Planejamento em Inteligência Artificial

Capítulo 4
Planejamento como busca no Espaço de Estados

Leliane Nunes de Barros

Motivação

- Planejamento é um problema de busca
  - *Busca em espaço de estados*
    - » Cada nó representa um estado do mundo
    - » Um plano é um caminho através do espaço de estados
  - *Busca em espaço de planos*
    - » Cada nó representa um plano parcial dado por um conjunto de operadores parcialmente instanciados e um conjunto de restrições de ordem
    - » Um plano é obtido adicionando-se cada mais e mais restrições, até obtermos um plano solução.

Planejamento como Busca

	Espaço de Estados	Espaço de Planos
Algoritmo	Planejamento Progressivo (busca para frente) (Planejamento Regressivo) (busca para trás)	POP *Partial-Order Planning*
Nós	Estados do Mundo	Planos Parciais
Arestas/ Transições	Ações Por exemplo, no mundo dos blocos: • move-A-from-B-to-C • move-B-from-A-to-Table • move-C-from-B-to-A • …	Refinamentos de Planos: • Step addition • Step reuse • Demotion • Promotion

Tópicos

- Planejamento como uma ***busca em espaço de estados***
  - » Planejamento Progressivo
  - » Planejamento Regressivo
  - » Lifting
  - » STRIPS
  - » Exemplo: O Mundo dos Blocos

Forward-search(O, s_0, g)

$s \leftarrow s_0$

$\pi \leftarrow$ the empty plan

loop

if s satisfies g then return π

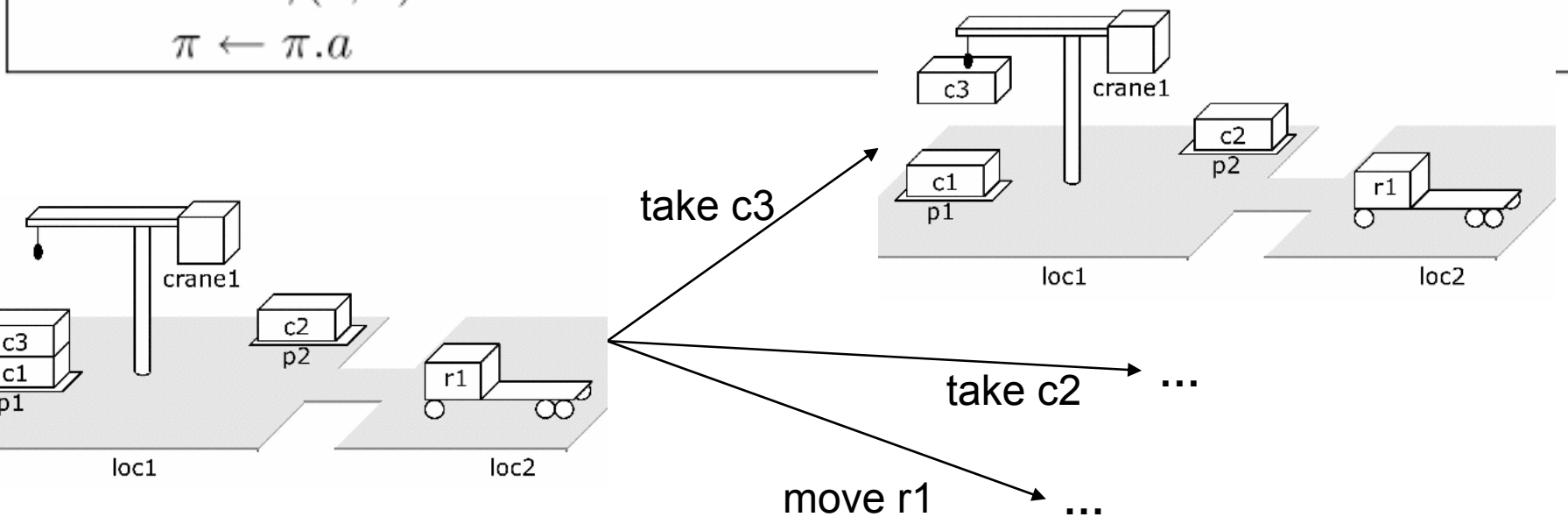
$E \leftarrow \{a | a$ is a ground instance an operator in O,
and $\text{precond}(a)$ is true in $s\}$

