Grafos de Planejamento

História

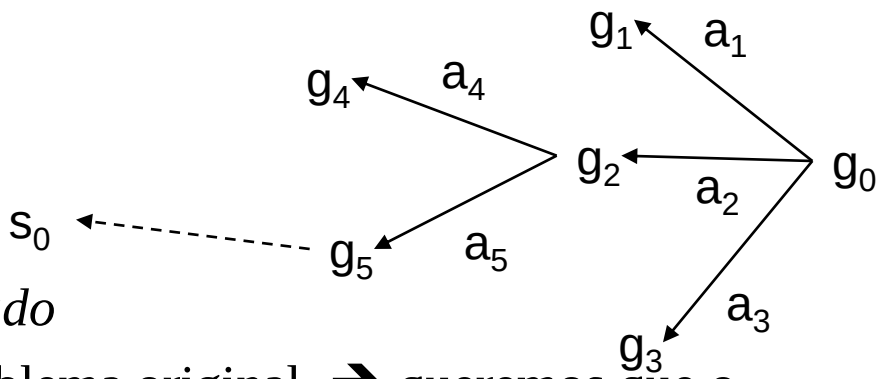
- Antes de *Graphplan*, muitos pesquisadores de planejamento estavam trabalhando com variações de planejadores do tipo POP
  - POP, SNLP, etc.
- *Graphplan* causou um grande impacto por ser muito mais eficiente: com a construção do grafo de planejamento
- Existem muitos planejadores que usam as idéias do Graphplan
  - IPP, STAN, GraphHTN, SGP, Blackbox, Medic, TGP, LPG, FF
  - … a maioria são muito mais eficientes do que o *Graphplan* original

Outline

- O algoritmo Graphplan
- Construção de grafos de planejamento
- Exclusão Mútua (mutex)
- Extração da solução
- Usando o grafo de planejamento para calcular heuristicas

Grafo de Planejamento: motivação

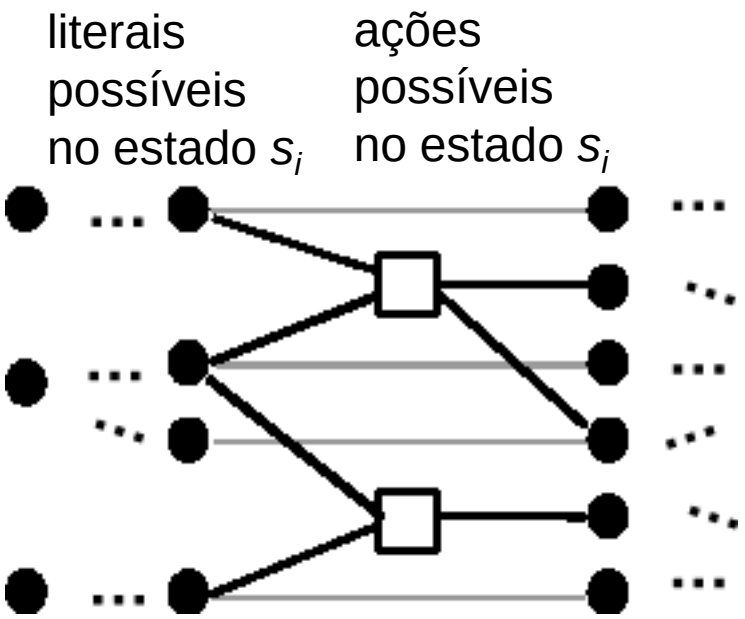
- Uma das razões da ineficiência dos algoritmos de busca é o ***fator de ramificação*** da árvore de busca, isto é, o número de filhos de cada nó
- Por exemplo: a busca regressiva pode selecionar muitas ações que não atingem o estado inicial
- Como reduzir o fator de ramificação?



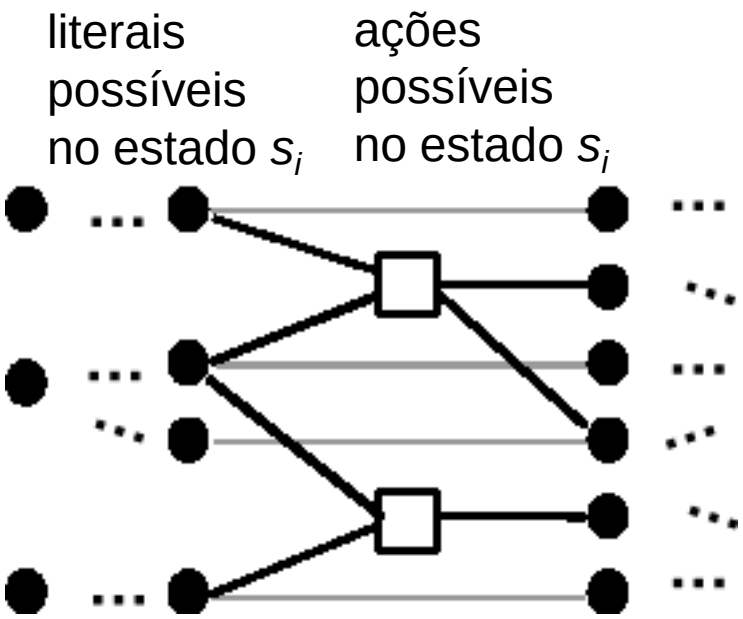
1. Primeiro você cria um *problema relaxado*
   - Remova algumas restrições do problema original ➔ queremos que o problema relaxado seja mais fácil de se resolver (tempo polinomial)

2. Depois, faça uma versão modificada da busca original
   - Restrinja o seu espaço de busca incluindo somente as ações que ocorrem no problema relaxado

Grafo de Planejamento



Grafo de Planejamento



Contém todos os literais que poderiam ser verdadeiros no tempo i (podem estar p e –p presentes)

Contém todas as ações que poderiam ter suas precondições satisfeitas no tempo i

GRAPHPLAN
[Blum&Furst 1997]

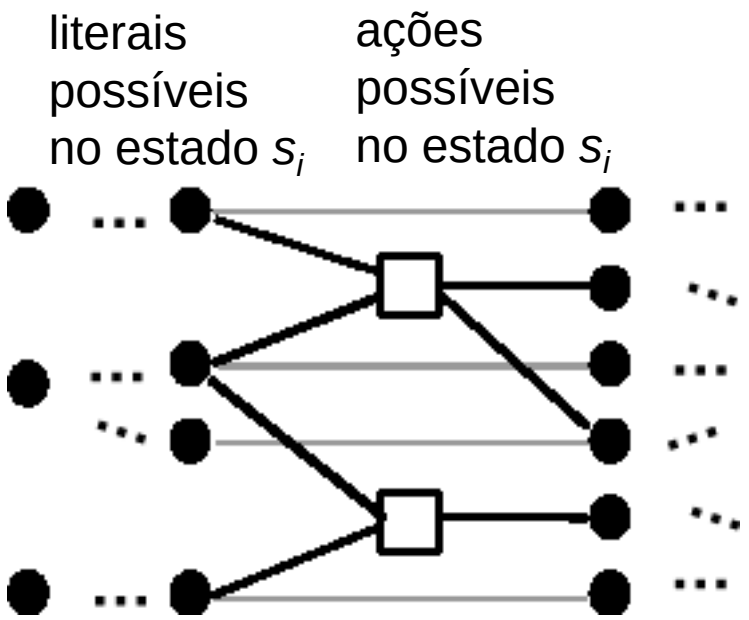
- O planejador *Graphplan* alterna entre duas fases:
  - Fase 1: expansão do grafo de planejamento: estende o grafo progressivamente até atingir uma condição necessária, mas não suficiente, para a existência de um plano (complexidade polinomial).
  - Fase 2: extração da solução: faz uma busca regressiva no grafo, a procura de um plano solução; se não existir um plano, *Graphplan* volta a expandir o grafo (complexidade exponencial).

Algoritmo Graphplan

- para $k = 0, 1, 2, \ldots$
  - ***Expansão do Grafo***:
    - » crie um "grafo de planejamento" que contém k "níveis"
  - Cheque se o grafo satisfaz uma condição necessária (mas não suficiente) para existir um plano solução

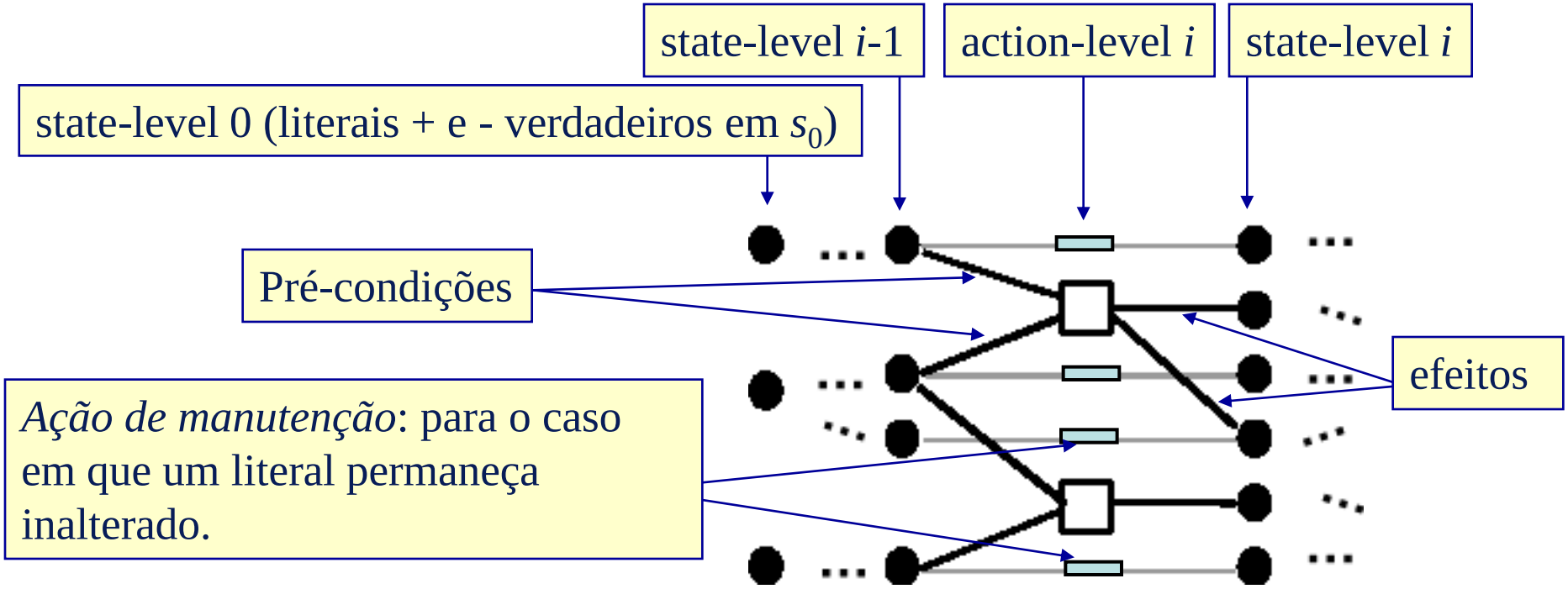
problema relaxado

  - Se a condição for verdadeira, então
    - » Faça ***extração da solução*** :
      - Busca regressiva, modificada para considerar apenas as ações no grafo de planejamento
      - Se existir uma solução então devolva o plano



O grafo de planejamento

- Alterna níveis de literais e ações
  - Nós no nível *i*: Todas as ações que podem ocorrer a cada instante (passo) do plano
  - Nós no nível *i:* Todos os literais que podem ser adicionadas por aquelas ações
  - Arestas: precondições e efeitos



Exemplo: Jantar Surpresa

» Daniel Weld (U. of Washington)

