Rappresentazione della conoscenza

Lezione 12

Sommario

- GOLOG (Reiter 6)

GOLOG: Complex Actions From Primitives

- Sequence:

  $move(B, C)$; $move(A, B)$.
- Test actions:

  $(\exists x).ontable(x) \wedge clear(x)?$
- Conditional actions:

  if $car\_in\_driveway$ **then** $drive$ **else** $walk$.
- Nondeterministic actions 1:

  $move(A, B) \mid move(A, C)$
- Nondeterministic actions 2:

$removeAblock \triangleq (\pi\ b)[block(b)?\ ;\ pickup(b)\ ;\ putOnFloor(b)]$.

- Iterative actions:

  $clear\_table \triangleq \mathbf{while}\ [(\exists b) block(b) \wedge ontable(b)]\ \mathbf{do}\ removeAblock.$

- Recursive actions:

  proc $destroyTower(x)$
  if $ontable(x) \wedge clear(x)$ **then** no_op
  else $(\pi\, y).on(x, y)?\ ;\ moveToTable(x)\ ;\ destroyTower(y)$
  end

Definizione di costruttori di azione complessi

$Do(\delta, s, s')$ – Eseguire δ in situazione s porta a s'.

- **Primitive actions** a:

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

- **Sequence**:

$$Do([a_1; a_2], s, s') \stackrel{def}{=} (\exists s'').Do(a_1, s, s'') \wedge Do(a_2, s'', s').$$

- **Nondeterministic actions**:

$$Do([a_1 \mid a_2], s, s') \stackrel{def}{=} Do(a_1, s, s') \vee Do(a_2, s, s').$$

- **Nondeterministic choice of object**:

$$Do((\pi\ x)a(x), s, s') \stackrel{def}{=} (\exists x)Do(a(x), s, s').$$

Test e Condizionali

Azioni di Test

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

ϕ è uno pseudo-fluente (mancano gli argomenti situazione)
$\phi[s]$ denota la formula $sitcalc$ ottenuta da ϕ ripristinando s

Se ϕ è: $(\forall x).ontable(x) \wedge \neg on(x, A),$

allora $\phi[s]$ sta per: $(\forall x).ontable(x, s) \wedge \neg on(x, A, s).$

Se ϕ è: $(\exists x)on(x, favoriteBlock(Mary)),$

allora $\phi[s]$ sta per: $(\exists x)on(x, favoriteBlock(Mary, s), s).$

Condizionale

if ϕ **then** δ_1 **else** δ_2 **endIf** $\stackrel{def}{=}$ $[\phi? \,;\, \delta_1] \,|\, [\neg\phi? \,;\, \delta_2],$

Cicli

Iterazione non deterministica Esegui δ 0 o più volte

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

Ciclo while

while ϕ **do** δ **endWhile** $\stackrel{def}{=}$ $[\phi? \; ; \; \delta]^* \; ; \; \neg\phi?.$

Procedure(ricorsive)

Definizione di un programma

proc $P_1\,(\vec{v}_1)\,\delta_1$ **endProc** ; $\cdots$; **proc** $P_n\,(\vec{v}_n)\,\delta_n$ **endProc** ; δ_0

Il risultato della valutazione è definito da:

$Do(\{\textbf{proc}\ P_1\,(\vec{v}_1)\delta_1\ \textbf{endProc}\ ;\cdots;\textbf{proc}\ P_n\,(\vec{v}_n)\delta_n\ \textbf{endProc}\ ;\ \delta_0\}, s, s')$
$\stackrel{def}{=} (\forall P_1, \ldots, P_n).[\bigwedge_{i=1}^{n} (\forall s_1, s_2, \vec{v}_i).Do(\delta_i, s_1, s_2) \supset P_i(\vec{v}_i, s_1, s_2)]$
$\supset Do(\delta_0, s, s').$

$P_1, \ldots, P_n$ sono le più piccole relazioni binarie chiuse rispetto alla valutazione di $\delta_1, \ldots, \delta_n$

Esempi

1. *down* significa che l'ascensore scende di un piano, *d(n)*, significa scendere di n piani.

