

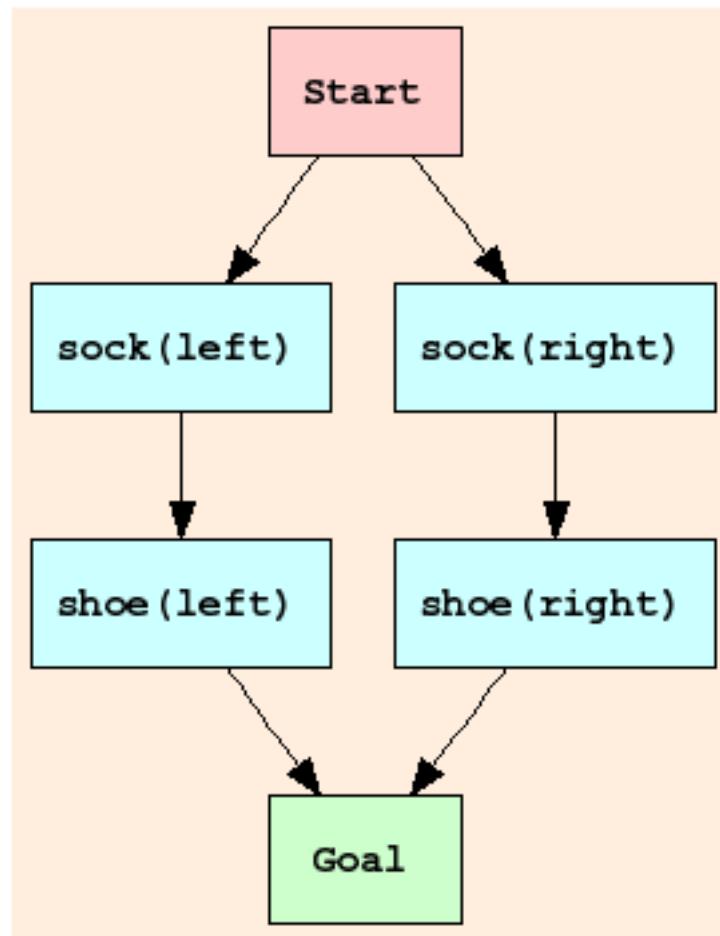
Planeamento Automático

Capítulo 11

O que é o planeamento?

- Um sistema de planeamento consiste num algoritmo que resolve problemas que são representados por estados e ações em Lógica Proposicional ou Lógica de 1^a Ordem
 - **Esta representação torna possível a criação de heurísticas eficientes e o desenvolvimento de algoritmos poderosos e flexíveis**
 - À semelhança do que acontece com os problemas de satisfação de restrições!

Planeamento: exemplo



Agentes de Planeamento

- Capítulo 3: agentes baseados em procura
- *Capítulo 7: agentes baseados em lógica*
- Neste capítulo: problemas de planeamento (clássicos) complexos
 - **Problemas clássicos: ambientes completamente observáveis, determinísticos, estáticos (mudança ocorre apenas quando o agente atua) e discretos (em tempo, ação, objetos e efeitos)**
 - **Problemas não clássicos: parcialmente observáveis e estocásticos**

Planeamento

- **1. Definição de planeamento clássico**
- **2. Algoritmos para planeamento clássico**
- **3. Heurísticas para planeamento**
- *Grafos de planeamento*
- *Planeamento de ordem parcial*

O que é o Planeamento

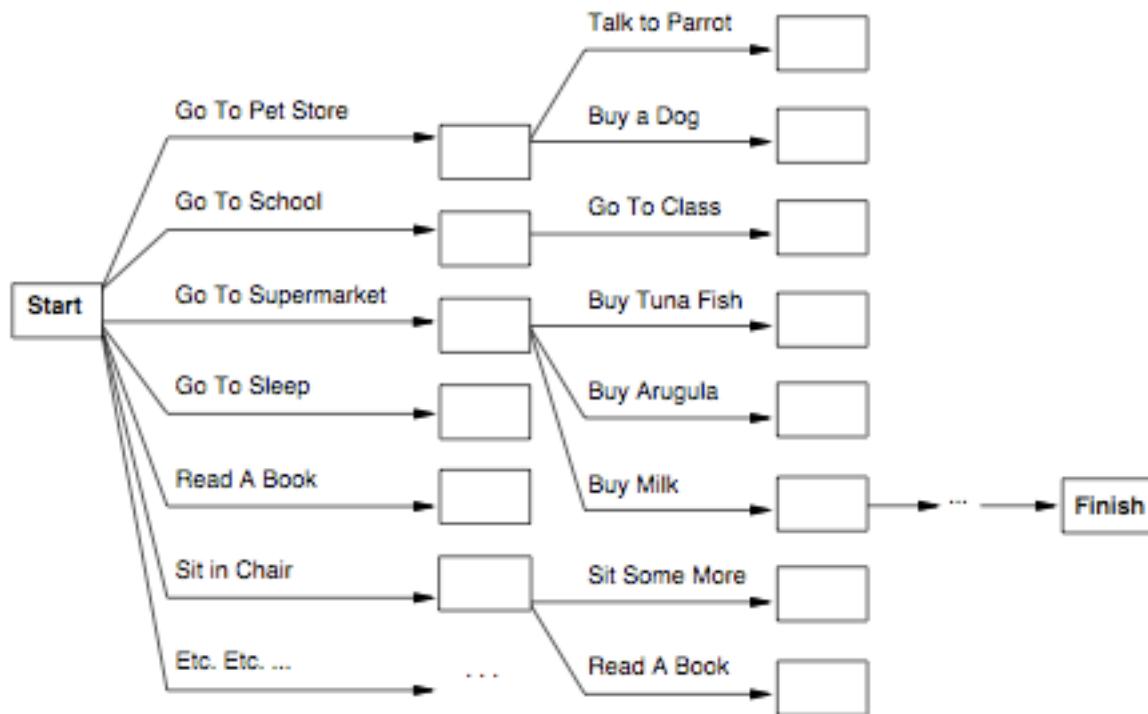
- Geração de sequências de ações para realizar tarefas e atingir objetivos
 - **Estados, ações e objetivos**
- Procura de soluções num espaço abstrato de planos
- Aplicações práticas
 - **Desenho e manufaturação**
 - **Operações militares**
 - **Jogos**
 - **Exploração de espaço**

Agente Planeador

- Recebe uma percepção e devolve uma ação
 - **Na prática determina um plano e vai devolvendo uma ação do plano de cada vez**
- Tem um contador de tempo
 - **Cada ação corresponde a um instante de tempo**
- Comunica com uma base de conhecimento
 - **Para aceder ao estado atual**
 - **Para ter acesso à descrição das ações**
 - **Para formular o objetivo em cada instante de tempo**
 - **Para atualizar o estado em função de cada ação**

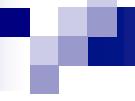
Procura vs. Planeamento

- Algoritmo de procura para comprar leite e pão é claramente ineficiente...



Dificuldade dos problemas do mundo real

- Considere-se um agente que resolve problemas usando um método de procura ...
 - **Que ações são relevantes?**
 - Procura tradicional (exaustiva)
 - **Compra de um livro com 10 dígitos de ISBN → 10 biliões de nós!!!**
 - Procura para trás (usada em planeamento!)
 - **Se o objetivo é Ter(ISBN0137903952), e considerando que Comprar(x) → Ter(x) então é necessário Comprar(ISBN0137903952)**
 - **O que são boas funções heurísticas?**
 - Boas estimativas do custo do estado?
 - Dependentes ou independentes do problema?
 - **Como decompor um problema?**
 - A maioria dos problemas do mundo real são *quase* decomponíveis
 - Decomposição reduz complexidade do problema



Linguagem de planeamento

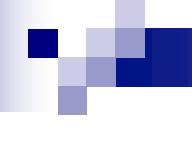
- O que é uma boa linguagem?
 - **Uma linguagem suficientemente expressiva para descrever uma grande variedade de problemas**
 - **Uma linguagem suficientemente restritiva para permitir que algoritmos eficientes operem sobre ela**
 - **Um algoritmo de planeamento deve ser capaz de explorar a estrutura lógica do problema**

PDDL: Planning Domain Definition Language

- Inspirada em *Lisp*
- Representação de estados
- Representação de ações
- Representação de estado inicial e objetivo

Características da PDDL

- Representação de estados
 - **Decomposição do mundo em condições lógicas e representação de um estado como uma conjunção de literais positivos**
 - Literais proposicionais:
 - *Pobre* \wedge *Desconhecido*
 - Literais sem variáveis e sem funções:
 - *Em(Avião1, Melbourne)* \wedge *Em(Avião2, Sydney)*
 - **Assumir que o mundo é fechado (*closed-world assumption*)**
 - as condições que não são mencionadas são falsas
 - **Assumir que nomes são únicos (*unique names assumption*)**
 - dois nomes distintos correspondem a dois objetos distintos.



Características da PDDL

- Ações descritas em termos de pré-condições e efeitos
 - **Pré-condições representam o que se tem de verificar para a ação poder ser executada**
 - **Efeitos representam os efeitos diretos da ação**
 - **Ao contrário dos estados, podemos usar variáveis na representação das ações**
- Esquemas de Ações ou Operadores
 - **Um operador pode corresponder a várias instanciações de ações**

Características da PDDL

- Esquema de uma Ação
 - **Nome da ação e lista de parâmetros**
 - **Pré-condição (literais sem funções)**
 - **Efeito (literais sem funções)**
 - o que é verdadeiro (P)
 - o que é falso ($\neg P$)
 - conjunção de literais pode ser separado em
 - **lista de adições – Add List**
 - **lista de remoções – Delete List**
 - *importante:* qualquer variável no efeito deve aparecer também na pré-condição

Características da PDDL

Ação(Voar(a,de,para),

**PRÉ-CONDIÇÃO: Em(a,de) \wedge Avião(a) \wedge
Aeroporto(de) \wedge Aeroporto(para)**

EFEITO: \neg Em(a,de) \wedge Em(a,para)

)

Instanciação

Ação(Voar(TAP_{15} ,Lisboa,Paris),

**PRÉ-CONDIÇÃO: Em(TAP_{15} ,Lisboa) \wedge
Avião(TAP_{15}) \wedge Aeroporto(Lisboa) \wedge
Aeroporto(Paris)**

EFEITO: \neg Em(TAP_{15} ,Lisboa) \wedge Em(TAP_{15} ,Paris)

)

Características da PDDL

- Representação de estado inicial e objetivo
 - **Estado inicial: conjunção de literais sem variáveis**
 - **Objetivo representado simplesmente com conjunto de literais**
 - Conjunção de literais positivos (podemos ter variáveis)
 - ***Rico* \wedge *Famoso***
 - Variáveis são tratadas como se estivessem quantificadas existencialmente
 - ***Em(a,Paris)* \wedge *Avião(a)***
 - **Este objetivo corresponde a ter qualquer avião em Paris**

exemplo: pneu sobresselente

$\text{Início}(\text{Em}(\text{Furado}, \text{Eixo}) \wedge \text{Em}(\text{Sobresselente}, \text{Bagageira}) \wedge \text{Pneu}(\text{Furado}) \wedge \text{Pneu}(\text{Sobresselente}))$
 $\text{objetivo}(\text{Em}(\text{Sobresselente}, \text{Eixo}))$

$Ação(\text{Remove}(obj, loc))$

PRÉ-CONDIÇÃO: $\text{Em}(obj, loc)$

FEITO: $\neg \text{Em}(obj, loc) \wedge \text{Em}(obj, Chão)$

$Ação(\text{Colocar}(p, Eixo))$

PRÉ-CONDIÇÃO: $\text{Pneu}(p) \wedge \text{Em}(p, Chão) \wedge \neg \text{Em}(\text{Furado}, \text{Eixo})$

FEITO: $\text{Em}(p, Eixo) \wedge \neg \text{Em}(p, Chão)$

$Ação(\text{DeixarDuranteANoite})$

PRÉ-CONDIÇÃO:

FEITO: $\neg \text{Em}(\text{Sobresselente}, \text{Chão}) \wedge \neg \text{Em}(\text{Sobresselente}, \text{Eixo}) \wedge \neg \text{Em}(\text{Sobresselente}, \text{Bagageira}) \wedge \neg \text{Em}(\text{Furado}, \text{Chão}) \wedge \neg \text{Em}(\text{Furado}, \text{Eixo}) \wedge \neg \text{Em}(\text{Furado}, \text{Bagageira})$

Exemplo: pneu sobresselente

$Init(Tire(Flat) \wedge Tire(Spare) \wedge At(Flat, Axle) \wedge At(Spare, Trunk))$

$Goal(At(Spare, Axle))$

$Action(Remove(obj, loc),$

PRECOND: $At(obj, loc)$

EFFECT: $\neg At(obj, loc) \wedge At(obj, Ground)$)

$Action(PutOn(t, Axle),$

PRECOND: $Tire(t) \wedge At(t, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Spare, Axle)$

EFFECT: $\neg At(t, Ground) \wedge At(t, Axle)$)

$Action(LeaveOvernight,$

PRECOND:

EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
 $\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Flat, Trunk)$)

Figure 11.2 The simple spare tire problem.

Características da PDDL

- Função ações(s)
 - **Quais as ações que podem ser executadas num dado estado s?**
 - **Uma ação é aplicável em qualquer estado que satisfaça a pré-condição**
 - **Aplicabilidade de uma ação pode envolver uma substituição θ para as variáveis na pré-condição**

$Em(A1,JFK) \wedge Em(A2,SFO) \wedge Avião(A1) \wedge Avião(A2) \wedge$
 $Aeroporto(JFK) \wedge Aeroporto(SFO)$

Satisfaz : $Em(a,de) \wedge Avião(a) \wedge Aeroporto(de) \wedge Aeroporto(para)$

Por exemplo com $\theta = \{a/A1, de/JFK, para/SFO\}$

Logo a ação $Voar(A1,JFK,SFO)$ é aplicável.