if $E = \emptyset$ then return failure

nondeterministically choose an action $a \in E$

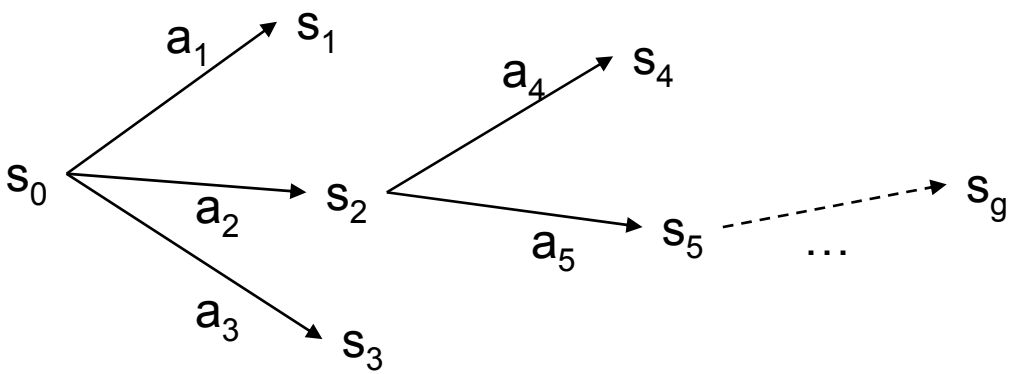
$s \leftarrow \gamma(s, a)$

$\pi \leftarrow \pi.a$



Planejamento Progressivo

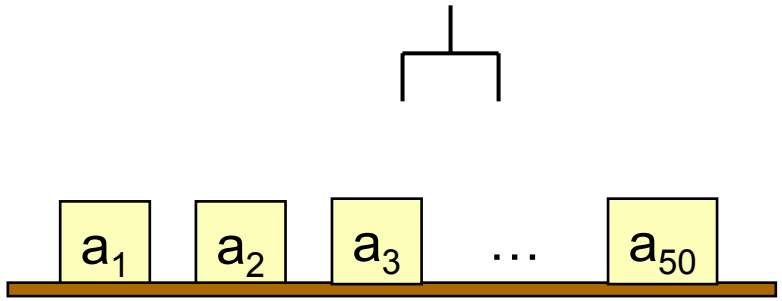
- Algumas implementações de busca para frente:
  - *breadth-first*
  - *best-first*
  - *depth-first*
  - *greedy*



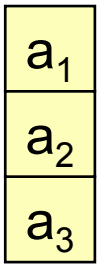
- Os algoritmos de busca *breadth-first* e *best-first* são corretos e completos
  - Porém, eles consomem muita memória: exponencial em função do tamanho da solução

- Na prática, é melhor usar uma busca *depth-first* ou *greedy*
  - Pior-caso: o uso de memória cresce linearmente em função do tamanho da solução
  - correto mas não completo
    - » como o planejamento clássico possui um número finito de estados, os caminhos não são infinitos mas podem entrar em loop ➔ é necessário evitar nós repetidos

Fator de ramificação do Planejamento Progressivo



Estado inicial



meta

- A busca para frente pode ter um fator de ramificação muito grande (veja exemplo)
- Porque isto é ruim:
  - podem gastar tempo tentando muitas ações irrelevantes
- É preciso construir boas funções heurísticas e/ou procedimento de poda.

Planejamento Regressivo

- No planejamento progressivo, começamos com o estado inicial e calculamos as transições de estados através da função de transição γ
  - $s' = \gamma(s,a)$
- No planejamento regressivo, começamos por um dos estados meta e calculamos a inversa da função de transição, γ^{-1}
  - Novo conjunto de sub-metas = $\gamma^{-1}(g,a)$

Transições inversas de estados

- O que significa $\gamma^{-1}(g,a)$?
- Primeiro precisamos definir *relevância*:
  - Uma ação a é relevante para uma meta g se
    - » a torna pelo menos um dos literais de g verdadeiro
      - $g \cap \text{effects}(a) \neq \emptyset$
    - » a não torna falso nenhum dos literais de g
      - $g^+ \cap \text{effects}^-(a) = \emptyset$
      - $g^- \cap \text{effects}^+(a) = \emptyset$
- Se a for relevante para g, então
  - $\gamma^{-1}(g,a) = (g - \text{effects(a)}) \cup \text{precond}(a)$

Backward-search(O, s_0, g)

