

PLANEAMENTO

CAP 10 (SEC 10.3)

Parcialmente adaptado de
<http://aima.eecs.berkeley.edu>
e de

Dana Nau: Lecture slides for Automated Planning

Resumo

- Planeamento de Ordem Parcial (POP)
- Grafos de Planeamento (GRAPHPLAN)

Planeamento de Ordem Parcial

- As técnicas anteriores produzem planos lineares totalmente ordenados (sequências de acções).
 - Não permitem que se tire partido da decomposição de problemas.
 - As decisões são tomadas sobre como encadear acções em todos os subproblemas.
- Ideia: adoptar uma estratégia de menor compromisso:
 - Adiar escolhas durante a procura

Exemplo: sapatos

Goal(RightShoeOn, LeftShoeOn)

Init()

Action(RightShoe,
 PRECOND: RightSockOn
 EFFECT: RightShoeOn)

Action(RightSock,
 PRECOND:
 EFFECT: RightSockOn)

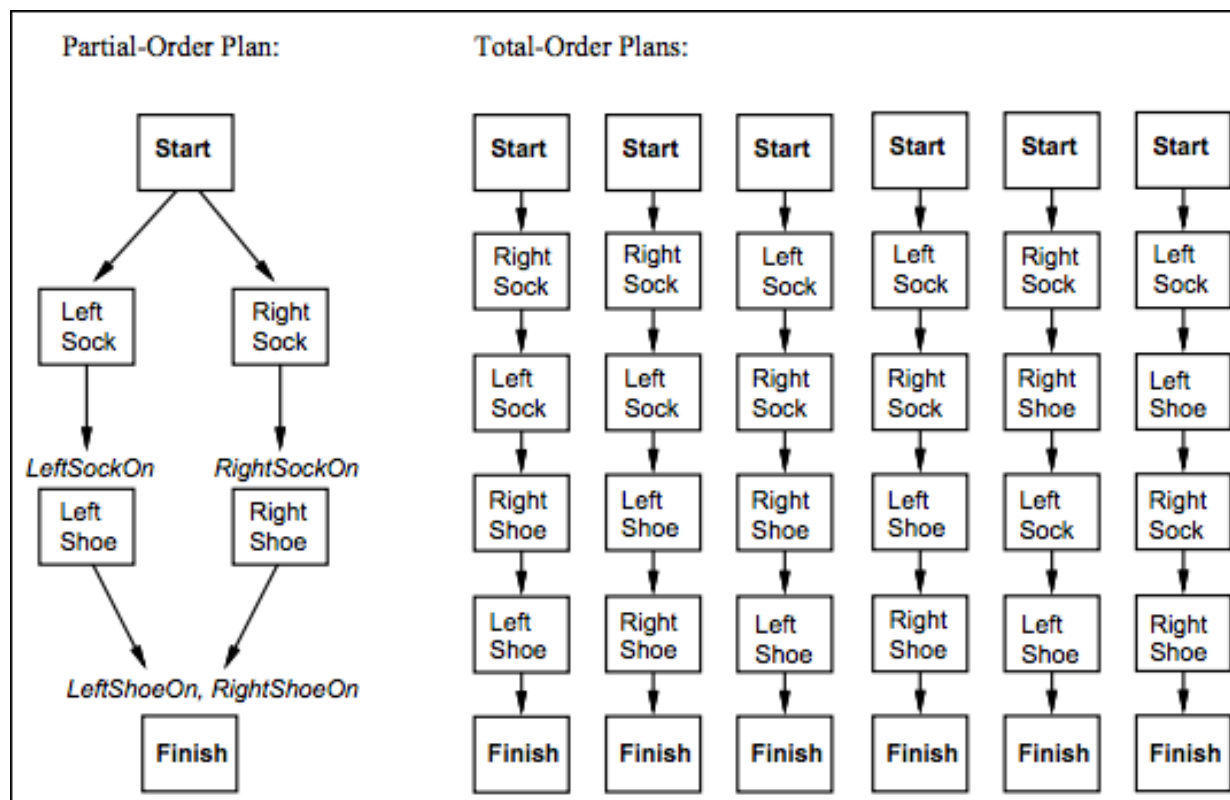
Action(LeftShoe,
 PRECOND: LeftSockOn
 EFFECT: LeftShoeOn)

Action(LeftSock,
 PRECOND:
 EFFECT: LeftSockOn)

- Planeador: combinar duas sequências de acções:
 - LeftSock, LeftShoe
 - RightSock, RightShoe

Planeamento de Ordem Parcial

- Qualquer algoritmo de planeamento que coloque duas acções num plano sem indicar qual deve ser executada primeiro é um Planeador de Ordem Parcial.



Planos Parciais

- Cada plano tem 4 componentes:
 - Um conjunto de **acções** (passos do plano)
 - Um conjunto de **restrições de ordenação** entre pares de acções $A < B$
 - $A < B$ lê-se “A antes de B”, mas não necessariamente imediatamente antes de B
 - Ciclos representam contradições: $A < B$, $B < C$, $C < A$
 - Um conjunto de **ligações causais** (intervalos de protecção) entre acções ($A \xrightarrow{p} B$) (entre o efeito p de uma acção e a precondição p de outra acção)
 - $A \xrightarrow{p} B$ lê-se “A obtém p para B” e p deve manter-se verdadeiro desde o momento em que a acção A é executada até ao momento em que a acção B é executada.
 - Um conjunto de **pre-condições abertas**
 - Se a precondição não foi atingida por uma acção no plano.
 - Uma pre-condição encontra-se alcançada sse é o efeito de um passo anterior e nenhum passo possivelmente interveniente a contrária.
 - Os planeadores reduzem o conjunto de pre-condições sem introduzir contradições.

Planos Parciais

- Um plano é **consistente** sse
 - não tem ciclos nas restrições de ordenação
 - Não tem conflitos nas ligações causais
- Um plano consistente sem pre-condições abertas é uma **solução**.

Planeamento de Ordem Parcial (POP) como um problema de pesquisa

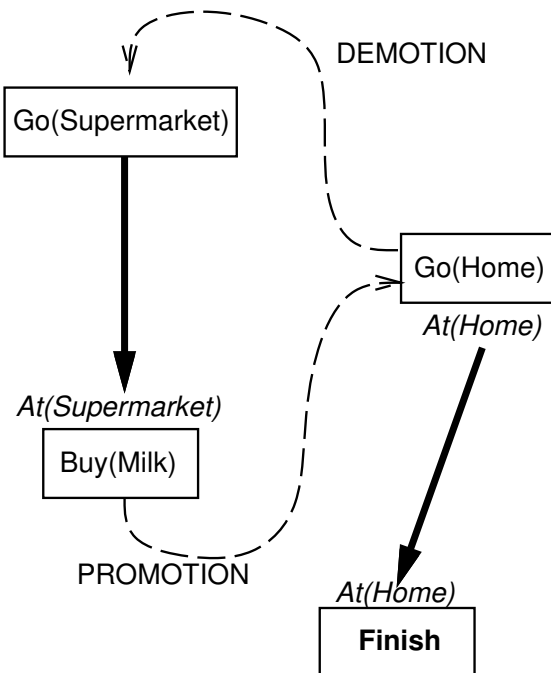
- O **plano inicial** contém:
 - As acções Start e Finish
 - Start desceve o estado inicial como os seus efeitos
 - Finish descreve o objectivo como as suas pre-condições
 - A restrição de ordenação $\text{Start} < \text{Finish}$
 - Nenhuma ligação causal
 - Todas as pre-condições de Finish estão abertas
- **Função Sucessora:**
 - Escolher uma pre-condição aberta p de uma acção B e gerar um plano sucessor para cada possibilidade consistente de escolher uma acção A que alcance p . Quando da geração de um plano sucessor:
 - A ligação causal $A \xrightarrow{p} B$ e as restrições de ordenação $A < B$ e $\text{Start} < A$ são adicionadas ao plano.
 - Devem ser resolvidos conflitos entre a nova ligação causal e todas as acções existentes
 - Devem ser resolvidos conflitos entre a acção A (se for nova) e todas ligações causais existentes
- **Objectivo:** não haver pré-condições abertas

Start
Literal_a, Literal_b, ...

Literal₁, Literal₂, ...
Finish

Resolução de Conflitos

- Um passo é uma ameaça (conflituante) quando potencialmente destrói uma condição já alcançada por uma ligação causal. E.g., Go(Home) é uma ameaça a At(Supermarket):



- Despromoção**
 - colocar antes de Go(Supermarket)

- Promoção**
 - colocar depois de Buy(Milk)

- O teste de consistência assegura que não foram introduzidos ciclos nas relações de ordem temporais.

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, Groundp) $\wedge$ $\neg$ *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))

Exemplo: pneu sobresselente

Start $At(Spare, Trunk)$
 $At(Flat, Axle)$

$At(Spare, Axle)$ **Finish**

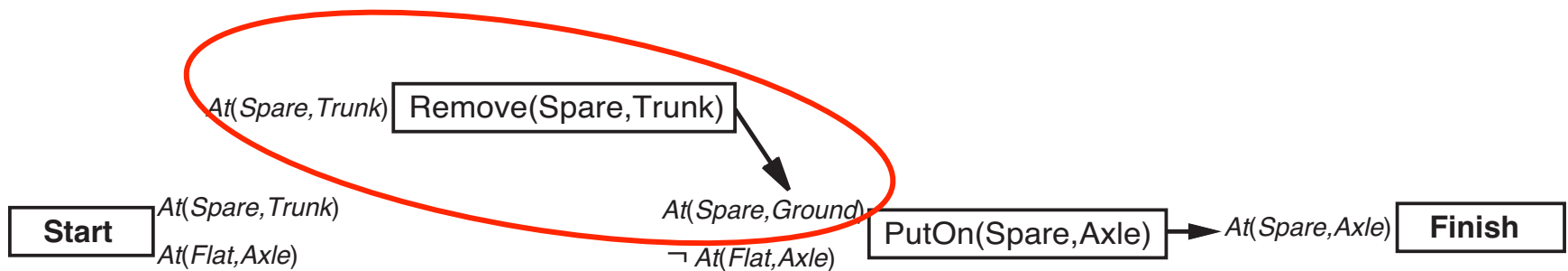
- Plano Inicial: Start com EFFECTS (estado inicial) e Finish com PRECOND (objectivo).

Exemplo: pneu sobresselente



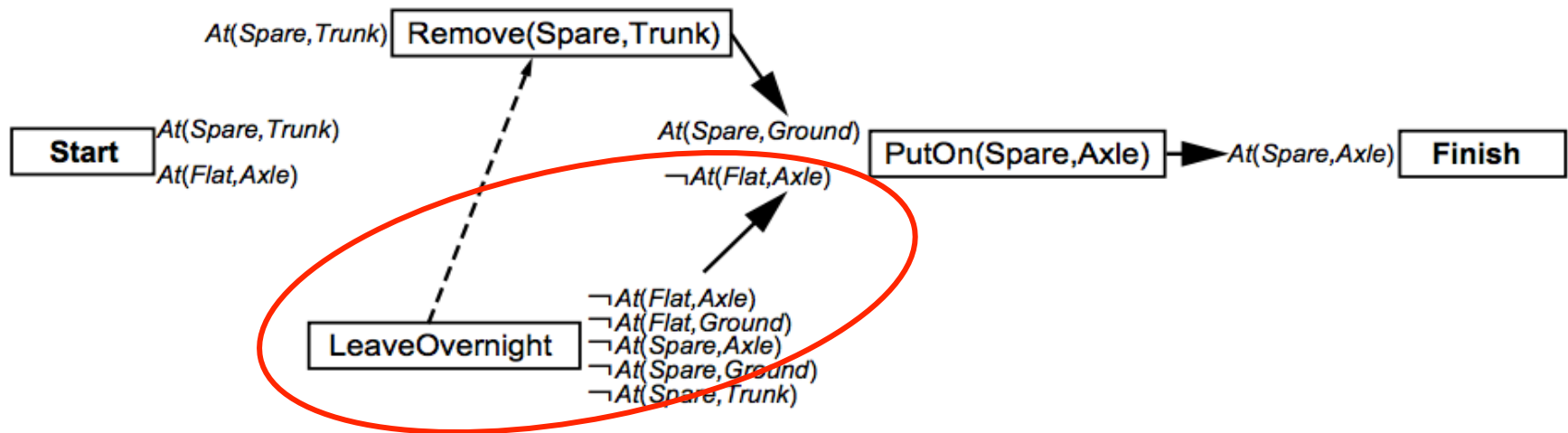
- Escolher uma precondição aberta: $At(Spare, Axle)$
- Escolher acção aplicável: $PutOn(Spare, Axle)$
- Adicional ligação causal: $PutOn(Spare, Axle) \xrightarrow{At(Spare, Axle)} Finish$
- Adicionar Restrição de ordenação: $PutOn(Spare, Axle) < Finish$

Exemplo: pneu sobresselente



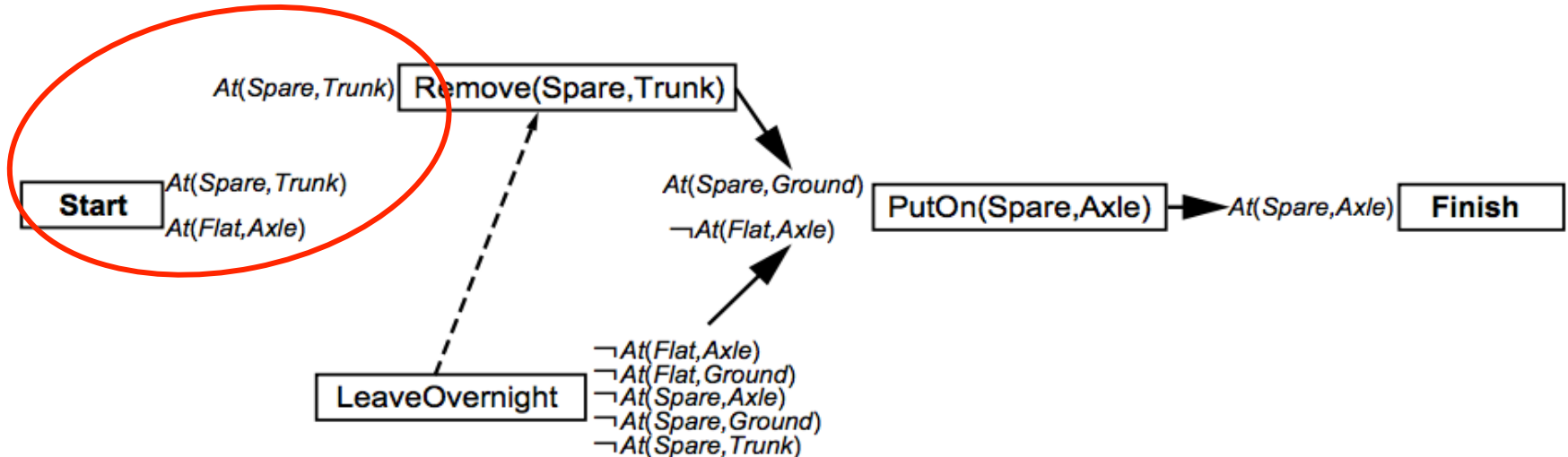
- Escolher uma precondição aberta: $At(Spare, Ground)$
- Escolher acção aplicável: $Remove(Spare, Trunk)$
- Adicional ligação causal: $Remove(Spare, Trunk) \xrightarrow{At(Spare, Ground)} PutOn(Spare, Axle)$
- Adicionar Restrição de ordenação: $Remove(Spare, Trunk) < PutOn(Spare, Axle)$