Características da PDDL

- Função resultado(s,a)
 - **O resultado de executar uma ação a num estado s é um estado s'**
 - **s' = Result(a,s) = (s – Del(a) U Add(a))**
 - **s' é o mesmo que s exceto**
 - Qualquer literal positivo P no efeito de a é adicionado a s'
 - Qualquer P correspondente a um literal negativo no efeito ($\neg P$) é removido de s'
 - **Em PDDL assume-se o seguinte: (para representar ausência de mudança)**
Qualquer literal que NÃO esteja no efeito permanece inalterado

Características da PDDL

- O problema é resolvido quando encontramos uma sequência de ações que acaba num estado s , correspondente a um estado objetivo
- Teste-objetivo(s)
 - **Verifica se a pré-condição correspondente ao objetivo definido é consequência lógica de s**
 - $s \models \text{obj}$
 - **Por outras palavras, verifica se s verifica as condições do objetivo**

exemplo: transporte aéreo de carga

$\text{Início}(\text{Em}(C1, \text{SFO}) \wedge \text{Em}(C2, \text{JFK}) \wedge \text{Em}(A1, \text{SFO}) \wedge \text{Em}(A2, \text{JFK}) \wedge \text{Carga}(C1) \wedge \text{Carga}(C2) \wedge \text{Avião}(A1) \wedge \text{Avião}(A2) \wedge \text{Aeroporto}(\text{JFK}) \wedge \text{Aeroporto}(\text{SFO}))$
 $\text{objetivo}(\text{Em}(C1, \text{JFK}) \wedge \text{Em}(C2, \text{SFO}))$

$ação(\text{Carregar}(c, a, l))$

PRÉ-CONDIÇÃO: $\text{Em}(c, l) \wedge \text{Em}(a, l) \wedge \text{Carga}(c) \wedge \text{Avião}(a) \wedge \text{Aeroporto}(l)$

FEITO: $\neg \text{Em}(c, l) \wedge \text{Dentro}(c, a)$

$ação(\text{Descarregar}(c, a, l))$

PRÉ-CONDIÇÃO: $\text{Dentro}(c, a) \wedge \text{Em}(a, l) \wedge \text{Carga}(c) \wedge \text{Avião}(a) \wedge \text{Aeroporto}(l)$

FEITO: $\text{Em}(c, l) \wedge \neg \text{Dentro}(c, a)$

$ação(\text{Voar}(a, de, para))$

PRÉ-CONDIÇÃO: $\text{Em}(a, de) \wedge \text{Avião}(a) \wedge \text{Aeroporto}(de) \wedge \text{Aeroporto}(para)$

FEITO: $\neg \text{Em}(a, de) \wedge \text{Em}(a, para)$

[Possível solução:

$\text{Carregar}(C1, A1, \text{SFO}), \text{Voar}(A1, \text{SFO}, \text{JFK}), \text{Descarregar}(C1, A1, \text{JFK}),$
 $\text{Carregar}(C2, A2, \text{JFK}), \text{Voar}(A2, \text{JFK}, \text{SFO}), \text{Descarregar}(C2, A2, \text{SFO})]$

Exemplo: transporte de carga

$Init(At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$
 $\wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
 $\wedge Airport(JFK) \wedge Airport(SFO))$

$Goal(At(C_1, JFK) \wedge At(C_2, SFO))$

$Action(Load(c, p, a),$

PRECOND: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$

EFFECT: $\neg At(c, a) \wedge In(c, p))$

$Action(Unload(c, p, a),$

PRECOND: $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$

EFFECT: $At(c, a) \wedge \neg In(c, p))$

$Action(Fly(p, from, to),$

PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

EFFECT: $\neg At(p, from) \wedge At(p, to))$

$[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),$
 $Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)]$

Exemplo: mundo dos blocos

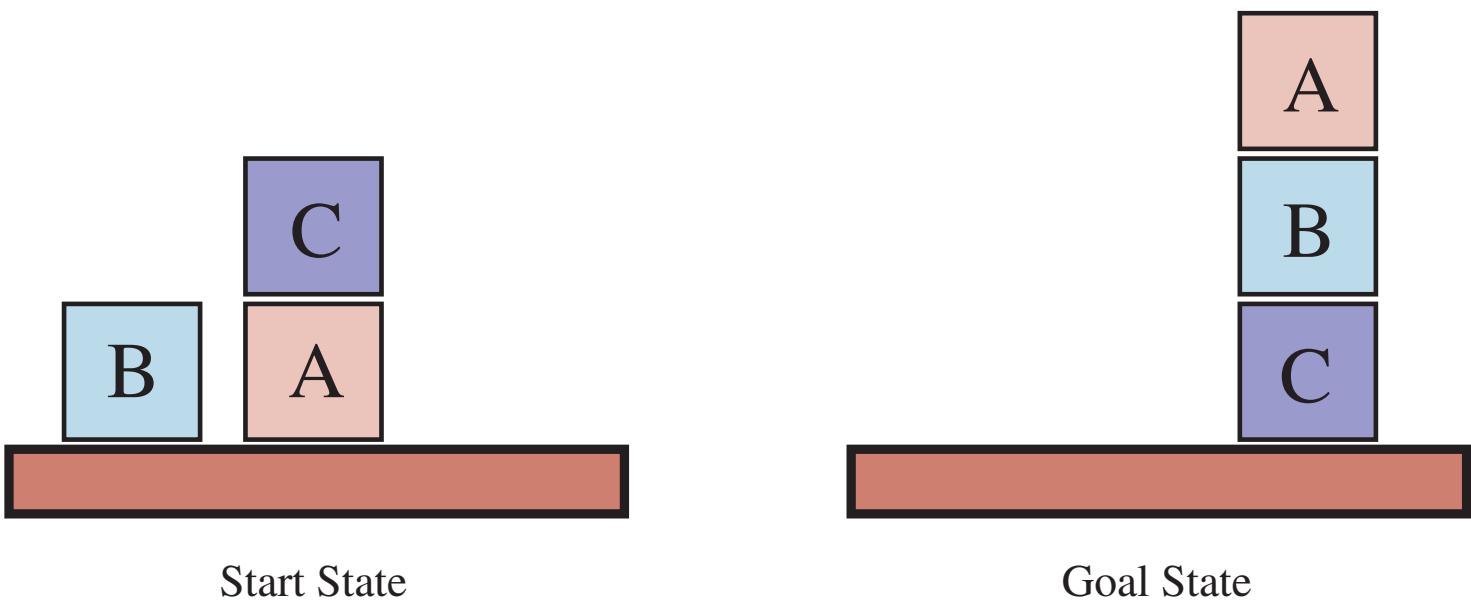


Figure 11.3 Diagram of the blocks-world problem in Figure 11.4.

exemplo: mundo dos blocos

Início(On(A, Table) \wedge On(B, Table) \wedge On(C, A) \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(B) \wedge Clear(C))

objetivo(On(A,B) \wedge On(B,C))

ação(Move(b,x,y))

PRÉ-CONDIÇÃO: *On(b,x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge (b \neq x) \wedge (b \neq y) \wedge (x \neq y)*

EFEITO: *On(b,y) \wedge Clear(x) \wedge \neg On(b,x) \wedge \neg Clear(y))*

ação(MoveToTable(b,x))

PRÉ-CONDIÇÃO: *On(b,x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x)*

EFEITO: *On(b,Table) \wedge Clear(x) \wedge \neg On(b,x))*

Exemplo: mundo dos blocos

$\text{Init}(\text{On}(A, \text{Table}) \wedge \text{On}(B, \text{Table}) \wedge \text{On}(C, A)$
 $\wedge \text{Block}(A) \wedge \text{Block}(B) \wedge \text{Block}(C) \wedge \text{Clear}(B) \wedge \text{Clear}(C) \wedge \text{Clear}(\text{Table}))$

$\text{Goal}(\text{On}(A, B) \wedge \text{On}(B, C))$

$\text{Action}(\text{Move}(b, x, y),$
PRECOND: $\text{On}(b, x) \wedge \text{Clear}(b) \wedge \text{Clear}(y) \wedge \text{Block}(b) \wedge \text{Block}(y) \wedge$
 $(b \neq x) \wedge (b \neq y) \wedge (x \neq y),$
EFFECT: $\text{On}(b, y) \wedge \text{Clear}(x) \wedge \neg \text{On}(b, x) \wedge \neg \text{Clear}(y))$

$\text{Action}(\text{MoveToTable}(b, x),$
PRECOND: $\text{On}(b, x) \wedge \text{Clear}(b) \wedge \text{Block}(b) \wedge \text{Block}(x),$
EFFECT: $\text{On}(b, \text{Table}) \wedge \text{Clear}(x) \wedge \neg \text{On}(b, x))$

Figure 11.4 A planning problem in the blocks world: building a three-block tower. One solution is the sequence $[\text{MoveToTable}(C, A), \text{Move}(B, \text{Table}, C), \text{Move}(A, \text{Table}, B)]$.

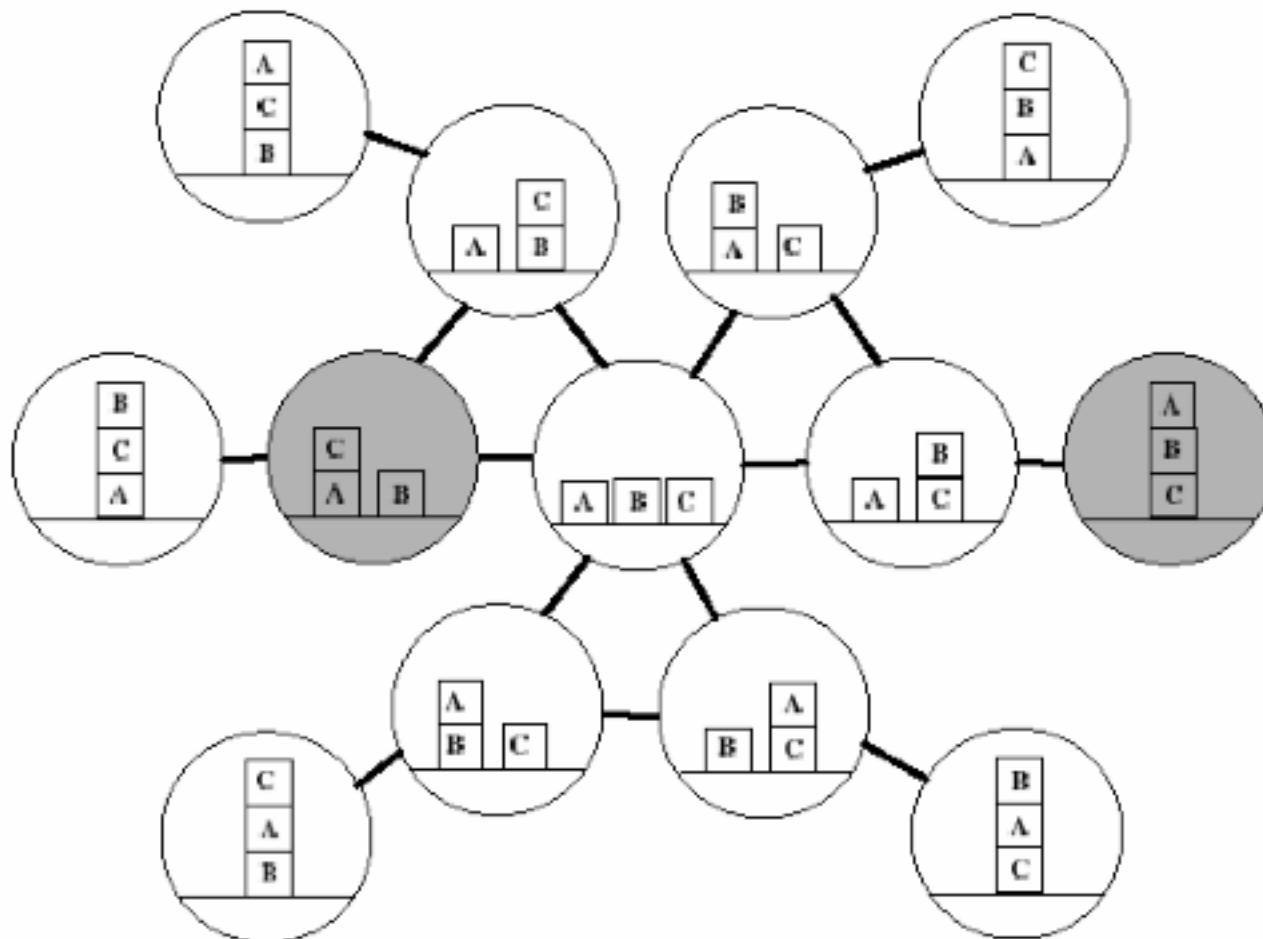
Planeamento

- 1. Definição de planeamento clássico
- 2. Algoritmos para planeamento clássico
 - Procura progressiva
 - Procura regressiva
- 3. Heurísticas para planeamento
- *Grafos de planeamento*
- *Planeamento de ordem parcial*

Procura em planeamento

- Algoritmos de planeamento usam procura em espaço de estados
 - **Procura tradicional:** nó = estado do mundo específico/concreto
 - **Planeamento:** nó = plano parcial
- Resolução de um problema de planeamento evolui a partir de planos vagos/incompletos para planos completos/correctos

Exemplo: espaço de estados para mundo dos blocos



Procura em espaço de estados

- Pode ser efetuada para a frente ou para trás
 - **3^a via: algoritmos de inferência lógica**
- Heurísticas usadas consideram sub-objetivos e relaxam o problema original

Planeamento com procura em espaço de estados

- Possibilidade de fazer procura progressiva ou regressiva
- Planeadores progressivos
 - **Procura progressiva em espaço de estados**
 - **Considerar o efeito de todas as ações possíveis num dado estado**
- Planeadores regressivos
 - **Procura regressiva em espaço de estados**
 - **Para alcançar um objetivo, considerar o que tem de ser verdadeiro no estado anterior**

Progressão e regressão

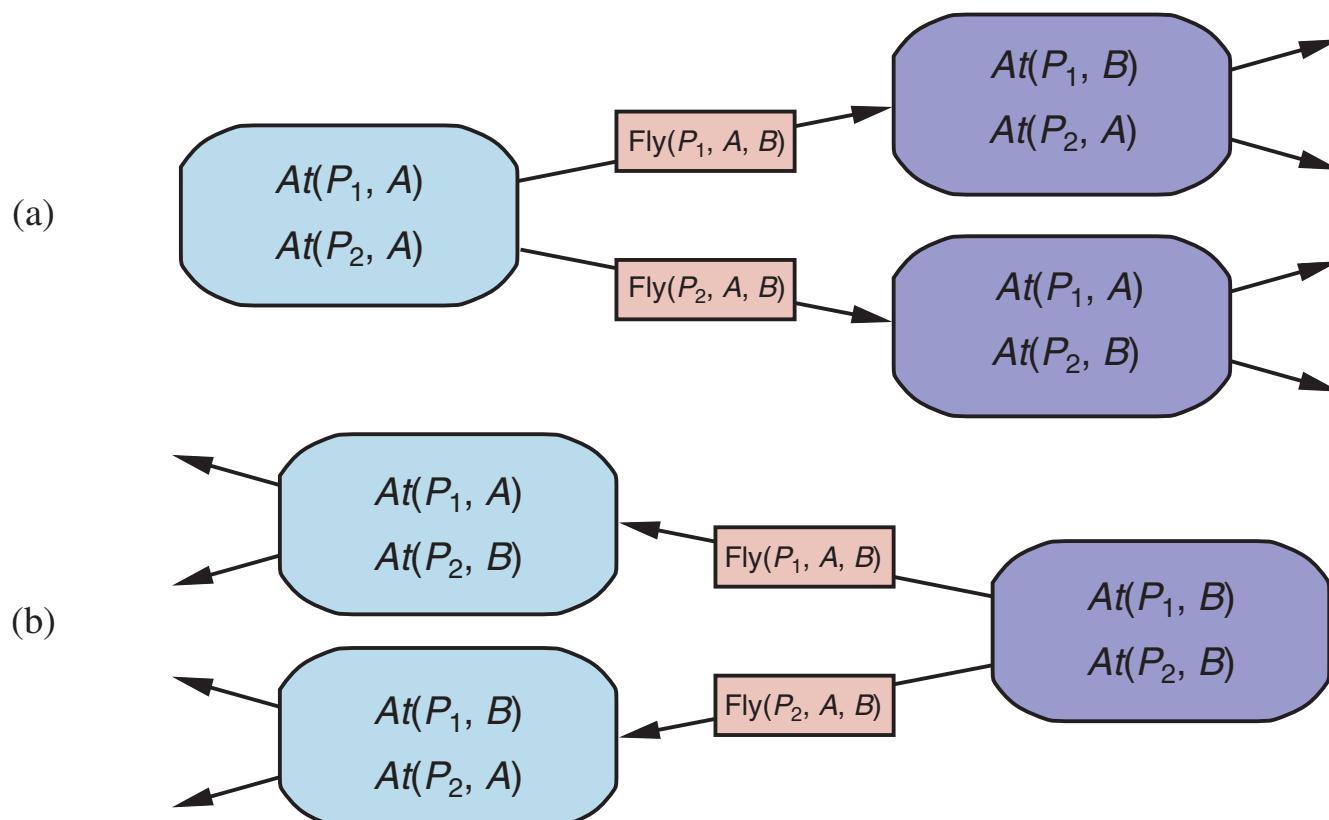


Figure 11.5 Two approaches to searching for a plan. (a) Forward (progression) search through the space of ground states, starting in the initial state and using the problem's actions to search forward for a member of the set of goal states. (b) Backward (regression) search through state descriptions, starting at the goal and using the inverse of the actions to search backward for the initial state.

