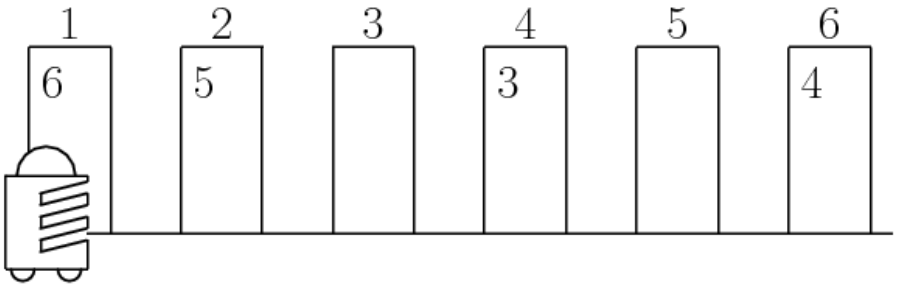
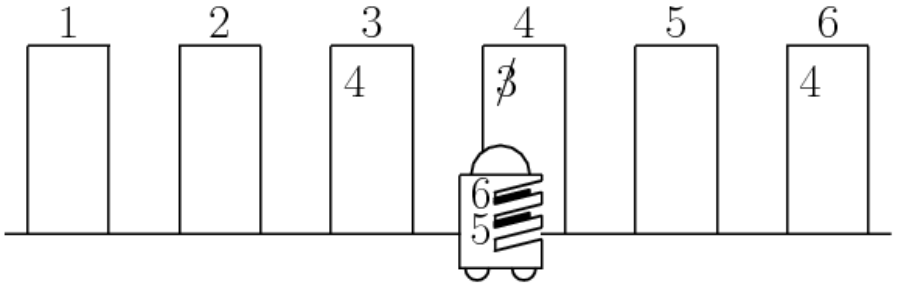
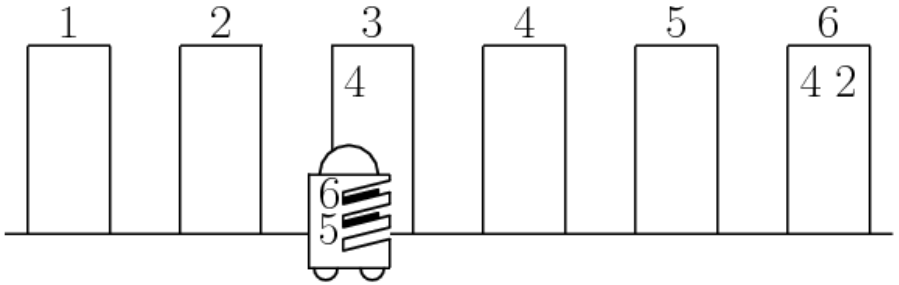
Extensions: Concurrency, Interrupts, Sensing

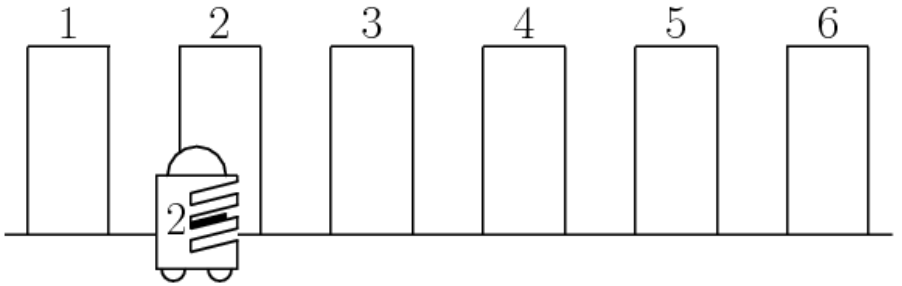
- Exogenous actions
- Concurrency and interrupts in GOLOG
- A transitional semantics for GOLOG
- Sensing actions

Mailbot in a Constantly Changing World









Example: Exogenous Actions

Symbol	Type
AddRequest	PACKAGE $\times$ ROOM $\times$ ROOM $\mapsto$ ACTION
CancelRequest	PACKAGE $\mapsto$ ACTION
Idle	$\mapsto$ ACTION

proc *Interaction*

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

endProc;

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

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

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

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

Concurrency and Interrupts in GOLOG

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

Modified GOLOG Program for Mail Delivery

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

...