- Suponha que você quer preparar um jantar surpresa para sua esposa (que está dormindo)

  s_0 = {garbage, cleanHands, quiet}

  g = {dinner, present, ¬garbage}

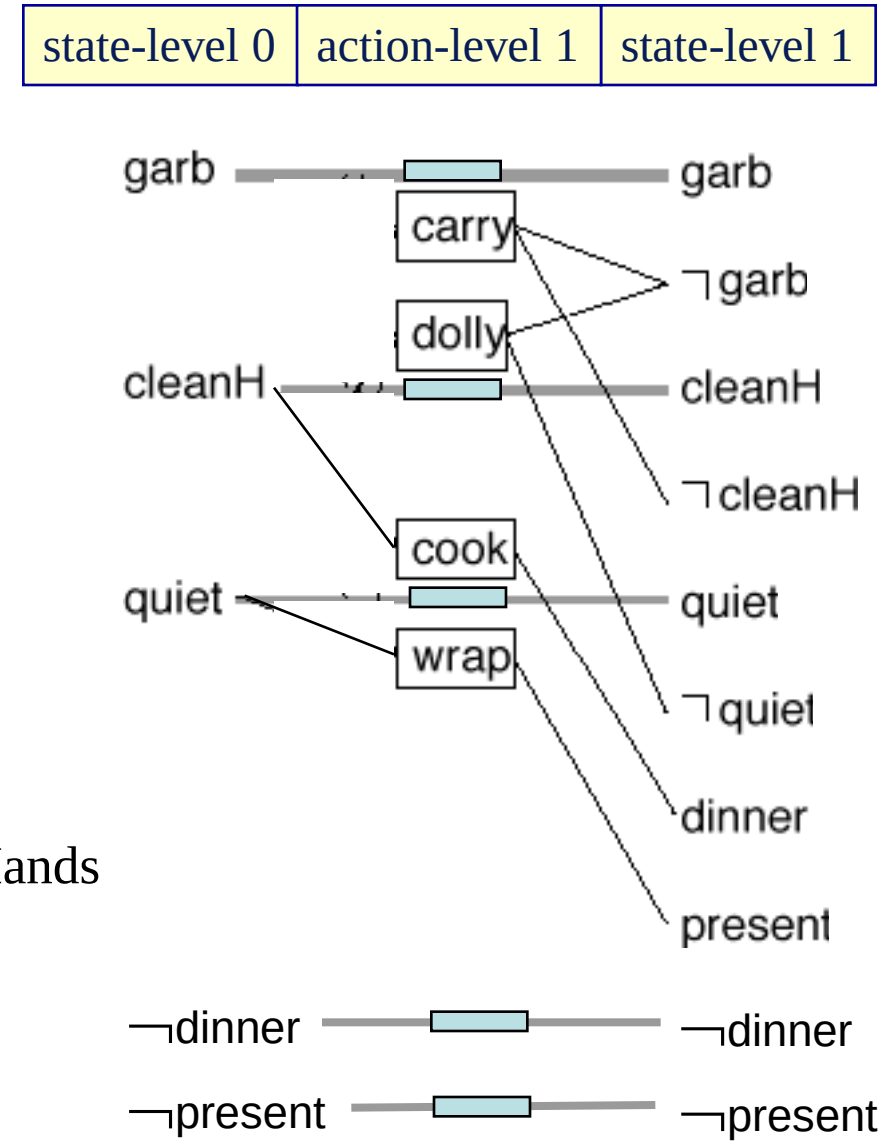
Ações	Pré-condições	Efeitos
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	*none*	¬garbage, ¬cleanHands
dolly()	*none*	¬garbage, ¬quiet

Também existem ações de manutenção (no-op) para cada literal. Por exemplo:

Ações	Pré-condições	Efeitos
no-op-cleanHands	cleanHands	cleanHands
no-op-quiet	quiet	quiet
no-op-not-quiet	¬quiet	¬quiet
…		

Exemplo (continuação)

- *state-level* 0:
  {todos os átomos em s_0} U
  {negações de todos os átomos
  que não estão em s_0}
- *action-level* 1:
  {todas as ações cujas pré-condições
  são satisfeitas em s_0}
- *state-level* 1:
  {todos os efeitos de todas as
  ações em *action-level* 1}



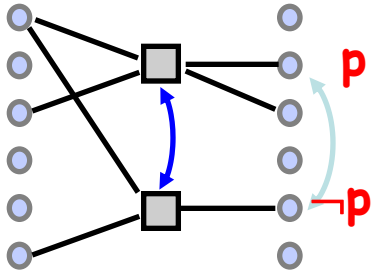
Ações	**Pré-condições**	**Efeitos**
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	*none*	¬garbage, ¬cleanHands
dolly()	*none*	¬garbage, ¬quiet

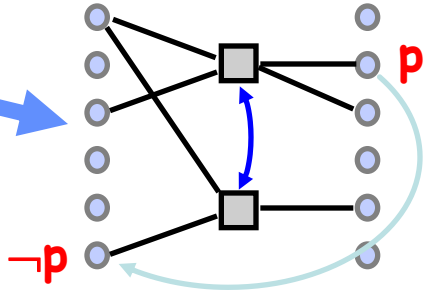
… e todas as ações de manutenção.

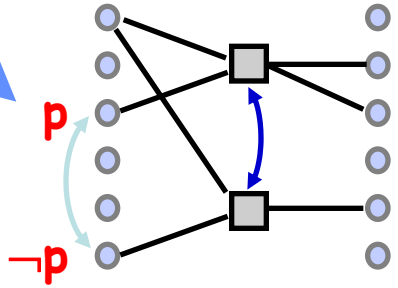
Exclusão Mútua (Mutex)

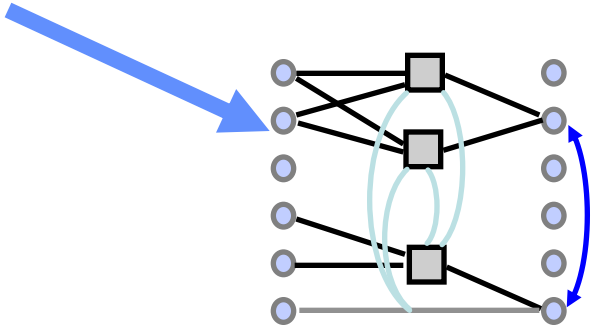
- Duas ações no mesmo nível são mutex se
  - ***Efeitos inconsistentes***: um efeito de uma ação nega um efeito de outra
  - ***Interferência***: uma ação elimina uma pré-condição de outra
  - ***Necessidades que competem***: ações com pré-condições que estão em mutex
  - Caso contrário, as ações não interferem entre si, isto é:
    - » Ambas podem fazer parte de um plano solução
- Dois literais num mesmo nível são mutex se
  - ***Suporte inconsistente***: um é a negação do outro ou

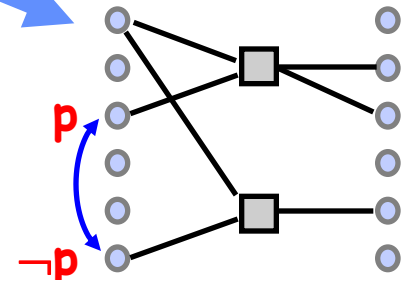
    todas as maneiras de satisfazê-los são mutex







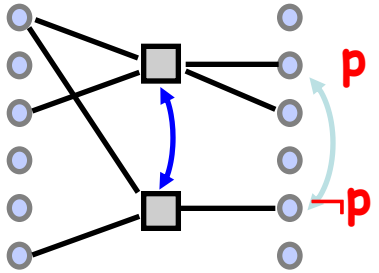


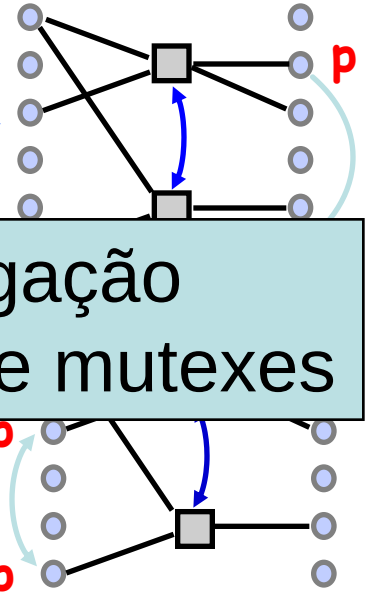


Exclusão Mútua (Mutex)

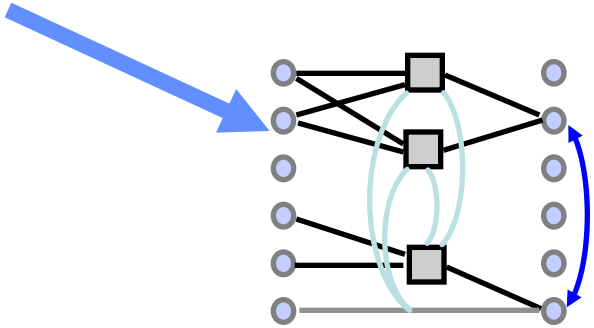
- Duas ações no mesmo nível são mutex se
  - ***Efeitos inconsistentes***: um efeito de uma ação nega um efeito de outra
  - ***Interferência***: uma ação elimina uma pré-condição de outra
  - ***Necessidades que competem***: ações com pré-condições que estão em mutex
  - Caso contrário, as ações não interferem entre si isto é:
    - » Ambas podem fazer parte de um plano solução
- Dois literais num mesmo nível são mutex se
  - ***Suporte inconsistente***: um é a negação do outro ou

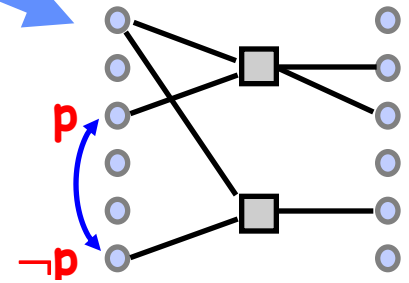
    todas as maneiras de satisfazê-los são mutex





Propagação recursiva de mutexes





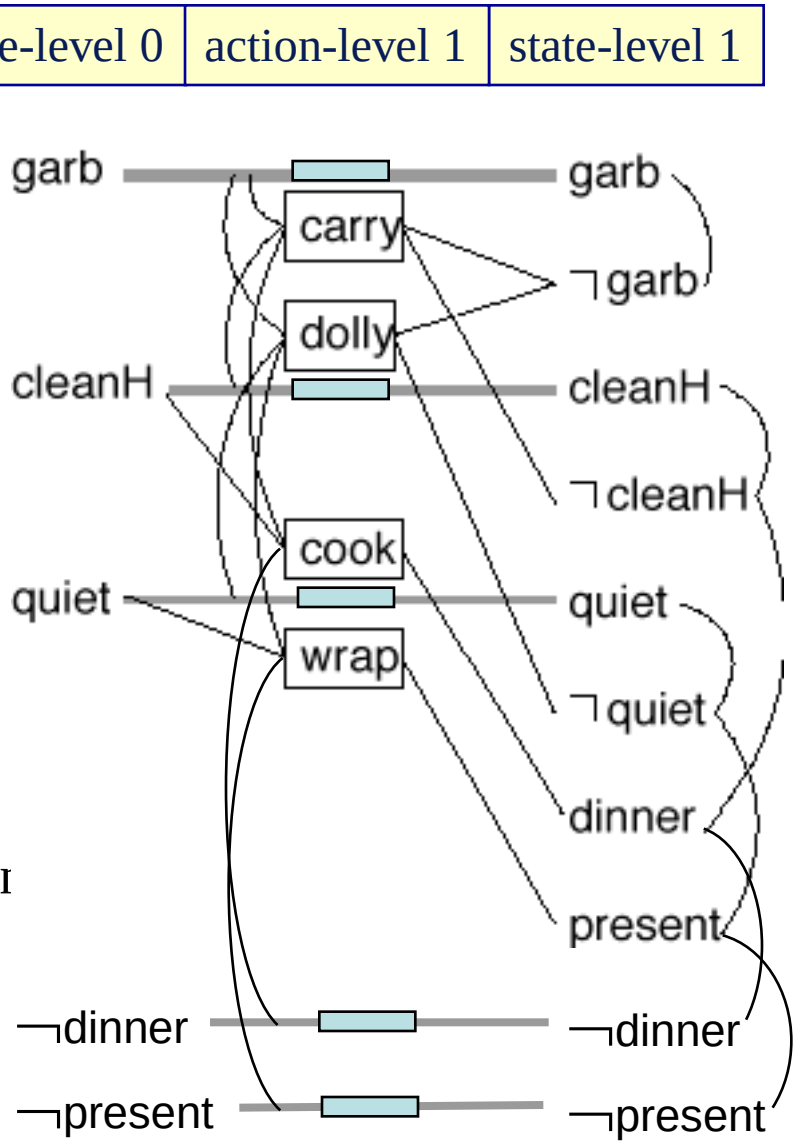
Exemplo (continuação)

Identificação de mutex no grafo:

- *carry* está em mutex com a ação de manutenção para *garbage*
  - efeitos inconsistentes
- *dolly* está em mutex com *wrap*
  - interferência
- *~quiet* está em mutex com *present*
  - suporte inconsistente
- as ações *cook* e *wrap* estão em mutex ações de manutenção

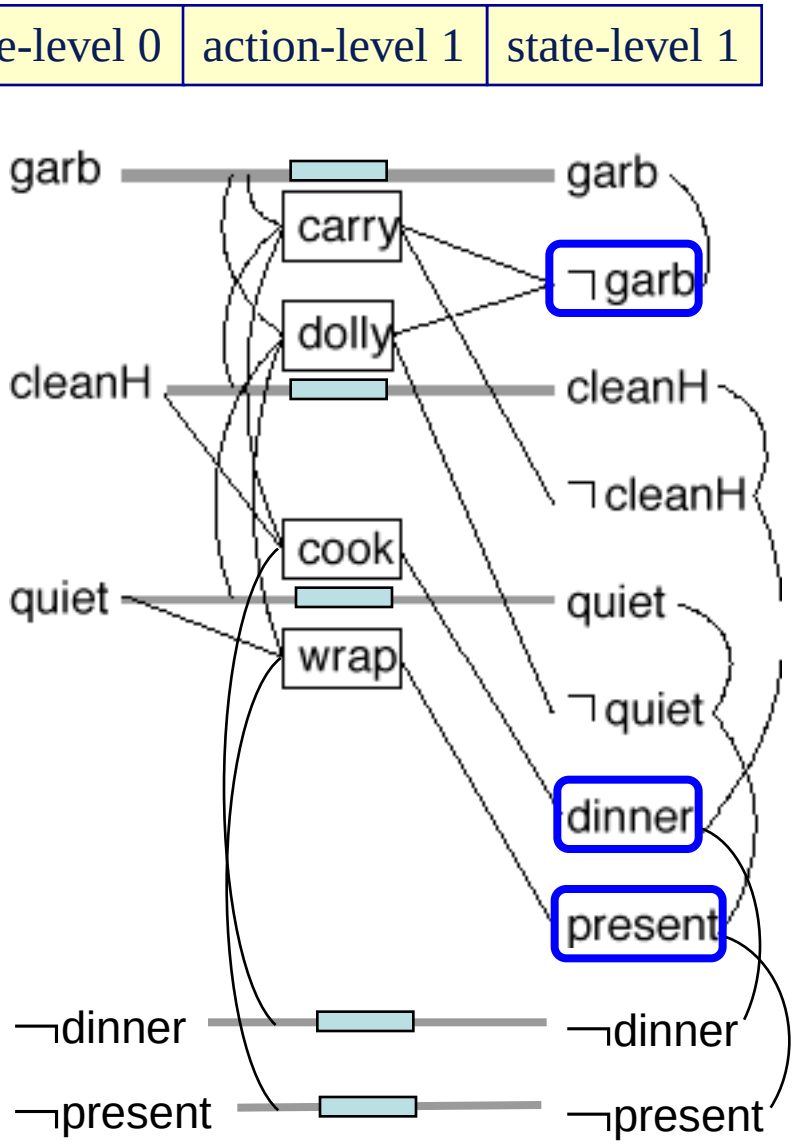
Ações	**Pré-condições**	**Efeitos**
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	*none*	¬garbage, ¬cleanHar
dolly()	*none*	¬garbage, ¬quiet

… e todas as ações de manutenção.



