Rappresentazione della conoscenza

Esercitazione 8

Sommario

♢ GOLOG (Reiter 6.3, 6.4)

Interprete GOLOG

- Valutazione di un piano 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:

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

GOLOG

GOLOG è programmazione logica perché:

1. L'interprete è un theorem prover (della logica del secondo ordine, in pratica, del primo ordine).
2. I programmi GOLOG vengono eseguiti per ottenere legami per le variabili quantificate esistenzialmente.

Interprete GOLOG in Prolog: sintassi

```
:- op(800, xfy, [&]).    /* Conjunction */
:- op(850, xfy, [v]).    /* Disjunction */
:- op(870, xfy, [=>]).   /* Implication */
:- op(880,xfy, [<=>]).   /* Equivalence */
:- op(950, xfy, [:]).    /* Action sequence */
:- op(960, xfy, [#]).    /* Nondeterministic action choice */
```

Interprete GOLOG in Prolog: semantica

```
do(E,S,do(E,S)) :- primitive_action(E), poss(E,S).
do(E1 : E2,S,S1) :- do(E1,S,S2), do(E2,S2,S1).
do(?(P),S,S) :- holds(P,S).
do(E1 # E2,S,S1) :- do(E1,S,S1) ; do(E2,S,S1).
do(if(P,E1,E2),S,S1) :-
              do(?(P) : E1 # ?(-P) : E2,S,S1).
do(star(E),S,S1) :- S1 = S ; do(E : star(E),S,S1).
do(while(P,E),S,S1):- do(star(?(P) : E) : ?(-P),S,S1).
do(pi(V,E),S,S1) :- sub(V,_,E,E1), do(E1,S,S1).
do(E,S,S1) :- proc(E,E1), do(E1,S,S1).
```

Valutazione delle condizioni

Una condizione p può essere atomica oppure una formula.

$\diamondsuit$ La verità dei fluenti viene verificata dagli assiomi di stato successore e dalle condizioni iniziali.

$\diamondsuit$ Per le formule si applicano le trasformazioni Lloyd-Topor basate sulla negazione come fallimento.

holds

```
holds(P & Q,S) :- holds(P,S), holds(Q,S).
holds(P v Q,S) :- holds(P,S); holds(Q,S).
holds(P => Q,S) :- holds(-P v Q,S).
holds(P <=> Q,S) :- holds((P => Q) & (Q => P),S).
holds(-(-P),S) :- holds(P,S).
holds(-(P & Q),S) :- holds(-P v -Q,S).
holds(-(P v Q),S) :- holds(-P & -Q,S).
holds(-(P => Q),S) :- holds(-(-P v Q),S).
holds(-(P <=> Q),S) :- holds(-((P => Q) & (Q => P)),S).
holds(-all(V,P),S) :- holds(some(V,-P),S).
holds(-some(V,P),S) :- \+ holds(some(V,P),S).  /* Negation */
holds(-P,S) :- isAtom(P), \+ holds(P,S).     /* by failure */
holds(all(V,P),S) :- holds(-some(V,-P),S).
holds(some(V,P),S) :- sub(V,_,P,P1), holds(P1,S).
```

Ripristino delle situazioni

```
holds(A,S) :- restoreSitArg(A,S,F), F ;
              \+ restoreSitArg(A,S,F), isAtom(A), A.

isAtom(A) :- \+ (A = -W ; A = (W1 & W2) ; A = (W1 => W2) ;
    A = (W1 <=> W2) ; A = (W1 v W2) ; A = some(X,W) ; A = all(X,W)).
```

Il programmatore GOLOG deve indicare una clausola per ogni fluente per il ripristino della situazione. Es.:

```
    restoreSitArg(ontable(X),S,ontable(X,S)).                */
```

