Procedural Action Programs

- GOLOG ("Algol in Logic")
- Semantics
- A GOLOG Interpreter
- Concurrency and Interrupts
- Sensing

GOLOG Commands

Command	Meaning
nil	empty program
a	primitive action
$\phi?$	test
$\delta_1 ; \delta_2$	sequential composition
$\delta_1 \mid \delta_2$	nondeterministic choice of sub-program
$\pi x.\delta(x)$	nondeterministic choice of argument
δ^*	nondeterministic iteration
$p(\vec{t})$	procedure call
if ϕ **then** δ_1 **else** δ_2 **endIf**	conditional
while ϕ **do** δ **endWhile**	loop

GOLOG Programs

proc $p_1(\vec{v}_1)$

δ_1

endProc;

$\vdots$

proc $p_n(\vec{v}_n)$

δ_n

endProc;

δ

Example: Mail Delivery (Fluents and Actions)

Symbol	Type
At	ROOM $\mapsto$ FLUENT
Empty	BAG $\mapsto$ FLUENT
Carries	BAG $\times$ PACKAGE $\times$ ROOM $\mapsto$ FLUENT
Request	PACKAGE $\times$ ROOM $\times$ ROOM $\mapsto$ FLUENT

Symbol	Type
Go	DIRECTION $\mapsto$ ACTION
Pick	PACKAGE $\times$ BAG $\mapsto$ ACTION
Drop	BAG $\mapsto$ ACTION

A GOLOG Program for the Mailbot (1)

proc *Deliver*

$\pi b.\ Drop(b)$

endProc;

proc *Collect*

$\pi b.\ \pi p.\ Pick(p, b)$

endProc;

A GOLOG Program for the Mailbot (2)

proc *Continue*

if $(\exists b)\ \neg Empty(b)$ **then**

$\pi b.\ \pi p.\ \pi r.\ \pi r'.\ (At(r)?\ ;\ Carries(b, p, r')?\ ;$

if $r < r'$ **then** *Go*(*Up*) **else** *Go*(*Down*) **endif**)

else

$\pi p.\ \pi r.\ \pi r_1.\ \pi r_2.\ (At(r)?\ ;\ Request(p, r_1, r_2)?\ ;$

if $r < r_1$ **then** *Go*(*Up*) **else** *Go*(*Down*) **endif**)

endif

endProc;

A GOLOG Program for the Mailbot (3)

proc *Control*
  while $(\exists b)\ \neg Empty(b) \vee (\exists p, r_1, r_2)\ Request(p, r_1, r_2)$ **do**
    if $(\exists b, p, r)\ (Carries(b, p, r) \wedge At(r))$ **then**
      Deliver
    else
      if $(\exists b, p, r_1, r_2)\ (Empty(b) \wedge Request(p, r_1, r_2) \wedge At(r_1))$ **then**
        Collect
      else
        Continue
      endif
    endif
  endWhile
endProc;

The Complete Program

proc *Deliver*

…

proc *Collect*

…

proc *Continue*

…

proc *Control*

…

endProc;

Control

Online- vs. Offline-Execution

Online-execution

- Actions are performed immediately
- Agent commits to every nondeterministic choice
- Program may not be completed successfully even though a successful run exists

Offline-execution

- Program is run in simulation first
- Allows to find a successful run if it exists; compare different runs to find the shortest; etc.
- Practically impossible for large programs with many nondeterministic operations

GOLOG Semantics

A General Action Calculus

Definition 3.3.1

A domain signature is a finite, sorted logic language which includes the sorts FLUENT, ACTION, and TIME along with the predicates

$$<: \text{TIME} \times \text{TIME}$$

$$\mathit{Holds}: \text{FLUENT} \times \text{TIME}$$

$$\mathit{Poss}: \text{ACTION} \times \text{TIME} \times \text{TIME}$$

$s \leq t$ will be used as abbreviation for $s < t \vee s = t$.