Exemplo (continuação)

- Cheque se pode existir um plano para a meta do problema:
  - {¬*garbage*, *dinner*, *present*}
- Note que
  - Todos os literais da meta estão no *state-level* 1
  - Nenhum deles estão em mutex
- Então há uma chance de existir um plano
  - Tente extrair uma solução



Extração da Solução

O conjunto de metas que estamos tentando alcançar

O nível do estado s_j

procedimento Extração-da-Solução(g,j)

se j=0 então devolva a solução

Para cada literal l em g

escolha não-determinística de uma ação para ser usada no estado s_{j-1} para atingir l

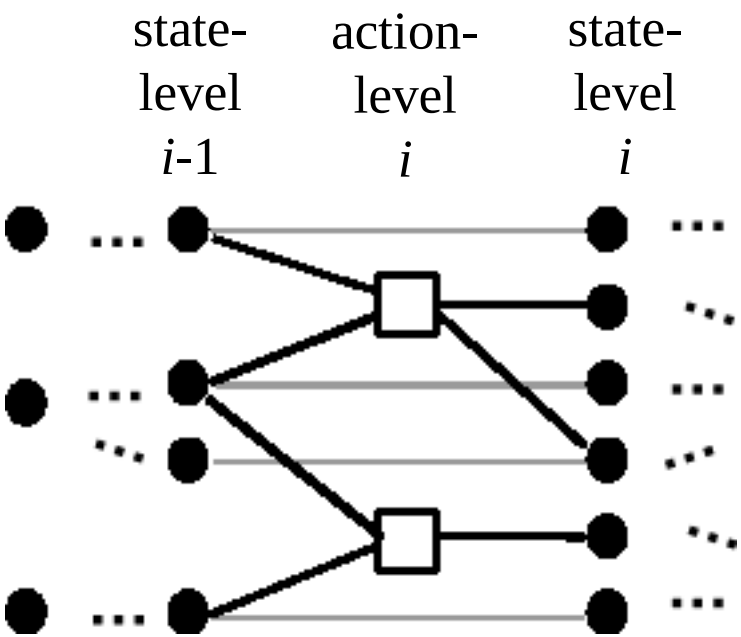
Se qualquer par possível de ações está em mutex então *backtracking*

g' := {pré-condições das ações selecionadas}

Extração-da-Solução(g', j–1)

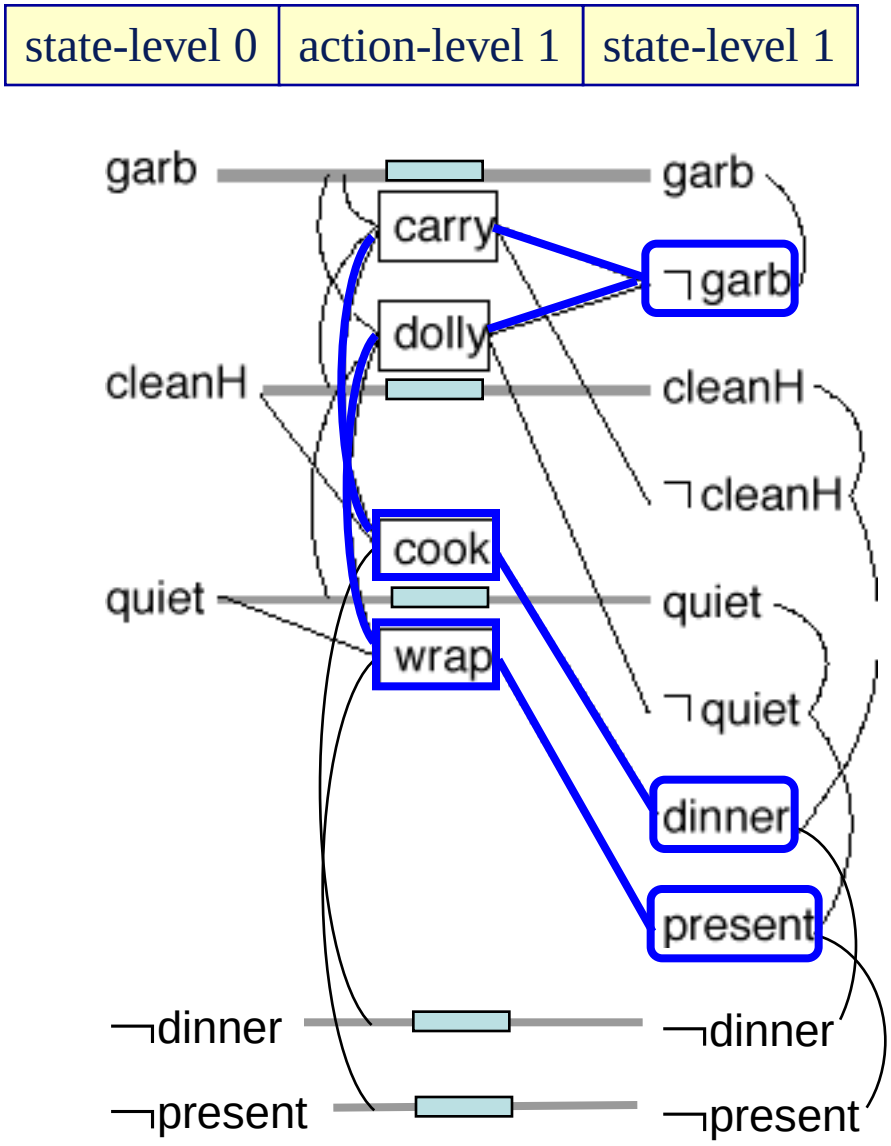
Fim

Uma ação real ou uma ação de manutenção



Exemplo (continuação)

- Existem 2 conjuntos de ações para as metas no ***state-level*** 1
- Nenhum deles funcionam:
  - ambos contém ações que são ***mutex***



O algoritmo Graphplan

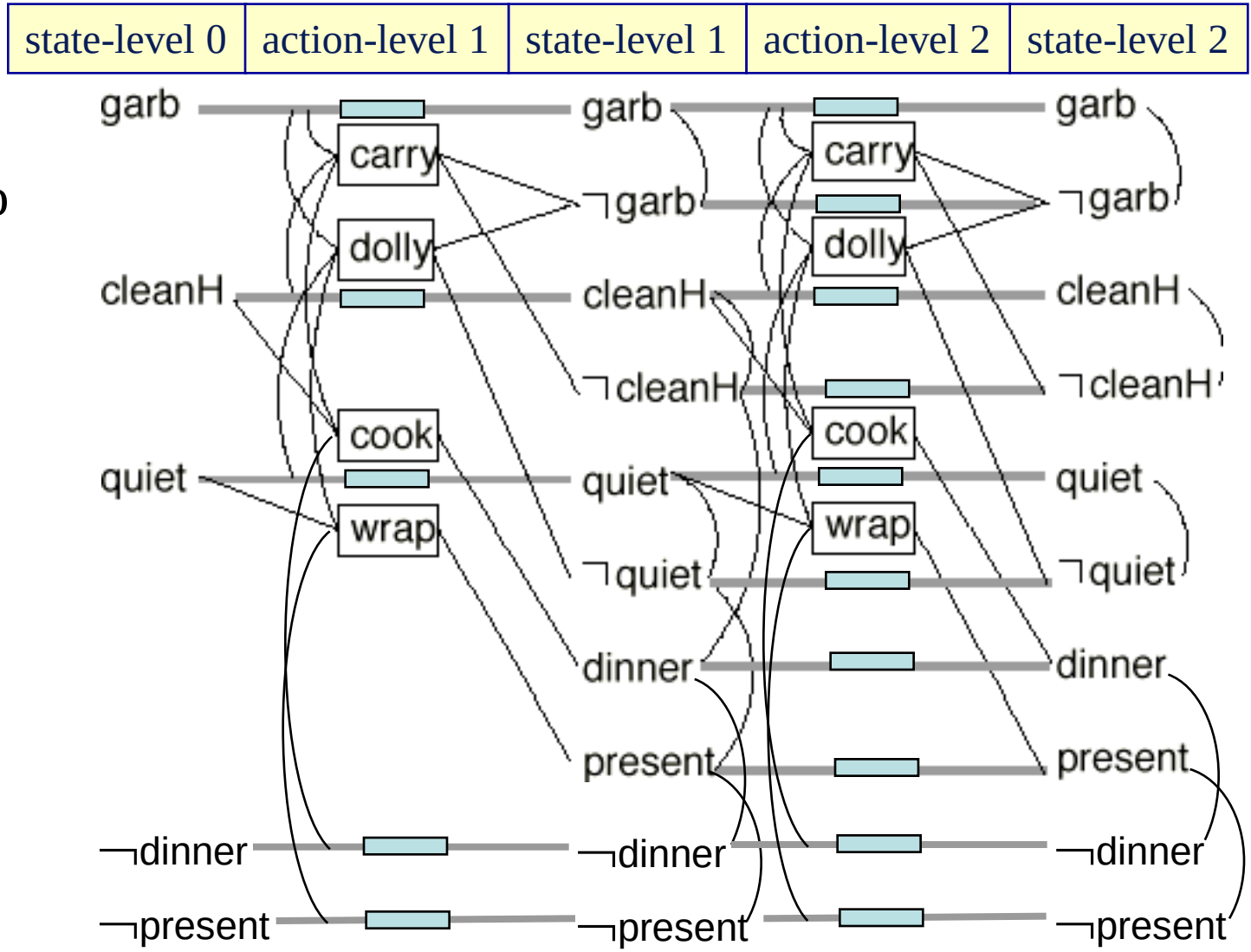
Inicializa o nível 0 com o Estado Inicial

Laço

- Se as submetas em G estão contidas no último nível do grafo e não estão em mutex (entre si)
- Então busca regressiva no grafo
  - Se uma solução for encontrada, devolva solução
- Senão estenda mais um nível do grafo (ações aplicáves + efeitos + mutex)
  - Se último nível do grafo é um ponto fixo (nos literais e mutex)
    então devolva Falha

