Semantics of GOLOG Constructs

GOLOG Semantics

The semantics of a program body δ is given by a formula

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

meaning that

- in situation s, program δ can be successfully executed;
- the execution may terminate in situation s'.

Intuition (I)

δ_1: *Go*(*Up*); $S_{final} = Do(Drop(B_1), Do(Go(Up), S_0)) \wedge$

Drop(B_1) $Poss(Go(Up), S_0) \wedge Poss(Drop(B_1), Do(Go(Up), S_0))$

δ_2: *Go*(*Up*) | $[S_{final} = Do(Go(Up), S_0) \wedge Poss(Go(Up), S_0)] \vee$

Go(*Down*) $[S_{final} = Do(Go(Down), S_0) \wedge Poss(Go(Down), S_0)]$

δ_3: *At*(6) ?; $S_{final} = Do(Drop(B_3), S_0) \wedge$

Drop(B_3) $\neg Holds(At(6), S_0) \wedge Poss(Drop(B_3), S_0)$

Executability and Result of GOLOG Programs

The executability and result of a GOLOG program usually depends on the circumstances of the agent.

For example, δ_1 succeeds if the robot carries in B_1 a package for the next room but fails if, e.g., $Holds(Empty(B_1), S_0)$.

Strategy δ_2 usually allows the robot to go either left or right, but if it starts in room 1 or 6 then there is only one way to execute the program.

Intuition (II)

δ_4: **if** *At*(1) **then**
Go(*Up*)
else
Go(*Down*)
endIf

$[Holds(At(1), S_0) \wedge Poss(Go(Up), S_0) \wedge S_{final} = Do(Go(Up), S_0)] \vee [\neg Holds(At(1), S_0) \wedge Poss(Go(Down), S_0) \wedge S_{final} = Do(Go(Down), S_0)]$

δ_5: $\pi p.\pi b.Pick(p,b)$ $(\exists b)(\exists p)(Poss(Pick(p,b), S_0) \wedge S_{final} = Do(Pick(p,b), S_0))$

Semantics of GOLOG Constructs (1)

$$DO(\textbf{nil}, s, s') \stackrel{\text{def}}{=} s' = s$$

$$DO(a, s, s') \stackrel{\text{def}}{=} Poss(a, s, s') \wedge s' = Do(a, s)$$

$$DO(\phi?, s, s') \stackrel{\text{def}}{=} \phi[s] \wedge s' = s$$

$$DO(\delta_1; \delta_2, s, s') \stackrel{\text{def}}{=} (\exists s'')\,(DO(\delta_1, s, s'') \wedge DO(\delta_2, s'', s'))$$

$$DO(\delta_1 | \delta_2, s, s') \stackrel{\text{def}}{=} DO(\delta_1, s, s') \vee DO(\delta_2, s, s')$$

$$DO(\pi x.\delta(x), s, s') \stackrel{\text{def}}{=} (\exists x)\, DO(\delta(x), s, s')$$

$\phi[s]$ is formula ϕ with all occurrences of f replaced by $Holds(f,s)$

Semantics of GOLOG Constructs (2)

if ϕ **then** δ_1 **else** δ_2 **endIf** $\stackrel{\text{def}}{=}$ $(\phi?\ ;\ \delta_1)\ |\ (\neg\phi?\ ;\ \delta_2)$

while ϕ **do** δ **endWhile** $\stackrel{\text{def}}{=}$ $(\phi?\ ;\ \delta)^*\ ;\ \neg\phi?$

Intuition (III)

$$Door(x,y) \equiv\ x = Kitchen \wedge y = Library \vee$$
$$x = Library \wedge y = SecretChamber \vee$$
$$x = SecretChamber \wedge y = Kitchen \vee$$
$$x = Alley \wedge y = Garden$$

$$Poss(Go(x), s,t) \equiv (\exists x')(Holds(At(x'), s) \wedge Door(x',x)) \wedge t = Do(Go(x), s)$$

$$Poss(Go(x), s,t) \supset (\exists x')(Holds(At(x'), s) \wedge$$
$$(\forall f)[f = At(x) \vee (Holds(f,s) \wedge f \neq At(x')) \supset Holds(f,t)] \wedge$$
$$(\forall f)[f = At(x') \vee (\neg Holds(f,s) \wedge f \neq At(x)) \supset \neg Holds(f,t)]$$

$$\delta_{wizard}: \quad [\pi x.Go(x)]^*$$

What are the reachable situations if the wizard executes its program starting in the kitchen?

