Planejamento

Capítulo 6

Planejamento

- Para muitos problemas, a noção de planejamento é a mesma de *busca por solução*
- Exemplo: 8-puzzle
- Outros domínios de problemas fazem clara distinção entre busca e planejamento:
  - aqueles cujo universo não é previsível
  - quando passos não podem ser desfeitos (não é possível backtracking).

Navegação de Robôs

- O objetivo é movimentar um robô de uma sala para outra sem bater em obstáculos.
  - O que fazer se uma cadeira é movimentada?
  - O que fazer se uma criança salta em frente ao computador?
  - O que fazer se uma pessoa tenta parar o robô e o robô é forçado a matá-la, e o corpo está bloqueando a porta?

343

Planejamento & Decomposição

- Problemas que são decomponíveis ou *parcialmente decomponíveis* podem sempre ser resolvidos por um sistema que:
  - Possa resolver cada subproblema de forma independente
  - Saiba quais subproblemas não são independentes
  - Possa revisar a solução dos subproblemas quando algo não esta de acordo com o *plano*.
- Esta técnica é chamada *planejamento*

344

Exemplo de Planejamento

- Robô desenvolve um *plano* para ir a outra sala. O plano envolve:
  - Movimentação em torno do sofá
  - Movimentação do sofá para a porta
  - Abertura da porta
  - Movimentação através da porta

345

Representação do Estado

- Este tipo de problema sempre inclui descrições de estado muito complexas
- Pode não ser possível criar uma cópia *completa* de toda informação para cada estado considerado durante a busca
- O estado sucessor pode ser descrito pelas mudanças que são feitas por um operador
- É necessário armazenar o caminho dessas mudanças para permitir backtraking e evitar loops.

346

Espaço de Estados do Robô

- Localização de todos os obstáculos, paredes, portas, salas, etc.
- Localização do robô
- Velocidade e direção do movimento do robô
- Posição de todos os periféricos robóticos (braços, sensores, câmeras).

347

Como resolver problemas?

- Nós criamos um plano que inclui as soluções a um determinado número de subproblemas
- Depois de executar a solução para o subproblema 1, algo vai mudar
- É necessário saber quais outros subproblemas serão afetados, e revisar cada uma de suas soluções
- A chave é que nós não queremos recomeçar do nada

348

Planejando um conjunto de soluções

- Algumas vezes é possível criar um número de planos, cada um pode trabalhar com obstáculos específicos que podem surgir
- Quando algo dá errado, nós já temos um plano de contingência pronto
- Nem sempre isto é possível ou prático (o conhecimento necessário não está disponível ou é muito grande).

349

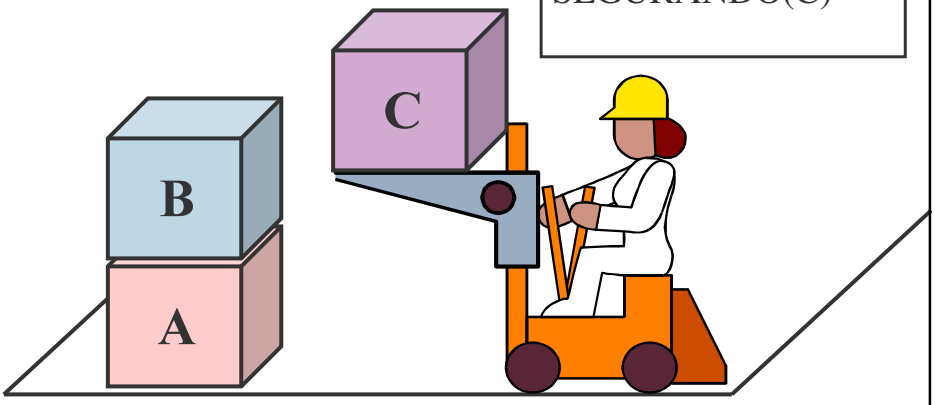
Dependências entre Subtarefas

- O plano inclui informação que descreve as dependências entre todas as subtarefas
- A utilização desta informação para revisar planos é chamada de *backtraking dirigido a dependência*
- É sempre mais fácil determinar estas dependências se o processo de planejamento trabalha a partir dos estados meta anteriores

350

Mundo dos Blocos

SOBRE(B,A)
SOBRE_MESA(A)
LIVRE(B)
SEGURANDO(C)



351

Mundo dos Blocos (Cont.)