Procura Progressiva

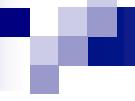
- Procura Progressiva (forward state-space search)
- Formulação como um problema de procura progressiva num espaço de estados:
 - **Estado inicial = estado inicial do problema de planeamento**
 - Literais que não aparecem são falsos
 - **Ações = aquelas cujas pré-condições são satisfeitas**
 - Adicionar efeitos positivos, remover efeitos negativos
 - **Teste objetivo = se o estado verifica o objetivo**
 - **Custo de transição = cada ação tem custo 1**

Procura Progressiva

- Não há funções ... logo qualquer procura em grafo completa é uma algoritmo de planeamento completo
- No entanto este tipo de procura sofre com:
 - **Ações irrelevantes**
 - Ex. Objetivo Ter(AIMA)
 - Ações: Comprar(livro)
 - **Compra de um livro com 10 dígitos de ISBN → 10 biliões de nós!!!**
 - **Fatores de ramificação muito elevados**
 - Eficiência muito dependente de uma boa heurística
 - **Felizmente existem boas heurísticas independentes do domínio**

Procura Regressiva

- Procura Regressiva (Backward relevant-states search)
 - **Procura trabalha com descrições de conjuntos de estados**
 - Ex. $\neg \text{Pobre} \wedge \text{Famoso}$
 - Representa os estados em que Pobre é Falso, Famoso é Verdade, e qualquer outra proposição pode ser verdadeira ou falsa
 - **Começamos do objetivo**
 - **Aplicamos as ações de trás para a frente**
 - **Até encontrarmos uma sequência de ações que nos leve até ao estado inicial**
- Apenas considera ações relevantes para o objetivo
 - **Se o objetivo é Ter(ISBN0137903952), e considerando que Comprar(x) → Ter(x) então é necessário Comprar(ISBN0137903952)**



Procura Regressiva

- Como determinar as ações relevantes?
 - **Devem contribuir para o objetivo**
 - Pelo menos um dos efeitos (positivo ou negativo) deve unificar com uma das condições do objetivo
 - Não pode ter nenhum efeito que negue uma condição do objetivo
 - **Porque se assim fosse, não poderia ser a última ação de uma solução**

Procura Regressiva

- Formalmente
 - **Considerem**
 - descrição de um objetivo g que contem um literal g_i
 - Ação A normalizada de modo a produzir A'
 - **Se**
 - A' tem um efeito e'_j tal que $\text{Unify}(g_i, e'_j) = \theta$
 - $a' = \text{subst}(\theta, A')$
 - não existe nenhum efeito em a' que é a negação de um literal de g
 - **Então**
 - a' é uma **ação relevante** para g

Procura Regressiva

- Exemplo

objetivo: Ter(0136042597)
ação(Comprar(i)),

Pré-condição:ISBN(i),
Efeito:Ter(i))

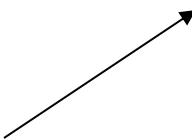
1) Normalização
ação (Comprar(i_1),
Pré-condição:ISBN(i_1),
Efeito:Ter(i_1)

2) Unificação

$$\theta = \{i_1/0136042597\}$$

3) Substituição

$a' = ação(Comprar(0136042597),$
Pré-condição:ISBN(0136042597)
Efeito:Ter(0136042597)



Normalização é necessária para considerarmos ações com variáveis não especificadas e podermos usar a mesma variável/ação mais que uma vez.
Ex: comprar dois livros quaisquer

Procura Regressiva

- Como obter os predecessores de um estado objetivo?
 - **Dado um objetivo g**
 - **Dada uma ação a que é relevante (para alcançar objetivos) e consistente (não invalida objetivos já alcançados)**

$$g' = (g - \text{Add}(a) \cup \text{Precond}(a))$$

- **Quaisquer efeitos positivos de a que aparecem em g são removidos**
- **Cada pré-condição de a é adicionada ao objetivo g', a não ser que já lá esteja**
- **Reparem que Del(a) não é utilizado**
 - Sabemos que os literais em Del(a) não são verdade depois da ação a ser executada
 - Mas não sabemos se são verdade ou não antes de ser executada

Procura Regressiva

- Exemplo

Estado objetivo = $Em(C1, B) \wedge Em(C2, B) \wedge \dots \wedge Em(C20, B)$

Ação que tem o primeiro objetivo como efeito: $Descarregar(C1, a, B)$

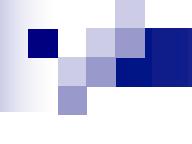
Funciona apenas se as pré-condições são satisfeitas

Estado anterior = $Em(C1, a) \wedge Em(a, B) \wedge Em(C2, B) \wedge \dots \wedge Em(C20, B)$

Sub-objetivo $Em(C1, B)$ já não está presente neste estado

Procura Regressiva

- Propriedades:
 - **Apenas considera ações relevantes para o objetivo**
 - Tipicamente fator de ramificação muito inferior ao da procura progressiva
 - **No entanto, devido à descrição representar conjunto de estados em vez de estados individuais**
 - Faz com que seja muito mais difícil construir boas heurísticas
 - **Por esta razão, hoje em dia a Procura Progressiva é preferida à Procura Regressiva**



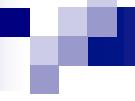
Planeamento

- 1. Definição de planeamento clássico
- 2. Algoritmos para planeamento clássico
- 3. Heurísticas para planeamento
- *Grafos de planeamento*
- *Planeamento de ordem parcial*

Heurísticas para procura em espaço de estados

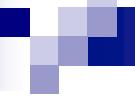
- As procura progressiva e regressiva não são eficientes sem uma boa heurística
 - **Quantas ações são necessárias para alcançar este objetivo?**
 - **Solução exata é NP difícil → encontrar uma boa heurística**
- Duas abordagens para encontrar uma heurística admissível:
 - **Uma solução ótima para um problema relaxado**
 - Remover todas as pré-condições das ações
 - **Assumir sub-objetivos independentes**

O custo de alcançar um conjunto de sub-objetivos é (aproximadamente) igual à soma dos custos de resolver os problemas independentemente



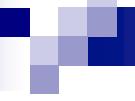
Heurísticas para procura em espaço de estados

- As procuras progressiva e regressiva não são eficientes sem uma boa heurística
 - **Custo de caminho**
 - Número de ações para atingir o objetivo
 - **Como estimar este custo?**
 - Uma solução ótima para um problema relaxado



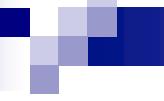
Heurística de ignorar pré-condições

- Heurística de ignorar pré-condições
 - **Considerar problema de planeamento relaxado onde as ações não têm pré-condições**
 - Todas as ações são aplicáveis em qualquer estado
 - Qualquer literal objetivo pode ser atingido com um único passo



Heurística de ignorar pré-condições

- O número de ações necessárias para resolver o problema relaxado é **quase** o número de literais objetivo ainda não atingidos
 - 1. Uma ação pode atingir mais que um literal objetivo**
 - 2. Uma ação pode desfazer o efeito de outras**
- Para muitos problemas, considerar 1) e ignorar 2) é uma boa heurística



Heurística de ignorar pré-condições

- Começamos por simplificar as ações
 - **Remover as precondições**
 - **Remover todos os efeitos que não sejam um literal objetivo**
 - **Como heurística conta-se o número mínimo de ações necessárias de modo a que a união dos efeitos dessas ações satisfaça o objetivo**
 - Problema: fazer isto é NP-difícil
 - Existe um algoritmo ganancioso que consegue calcular este valor rapidamente
 - Mas perde garantia de admissibilidade

Heurística de ignorar pré-condições

- Podemos remover apenas algumas precondições
 - **Ex: 8-puzzle**
Ação(**Mover(p, pos1, pos2)**,
Pré-cond:
Em(p, pos1) \wedge Peça(p) \wedge Vazia(pos2) \wedge Adjacente(pos1, pos2)
Efeito:
Em(p, pos2) \wedge Vazia(pos1) \wedge \neg Em(p, pos1) \wedge \neg Vazia(pos2))
 - **Se removermos pré-cond Vazia(pos2), obtemos a heurística da distância de Manhattan.**
 - **Heurística derivada automaticamente do esquema de ação.**
 - **É a grande vantagem da representação utilizada para planeamento.**

Heurística de ignorar lista de remoções

- Heurística de ignorar lista de remoções
 - **Ignore delete lists heuristic**
 - **Assumir que não existem pós-condições com literais negativos**
 - **Remover todos os literais negativos dos efeitos**
 - **Problema relaxado**
 - Nenhuma ação vai desfazer o progresso feito por outra
 - Uma solução aproximada para este problema pode ser encontrada em tempo polinomial usando Hill-Climbing

Heurísticas de decomposição

- Heurísticas de decomposição
 - **Decompor o objetivo em vários subobjetivos g_1, g_2, \dots, g_n**
 - **$H = \text{Max}(c(g_1), c(g_2), \dots, c(g_n))$**
 - Heurística admissível
 - **$H = c(g_1) + c(g_2) + \dots + c(g_n)$**
 - Boa estimativa
 - Mas não garante admissibilidade
 - **$c(g_1) + c(g_2) > h^*$ quando a solução para g_1 tem ações redundantes com a solução para g_2**
 - **A não ser para subobjetivos g_1, g_2, \dots, g_n independentes**

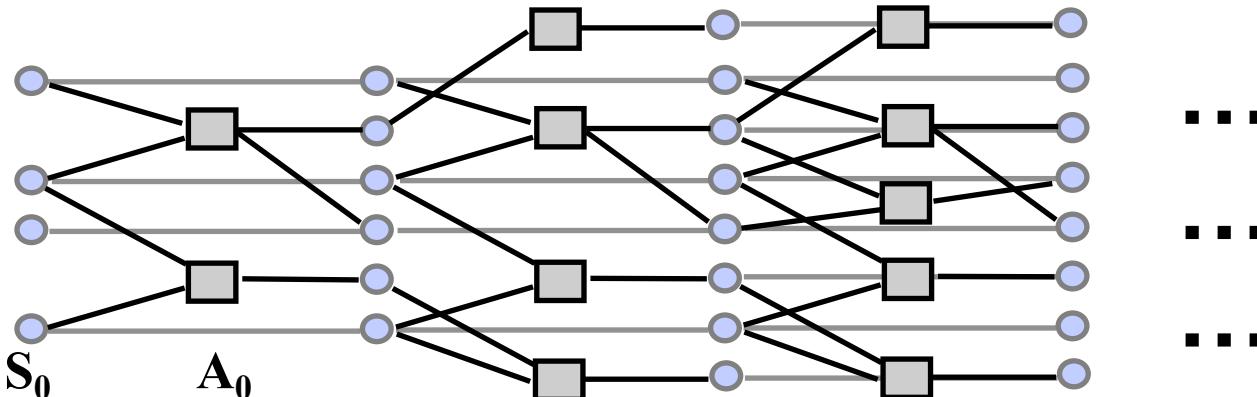
Planeamento

- 1. Definição de planeamento clássico
- 2. Algoritmos para planeamento clássico
- 3. Heurísticas para planeamento
- *Grafos de planeamento*
- *Planeamento de ordem parcial*

Grafos de planeamento

- Estrutura de dados usada para obter estimativas mais precisas para as heurísticas
 - **Estimativa admissível de quantos passos são necessários para atingir um estado objetivo g**
- Solução também pode ser diretamente extraída a partir de um grafo de planeamento usando o algoritmo GRAPHPLAN

Grafos de planeamento

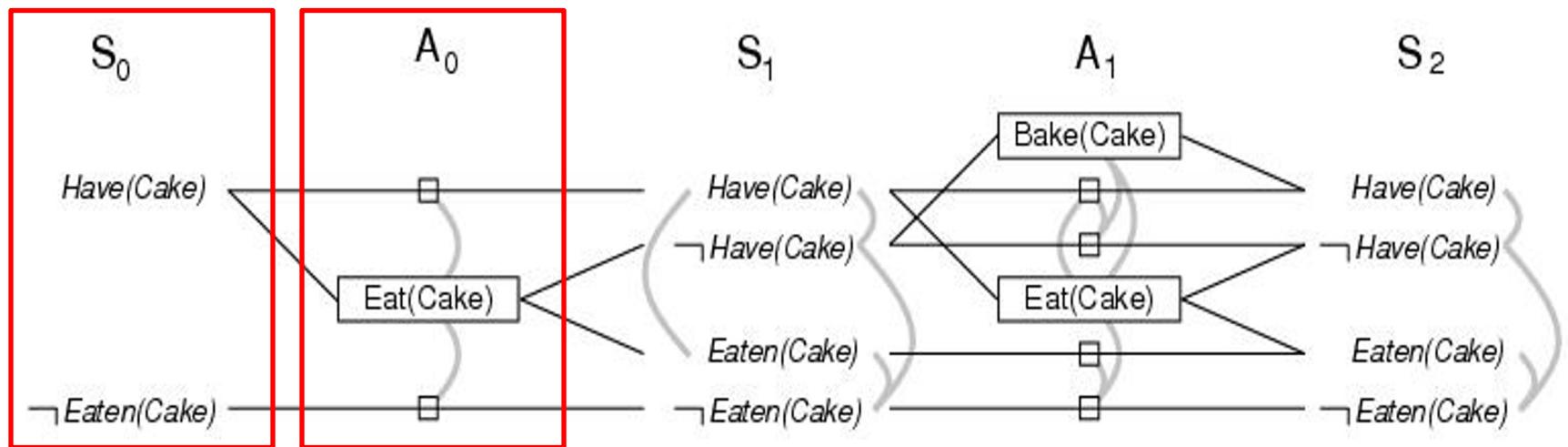


- Consiste numa sequência de níveis que correspondem a instantes de tempo no plano
 - **S_0 é o estado inicial**
 - **Cada nível ($S_i + A_i$) consiste num conjunto de literais e num conjunto de ações**
 - $S_i = \text{Literais}$ (light blue circle) = todos os literais que *podem* ser verdadeiros nesse instante de tempo, dependendo das ações executadas no instante de tempo anterior
 - $A_i = \text{Ações}$ (light gray square) = todas as ações que *podem* ter as suas pré-condições satisfeitas nesse instante de tempo, dependendo dos literais verdadeiros nesse instante

Grafos de planeamento

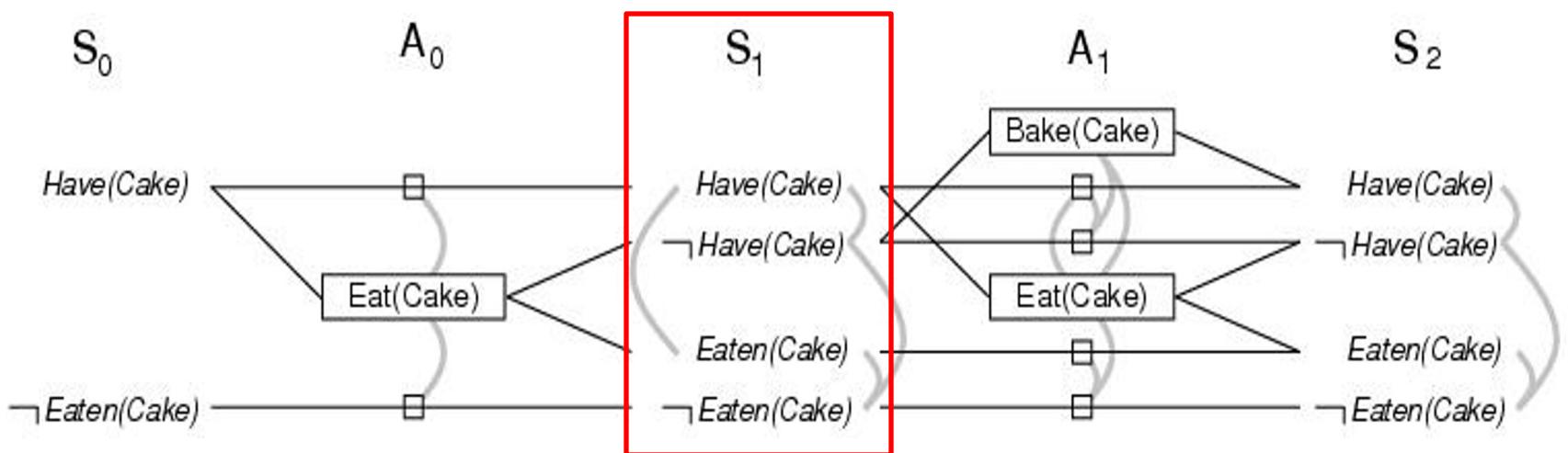
- Significado de “podem”?
 - **Registo de apenas um sub-conjunto restrito de possíveis interações negativas entre ações**
- Funciona apenas para problemas proposicionais
 - **Isto é, sem variáveis**
- Exemplo em PDDL:
 - Init(Have(Cake))
 - Goal(Have(Cake) \wedge Eaten(Cake))
 - Action(Eat(Cake),
 - PRECOND: Have(Cake)
 - EFFECT: \neg Have(Cake) \wedge Eaten(Cake))
 - Action(Bake(Cake),
 - PRECOND: \neg Have(Cake)
 - EFFECT: Have(Cake))