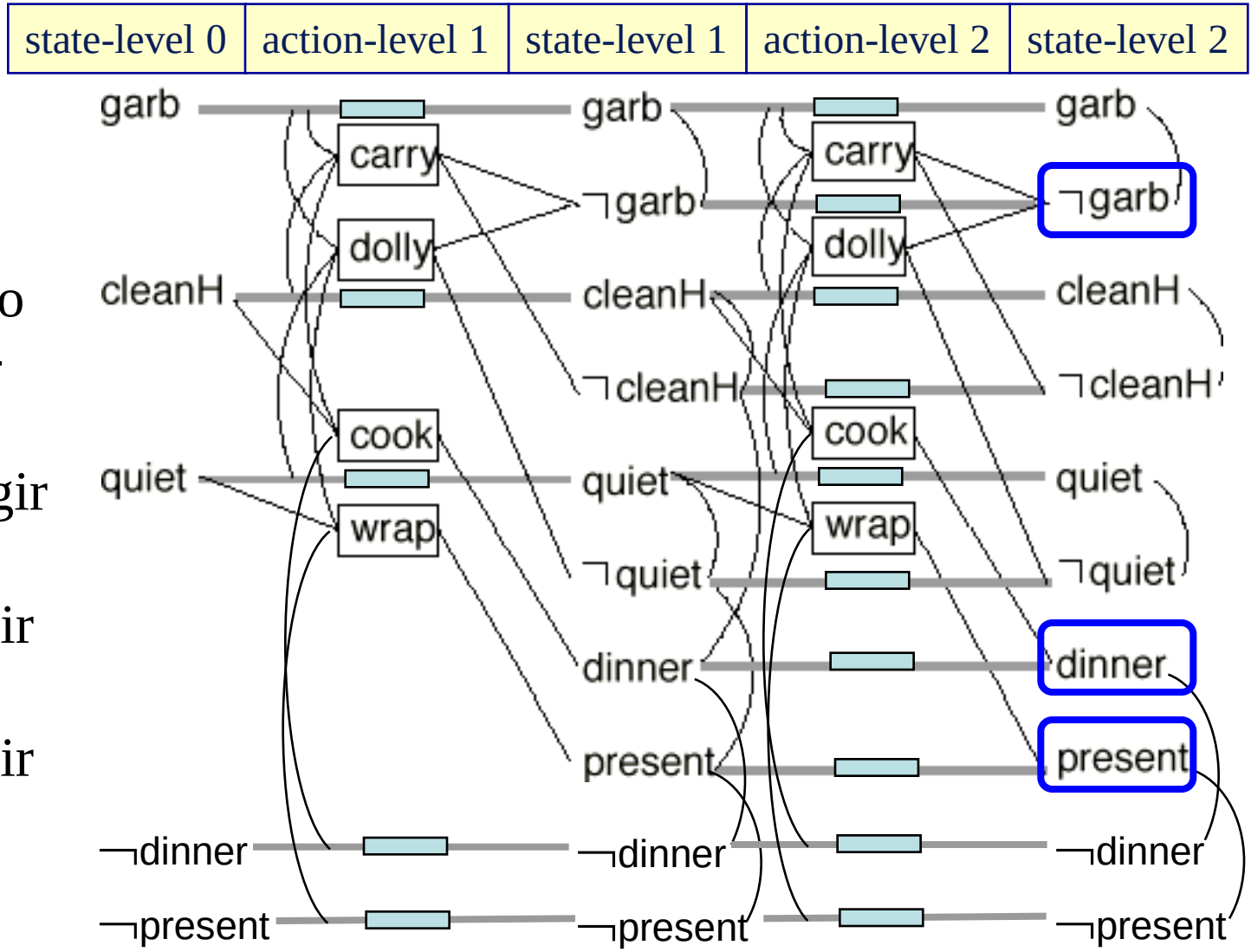
Exemplo (continuação)

- Faça mais uma expansão do grafo
- Gere outro ***action-level*** e outro ***state-level***



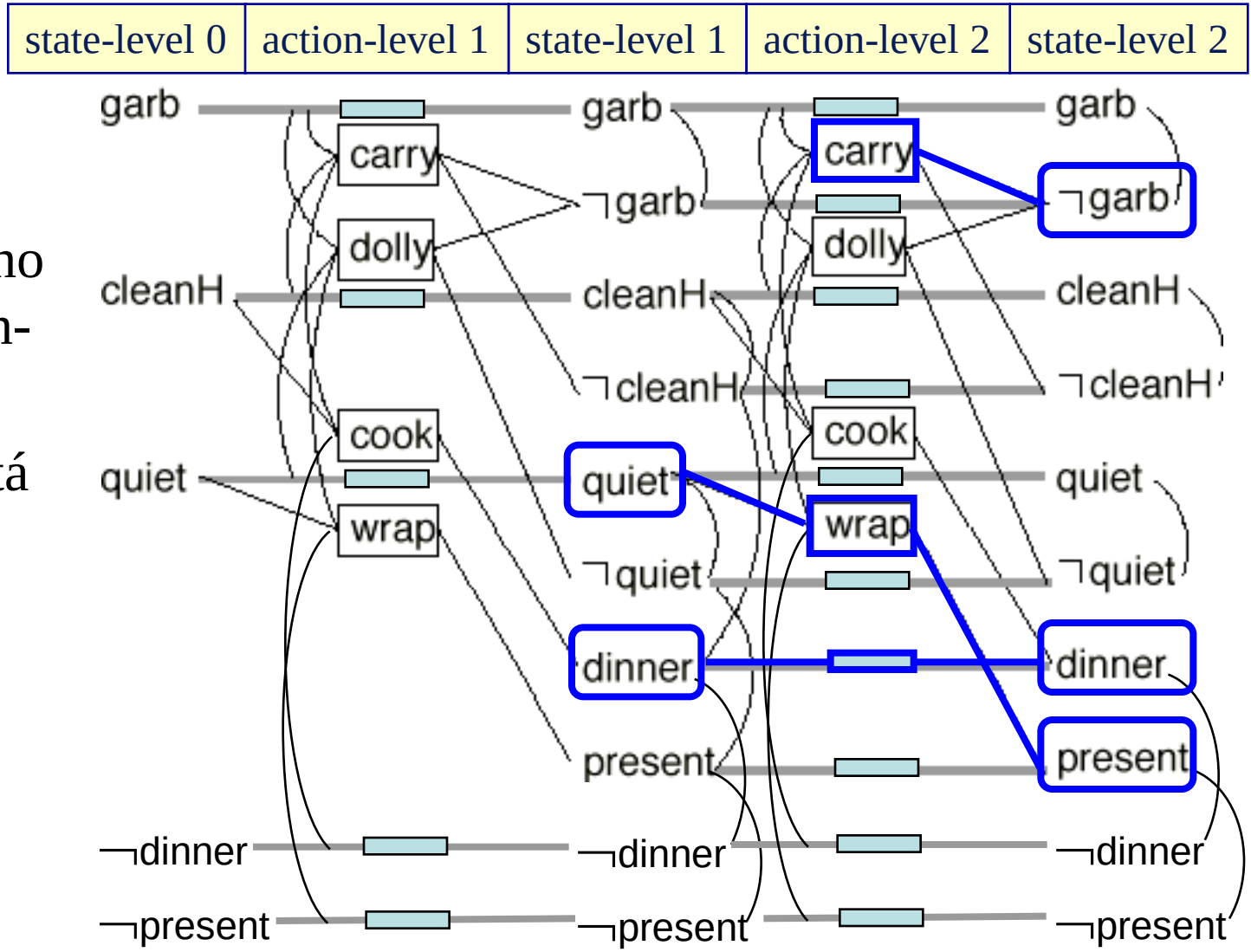
Exemplo (continuação)

- Extração da solução
- Existema 12 combinações no nível 4 (action-level 2)
  - 3 para atingir ***¬garb***
  - 2 para atingir ***dinner***
  - 2 para atingir ***present***



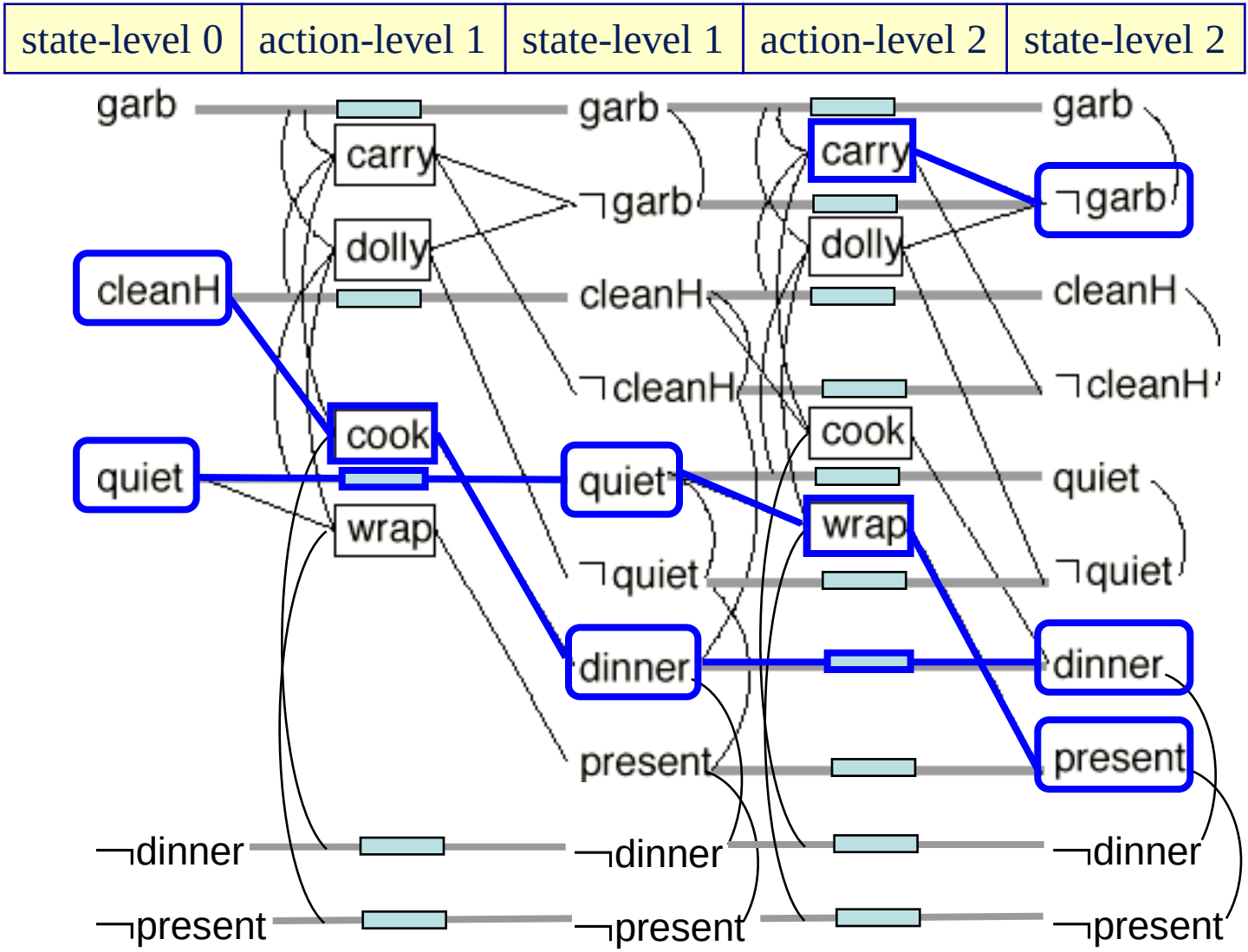
Exemplo (continuação)

- Várias das combinações parecem OK no nível 2 (action-level 1).
- Uma delas está em azul.



Exemplo (continuação)

- Chama Extração-da-Solução recursivamente no nível 2
- Sucesso!
- Solução de *tamanho paralelo* igual a 2



Condição de parada do Graphplan

- O conjunto de literais cresce monotonicamente
  - Uma vez que um literal aparece em uma dada camada ele aparecerá nas camadas subsequentes (devido às ações persistentes)

- O conjunto de ações cresce monotonicamente
  - Uma vez que um ação aparece em uma dada camada ela aparecerá nas camadas subsequentes (consequência do item anterior)

- O conjunto de mutex decrescem monotonicamente
  - Se duas ações são mutex no nível i, elas também serão mutex para todos os níveis anteriores nos quais elas aparecem. O mesmo acontece com os mutex entre literais.

Condição de parada do Graphplan

- O conjunto de literais cresce monotonicamente
  - Uma vez que um literal aparece em uma dada camada ele aparecerá nas camadas subsequentes (devido às ações persistentes)
- O conjunto de a
  - Uma vez que
    aparecerá na
    anterior)
- O conjunto de m
  - Se duas ações são mutex no nível i, elas também serão mutex para todos os níveis anteriores nos quais elas aparecem. O mesmo acontece com os mutex entre literais.

Como existe um número finito de ações e de literais, na fase expansão encontraremos um nível i com o mesmo número de ações e literais que o nível i-1 (ponto-fixo) ➔ isso garante a parada do Graphplan (devolvendo falha caso não exista solução).

Comparação com o POP

- Vantagem:
  - A parte de busca regressiva do *Graphplan* — que é a parte mais difícil — somente olha as ações no grafo de planejamento!
  - Espaço de busca menor que o POP ➔ por isso é mais rápido

- Desvantagem:
  - Diferente do POP, o planejador *Graphplan* cria instâncias *ground* de todos os operadores
  - Muitas delas podem ser irrelevantes !!

- Para planejamento clássico, as vantagens superam as desvantagens
  - GraphPlan resolve problemas de planejamento clássico muito mais rápido do que o POP.

Heurísticas

- Um grafo de planejamento é uma fonte rica de informações sobre o problema:
  - Se algum literal da meta não aparece no ultimo nível do grafo, então o problema não tem solução
  - Podemos estimar o custo de alcançar qualquer literal g_i como sendo o nível no qual g_i aparece pela primeira vez no grafo: **custo de nível de g_i**.
    - » Essa estimativa é admissível para os objetivos individuais?