State Formulas

Definition 3.3.2

A state formula in variables $\vec{t}$ is a first-order formula $\Phi[\vec{t}]$ in which the elements of $\vec{t}$ occur free and such that

- for each occurence of $\mathit{Holds}(f, t)$ in Φ we have $t \in \vec{t}$;
- predicate *Poss* does not occur in Φ.

Example: State Formula

$$Holds(f, s) \equiv f = At(1) \vee f = Empty(B_1) \vee f = Empty(B_2) \vee f = Empty(B_3) \vee$$
$$f = Request(P_1, 1, 2) \vee f = Request(P_2, 1, 5) \vee \ldots \vee$$
$$f = Request(P_8, 6, 1) \vee f = Request(P_9, 6, 4)$$

Example Time Structure: Linear Time

$$(\forall t)\, 0 \le t$$
$$(\forall s, t)\, s < t \vee t < s \vee s = t$$
$$(\forall r, s, t)\, r < s \wedge s < t \supset r < t$$

$$Poss(Go(x, y, v), s, t) \equiv Holds(At(x), s) \wedge t = s + \frac{1}{v} \bullet Distance(x, y)$$

Example Time Structure: Branching Time

$$(\forall s)\, S_0 \le s$$
$$(\forall a, a', s, s')(Do(a, s) = Do(a', s') \supset a = a' \wedge s = s')$$
$$(\forall a, s, s')(s < Do(a, s') \equiv s \le s')$$

Precondition Axioms

Definition 3.3.3

A precondition axiom is of the form

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

where $\pi_A[s]$ is a state formula in s with free variables among $s, t, \vec{x}$.

Example: Precondition Axioms

$$Poss(Go(d), s, t) \equiv t = Do(Go(d), s) \wedge (\exists r)(Holds(At(r), s) \wedge [d = Up \wedge r < 6 \vee d = Down \wedge r > 1])$$

$$Poss(Pick(p, b), s, t) \equiv t = Do(Pick(p, b), s) \wedge (\exists r, r')(Holds(At(r), s) \wedge Holds(Request(p, r, r'), s) \wedge Holds(Empty(b), s))$$

$$Poss(Drop(b), s, t) \equiv t = Do(Drop(b), s) \wedge (\exists p, r)(Holds(At(r), s) \wedge Holds(Carries(b, p, r), s))$$

Effect Axioms

Definition 3.3.3 (cont'd)

An effect axiom is of the form

$$Poss(A(\vec{x}), s, t) \supset \Upsilon_1[s, t] \vee \ldots \vee \Upsilon_k[s, t]$$

where $k \geq 1$ and each $\Upsilon_i[s, t]$ $(1 \leq i \leq k)$ is a formula of the form

$$(\exists \vec{y}_i)(\Phi_i[s] \wedge (\forall f)[\Gamma_i^+[s, t] \supset Holds(f, t)] \wedge (\forall f)[\Gamma_i^-[s, t] \supset \neg Holds(f, t)])$$

in which $\Phi_i[s]$ is a state formula in s with free variables among $s, \vec{x}, \vec{y}_i$, and both $\Gamma_i^+[s, t]$ and $\Gamma_i^-[s, t]$ are state formulas in s, t with free variables among $f, s, t, \vec{x}, \vec{y}_i$.

Example: Effect Axiom (1)

$$\begin{aligned} &Poss(Go(d), s, t) \supset \\ &\quad d = Up \wedge (\exists r)(Holds(At(r), s) \wedge \\ &\qquad (\forall f)[f = At(r + 1) \vee (Holds(f, s) \wedge f \neq At(r)) \supset Holds(f, t)] \\ &\qquad \wedge \\ &\qquad (\forall f)[f = At(r) \vee (\neg Holds(f, s) \wedge f \neq At(r + 1)) \supset \neg Holds(f, t)]) \\ &\quad \vee \\ &\quad d = Down \wedge (\exists r)(Holds(At(r), s) \wedge \\ &\qquad (\forall f)[f = At(r - 1) \vee (Holds(f, s) \wedge f \neq At(r)) \supset Holds(f, t)] \\ &\qquad \wedge \\ &\qquad (\forall f)[f = At(r) \vee (\neg Holds(f, s) \wedge f \neq At(r - 1)) \supset \neg Holds(f, t)]) \end{aligned}$$