Predicados usados para descrever o estado:

SOBRE(A,B)	Bloco A está sobre B
SOBRE_MESA(A)	Bloco A está sobre a mesa
LIVRE(A)	Hão há nada sobre A
SEGURANDO(A)	O robô está segurando A
BRAÇOVAZIO	O braço do robô está vazio

352

Operações no Mundo dos Blocos

- *retira(A,B)* retira A de cima de B
- *coloca(A,B)* coloca A sobre B
- *levanta(A)* levanta o bloco A.
- *abaixa(A)* coloca A sobre a mesa
- *larga(A)* larga A.

353

Exemplo de Restrições de Operação

retira(A,B):

BRAÇOVAZIO	o braço deve estar vazio
LIVRE(A)	nada sobre A
SOBRE(A,B)	A deve estar sobre B

354

Alguns Axiomas do Mundo dos Blocos

$$[\forall x : \text{SEGURANDO}(x)] \Rightarrow \neg\ \text{BRAÇOVAZIO}$$

$$\forall\ x : \text{SOBRE\_MESA}(x) \Rightarrow \neg\ \forall y : \text{SOBRE}(x,y)$$

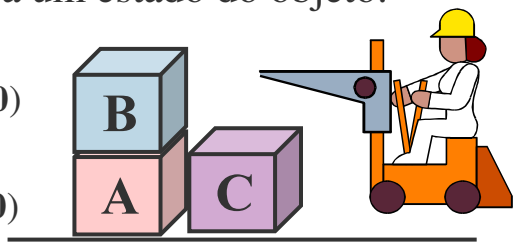
$$\forall\ x : [\neg\ \forall y : \text{SOBRE}(y,x)] \Rightarrow \text{LIVRE}(x)$$

355

Raciocinando no Mundo dos Blocos

- Representação do estado é um conjunto de predicados que são verdadeiros para aquele estado
- Cada predicado se refere a um estado do objeto:

  SOBRE(B,A,**estado0**)
  SOBRE_MESA(A, **estado0**)
  LIVRE(B, **estado0**)
  SOBRE_MESA(C, **estado0**)
  LIVRE(C, **estado0)**



356

Mundo dos Blocos & Resolução

- O efeito de cada operador é codificado em lógica de predicados:

  LIVRE(x,s) $\wedge$ SOBRE(x,y,s) $\Rightarrow$
  SEGURANDO(x, *Faça(retire(x,y),s)*) $\wedge$
  LIVRE(y, *Faça(retire(x,y),s)*)

  Faça é uma função que especifica o estado resultante da aplicação de um operador

357

Estado0:

SOBRE(B,A,**estado0**)
SOBRE_MESA(A, **estado0**)
LIVRE(B, **estado0**)
SOBRE_MESA(C, **estado0**)
LIVRE(C, **estado0**)

Operador Retire

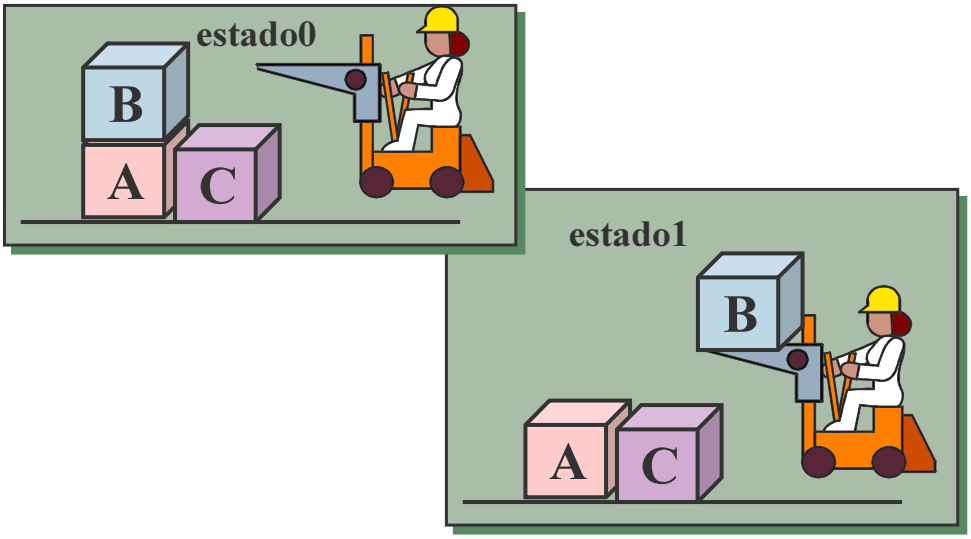
LIVRE(x,s) $\wedge$ SOBRE(x,y,s) $\Rightarrow$
SEGURANDO(x, *Faça(retire(x,y),s)*)
$\wedge$
LIVRE(y, *Faça(retire(x,y),s)*)