Exemplo do bolo



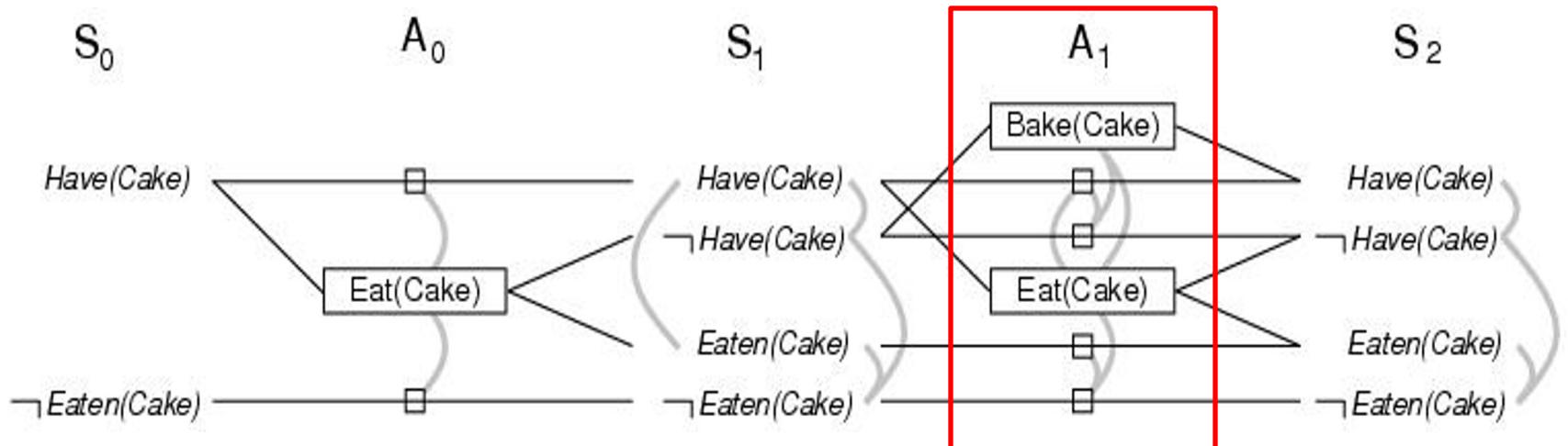
- Início em S_0
 - **Representa o estado inicial**
- A_0 contém as ações cujas pré-condições são satisfeitas por S_0
 - **Inação é representada pela persistência de ações** (\square); para cada condição c em S criar ação com precondição e efeito c
- Conflitos entre ações são representadas por relações *mutex*
 - **Representadas pelas linhas curvas a cinzento**
 - **Representam exclusividade mútua**: neste caso ação Eat(Cake) tem como efeitos $\neg \text{Have(Cake)} \wedge \text{Eaten(Cake)}$

Exemplo do bolo



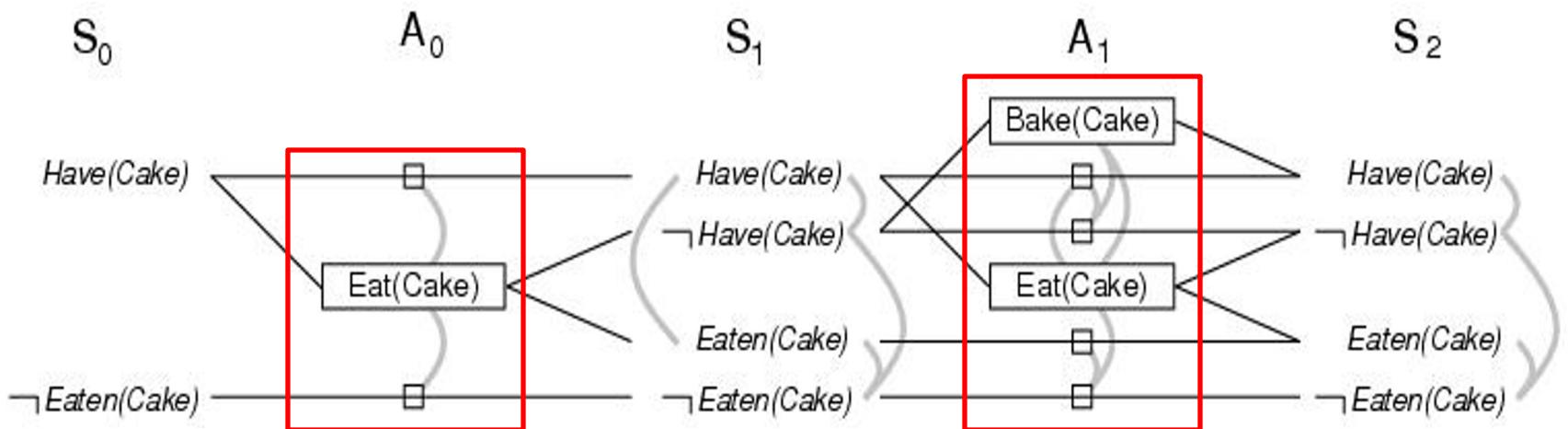
- S_1 contém todos os literais resultantes das ações em A_0
- S_1 contém também relações *mutex*
 - **Contradições:** $\neg \text{Have(Cake)}$ e Have(Cake) , $\neg \text{Eaten(Cake)}$ e Eaten(Cake)
 - **Outros casos:** Have(Cake) e Eaten(Cake) , $\neg \text{Have(Cake)}$ e $\neg \text{Eaten(Cake)}$
 - Consequência de só poder ser escolhida uma ação em A_0

Exemplo do bolo



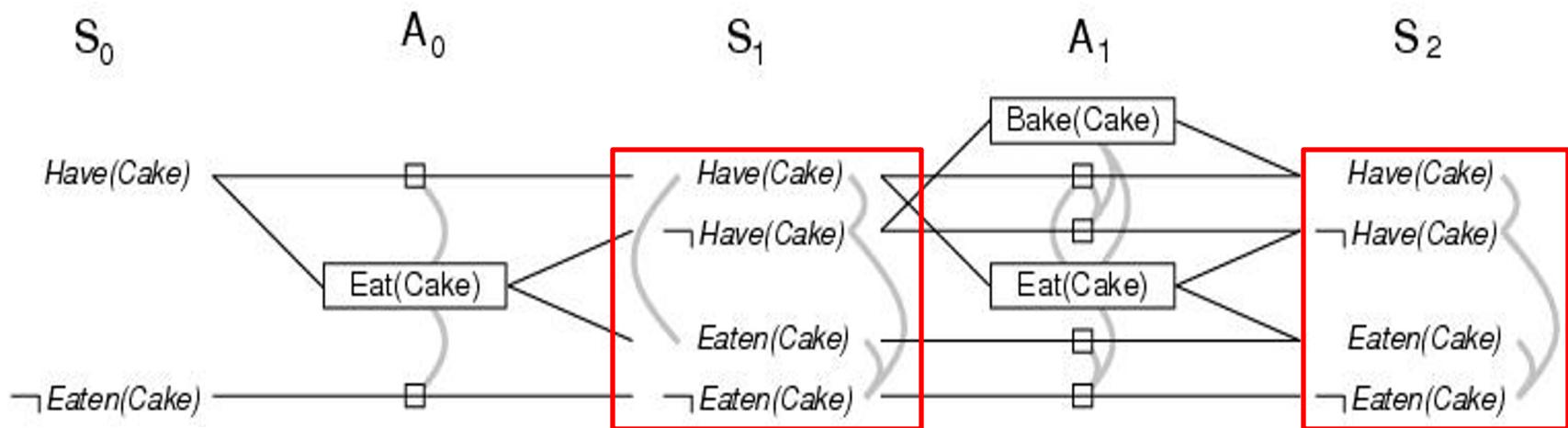
- Em A_1 podem ter lugar as duas ações
- Relações *mutex* em A_1
 - **Contradições:** $\neg \text{Have}(\text{Cake})$ e $\text{Have}(\text{Cake})$, $\neg \text{Eaten}(\text{Cake})$ e $\text{Eaten}(\text{Cake})$
 - **Outros casos:** $\text{Bake}(\text{Cake})$ e $\neg \text{Have}(\text{Cake})$, $\text{Bake}(\text{Cake})$ e $\text{Eat}(\text{Cake})$, $\neg \text{Eaten}(\text{Cake})$ e $\text{Eat}(\text{Cake})$

Exemplo do bolo



- Uma relação *mutex* é estabelecida entre duas ações quando:
 - **Inconsistência**: uma ação nega o efeito da outra e.g. ações $\text{Eat}(\text{Cake})$ e $\text{Have}(\text{Cake})$
 - **Interferência**: um dos efeitos de uma ação é a negação da pré-condição de outra e.g. ações $\text{Eat}(\text{Cake})$ e $\text{Have}(\text{Cake})$
 - **Competição**: uma das pré-condições de uma ação é mutuamente exclusiva em relação à pré-condição de outra e.g. ações $\text{Eat}(\text{Cake})$ e $\text{Bake}(\text{Cake})$

Exemplo do bolo

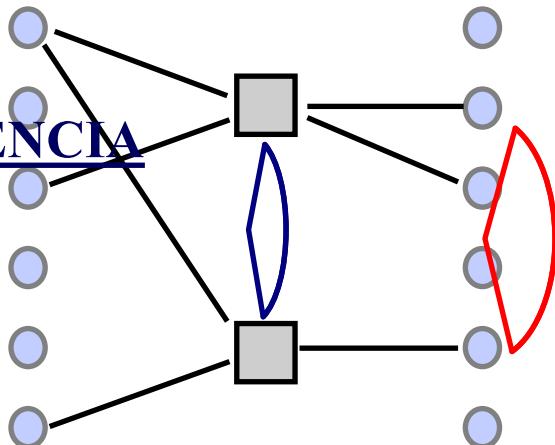


- Relação *mutex* entre **dois literais** (*suporte inconsistente*) quando:
 - **Um é a negação do outro**
 - **Cada par de ações possível que podia produzir esses literais é mutex**
 - Em S_1 : literais Have(Cake) e Eaten(Cake) não são possíveis porque a única forma de obter Have(Cake) (ação Have(Cake) \square) é incompatível com a única forma de obter Eaten(Cake) (ação Eat(Cake))
 - Em S_2 : literais Have(Cake) e Eaten(Cake) já são possíveis porque podem ser adquiridos por duas ações compatíveis (Bake(Cake) e Eaten(Cake))

Relações mutex: resumo

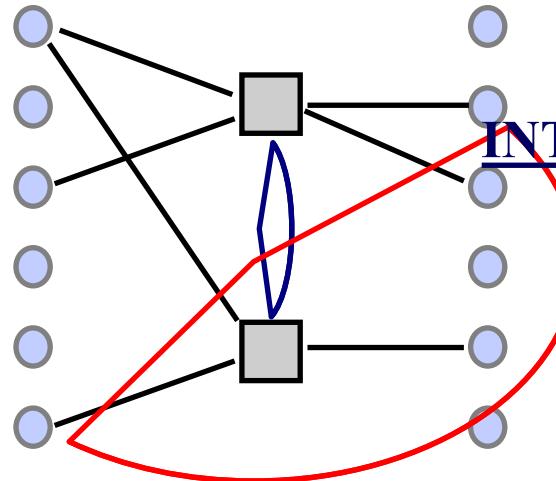
INCONSISTÊNCIA

uma ação nega o efeito da outra



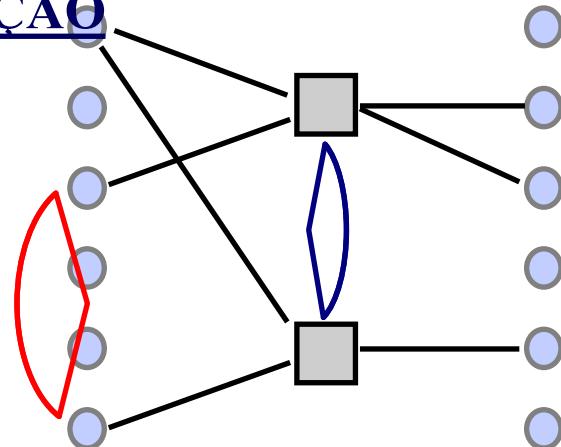
INTERFERÊNCIA

um efeito nega uma pré-condição



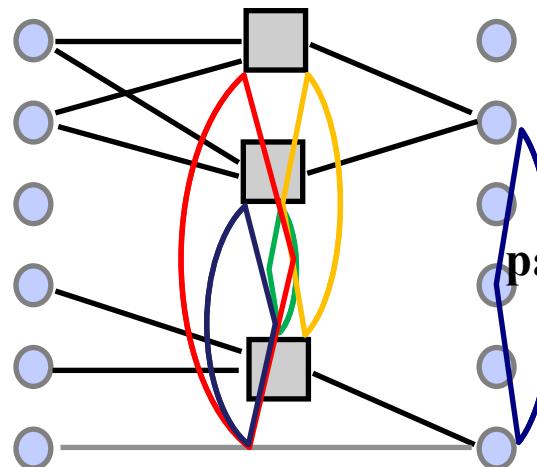
COMPETIÇÃO

uma pré-condição nega outra

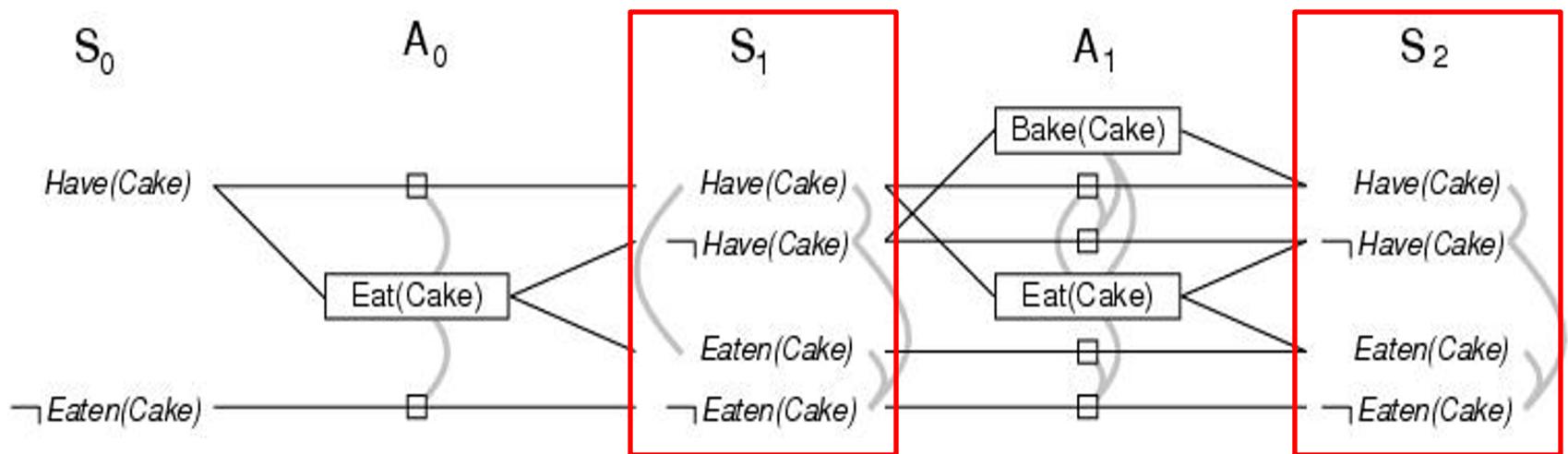


SUPORTE INCONSISTENTE

literais são efeito de pares de ações mutex



Exemplo do bolo



- Continuar até que dois níveis consecutivos tenham os mesmos literais:
grafo está *leveled off*
 - Significa que expansão adicional é desnecessária**