Example: Effect Axiom (2)

$$\begin{aligned} &Poss(Pick(p, b), s, t) \supset \\ &\quad (\exists r, r')(Holds(Request(p, r, r'), s) \wedge \\ &\qquad (\forall f)[f = Carries(b, p, r') \vee (Holds(f, s) \wedge f \neq Empty(b) \wedge f \neq Request(p, r, r')) \\ &\qquad\qquad \supset Holds(f, t)] \\ &\qquad \wedge \\ &\qquad (\forall f)[f = Empty(b) \vee f = Request(p, r, r') \vee (\neg Holds(f, s) \wedge f \neq Carries(b, p, r')) \\ &\qquad\qquad \supset \neg Holds(f, t)]) \end{aligned}$$

Example: Effect Axiom (3)

$$Poss(Drop(b), s, t) \supset$$
$$(\exists p, r)(Holds(Carries(b, p, r), s) \wedge$$
$$(\forall f)[f = Empty(b) \vee (Holds(f, s) \wedge f \neq Carries(b, p, r)) \supset Holds(f, t)]$$
$$\wedge$$
$$(\forall f)[f = Carries(b, p, r) \vee (\neg Holds(f, s) \wedge f \neq Empty(b)) \supset \neg Holds(f, t)])$$

Signatures with Functional Fluents

It is often more natural and economic to describe the state space by functional fluents instead of relational ones.

Definition 3.3.4

A functional domain signature is a finite, sorted logic language which includes the sorts FLUENT, ACTION, TIME, and VALUE along with the predicates

$<$: TIME $\times$ TIME

Poss: ACTION $\times$ TIME $\times$ TIME

and the function

Val: FLUENT $\times$ TIME $\mapsto$ VALUE

Functional Domain Axiomatizations (1)

Definition 3.3.4 (cont'd)

A state formula in variables $\vec{t}$ is a first order formula $\Phi[\vec{t}]$ in which the elements of $\vec{t}$ occur free and such that

- for each occurence of $Val(f, t)$ in Φ we have $t \in \vec{t}$;
- predicate *Poss* does not occur in Φ.

A precondition axiom is of the form

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

where $\pi_A[s]$ is a state formula in s with free variables among s, t, $\vec{x}$.

Functional Domain Axiomatizations (2)

Definition 3.3.4 (cont'd)

An effect axiom is of the form

$$Poss(A(\vec{x}), s, t) \supset \Upsilon_1[s, t] \vee \ldots \vee \Upsilon_k[s, t]$$

where $k \geq 1$ and each $\Upsilon_i[s, t]$ $(1 \leq i \leq k)$ is a formula of the form

$$(\exists \vec{y}_i)(\Phi_i[s] \wedge (\forall f, v)[\Gamma_i[s, t] \supset Val(f, t) = v])$$

in which $\Phi_i[s]$ is a state formula in s with free variables among, s, $\vec{x}$, $\vec{y}_i$, and $\Gamma_i[s, t]$ is a state formula in s, t with free variables among f, s, t, $\vec{x}$, $\vec{y}_i$, v.