$\pi \leftarrow$ the empty plan

loop

if s_0 satisfies g then return π

$A \leftarrow \{a | a$ is a ground instance of an operator in O

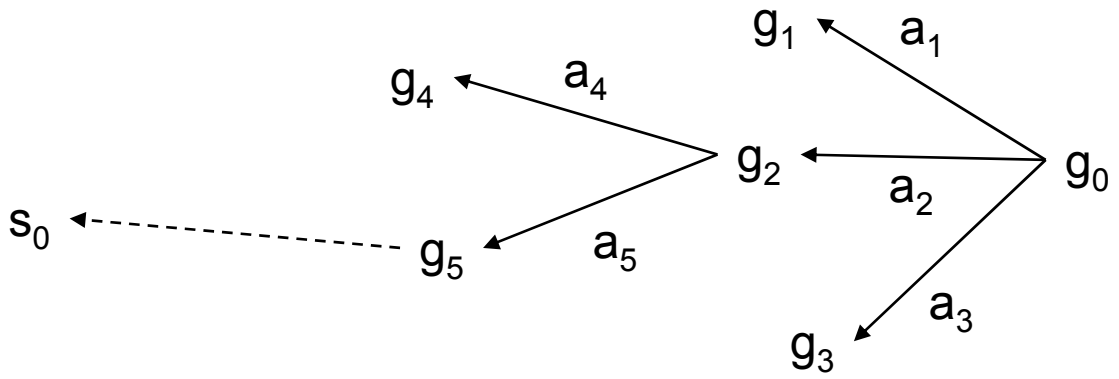
and $\gamma^{-1}(g, a)$ is defined$\}$

if $A = \emptyset$ then return failure

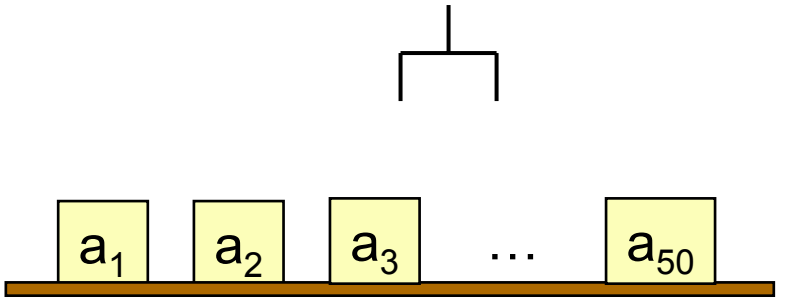
nondeterministically choose an action $a \in A$

