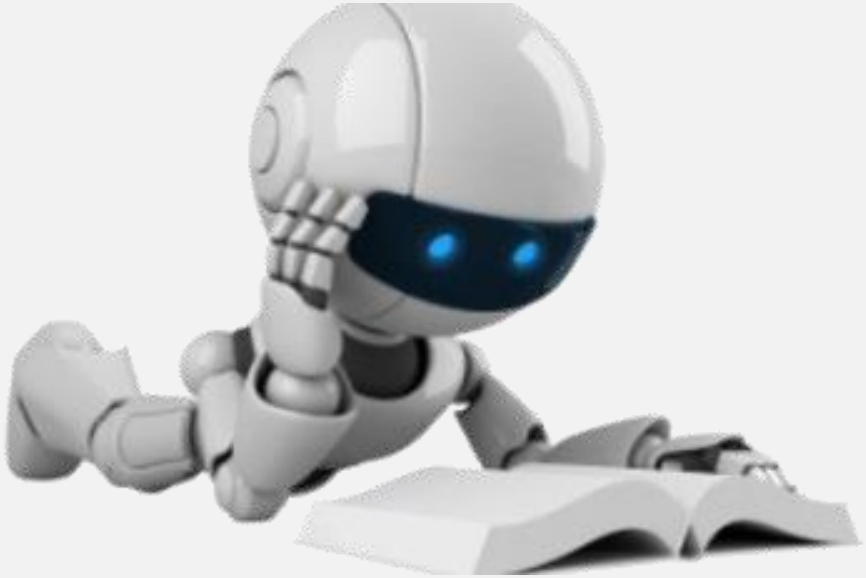
TÉCNICO LISBOA

Planeamento Clássico



TÉCNICO LISBOA

Sumário

- O que é Planeamento (clássico)
- Linguagens Representação para Planeamento
- Planeamento com Procura em Espaço de Estados
  - Procura para a frente
  - Procura para trás
  - Heurísticas para planeamento
- Grafos de Planeamento
- Outras abordagens
  - SATPlan
  - POP Planning

TÉCNICO LISBOA

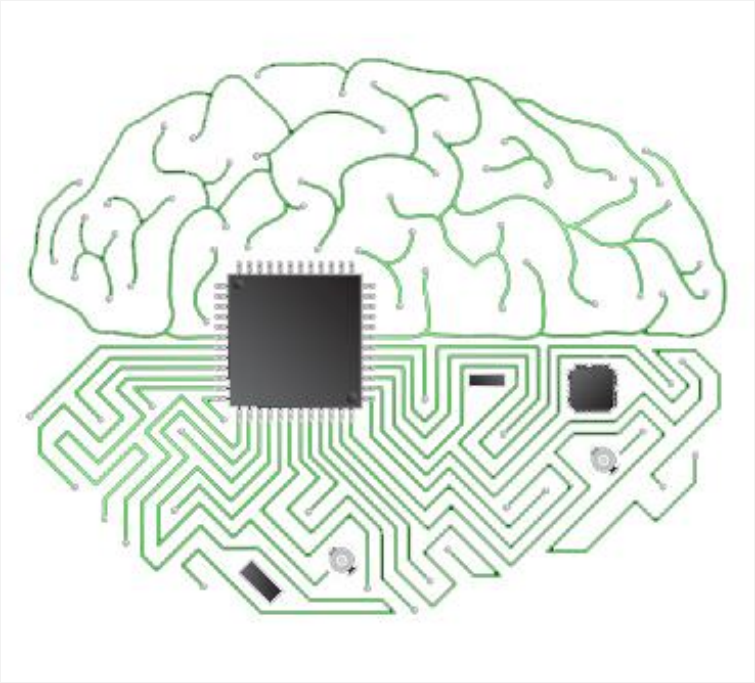
Relação com o livro

- Capítulo 10
  - 10.1,10.2,10.3,10.4.3

As áreas de IA

Robótica

Representação do Conhecimento e Raciocínio



Visão

Procura

Agentes

Planeamento de acções

Língua Natural

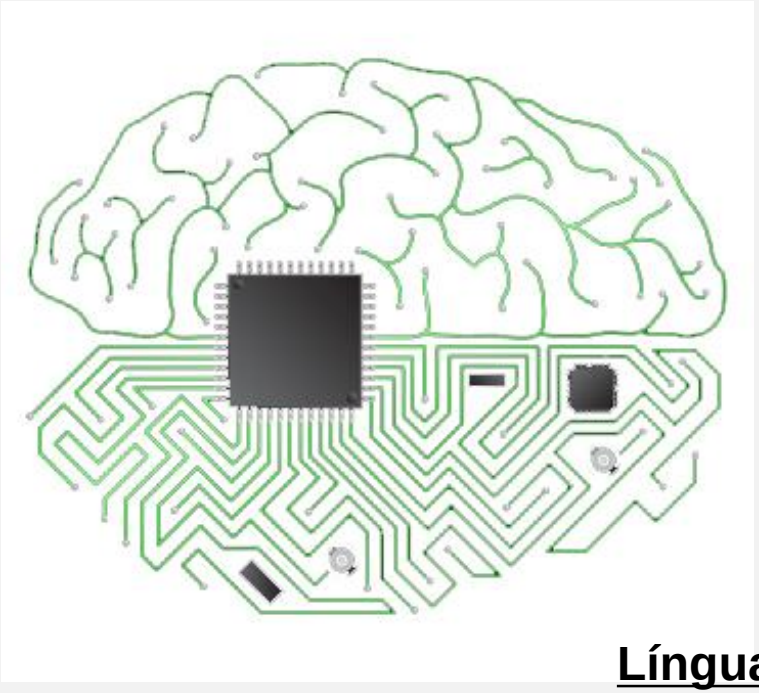
Aprendizagem

Jogos

As áreas de IA

Robótica

Representação do Conhecimento e Raciocínio



Visão

Procura

Agentes

Planeamento de acções

Língua Natural

Aprendizagem

Jogos

TÉCNICO LISBOA

O que é Planeamento?

- Planeamento – construir um plano de acções para atingir um dado estado do mundo (objectivo)
  - Plano – sequência de acções
  - Tem em conta estados, acções e objectivos

TÉCNICO LISBOA

O que é Planeamento?

- Um sistema de planeamento clássico consiste num algoritmo que resolve problemas que são representados por estados e acções em Lógica Proposicional ou Lógica de 1ª Ordem
  - Lógica + Procura
  - Esta representação torna possível a criação de heurísticas eficientes e o desenvolvimento de algoritmos poderosos e flexíveis

TÉCNICO LISBOA

Linguagens 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

TÉCNICO LISBOA

Linguagens Planeamento

- Linguagem **STRIPS**
  - **ST**anford **R**easearch **I**nstitute **P**roblem **S**olver
  - Uma das 1.ª linguagens a ser usadas
  - Usada nas edições anteriores do livro
- Linguagem PDDL
  - **P**lanning **D**omain **D**efinition **L**anguage
  - Usada na 3.ª edição livro
  - Menos restrita que STRIPS
    - Ex: STRIPS não permite literais negativos nas precondições de acções e objectivos
    - PDDL permite

TÉCNICO LISBOA

Linguagens Planeamento

- Representação de estados
- Representação de acções
- Representação de objectivos

TÉCNICO LISBOA

Características gerais da linguagem 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 distinctos correspondem a dois objectos distintos.

TÉCNICO LISBOA

Características gerais da linguagem PDDL

- Acções descritas em termos de pré-condições e efeitos
  - Précondições representam o que se tem de verificar para a acção poder ser executada
  - Efeitos representam os efeitos directos da acção
  - Ao contrário dos estados, podemos usar variáveis na sua representação
- Esquemas de acções ou Operadores
  - Um operador pode corresponder a várias instanciações de acções

TÉCNICO LISBOA

Características gerais da linguagem PDDL

- Representação Esquema de acção/Operador
  - Nome da acção e lista de parâmetros
  - Pré-condição (conj. de literais sem funções):
  - Efeito (conj. de literais sem funções):
    - o que é verdadeiro (P)
    - o que é falso (¬P)
    - conjunção de literais pode ser separado em
      - lista de adições – Add List
      - lista de remoções – Delete List
    - restrição: qualquer variável no efeito deve aparecer também na pre-condição.

Características gerais da linguagem PDDL

Acção(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

Acção(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 gerais da linguagem PDDL

- Representação de objectivos
  - Objectivo representado simplesmente como uma pre-condição.
    - 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 objectivo corresponde a ter qualquer avião em Paris

Características gerais da linguagem PDDL

- Função accoes(s)
  - Quais as acções que podem ser executadas num dado estado s?
  - Uma acção é aplicável em qualquer estado que satisfaça a pré-condição
  - Aplicabilidade de uma acção 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 θ ={*a/A1,de/JFK,para/SFO*}

    Logo a acção *Voar(A1,JFK,SFO)* é aplicável.

Características gerais da linguagem PDDL

- Função resultado(s,a)
  - O resultado de executar uma acção a num estado s é um estado s'
  - s' = Result(a,s) = (s – Del(a) U Add(a))
  - s' é o mesmo que s excepto
    - Qualquer literal positivo *P* no efeito de *a* é adicionado a *s'*
    - Qualquer *P correspondente a um* literal negativo no efeito (¬*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

TÉCNICO LISBOA

Características gerais da linguagem PDDL

- O problema é resolvido quando encontramos uma sequência de acções que acaba num estado s, correspondente a um estado objectivo

- Teste-objectivo(s)
  - Verifica se a pré-condição correspondente ao objectivo definido é consequência lógica de s
  - $s \models obj$
  - Por outras palavras, verifica se s verifica as condições do objectivo

exemplo: transporte aéreo de carga

Início(Em(C1, SFO) ∧ Em(C2,JFK) ∧ Em(A1,SFO) ∧ Em(A2,JFK) ∧ Carga(C1) ∧ Carga(C2) ∧ Avião(A1) ∧ Avião(A2) ∧ Aeroporto(JFK) ∧ Aeroporto(SFO))

Objectivo(Em(C1,JFK) ∧ Em(C2,SFO))

Acção(Carregar(c,a,l)

PRÉ-CONDIÇÃO: *Em(c,l) ∧Em(a,l) ∧Carga(c) ∧Avião(a) ∧Aeroporto(l)*

EFEITO: ¬*Em(c,l) ∧Dentro(c,a))*

Acção(Descarregar(c,a,l)

PRÉ-CONDIÇÃO: *Dentro(c,a) ∧Em(a,l) ∧Carga(c) ∧Avião(a) ∧Aeroporto(l)*

EFEITO: *Em(c,l) ∧* ¬*Dentro(c,a))*

Acção(Voar(a,de,para)

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

EFEITO: ¬ *Em(a,de) ∧ Em(a,para))*

[Possível solução:

Carregar(C1,A1,SFO), Voar(A1,SFO,JFK), Descarregar(C1,A1,JFK), Carregar(C2,A2,JFK), Voar(A2,JFK,SFO), Descarregar(C2,A2,SFO)]

exemplo: pneu sobresselente

Início(Em(Furado, Eixo) $\wedge$ *Em(Sobresselente,Bagageira)* $\wedge$ *Pneu(Furado)* $\wedge$ *Pneu(Sobressalente))*

Objectivo(Em(Sobresselente,Eixo))

Acção(Remove(obj,loc)

PRÉ-CONDIÇÃO: *Em(obj,loc)*

EFEITO: ¬*Em(obj,loc)* $\wedge$ *Em(obj,Chão))*

Acção(Colocar(p,Eixo)

PRÉ-CONDIÇÃO: Pneu(p) $\wedge$ *Em(p,Chão)* $\wedge$ ¬*Em(Furado,Eixo)*

EFEITO: *Em(p,Eixo)* $\wedge$ ¬*Em(p,Chão))*

Acção(DeixarDuranteANoite

PRÉ-CONDIÇÃO:

EFEITO: ¬ *Em(Sobresselente,Chão)* $\wedge$ ¬ *Em(Sobresselente,Eixo)* $\wedge$ ¬*Em(Sobresselente,Bagageira)* $\wedge$ ¬ *Em(Furado,Chão)* $\wedge$ ¬ *Em(Furado,Eixo)* $\wedge$ ¬*Em(Furado,Bagageira)*

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))*

Objectivo(On(A,B) $\wedge$ *On(B,C))*

Acção(Move(b,x,y)

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

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

Acção(MoveToTable(b,x)