Grafos de planeamento e heurísticas

- Planeamento com grafos disponibiliza informação sobre o problema
 - **Um literal que não aparece no nível final do grafo não pode ser alcançado por nenhum plano**
 - **Custo de atingir um literal do objetivo g_i**
 - Estimado como o nível onde o literal g_i aparece no grafo de planeamento
 - Chamado custo de nível de g_i
 - **Estimativa é admissível**

Grafos de planeamento e heurísticas

- Estimativa custo de nível
 - **Não é correta sempre**
 - **Num grafo de planeamento podem ocorrer várias ações por nível (se não forem *mutex*)**
 - **Estimativa apenas conta o nível e não o número de ações**
- Grafos de Planeamento em série
 - **Um grafo em série garante que só pode ocorrer uma ação por nível**
 - **Adicionar ligações *mutex* entre cada par de ações, exceto para ações persistentes**
 - **Custos de níveis extraídos de grafos em série são estimativas bastante razoáveis dos custos reais**

Grafos de planeamento e heurísticas

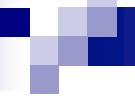
- Como estimar o custo de uma conjunção de literais objetivo?
 - **Heurística de máximo nível (max-level heuristic)**
 - Máximo do custo de nível de um objetivo
 - Admissível, mas não necessariamente a mais correta
 - **Heurística de soma de níveis (level-sum heuristic)**
 - Soma do custo de nível de cada objetivo
 - Não é admissível
 - Mas devolve valores muito próximos do real
 - **Heurística de nível de conjunto (set-level heuristic)**
 - Encontrar o nível onde todos os literais objetivo aparecem no grafo de planeamento sem qualquer par deles ser mutex
 - Admissível
 - Domina heurística máximo nível

Planeamento

- 1. Definição de planeamento clássico
- 2. Algoritmos para planeamento clássico
- 3. Heurísticas para planeamento
- *Grafos de planeamento*
- *Planeamento de ordem parcial*

Planeamento de ordem parcial

- Planeamento com procura progressiva e regressiva resulta em planos de procura totalmente ordenados
 - **Não é possível obter as vantagens da decomposição de problemas**
 - Decisões devem ser feitas de modo a encontrar sequências de ações para todos os sub-problemas
- Estratégia do compromisso mínimo
 - **Adiar decisões durante a procura**



Planeamento de ordem parcial

- Planeamento de ordem parcial
 - **Procura espaço de estados**
 - Mas *estados* são planos

Exemplo dos sapatos

Goal(*RightShoeOn* \wedge *LeftShoeOn*)

Init()

Action(*RightShoe*, PRECOND: *RightSockOn*, EFFECT: *RightShoeOn*)

Action(*RightSock*, PRECOND: , EFFECT: *RightSockOn*)

Action(*LeftShoe*, PRECOND: *LeftSockOn*, EFFECT: *LeftShoeOn*)

Action(*LeftSock*, PRECOND: , EFFECT: *LeftSockOn*)

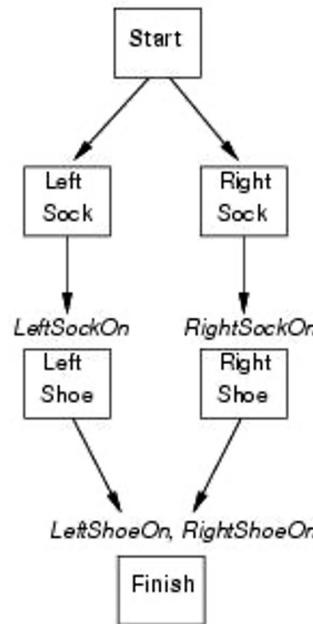
Plano: combinar duas sequências de ações (1)

leftsock, leftshoe (2) rightsock, rightshoe

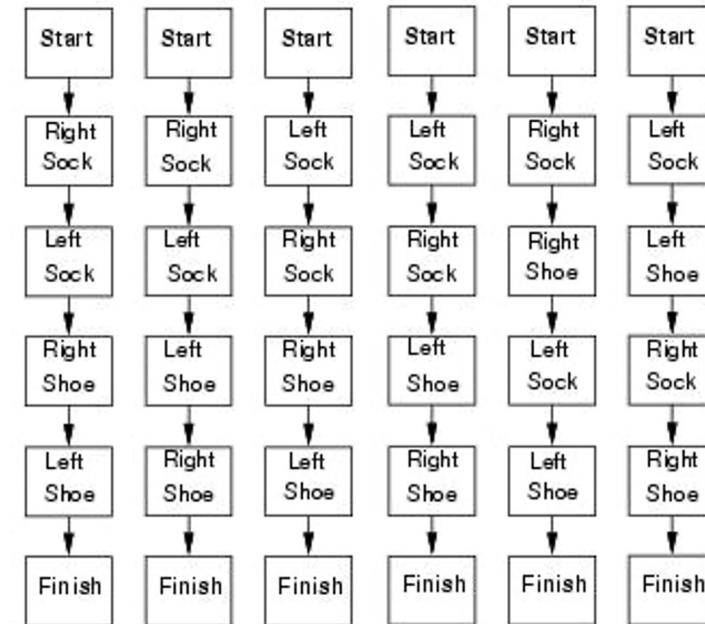
Planeamento de ordem parcial(POP)

- Ações num plano com ordem de execução não determinada; planos com ordem total são uma linearização da ordem parcial

Partial Order Plan:



Total Order Plans:



POP como problema de procura

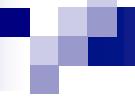
- Estados são (tipicamente) planos inacabados
 - **Plano vazio contém apenas ações iniciar e terminar**
- Cada plano tem 4 componentes:
 - **Um conjunto de ações (etapas do plano)**
 - **Um conjunto de restrições de ordem: $A < B$**
 - A tem que ser executado antes de B
 - Ciclos representam contradições
 - **Um conjunto de ligações causais** $A \xrightarrow{p} B$
 - Representa que a ação A atinge a precondição p para B
 - Ligação protegida
 - **Não podemos adicionar nenhuma ação entre A e B que tenha como efeito $\neg p$**
 - **Um conjunto de pré-condições abertas**
 - Pré-condições que ainda não foram alcançadas através de uma ação

POP como problema de procura

- Um plano é *consistente* se e só se não existem ciclos nas restrições de ordem e não existem conflitos com as *ligações causais*
- Um plano consistente sem pré-condições abertas é uma *solução*
- Um plano de ordem parcial é executado ao executar repetidamente qualquer uma das próximas ações possíveis
 - **Esta flexibilidade é vantajosa em ambientes não cooperativos**

Algoritmo POP

- Considerar problemas de planeamento proposicionais:
 - **O plano inicial contém *Iniciar* e *Terminar*, a restrição de ordem *Iniciar* < *Terminar*, não há ligações causais, todas as pré-condições em *Terminar* estão abertas**
 - **Função sucessores:**
 - Escolher uma pré-condição p de uma ação B
 - Gerar um plano sucessor para todas as formas consistentes de satisfazer p
 - **Teste objetivo**
 - Plano consistente sem precondições abertas



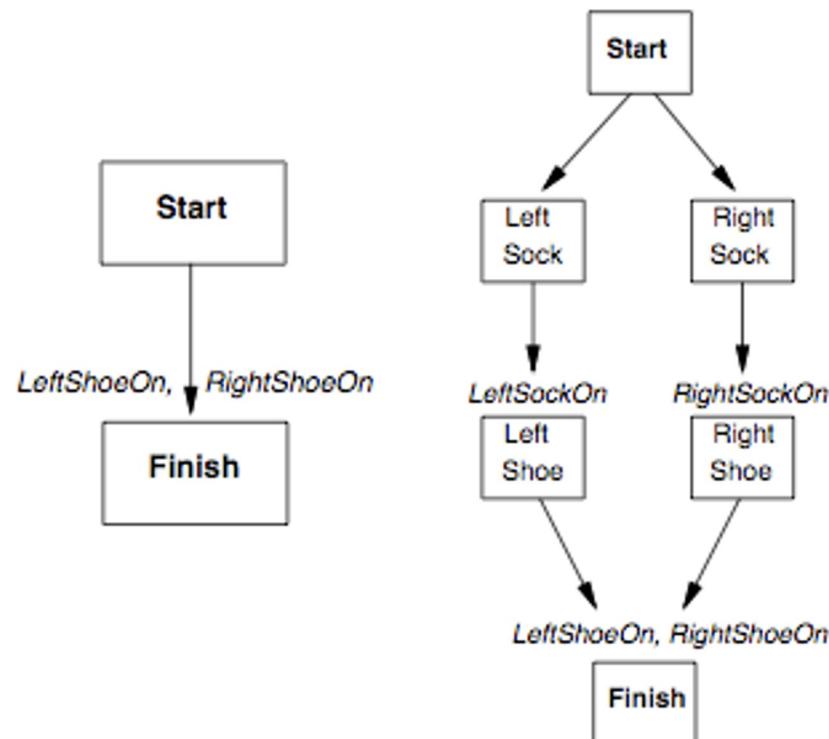
Algoritmo POP

- Satisfazer uma precondição p
 - 1. Verificar se a precondição p já é verificada no estado inicial**
 - 2. Tentar encontrar uma ação do plano que satisfaça p**
 - 3. Adicionar uma nova ação ao plano que satisfaça p**

Consistência

- Ao gerar um plano sucessor:
 - A relação causal $A \xrightarrow{P} B$ e a restrição de ordem $A < B$ é adicionada ao plano
 - Se A é novo então adicionar também Start < A e A < Finish ao plano
 - Precondições de A são adicionadas como precondições abertas ao plano
 - Resolver conflitos entre novas ligações causais e as ações existentes
 - Resolver conflitos entre a ação A (se for nova) e todas as ligações causais existentes

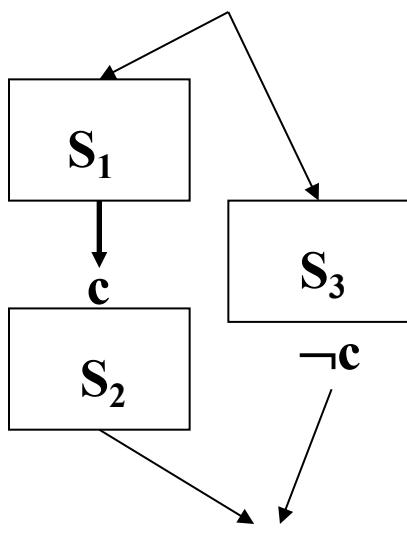
Exemplo: sapatos e peúgas



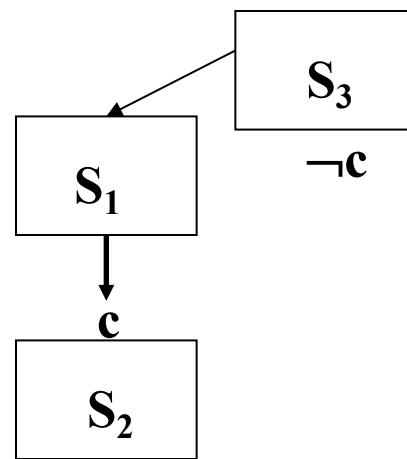
Resolução de conflitos

- Conflito:
 - **Existe uma ação C no plano com $\neg p$ que entra em conflito com a ligação causal**
$$A \xrightarrow{p} B$$
 - **Se C poder ser executado entre A e B**
- Conflictos são resolvidos com restrições de ordem ($<$)
 - **Demoção da ligação causal:** $C < A$
 - **Promoção da ligação causal:** $B < C$

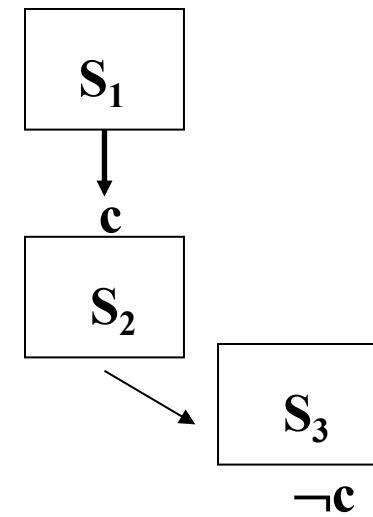
Promoção e Demoção



(a)



(b)
Demotion



(c)
Promotion

Algoritmo POP

```
function POP(initial, goal, operators) returns plan
    plan  $\leftarrow$  MAKE-MINIMAL-PLAN(initial, goal)
    loop do
        if SOLUTION?(plan) then return plan
        Sneed, c  $\leftarrow$  SELECT-SUBGOAL(plan)
        CHOOSE-OPERATOR(plan, operators, Sneed, c)
        RESOLVE-THREATS(plan)
    end
```

```
function SELECT-SUBGOAL(plan) returns Sneed, c
    pick a plan step Sneed from STEPS(plan)
        with a precondition c that has not been achieved
    return Sneed, c
```

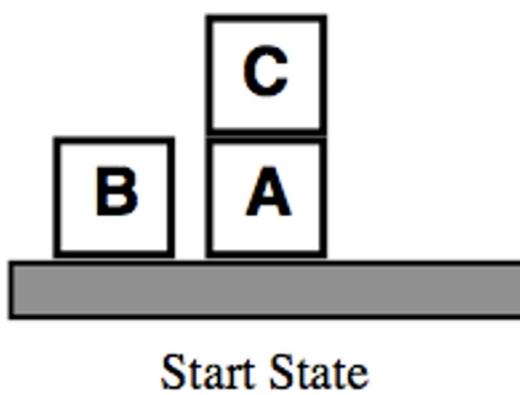
Algoritmo POP (cont.)

```
procedure CHOOSE-OPERATOR(plan, operators, Sneed, c)
    choose a step Sadd from operators or STEPS(plan) that has c as an effect
    if there is no such step then fail
    add the causal link  $S_{add} \xrightarrow{c} S_{need}$  to LINKS(plan)
    add the ordering constraint  $S_{add} \prec S_{need}$  to ORDERINGS(plan)
    if Sadd is a newly added step from operators then
        add Sadd to STEPS(plan)
        add Start  $\prec S_{add} \prec$  Finish to ORDERINGS(plan)
```

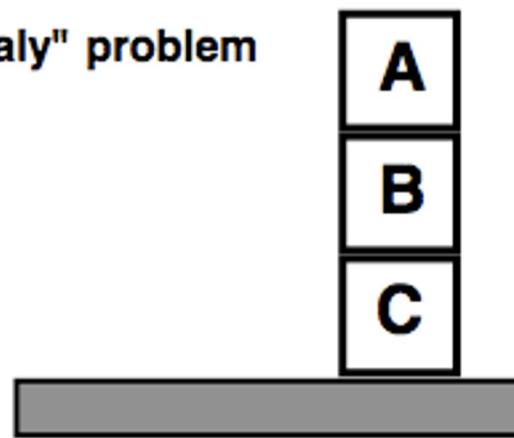
```
procedure RESOLVE-THREATS(plan)
    for each Sthreat that threatens a link  $S_i \xrightarrow{c} S_j$  in LINKS(plan) do
        choose either
            Demotion: Add  $S_{threat} \prec S_i$  to ORDERINGS(plan)
            Promotion: Add  $S_j \prec S_{threat}$  to ORDERINGS(plan)
        if not CONSISTENT(plan) then fail
    end
```