$\pi \leftarrow a.\pi$

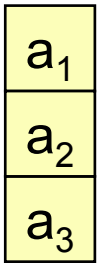
$g \leftarrow \gamma^{-1}(g, a)$



Eficiência do Planejamento Regressivo



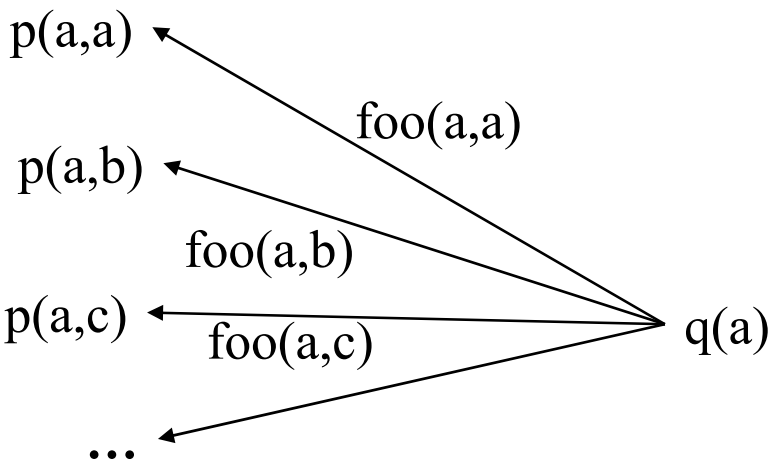
Estado inicial



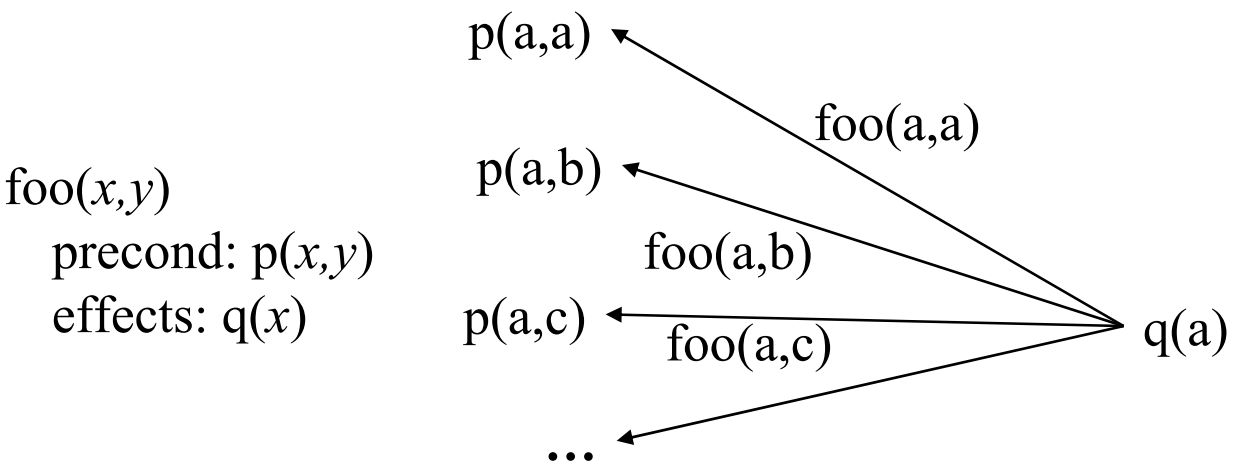
meta

- O fator de ramificação da busca para trás é pequena no exemplo
- Existem casos em que a ramificação pode ser muito grande
  - Muitas instâncias de operadores são avaliadas

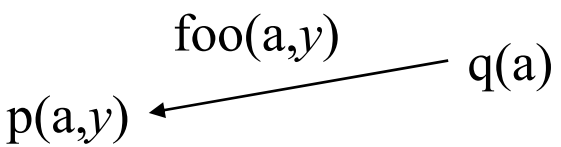
foo(*x*,*y*)
precond: p(*x*,*y*)
effects: q(*x*)



Lifting



- Podemos reduzir o fator de ramificação se nós instanciamos *parcialmente* os operadores
  - Isto é chamado de *lifting*



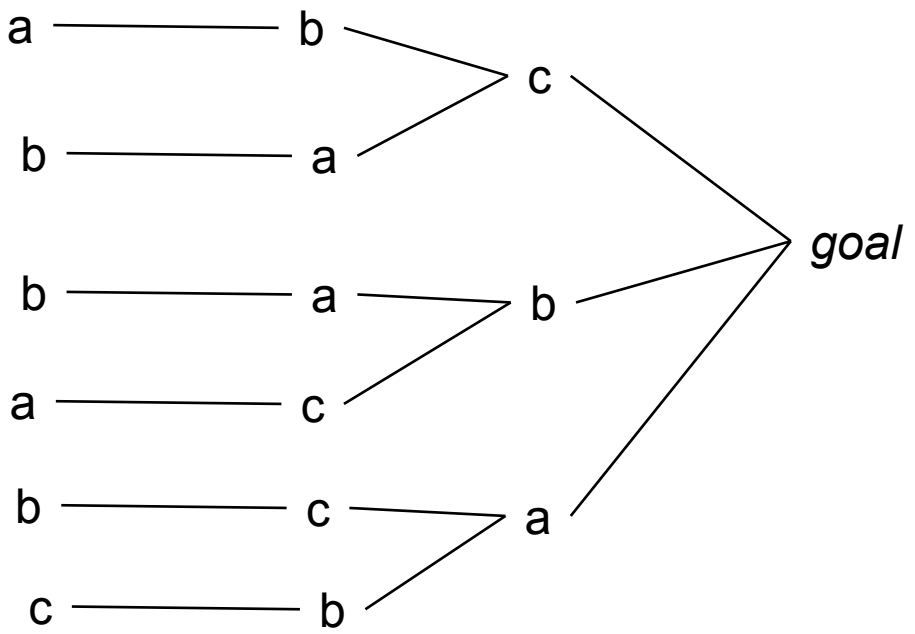
Busca para trás Lifted

- Mais complicado que o planejamento regressivo anterior
- Porém, tem um fator de ramificação muito menor

Lifted-backward-search(O, s_0, g)
  $\pi \leftarrow$ the empty plan
  loop
    if s_0 satisfies g then return π
    $A \leftarrow \{(o, \theta) | o$ is a standardization of an operator in O,
      θ is an mgu for an atom of g and an atom of $\text{effects}^+(o)$,
      and $\gamma^{-1}(\theta(g), \theta(o))$ is defined$\}$
    if $A = \emptyset$ then return failure
    nondeterministically choose a pair $(o, \theta) \in A$
    $\pi \leftarrow$ the concatenation of $\theta(o)$ and $\theta(\pi)$
    $g \leftarrow \gamma^{-1}(\theta(g), \theta(o))$

Problema: o espaço de busca é ainda muito grande

- A busca Lifted-backward-search gera um espaço de busca menor que Backward-search, porém este ainda pode ser muito grande
  - No pior caso é preciso examinar todas as ordenações possíveis antes de perceber que não há solução
  - Mais sobre isto no Capítulo 5 (Planejamento em Espaço de Planos)