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

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

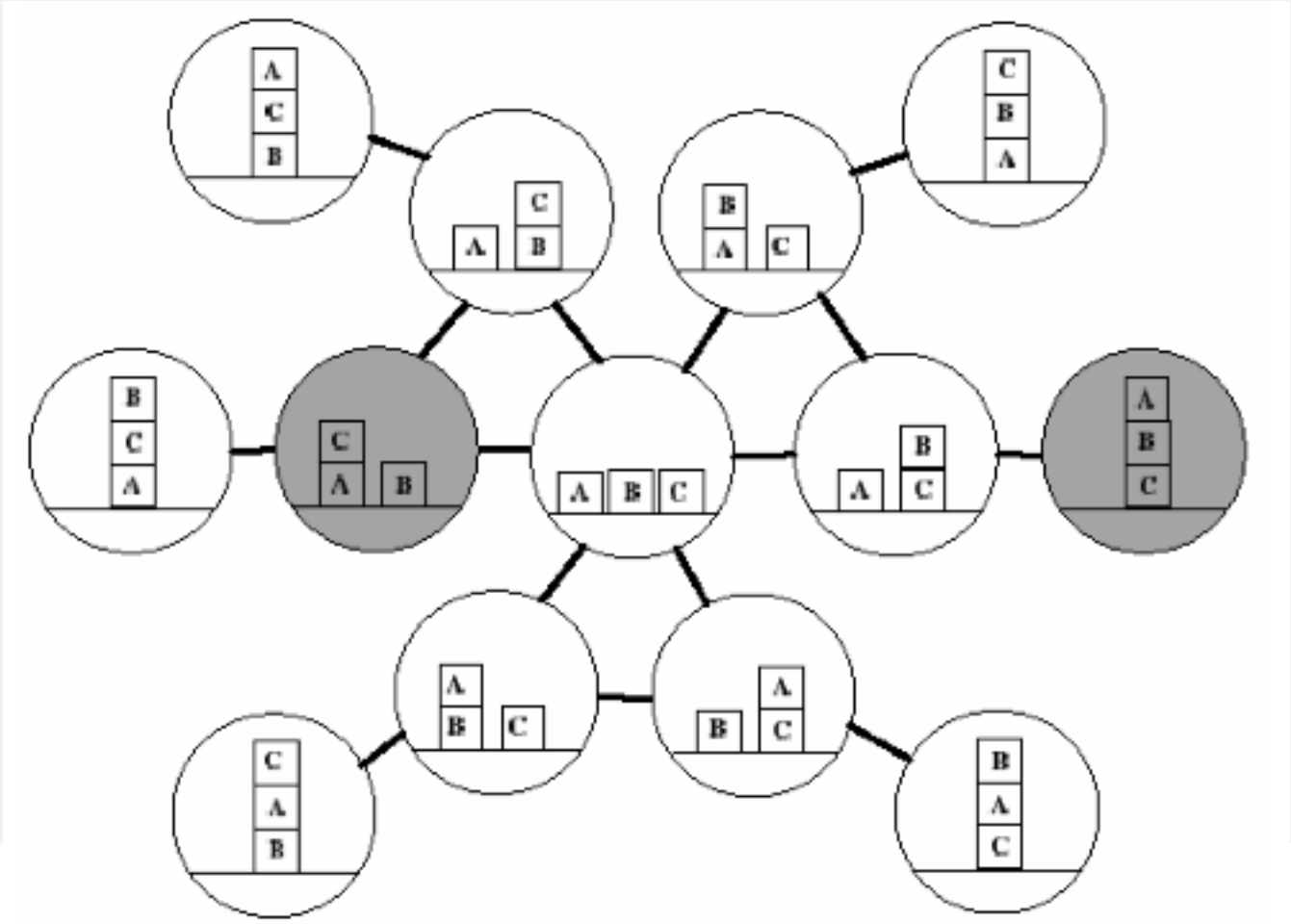
TÉCNICO LISBOA

Planeamento com procura em espaço de estados

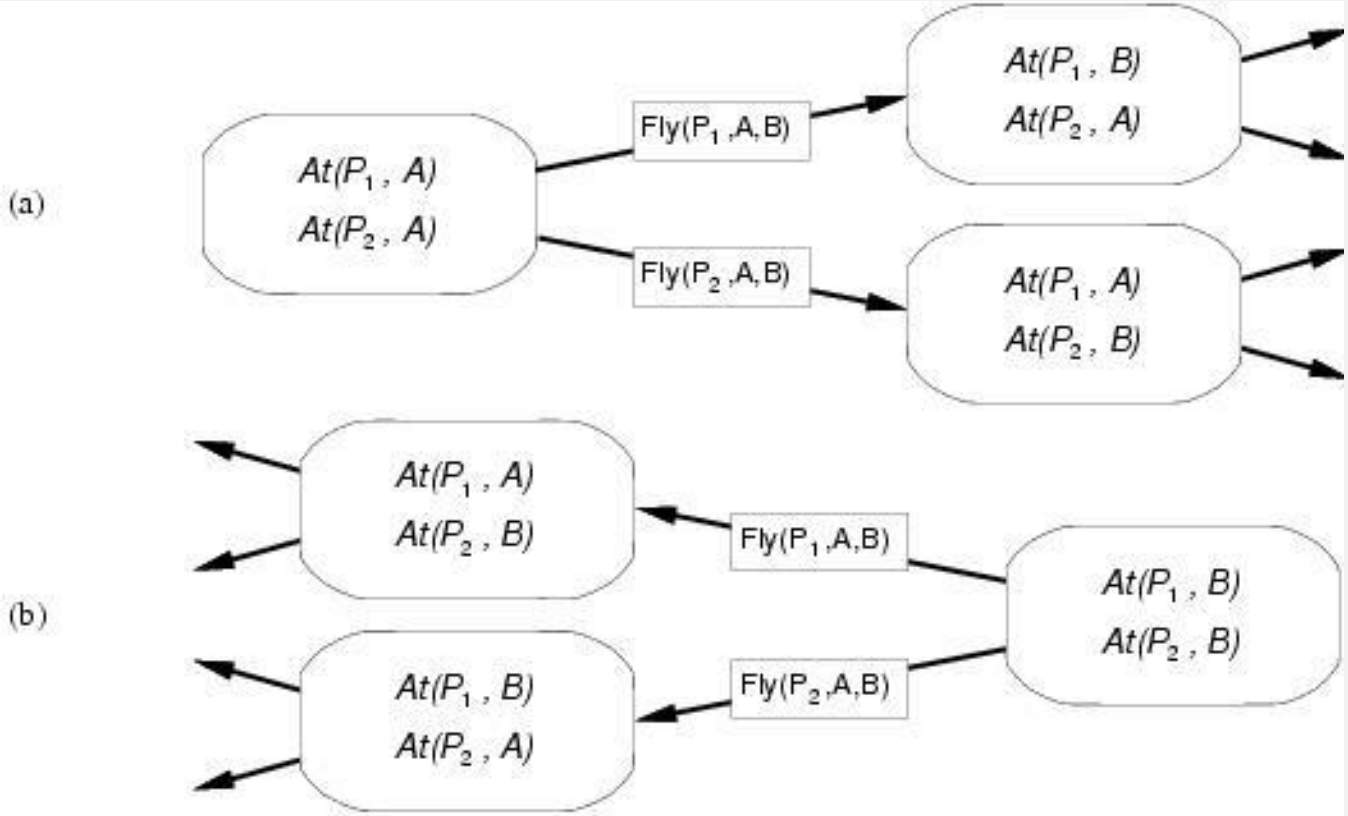
- Podemos ver um problema de planeamento como um problema de procura em espaço de estados
  - Devido à representação declarativa de acções
    - Possibilidade de fazer procura progressiva ou regressiva
- Planeadores progressivos
  - Procura progressiva em espaço de estados
  - Considerar o efeito de todas as acções possíveis num dado estado
- Planeadores regressivos
  - Procura regressiva em espaço de estados
  - Para alcançar um objectivo, considerar o que tem de ser verdadeiro no estado anterior

TÉCNICO LISBOA

Exemplo: espaço de estados para mundo dos blocos



Progressão e regressão



TÉCNICO LISBOA

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
  - Acções = aquelas cujas pré-condições são satisfeitas
    - Adicionar efeitos positivos, remover efeitos negativos
  - Teste objectivo = se o estado verifica o objectivo
  - Custo de transição = cada acção tem custo 1

TÉCNICO LISBOA

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:
  - Acções irrelevantes
    - Ex. Objectivo Ter(AIMA)
    - Acções: Comprar(livro)
      - Compra de um livro com 10 dígitos de ISBN → 10 biliões de nós!!!
  - Factores de ramificação muito elevados
- Eficiência muito dependente de uma boa heurística
  - Felizmente existem boas heurísticas independentes do domínio

TÉCNICO LISBOA

Procura Regressiva

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

TÉCNICO LISBOA

Procura Regressiva

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

TÉCNICO LISBOA

Procura Regressiva

- Formalmente
  - Considerem
    - descrição de um objectivo g que contem um literal g_i
    - Acção A normalizada de modo a produzir A'
  - Se
    - A' tem um efeito e'_j tal que Unify(g_i,e'_j) = θ
    - a' = subst(θ,A')
    - não existe nenhum efeito em a' que é a negação de um literal de g
  - Então
    - a' é uma acção relevante para g

Procura Regressiva

- Exemplo

  Objectivo: Ter(0136042597)
  Acção(Comprar(i),
  Pré-condição:ISBN(i),
  Efeito:Ter(i))

  1) Normalização

  Acção (Comprar(i_1),
  Pré-condição:ISBN(i_1),
  Efeito:Ter(i_1)

  2) Unificação

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

  3) Substituição

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

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

Procura Regressiva

- Como obter os predecessores de um estado objectivo?
  - Dado um objectivo g
  - Dada uma acção a que é relevante (para alcançar objectivos) e consistente (não invalida objectivos já alcançados)

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

  - Quaisquer efeitos positivos de **a** que aparecem em **g** são removidos
  - Cada pré-condição de **a** é adicionada ao objectivo **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 acção a ser executada
    - Mas não sabemos se são verdade ou não antes de ser executada

Procura Regressiva

- Exemplo
  - Estado objectivo = *Em(C1, B)* $\wedge$ *Em(C2, B)* $\wedge$ … $\wedge$ *Em(C20, B)*
  - Acção que tem o primeiro objectivo 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)* $\wedge$ … $\wedge$ *Em(C20, B)*
  - Sub-objectivo *Em(C1,B)* já não está presente neste estado

TÉCNICO LISBOA

Procura Regressiva

- Propriedades:
  - Apenas considera acções relevantes para o objectivo
    - Tipicamente factor 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

TÉCNICO LISBOA

Heurísticas para procura em espaço de estados

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

TÉCNICO LISBOA

Heurística de ignorar pré-condições

- Heurística de ignorar pré-condições
  - Ignore preconditions heuristic
  - Considerar problema de planeamento relaxado onde as acções não têm pré-condições
    - Todas as acções são aplicáveis em qualquer estado
    - Qualquer literal objectivo pode ser atingido com um único passo

TÉCNICO LISBOA

Heurística de ignorar pré-condições

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

TÉCNICO LISBOA

Heurística de ignorar pré-condições

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

TÉCNICO LISBOA

Heurística de ignorar pré-condições

- Podemos remover apenas algumas pre-condições
  - Ex: 8-puzzle

    Acção(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) $\wedge$ ¬Em(p,pos1) $\wedge$ ¬Vazia(pos2))

  - Se removermos pré-cond Vazia(pos2), obtemos a heurística da distância de Manhattan.
  - Heurística derivada automaticamente do esquema de acção.
  - É a grande vantagem da representação utilizada para planeamento.

TÉCNICO LISBOA

Heurística de ignorar lista de remoções

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

TÉCNICO LISBOA

Heurísticas de decomposição

- Heurísticas de decomposição
  - Decompor o objectivo em vários subobjectivos g1,g2,..,gn
  - H = Max(c(g1),c(g2),…,c(gn))
    - Heurística admissível
  - H = c(g1) + c(g2) + … + c(gn)
    - Boa estimativa
    - Mas não garante admissibilidade
      - c(g1) + c(g2) > h* quando a solução para g1 tem acções redundantes com a solução para g2
      - A não ser para subobjectivos g1,g2,…, gn independentes

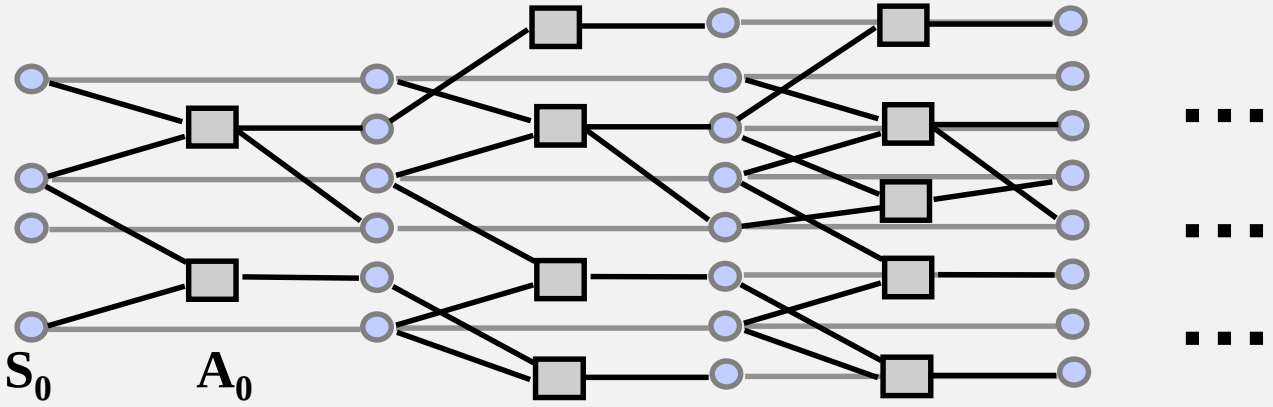
TÉCNICO LISBOA

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 objectivo g

- Solução também pode ser directamente 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 acções
    - *S_i = Literais* ( ) = todos os literais que *podem* ser verdadeiros nesse instante de tempo, dependendo das acções executadas no instante de tempo anterior
    - *A_i = Acções* ( ) = todas as acções que *podem* ter as suas pré-condições satisfeitas nesse instante de tempo, dependendo dos literais verdadeiros nesse instante