Exemplo: pneu sobresselente



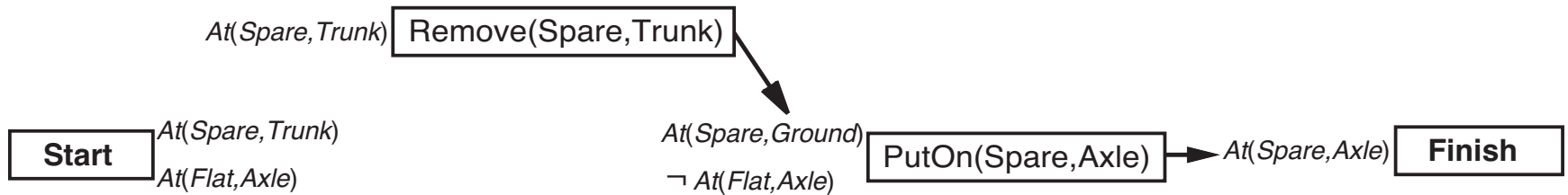
- Escolher uma precondição aberta: $\neg At(Flat, Axle)$
- Escolher acção aplicável: **LeaveOvernight**
- Conflito: $Remove(Spare, Trunk) \xrightarrow{At(Spare, Ground)} PutOn(Spare, Axle)$
- Para resolver, adicionar restrição: **LeaveOvernight < Remove(Spare, Trunk)**
- Adicional ligação causal: $LeaveOvernight \xrightarrow{\neg At(Flat, Axle)} PutOn(Spare, Axle)$

Exemplo: pneu sobresselente

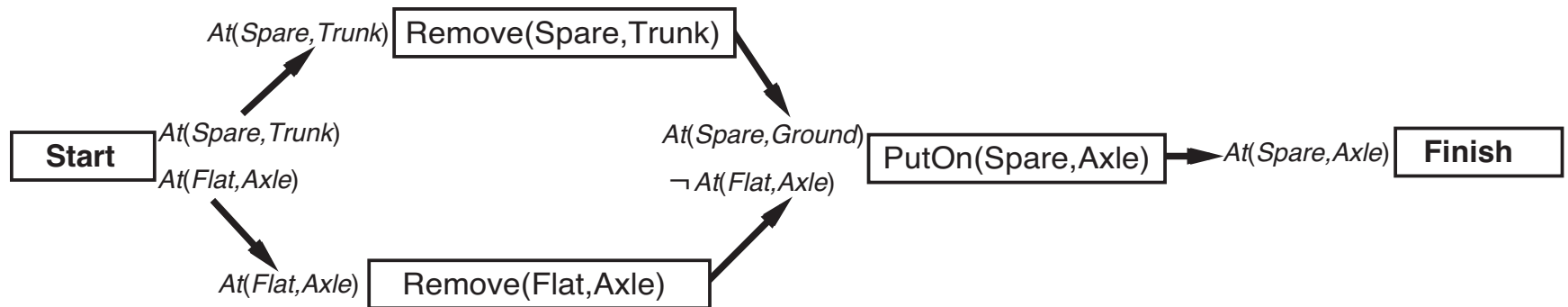


- Escolher uma precondição aberta: $At(Spare, Trunk)$
- Escolher acção aplicável: **Start**
- Adicional ligação causal: $Start \xrightarrow{At(Spare, Trunk)} Remove(Spare, Trunk)$
- Conflito: ligação causal com efeito $At(Spare, Trunk)$ em **LeaveOvernight**
 - Não há reordenação possível!
- Backtrack!

Exemplo: pneu sobresselente



Exemplo: pneu sobresselente



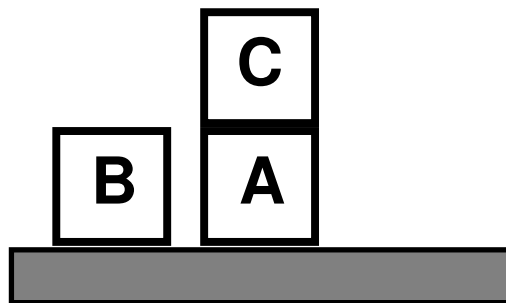
- Resultado final.

Propriedades do POP

- Algoritmo não-determinista: retrocede no caso de falha para pontos de escolha:
 - escolha de S_{add} para alcançar S_{need}
 - escolha de promoção ou despromoção
 - selecção de S_{need} é irrevogavel
- POP é **solido**, **completo**, e **sistemático** (sem repetições)
- Extensões para disjunção, universais, negação e condicionais na definição dos operadores (linguagem ADL).
- POP decompõe o problema, particularmente bom para problemas com muitos sub-objectivos fracamente interligados
- Não representa os estados explicitamente:
 - Difícil desenhar heurísticas para estimar distancia ao objectivo.
- Uma heurística pode ser usada para escolher a condição a satisfazer:
 - Escolher a mais restritiva i.e. a satisfeita pelo menor número de acções
 - Retroceder quando não houver acções que satisfaçam uma pre-condição, mesmo que não haja conflito
- Consegue-se eficiência com boas heurísticas derivadas da descrição do problema (ver mais à frente...)

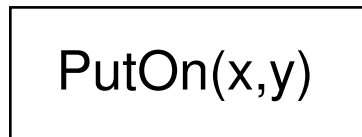
Exemplo: mundo dos blocos

"Sussman anomaly" problem

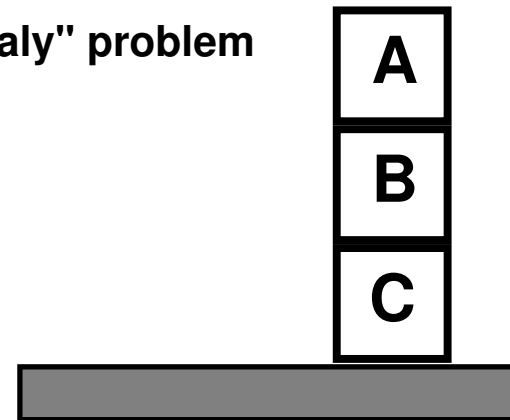


Start State

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



$\sim On(x,z)$ $\sim Clear(y)$
 $Clear(z)$ $On(x,y)$



Goal State

$Clear(x)$ $On(x,z)$



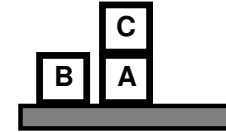
$\sim On(x,z)$ $Clear(z)$ $On(x, Table)$

+ several inequality constraints

Exemplo: mundo dos blocos

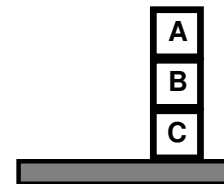
START

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

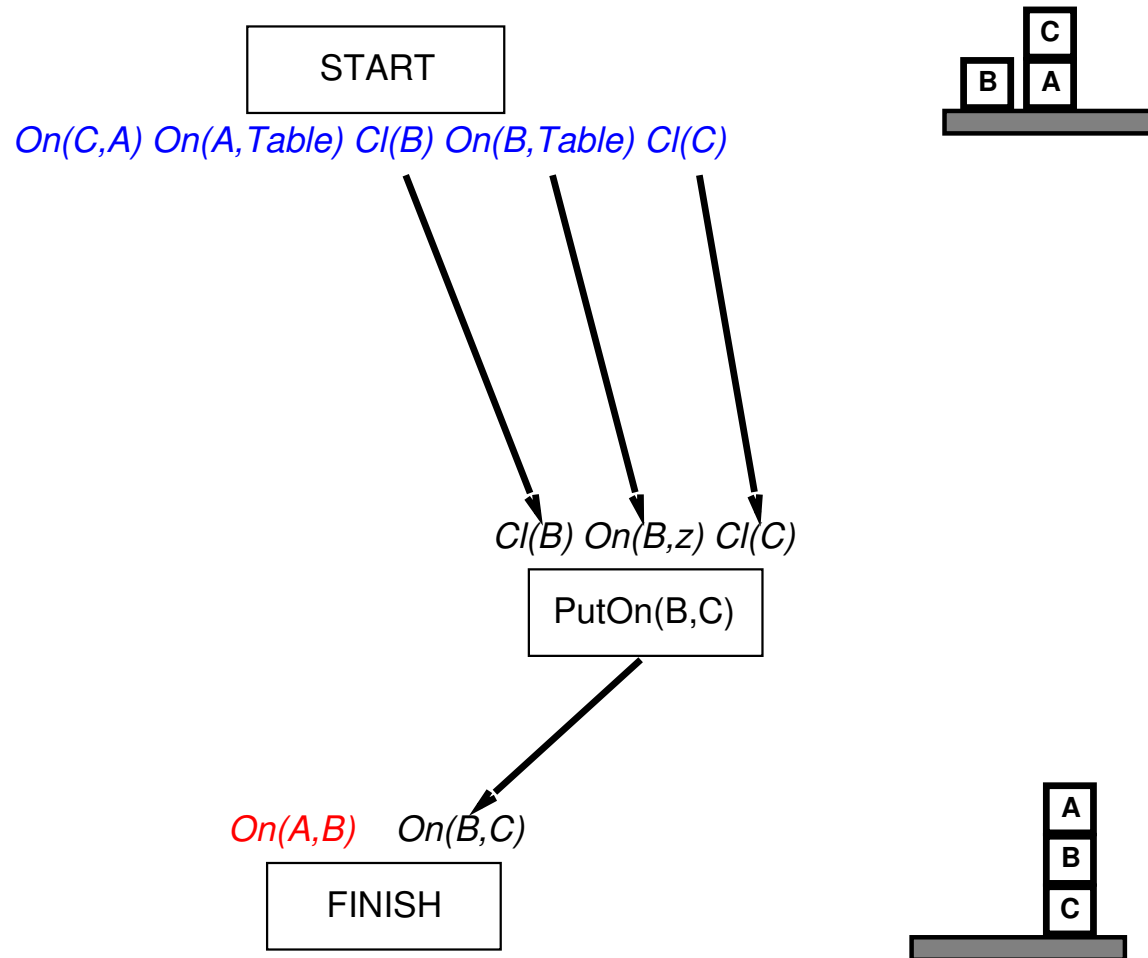


On(A,B) On(B,C)

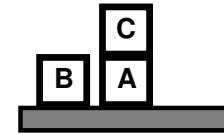
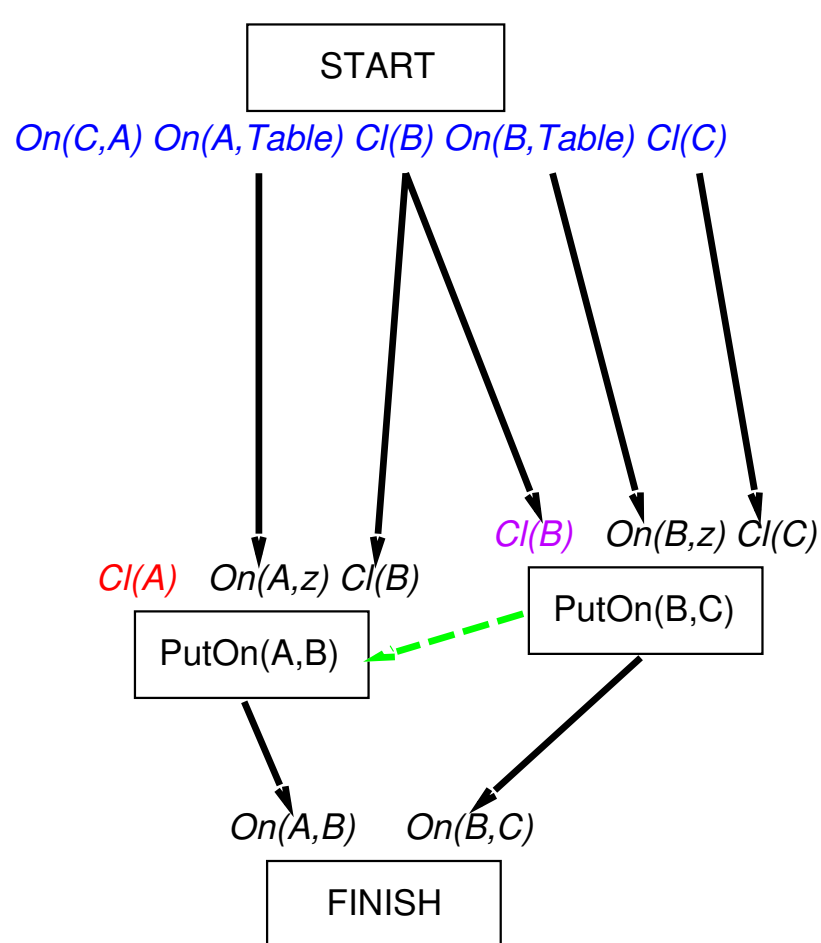
FINISH



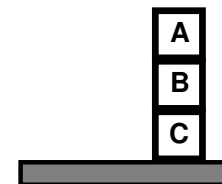
Exemplo: mundo dos blocos



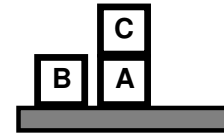
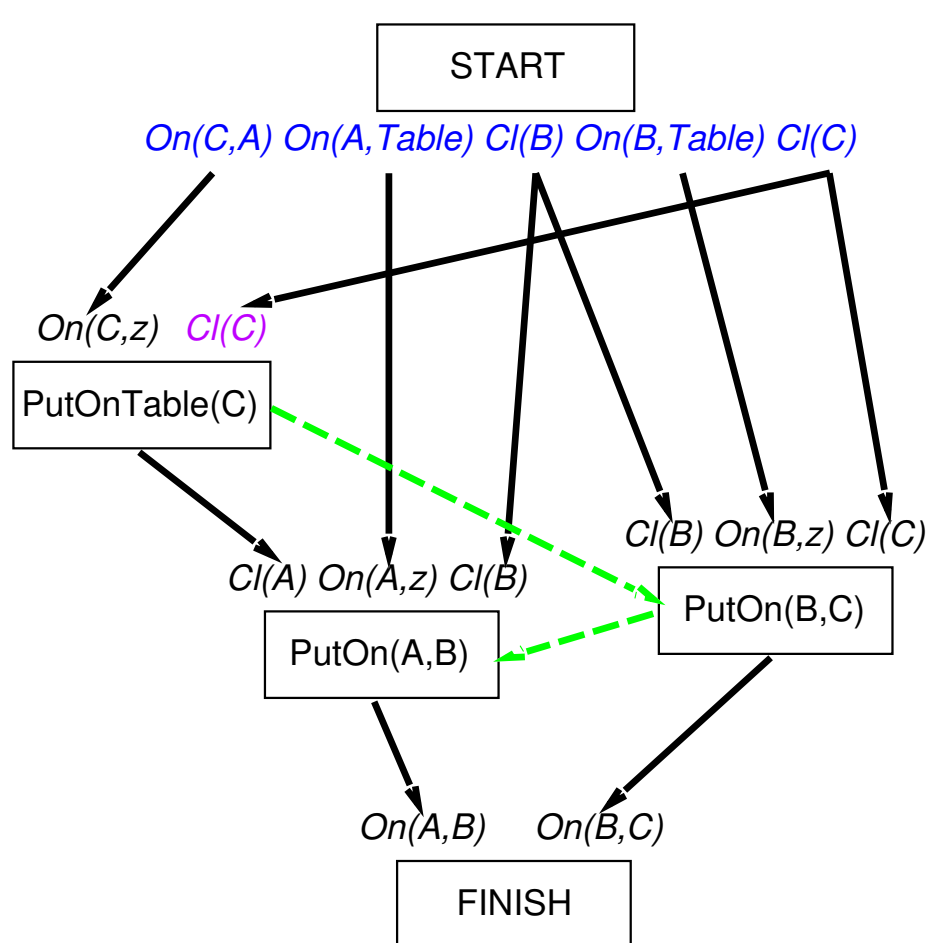
Exemplo: mundo dos blocos



PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)

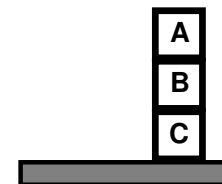


Exemplo: mundo dos blocos



PutOn(A,B)
clobbers $Cl(B)$
 $\Rightarrow$ order after
PutOn(B,C)