Outras formas de reduzir a Busca

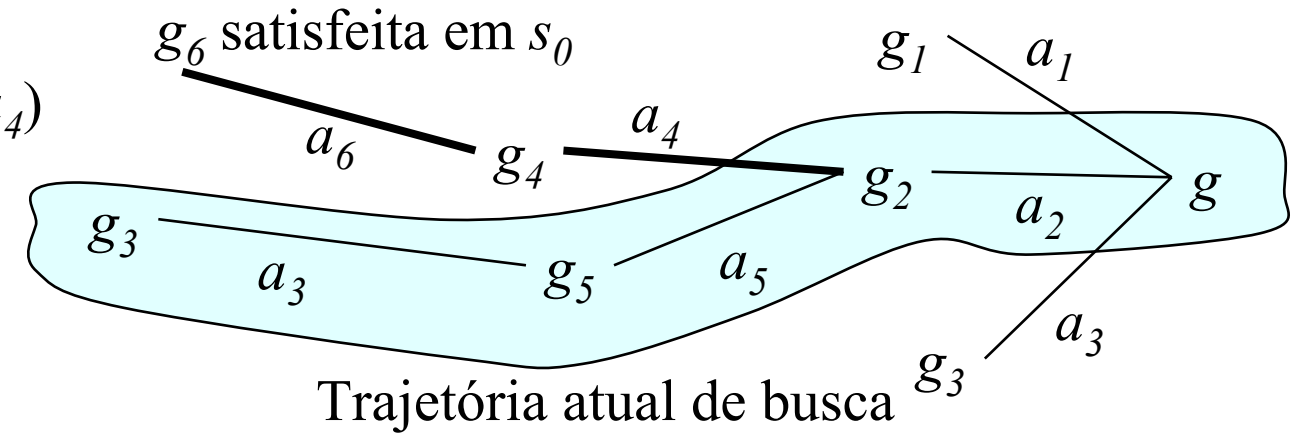
- *Estratégias de controle de busca*
  - Estratégias gerais serão tratadas na Parte III do livro
  - Aqui veremos dois exemplos de estratégias específicas:
    - » STRIPS
    - » Empilhamento de blocos “block-stacking”

STRIPS

- $\pi \leftarrow$ o plano vazio
- Fazer uma busca para trás modificada desde g
  - No lugar de $\gamma^{-1}(s,a)$, cada novo conjunto sub-metas é só precond(a)
  - Cada vez que você acha uma ação que é executável no estado atual, então STRIPS compromete a execução desse operador e não deixa fazer "backtracking" do compromisso
  - Repita até que todas as metas sejam satisfeitas

$\pi = \langle a_6, a_4 \rangle$

$s = \gamma(\gamma(s_0,a_6),a_4)$



STRIPS

Ground-STRIPS(O, s, g)
    $\pi \leftarrow$ the empty plan
    loop
        if s satisfies g then return π
        $A \leftarrow \{a \mid a$ is a ground instance of an operator in O,
                and a is relevant for $g\}$
        if $A = \emptyset$ then return failure
        nondeterministically choose any action $a \in A$
        $\pi' \leftarrow$ Ground-STRIPS($O, s,$ precond(a))
        if $\pi' =$ failure then return failure
        ;; *if we get here, then* π' *achieves* precond(a) *from* s
        $s \leftarrow \gamma(s, \pi')$
        ;; s *now satisfies* precond(a)
        $s \leftarrow \gamma(s, a)$
        $\pi \leftarrow \pi . \pi' . a$

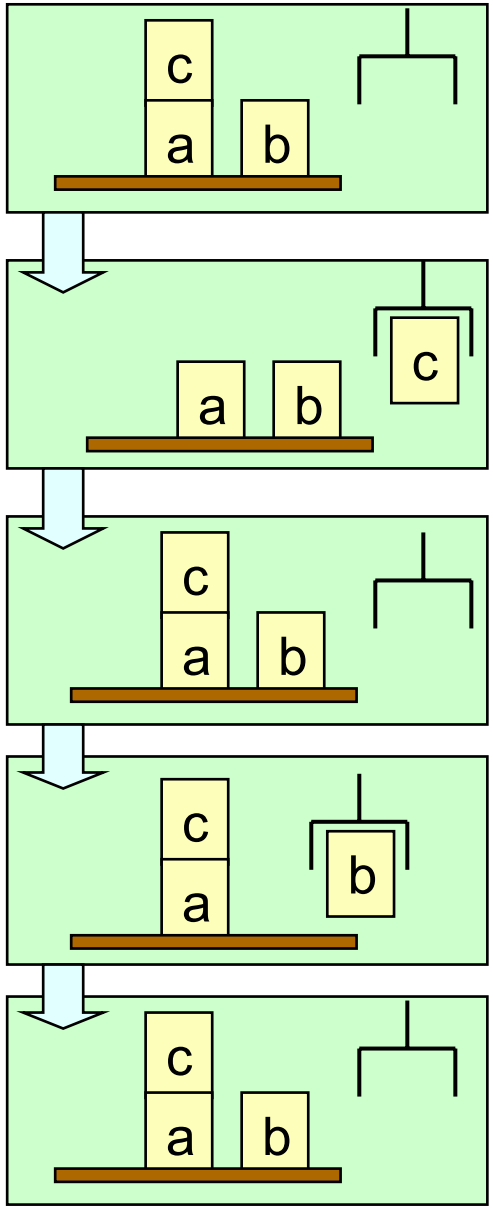
Mundo dos Blocos (revisão)