   proc $d(n)$ $(n = 0)?$ | $d(n-1)$; $down$ **endProc**

2. parcheggiare l'ascensore a piano terra:

   proc $park$ $(\pi\ m)[atfloor(m)?$; $d(m)]$ **endProc**

Esempi: mondo dei blocchi

$clean$ significa mettere a posto nella scatola i blocchi ...

proc $clean\ (\forall x)[block(x) \supset in(x, Box)]?\ \mid$
$(\pi\ x)[(\forall y)\neg on(y, x)?\ ;\ put(x, Box)]\ ;\ clean$ **endProc**

proc $maketower\,(n)$ % Make a tower of n blocks.
$(\pi\, x, m)[tower(x, m)?\ ;$ % $tower(x, m)$ means that there is a
% tower of m blocks whose top block is x.
if $m \leq n$ **then** $stack(x, n - m)$
else $unstack(x, m - n)$
endIf]
endProc ;

proc $stack\,(x, n)$ % Put n blocks on the tower with top block x.
$n = 0? \mid (\pi\, y)[put(y, x)\ ;\ stack(y, n - 1)]$
endProc ;

proc $unstack\,(x, n)$ % Remove n blocks from the tower x.
$n = 0? \mid (\pi\, y)[on(x, y)?\ ;\ moveToTable(x)\ ;\ unstack(y, n - 1)]$
endProc ;

% main: create a seven block tower, with A clear at the end.

$maketower(7)\ ; \neg(\exists x) on(x, A)?$

Note sulla definizione di GOLOG

- Le definizioni sono macro:

  $Do(\delta, s, s')$ è una formula del $sitcalc$ che asserisce che s' è la situazione ottenuta valutando δ, a partire da s.
- valutare un **programma/piano** δ è un problema deduttivo:

$$Axioms \vdash (\exists s) Do(\delta, S_0, s).$$

Espansione di un programma GOLOG

Sia δ

$$(\pi\, x).A(x)\,;\,\phi?\,;\,[B(x)\,|\,C(x)\,;\,\psi?].$$

Costruiamo l'espansione di $Do(\delta, S_0, s)$:

$$\begin{aligned}&(\exists x).(\exists s_1).Poss(A(x), S_0) \wedge s_1 = do(A(x), S_0) \wedge\\ &\quad(\exists s_2).\phi[s_1] \wedge s_1 = s_2 \wedge\\ &\qquad[Poss(B(x), s_2) \wedge s = do(B(x), s_2) \vee\\ &\qquad\quad(\exists s_3).Poss(C(x), s_2) \wedge s_3 = do(C(x), s_2) \wedge \psi[s_3] \wedge s_3 = s].\end{aligned}$$

Semplificando:

$$\begin{aligned}&(\exists x).Poss(A(x), S_0) \wedge \phi[do(A(x), S_0)] \wedge\\ &\qquad[Poss(B(x), do(A(x), S_0)) \wedge s = do(B(x), do(A(x), S_0)) \vee\\ &\qquad\quad Poss(C(x), do(A(x), S_0)) \wedge \psi[do(C(x), do(A(x), S_0))] \wedge\\ &\qquad\qquad\qquad s = do(C(x), do(A(x), S_0))].\end{aligned}$$

GOLOG: nuovo linguaggio di LP

- GOLOG = alGOl in LOGic.
- Situation calculus semantics.
- L'interprete GOLOG è un theorem-prover.
- Assiomi della teoria:
  - Situatione iniziale: lo stato prima dell'esecuzione del piano.

$$ontable(A, S_0),\ blue(B, S_0) \vee red(B, S_0),$$

  - Assiomi di precondizione, per ogni azione primitiva.
  - Assiomi di stato Successore, per ogni fluente.

GOLOG: valutazione dei piani

- Valutazione di un programma δ:

$$Axioms \vdash (\exists s) Do(\delta, S_0, s).$$

  La valutazione si ottiene tramite la macro-espansione di $Do(\delta, S_0, s)$ in una formula del situation calculus.
- Trovare s per ottenere una **traccia** di esecuzione, che rappresenta le azioni che l'agente deve eseguire.
- ogni valutazione di un programma GOLOG fornisce un programma eseguibile:

  $\Sigma \models (\forall s).Do(\delta, S_0, s) \supset executable(s).$

Proprietà dei programmi GOLOG

Pianificazione = sintesi di programmi

$$Axioms \models (\exists \delta, s).Do(\delta, S_0, s) \wedge Goal(s).$$

La definizione di GOLOG non consente di sintetizzare programmi (s può essere solo un termine "situazione", cioè una sequenza di azioni).