PutOn(B,C)
clobbers $Cl(C)$
 $\Rightarrow$ order after
PutOnTable(C)



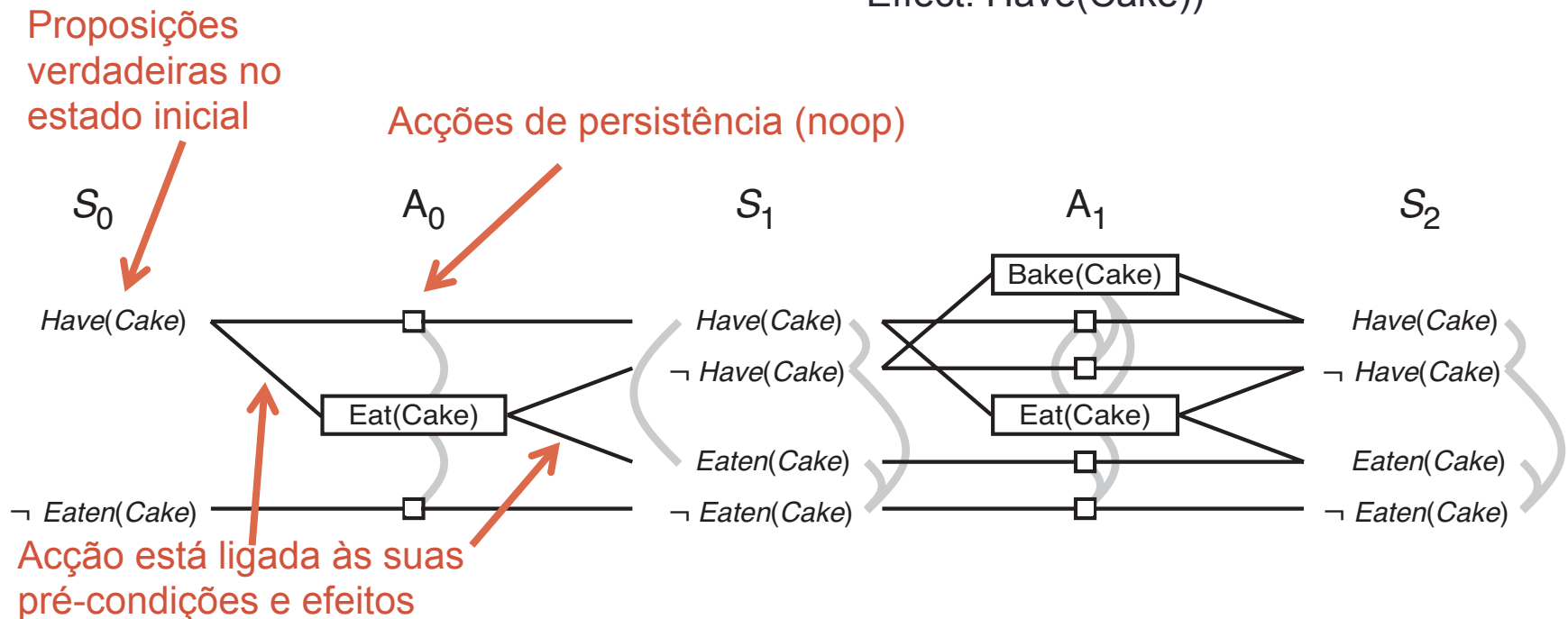
Grafos de Planeamento

- Estruturas de dados especiais usadas para:
 - Derivar melhores heurísticas e.g. para o POP
 - Extrair soluções para o problema de planeamento (algoritmo GRAPHPLAN)

Exemplo: Grafo de Planos

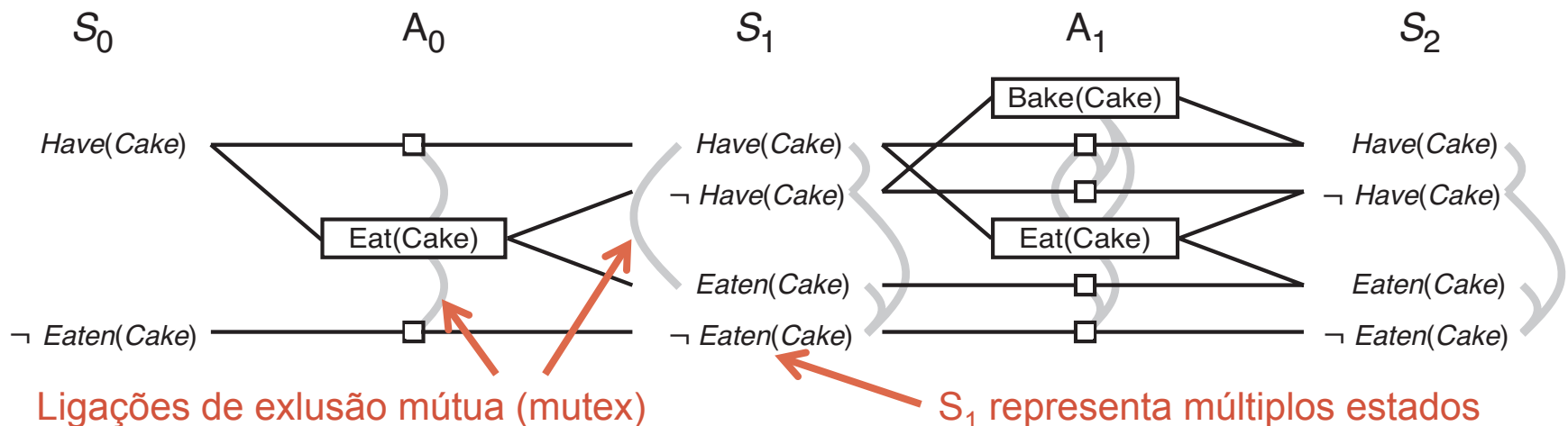
Init(Have(Cake))
Goal(Have(Cake) $\wedge$ Eaten(Cake))

Action(Eat(Cake))
Precond: Have(Cake)
Effect: $\neg$ Have(Cake) $\wedge$ Eaten(Cake))
Action(Bake(Cake))
Precond: $\neg$ Have(Cake)
Effect: Have(Cake))



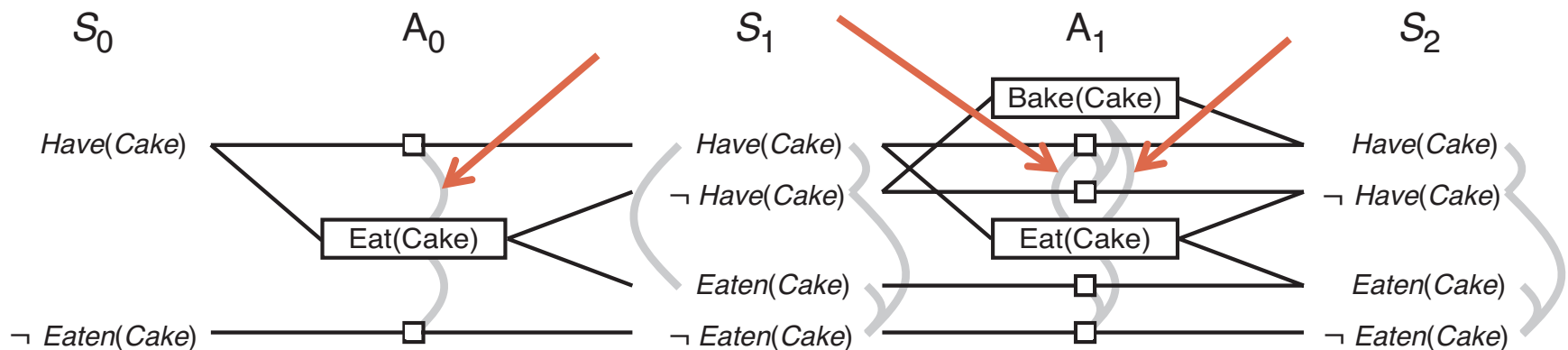
Exemplo: Grafo de Planos

- Em cada nível de estados, listar todos os literais que podem ser verdadeiros nesse estado
- Em cada nível de acções, listar todas as no-ops e todas as acções cujas pré-condições podem ser verdadeiras em estados anteriores
- Repetir até que o plano “estabilize” (chegue a um ponto fixo) i.e. quando novos literais não aparecem ($S_i = S_{i+1}$)
- Construir o Grafo de Planos é um processo polinomial
- Adicionar ligações (binárias) de exclusão mútua (mutex) entre acções em conflito e entre literais em conflito.



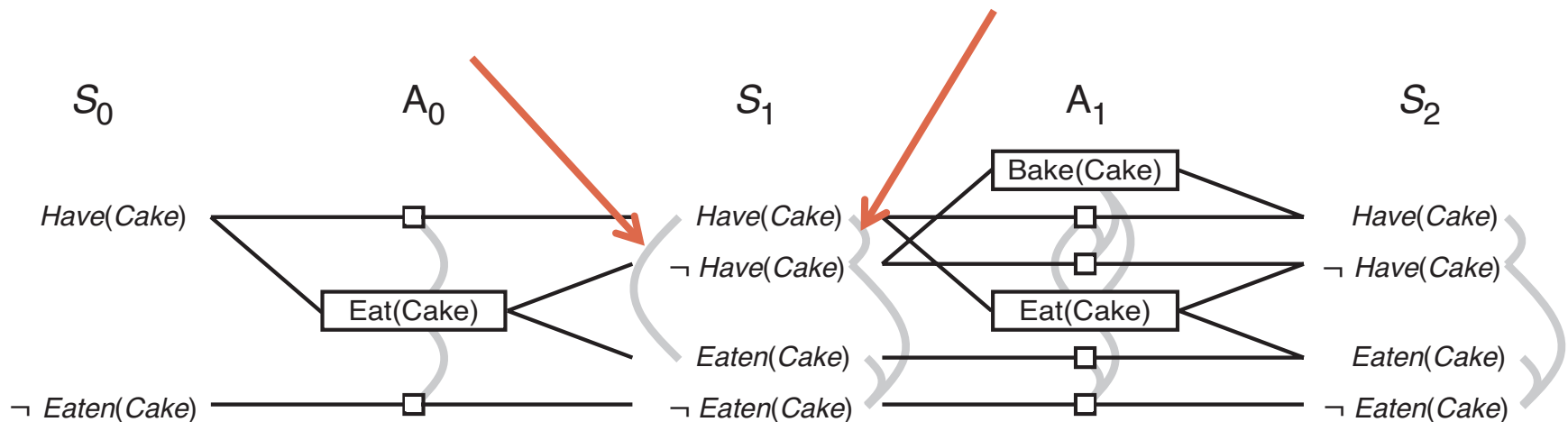
Ligações mutex entre acções

1. **Efeitos inconsistentes:** uma acção nega o efeito de outra
 - $\text{Eat}(\text{Cake})$ e noop de $\text{Have}(\text{Cake})$ discordam no efeito $\text{Have}(\text{cake})$
2. **Interferência:** uma acção nega a pré-condição de outra
 - $\text{Eat}(\text{Cake})$ nega a pré-condição da noop de $\text{Have}(\text{Cake})$
3. **Competição de recursos:** a pré-condição de uma acção é mutex com a pré-condição de outra
 - $\text{Bake}(\text{Cake})$ e $\text{Eat}(\text{Cake})$: competem na pré-condição $\text{Have}(\text{Cake})$



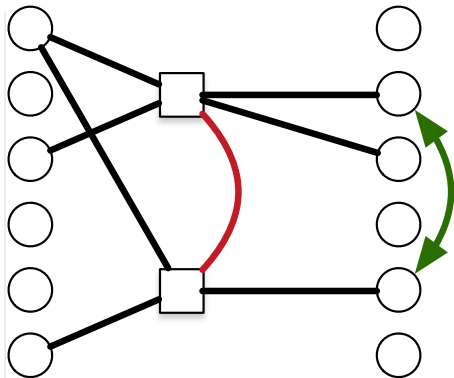
Ligações mutex entre literais

1. **Complementaridade:** Os dois literais são a negação um do outro
2. **Suporte inconsistente:** Todo o par de acções que consegue atingir os dois literais é mutex:
 - Em S_1 , $\text{Have}(\text{Cake})$ e $\text{Eaten}(\text{Cake})$ são mutex
 - Em S_2 , os mesmos literais não são mutex pois $\text{Bake}(\text{Cake})$ e a no-op de $\text{Eaten}(\text{Cake})$ não são mutex.

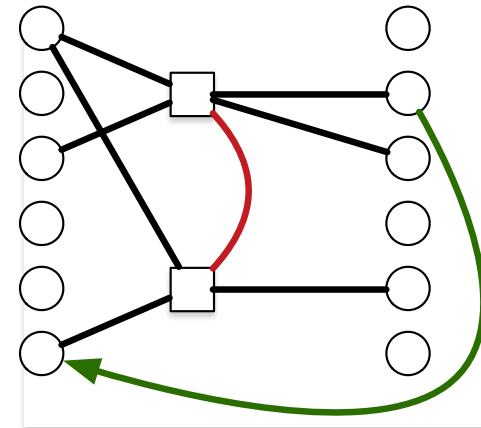


Ligações Mutex (resumo)

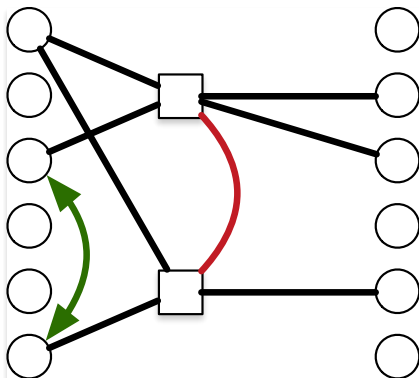
Efeitos Inconsistentes



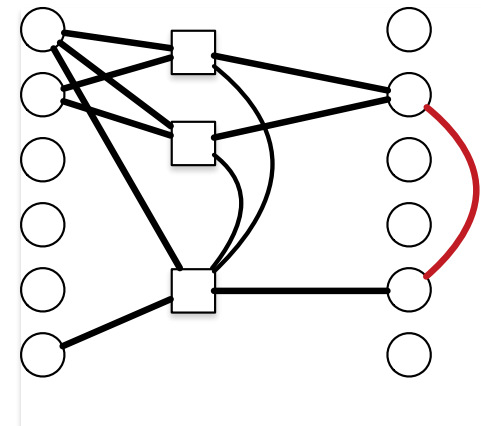
Interferência



Competição de Recursos



Suporte Inconsistente



Grafos de Planos para estimação de heurísticas

- O grafo de planeamento pode ser gerado em tempo polinomial e termina sempre após um número finito de passos:
 - Critério de paragem: quando dois níveis são idênticos.
- Pode-se utilizar o grafo de planeamento para obter heurísticas para o POP.
- Constrói-se o grafo de planeamento até obter um nível de estado em que todos os objectivos sejam alcançados e não sejam mutex dois a dois.

Grafos de Planos para estimação de heurísticas

- Um literal que não aparece no nível final não pode ser atingido por qualquer plano
 - **Planeamento por Pesquisa em espaço de estados**: qualquer estado contendo um literal não atingível tem um custo $h(n)=\infty$
 - **POP**: Qualquer plano com uma condição aberta não atingível tem um custo $h(n)=\infty$
- A estimativa de custo de qualquer literal é o primeiro nível em que ele aparece
 - Estimativa é admissível para literais individuais
 - Estimativa pode ser melhorada serializando o grafo (uma acção por nível), acrescentando mutex entre todas as acções num dado nível
- A estimativa de uma conjunção de literais
 - Três heurísticas: max-level, level sum e set level

Estimativa para um conjunto de literais

- **Max-level**
 - O nível mais elevado de um literal na conjunção
 - Admissível, mas não muito precisa
- **Level sum**
 - Assumindo o pressuposto de independência de sub-objectivos, soma o custo dos níveis de cada literal
 - Inadmissível, mas funciona bem em problemas essencialmente decomponíveis
- **Set level**
 - Encontra o primeiro nível em que os literais aparecem todos sem que qualquer par deles seja mutex.
 - Domina max-level, funciona muito bem em problemas onde existe bastante interacção entre sub-planos.