Exemplo: “Have a cake and eat cake too”

Action (*Eat*(*Cake*)

Precond: *Have*(*Cake*)

Effect: $\neg$ *Have*(*Cake*) $\wedge$ *Eaten*(*Cake*))

Action (*Bake*(*Cake*)

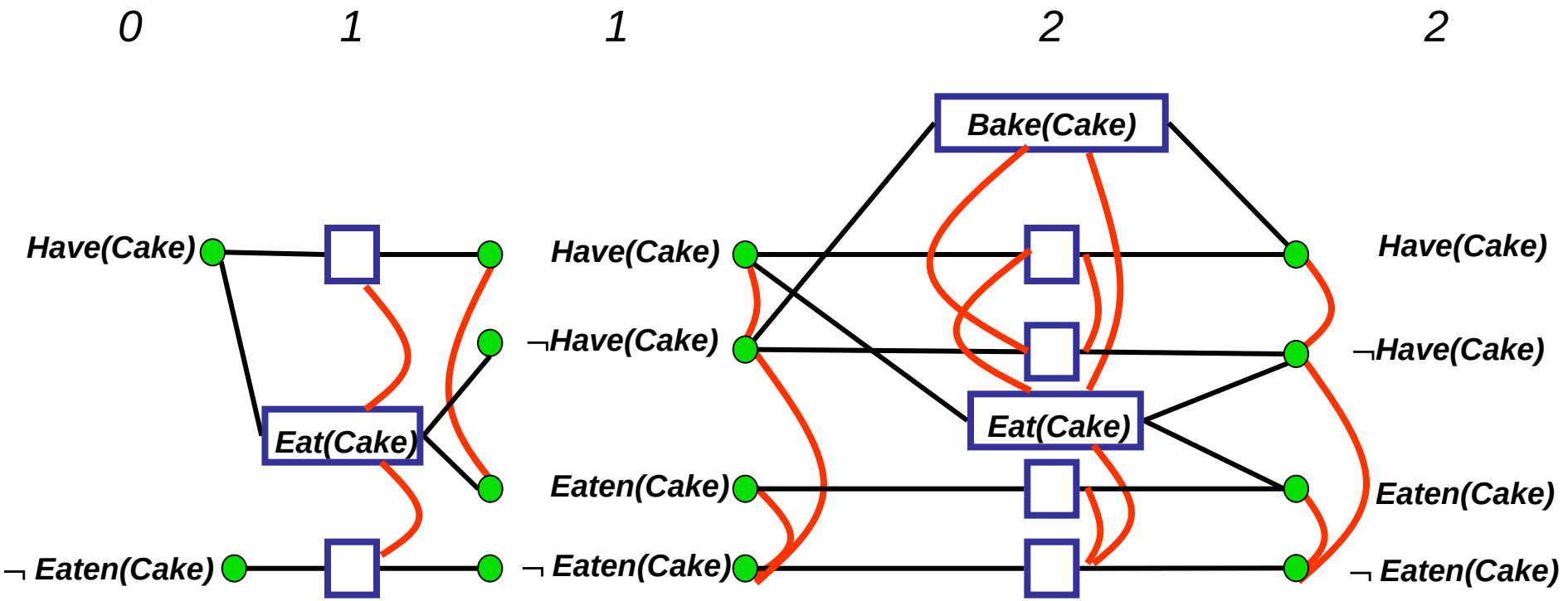
Precond: $\neg$ *Have*(*Cake*)

Effect: *Have*(Cake))

Init (*Have*(*Cake*))

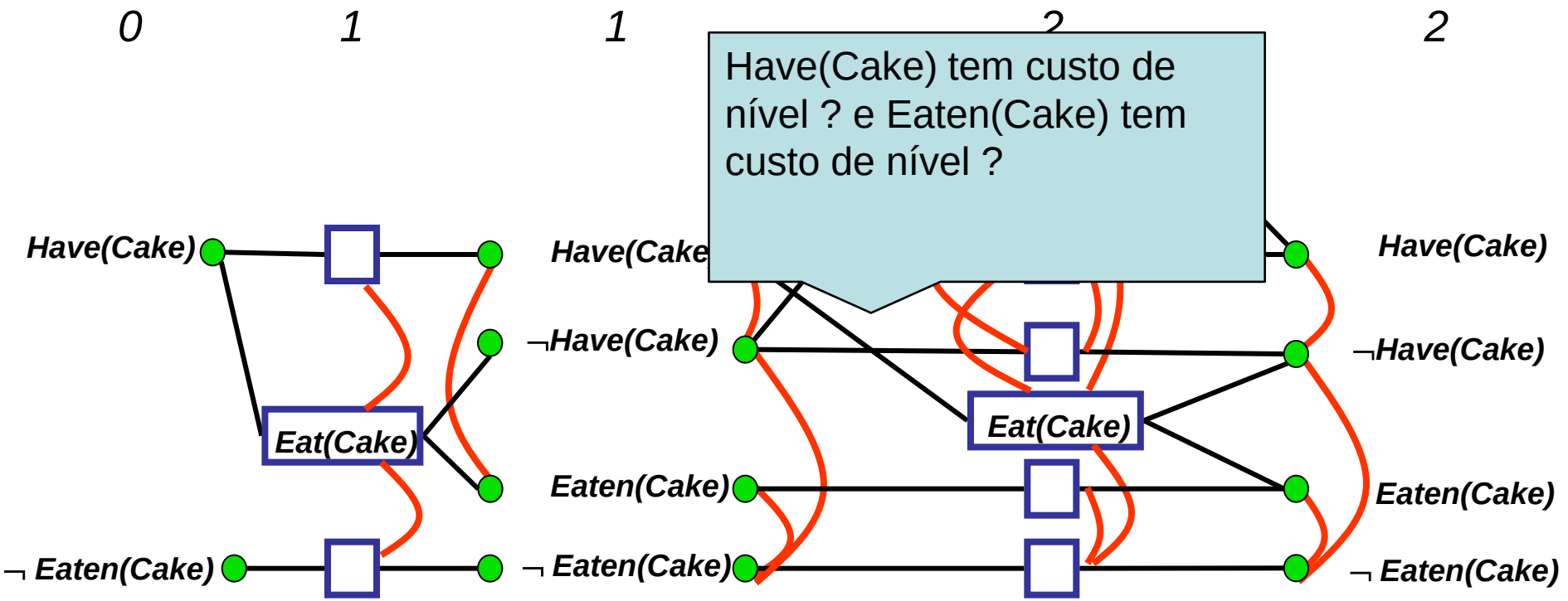
Goal (*Have*(*Cake*) $\wedge$ *Eaten*(*Cake*))

Calculando a heurística



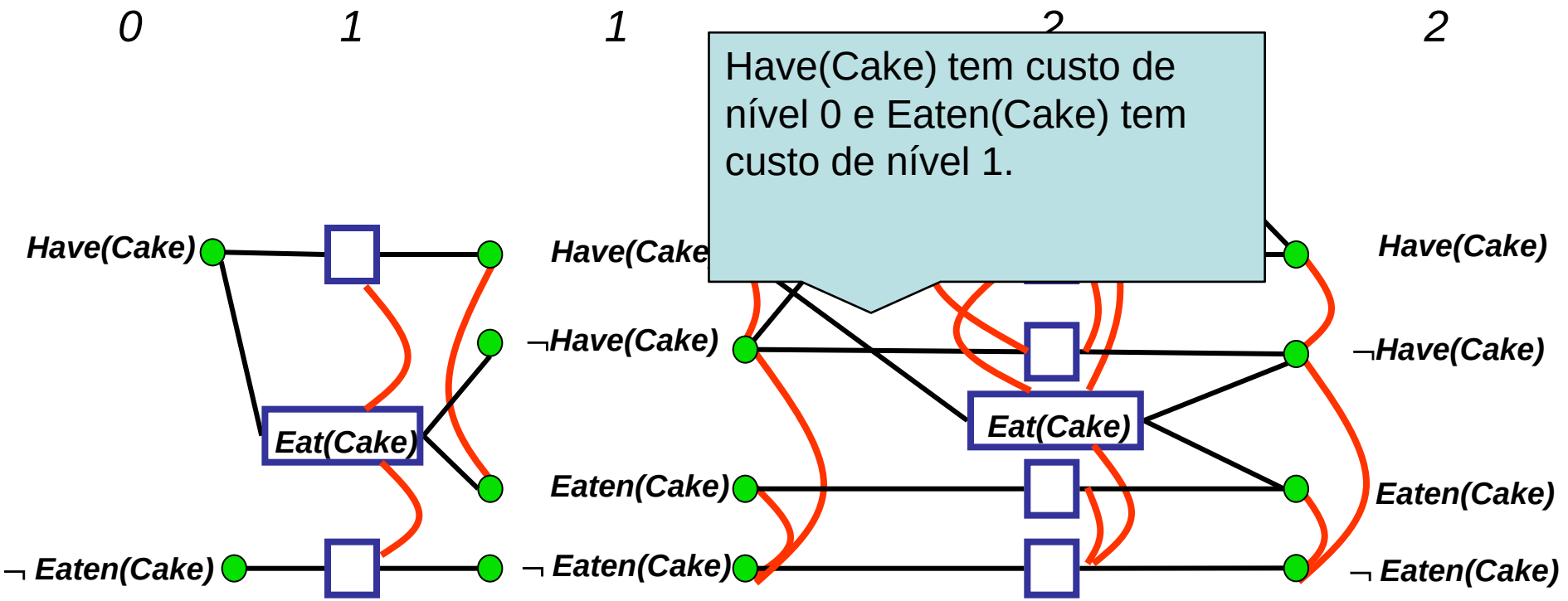
$Goal\ (Have(Cake) \wedge Eaten(Cake))$

Calculando a heurística



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

Calculando a heurística



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

Heurísticas

- Um grafo de planejamento é uma fonte rica de informações sobre o problema:
  - Se algum literal da meta não aparece no ultimo nível do grafo, então o problema não tem solução
  - Podemos estimar o custo de alcançar qualquer literal g_i como sendo o nível no qual g_i aparece pela primeira vez no grafo: **custo de nível de g_i**.
    - » Essa estimativa é admissível para os objetivos individuais?
      - Sim. Porém ela não leva em conta o número de ações.
  - Por essa razão, é comum utilizar um grafo de planejamento serial, em que apenas uma ação pode acontecer em qualquer passo de tempo, isso é feito levando em conta os mutex.
    - » Fornecem boas estimativas dos custos reais

Heurísticas

- Formas de estimar o custo de uma conjunção de sub objetivos:
  - Heurística de nível máximo
  - Heurística de soma de níveis
  - Heurística de nível de conjunto

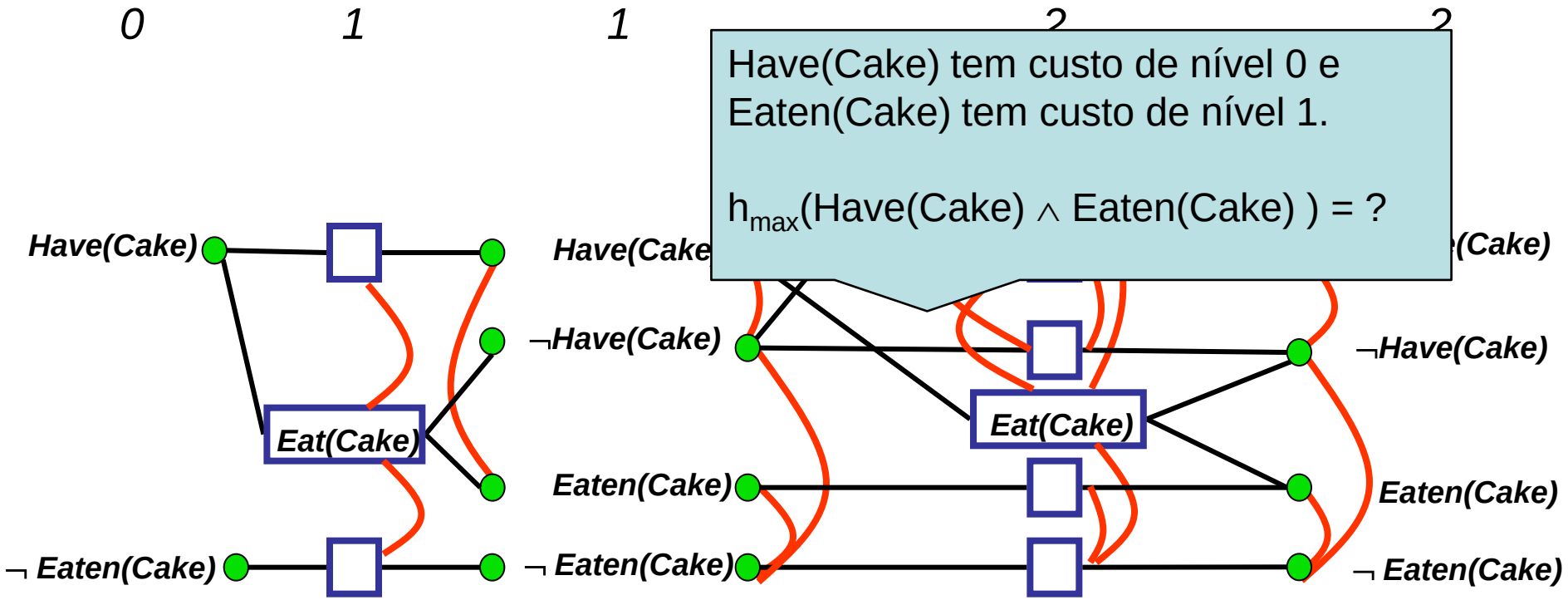
Heurísticas

- Heurística de nível máximo:
  - Considera o custo de nível máximo de qualquer dos sub objetivos
  - É admissível?

