

## LEZIONE 3

### Planning III

## Pianificatori

- Piano: sequenza di azioni che risolve un goal
- Pianificatore: algoritmo di generazione di un piano
  - ha in input una rappresentazione del mondo nel suo stato iniziale, delle azioni e dei goals
    - eventualmente in una delle rappresentazioni viste nella lezione precedente
  - genera un piano la cui esecuzione porta dal mondo iniziale ad un mondo finale nel quale il goal è raggiunto
  - distinguere fra piano ed esecuzione del piano
    - costi di calcolo del piano
    - costi di esecuzione del piano

## 1. Forward Planning

- Rappresentazione basata sugli stati (STRIPS, Situation C.)
- Ricerca di un piano come ricerca di un cammino nello spazio degli stati
  - neighbors(S,[S1,...,Sn]) con action(Ai,S,Si)
  - algoritmo: A\* con una buona euristica (fattore di ramificazione alto!) o iterative deepening
- descrizione degli stati S:
  - descrizione completa del mondo: occupa spazio, richiede tempo per generare i mondi
  - descrizione tipo le situazioni del Situation Calculus: più compatta, ma: tempo calcolo se sono necessarie molte informazioni sul mondo, ad es. decidere se due mondi sono uguali per tagliare cicli

## 2. STRIPS Planner

- L'idea è la ricerca all'indietro del piano, a partire dal goal che si vuol conseguire, in modo simile a quanto accade con la ricerca di un albero di prova
- ma gli stati del mondo entrano nel procedimento di ricerca come informazioni essenziali
- nella rappresentazione che mostriamo, gli stati sono situazioni rappresentate come nel situation calculus, che usiamo come meta-linguaggio per formalizzare STRIPS e descrivere l'algoritmo
- Nota. L'algoritmo stesso può essere usato nel Situation Calculus, indipendentemente dalla rappresentazione di STRIPS

### A. Rappresentazione di STRIPS nel situation calculus

```
/** i fatti che diventano veri perché nella addlist dell'azione A **/  
holds(C, do(A,W)) :- preconditions(A,P),  
                    holdsall(P,W),  
                    addList(A,AL),  
                    member(C,AL).  
  
/** i fatti che rimangono veri per inerzia **/  
holds(C, do(A,W)) :- preconditions(A,P),  
                    holdsall(P,W),  
                    deleteList(A,DL),  
                    not(member(C,DL)),  
                    holds(C,W).  
  
/** gli altri fatti, quelli in DL, non valgono per CWA **/
```

Esercizio: scrivere holdsall

### B. Pianificatore STRIPS

```
/** achieve(G,W0,W1): W1 è il mondo risultante avendo raggiunto  
    il goal G, a partire dal mondo W0 **/  
  
/***** già raggiunto G *****/  
achieve(G,W,W) :- holds(G,W).  
  
/***** G relazione dinamica derivata, definito da G ← B *****/  
achieve(G,W0,W1) :- clause(G,B),  
                    achieve_all(B,W0,W1).  
  
/***** G relazione dinamica primitiva ottenuta con Act *****/  
achieve(G,W0,do(Act,W1)) :- achieves(Act,G),  
                             preconditions(Act,Pre),  
                             achieve_all(Pre,W0,W1).
```

### ESERCIZIO:

- Dare `achieve_all(ListaGoals, W0, W1)`, che cerca di raggiungere tutti i goals della lista nell'ordine in cui si trovano.
- Vedete dei problemi nel fatto di prendere per buono l'ordine con cui si trovano i goals nella lista? Confrontate con quanto accade nell'algoritmo top-down di ricerca di un albero di prova in DCL e il tipo di non-determinismo nella scelta del goal da risolvere:
  - era don't care
  - lo è ancora?
- Suggestimento: pensate a dover raggiungere la lista di goals:  
[in(s2,p), sporco(s1,0)]  
dove per ottenere sporco(s1,0) p si deve recare in s1;  
se va(p,s1,s2) è possibile .....