Example: Fluents and Actions for Checkers

Symbol	Type	Range
Cell	FILE × ROW ↦ FLUENT	{*Blank*, *White*, *WhiteKing*, *Black*, *BlackKing*}
Control	↦ FLUENT	{*White*, *Black*}

Symbol	Type
Move	FILE × ROW × FILE × ROW ↦ ACTION

$Val(Control, 0) = Black \wedge$

$Val(Cell(a,1), 0) = White \wedge \ldots \wedge Val(Cell(h,8),0) = Black$

Example: Precondition Axiom for Checkers

$Poss(Move(x_1, y_1, x_2, y_2), s, t) \equiv$

$Occurs(Move(x_1, y_1, x_2, y_2), s) \wedge t = s + 1 \wedge$

$[LegalWhiteMove \vee LegalBlackMove \vee LegalKingMove \vee LegalJump]$

where

$LegalWhiteMove :<=> Val(Control, s) = White \wedge Val(Cell(x_1, y_1), s) = White \wedge$

$Val(Cell(x_2, y_2), s) = Blank \wedge y_1 < 8 \wedge y_2 = y_1 + 1 \wedge$

$NeighborFiles(x_1, x_2)$

etc.

Example: Effect Axioms for Checkers

$Poss(Move(x_1, y_1, x_2 , y_2), s, t) \supset LegalWhiteMove \wedge WhiteMoveUpdate \vee$

$LegalBlackMove \wedge BlackMoveUpdate \vee$

$LegalKingMove \wedge KingMoveUpdate \vee$

$LegalJump \wedge JumpUpdate$

where

$WhiteMoveUpdate :<=> (\forall f, v)[f = Cell(x_1, y_1) \wedge v = Blank \vee$

$f = Cell(x_2, y_2) \wedge (y_2 < 8 \wedge v = White \vee y_2 = 8 \wedge v = WhiteKing) \vee$

$f = Control \wedge v = Black \vee$

$f \neq Cell(x_1, y_1) \wedge f \neq Cell(x_2, y_2) \wedge f \neq Control \wedge v = Val(f, s) \supset Val(f, t) = v)$

etc.

GOLOG Semantics

The semantics 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'.

Semantics of GOLOG Constructs (1)

$DO(\textbf{nil}, s, s') :<=> s' = s$

$DO(a, s, s') :<=> Poss(a, s, s') \wedge s' = Do(a, s)$

$DO(\phi?, s, s') :<=> \phi[s] \wedge s' = s$

$DO(\delta_1; \delta_2, s, s') :<=> (\exists s'')\,(DO(\delta_1, s, s'') \wedge DO(\delta_2, s'', s'))$

$DO(\delta_1 | \delta_2, s, s') :<=> DO(\delta_1, s, s') \vee DO(\delta_2, s, s')$

$DO(\pi x.\delta(x), s, s') :<=> (\exists x)\, DO(\delta(x), s, s')$

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

Semantics of GOLOG Constructs (2)

$DO(\delta^*, s, s') :<=> (\forall P)\,([(\forall s_1)\, P(s_1, s_1) \wedge (\forall s_1, s_2, s_3)\,(P(s_1, s_2) \wedge DO(\delta, s_2, s_3) \supset P(s_1, s_3))] \supset P(s, s'))$

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 $(s_1, s_3) \in P$

Semantics of GOLOG Constructs (3)

if ϕ **then** δ_1 **else** δ_2 **endIf** $:<=> (\phi?; \delta_1) | (\neg\phi?; \delta_2)$

while ϕ **do** δ **endWhile** $:<=> (\phi?; \delta)^*; \neg\phi?$

Example

$DO(Collect, s, s') :<=> Do(\pi b.\pi p.Pick(p, b), s, s')$

$:<=> (\exists b)(\exists p)\,(Poss(Pick(p, b), s, s') \wedge s' = Do(Pick(p, b), s))$

A GOLOG Implementation

Situation Calculus

Precondition axioms in 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 $F(\vec{y})$ positive effect
- γ_F^- state formula, describing the conditions under which $F(\vec{y})$ 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 $A(\vec{x})$:

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

where

- $\forall_F$ 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=R1+1, A=go(up).
holds(at(R),do(A,S)) :- holds(at(R1),S), R=R1-1, 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 Situation

```
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).
do(A,S,do(A,S)) :- poss(A,S).
do([E|L],S,S1) :- do(E,S,S2), do(L,S2,S1).
do(?(P),S,S) :- holds(P,S).
do(E1#E2,S,S1) :- do(E1,S,S1).
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).
```