Heurísticas

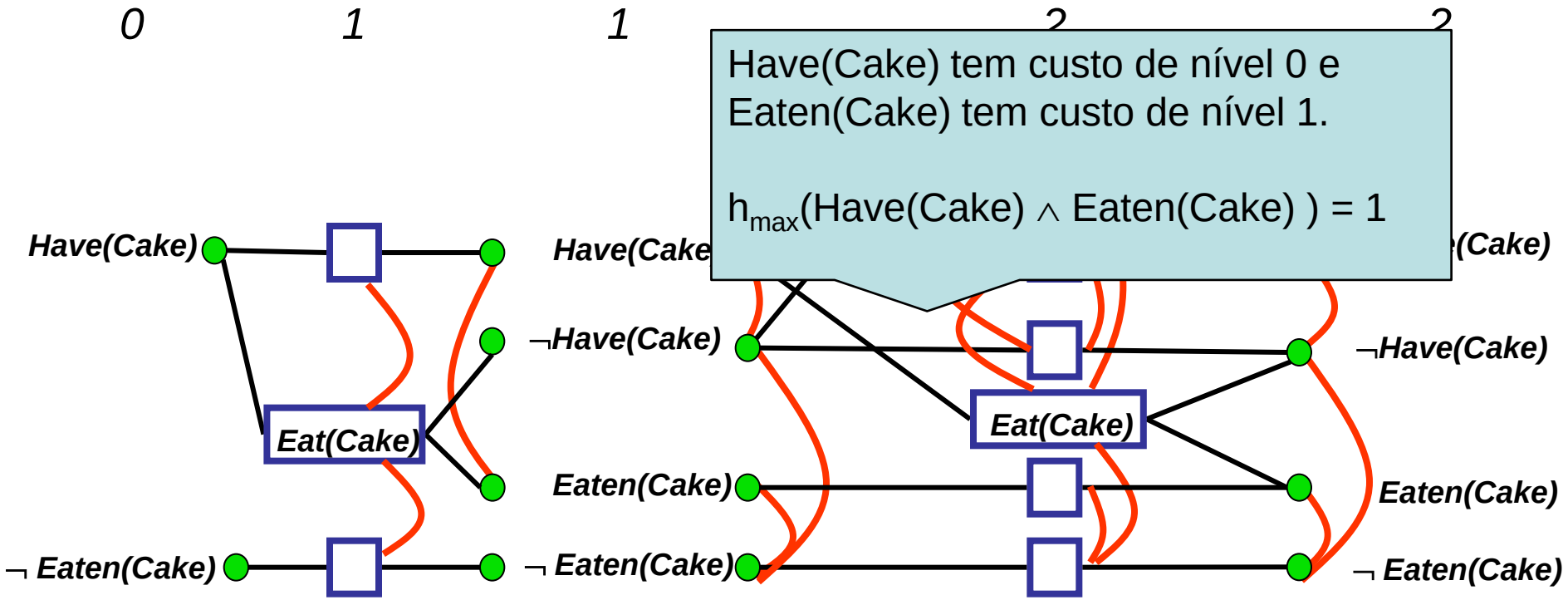
- Heurística de nível máximo:
  - Considera o custo de nível máximo de qualquer dos sub objetivos
  - É admissível ? Sim
  - Mas não necessariamente muito precisa

Calculando a heurística



Goal (*Have*(*Cake*) ∧ *Eaten*(*Cake*))

Calculando a heurística



Goal (*Have*(*Cake*) ∧ *Eaten*(*Cake*))

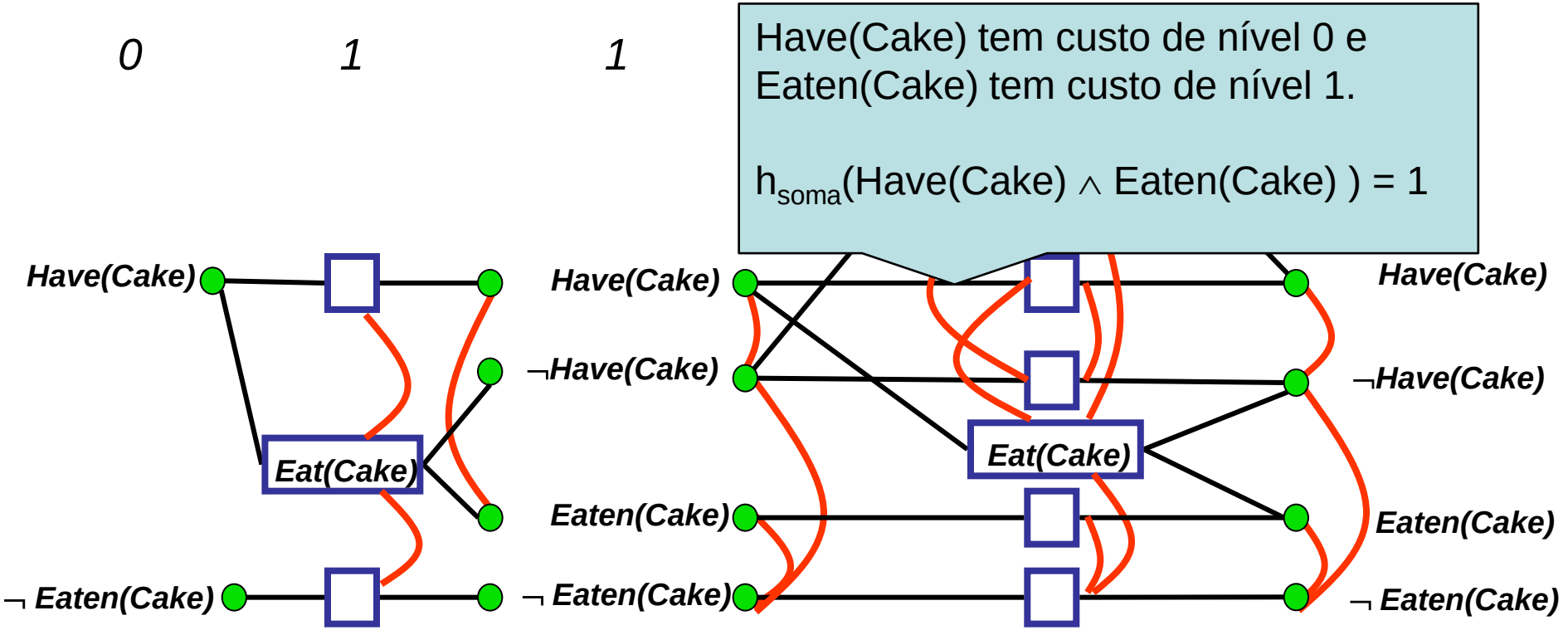
Heurísticas

- Heurística de soma de níveis:
  - Calcula a soma dos custos de nível dos sub objetivos
  - É admissível?

Heurísticas

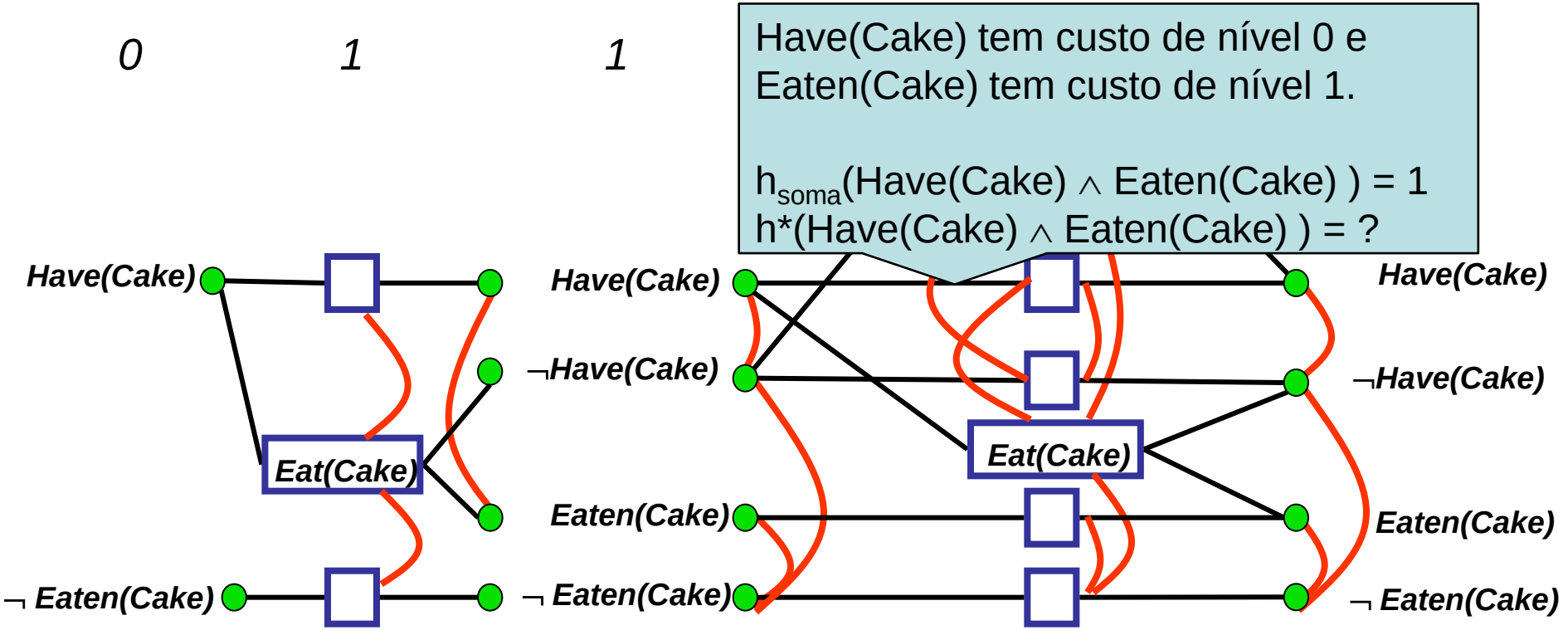
- Heurística de soma de níveis:
  - Calcula a soma dos custos de nível dos sub objetivos
  - É admissível? Não
  - Porém, funciona bem na prática

Calculando a heurística



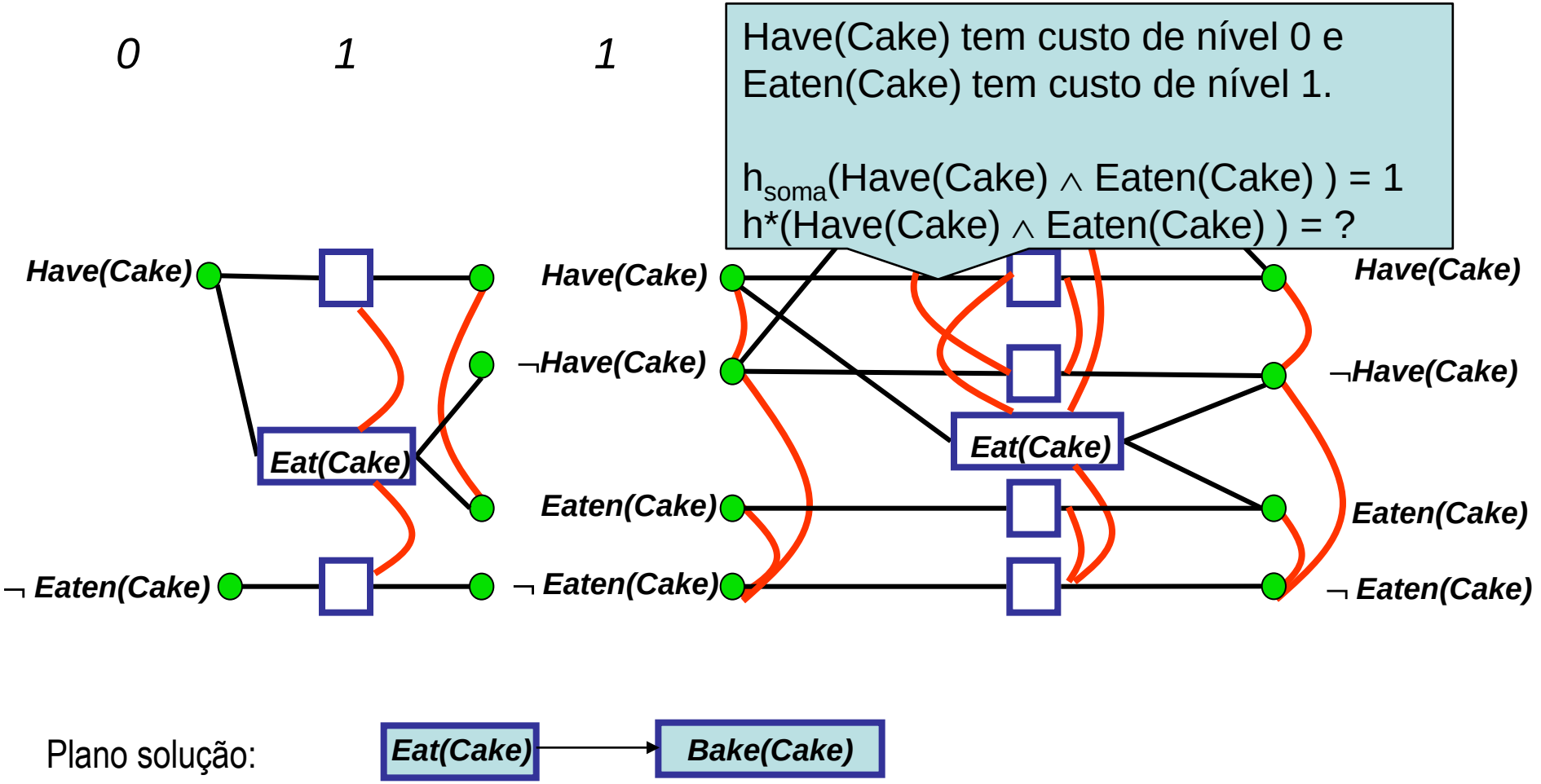
Goal (*Have*(*Cake*) ∧ *Eaten*(*Cake*))