Algoritmo GRAPHPLAN

Também se pode usar o Grafo de Planos para extrair um plano.

```
GRAPHPLAN(problem) returns solution or failure  
graph ← INITIALPLANNINGGRAPH(problem)  
goals ← GOALS[problem]  
loop do  
  if goals all non-mutex in last level of graph then do  
    solution ← EXTRACTSOLUTION(graph,goals,LENGTH(graph))  
    if solution ≠ failure then return solution  
    else if NOSOLUTIONPOSSIBLE(graph) then return failure  
  graph ← EXPANDGRAPH (graph,problem)
```

- Duas fases principais
 1. Extraír solução
 2. Expandir o grafo

Exemplo: Execução do GRAPHPLAN

- $At(Spare, Axle)$ não está em S_0 .
- Não é necessário extrair solução
- Expandir o grafo

S_0

$At(Spare, Trunk)$

$At(Flat, Axle)$

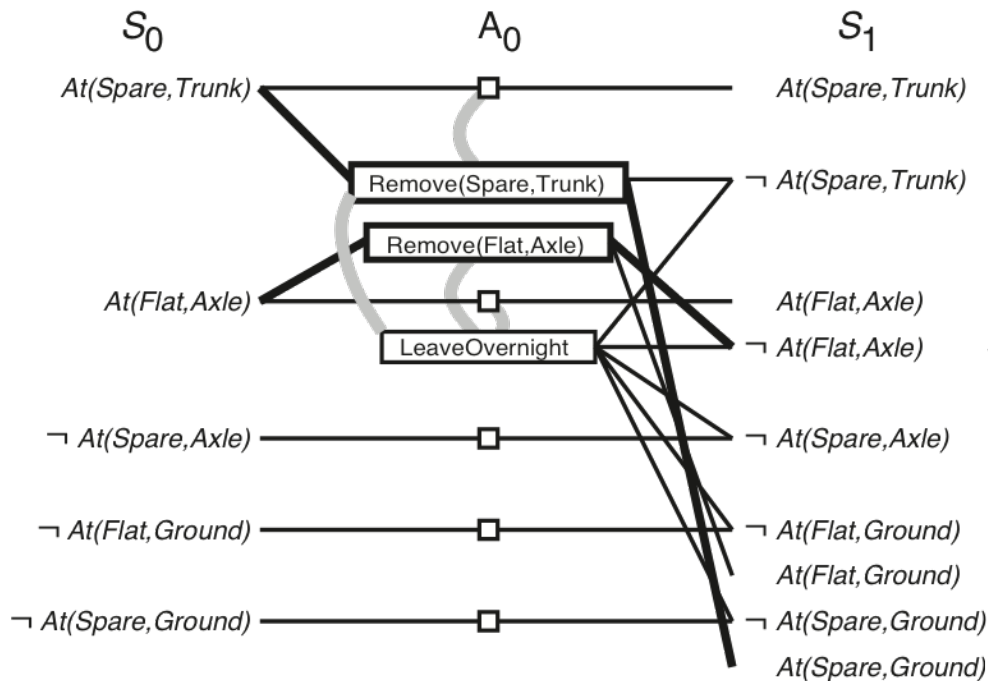
$\neg At(Spare, Axle)$

$\neg At(Flat, Ground)$

$\neg At(Spare, Ground)$

Exemplo: Execução do GRAPHPLAN

- Três acções são aplicáveis
 - Três acções e 5 no-ops são adicionados
 - Ligações mutex são adicionadas
 - Nem todas as ligações mutex são ilustradas, para manter o grafo legível...
- $At(Spare, Axle)$ não está em S_1 .
 - Não é necessário extrair solução
 - Expandir o grafo

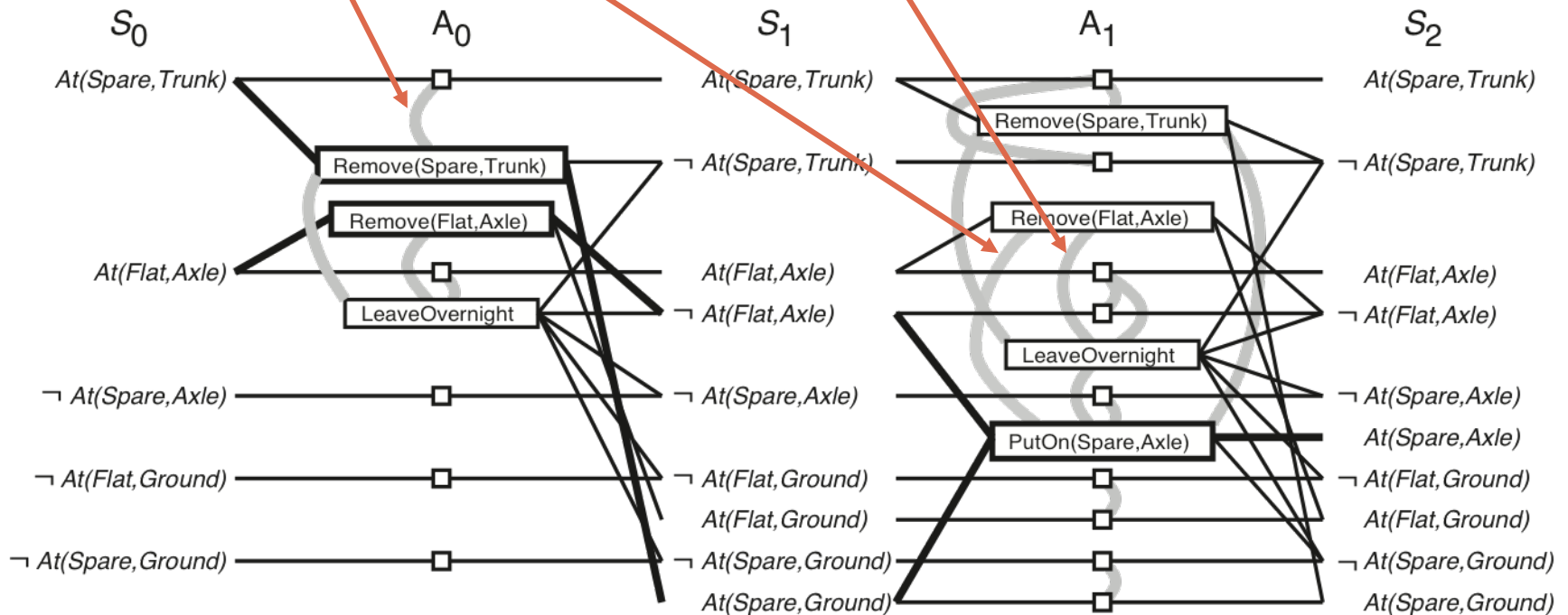


Exemplo: Execução do GRAPHPLAN

- O grafo de planos ilustra bem as ligações mutex:

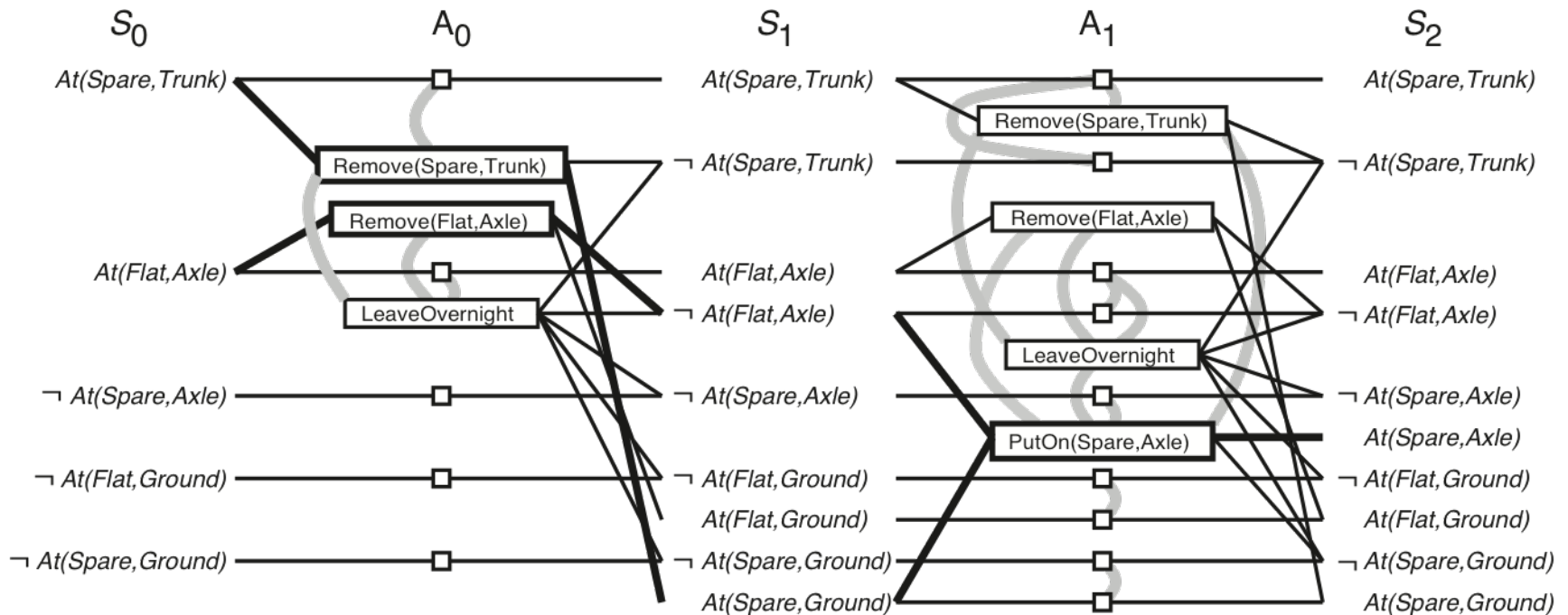
- Efeitos inconsistentes, interferência, competição de recursos, suporte inconsistente.

não
ilustrado...



Extracção da Solução

1. Resolver um problema de satisfação de restrições (CSP):
variáveis são as acções, domínios são {0=fora do plano, 1=no plano}, restrições são as ligações mutex
2. Procura por retrocesso a partir do último nível.



Extração da Solução com procura por retrocesso

- Começando no último nível de estado
 - Cada objectivo é colocado numa lista de objectivos para o nível de estado corrente
 - A pesquisa itera por todos os factos na lista de objectivos, tentando encontrar uma acção que o suporte e que não seja mutex com qualquer outra acção escolhida
 - Quando uma acção é escolhida, as suas pré-condições são adicionadas à lista de objectivos do nível abaixo
 - Quando todos os factos na lista de objectivos do nível corrente têm uma atribuição consistente de acções, a procura passa para o nível seguinte
- A procura retrocede (backtrack) ao nível anterior se não consegue atribuir uma acção a cada facto da lista de objectivos num dado nível.
- A procura sucede quando o primeiro nível é atingido.

Terminação do GRAPHPLAN

- O GRAPHPLAN termina
 - Literais aumentam monotonicamente
 - Acções aumentam monotonicamente
 - Mutexes decrescem monotonicamente
- É garantido que uma solução não existe quando:
 - Grafo estabiliza com todos os objectivos presentes e não-mutex e
 - EXTRASOLUTION não encontra uma solução.

Optimalidade do GRAPHPLAN

- Os planos gerados pelo GRAPHPLAN
 - São óptimos no número de paços necessários para executar o plano
 - Não são necessariamente óptimos no número de acções no plano (GRAPHPLAN produz planos parcialmente ordenados)

Comparação com POP

- Vantagem: a procura regressiva do GRAPHPLAN— que é a parte difícil—só olha para as acções no grafo de planeamento
 - Espaço de procura mais pequeno do que o do POP, logo mais rápido
- Desvantagem: para gerar o grafo de planeamento, GRAPHPLAN cria um número enorme de literais concretos (instanciados)
 - Muitos deles podem ser irrelevantes
- Pode-se aliviar este problema (mas não eliminar) atribuindo tipos de dados a variáveis e constantes
 - Só se instanciam variáveis com termos do mesmo tipo de dados

Slides adicionais

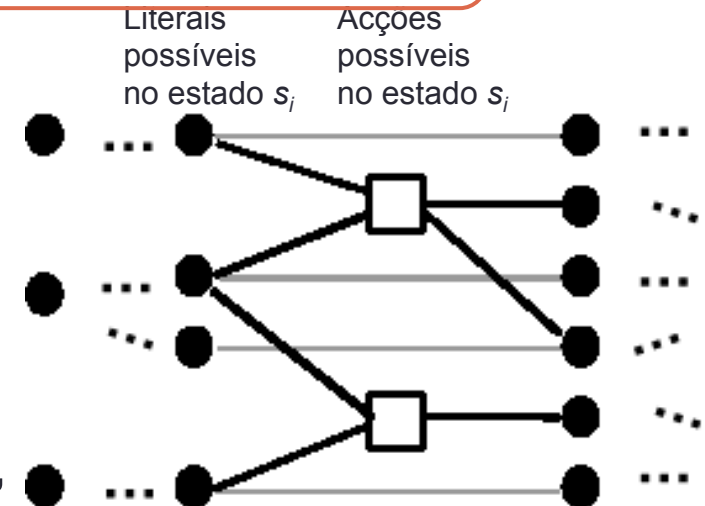
- Apresentação alternativa do GRAPHPLAN, com outro exemplo
- Exemplos adicionais do POP

GRAPHPLAN

Graphplan

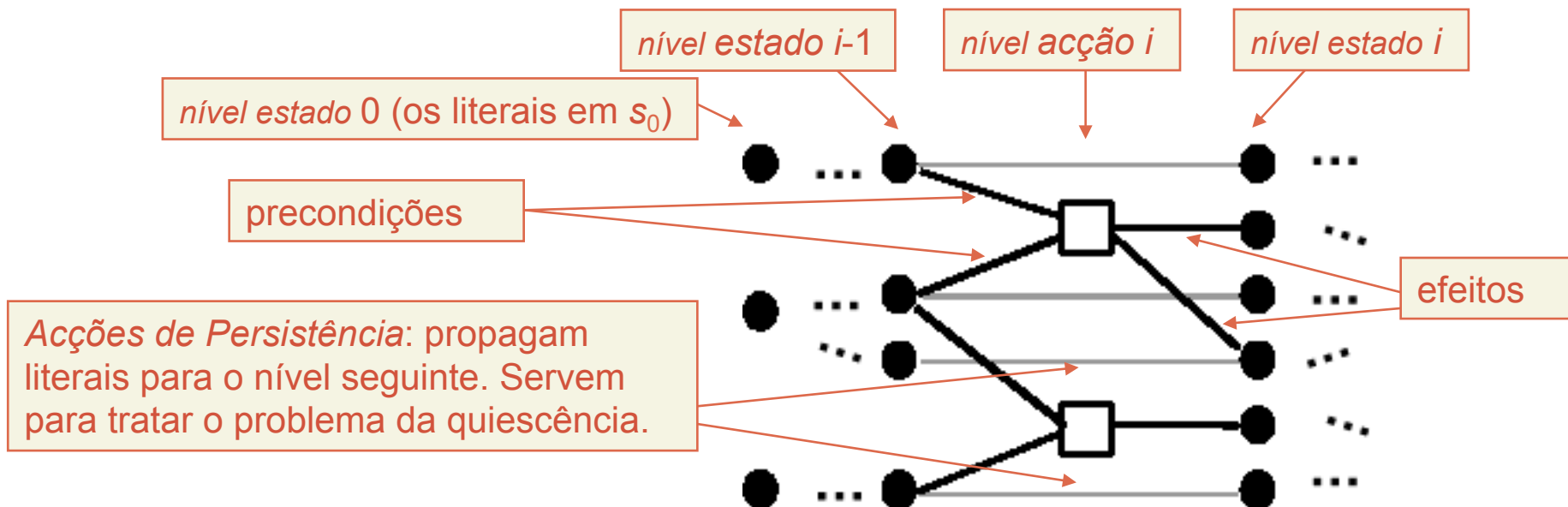
procedure Graphplan:

- for $k = 0, 1, 2, \dots$
 - *Expansão do grafo:*
 - criar um “grafo de planeamento” contendo k “níveis”
 - Verificar se o grafo de planeamento satisfaz uma condição necessária (mas insuficiente) para a existência de um plano problema relaxado
 - Se satisfizer, então
 - *Extrair solução:*
 - Procura regressiva, modificada para considerar apenas as acções no grafo de planeamento
 - Caso se encontre uma solução, então devolvê-la



O Grafo de Planeamento

- Camadas alternadas de literais e acções concretas (ground)
 - Todas as acções que **possivelmente** podem ocorrer em cada instante de tempo
 - Todos os literais produzidos por essas acções



Exemplo

- Suponha-se que pretende preparar um jantar de surpresa para o seu/sua cara-metade (que está a dormir e não deve ser acordado)

$s_0 = \{\text{garbage}, \text{cleanHands}, \text{quiet}\}$

$g = \{\text{dinner}, \text{present}, \neg \text{garbage}\}$

<u>Acção</u>	<u>Precondições</u>	<u>Efeitos</u>
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	<i>nenhuma</i>	$\neg \text{garbage}, \neg \text{cleanHands}$
dolly()	<i>nenhuma</i>	$\neg \text{garbage}, \neg \text{quiet}$

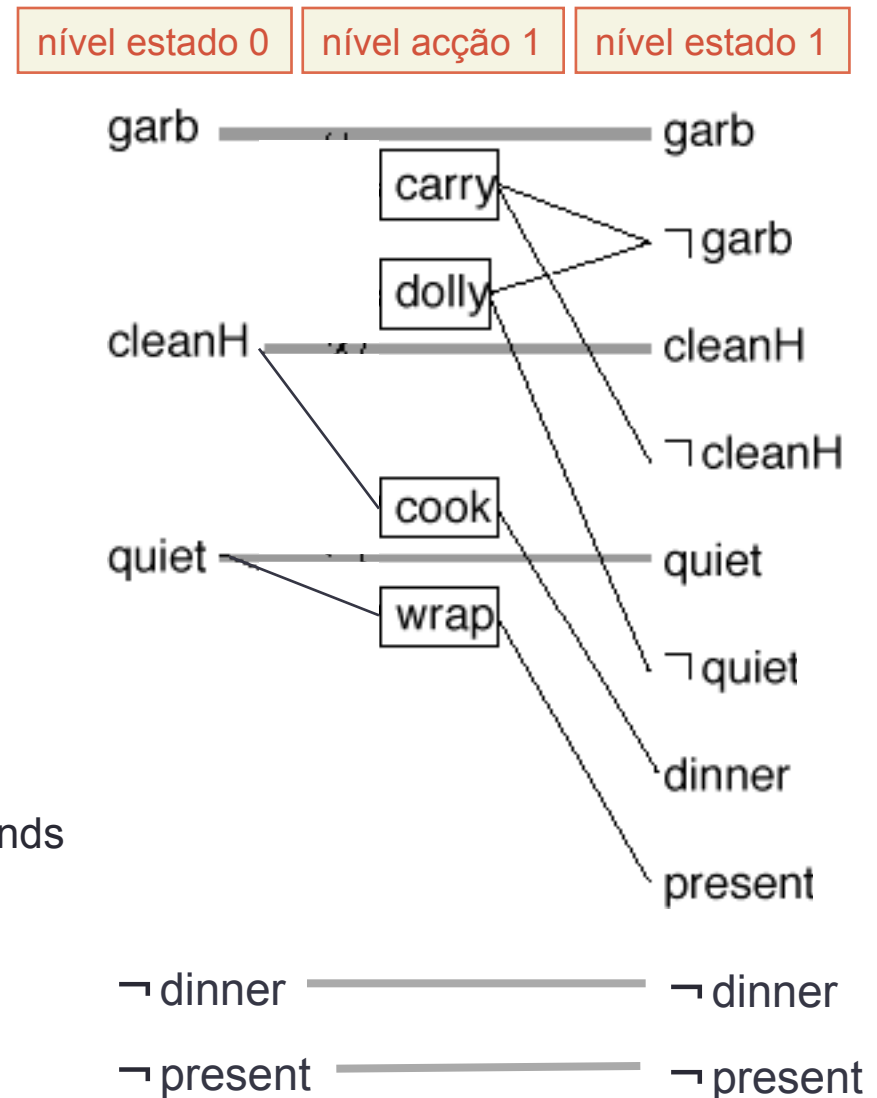
Também se adicionam acções de persistência: uma para cada literal L , com precondição L e efeito L .

Exemplo

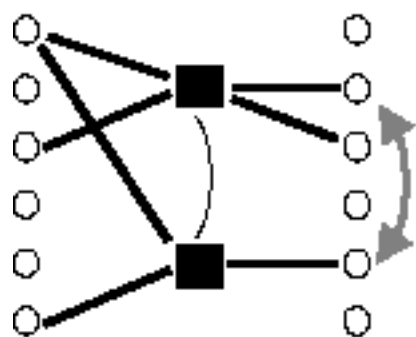
- nível estado 0:
 $\{\text{todos os átomos em } s_0\} \cup$
 $\{\text{negação de todos os átomos não em } s_0\}$
- nível acção 1:
 $\{\text{todas as acções cujas precondições estão satisfeitas em } s_0\}$
- nível estado 1:
 $\{\text{todos os efeitos de todas as acções no nível acção 1}\}$

<u>Acção</u>	<u>Precondições</u>	<u>Efeitos</u>
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	<i>nenhum</i>	$\neg$ garbage, $\neg$ cleanHands
dolly()	<i>nenhum</i>	$\neg$ garbage, $\neg$ quiet

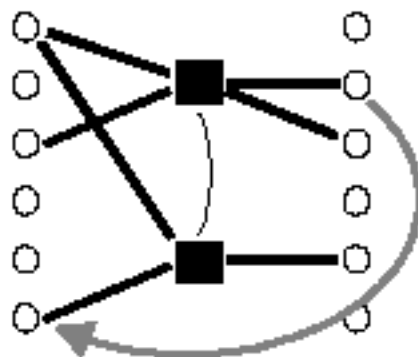
Juntamente com as acções de persistência



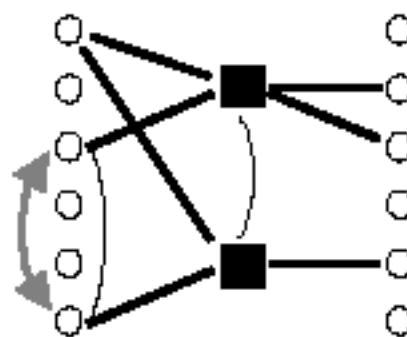
Exclusão Mútua



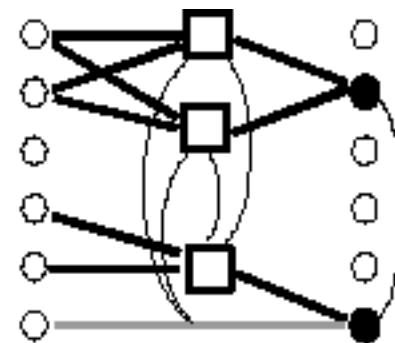
Inconsistent Effects



Interference



Competing Needs



Inconsistent Support

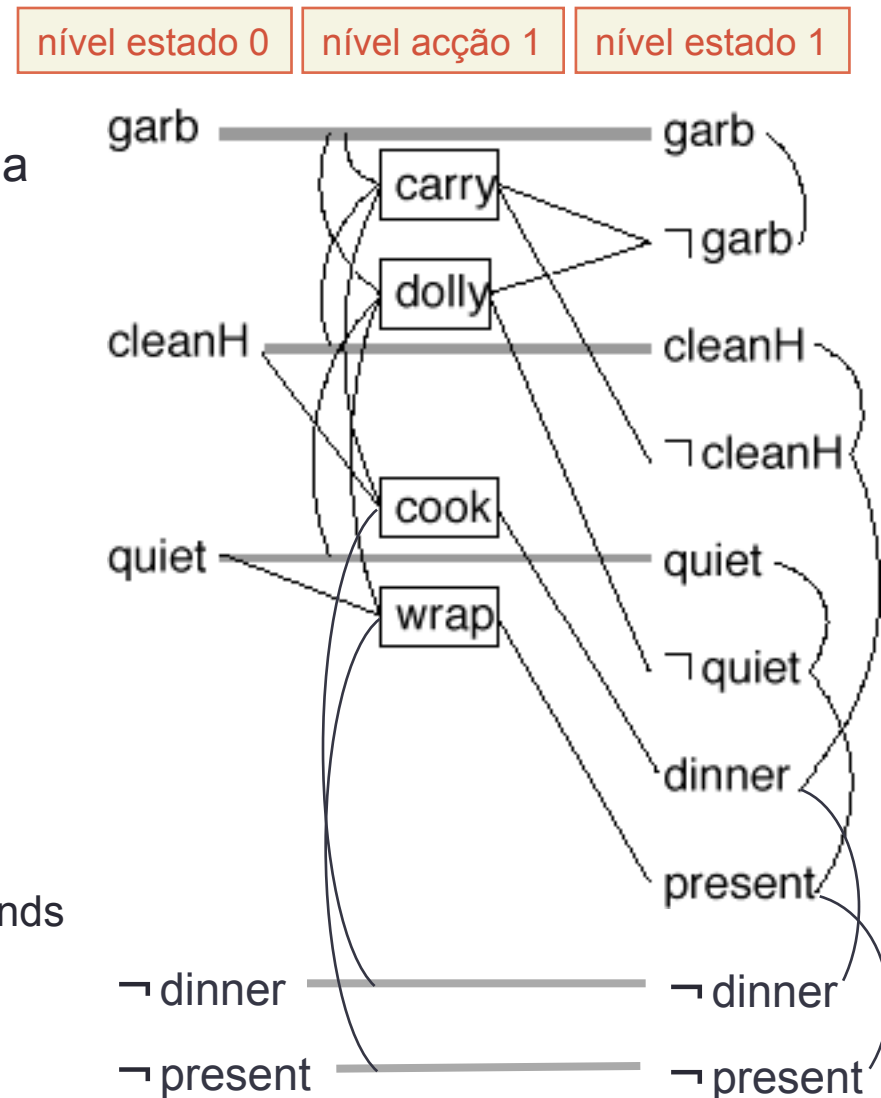
- Duas acções no mesmo nível de acção são **mutex** se
 - *Efeitos inconsistentes* : o efeito de uma nega o efeito da outra
 - *Interferência*: uma remove a precondition da outra
 - *Necessidades em competição*: têm precondições mutuamente exclusivas
- Caso contrário, não interferem uma com a outra
 - Ambas podem aparecer num plano solução
- Dois literais no mesmo nível estado são **mutex** se
 - *Suporte inconsistente*: um é a negação do outro, ou todas as maneiras de os alcançar são mutex

Exemplo

- Aumentar o grafo para indicar mutexes
- *carry* é mutex com acção de persistência para *garbage*
 - efeitos inconsistentes
- *dolly* é mutex com *wrap*
 - interferência
- $\neg$ *quiet* é mutex com *present*
 - suporte inconsistente
- quer *cook* quer *wrap* é mutex com uma acção de persistência

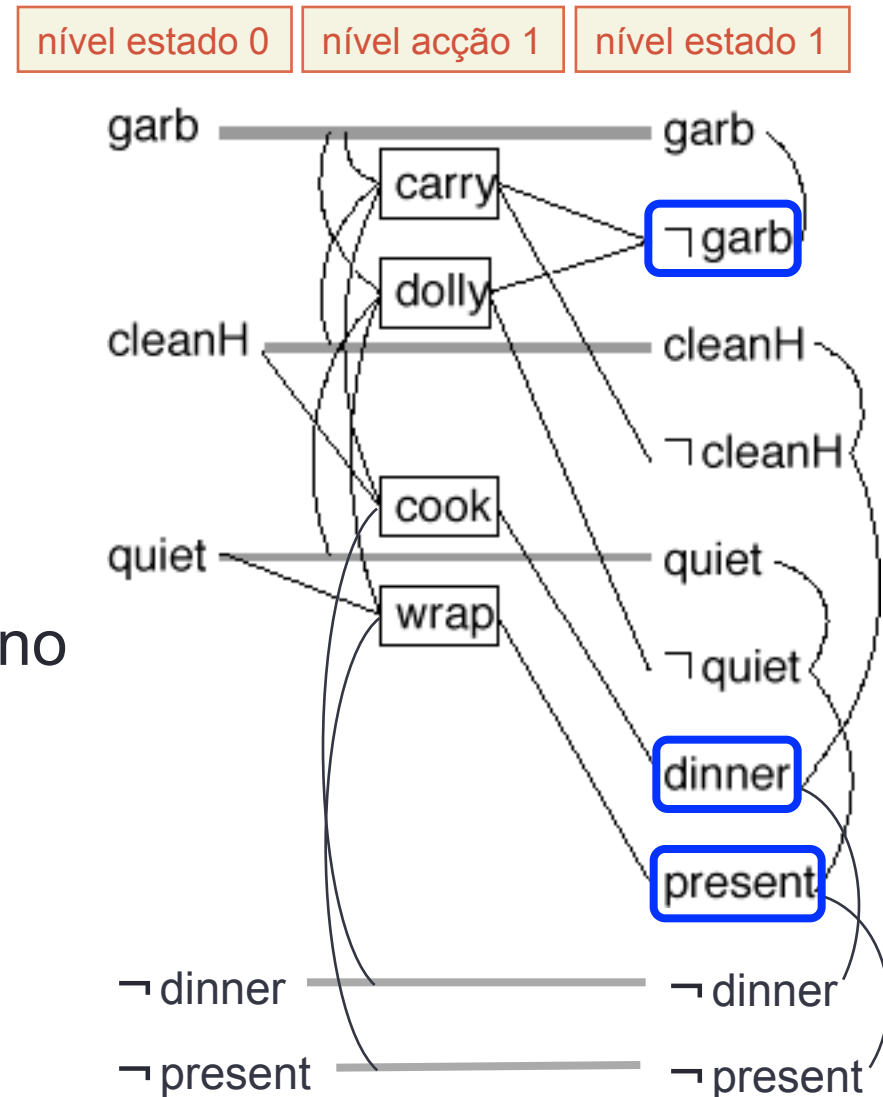
Acção	Precondições	Efeitos
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	<i>nenhum</i>	$\neg$ garbage, $\neg$ cleanHands
dolly()	<i>nenhum</i>	$\neg$ garbage, $\neg$ quiet

Juntamente com as acções de persistência



Exemplo

- Verificar se existe um plano possível
- O objectivo é
 - $\{\neg \text{garbage}, \text{dinner}, \text{present}\}$
- Repare-se que
 - Todos são possíveis em s_1
 - Nenhum é mutex com qq. outro
- Logo, há hipóteses de um plano existir
- Tentar encontrá-lo
 - Extracção da solução



Extracção da solução

Os objectivos que estamos a tentar atingir

O nível de estado s_j

procedure Solution-extraction(g, j)

se $j=0$ então devolver solução

para cada literal l em g

escolher não deterministicamente
uma acção a a usar no estado s_{j-1} para
alcançar l