TÉCNICO LISBOA

Grafos de planeamento

- Significado de “podem”?
  - Registo de apenas um sub-conjunto restrito de possíveis interacções negativas entre acções
- Funciona apenas para problemas proposicionais
  - Isto é, sem variáveis
- Exemplo em PPDL:

  Init(Have(Cake))

  Goal(Have(Cake) $\wedge$ Eaten(Cake))

  Action(Eat(Cake),

  PRECOND: Have(Cake)

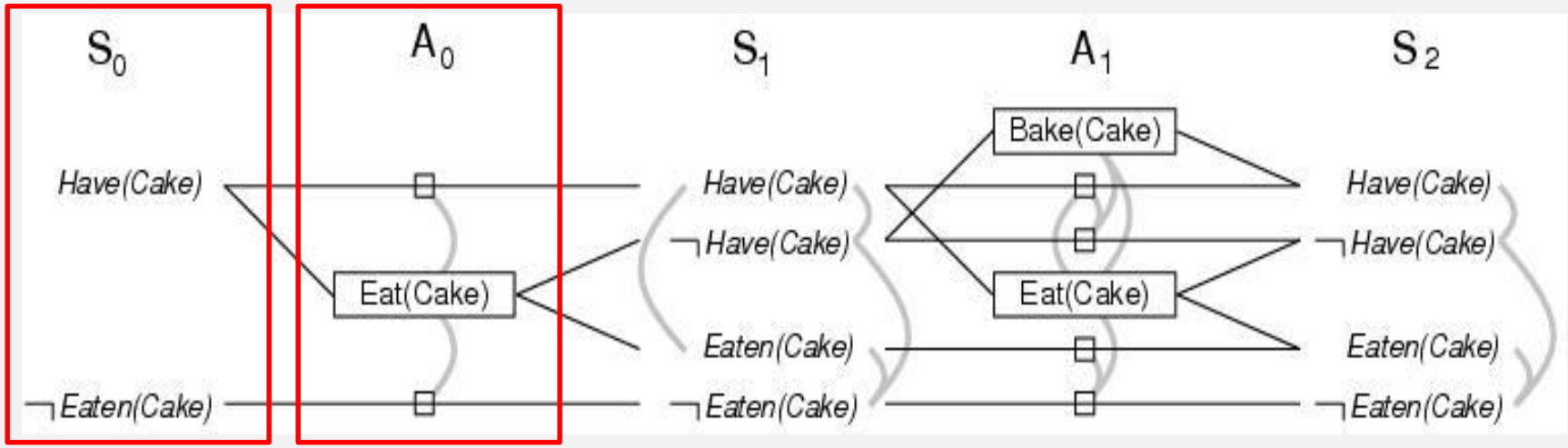
  EFFECT: ¬Have(Cake) $\wedge$ Eaten(Cake))

  Action(Bake(Cake),

  PRECOND: ¬Have(Cake)

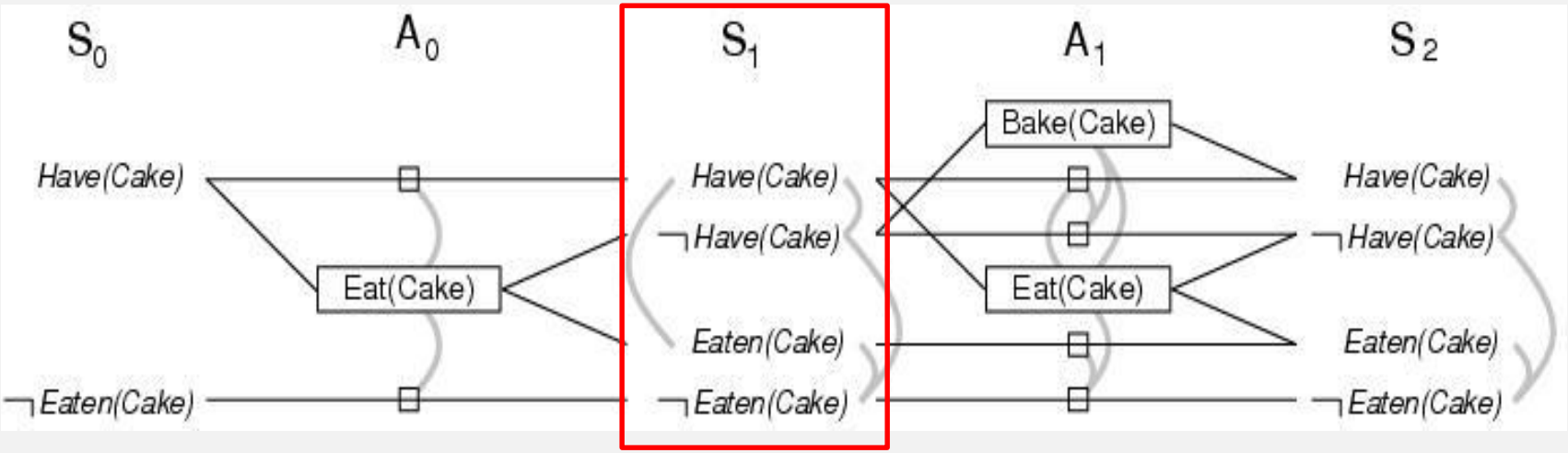
  EFFECT: Have(Cake))

Exemplo do bolo



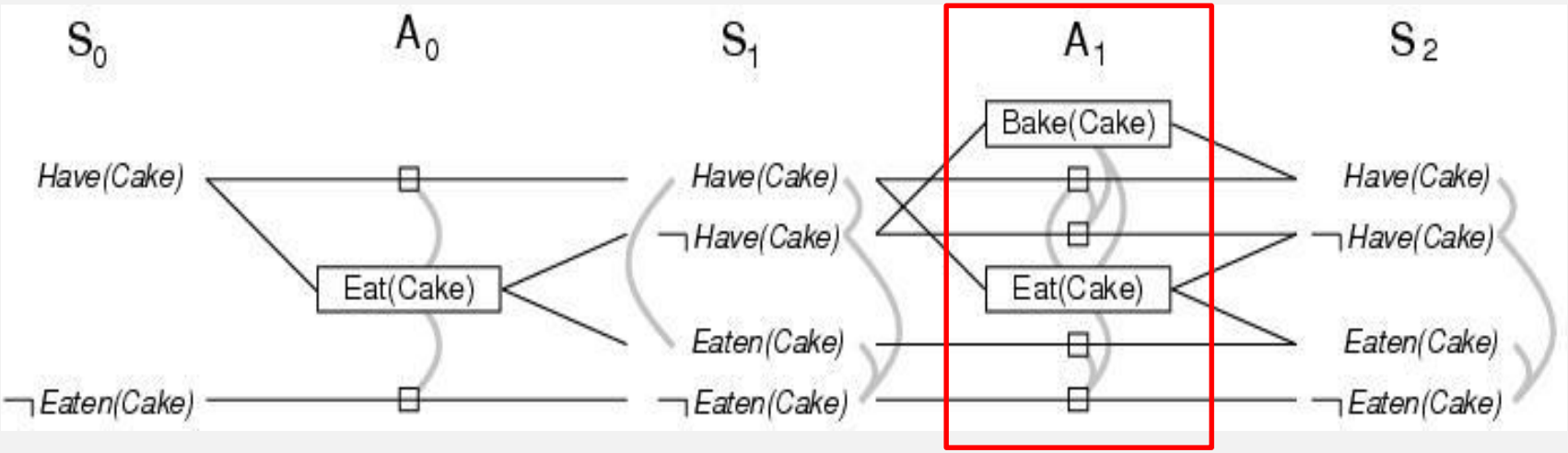
- Início em S_0
  - Representa o estado inicial
- A_0 contém as acções cujas pré-condições são satisfeitas por S_0
  - **Inacção é representada pela persistência de acções (□)**
  - Para cada condição c em S criar acção com precondição e efeito c
- Conflitos entre acções são representadas por relações *mutex*
  - Representadas pelas linhas curvas a cinzento
  - Representam exclusividade mútua: neste caso acção Eat(Cake) tem como efeitos ¬Have(Cake) ∧ Eaten(Cake)

Exemplo do bolo



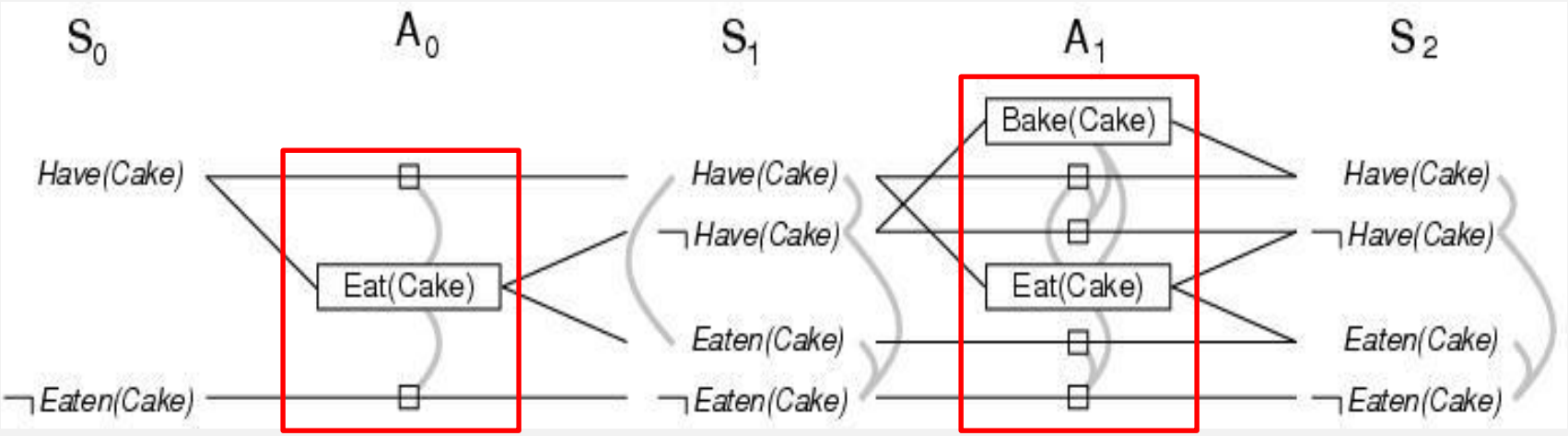
- S_1 contém todos os literais resultantes das acções em A_0
- S_1 contém também relações mutex
  - Contradições: ¬Have(Cake) e Have(Cake), ¬Eaten(Cake) e Eaten(Cake)
  - Outros casos: Have(Cake) e Eaten(Cake), ¬Have(Cake) e ¬Eaten(Cake)
    - Consequência de só poder ser escolhida uma acção em A_0

Exemplo do bolo



- Em A1 podem ter lugar as duas acções
- Relações *mutex* em A1
  - **Contradições: ¬Have(Cake) e Have(Cake), ¬Eaten(Cake) e Eaten(Cake)**
  - **Outros casos: Bake(Cake) e ¬Have(Cake), Bake(Cake) e Eat(Cake), ¬Eaten(Cake) e Eat(Cake)**

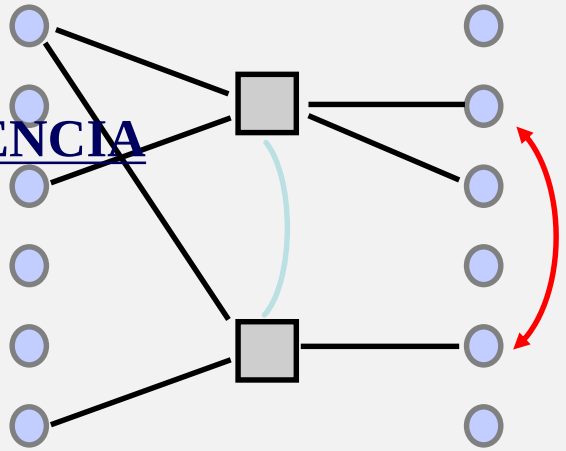
Exemplo do bolo



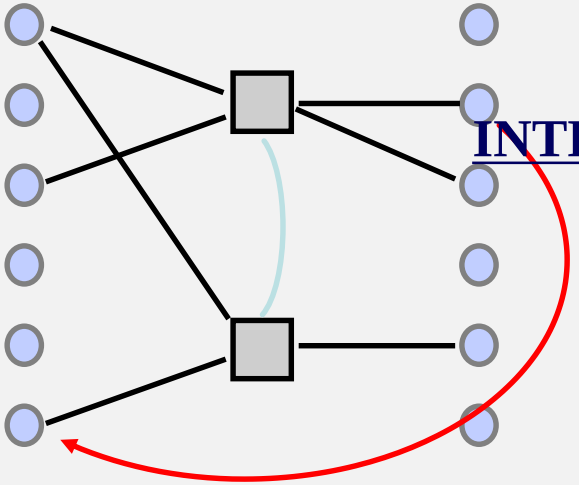
- Uma relação *mutex* é estabelecida entre **duas acções** quando:
  - *Inconsistência*: uma acção nega o efeito da outra e.g. acções Eat(Cake) e Have(Cake)
  - *Interferência*: um dos efeitos de uma acção é a negação da pré-condição de outra e.g. acções Eat(Cake) e Have(Cake)
  - *Competição*: uma das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