Exemplo: mundo dos blocos

"Sussman anomaly" problem



Start State



Goal State

$\text{Clear}(x) \text{ On}(x,z) \text{ Clear}(y)$

PutOn(x,y)

$\sim\text{On}(x,z) \sim\text{Clear}(y)$
 $\text{Clear}(z) \text{ On}(x,y)$

$\text{Clear}(x) \text{ On}(x,z)$

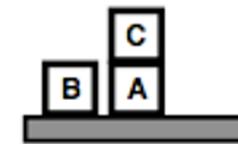
PutOnTable(x)

$\sim\text{On}(x,z) \text{ Clear}(z) \text{ On}(x,\text{Table})$

Exemplo: mundo dos blocos

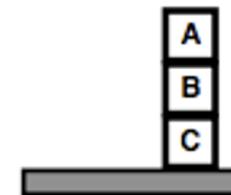
START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

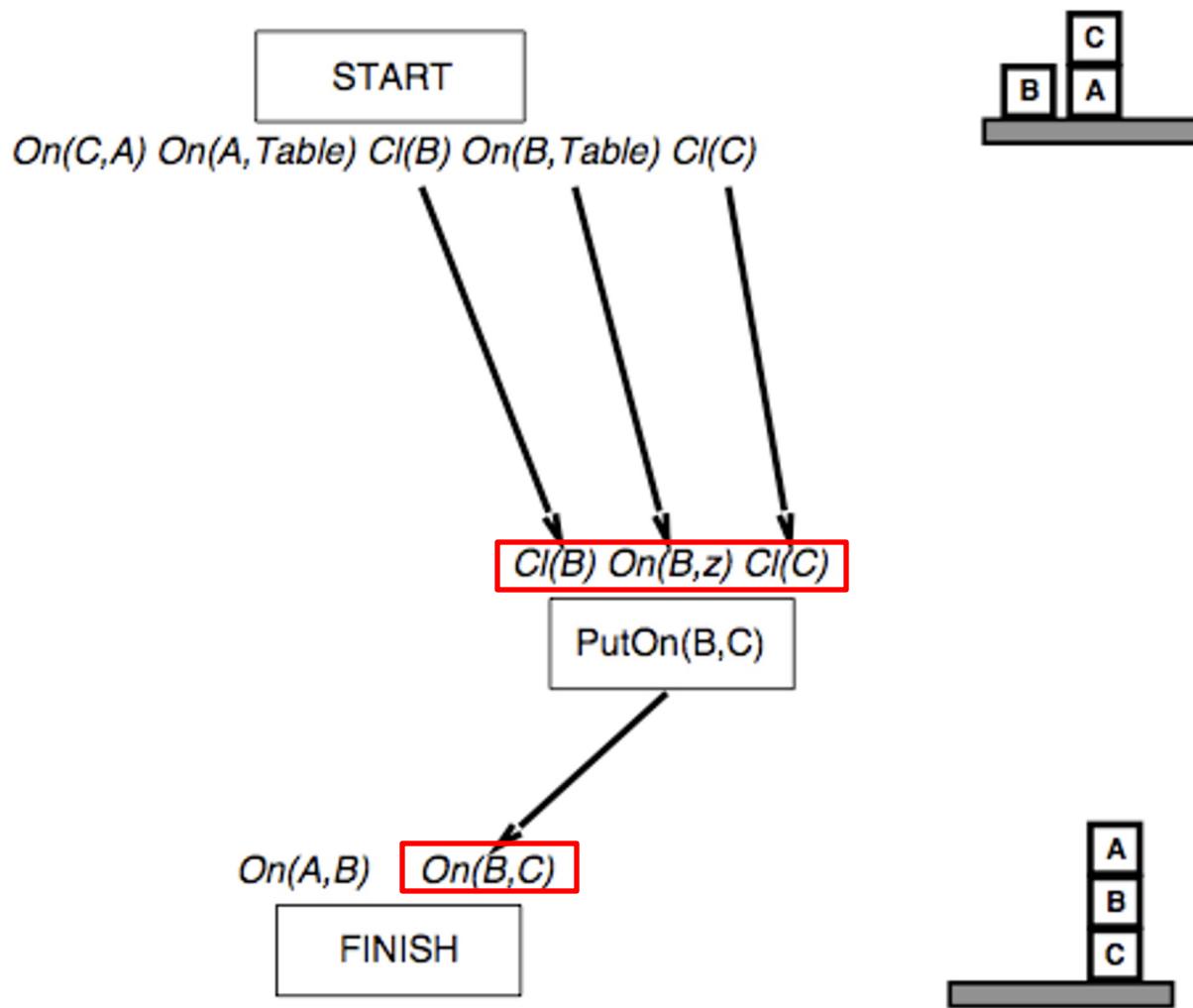


On(A,B) On(B,C)

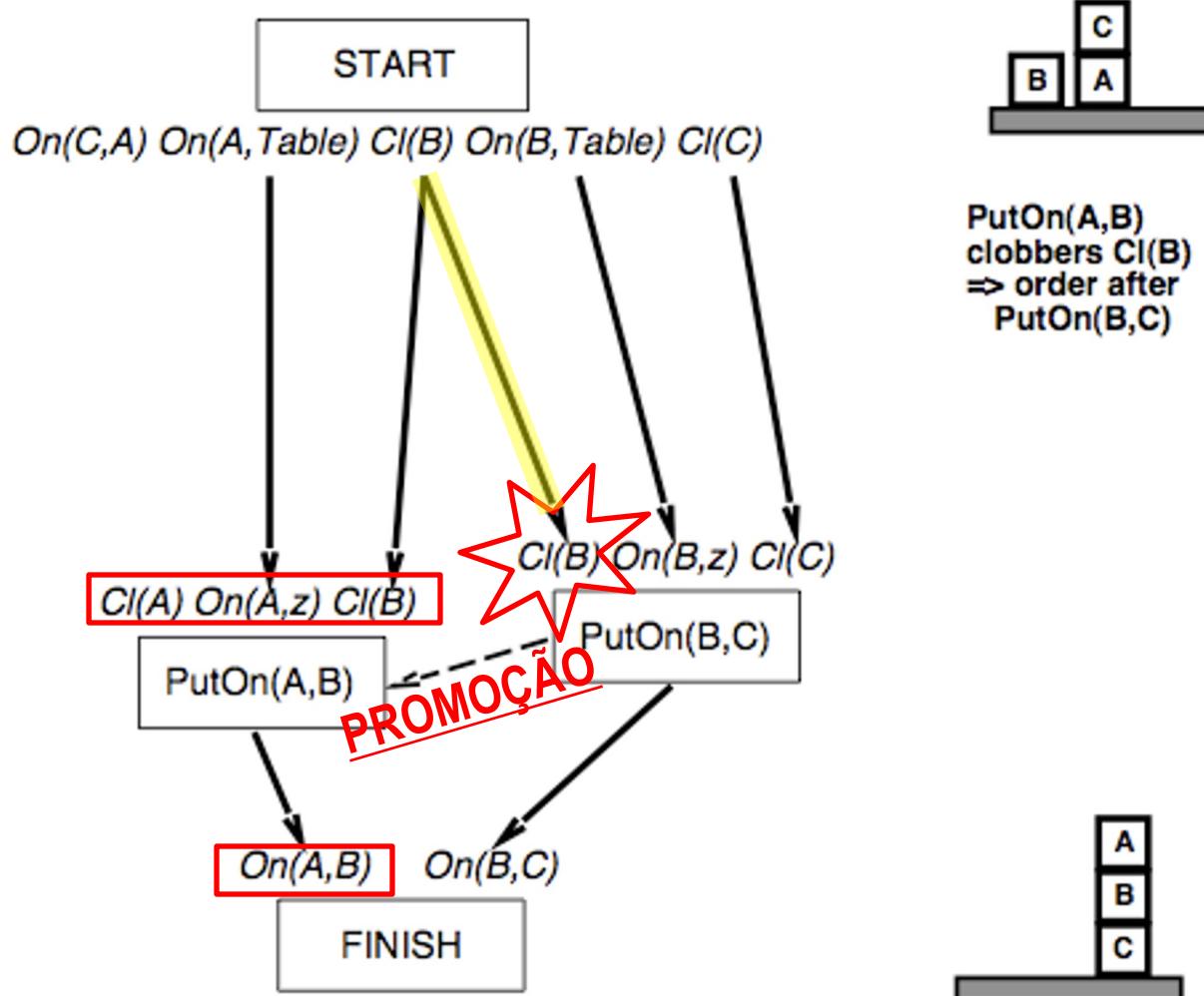
FINISH



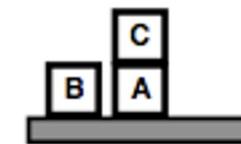
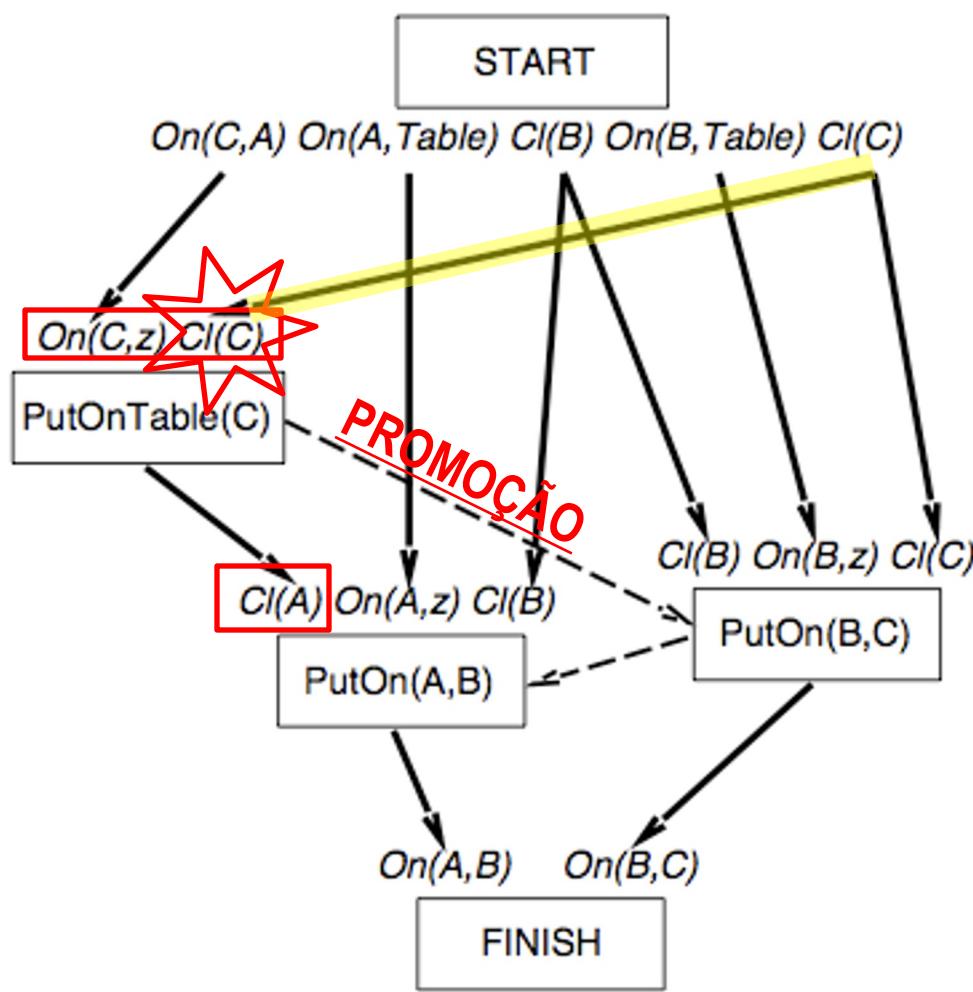
Exemplo: mundo dos blocos



Exemplo: mundo dos blocos

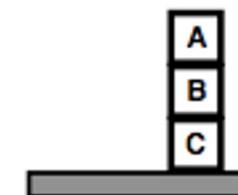


Exemplo: mundo dos blocos



$\text{PutOn}(A,B)$
clobbers $Cl(B)$
=> order after
 $\text{PutOn}(B,C)$

$\text{PutOn}(B,C)$
clobbers $Cl(C)$
=> order after
 $\text{PutOnTable}(C)$



Exemplo: Pneu sobresselente

Init(At(Flat, Axle) \wedge At(Spare, Trunk))

Goal(At(Spare, Axle))

Action(Remove(Spare, Trunk))

PRECOND: *At(Spare, Trunk)*

EFFECT: \neg *At(Spare, Trunk) \wedge At(Spare, Ground)*)

Action(Remove(Flat, Axle))

PRECOND: *At(Flat, Axle)*

EFFECT: \neg *At(Flat, Axle) \wedge At(Flat, Ground)*)

Action(PutOn(Spare, Axle))

PRECOND: *At(Spare, Ground) \wedge \neg At(Flat, Axle)*

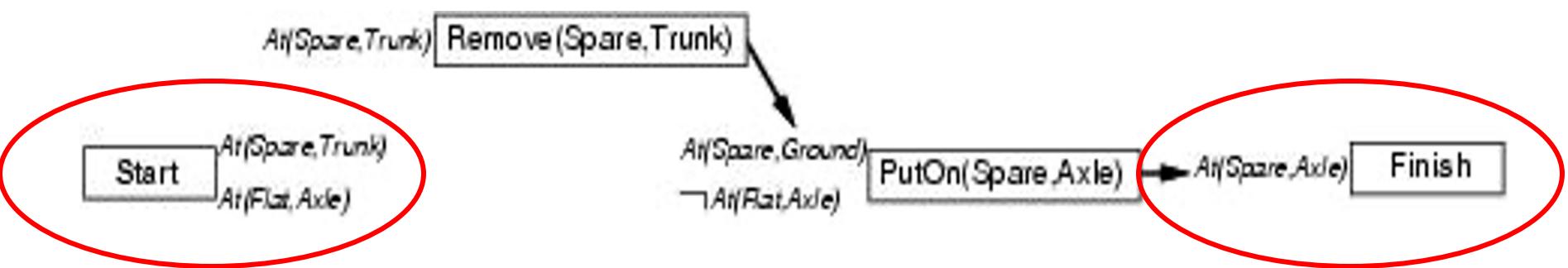
EFFECT: *At(Spare, Axle) \wedge \neg At(Spare, Ground)*)

Action(LeaveOvernight)

PRECOND:

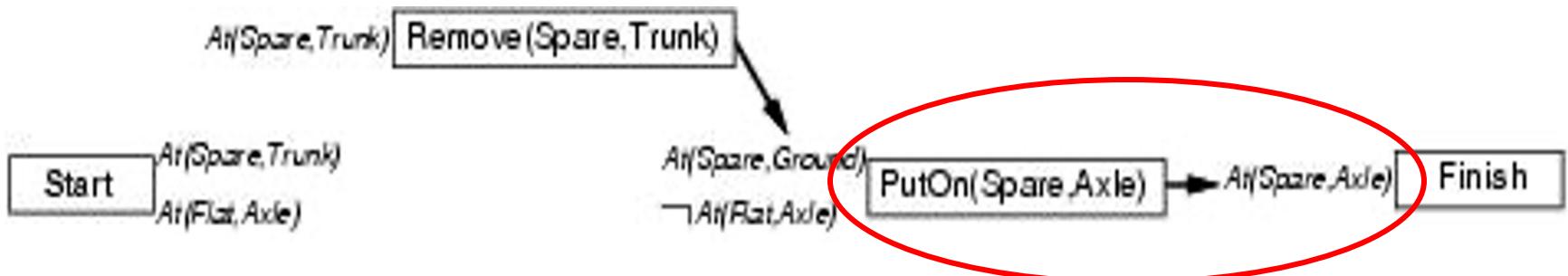
EFFECT: \neg *At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk) \wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))*