Example Encoding (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))])))
     )
  ).
```

Example Encoding (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)
     )
   )
).
```


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
```


Concurrency and Interrupts

ConGOLOG: Concurrency and Interrupts

Command	Meaning
$\delta_1 \parallel \delta_2$	concurrent execution
$\delta_1 \rangle\rangle \delta_2$	concurrent execution with priority
$\delta^{\parallel}$	concurrent iteration
$\langle \phi \rightarrow \delta \rangle$	interrupt

Example: Exogenous Actions

Symbol	Type
AddRequest	PACKAGE × ROOM × ROOM↦ ACTION
CancelRequest	PACKAGE ↦ ACTION
Idle	↦ ACTION

proc *Interaction*

$\pi p.\pi r_1.\pi r_2.$ *AddRequest*$(p, r_1, r_2) \mid \pi p.$ *CancelRequest*(p)

endProc;

Modified GOLOG Program for Mail Delivery

proc *Deliver* **... endProc**;

...

proc *Interaction* **... endProc**;

*Interaction** || (**while** $\neg(\exists p, r_1, r_2)$ *Request*(p, r_1, r_2) **do** *Idle* **endWhile**; *Control*)

Precondition Axioms

$$Poss(AddRequest(p, r_1, r_2), s, t) \equiv t = Do(AddRequest(p, r_1, r_2), s) \wedge \neg(\exists r'_1, r'_2)\, Holds(Request(p, r'_1, r'_2), s) \wedge \neg(\exists b, r')\, Holds(Carries(b, p, r'), s) \wedge r_1 \neq r_2$$

$$Poss(CancelRequest(p), s, t) \equiv t = Do(CancelRequest(p), s) \wedge (\exists r_1, r_2)\, Holds(Request(p, r_1, r_2), s)$$

$$Poss(Idle, s, t) \equiv t = Do(Idle, s)$$

Effect Axioms

$$Poss(AddRequest(p, r, r_2), s, t) \supset (\forall f)[f = Request(b, r_1, r_2) \vee Holds(f, s) \supset Holds(f, t)] \wedge (\forall f)[f \neq Request(p, r_1, r_2) \wedge \neg Holds(f, s) \supset \neg Holds(f, t)]$$

$$Poss(CancelRequest(p), s, t) \supset (\exists r_1, r_2)(Holds(Request(p, r_1, r_2), s) \wedge (\forall f)[Holds(f, s) \wedge f \neq Request(p, r_1, r_2) \supset Holds(f, t)] \wedge (\forall f)[f = Request(p, r_1, r_2) \vee \neg Holds(f, s) \supset \neg Holds(f, t)])$$

$$Poss(Idle\ s, t) \supset (\forall f)[Holds(f, s) \supset Holds(f, t)] \wedge (\forall f)[\neg Holds(f, s) \supset \neg Holds(f, t)]$$

Transition Semantics

The semantics for ConGOLOG is given by formulas

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

$Final(\delta, s)$

meaning that

- executing one step of δ in s may result in s', and δ' is what remains of δ after this step;
- δ can be considered completed in s.

Transition Semantics for GOLOG Constructs

$$Trans(\mathbf{nil}, s, \delta', s') \equiv False$$

$$Trans(a, s, \delta', s') \equiv Poss(a, s, s') \wedge \delta' = \mathbf{nil} \wedge s' = Do(a, s)$$

$$Trans(\phi?, s, \delta', s') \equiv \phi[s] \wedge \delta' = \mathbf{nil} \wedge s' = s$$

$$Trans(\delta_1;\delta_2, s, \delta', s') \equiv (\exists \delta_1')\,(Trans(\delta_1, s, \delta_1', s') \wedge \delta' = (\delta_1';\delta_2)) \vee Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta', s')$$

$$Trans(\delta_1|\delta_2, s, \delta', s') \equiv Trans(\delta_1, s, \delta', s') \vee Trans(\delta_2, s, \delta', s')$$