Relações mutex: resumo

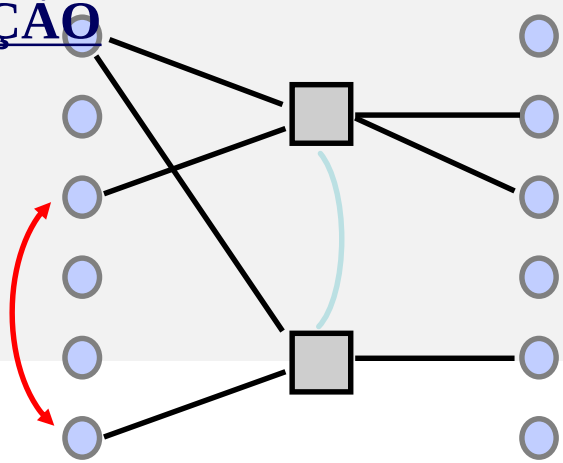
INCONSISTÊNCIA
uma acção nega o efeito da outra



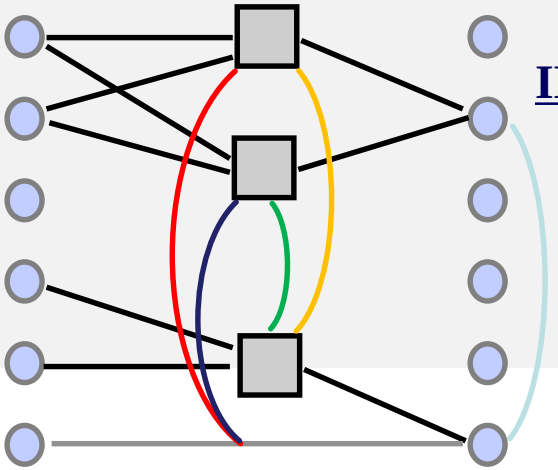
INTERFERÊNCIA
um efeito nega uma pré-condição



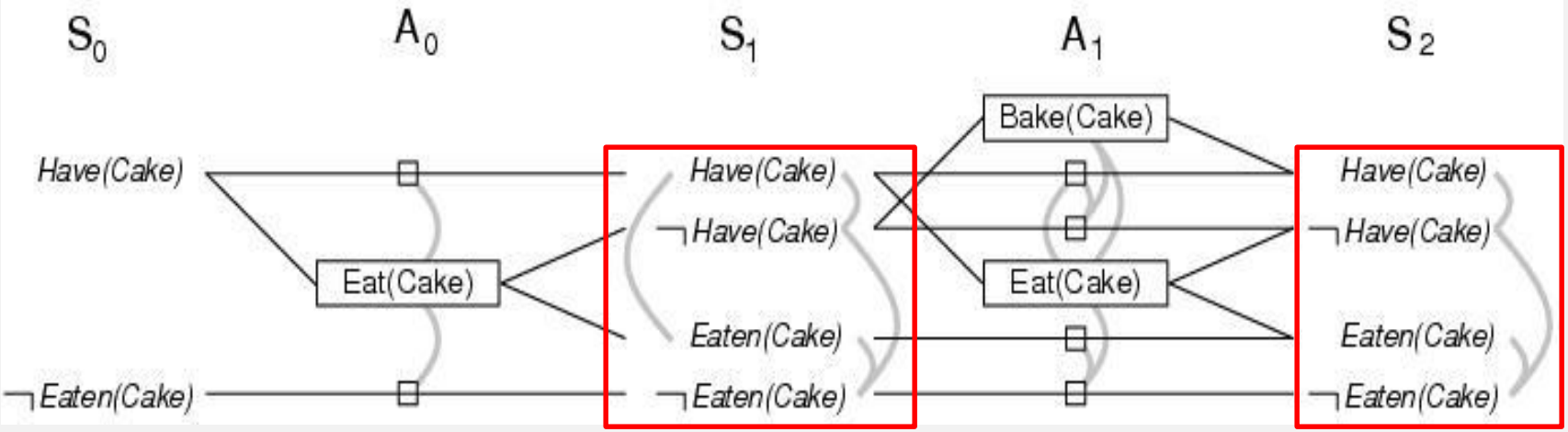
COMPETIÇÃO
uma pré-condição nega outra



SUPORTE INCONSISTENTE
literais são efeito de pares de acções mutex

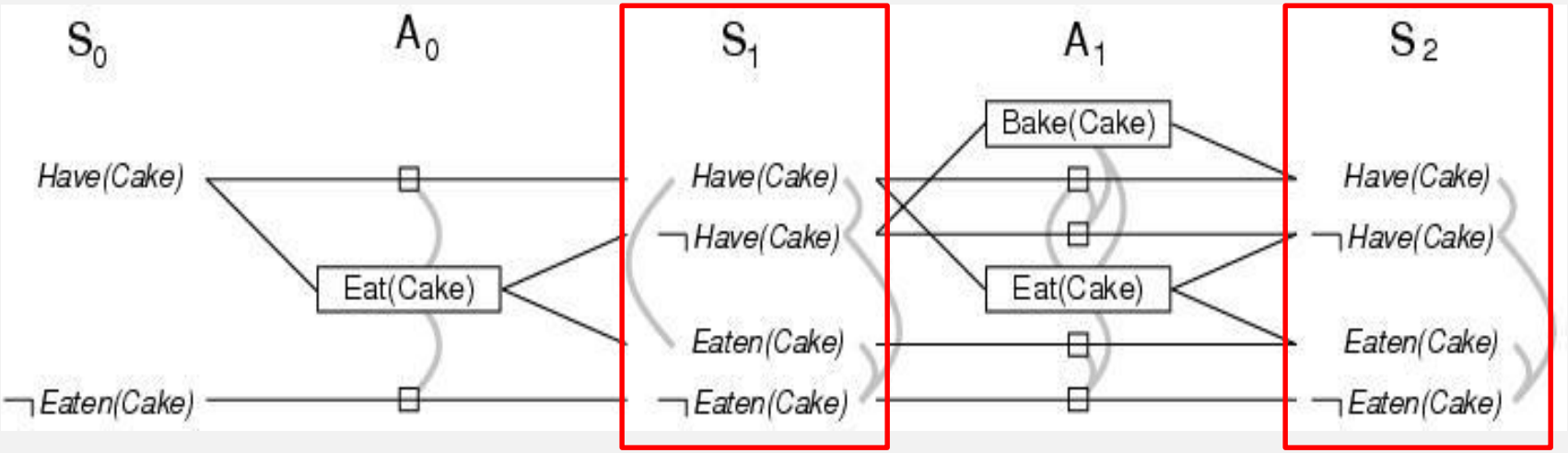


Exemplo do bolo



- Relação *mutex* entre **dois literais** (*suporte inconsistente*) quando:
  - Um é a negação do outro
  - Cada par de acções 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) (acção Have(Cake) □) é incompatível com a única forma de obter Eaten(Cake) (acção Eat(Cake))
  - Em S_2: literais Have(Cake) e Eaten(Cake) já são possíveis porque podem ser adquiridos por duas acções compatíveis (Bake(Cake) e Eaten(Cake))

Exemplo do bolo



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

TÉCNICO LISBOA

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 objectivo 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

TÉCNICO LISBOA

Grafos de planeamento e heurísticas

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

TÉCNICO LISBOA

Grafos de planeamento e heurísticas

- Como estimar o custo de uma conjunção de literais objectivo?
  - Heurística de máximo nível (max-level heuristic)
    - Maximo do custo de nível de um objectivo
    - Admissível, mas não necessariamente a mais correcta
  - Heurística de soma de níveis (level-sum heuristic)
    - Soma do custo de nível de cada objectivo
    - 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 objectivo aparecem no grafo de planeamento sem qualquer par deles ser mutex
    - Admíssivel
    - Domina heurística máximo nível

TÉCNICO LISBOA

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 acções 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

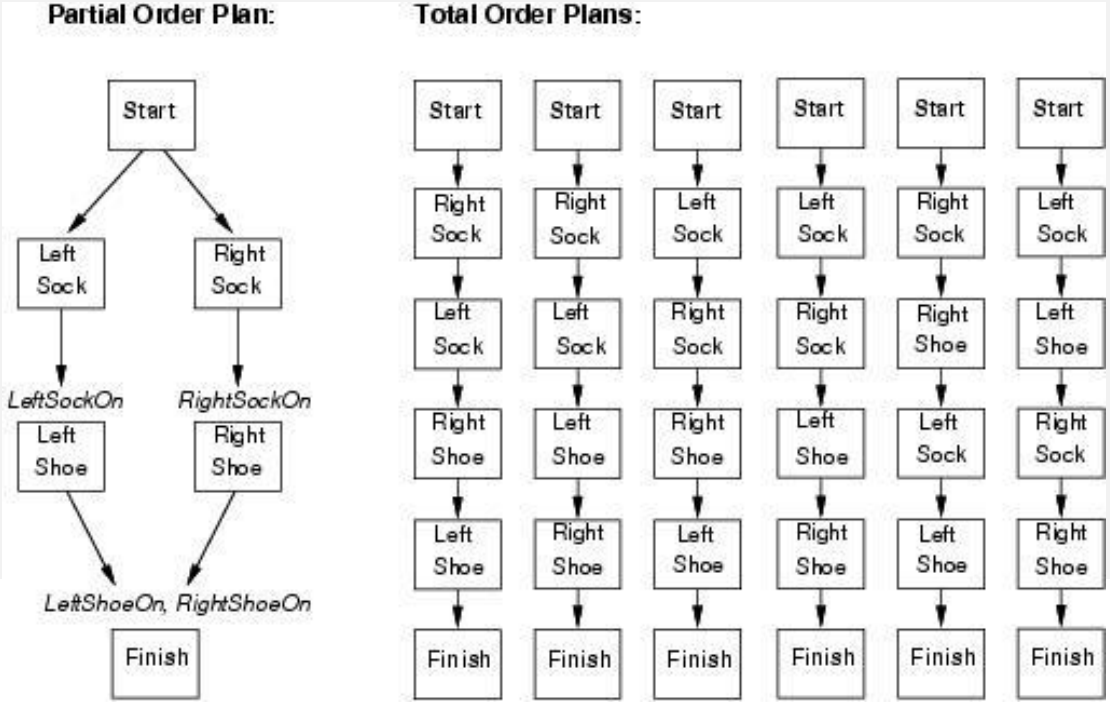
Exemplo dos sapatos

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 acções (1) leftsock, leftshoe (2) rightsock, rightshoe

Planeamento de ordem parcial(POP)

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



POP como problema de procura

- Estados são (tipicamente) planos inacabados
  - Plano vazio contém apenas acções iniciar e terminar
- Cada plano tem 4 componentes:
  - Um conjunto de acções (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 acção A atinge a precondição p para B
    - Ligação protegida
      - Não podemos adicionar nenhuma acção entre A e B que tenha como efeito ¬p
  - Um conjunto de pré-condições abertas
    - Pré-condições que ainda não foram alcançadas através de uma acção

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 acções possíveis
  - Esta flexibilidade é vantajosa em ambientes não cooperativos

TÉCNICO LISBOA

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 acção *B*
    - Gerar um plano sucessor para todas as formas consistentes de satisfazer p
  - Teste objectivo
    - Plano consistente sem precondições abertas