Altre proprietà

1. **Correttezza**:

   $Axioms \models (\forall s).Do(\delta, S_0, s) \supset P(s).$ oppure

   $Axioms \models (\forall s_0, s).I(s_0) \wedge Do(\delta, s_0, s) \supset P(s).$

2. **Terminazione**:

   $Axioms \models (\exists s)Do(\delta, S_0, s).$ oppure

   $Axioms \models (\forall s_0).I(s_0) \supset (\exists s)Do(\delta, s_0, s).$

Esempi di piani in GOLOG: Ascensore

Primitive actions:

- $up(n)$. L'ascensore sale al piano n.
- $down(n)$. L'ascensore scende al piano n.
- $turnoff(n)$. L'ascensore spegne il bottone al piano n.
- $open$. L'ascensore apre la porta.
- $close$. L'ascensore chiude la porta.

Esempi di piani in GOLOG: Ascensore

Fluents:

$currentFloor(s) = n$. In s, l'ascensore è al piano n.

$on(n, s)$. In s, il bottone di chiamata al piano n è acceso.

Primitive action preconditions:

$Poss(up(n), s) \equiv currentFloor(s) < n.$

$Poss(down(n), s) \equiv currentFloor(s) > n.$

$Poss(open, s) \equiv true.$

$Poss(close, s) \equiv true.$

$Poss(turnoff(n), s) \equiv on(n, s).$

Esempi di piani in GOLOG: Ascensore

Successor state axioms:

$$currentFloor(do(a,s)) = m \equiv \{a = up(m) \vee a = down(m) \vee currentFloor(s) = m \wedge \neg(\exists n)a = up(n) \wedge \neg(\exists n)a = down(n)\}.$$

$$on(m, do(a,s)) \equiv on(m,s) \wedge a \neq turnoff(m).$$

An abbreviation:

$$nextFloor(n,s) \stackrel{def}{=} on(n,s) \wedge (\forall m).on(m,s) \supset |m - currentFloor(s)| \geq |n - currentFloor(s)|.$$

Definisce il prossimo piano da servire come il più vicino a quello in cui si trova l'ascensore.

I piani GOLOG

proc $serve(n)$
$goFloor(n)$; $turnoff(n)$; $open$; $close$ **endProc**.

proc $goFloor(n)$
$(currentFloor = n)?$ | $up(n)$ | $down(n)$ **endProc**.

proc $serveAfloor$
$(\pi\ n)[nextFloor(n)?$; $serve(n)]$ **endProc**.

proc $control$
[**while** $(\exists n)on(n)$ **do** $serveAfloor$ **endWhile**] ; $park$
endProc.

proc $park$
if $currentFloor = 0$ **then** $open$ **else** $down(0)$; $open$ **endIf**
endProc.

Situatione iniziale ed Esecuzione del piano

$$currentFloor(S_0) = 4, \quad on(b, S_0) \equiv b = 3 \vee b = 5.$$

OK per il Prolog!

$$Axioms \models (\exists s) Do(control, S_0, s).$$

$Axioms$ = assiomi fondazionali del SitCalc, assiomi della teoria dell'ascensore e situazione iniziale.

Commenti

$Do(control, S_0, s)$ sembra una formula atomica, ma Do si macro-espande in una frase del SitCalc che comprende le azioni up, $down$, $turnoff$, $open$, $close$ ed i fluenti $currentFloor$, on, ed i simboli do, S_0, $Poss$ del SitCalc.

L' "esecuzione" del piano potrebbe ritornare il seguente legame per s:

$$s = do(open, do(down(0), do(close, do(open, do(turnoff(5),\\ do(up(5), do(close, do(open, do(turnoff(3), do(down(3), S_0)))))))))$$

Molto ancora da vedere ...

- **azioni sensoriali**: acquisizione di informazioni e conoscenza
- **azioni esogene**: il mondo non cambia solo per effetto dell'agente
- **azioni non istantanee**: un'azione può richiedere un certo tempo prima di produrre gli effetti desiderati
- **azioni concorrenti**: l'agente può, in genere, eseguire più azioni simultaneamente