estado1 = *Faça(retire(B,A),* **estado0**)

SEGURANDO(B, **estado1**) $\wedge$ LIVRE(A, **estado1**)

358

estado1 = *Faça(retire(B,A),* **estado0**)



359

SOBRE_MESA(A,**estado1**) é verdade ?

- O operador *retire* que produziu o estado1 não diz nada sobre o que foi modificado
- Nós precisamos adicionar algum *axioma* que especifique o que não mudou quando um operador foi aplicado

SOBRE_MESA(z,s) ⇒ SOBRE_MESA(z, *Faça(retire(x,y),*s))

360

Usando Resolução

- Em geral, nós precisamos adicionar muitos *axiomas* para ter certeza que a informação pode ser derivada
- Se nós especificamos todos os axiomas necessários, Resolução é o único mecanismo necessário
- Nem sempre é possível especificar todos os axiomas necessários

361

Operadores Híbridos

Para cada Operador:

precondições: predicados que devem ser verdadeiros antes que o operador possa ser aplicado (restrições).

adições: predicados que o operador torna verdadeiros

deleções: predicados que o operador torna falsos

Qualquer coisa não especificada é assumida ser não modificada

362

coloca(x,y)

Precondições: LIVRE(y) ∧ SEGURANDO(x)

Adições: BRAÇOVAZIO ∧ SOBRE(x,y)

Deleções: LIVRE(y) ∧ SEGURANDO(x)

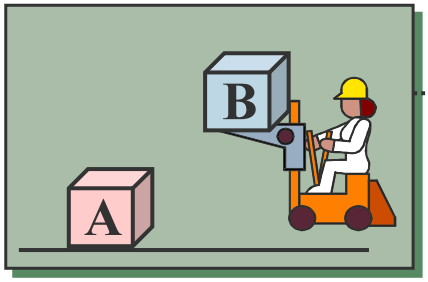
retira(x,y)

Precondições:

Adições:

Deleções:

363

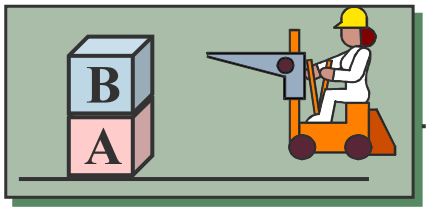


BORE_MESA(A) ∧ LIMPO(B) ∧
LIMPO(A)∧SEGURANDO(B) ∧
¬ BRAÇOVAZIO

+

retira(B,A)

Adições: BRAÇOVAZIO ∧ SOBRE(x,y)
Dels.: LIVRE(y) ∧ SEGURANDO(x)



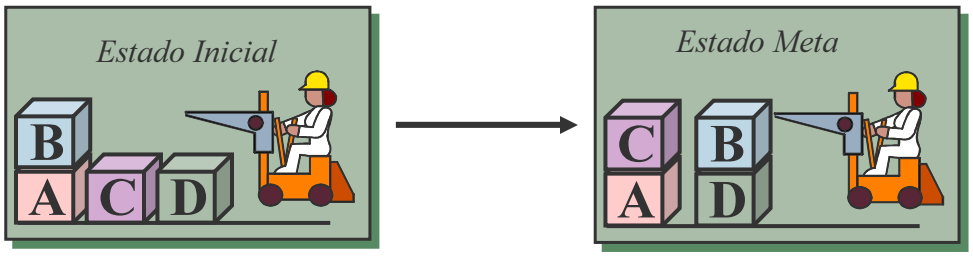
SOBRE(B,A) ∧ SOBRE_MESA(A) ∧
LIVRE(B) ∧ BRAÇOVAZIO