unstack(x,y)
Pre: on(x,y), clear(x), handempty
Eff: ~on(x,y), ~clear(x), ~handempty,
holding(x), clear(y)

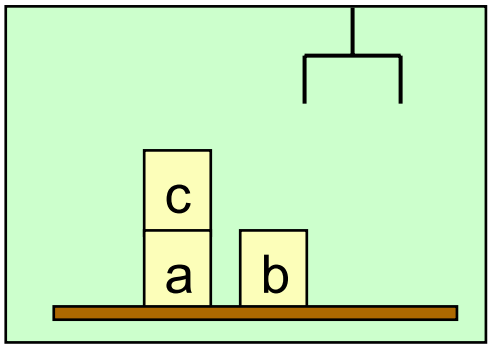
stack(x,y)
Pre: holding(x), clear(y)
Eff: ~holding(x), ~clear(y),
on(x,y), clear(x), handempty

pickup(x)
Pre: ontable(x), clear(x), handempty
Eff: ~ontable(x), ~clear(x), ~handempty, holding(x)

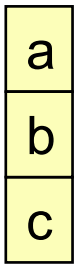
putdown(x)
Pre: holding(x)
Eff: ~holding(x), ontable(x), clear(?x), handempty



A Anomalia de Sussman



Estado inicial



meta

- Para este problema, STRIPS não consegue encontrar uma solução sem redundâncias

O Problema de Atribuição de Registros

- Formulação usando variáveis de estado:

  Estado Inicial: {valor(r1)=3, valor(r2)=5, valor(r3)=0}

  Meta: {valor(r1)=5, valor(r2)=3}

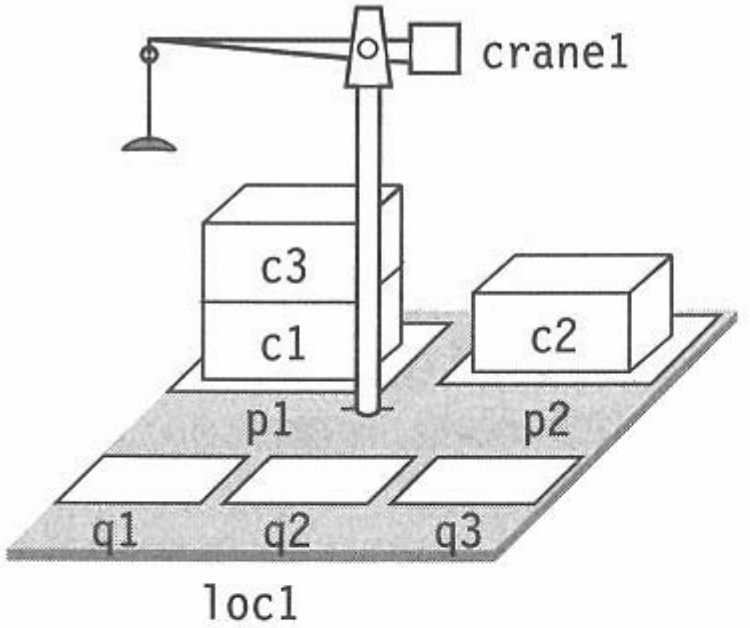
  Operador: atribuir(r,v,r',v')

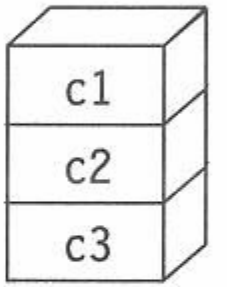
  precond: valor(r)=v, valor(r')=v'

  efeitos: valor(r)=v'

- STRIPS não consegue resolver este problema

Versão DWR da anomalia de Sussman





s_0={in(c3,p1), top(c3,p1), in(c1,p1), on(c3,c1)
on(c1,pallet), in(c2,p2), top(c2,p2),
on(c2,pallet), top(pallet,q1), top(pallet,q2)
top(pallet,q3), empty(crane1)}

g={on(c1,c2)
on(c2,c3)}

A DWR version of the Sussman anomaly.

Como solucionar isto?