Resolução do problema



- Plano inicial:
 - **START** com **EFEITO** = Estado Inicial
 - **Finish** com **PRÉ-CONDIÇÃO** = Objetivo

Resolução do problema



- Escolher uma pré-condição aberta: $At(Spare, Axle)$
- Somente a ação $PutOn(Spare, Axle)$ é aplicável
- Adicionar ligação causal entre $PutOn(Spare, Axle)$ e $Finish$
- Adicionar restrição de ordem: $PutOn(Spare, Axle) < Finish$

Resolução do problema

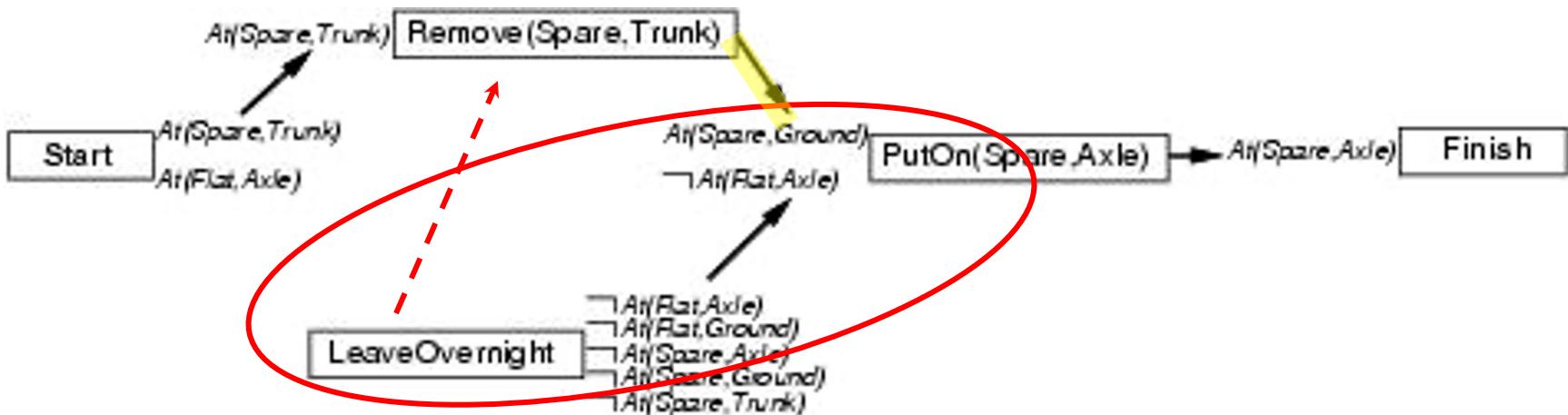


- Escolher uma pré-condição aberta: $At(Spare, Ground)$
- Somente $Remove(Spare, Trunk)$ é aplicável
- Adicionar ligação causal:

$Remove(Spare, Trunk) \xrightarrow{At(Spare, Ground)} PutOn(Spare, Axle)$

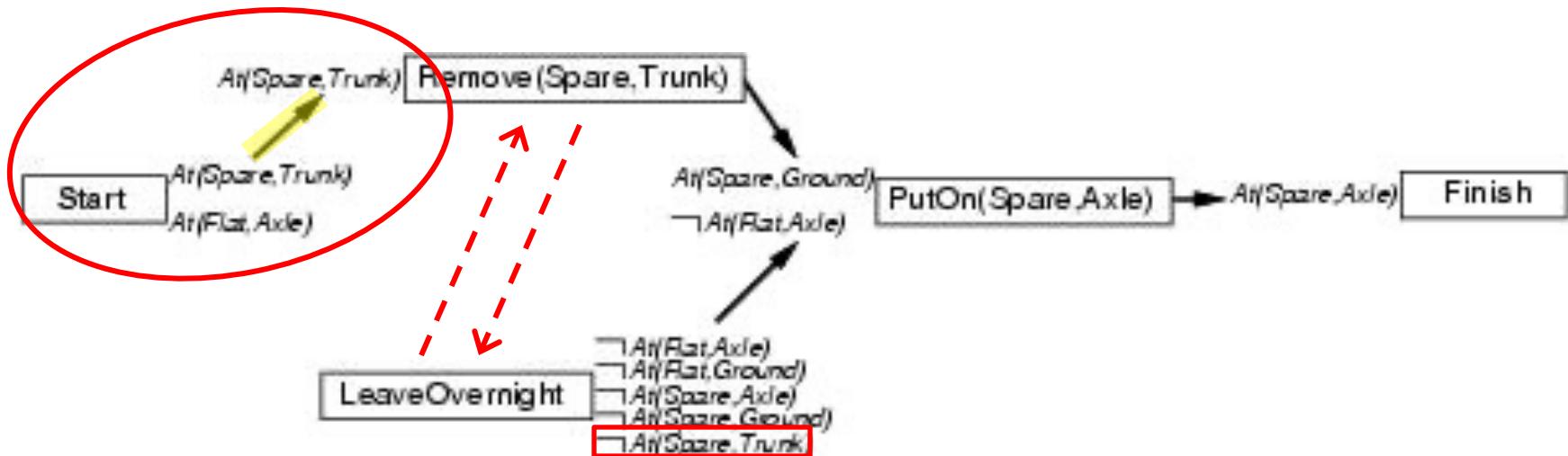
- Adicionar restrição de ordem:
 $Remove(Spare, Trunk) < PutOn(Spare, Axle)$

Resolução do problema



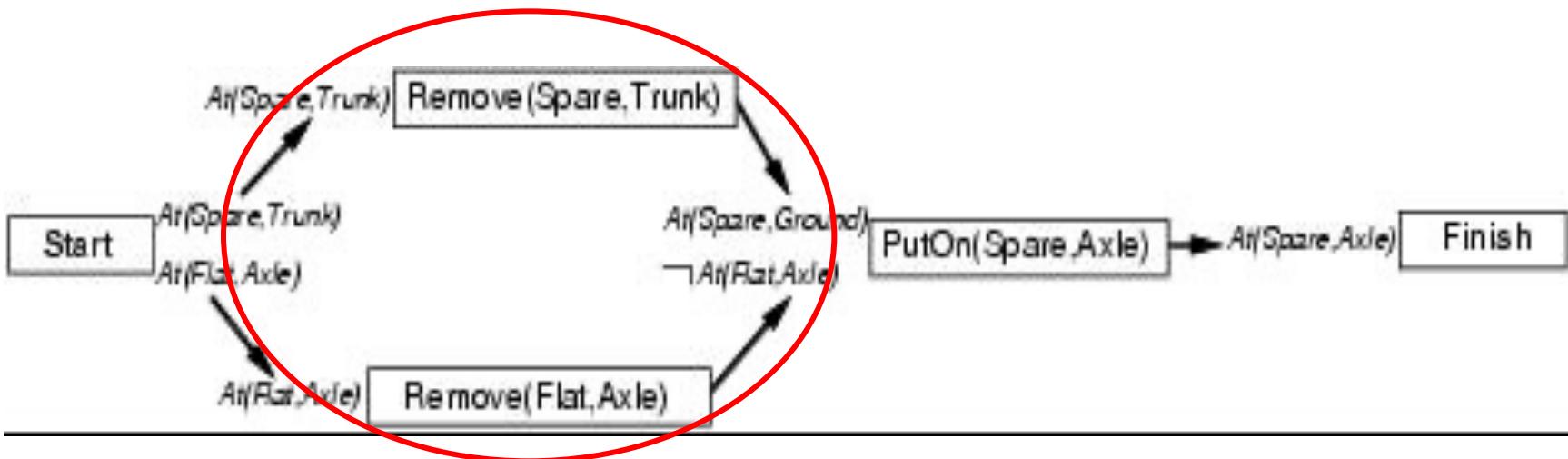
- Escolher uma pré-condição aberta: $\neg At(Flat, Axle)$
- *LeaveOverNight* é aplicável
- Conflito: $Remove(Spare, Trunk) \xrightarrow{At(Spare, Ground)} PutOn(Spare, Axle)$
- Para o resolver, adicionar restrição (**DEMOÇÃO**):
$$LeaveOverNight < Remove(Spare, Trunk)$$
- Adicionar ligação causal:
$$LeaveOverNight \xrightarrow{\neg At(Flat, Axle)} PutOn(Spare, Axle)$$

Resolução do problema



- Escolher uma pré-condição aberta : $At(Spare, Trunk)$
- Somente *Start* é aplicável
- Adicionar ligação causal: $Start \xrightarrow{At(Spare,Trunk)} Remove(Spare,Trunk)$
- Conflito: da ligação causal com o efeito $At(Spare, Trunk)$ em *LeaveOverNight*
 - **Não é possível encontrar uma solução mesmo com restrições de ordem**
- Retrocesso é a única saída!

Resolução do problema

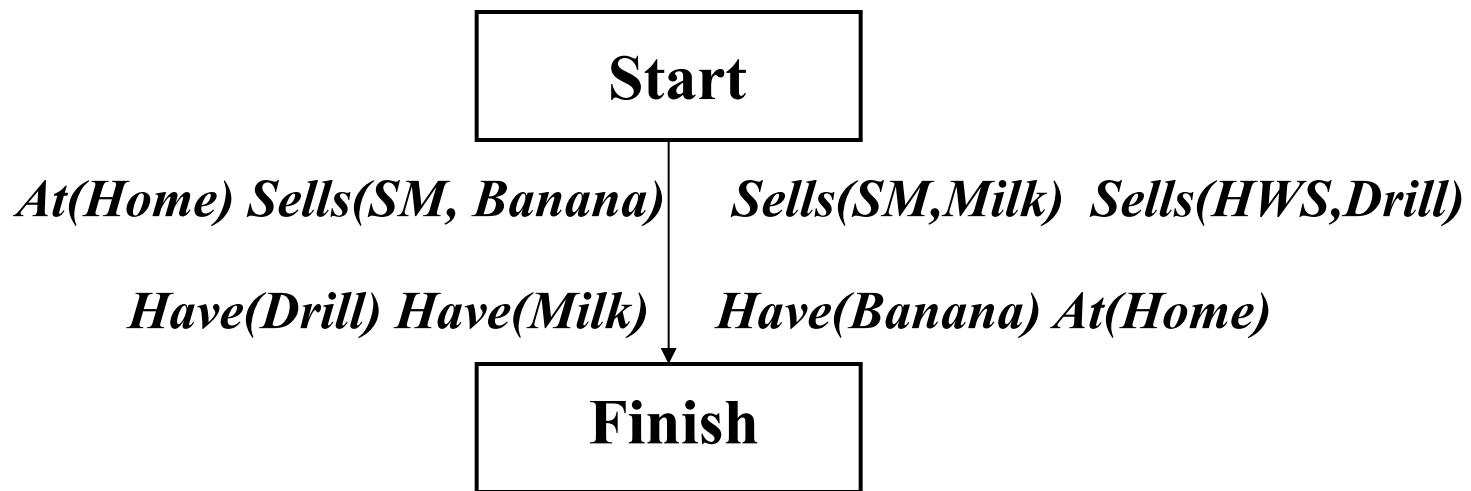


- Remover *LeaveOverNight*, *Remove(Spare, Trunk)* e ligações causais
- Repetir passo com *Remove(Spare, Trunk)*
- Adicionar também *Remove(Flat, Axle)*

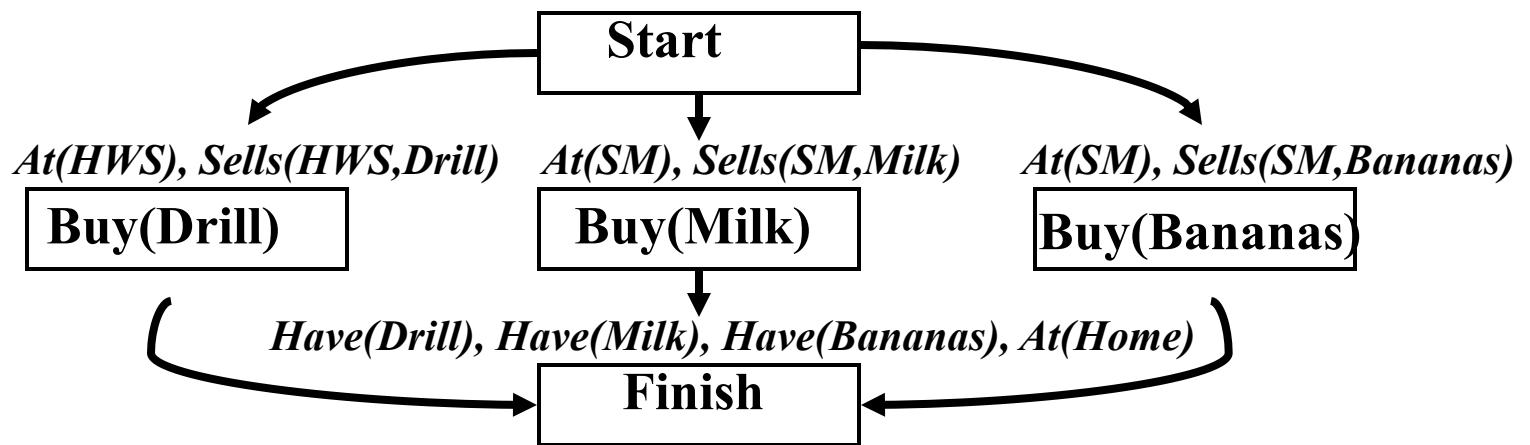
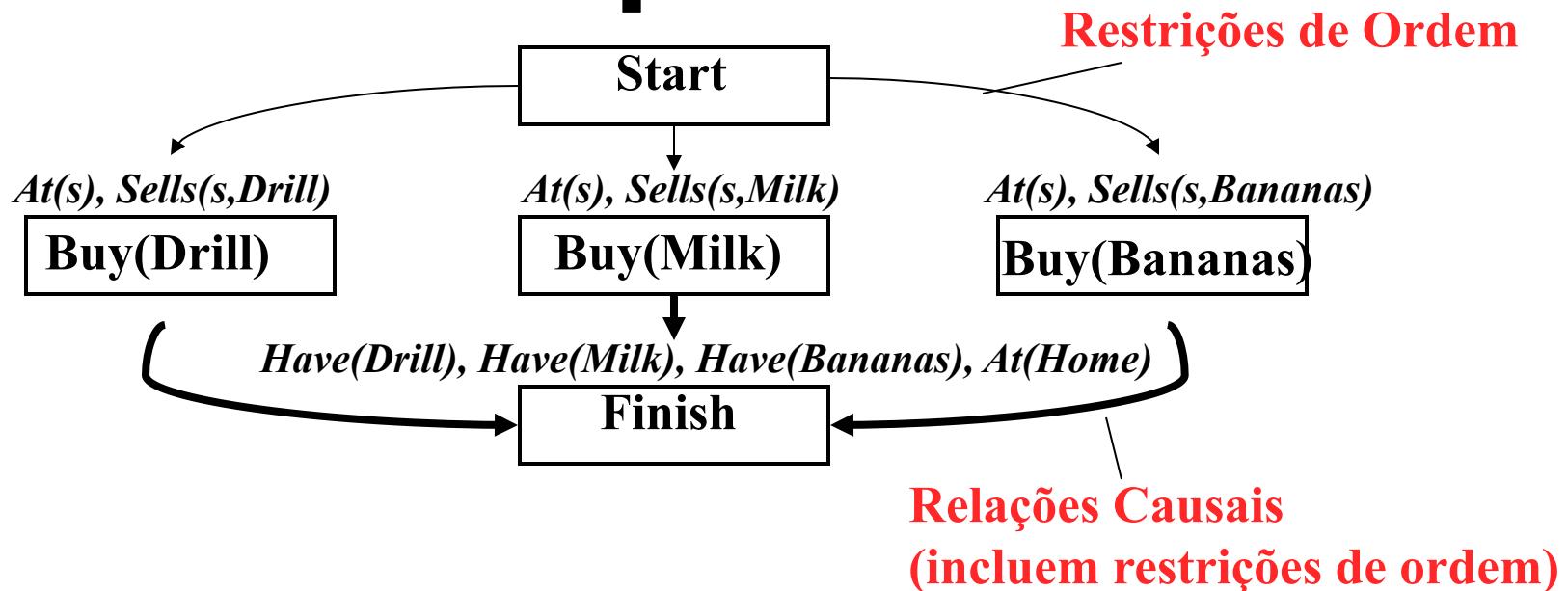
Mais um exemplo