$$Trans(\pi x.\delta(x), s, \delta', s') \equiv (\exists x)\, Trans(\delta(x), s, \delta', s')$$

$$Trans(\delta^*, s, \delta', s') \equiv (\exists \delta'')\,(Trans(\delta, s, \delta'', s') \wedge \delta' = (\delta'';\delta^*))$$

Transition Semantics for GOLOG Macros

$$Trans(\mathbf{if}\ \phi\ \mathbf{then}\ \delta_1\ \mathbf{else}\ \delta_2\ \mathbf{endIf}, s, \delta', s') \equiv \phi[s] \wedge Trans(\delta_1, s, \delta', s') \vee \neg\phi[s] \wedge Trans(\delta_2, s, \delta', s')$$

$$Trans(\mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{endWhile}, s, \delta', s') \equiv (\exists \delta'')\,(\phi[s] \wedge Trans(\delta, s, \delta'', s') \wedge \delta' = (\delta''; \mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{endWhile}))$$

Transition Semantics for ConGOLOG Constructs

$$Trans(\delta_1||\delta_2, s, \delta', s') \equiv (\exists \delta)\,(Trans(\delta_1, s, \delta, s') \wedge \delta' = (\delta||\delta_2)) \vee (\exists \delta)\,(Trans(\delta_2, s, \delta, s') \wedge \delta' = (\delta_1||\delta))$$

$$Trans(\delta_1\rangle\rangle\delta_2, s, \delta', s') \equiv (\exists \delta)\,(Trans(\delta_1, s, \delta, s') \wedge \delta' = (\delta\rangle\rangle\delta_2)) \vee (\exists \delta)\,(Trans(\delta_2, s, \delta, s') \wedge \delta' = (\delta_1\rangle\rangle\delta)) \wedge \neg(\exists \delta'', s'')\, Trans(\delta_1, s, \delta'', s'')$$

$$Trans(\delta^{||}, s, \delta', s') \equiv (\exists \delta'')\,(Trans(\delta, s, \delta'', s') \wedge \delta' = (\delta''||\delta^{||}))$$

Interrupts

$\langle \phi \rightarrow \delta \rangle$:<=> **while** *True* **do**
if ϕ **then** δ **else** *False*? **endIf**
endWhile

To terminate the interrupt after one execution, use an arbitrary fluent that is true initially and set to false by δ.

Transitional Semantics: Final Configurations (1)

$$Final(\textbf{nil}, s) \equiv True$$
$$Final(a, s) \equiv False$$
$$Final(\phi?, s) \equiv False$$
$$Final(\delta_1;\delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$$
$$Final(\delta_1|\delta_2, s) \equiv Final(\delta_1, s) \vee Final(\delta_2, s)$$
$$Final(\pi x.\delta(x), s) \equiv (\exists x)\, Final(\delta(x), s)$$
$$Final(\delta^*, s) \equiv True$$

Transitional Semantics: Final Configurations (2)

$$Final(\delta_1||\delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$$
$$Final(\delta_1\rangle\rangle\delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$$
$$Final(\delta^{||}, s) \equiv True$$
$$Final(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf}, s) \equiv \phi[s] \wedge Final(\delta_1, s) \vee \neg\phi[s] \wedge Final(\delta_2, s)$$
$$Final(\textbf{while } \phi \textbf{ do } \delta \textbf{ endWhile}, s) \equiv \neg\phi[s] \vee Final(\delta, s)$$

Overall ConGOLOG Semantics

$$DO(\delta, s, s') :<=> (\exists \delta')(Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s'))$$

where