TÉCNICO LISBOA

Algoritmo POP

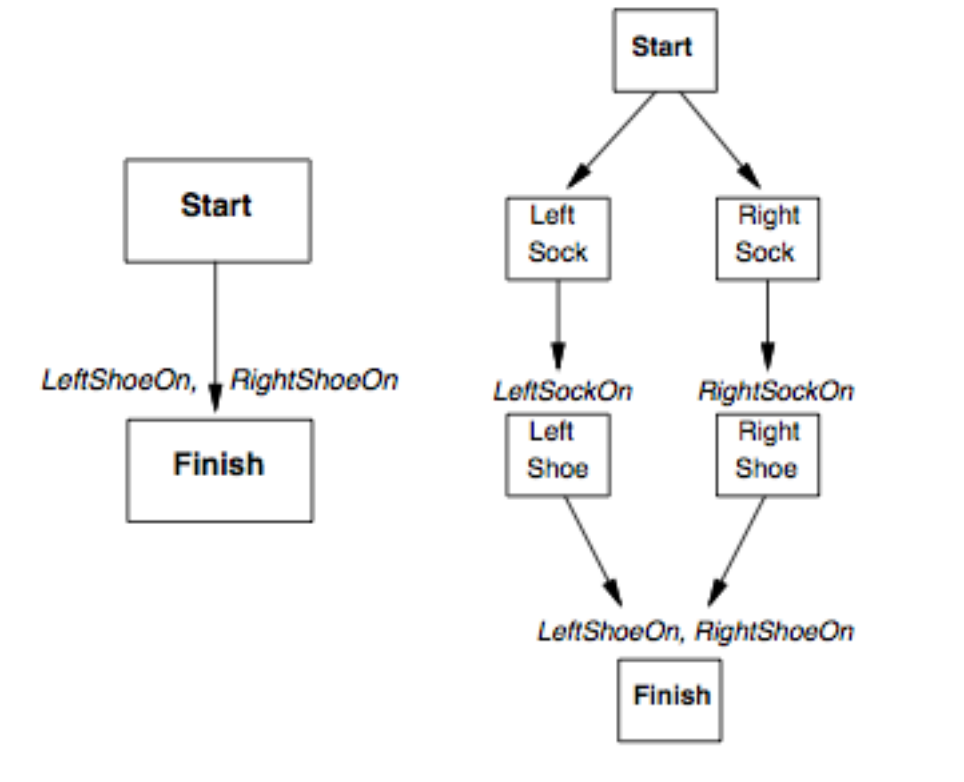
- Satisfazer uma precondição p
  1. Verificar se a precondição p já é verificada no estado inicial
  2. Tentar encontrar uma acção do plano que satisfaça p
  3. Adicionar uma nova acção ao plano que satisfaça p

TÉCNICO LISBOA

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 acções existentes
  - Resolver conflitos entre a acção A (se for nova) e todas as ligações causais existentes

Exemplo: sapatos e peúgas



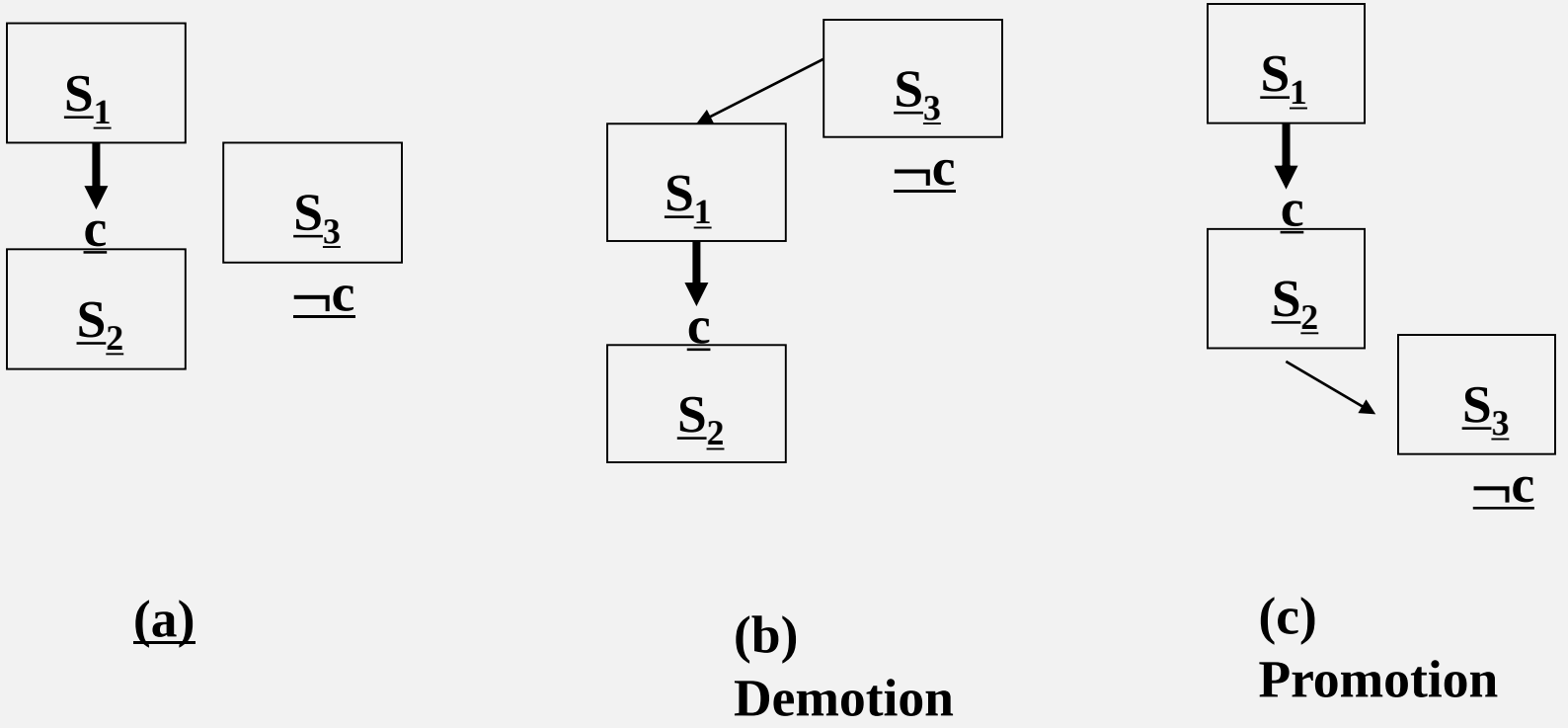
Resolução de conflitos

- Conflito:
  - Existe uma acção C no plano com ¬p que entra em conflicto 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



TÉCNICO LISBOA

Algoritmo POP

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

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

TÉCNICO LISBOA

Algoritmo POP (cont.)

procedure CHOOSE-OPERATOR($plan$, $operators$, S_{need}, c)

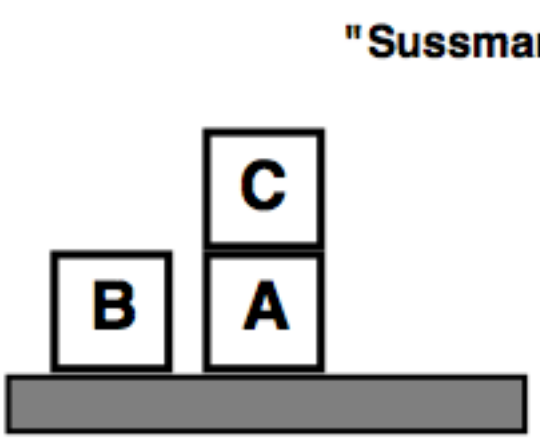
choose a step S_{add} from $operators$ or STEPS($plan$) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS($plan$)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS($plan$)
if S_{add} is a newly added step from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

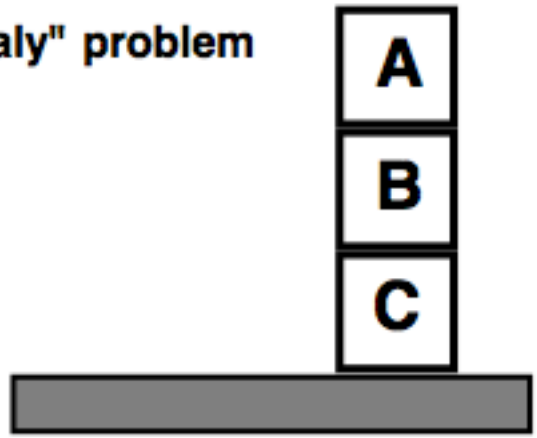
---

procedure RESOLVE-THREATS($plan$)

for each S_{threat} 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





Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)

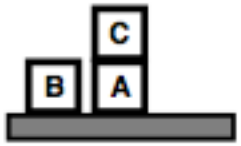
PutOnTable(x)

~On(x,z) Clear(z) On(x,Table)

Exemplo: mundo dos blocos

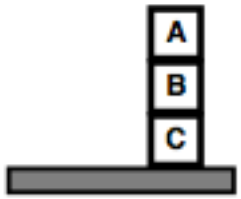
START

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

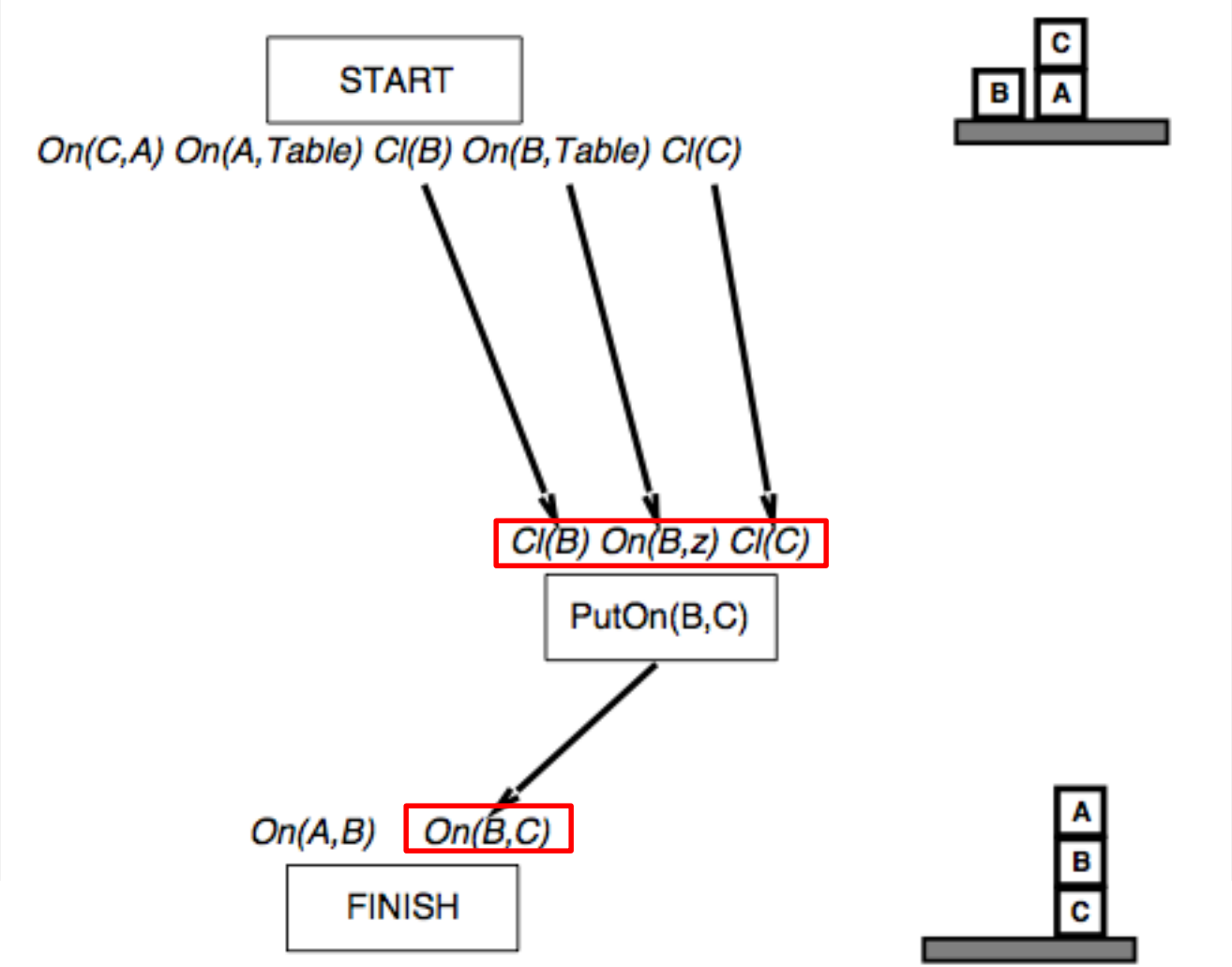


On(A,B) On(B,C)