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

*Interaction** || (**while** ¬(∃p, r_1, r_2) *Request*(p, r_1, r_2) **do** *Idle* **endWhile**; *Control*)

A Transitional Semantics for GOLOG

The transitional semantics for GOLOG is given by formulas

Trans(δ, s, δ', s')

Final(δ, 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.

Transitional Semantics for GOLOG Constructs

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

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

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

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

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

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

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

Transitional Semantics for GOLOG Macros

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

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

Transitional Semantics for Concurrency

$$\begin{aligned} \mathit{Trans}(\delta_1 \| \delta_2, s, \delta', s') \equiv\ & (\exists \delta)\ (\mathit{Trans}(\delta_1, s, \delta, s') \wedge \delta' = (\delta \| \delta_2)) \\ & \vee \\ & (\exists \delta)\ (\mathit{Trans}(\delta_2, s, \delta, s') \wedge \delta' = (\delta_1 \| \delta)) \end{aligned}$$

$$\begin{aligned} \mathit{Trans}(\delta_1 \rangle\rangle \delta_2, s, \delta', s') \equiv\ & (\exists \delta)\ (\mathit{Trans}(\delta_1, s, \delta, s') \wedge \delta' = (\delta \rangle\rangle \delta_2)) \\ & \vee \\ & (\exists \delta)\ (\mathit{Trans}(\delta_2, s, \delta, s') \wedge \delta' = (\delta_1 \rangle\rangle \delta)) \\ & \quad \wedge \neg(\exists \delta'', s'')\ \mathit{Trans}(\delta_1, s, \delta'', s'') \end{aligned}$$

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

Transitional Semantics for Interrupts

$\langle \phi \rightarrow \delta \rangle \stackrel{\text{def}}{=}$ **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)

$$\mathit{Final}(\mathbf{nil}, s) \equiv \mathit{True}$$

$$\mathit{Final}(a, s) \equiv \mathit{False}$$

$$\mathit{Final}(\phi?, s) \equiv \mathit{False}$$

$$\mathit{Final}(\delta_1;\delta_2, s) \equiv \mathit{Final}(\delta_1, s) \wedge \mathit{Final}(\delta_2, s)$$

$$\mathit{Final}(\delta_1|\delta_2, s) \equiv \mathit{Final}(\delta_1, s) \vee \mathit{Final}(\delta_2, s)$$

$$\mathit{Final}(\pi x.\delta(x), s) \equiv (\exists x)\, \mathit{Final}(\delta(x), s)$$

$$\mathit{Final}(\delta^*, s) \equiv \mathit{True}$$

Transitional Semantics: Final Configurations (2)

$$Final(\delta_1 || \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$$

$$Final(\delta_1 \rangle\rangle \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$$

$$Final(\delta^{||}, s) \equiv True$$

$$Final(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf}, s) \equiv \phi[s] \wedge Final(\delta_1, s) \vee \neg\phi[s] \wedge Final(\delta_2, s)$$

$$Final(\textbf{while } \phi \textbf{ do } \delta \textbf{ endWhile}, s) \equiv \neg\phi[s] \vee Final(\delta, s)$$