- Várias formas:
  - Busca no Espaço de Planos, Grafos de Planejamento, Planejamento como Satisfazibilidade e uso de Técnicas de Satisfação de Restrições (Capítulo 5–8)

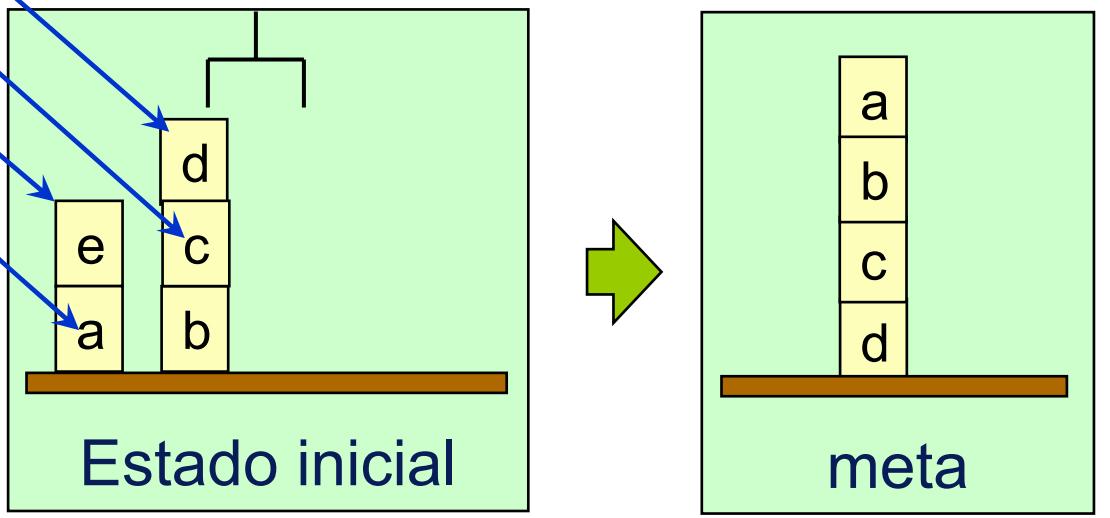
  - Ou ainda, usar busca no espaço de estados para frente ou para trás, com conhecimento *específico do domínio* para podar o espaço de busca
    - » Podemos resolver os dois problemas de forma fácil
    - » Exemplo: “block stacking” usando busca para frente

Conhecimento Específico do Domínio

- Um problema de planejamento do mundo dos blocos $P = (O,s_0,g)$ tem solução se s_0 e g satisfazem algumas condições de consistência simples
  - » g não deve envolver nenhum bloco não mencionado em s_0
  - » um bloco não pode estar sobre dois blocos ao mesmo tempo
  - » etc.
    - Podem ser checadas em tempo O(n log n)
- Se P tem uma solução, podemos facilmente construir uma solução de tamanho O($2m$), onde m é o número de blocos
  - Mover todos os blocos para a mesa e então construir pilhas de baixo para cima
    - » Isso pode ser feito em tempo O(n)
- Com conhecimento específico adicional do domínio podemos melhorar ainda mais…

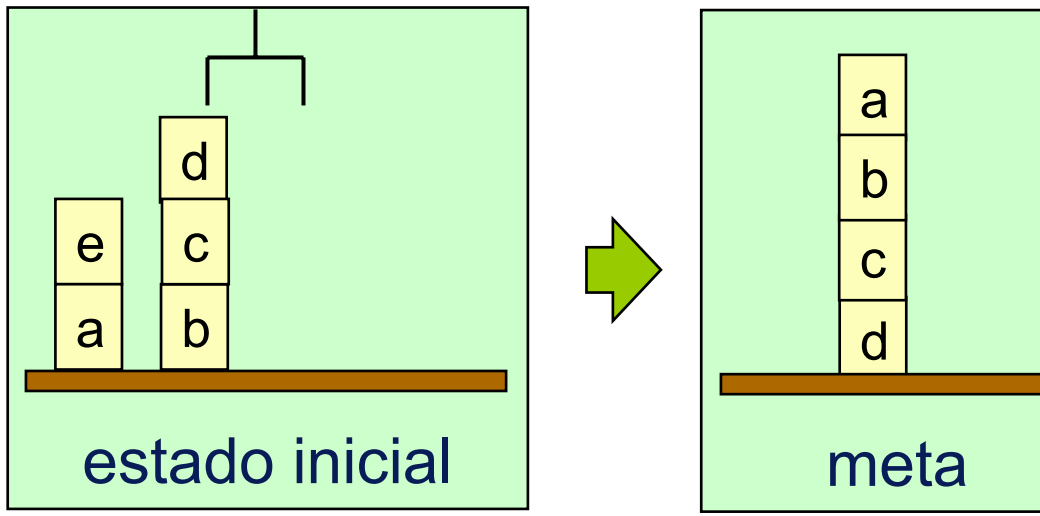
Conhecimento Específico Adicional do Domínio