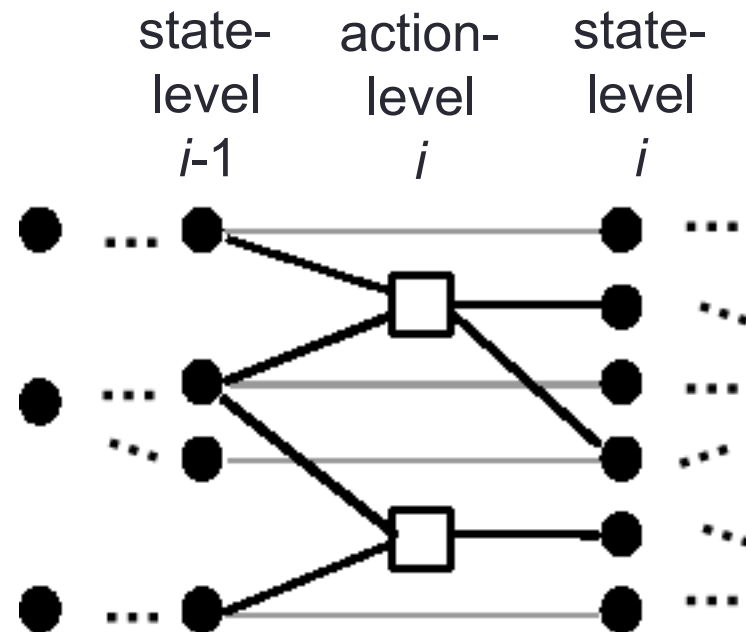
Uma acção real ou de persistência

se qualquer par de acções é mutex
então retroceder

$g' := \{ \text{as precondições} \\ \text{das acções escolhidas} \}$

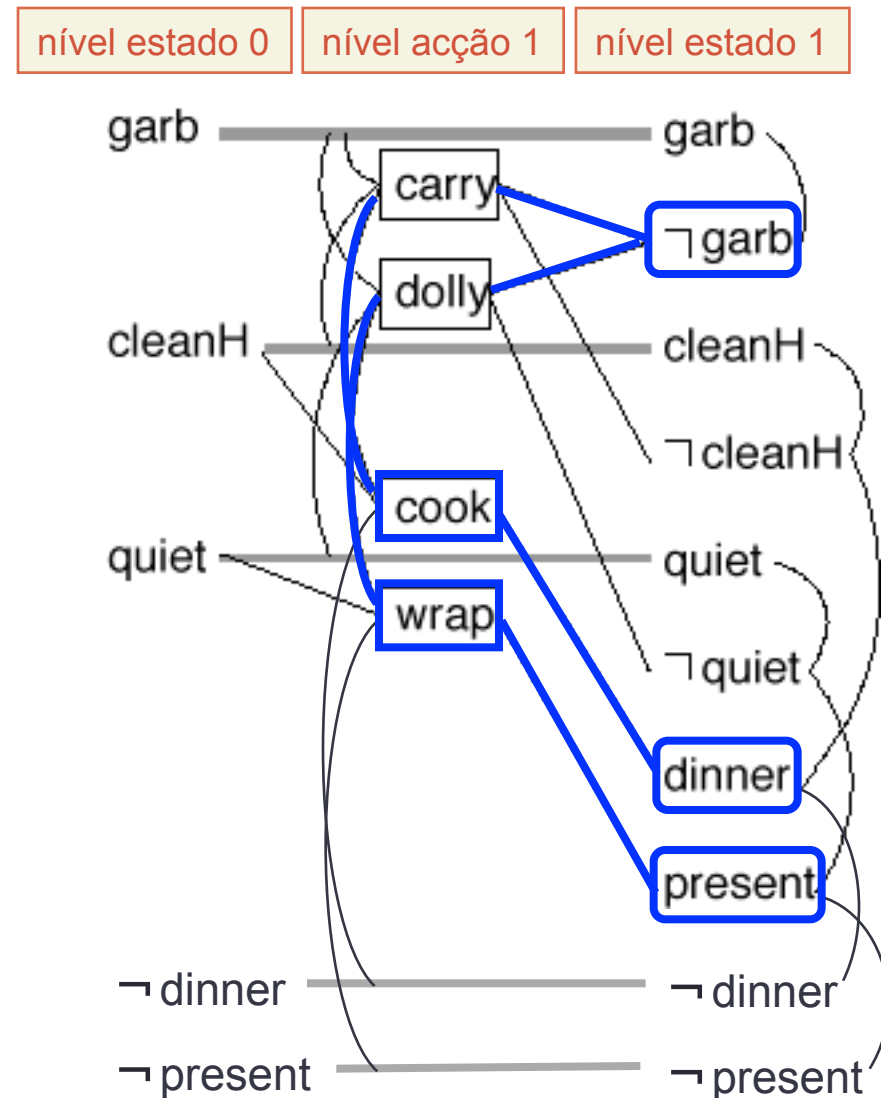
Solution-extraction($g', j-1$)

end Solution-extraction



Exemplo

- Dois conjuntos de acções para os objectivos no nível estado 1
- Nenhum serve: ambos os conjuntos contêm acções que são mutex



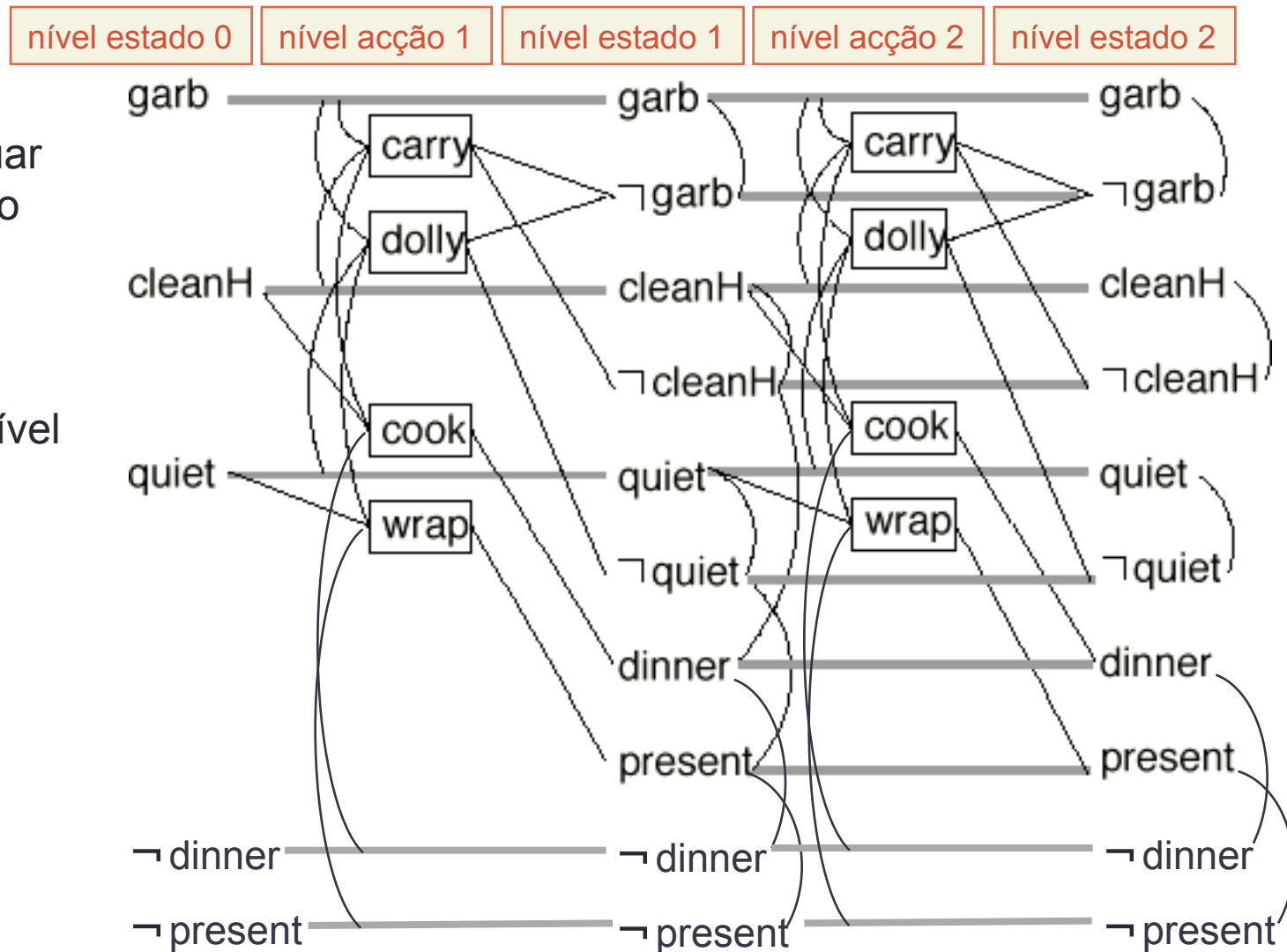
Relembrar como o algoritmo funciona

procedure Graphplan:

- for $k = 0, 1, 2, \dots$
 - *Expansão do grafo:*
 - criar um “grafo de planeamento” contendo k “níveis”
 - Verificar se o grafo de planeamento satisfaz uma condição necessária (mas insuficiente) para a existência de um plano
 - Se satisfizer, então
 - *Extrair solução:*
 - Procura regressiva, modificada para considerar apenas as acções no grafo de planeamento
 - Caso se encontre uma solução, então devolvê-la

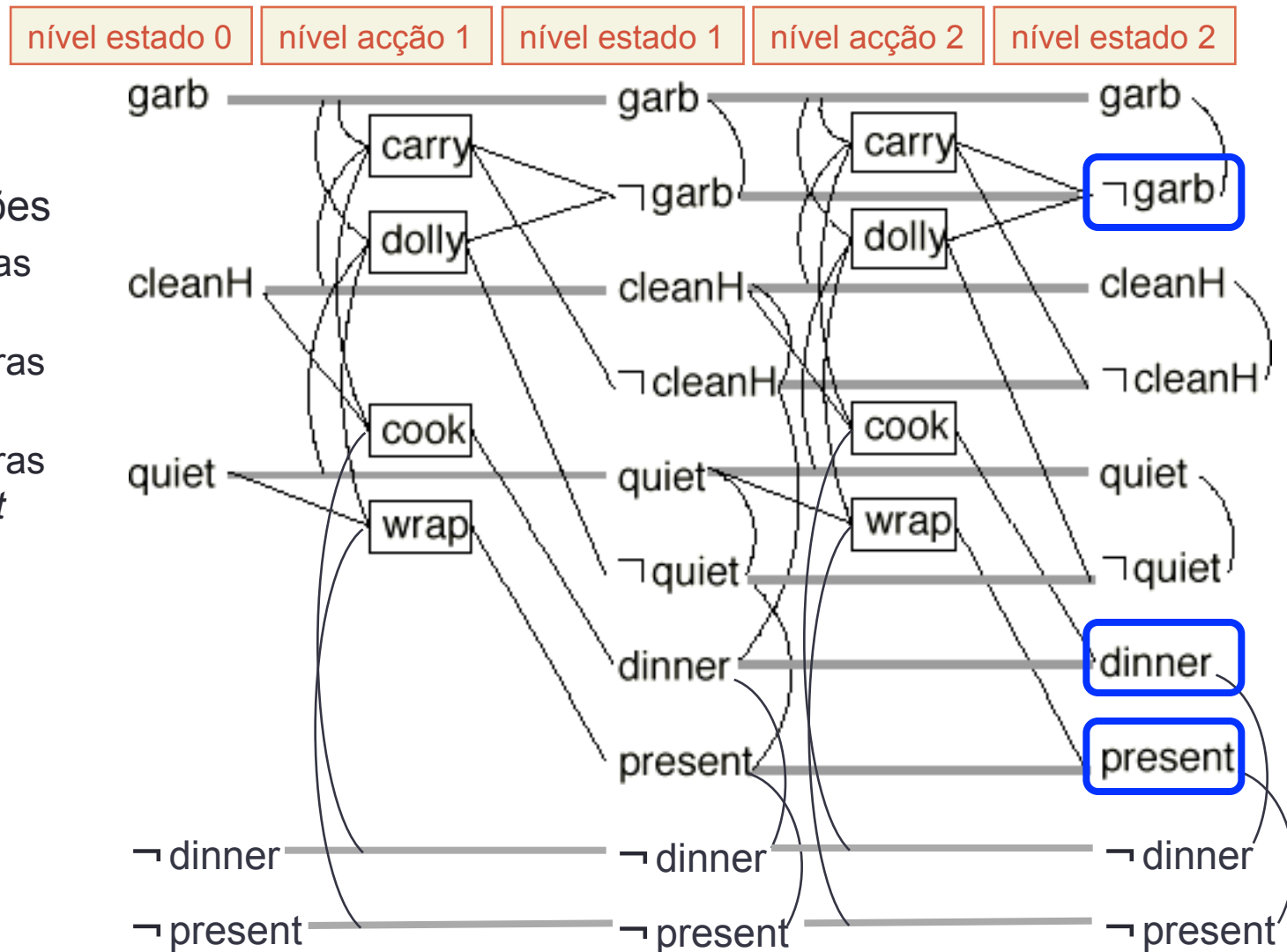
Exemplo

- Iterar e efectuar uma expansão do grafo adicional
- Gerar outro nível acção e outro nível estado



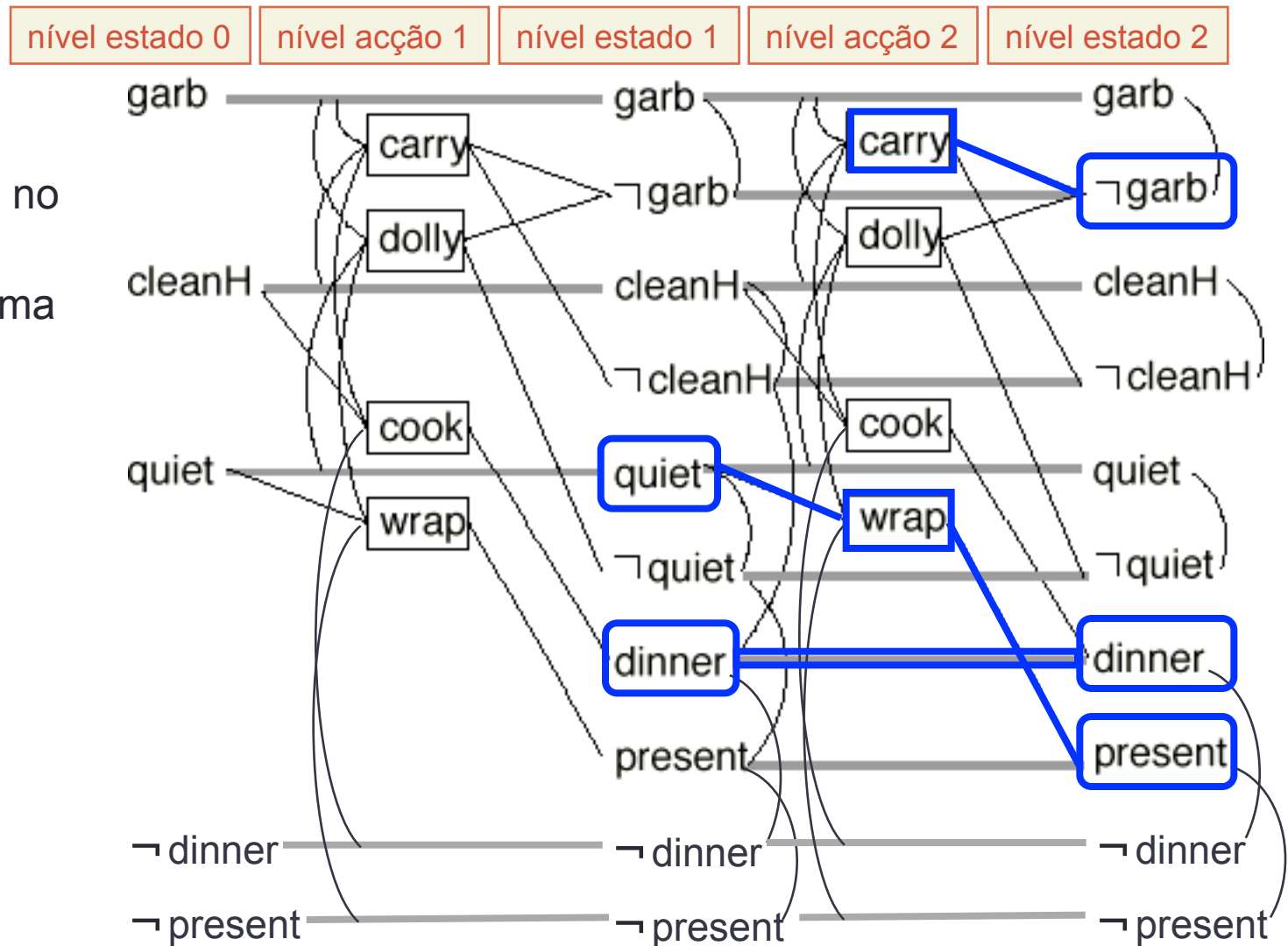
Exemplo

- Extração da solução
- 12 combinações
 - Três maneiras para $\neg garb$
 - Duas maneiras para *dinner*
 - Duas maneiras para *present*