364

Planejando Metas Empilhadas

- Divida o problema em subproblemas
- Para cada subproblema:
  - Coloque a descrição da submeta em uma pilha
- Enquanto não realizada
  - obtenha a submeta da pilha
  - Encontre a seqüência de operadores que alcançarão a submeta. Aplique operadores no corrente estado

365



SOBRE(B,A) ∧
SOBRE_MESA(A) ∧
SOBRE_MESA(C) ∧
SOBRE_MESA(D) ∧
BRAÇOVAZIO

SOBRE(C,A) ∧
SOBRE(B,D) ∧
SOBRE_MESA(A) ∧
SOBRE_MESA(D)

366

Pilha de Metas

1. SOBRE(C,A) ∧ SOBRE(B,D) ∧ SOBRE_MESA(A)∧ SOBRE_MESA(D)

Obtenha a meta da pilha, divida-a em submetas, coloque a meta original na pilha, seguida por cada meta insatisfeita

1. SOBRE(C,A)
2. SOBRE(B,D)
3. SOBRE(C,A) ∧ SOBRE(B,D) ∧ SOBRE_MESA(A)∧ SOBRE_MESA(D)

367

Submeta: SOBRE(C,A)

- Examine operadores para uma submeta que tenha um predicado SOBRE na lista *Adds*
- Encontre aquele *coloca*(C,A) que realiza a tarefa
- Coloque *coloca*(C,A) na pilha de metas
  - Não precisa colocar SOBRE(C,A) na pilha, uma vez que coloca(C,A) torna SOBRE(C,A) verdade

368

Submeta: *coloca*(C,A)

- *coloca*(C,A) tem precondições, coloque-as na pilha de metas

1. LIVRE(A) ∧ SEGURANDO(C)
2. coloca(C,A)
2. SOBRE(B,D)
3. SOBRE(C,A) ∧ SOBRE(B,D) ∧ SOBRE_MESA(A)∧SOBRE_MESA(D)

369

Submeta: LIVRE(A) ∧ SEGURANDO(C)

- Separe em 2 submetas e coloque-as na pilha. Neste caso é claro que a ordem pode ser muito importante!

1. LIVRE(A)
2. SEGURANDO(C)
3. LIVRE(A) ∧ SEGURANDO(C)
2. coloca(C,A)
2. SOBRE(B,D)
3. SOBRE(C,A) ∧ SOBRE(B,D) ∧ SOBRE_MESA(A)∧SOBRE_MESA(D)

370

Submeta: LIVRE(A)

- LIVRE(A) não é verdade neste momento, mas retira(B,A) a tornará verdade
- Coloque retira(B,A) na pilha no lugar de LIVRE(A)
- retira(B,A) tem precondições:
  - SOBRE(B,A) $\wedge$ LIVRE(B) $\wedge$ BRAÇOVAZIO

371

Backtracking

- A Divisão de uma meta em submetas e a colocação de cada uma delas em uma pilha é uma forma de planejamento
- Quando a solução para uma submeta aponta para um próximo passo, backtracking é necessário
- Backtracking não é feito explicitamente, mas implicado por processos
  - cada submeta tomada da pilha é resolvida em relação ao estado corrente

372

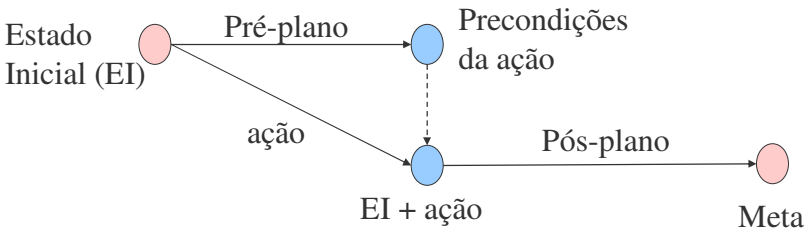
A Solução

- A solução é a seqüência de operadores que foram aplicados
- Uma vez que nós sabemos o que cada operador faz (das *adições* e *deleções*) é possível reduzir uma seqüência de operadores
- Exemplo: coloca(A,B), retira(A,B) => { }

