
Inteligência Artificial
Planejamento
1

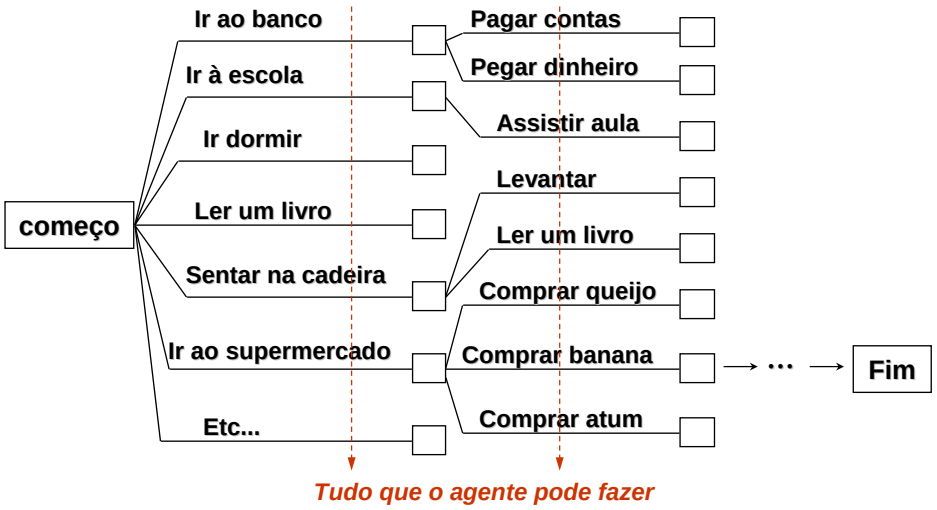

Planejamento:Conceitos básicos
Planejador: objetiva encontrar um plano que permita um agente executar uma tarefa, a partir de uma situação inicial.
Plano: seqüência ordenada de ações
tarefa: obter banana, leite e uma furadeira
plano: ir ao supermercado, ir à seção de frutas, pegar as bananas, ir à seção de leite, pegar uma caixa de leite, ir ao caixa, pagar tudo, ir a uma loja de ferramentas, ..., voltar para casa.
2

Busca x planejamento

- **Representação usando busca**
  - **Ações**: programas que geram o estado sucessor
  - **Estados**: descrição completa
    - *problemático em ambientes inacessíveis*
  - **Objetivos**: função de teste e heurística
  - **Planos**: totalmente ordenados e criados incrementalmente a partir do estado inicial
    - *Ex. posições das peças de um jogo*
- **Exemplo do supermercado**
  - **estado inicial:** em casa sem objetos desejados
  - **estado final**: em casa com objetos desejados
  - **operadores**: tudo o que o agente pode fazer
  - **heurística**: número de objetos ainda não possuídos

3

Exemplo em resolução de problemas usando busca



Limitações desta abordagem

- **Fator de ramificação grande;**
- **A função heurística apenas escolhe o estado mais próximo do objetivo. Não permite descartar ações a priori;**
- **Não permite abstração dos estados parciais;**
- **Considera ações a partir do estado inicial, uma após a outra;**
- **Objetivo é testado para cada estado; para cada novo estado, um novo teste idêntico precisa ser feito.**

5

Planejamento: 3 idéias principais

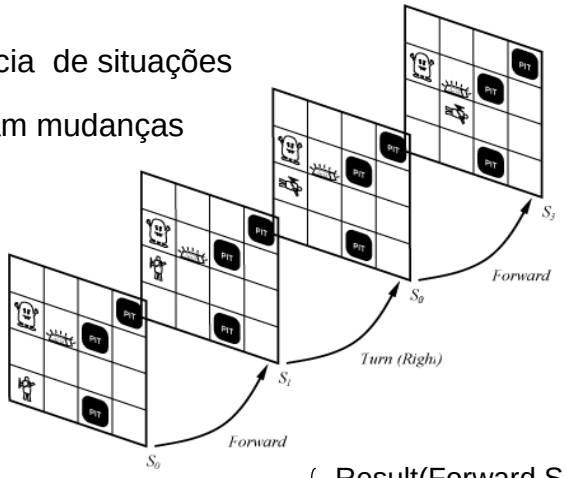
- **Representação dos estados, objetivos e ações usando LPO (descrições parciais dos estados)**
  - pode conectar diretamente estados e ações. Ex. estado: Have (Milk), ação: Buy(milk)➔ Have(Milk)
- **Adiciona ações ao plano quando forem necessárias**
  - **ordem de planejamento ≠ ordem de execução**
  - primeiro, o que é importante : Buy(Milk) – pode-se colocar esta ação no plano, mesmo sem saber como chegar ao supermercado.
  - diminui fator de ramificação
- **Uso da estratégia de dividir-e-conquistar**
  - Definição de sub-planos: sub-plano supermercado, sub-plano loja de ferramentas (sub-metas)