- Um bloco x precisa ser movimentado se alguma das seguintes condições for verdade:
  - s contém ontable(x) e g contém on(x,y)
  - s contém on(x,y) e g contém ontable(x)
  - s contém on(x,y) e g contém on(x,z) para algum $y \neq z$
  - s contém on(x,y) e y precisa ser movimentado



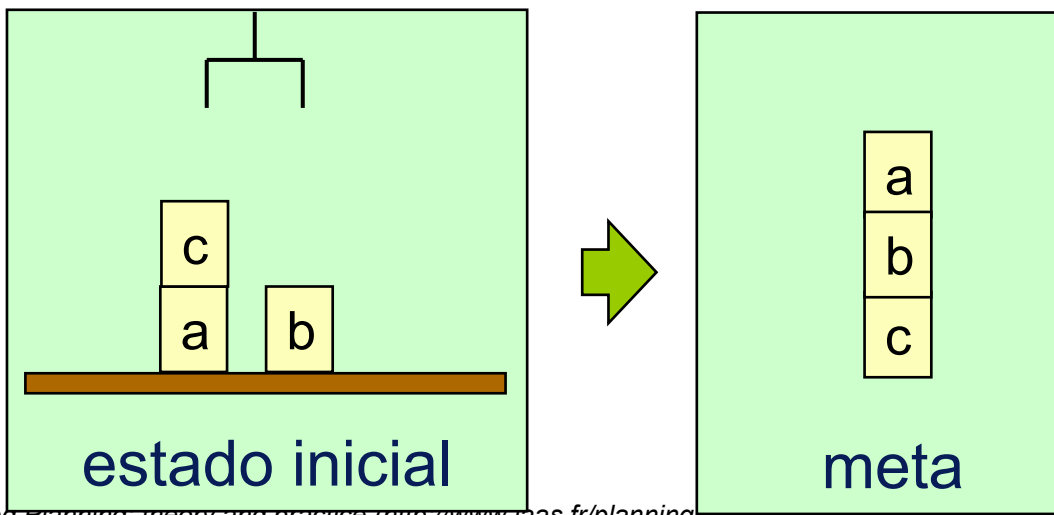
block-stacking: Algoritmo Específico do Domínio

loop
  if there is a clear block *x* such that
    x needs to be moved **and**
    x can be moved to a place where it won't need to be moved
  then move *x* to that place
  else if there is a clear block *x* such that
    x needs to be moved
  then move *x* to the table
  else if the goal is satisfied
  then return the plan
  else return failure
repeat



Solução da Anomalia de Sussman

loop
  if there is a clear block *x* such that
    x needs to be moved **and**
    x can be moved to a place where it won't need to be moved
  then move *x* to that place
  else if there is a clear block *x* such that
    x needs to be moved
  then move *x* to the table
  else if the goal is satisfied
  then return the plan
  else return failure
repeat



Propriedades

- O algoritmo ***block-stacking*** é:
  - correto, completo, com garantia de terminação
  - Executa em tempo $O(n^3)$
    - » Pode ser modificado para executar em tempo $O(n)$
  - Em geral, acha soluções ótimas (mais curtas)
  - Porém, algumas vezes somente perto do ótimo (Exercício 4.22 no livro)
    - » Lembre que PLAN LENGTH é NP-completo

Algoritmo de empilhamento de *containers*

Stack-containers(O, s_0, g):
    if g does not satisfy the consistency conditions then
        return failure ;; *the planning problem is unsolvable*
    $\pi \leftarrow$ the empty plan
    $s \leftarrow s_0$
    loop
        if s satisfies g then return π
        if there are containers b and c at the tops of their piles such that
                position(c, s) is consistent with g and on(b, c) $\in g$
        then
            append actions to π that move b to c
            $s \leftarrow$ the result of applying these actions to s
            ;; *we will never need to move b again*
        else if there is a container b at the top of its pile
                such that position(b, s) is inconsistent with g
                and there is no c such that on(b, c) $\in g$

Algoritmo de empilhamento de *containers* (continuação)

then
    append actions to π that move b to an empty auxiliary pile
    $s \leftarrow$ the result of applying these actions to s
    ;; we will never need to move b again
else
    nondeterministically choose any container c such that c is
        at the top of a pile and position(c, s) is inconsistent with g
    append actions to π that move c to an empty auxiliary pallet
    $s \leftarrow$ the result of applying these actions to s