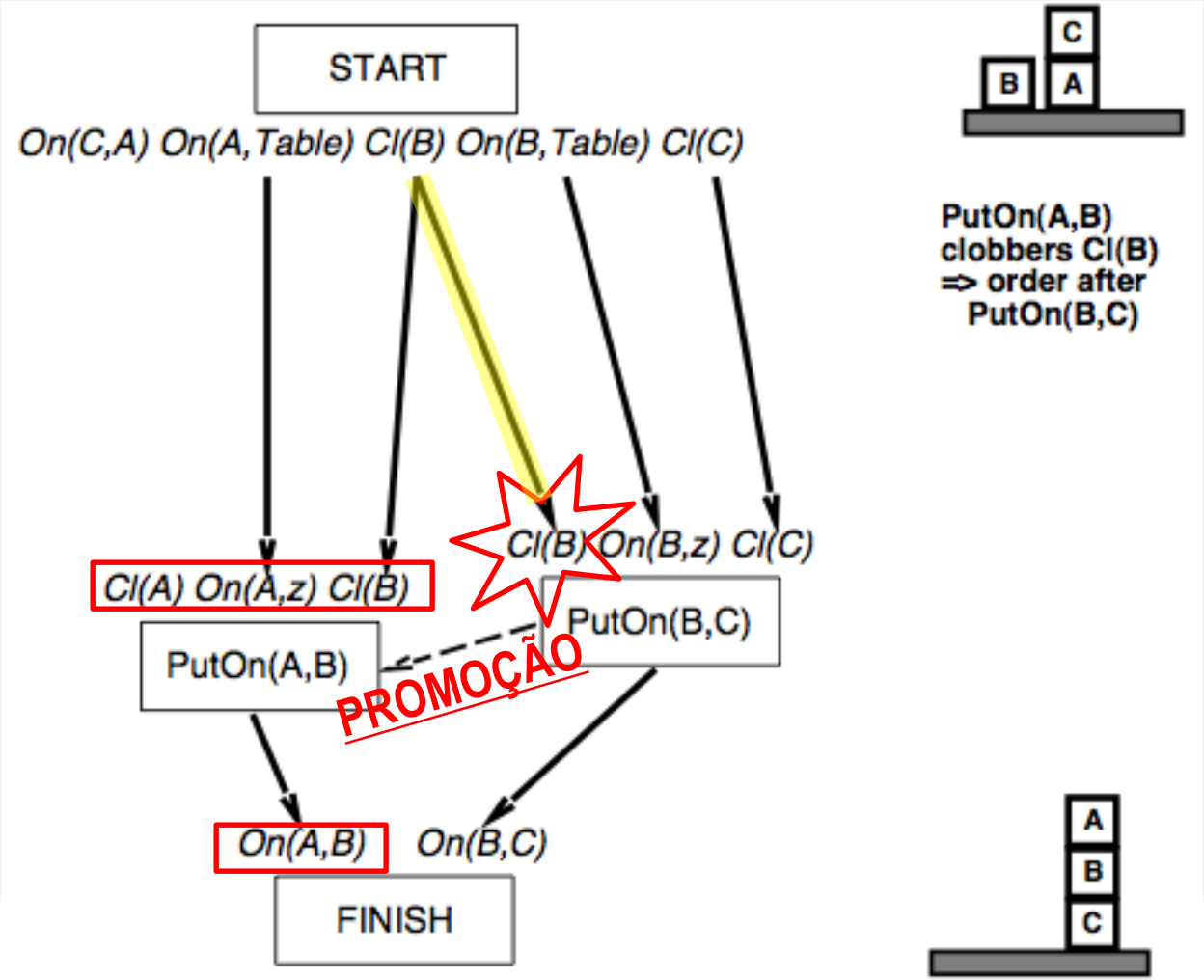
FINISH



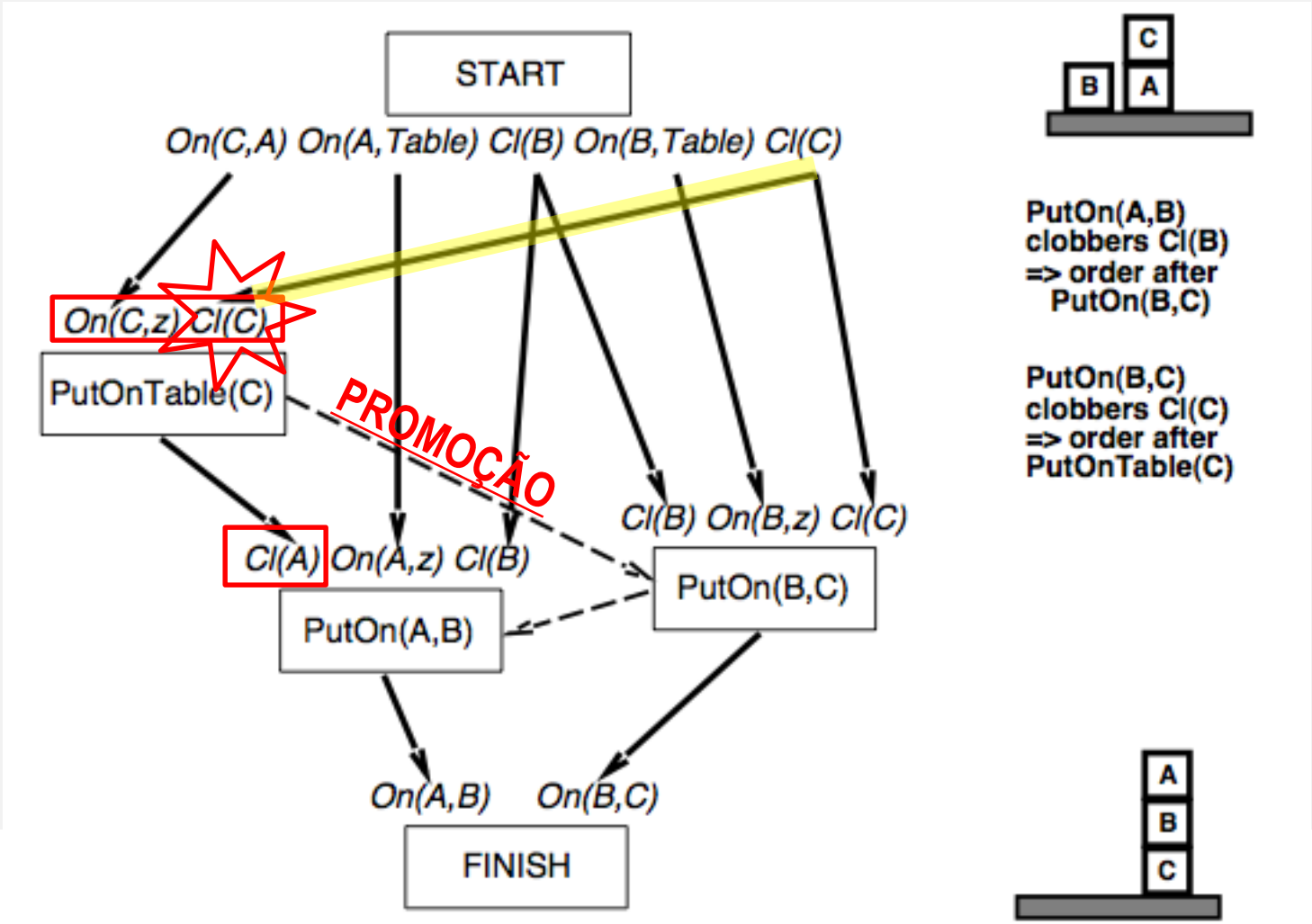
Exemplo: mundo dos blocos



Exemplo: mundo dos blocos



Exemplo: mundo dos blocos



Exemplo: Pneu sobresselente

Init(At(Flat, Axle) ∧ At(Spare,Trunk))

Goal(At(Spare,Axle))

Action(Remove(Spare,Trunk)

PRECOND: *At(Spare,Trunk)*

EFFECT: *¬At(Spare,Trunk) ∧ At(Spare,Ground))*

Action(Remove(Flat,Axle)

PRECOND: *At(Flat,Axle)*

EFFECT: *¬At(Flat,Axle) ∧ At(Flat,Ground))*

Action(PutOn(Spare,Axle)

PRECOND: *At(Spare,Ground) ∧¬At(Flat,Axle)*

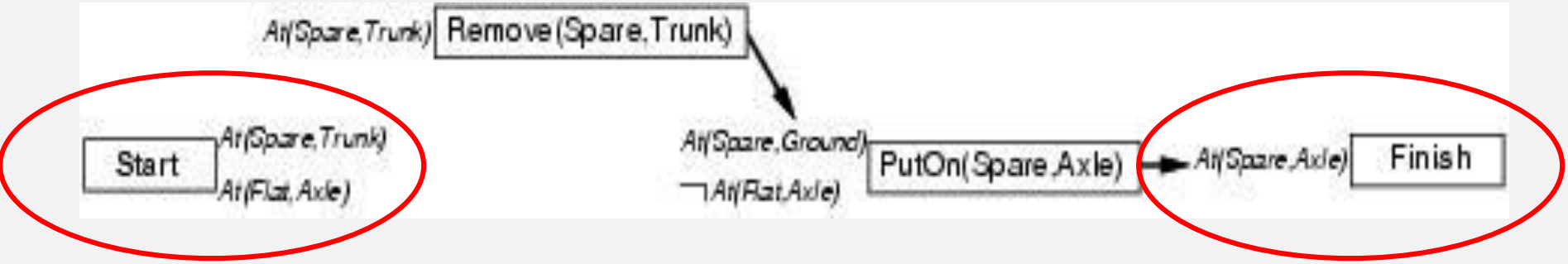
EFFECT: *At(Spare,Axle) ∧ ¬At(Spare,Ground))*

Action(LeaveOvernight

PRECOND:

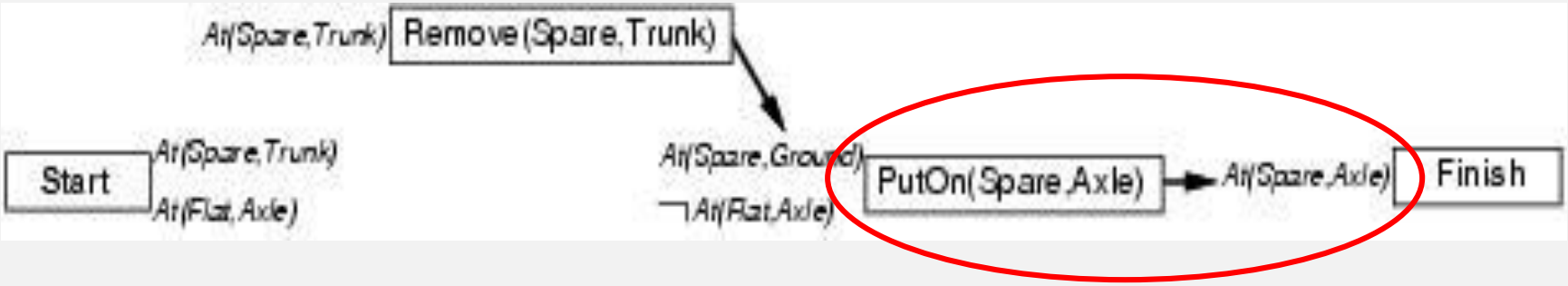
EFFECT: *¬ At(Spare,Ground) ∧ ¬ At(Spare,Axle) ∧ ¬ At(Spare,Trunk) ∧ ¬ At(Flat,Ground) ∧ ¬ At(Flat,Axle))*

Resolução do problema



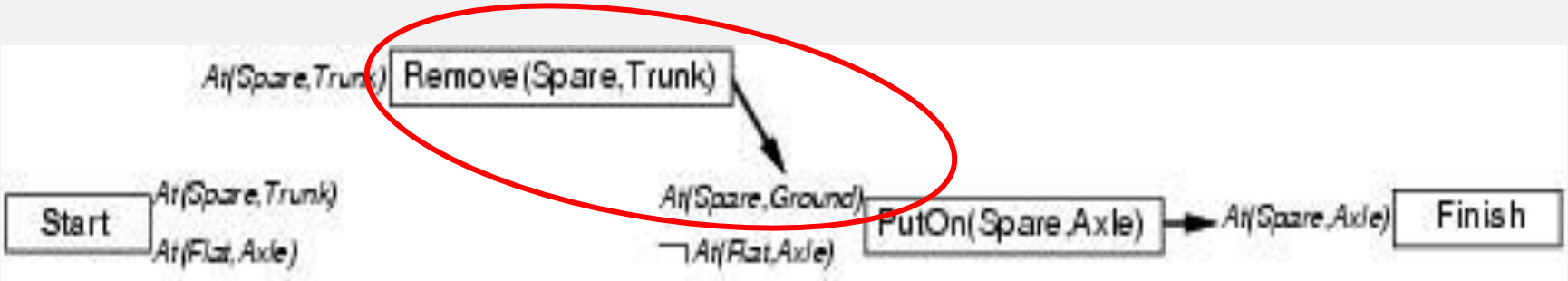
- Plano inicial:
  - *START* com EFEITO = Estado Inicial
  - *Finish* com PRÉ-CONDIÇÃO = Objectivo

Resolução do problema



- Plano inicial: *Start* com EFEITO e *Finish* com PRÉ-CONDIÇÃO
- Escolher uma pré-condição aberta: *At(Spare, Axle)*
- Somente a acção *PutOn(Spare, Axle)* é aplicável
- Adicionar ligação causal: $PutOn(Spare, Axle) \xrightarrow{At(Spare,Axle)} Finish$
- Adicionar restrição de ordem: *PutOn(Spare, Axle)* < *Finish*

