLS97], $Do(\delta, s, s')$ is defined by induction on the structure of the program instead of using structural rules as above.

The main advantage of this definition is that $Do(\delta, s, s')$ can be is simply viewed as an abbreviation for a formula of the SitCalc.

Programs do not even need to be formally introduced!!!

Original Definition of Do (cont.)

$Act:$	$Do(a, s, s') \stackrel{def}{=} Poss(a[s], s) \wedge s' = do(a[s], s)$
$Test:$	$Do(\phi?, s, s') \stackrel{def}{=} \phi[s] \wedge s = s'$
$Seq:$	$Do(\delta_1; \delta_2,\ s, s') \stackrel{def}{=} \exists s''.\ Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$
$Nondetbranch:$	$Do(\delta_1 \mid \delta_2,\ s, s') \stackrel{def}{=} Do(\delta_1, s, s') \vee Do(\delta_2, s, s')$
$Nondetchoice:$	$Do(\pi\, x.\ \delta(x), s, s') \stackrel{def}{=} \exists x.\ Do(\delta(x), s, s')$
$Nondetiter:$	It is not definable in 1st-order logic! ...

Original Definition of Do (cont. 2)

Nondeterministic iteration:

$$
\begin{aligned}
Do(\delta^*, s, s') \stackrel{def}{=} \; & \forall P.\{ \\
& \quad \forall [\; P(s,s) \;] \wedge \\
& \quad \forall [\; P(s,s'') \wedge Do(\delta, s'', s') \supset P(s,s') \;] \\
& \} \supset P(s,s').
\end{aligned}
$$

i.e., doing action δ zero or more times takes you from s to s' iff (s, s') is in every set (and thus, the smallest set) s.t.:

1. (s, s) is in the set for all situations s.

2. Whenever (s, s'') is in the set, and doing δ in situation s'' takes you to situation s', then (s, s'') is in the set.

Must use 2nd-order logic because transitive closure is not 1st-order definable.

And concurrency?

Unfortunately evaluation semantics does not extend to construct for concurrency.

We need a finer form of semantics, namely **Transition Semantics**, where we specify what executing a **single step** of the program amounts to.