The Overall Transitional Semantics

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

where

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

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, e.g.,

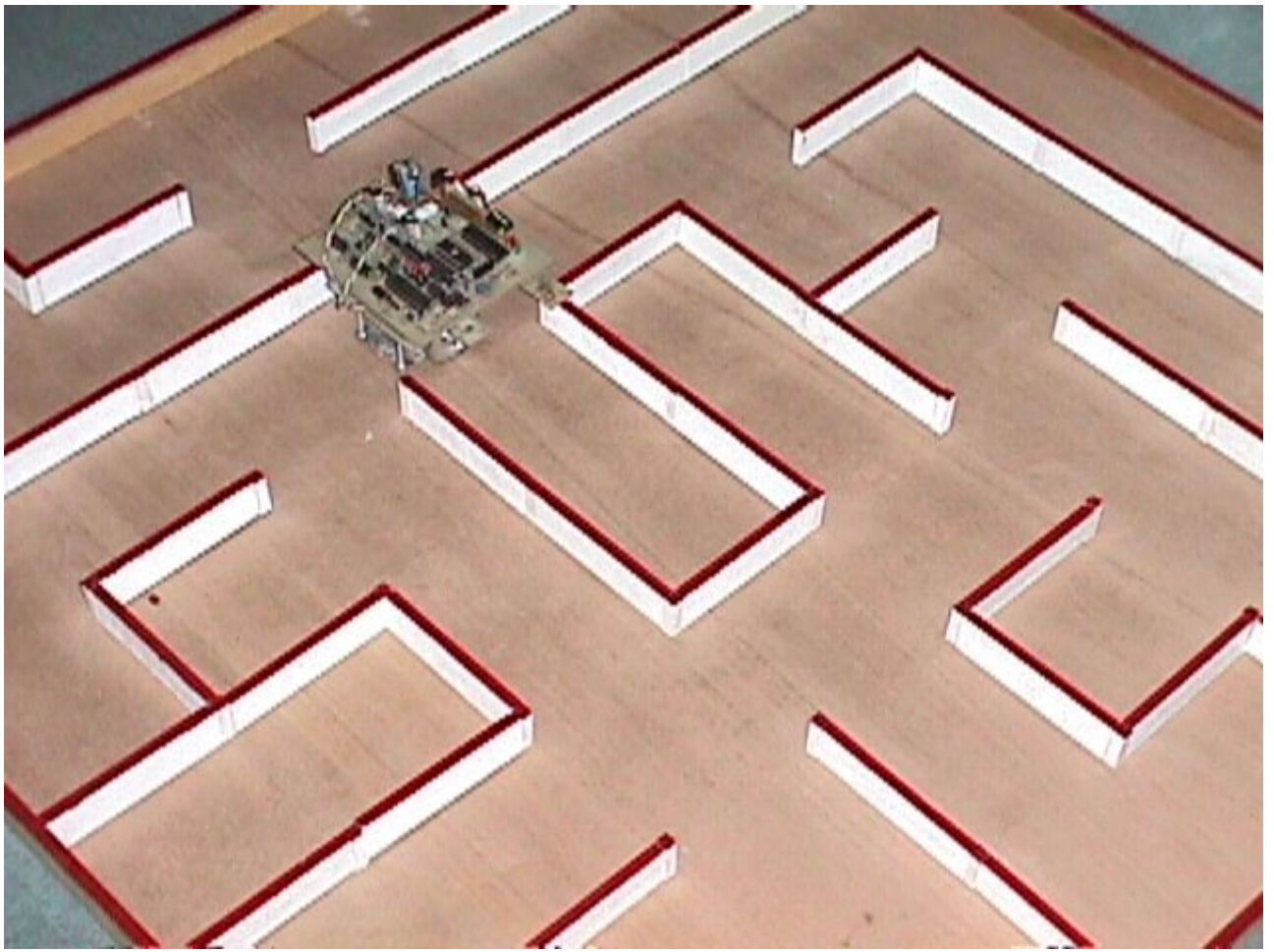
$S = Do(Drop(B_1), Do(Go(Up), Do(Pick(P_2, B_1), Do(Go(Down),$

$Do(CancelRequest(P_1), Do(AddRequest(P_2, 1, 2),$

$Do(Go(Up), Do(AddRequest(P_1, 2, 1), Do(Idle, S_0)))))))))$

Programs with Sensing

Example: Having a Robot Negotiate a Maze



A High-Level Control Program for the Robot

proc *TakeNextStep*

TurnRight;

SenseBlocked;

while *Blocked* **do**

TurnLeft; *SenseBlocked*

endWhile;

GoForward

endProc;

while ¬*At*(*Exit*) **do** *TakeNextStep* **endWhile**

Recap: Online- vs. Offline-Execution

Online-execution

- Agent commits immediately to every non-deterministic choice
- Program may not be completed althoug *s* exists s.th. $DO(\delta, S_0, s)$

  Example:

  [*Go*(*Up*); (∃*b*) *Empty*(*b*)?] | *Go*(*Down*)

Offline-execution

- Agent searches for *s* s.th. $DO(\delta, S_0, s)$ before executing any action

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

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

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

  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 $\wedge$

$\neg$*Holds*(*Blocked*, *Do*(*SenseBlocked*, *Do*(*TurnRight*, S_0))) $\wedge$

True $\wedge$ ...

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

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

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