$$Trans^*(\delta, s, \delta', s') :<=> (\forall P)([(\forall \delta_1, s_1)P(\delta_1, s_1, \delta_1, s_1) \wedge (\forall \delta_1, \ldots, s_3)(P(\delta_1, s_1, \delta_2, s_2) \wedge Trans(\delta_2, s_2, \delta_3, s_3) \supset P(\delta_1, s_1, \delta_3, s_3))] \supset P(\delta, s, \delta', s'))$$

Example

Suppose $Holds(f, S_0) \equiv f = At(1) \vee f = Empty(B_1) \vee f = Empty(B_2) \vee f = Empty(B_3)$

Then $DO(\delta_{mail}, S_0, S)$ is true for

$S = Do(Drop(B_1), Do(Go(Up), Do(Pick(P_2, B_1), Do(Go(Down),$
$Do(CancelRequest(P_1), Do(AddRequest(P_2, 1, 2),$
$Do(Go(Up), Do(AddRequest(P_1, 2, 1), Do(Idle, S_0)))))))))$

GOLOG Programs with Sensing

Example: Robot in a Maze

proc *TakeNextStep*
    TurnRight;
    SenseBlocked;
    while *Blocked* **do**
        TurnLeft; *SenseBlocked*
    endWhile;
    GoForward
endProc;

while $\neg At(Exit)$ **do** *TakeNextStep* **endWhile**

Offline vs. Online Execution in the Presence of Sensing

Without sufficient knowledge, GOLOG programs with sensing cannot be executed offline.

Without sufficient knowledge of fluent *Blocked*, there is no provably successful run of δ_{maze}.

Online execution provides sufficient information about initially unknown fluents when needed. But the advantages of offline execution (to enable search) are lost.

The Search Operator

Only subprograms in the scope of the search operator ∑ are executed offline.

Example: ∑{(*PlanA* | *PlanB*); *SubGoal*?}; (*ActC* | *ActD*); *Test*?

- Action *PlanA* or action *PlanB* is chosen, depending on which achieves "goal" *SubGoal*.
- Action *ActC* or action *ActD* is chosen, and the agent commits to this choice prior to testing *Test*.

Axiomatizing Sensing Actions

The special predicate *SF*(*a*, *s*) is true if what is sensed by action *a* holds in situation *s*.

$$SF(SenseBlocked, s) \equiv Holds(Blocked, s)$$
$$SF(TurnLeft, s) \equiv True$$
$$SF(TurnRight, s) \equiv True$$
$$SF(GoForward, s) \equiv True$$

Histories

A history *h* is a sequence of (action, sensing result)-pairs

$$(a_1, v_1), (a_2, v_2), \ldots, (a_n, v_n) \qquad (n \geq 0)$$

- $End[h, s] :<=> Do(a_n, \ldots, Do(a_2, Do(a_1, s)) \ldots)$
- $Sensed[h, s] :<=> \{\neg\}\, SF(a_1, Do(a_1, s)) \wedge \ldots \wedge \{\neg\} SF(a_n, End[h, s])$

  where "¬" is placed wrt. a_i iff $v_i = false$

Example

The history

$$h = (TurnRight, true), (SenseBlocked, false), (GoForward, true)$$

determines this formula as $Sensed[h, S_0]$:

$$True \wedge$$
$$\neg Blocked(Do(SenseBlocked, Do(TurnRight, S_0))) \wedge$$
$$True \wedge \ldots$$

Transitional Semantics in the Presence of Sensing

Given a history h and domain axiomatization Ax

- A transition is possible if

  $Ax \wedge Sensed[h, s] \models Trans(\delta, End[h, s], \delta', s')$

- A program may terminate if

  $Ax \wedge Sensed[h, s] \models Final(\delta, End[h, s])$

Semantics of Search Operator

$$Trans(\sum\delta, s, \delta', s') \equiv (\exists\delta'')(Trans(\delta, s, \delta'', s') \wedge \delta' = \sum\delta'' \wedge (\exists\delta_e, s_e)(Trans^*(\delta'', s', \delta_e, s_e) \wedge Final(\delta_e, s_e)))$$

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