Resolução do problema

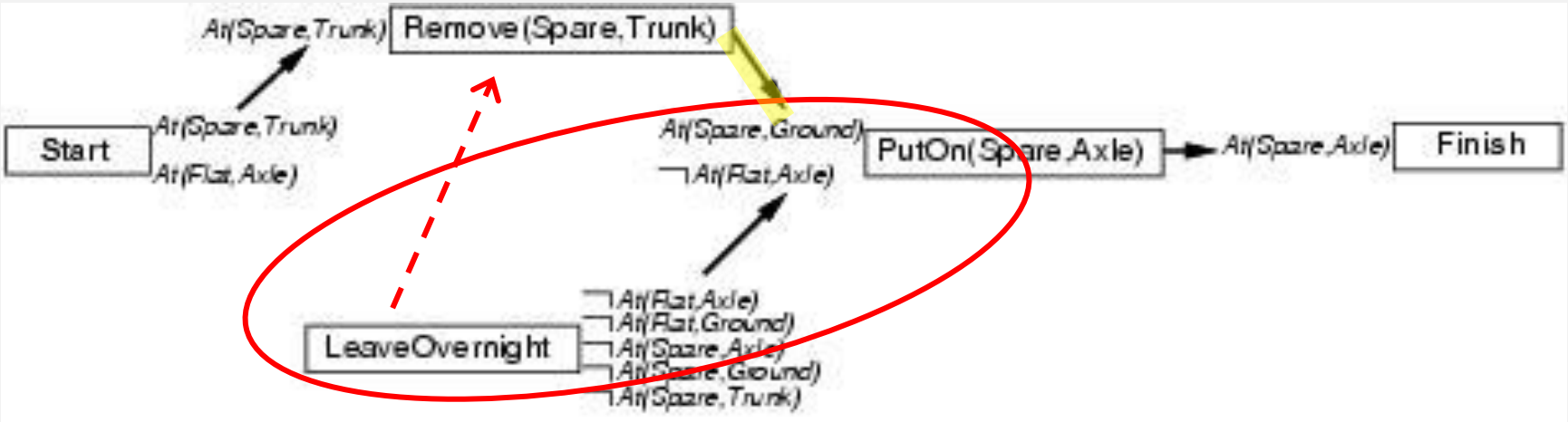


- Escolher uma pré-condição aberta: *At(Spare, Ground)*
- Somente *Remove(Spare, Trunk)* é aplicável
- Adicionar ligação causal:

$$\text{Re}\,move(Spare, Trunk) \xrightarrow{At(Spare,Ground)} PutOn(Spare, Axle)$$

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

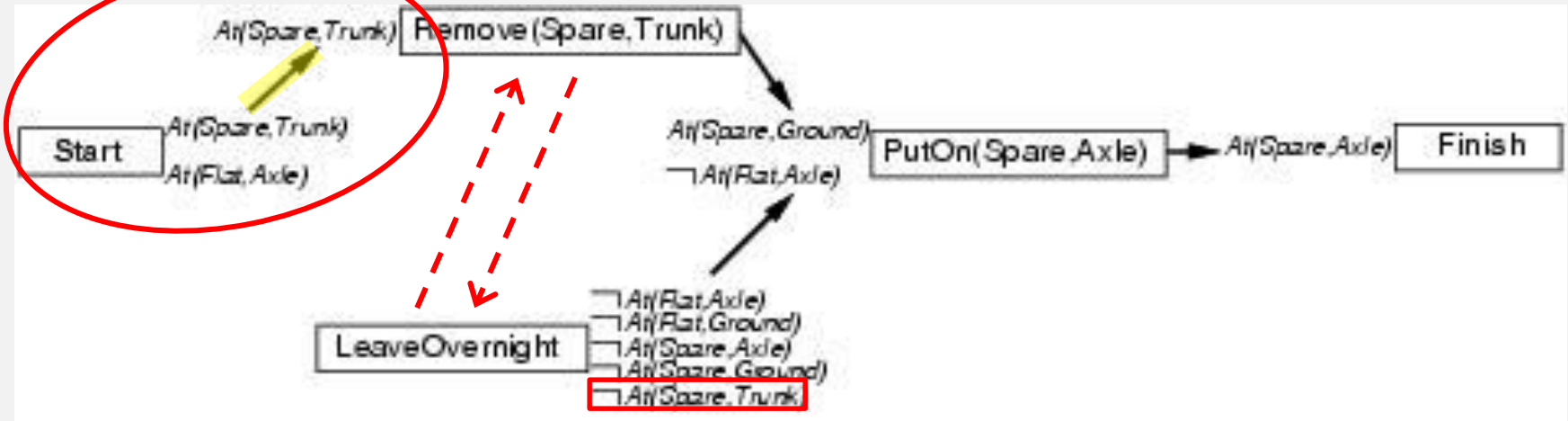
Resolução do problema



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

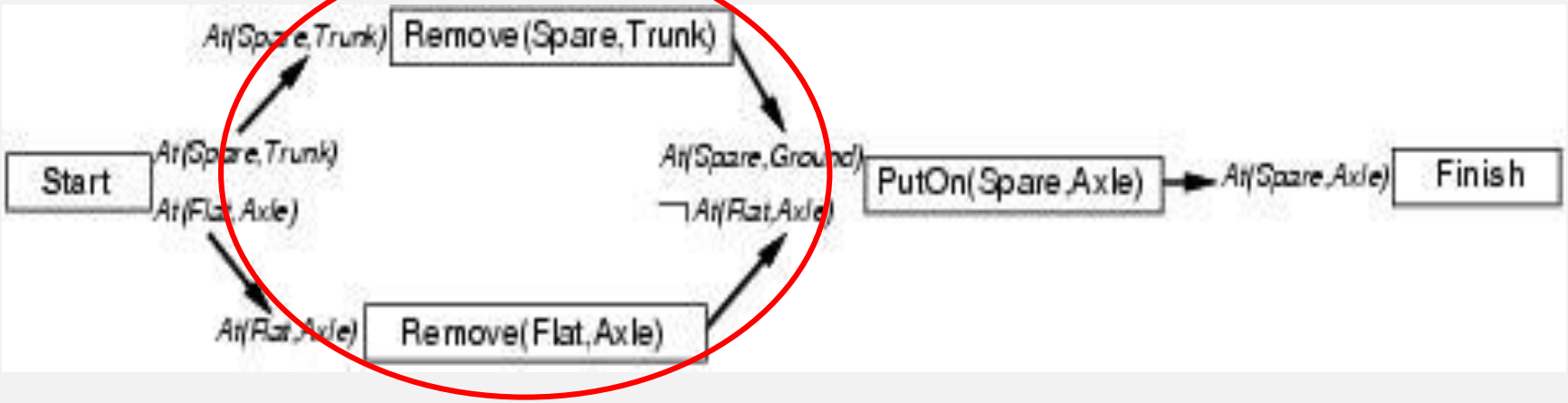
$$LeaveOverNight \xrightarrow{\neg At(Flat,Axle)} PutOn(Spare, Axle)$$

Resolução do problema



- Escolher uma pré-condição aberta : *At(Spare, Trunk)*
- Somente *Start* é aplicável
- Adicionar ligação causal: $Start \xrightarrow{At(Spare,Trunk)} \text{Re}\,move(Spare,Trunk)$
- 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

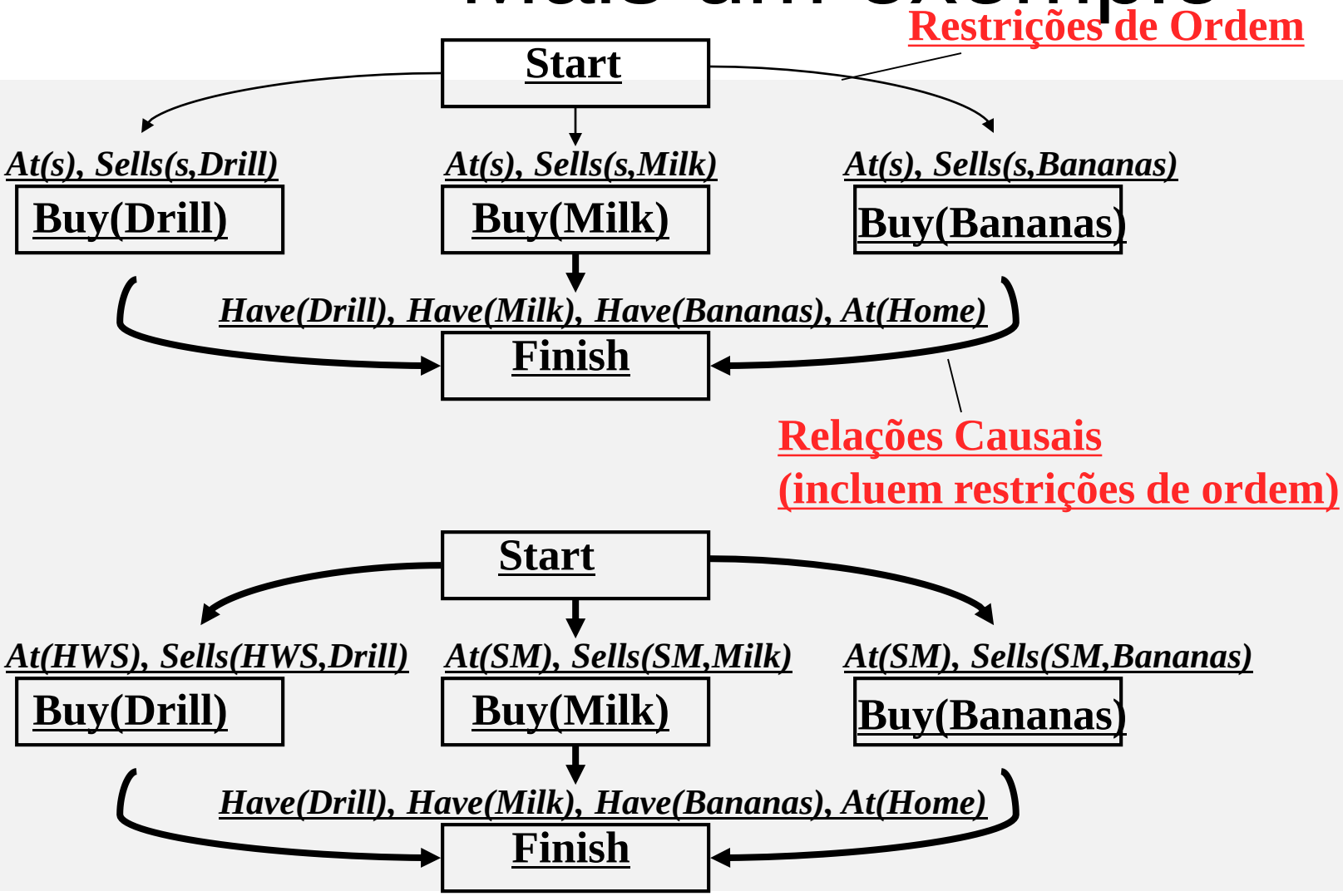


- Remover *LeaveOverNight*, *Remove(Spare, Trunk)* e ligações causais
- Repetir passo com *Remove(Spare,Trunk)*
- Adicionar também *Remove(Flat,Axle)*

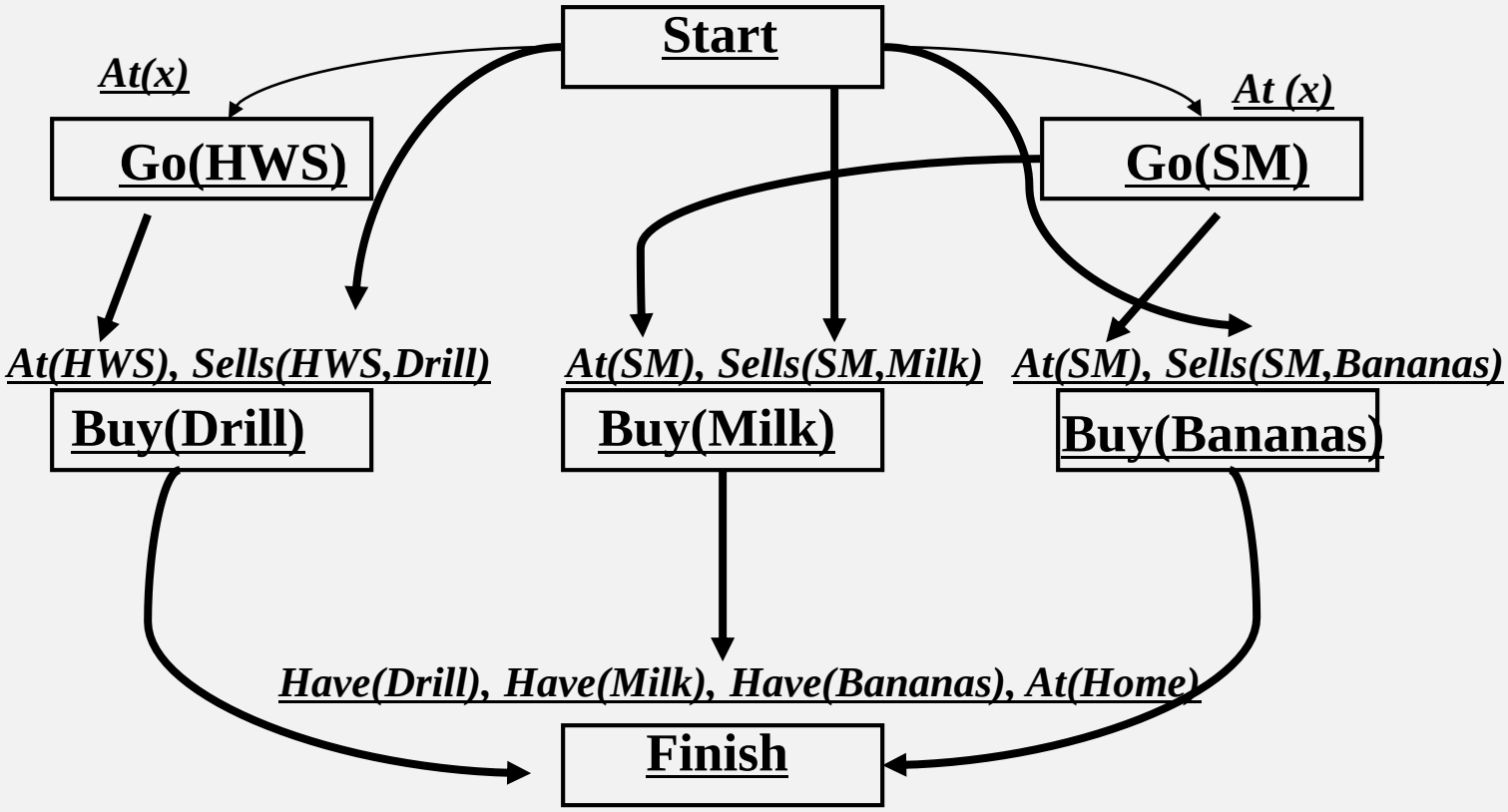
Mais um exemplo

Op(ACTION: Go(there),
PRECOND: At(here),
EFFECT: At(there) ∧ ¬At(here))
Op(ACTION: Buy(x),
PRECOND: At(store) ∧ Sells(store,x)
EFFECT: Have(x))