373

Análise Meios-Fins em Planejamento

- Encontre uma ação útil (que reduza a diferença com a meta)



- Crie dois novos subproblemas:
  - tornar verdadeiras as precondições da ação a partir de EI
  - encontrar a meta a partir do resultado da ação sobre EI

374

Análise Meios-Fins – Algoritmo GPS

GPS(*estado-inicial*, *meta*)

1. Se *meta* $\subseteq$ *estado-inicial*, então retorne *true*
2. Selecione uma diferença *d* entre *meta* e o *estado-inicial*
3. Selecione um operador *O* que reduz a diferença *d*
4. Se nenhum outro operador então *falhe*
5. *Estado* = **GPS**(*estado-inicial*,*precondições(O)*)
6. Se Estado, então retorne **GPS**(**apply**(*O*,*estado-inicial*), *meta*)

375

Resolvendo o mundo dos blocos com Meios-Fins

```
meta([livre(a),sobre(a,b),sobre(b,c),
      sobre(c,d),sobre_a_mesa(d),braco_vazio]).

atual([livre(d),sobre(b,a),sobre(c,b),sobre(d,c),
       sobre_a_mesa(a),braco_vazio]).

operador(desempilhar,
         [sobre(X,Y),livre(X), braco_vazio],
         [sobre(X,Y),braco_vazio],
         [segurando(X),livre(Y)],(X\=Y)).
operador(empilhar,
         [livre(Y),segurando(X)],
         [livre(Y),segurando(X)],
         [braco_vazio,sobre(X,Y)],(X\=Y)).

operador(pegar,
         [livre(X),sobre_a_mesa(X),braco_vazio],
         [sobre_a_mesa(X),braco_vazio],
         [segurando(X)],true).

operador(largar,
         [segurando(X)],
         [segurando(X)],
         [sobre_a_mesa(X),braco_vazio],true).
```

376

Resolvendo o mundo dos blocos com Meios-Fins

```
seleciona_operador1(Diferenca,Atual,Meta) :-
        Diferenca \= [],
        operador(O,_,_,_,_),  % Encontra um operador
        faca(O,Atual,S,P),    % Executa o operador sobre atual
        nao_visitado([O,S]),  % Verifica se estado não foi visitado
        diferenca(S,Diferenca,L),length(L,X),length(Diferenca,Y), X =< Y,
        \+ go_back_action([O,S]),
        insert_last_action([O,Atual]),
        write([O,S]),nl,
        meios_fins(Atual,P),
        meios_fins(S,Meta).
```

Verifica se a diferença diminuiu

Verifica se ação não irá desfazer a ação anterior

377

Planejamento Não-Linear

- Algumas vezes é necessário atacar múltiplas submetas ao mesmo tempo
- O desenvolvimento de um plano que pode fazer isto é chamado *planejamento não-linear*
- O uso de listas de precondição, adição e deleção torna possível determinar o efeito de intercalar soluções (operadores).

378

Planejamento Hierárquico

1. Tente estabelecer, primeiramente, um plano geral

2. Refine cada um dos passos do plano

3. Refine cada refinamento (e assim sucessivamente…)

379

Planejamento Hierárquico (Cont.)

Duas Técnicas:

- Macro Operadores - constrói operadores complexos além dos pequenos operadores

- Associa pesos a precondições. Inicialmente procure apenas satisfazer precondições importantes (mas resolva o problema inteiro). Então se preocupe com precondições menos importantes

380

Sistemas Reativos

- Desenvolva uma submeta, e entre em ação para alcançar a submeta
- Dê uma olhada no resultado, invente uma nova submeta
- Continue indo até alcançar a meta

381

Sistemas Reativos (Cont.)

- Trabalha bem para muitos tipos de tarefas simples.
- Freqüentemente uma combinação de reação e planejamento é requerida:
  - Não pode esperar pelo melhor plano antes de tomar alguma em ação
  - Tomada de uma ação realmente ruim pode conduzir a desastre

382