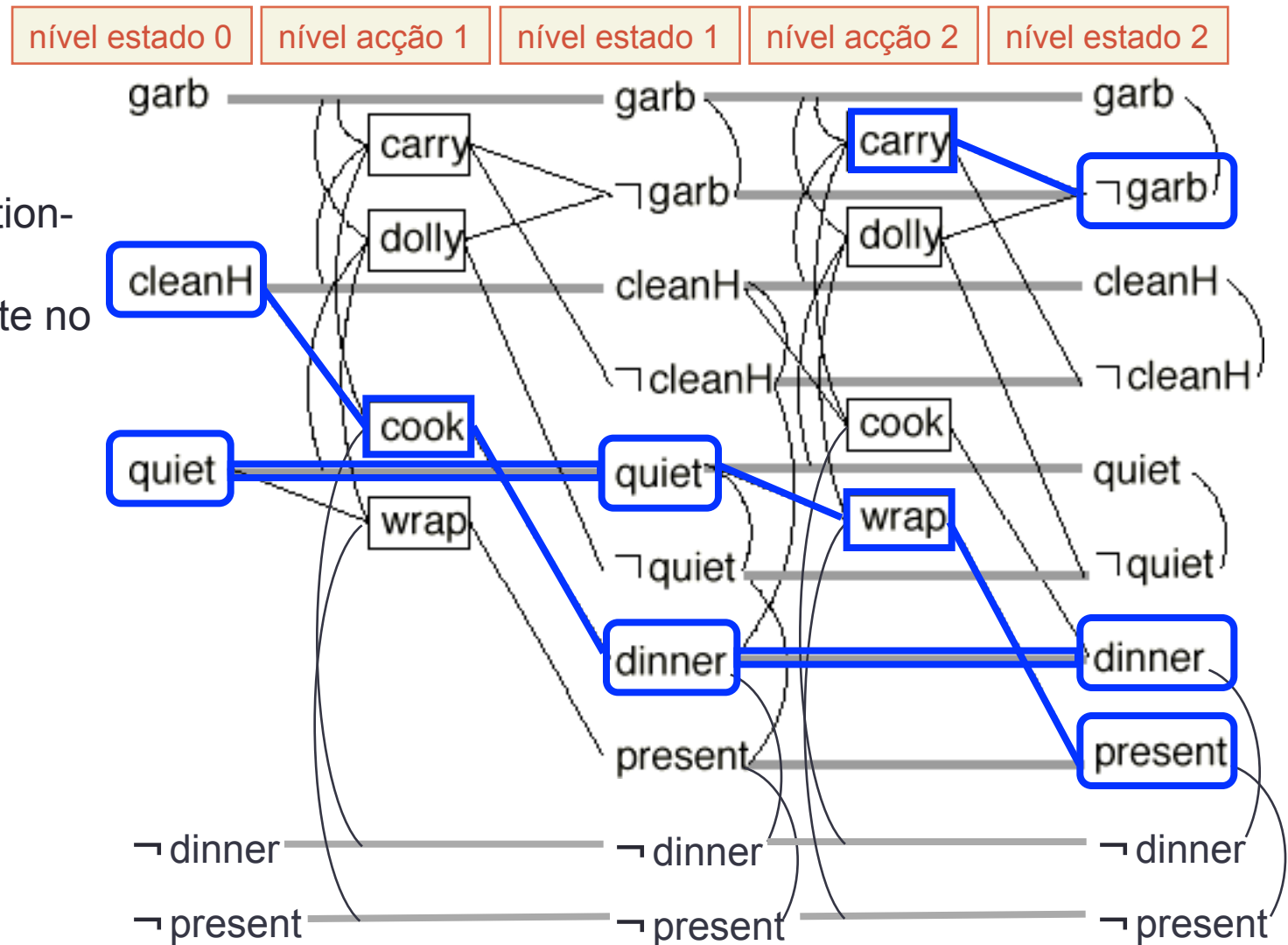
Exemplo

- Algumas combinações parecem bem no nível 2
- Assinala-se uma delas



Exemplo

- Chamar Solution-Extraction recursivamente no nível 2
- Tem sucesso
- Solução cujo *comprimento paralelo* é 2



POP – EXEMPLOS

Exemplo: Casa e Descasa

- A agência de matrimónios Casa&Descasa decidiu investir num sistema de Inteligência Artificial para melhorar o atendimento aos seus clientes. O sistema mantém informação sobre os homens e mulheres registados no sistema, os casais (heterossexuais) e indivíduos solteiros. O objectivo do sistema é aconselhar casamentos e divórcios sugerindo ainda "falecimentos" de pessoas. A poligamia não é permitida.
- As acções são: casar, divorciar, matar homem casado e matar mulher casada.
- Por exemplo, o sistema devera ser capaz de indicar os passos que levam o individuo
- Aníbal, homem e solteiro, a ficar casado com Belarmina que inicialmente está casada com César
- Existe um número infinito de planos?
- Porque e que não é possível poligamia ?

Exemplo: Casa e Descasa

Acção: Casar(H,M)

Precondição: solteiro(H), homem(H), solteiro(M), mulher(M)

Efeito: casados(H,M), $\neg$ solteiro(H) $\neg$ solteiro(M)

Acção: Divorciar(H,M)

Precondição: casados(H,M)

Efeito: solteiro(H), solteiro(M) $\neg$ casados(H,M)

Acção: Matar_Homem_Casado(H)

Precondição: casados(H,M)

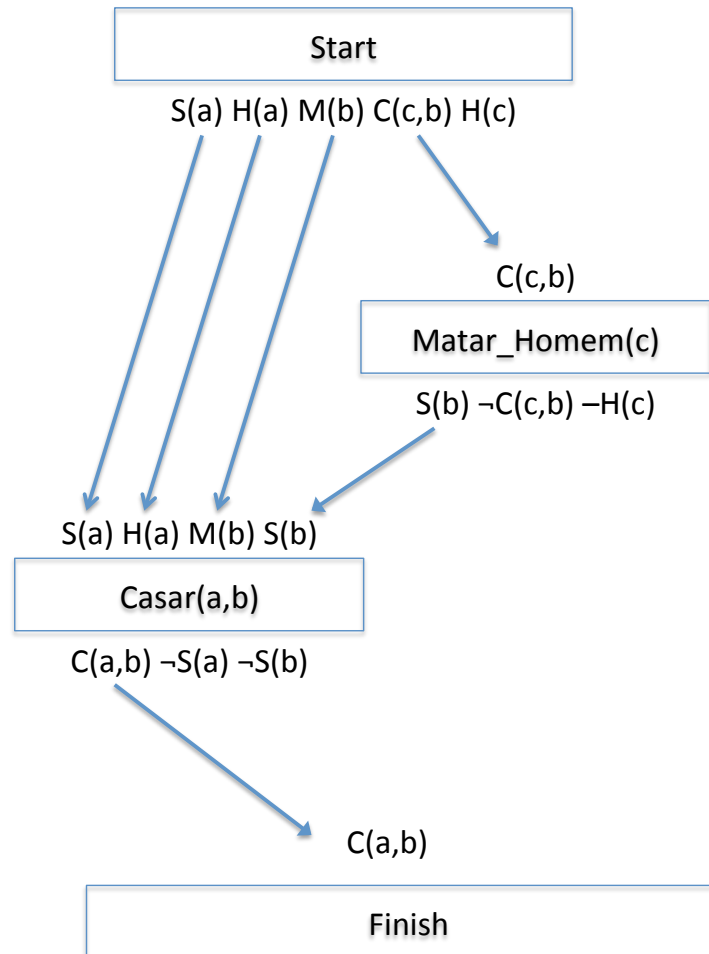
Efeito: solteiro(M), $\neg$ casados(H,M), $\neg$ homem(H)

Acção: Matar_Mulher_Casada(M)

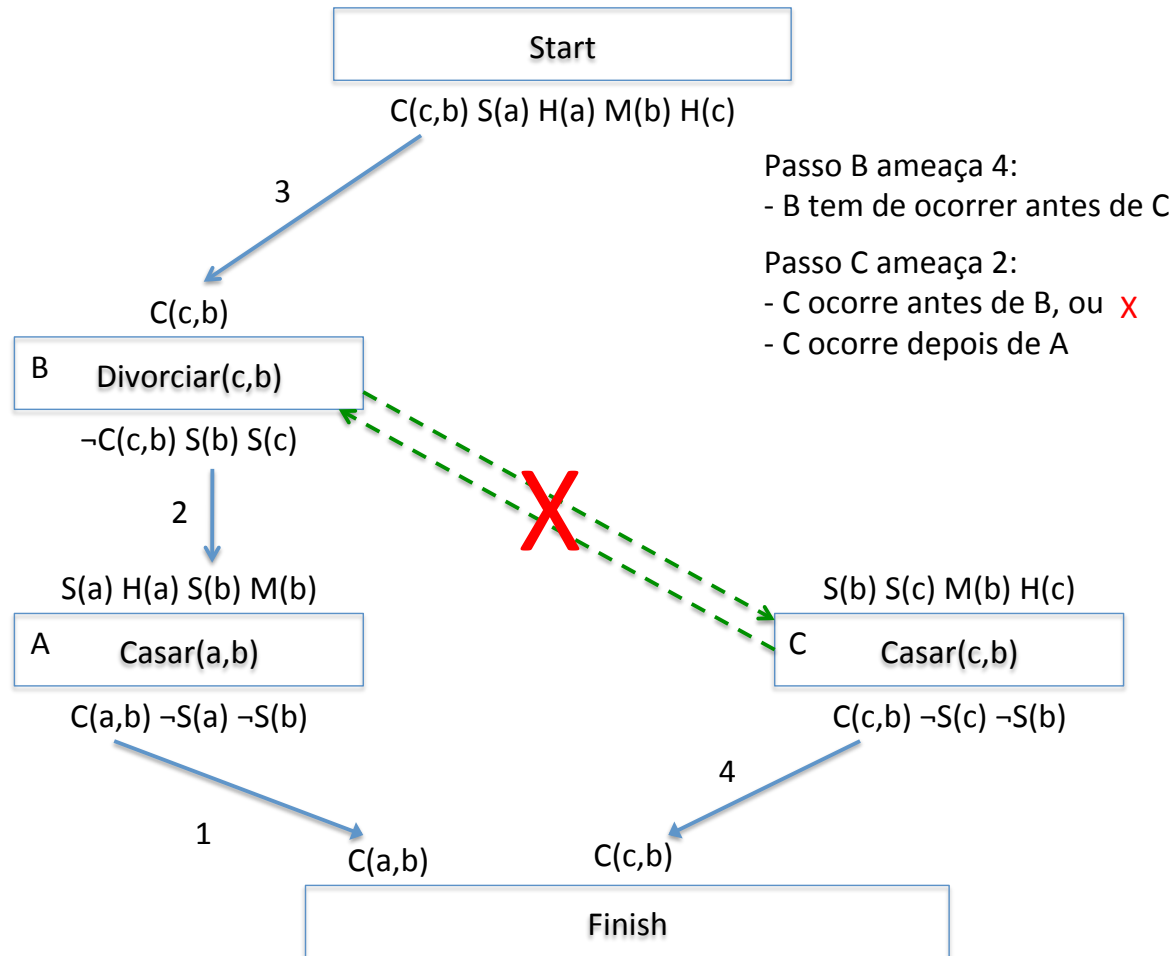
Precondição: casados(H,M)

Efeito: solteiro(H), $\neg$ casados(H,M), $\neg$ mulher(M)