Calculando a heurística



Goal (*Have*(*Cake*) ∧ *Eaten*(*Cake*))

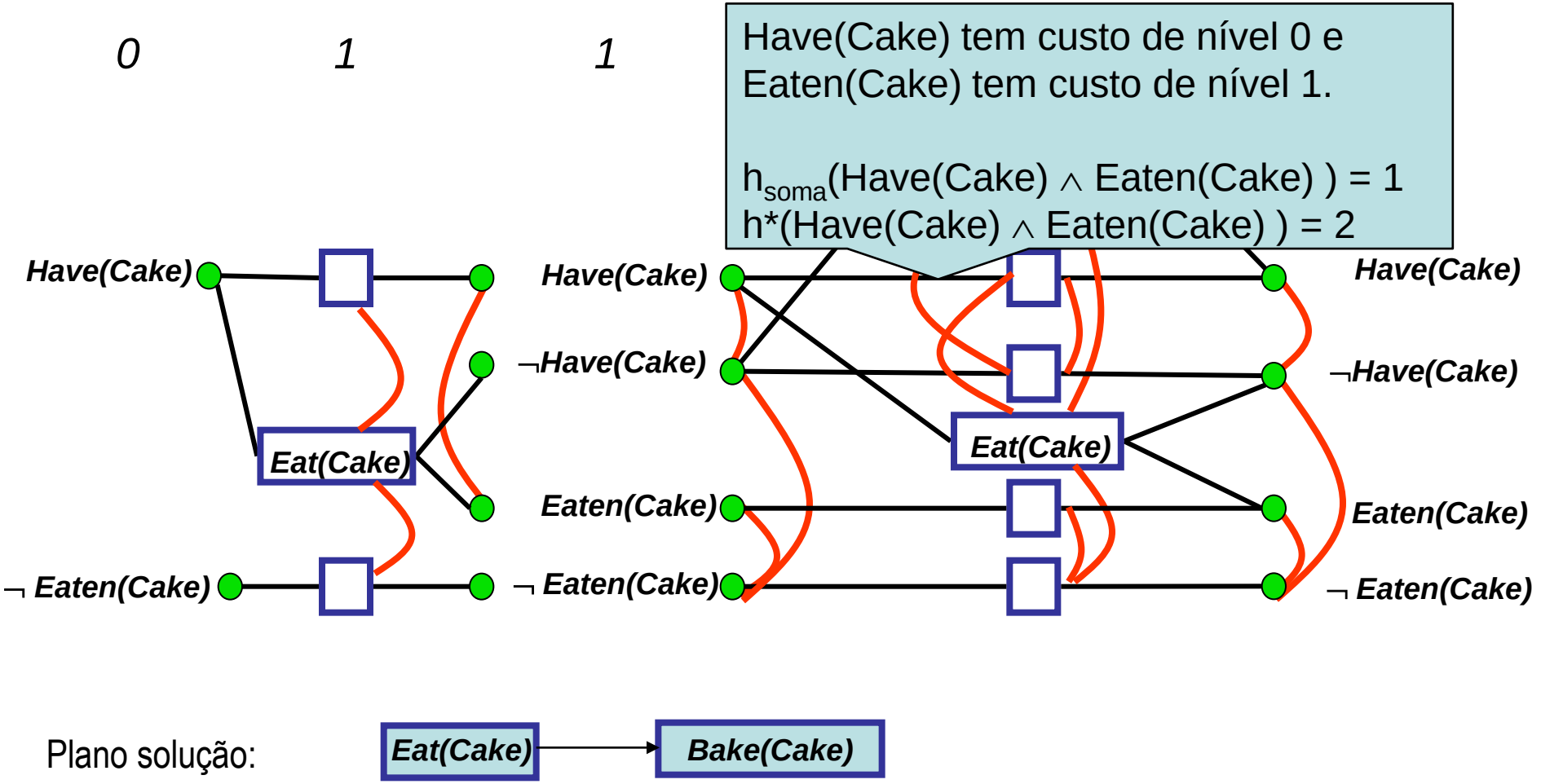
Calculando a heurística



Plano solução:

Goal (*Have*(*Cake*) ∧ *Eaten*(*Cake*))

Calculando a heurística



Plano solução:

Goal (*Have*(*Cake*) ∧ *Eaten*(*Cake*))

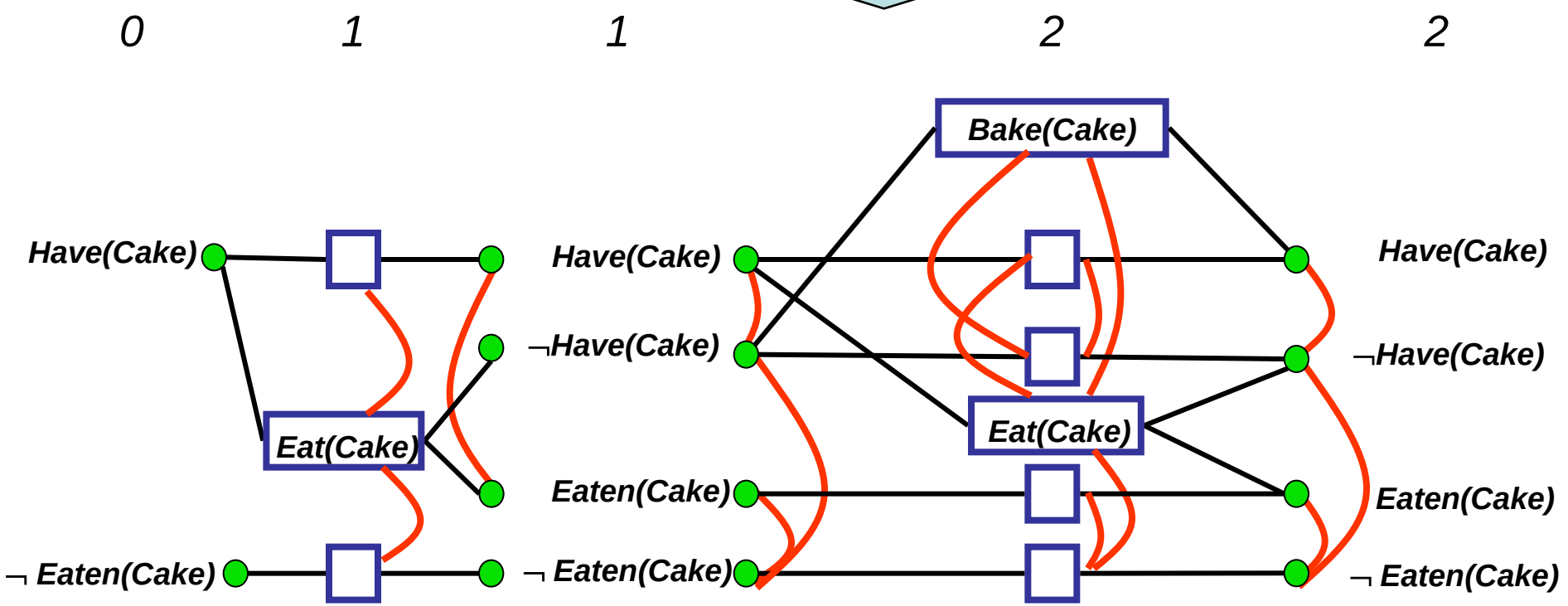
Heurísticas

- Heurística de nível de conjunto:
  - Encontra o nível no qual todos os literais no objetivo conjuntivo aparecem no grafo de planejamento sem que nenhum par seja mutuamente exclusivo.
  - É admissível?

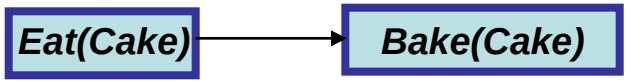
Calcula

Have(Cake) tem custo de nível 0 e
Eaten(Cake) tem custo de nível 1.

$h_{conjunto}(Have(Cake) \wedge Eaten(Cake)) = 2$
$h^*(Have(Cake) \wedge Eaten(Cake)) = 2$



Plano solução:



Goal (*Have*(*Cake*) ∧ *Eaten*(*Cake*))

Heurísticas

- Heurística de nível de conjunto:
  - Encontra o nível no qual todos os literais no objetivo conjuntivo aparecem no grafo de planejamento sem que nenhum par seja mutuamente exclusivo.
  - É admissível? Sim
  - Ela “domina” a heurística do nível máximo
  - Porém, não é perfeita pois ...

Heurísticas

- Heurística de nível de conjunto:
  - Encontra o nível no qual todos os literais no objetivo conjuntivo aparecem no grafo de planejamento sem que nenhum par seja mutuamente exclusivo.
  - É admissível? Sim
  - Ela “domina” a heurística do nível máximo
  - Porém, não é perfeita pois ignora as interações entre três ou mais literais.

FastForward (Hoffmann, 2005)

- Planejador de busca no espaço de estados progressivo
- Utiliza Hill Climbing com uma heurística
  - A heurística é gerada usando GraphPlan para o problema relaxado, em que são ignorados os efeitos negativos
  - Significa que o GraphPlan é executado para cada estado avaliado na busca!
- Quando nenhuma ação leva a um estado com melhor valor heurístico, o FF usa busca em profundidade iterativa até:
  - Encontrar um estado que seja melhor ou
  - desiste e reinicia o Hill Climbing

Heurística do FastForward (Hoffmann, 2005)

Como não existem efeitos negativos no problema relaxado => não existem mutex, o que faze com que:

- A expansão do grafo de planejamento seja rápida (complexidade polinomial).
- A extração da solução seja também rápida (complexidade polinomial).

Heurística do FastForward (Hoffmann, 2005)

Primeiro criamos o problema relaxado ignorando os efeitos negativos

Para calcular a heurística de um estado *s*, executamos o algoritmo GraphPlan com o problema relaxado e o estado *s* como estado inicial:

- Fase 1: expansão do grafo de planejamento: estende o grafo progressivamente até que as submetas em G estão contidas no último nível.

- Fase 2: extração da solução: faz uma busca regressiva no grafo, a procura de um plano solução:
  - » Algumas heurísticas são usadas para fazer que a busca regressiva seja mais rápida. Uma delas é a heurística no-op first que sempre escolhe primeiro no-op para atingir um literal.