Transitive Closure

The simple attempt to define

$$Do(\delta^*, s, s') \text{ as } \quad s = s' \vee$$
$$(\exists s'')(Do(\delta, s, s'') \wedge Do(\delta^*, s'', s'))$$

won't do because it does not allow to derive that

$$\neg(\exists s)(Do(\delta_{wizard}, S_0, s) \wedge Holds(At(Garden), s))$$

The reason is that transitive closure is not first-order definable.

Second-Order Logic

Second-order logic allows to quantify over predicate variables.

A predicate variable may be interpreted by any relation over the domain elements (of the right arity).

Example: $(\forall P)(\exists x)P(x)$ (A formula with no model!)

Substitutions for predicate variables use λ-expressions $\lambda x_1, ..., x_n . \Phi$

Example: $(\exists x)P(x)\ \{P/\lambda s.\ s = S_0 \wedge s \neq S_0\}$ yields $x = S_0 \wedge x \neq S_0$

Semantics of GOLOG Constructs (3)

$$DO(\delta^*, s, s') \stackrel{\text{def}}{=} (\forall P)\,([(\forall s_1)\,P(s_1, s_1) \wedge (\forall s_1, s_2, s_3)\,(P(s_1, s_2) \wedge DO(\delta, s_2, s_3) \supset P(s_1, s_3))] \supset P(s, s'))$$

The right-hand-side defines (s, s') to be in every relation such that

- $(s_1, s_1) \in P$
- whenever $(s_1, s_2) \in P$ and $DO(\delta, s_2, s_3)$ then $(s_1, s_3) \in P$

Example

$Do(\delta_{wizard}, S_0, s)$ is

(1) $(\forall s_1)\, P(s_1, s_1) \wedge$

(2) $(\forall s_1, s_2, s_3)\, (P(s_1, s_2) \wedge$

$(\exists x)(Poss(Go(x), s_2) \wedge s_3 = Do(Go(x), s_2) \supset P(s_2, s_3))$

$\supset P(S_0, s)$

Substitute $P\backslash\lambda s,s'.\ \neg Holds(At(Alley),s) \wedge \neg Holds(At(Garden),s)$

$\supset \neg Holds(At(Alley),s') \wedge \neg Holds(At(Garden),s')$

Then the domain axioms on Slide 8 eintail (1) and (2). Hence it follows

$\neg Holds(At(Alley),S_0) \wedge \neg Holds(At(Garden),S_0)$

$\supset \neg Holds(At(Alley),s) \wedge \neg Holds(At(Garden),s)$

A GOLOG Interpreter in Prolog

A Simple Action Description Language: Situation Calculus

Precondition axioms in the Situation Calculus are of the simple form

$$Poss(A(\vec{x}), s) \equiv \pi_A[s]$$

abbreviating the general form

$$Poss(A(\vec{x}), s, t) \equiv \pi_A[s] \wedge t = Do(A(\vec{x}), s)$$

Situation Calculus: Successor State Axioms

For all functions *F* into sort FLUENT, there is a successor state axiom

$$Holds(F(\vec{y}), Do(a, s)) \equiv \gamma_F^+[a, s, \vec{y}] \vee (Holds(F(\vec{y}), s) \wedge \neg\gamma_F^-[a, s, \vec{y}])$$

where

- γ_F^+ state formula, describing the conditions under which $F(\vec{y})$ is positive effect
- γ_F^- state formula, describing the conditions under which $F(\vec{y})$ is negative effect

Successor State Axioms as General Effect Axioms

A set of successor state axioms can be mapped onto this general effect axiom for an action A($\vec{x}$):

$$Poss(A(\vec{x}), s, t) \supset (\forall f)[\mathrm{V}_F(\exists \vec{y})(f = F(\vec{y}) \wedge \Gamma_A[\vec{x}, s]) \supset Holds(f, t)]$$

$$\wedge$$

$$(\forall f)[\mathrm{V}_F(\exists \vec{y})(f = F(\vec{y}) \wedge \neg\Gamma_A[\vec{x}, s]) \supset \neg Holds(f, t)]$$

where

- V_F ranges over all fluent functions
- $\Gamma_A[\vec{x}, s]$ stands for

$$\gamma_F^+[a/A(\vec{x}), s, \vec{y}] \vee (Holds(F(\vec{y}), s) \wedge \neg\, \gamma_F^-[a/A(\vec{x}), s, \vec{y}])$$