Mais um exemplo

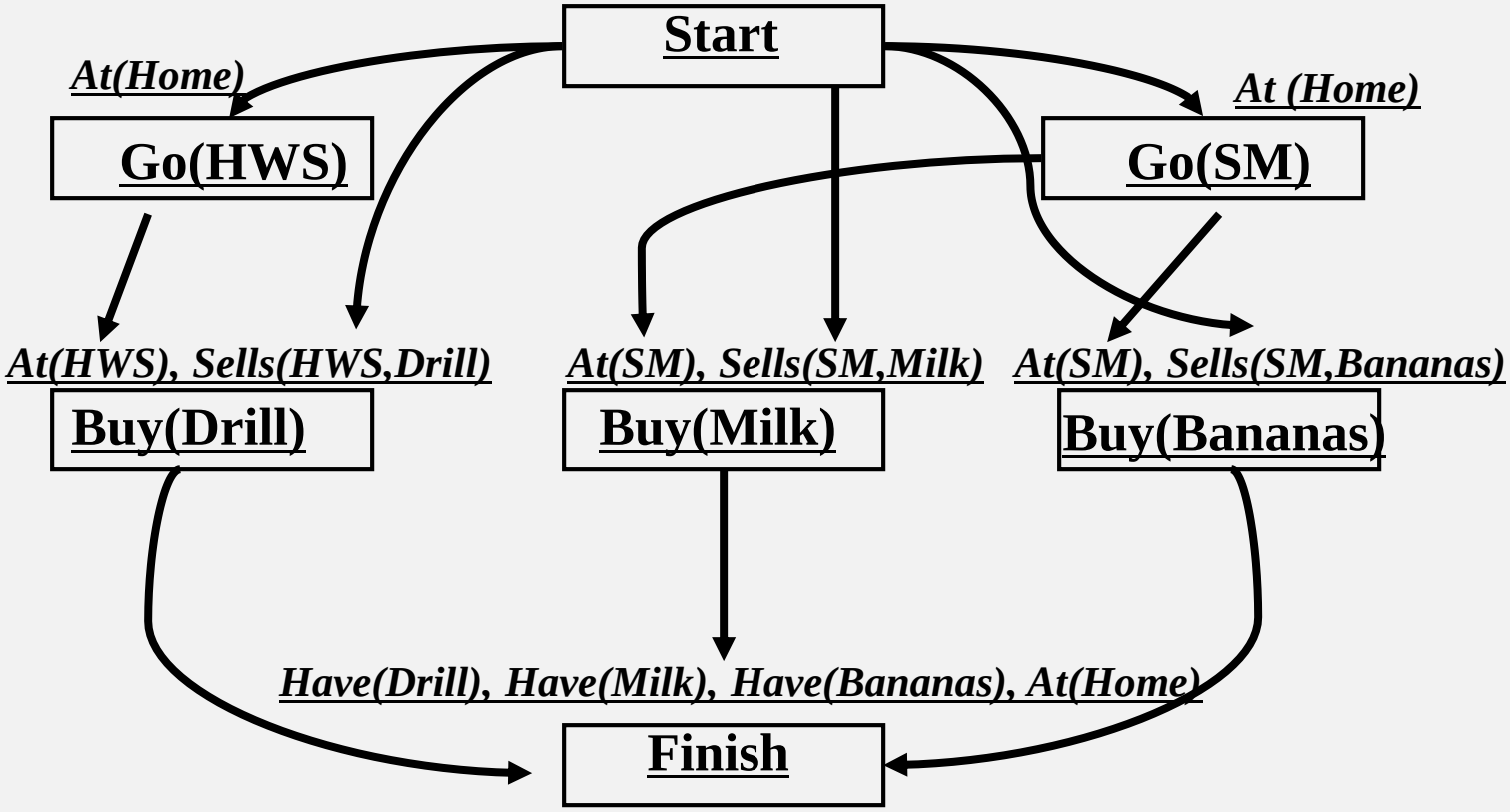


Mais um exemplo



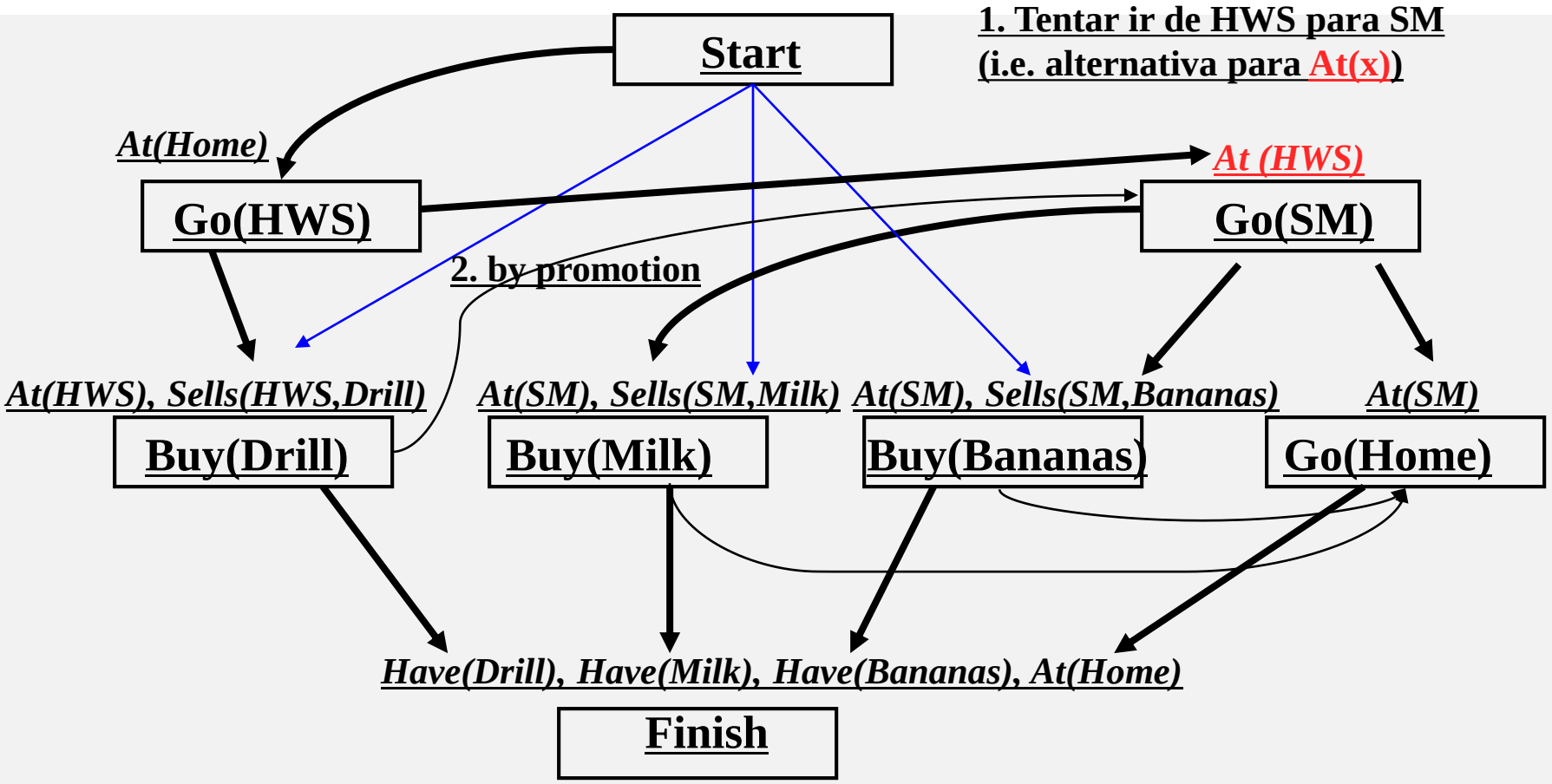
Mais um exemplo

Conflito não se resolve → retroceder e fazer outra escolha



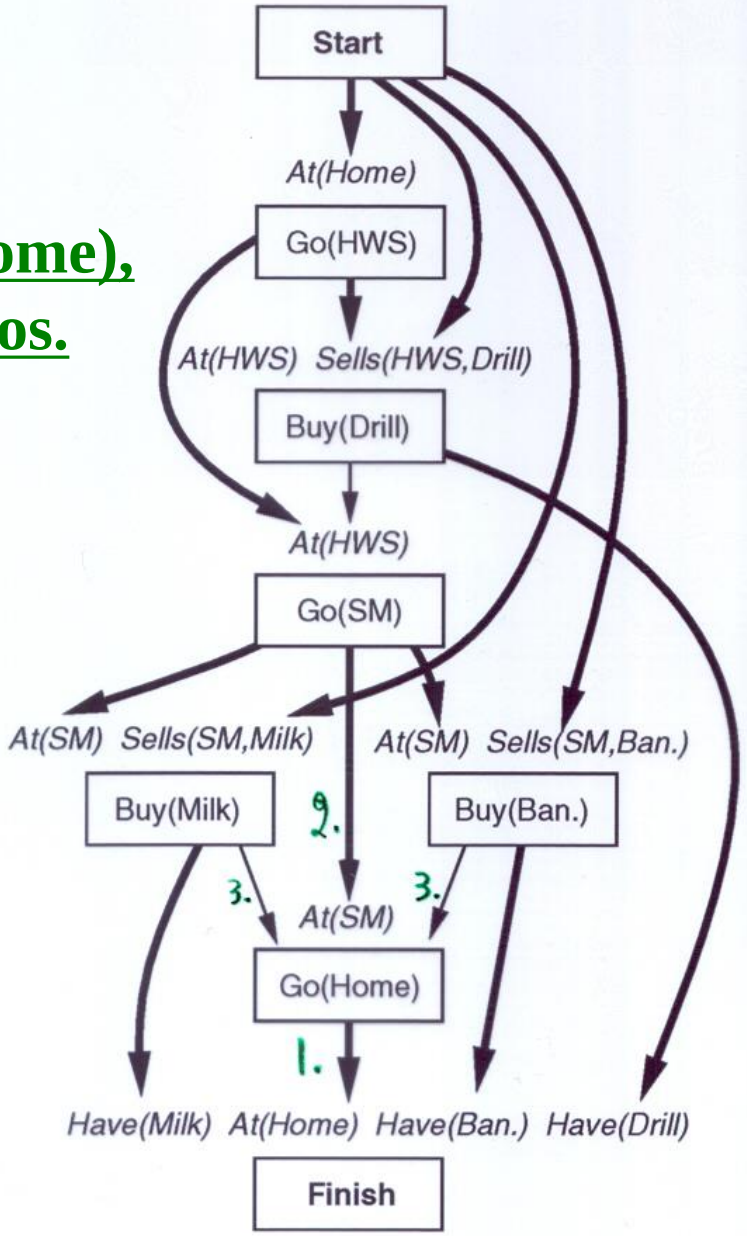
TÉCNICO LISBOA

Mais um exemplo



Mais um exemplo

Se em 2 usássemos At(HWS) ou At(Home), os conflitos não poderiam ser resolvidos.



TÉCNICO LISBOA

Alguns pormenores …

- O que acontece quando é usada uma representação em LPO que inclui variáveis?
  - Complica o processo de detectar 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 seleccionarem uma PRÉ-CONDIÇÃO