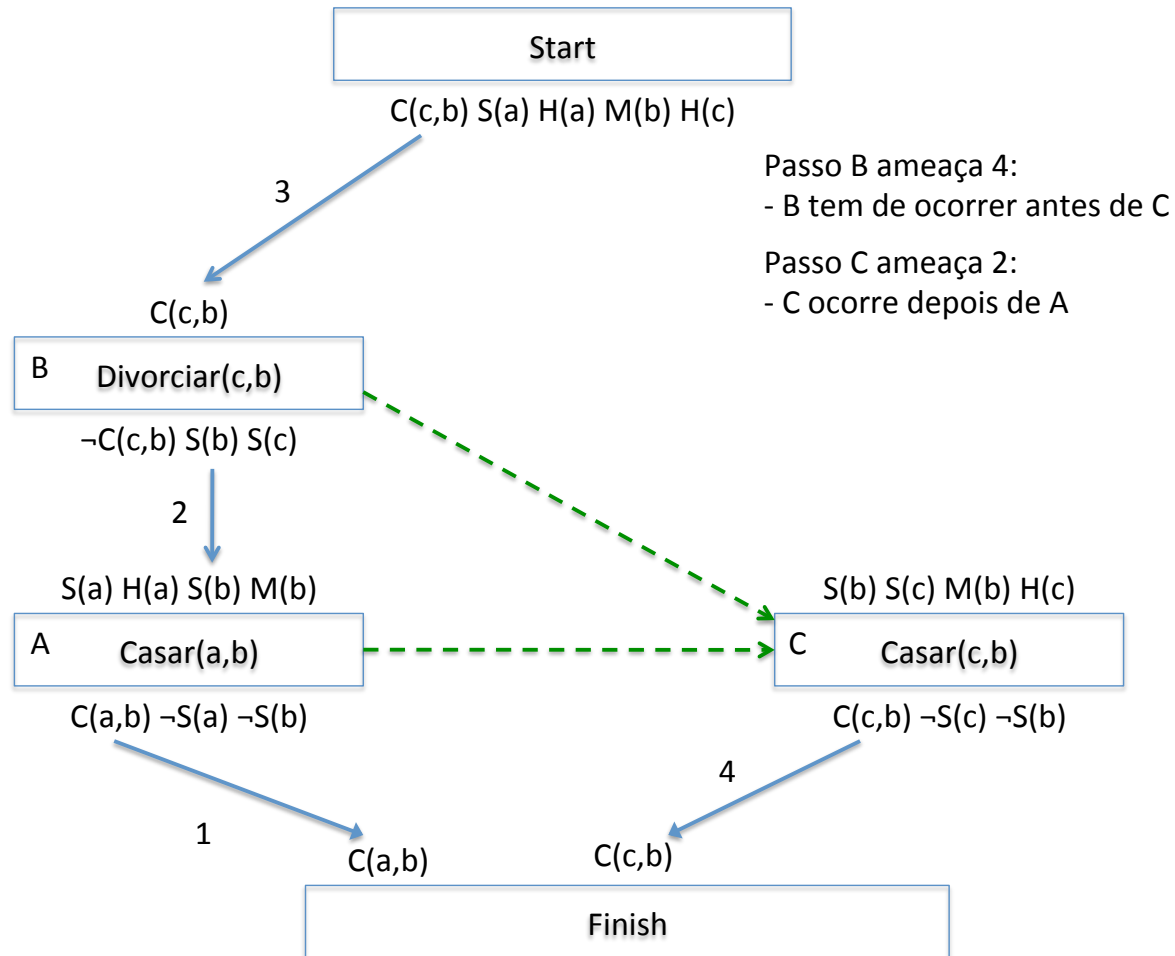
Exemplo: Casa e Descasa



Exemplo: Casa e Descasa



Exemplo: Casa e Descasa



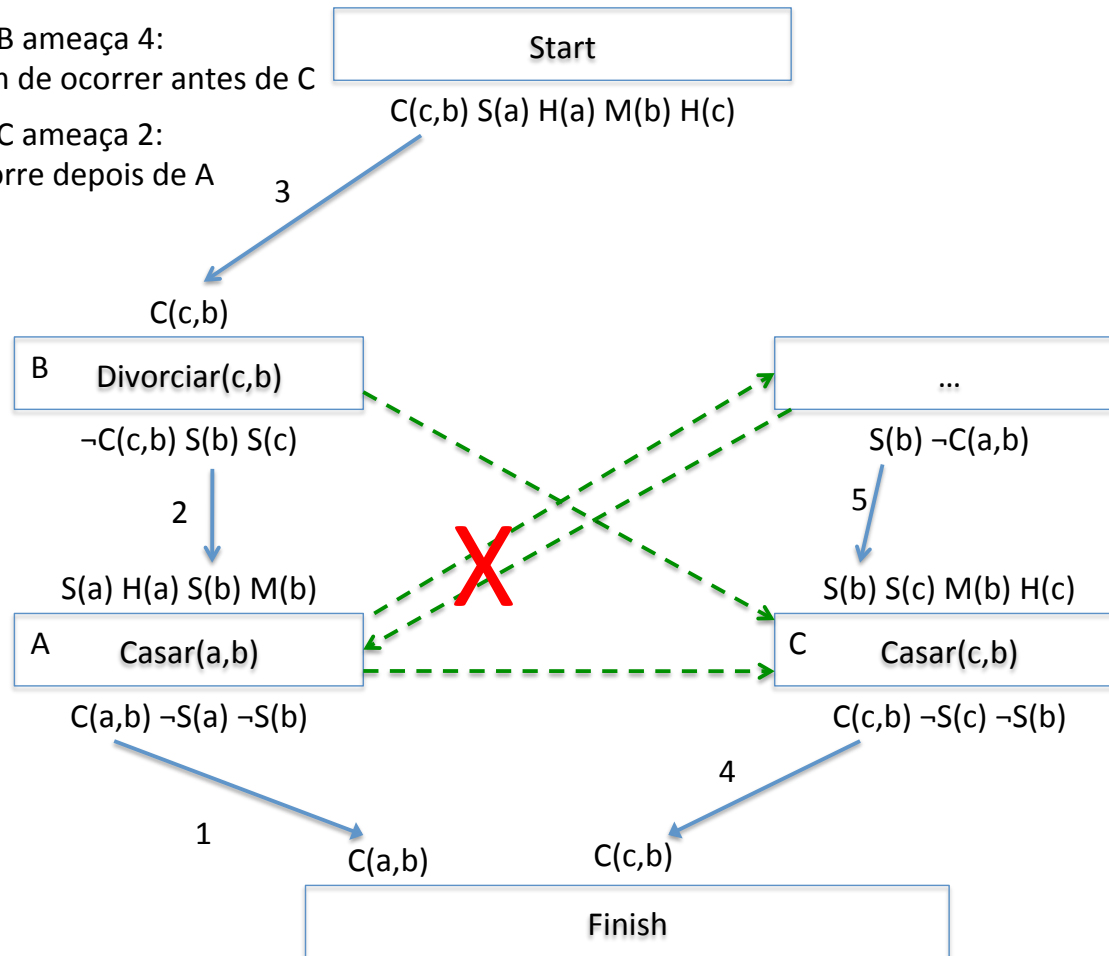
Exemplo: Casa e Descasa

Passo B ameaça 4:

- B tem de ocorrer antes de C

Passo C ameaça 2:

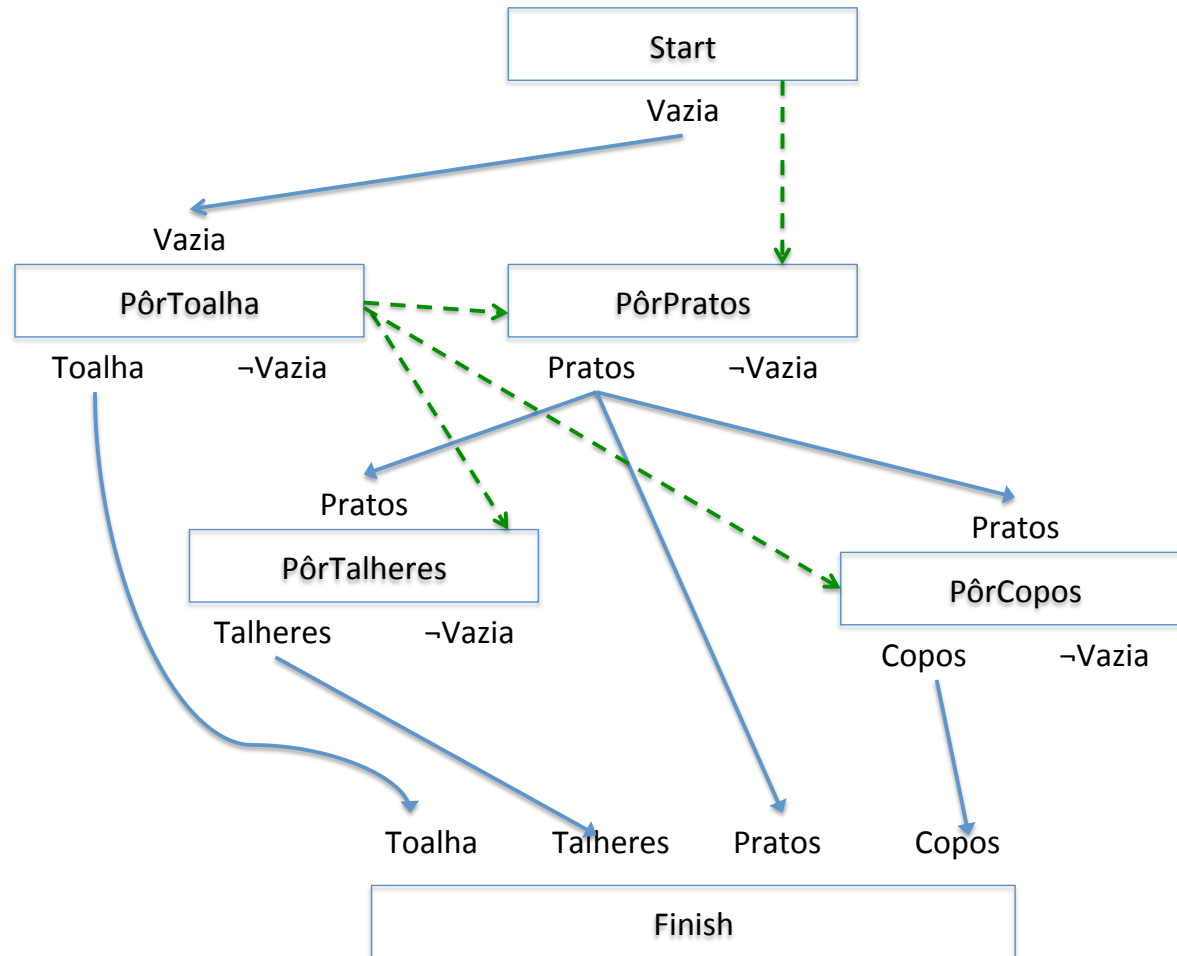
- C ocorre depois de A



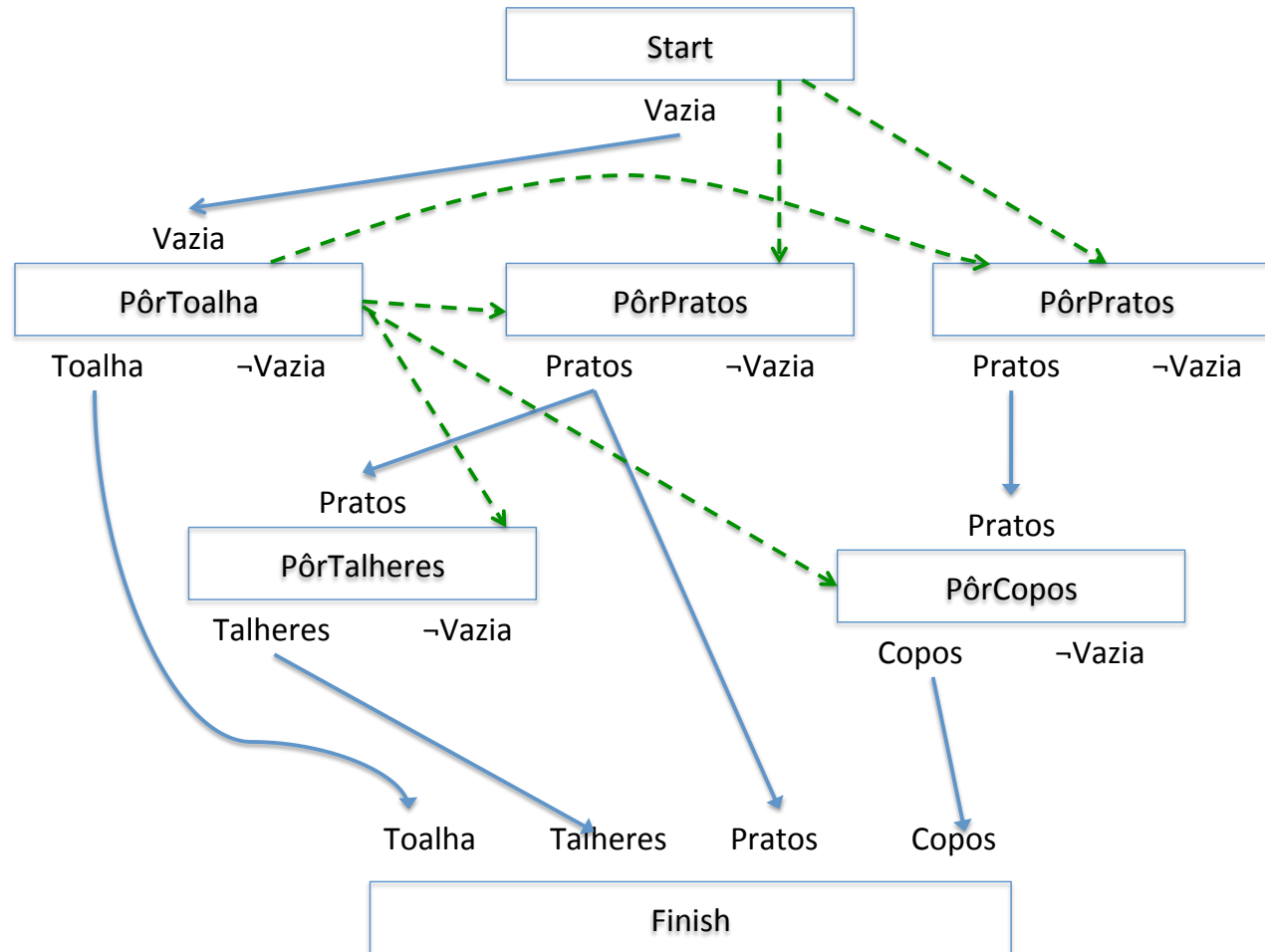
Exemplo: pôr a mesa

- O Pedro decidiu ajudar a mãe a pôr a mesa, tendo-lhe perguntado como é que o devia fazer, e recebido as seguintes dicas:
 - Podes pôr uma toalha de mesa, que é opcional, mas só o podes fazer se não estiver nada em cima da mesa.
 - Para a mesa ficar arranjadinha só deves colocar os talheres e os copos depois dos pratos.
 - A ordem de colocação dos talheres e dos copos é indiferente, mas só depois dos pratos!
- Temos que ajudar o Pedro a encontrar um plano.
 - Podem existir planos com acções repetidas?

Exemplo: pôr a mesa



Exemplo: pôr a mesa



Exemplo: supermercado

Estado inicial: At(Home), Sells(HWS, Drill), Sells(SM,Milk), Sells(SM,Ban)

Acções:

Acção: Buy(x)

Precondição: At(p), Sells(p, x)

Efeito: Have(x)

Acção: Go(x)

Precondição: At(y)

Efeito: Have(x)

Objectivo: Have(Milk), At(Home), Have(Ban.), Have(Drill)

Exemplo: supermercado

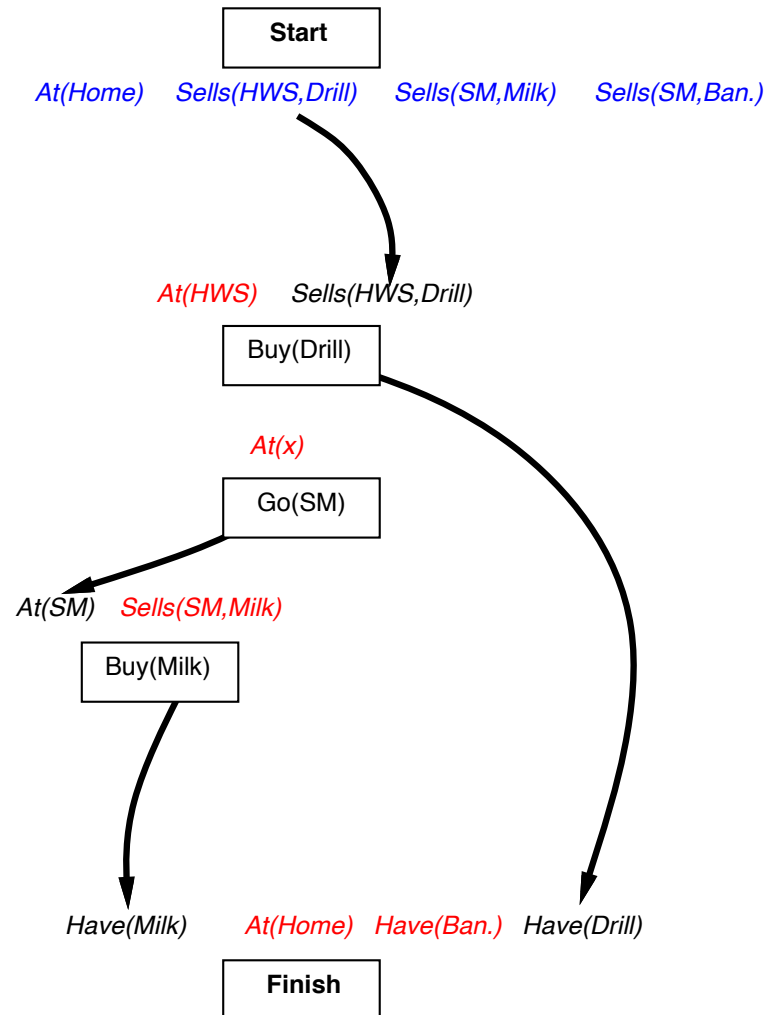
Start

At(Home) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)

Have(Milk) At(Home) Have(Ban.) Have(Drill)

Finish

Exemplo: supermercado



Exemplo: supermercado