Heurística do FastForward (Hoffmann, 2005)

- Blocks world

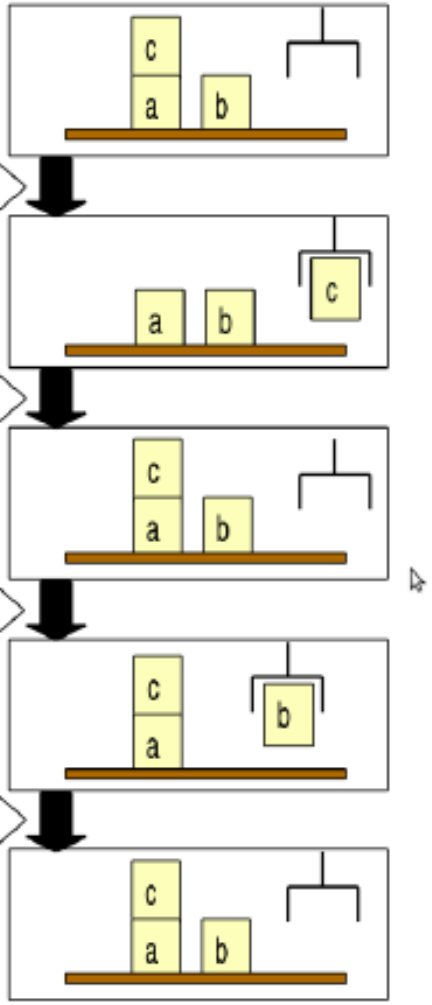
Operadores:

unstack(*x*,*y*)
Precond: on(*x*,*y*), clear(*x*), handempty
Effects: ~on(*x*,*y*), ~clear(*x*), ~handempty, holding(*x*), clear(*y*)

stack(*x*,*y*)
Precond: holding(*x*), clear(*y*)
Effects: ~holding(*x*), ~clear(*y*), on(*x*,*y*), clear(*x*), handempty

pickup(*x*)
Precond: ontable(*x*), clear(*x*), handempty
Effects: ~ontable(*x*), ~clear(*x*), ~handempty, holding(*x*)

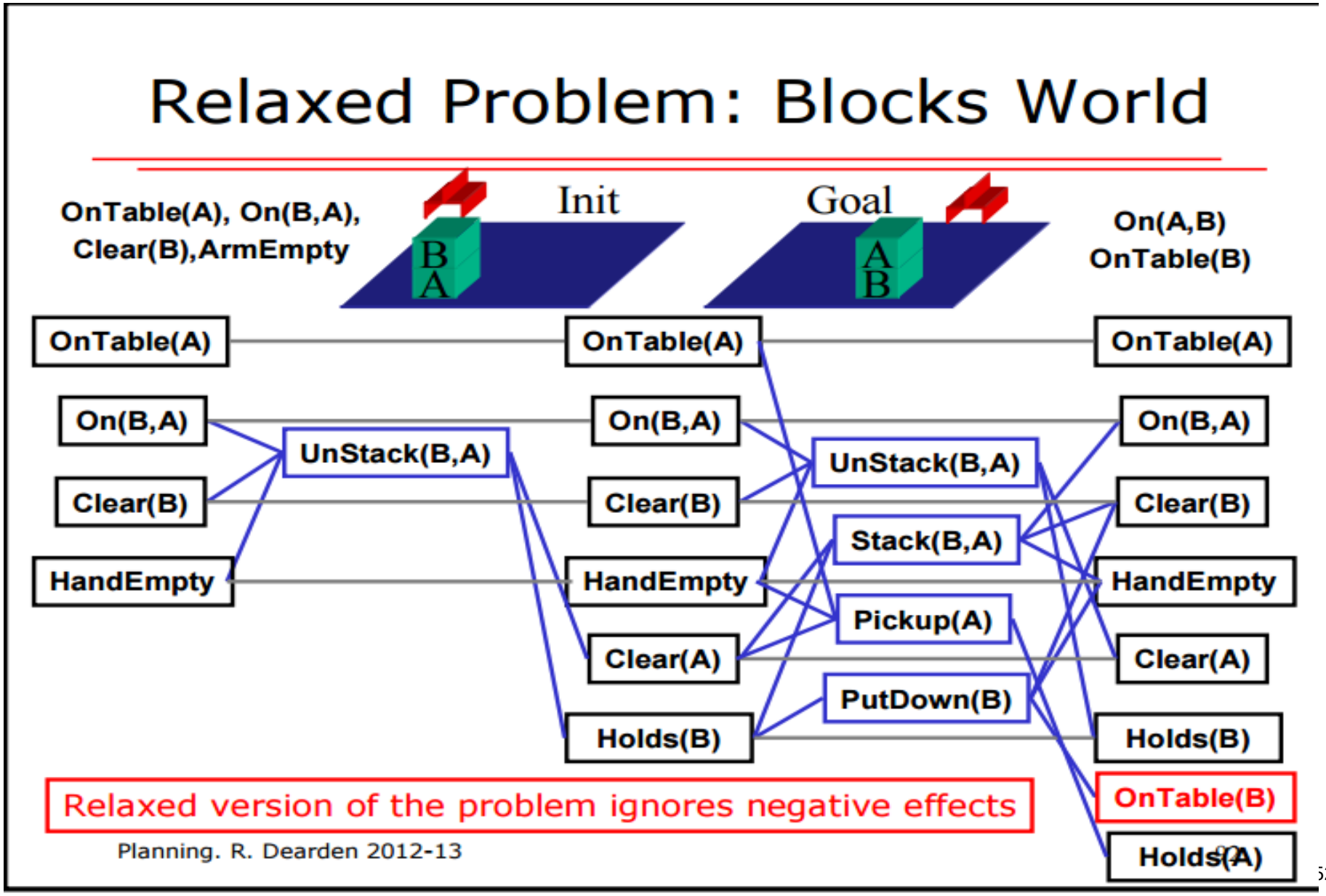
putdown(*x*)
Precond: holding(*x*)
Effects: ~holding(*x*), ontable(*x*), clear(*x*), handempty



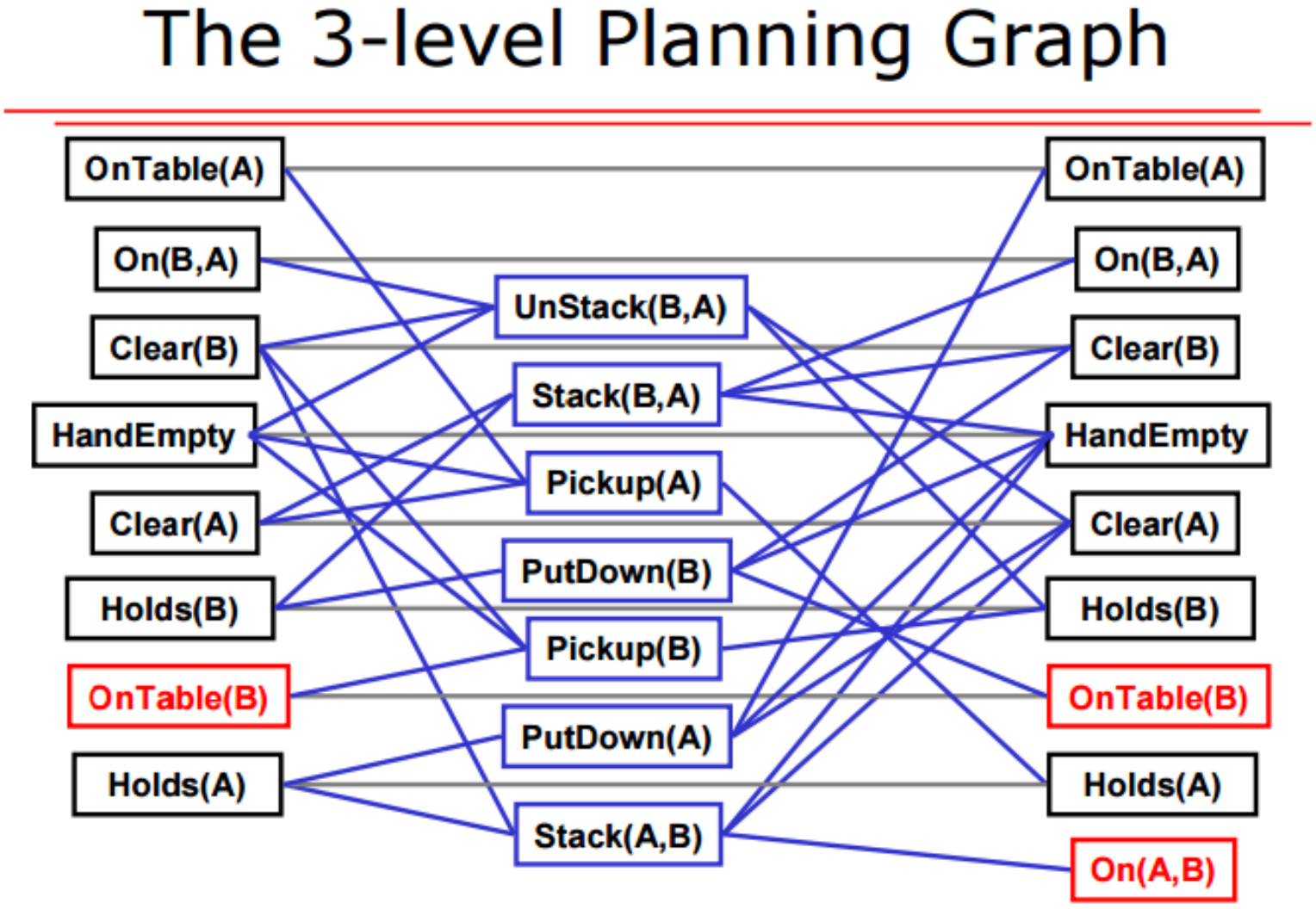
Heurística do FastForward (Hoffmann, 2005)

- Sabendo que o estado meta é:
  - » on(A,B), ontable(B)
- Calcular a heurística para o estado
  - » ontable(A), on (B,A), clear(B), handempty

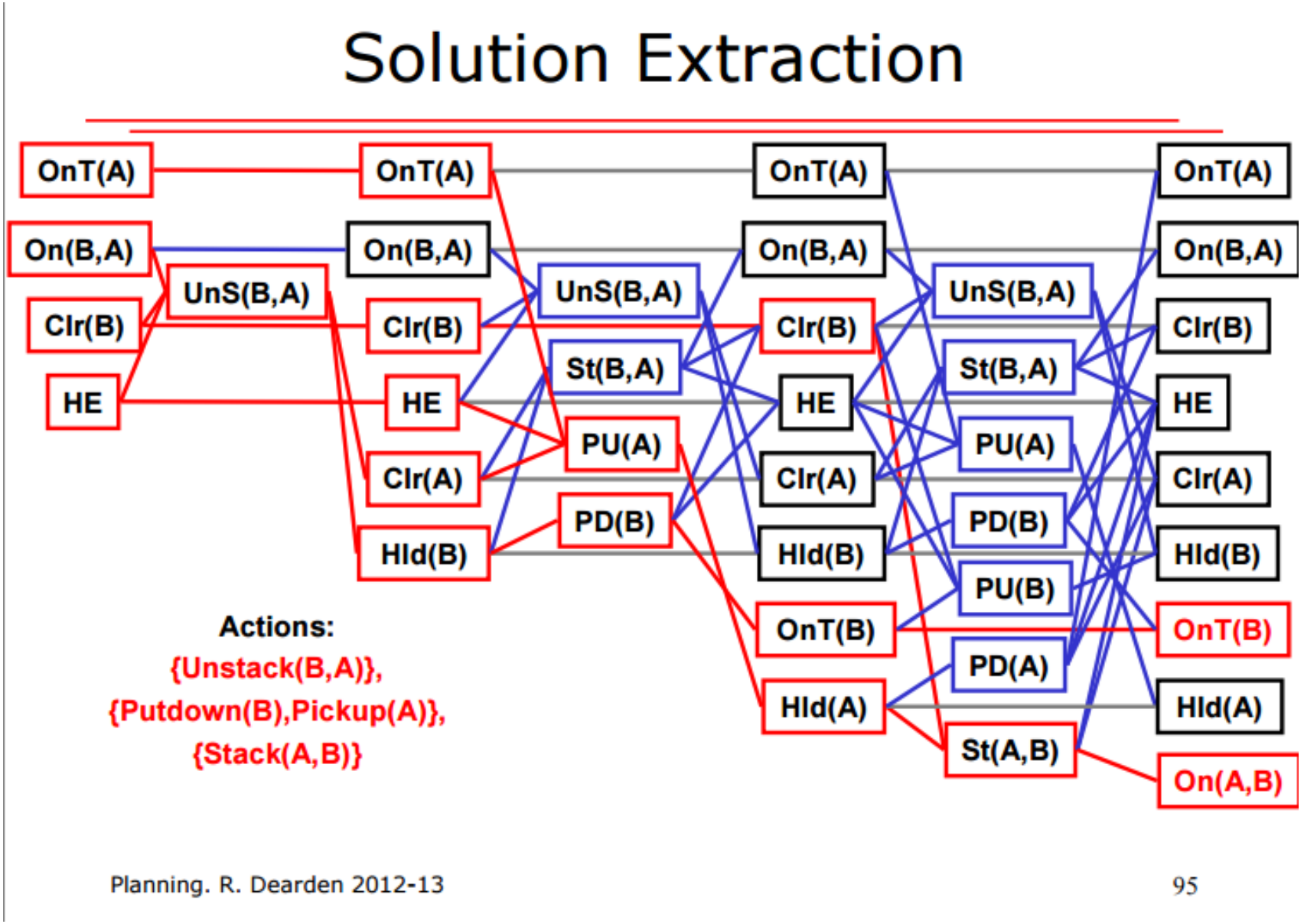
Heurística do FastForward (Hoffmann, 2005)



Heurística do FastForward (Hoffmann, 2005)



Heurística do FastForward (Hoffmann, 2005)



Heurística do FastForward (Hoffmann, 2005)

- Sabendo que o estado meta é:
  - » on(A,B), ontable(B)
- Calcular a heurística para o estado s:
  - » ontable(A), on (B,A), clear(B), handempty
- O plano encontrado pelo graphPlan para o problema relaxado é:
  - » unstack(B,A), putdown(B), pickup(A),stack(A,B)
- Logo h(s)=?

Heurística do FastForward (Hoffmann, 2005)

- Sabendo que o estado meta é:
  - » on(A,B), ontable(B)
- Calcular a heurística para o estado s:
  - » ontable(A), on (B,A), clear(B), handempty
- O plano encontrado pelo graphPlan para o problema relaxado é:
  - » unstack(B,A), putdown(B), pickup(A),stack(A,B)
- Logo h(s)=comprimento do plano = 4