Sostituzioni

```
sub(_,_,T1,T2) :- var(T1), T2 = T1.
sub(X1,X2,T1,T2) :- \+ var(T1), T1 = X1, T2 = X2.
sub(X1,X2,T1,T2) :- \+ T1 = X1, T1 =..[F|L1],
                    sub_list(X1,X2,L1,L2),
                    T2 =..[F|L2].
sub_list(_,_,[],[]).
sub_list(X1,X2,[T1|L1],[T2|L2]) :-
        sub(X1,X2,T1,T2),
        sub_list(X1,X2,L1,L2).
```

Ipotesi per l'implementazione: regressione

- fluenti relazionali (in numero finito) ed un numero finito di azioni
- in S_0
  - assiomi di nome unico
  - una definizione per ogni predicato non-fluente
  - fluenti relazionali:$F(\vec{x}, S_0) \equiv \Psi_F(\vec{x}, S_0)$

Garantiscono la correttezza dell'implementazione in Prolog della regressione (dopo le trasformazioni Lloyd-Topor).

Teoria del dominio in Prolog

- non-fluenti: $\Theta_P(\vec{x}) \supset P(\vec{x})$
  al posto di $\Theta_P(\vec{x}) \equiv P(\vec{x})$
- fluenti in S_0: $\Psi_F(\vec{x}, S_0) \supset F(\vec{x}, S_0)$
  al posto di $\Psi_F(\vec{x}, S_0) \equiv F(\vec{x}, S_0)$
- assiomi precondizione: $\Pi_A(\vec{x}, s) \supset Poss(A(\vec{x}), s)$
  al posto di $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$
- assiomi stato successore: $\Phi_F(\vec{x}, a, s) \supset F(\vec{x}, do(a, s))$
  al posto di $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$

Specificata dal programmatore GOLOG.

Specifica GOLOG

- `primitive_action(act)` per ogni azione primitiva
- `proc(name,body)` per ogni procedura
- `restoreSitArg(fluentAtom,S,fluentAtom[S])` per ogni fluente

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.

Specifica Prolog: azioni primitive

```
primitive_action(turnoff(_N)).
primitive_action(open).
primitive_action(close).
primitive_action(up(_N)).
primitive_action(down(_N)).
```

Specifica Prolog: procedure

```
proc(goFloor(N), ?(currentFloor(N)) # up(N) # down(N)).
proc(serve(N), goFloor(N) : turnoff(N) : open : close).
proc(serveAfloor, pi(n, ?(nextFloor(n)) : serve(n))).
proc(park, if(currentFloor(0), open, down(0) : open)).

proc(control, while(some(n, on(n)), serveAfloor) : park).
```

Specifica Prolog: ripristino situazioni

```
restoreSitArg(on(N),S,on(N,S)).
restoreSitArg(nextFloor(N),S,nextFloor(N,S)).
restoreSitArg(currentFloor(M),S,currentFloor(M,S)).
```

Specifica Prolog: precondizioni

```
poss(up(N),S) :- currentFloor(M,S), M < N.
poss(down(N),S) :- currentFloor(M,S), M > N.
poss(open,_S).
poss(close,_S).
poss(turnoff(N),S) :- on(N,S).
```

Specifica Prolog: assiomi stato successore

```
currentFloor(M,do(A,S)) :-
    A = up(M) ;
    A = down(M) ;
    not(A = up(N)), not(A = down(N)), currentFloor(M,S).

on(M,do(A,S)) :- on(M,S), not(A = turnoff(M)).
```

Specifica Prolog: stato iniziale

```
on(3,s0).
on(5,s0).
currentFloor(4,s0).

% scelta del piano da servire
nextFloor(N,S) :- on(N,S).
```

Osservazioni

$Do(control, S_0, s)$ 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:

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

Sommario

- È necessario che gli assiomi di dominio siano rappresentabili in Prolog.
- Si esprimono le parti-if degli assiomi di stato successore e di precondizione.
- Lo stato iniziale deve essere esprimibile in Prolog

  $\Longrightarrow$

  tenendo conto della *Closed World Assumption* che completa l'informazione dello stato iniziale.
- In queste ipotesi l'interprete è corretto (Levesque).
- La teoria vale in ambito più generale anche quando si ha informazione **incompleta**.

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).