GOLOG in Prolog: Preconditions

```
poss(go(up),S)    :- holds(at(R),S), R<6.
poss(go(down),S)  :- holds(at(R),S), R>1.
poss(pick(P,B),S):- holds(at(R),S), holds(request(P,R,R1),S),
                    holds(empty(B),S).
poss(drop(B),S)   :- holds(at(R),S), holds(carries(B,P,R),S).
```

GOLOG in Prolog: Successor State Axioms (1)

```
holds(at(R),do(A,S)) :- holds(at(R1),S), R is R+1, A=go(up).
holds(at(R),do(A,S)) :- holds(at(R1),S), R is R-1, A=go(down).
holds(at(R),do(A,S)) :- holds(at(R),S), not A=go(D).

holds(request(P,R1,R2),do(A,S)) :- holds(request(P,R1,R2),S),
                                   not A=pick(P,B).
```

GOLOG in Prolog: Successor State Axioms (2)

```
holds(empty(B),do(A,S)) :- A=drop(B).
holds(empty(B),do(A,S)) :- holds(empty(B),S), not A=pick(P,B).

holds(carries(B,P,R),do(A,S)) :- A=pick(P,B),
                                 holds(request(P,R1,R),S).
holds(carries(B,P,R),do(A,S)) :- holds(carries(B,P,R),S),
                                 not A=drop(B).
```

GOLOG in Prolog: Initial State

```
holds(at(1),s0).
holds(empty(b1),s0).
holds(empty(b2),s0).
holds(empty(b3),s0).
holds(request(p1,1,2),s0).
...
holds(request(p9,6,4),s0).
```

GOLOG in Prolog: The Regression Principle

```
?- holds(carries(b1,p1,2),do(go(up),do(pick(p1,b1),s0)))

↳ ?- holds(carries(b1,p1,2),do(pick(p1,b1),s0)),
     not go(up)=drop(b1)

↳ ?- pick(p1,b1)=pick(p1,b1),
     holds(request(p1,R1,2),s0),
     not go(up)=drop(b1)

↳ R1=1
```

A Generic GOLOG Interpreter (1)

```
do([],S,S).                                          % nil

do(A,S,do(A,S)) :- poss(A,S).

do([E|L],S,S1) :- do(E,S,S2), do(L,S2,S1).           % sequence

do(?(P),S,S) :- holds(P,S).                          % test

do(E1#E2,S,S1) :- do(E1,S,S1).                       % choice
do(E1#E2,S,S1) :- do(E2,S,S1).
```

A Generic GOLOG Interpreter (2)

```
do(pi(V,E),S,S1) :- sub(V,X,E,E1), do(E1,S,S1).
do(star(E),S,S1) :- do([]#[E,star(E)],S,S1).
do(if(P,E1,E2)S,S1) :- do([?(P),E1]#[?(neg(P)),E2],S,S1).
do(while(P,E),S,S1) :- do([star([?(P),E],?(neg(P))],S,S1).
do(P,S,S1) :- proc(P,E), do(E,S,S1).
```

GOLOG Program for the Mailbot (1)

```
proc(deliver, pi(b,drop(b))).

proc(collect, pi(b,pi(p,pick(p,b)))).

proc(continue, if(exists(b,not(empty(b))),
      pi(b,pi(p,pi(r,pi(r1,[?(at(r)),?(carries(b,p,r1)),
                  if(?r<r1,go(up),go(down))])))),
      pi(p,pi(r,pi(r1,pi(r2,[?(at(r)),?(request(p,r1,r2)),
                  if(?r<r1,go(up),go(down))])))
    )
  ).
```

Compare this to Slides IIa/8 and 9.

GOLOG Program for the Mailbot (2)

```
proc(control, while(or(exists(b,empty(b)),
     exists(p,exists(r1,exists(r2,request(p,r1,r2))))))
if(exists(b,exists(p,exists(r,and(carries(b,p,r),at(r))))),
     deliver,
     if(exists(b,exists(p,exists(r1,exists(r2,
               and(empty(b),
               and(request(p,r1,r2),at(r1))))))),
        collect,
        continue)
    )
  )
).
```

Compare this to Slides IIa/7.

Example Query

```
holds(at(1),s0).
holds(empty(b1),s0).
holds(request(p1,1,2),s0).
holds(request(p2,1,5),s0).

?- do(deliver,s0,S).

no

?- do(collect,s0,S).

S = do(pick(p1,b1)) More?

S = do(pick(p2,b1)) More?

no
```