### Soluzioni al problema evidenziato nell'esercizio

- Provare per diverse permutazioni dei goals
  - costoso
- Stabilire una relazione di precedenza
  - se l'azione per ottenere un goal G disfa un goal G', G deve precedere G';
  - così [in(s2,pulitore), sporco(s1,0)] va riordinato:  
[sporco(s1,0), in(s2,pulitore)]

### 3. Altra soluzione: regression planner

- Idea: organizzare la ricerca di un piano in uno spazio di ricerca basato sulle pre-condizioni più deboli wp (weakest precondition)
  - nei programmi (Dijkstra):
    - $wp(\{H(x)\}, x:=t) = \{H(t)\}$ , ecc.
  - nelle azioni Act rappresentate alla STRIPS, con `pre(Act)`, `addList(Act)`, `delList(Act)`:

$$wp(\text{Goals}, \text{Act}) = \text{pre}(\text{Act}) \cup \{G \mid G \in \text{Goals} \text{ e } G \notin \text{delList}(\text{Act})\}$$

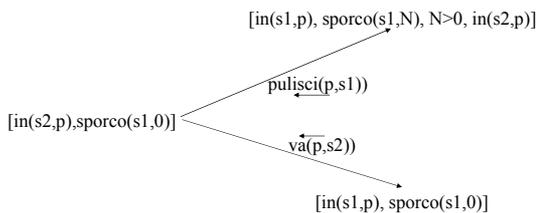
azione pulisci(P,S): pre: [in(S,P), sporco(S,N), N>0]  
del: [sporco(S,N)]  
add: [sporco(S,0)]

$wp([\text{in}(s2,p), \text{sporco}(s1,0)], \text{pulisci}(p,s1))$   
= [in(s1,p), sporco(s1,N), N>0, in(s2,p)]

azione va(P,S,S1): pre: [in(S,P)]  
del: [in(S,P)]  
add: [in(S1,P)]

$wp([\text{in}(s2,p), \text{sporco}(s1,0)], \text{va}(p,s2))$   
= [in(s1,p), sporco(s1,0)]

Spazio ricerca: nodi collegati in base a wp:



Scopo: raggiungere lo stato iniziale all'indietro  
(dimostrare dallo stato iniziale si raggiunge la situazione desiderata)

Un planner di regressione

*/\* solve(Goals, W) : i goal in Goals sono veri in W \*/*

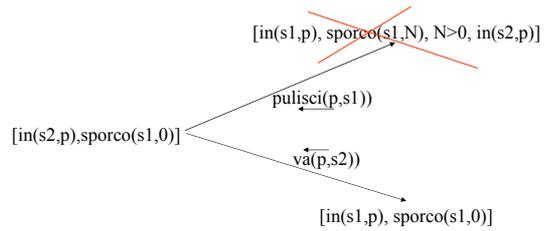
```
solve(Goals, init) :- holdsall(Goals,init).
solve(Goals, do(A,W)) :- consistent(Goals),
                           chooseGoal(G,Goals),
                           chooseAct(Act,G),
                           wp(Act,Goals,NewGoals),
                           solve(NewGoals,W).
```

*/\* chooseAct(Act,G): Act è un'azione che raggiunge G \*/*

## Il problema della consistenza

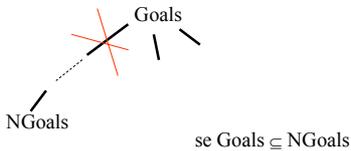
- Può accadere che una wp sia inconsistente, cioè contenga coppie  $A, \neg A$ ; questo caso si riconosce e si taglia
- Può accadere che una wp sia inconsistente nel dominio di problema, ad esempio il pulitore non può trovarsi contemporaneamente in 2 stanze
  - andrebbe tagliata dallo spazio di ricerca
  - ma il problema di derivare le inconsistenze dimostrabili nel dominio di problema può non essere facile; può convenire mantenere un'opportuna base di conoscenza sul dominio

p non può trovarsi in s1 e in s2:



## Tagliare i cicli

- Aggiungere meccanismo per tagliare cicli: se un antenato è contenuto nell'insieme di goals correnti, basta risolvere l'antenato



## 4. Partial order planning, cenni

- Abbiamo visto che, nel caso del planner STRIPS, è utile stabilire una relazione di precedenza
  - se l'azione per ottenere un goal  $G$  disfa un goal  $G'$ ,  $G$  deve precedere  $G'$ ;
  - così  $[in(s2,pulitore), sporco(s1,0)]$  va riordinato:  $[sporco(s1,0), in(s2,pulitore)]$
- In realtà solo certe precedenze sono obbligate, mentre in molti casi non vi è un ordine imposto e la scelta è don't care
  - su quest'osservazione si basano i Partial Order Planner o pianificatori non lineari che vedremo la prossima volta