Op(ACTION: Go(y),
PRECOND: At(x),
EFFECT: At(y) \wedge \neg At(x))

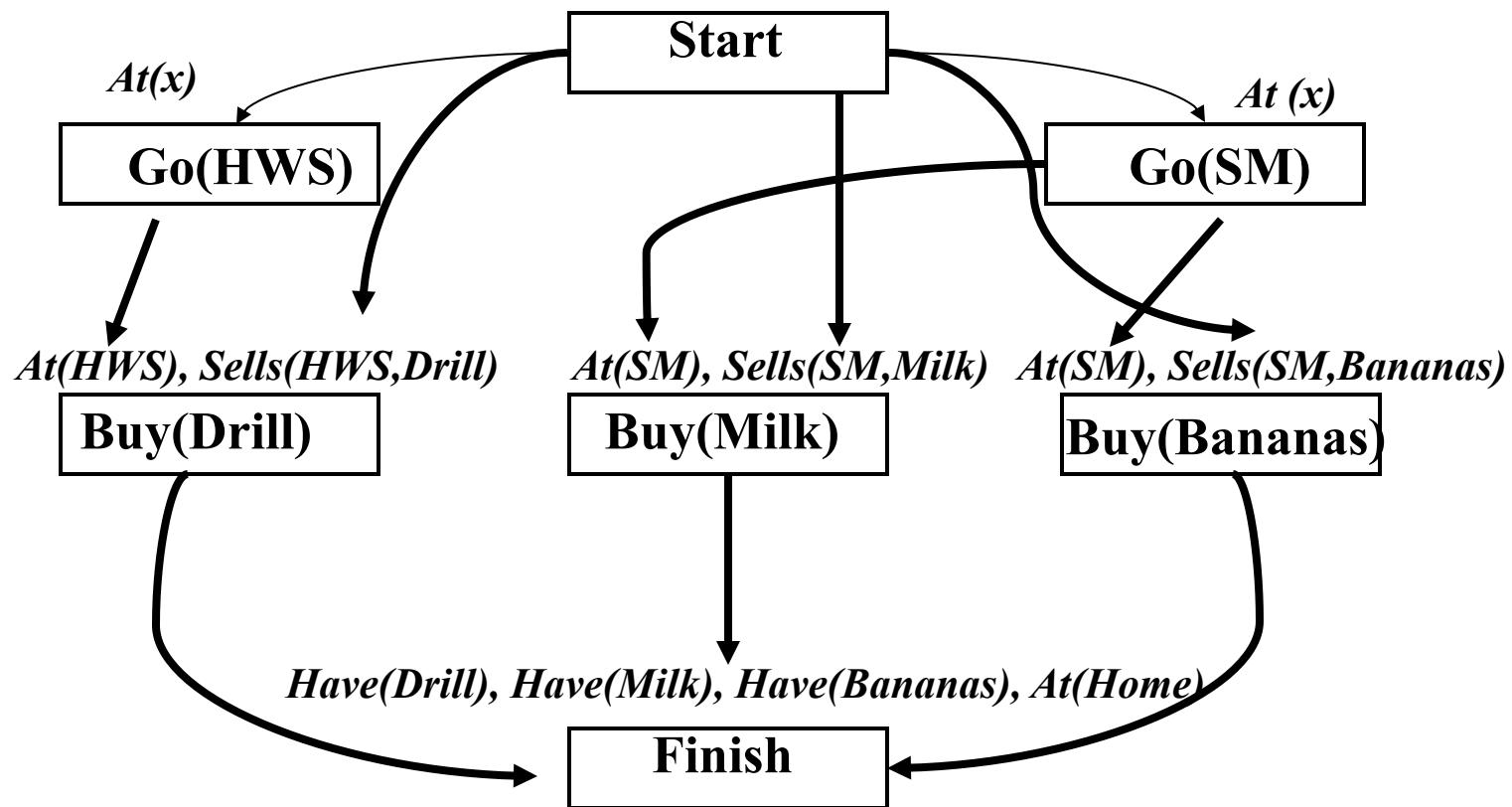
Op(ACTION: Buy(x),
PRECOND: At(s) \wedge Sells(s,x)
EFFECT: Have(x))



Mais um exemplo

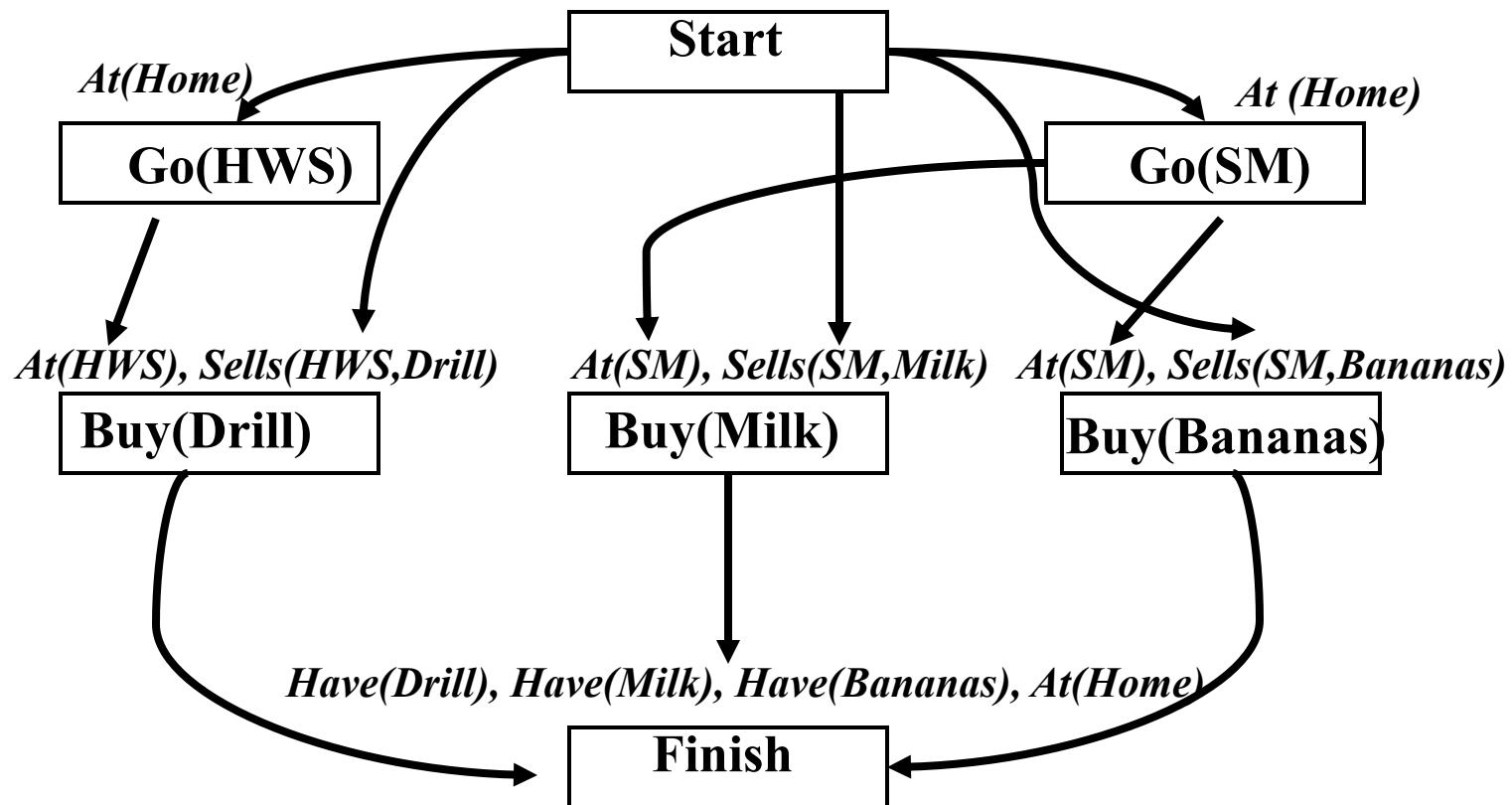


Mais um exemplo

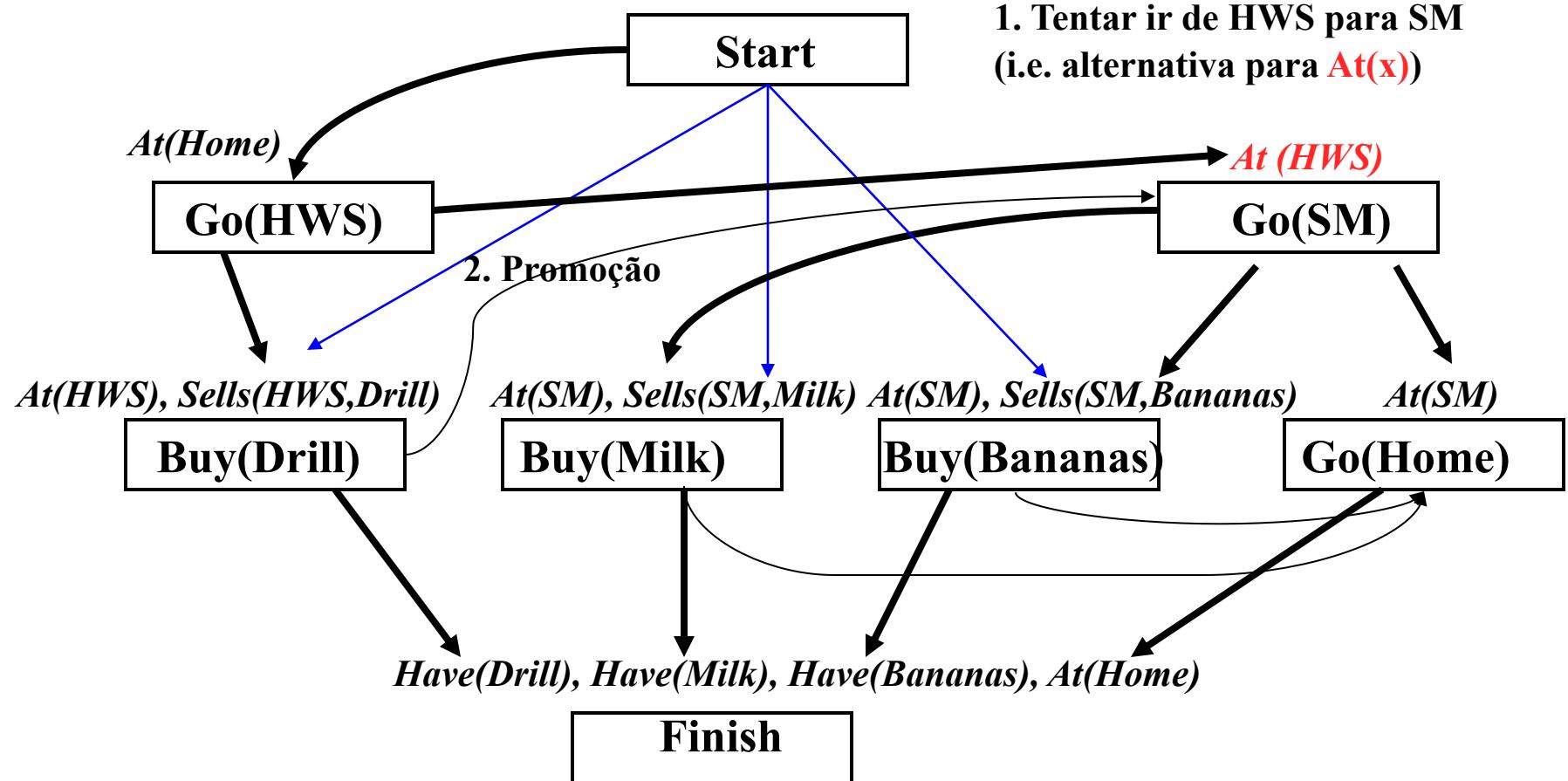


Mais um exemplo

Conflito não se resolve → retroceder e fazer outra escolha

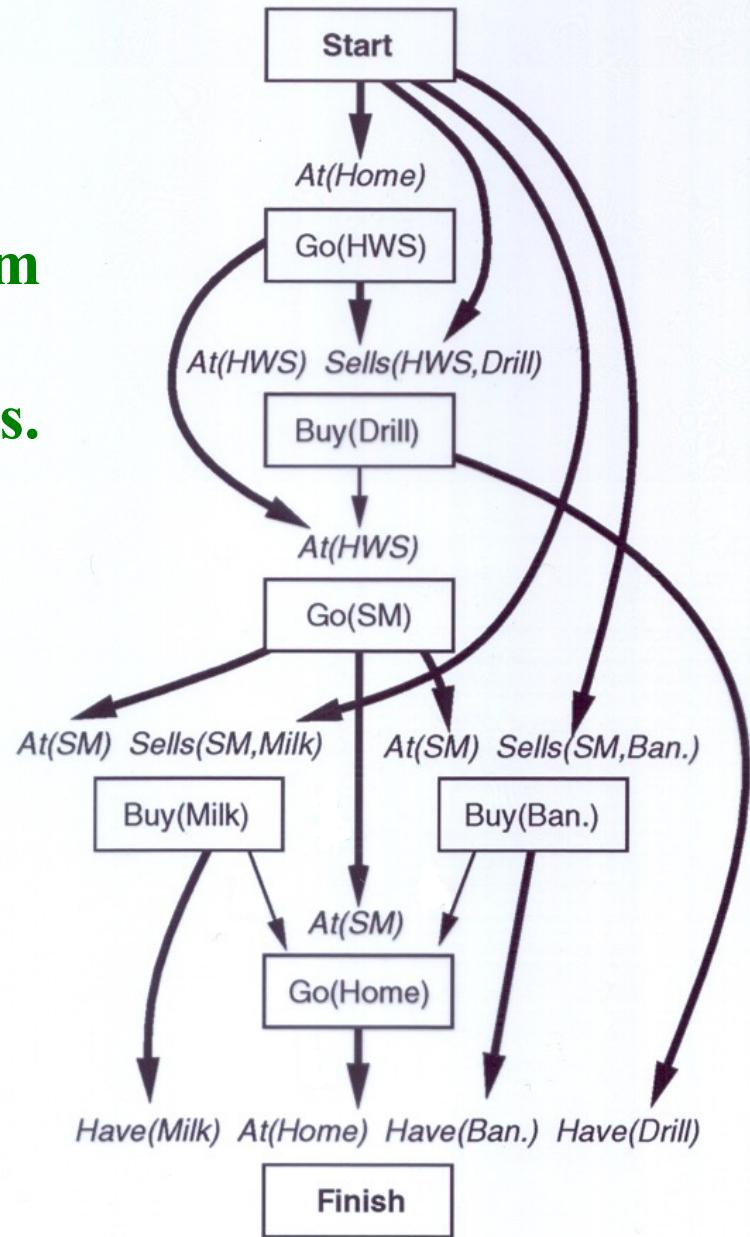


Mais um exemplo



Mais um exemplo

Se usássemos At(HWS) ou At(Home) em vez de At(SM) no final,
os conflitos não poderiam ser resolvidos.



Alguns pormenores ...

- O que acontece quando é usada uma representação em LPO que inclui variáveis?
 - **Complica o processo de detetar e resolver conflitos**
 - **Podem ser solucionados introduzindo restrições de desigualdade**
- CSPs: heurística da variável com mais restrições pode ser usada para os algoritmos de planeamento selecionarem uma PRÉ-CONDIÇÃO