6

Relembrando o Cálculo Situacional

Mundo: seqüência de situações

Ações: provocam mudanças na situação



Representação das mudanças no mundo:

$$\begin{cases} \text{Result}(\text{Forward}, S_0) = S_1 \\ \text{Result}(\text{Turn}(\text{Right}), S_1) = S_2 \\ \text{Result}(\text{Forward}, S_2) = S_3 \end{cases}$$

7

Planejando com Cálculo de Situações

- **Estado inicial: sentença lógica**
  At(Home, S0) $\wedge \neg$ Have(Milk , S0) $\wedge \neg$ Have(Bananas, S0) $\wedge \neg$ Have(Drill , S0)
- **Estado Objetivo: pergunta lógica** (p/ unificação)
  At(Home, S) $\wedge$ Have(Milk , S) $\wedge$ Have(Bananas , S) $\wedge$ Have(Drill , S)
- **Operadores: conjunto de axiomas de estado sucessor**
  $\forall$ a,s Have(Milk, Result(a, s)) $\Leftrightarrow$ [(a = Buy(Milk) $\wedge$ At(supermarket, s) $\vee$ (Have(Milk, s) $\wedge$ a $\neq$ Drop(Milk))]
- **Notação**
  - Result(a,s) - uma ação executada na situação s;
  - Result'(p,s) - seqüência de ações ➔ S = Result'(p,S0)

8

Planejando com Cálculo de Situações

- **Reescrevendo o Estado Objetivo: pergunta lógica**
  At(Home,Result'(p, S0)) ∧ Have(Milk, Result'(p, S0)) ∧ Have(Bananas, Result'(p, S0)) ∧ Have(Drill, Result'(p, S0))
- **Solução:**
  p = [Go(SuperMarket), Buy(Milk), Buy(Bananas), Go(HardwareStore), Buy(Drill), Go(home)]
- **Limitações**
  - Eficiência da inferência em lógica de primeira ordem: não OK!
  - Nenhuma garantia sobre a qualidade da solução
    – *ex. pode haver passos redundantes no meio do plano*

➔ **Solução: especializar linguagem (STRIPS) e definir um algoritmo para planejar (POP)**

9

STRIPS

- **STRIPS:** ***STanford Research Institute Problem Solver***
- **Estados:** conjunção de literais sem variáveis
  - ***Inicial***: At(Home)
  - *Por* default, *literal não mencionado é falso (Hipótese do mundo fechado):* ¬ Have(Milk) ^ ¬ Have(Bananas) ^ ¬ Have(Drill)
  - ***Final***: At(Home) ^ Have(Milk) ^ Have(Bananas) ^ Have(Drill)
- **Objetivos:** conjunção de literais e possivelmente variáveis (∃)
  – *At(Home) ^ Have(Milk) ^ Have(Bananas) ^ Have(Drill)*
  – *At(x)^ Sells(x, Milk)*

10

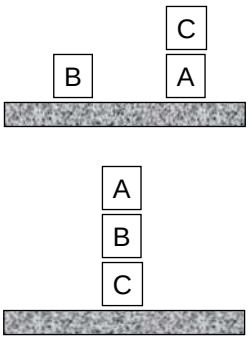
Ações em STRIPS

- **Ações:**
  - **Descritor da ação**: predicado lógico
  - **Pré-condição:** conjunção de literais positivos
  - **Efeito**: conjunção de literais (positivos ou negativos) – lista de literais a serem *adicionados* e lista de literais a serem *removidos*.
- **Operador para ir de um lugar para outro**
  - *Op(ACTION: Go(there),*
    PRECOND:At(here) ^ Path(here, there),
    EFFECT:At(there) ^ ¬ At(here))
  - *ADD: At(there) DEL: ¬ At(here)*

Notação alternativa:

At(here), Path(here, there)
Go(there)
At(there), ¬ At(here)

11



Estado inicial:
On(C,A) Clear(B)
On(A, Table) Clear(C)
On(B, Table)

Estado final (objetivo):
On(A,B) Clear(A)
On(B, C)
On(C, Table)

Ações:

PutOn(x,y) ➔ **P:** On(x,z), Clear(y), Clear(x)
Add: On(x,y), Clear(z)
Del: On(x,z), Clear(y)

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

PutOnTable(x) ➔ **P:** On(x,z), Clear(x)
Add: On(x,Table), Clear(z)
Del: On(x,z)

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

12

Tipos de Planejadores

- **Controle**
  - Progressivo: estado inicial ➔ objetivo
  - Regressivo: objetivo ➔ estado inicial
    - *mais eficiente (há menos caminhos partindo do objetivo do que do estado inicial)*
    - *problemático se existem múltiplos objetivos*
- **Espaços de busca**
  - Espaço de situações (nó = estado do mundo)
  - Espaço de planos (nó = plano parcial)
    - *mais flexível*
    - *evita engajamento prematuro*

13

Busca no espaço de planos

- **Idéia**
  - Buscar um plano desejado em vez de uma situação desejada (espécie de meta-busca)
  - parte-se de um plano inicial (parcial), e aplica-se operadores até chegar a um plano final (completo)
- **Plano inicial**
  - passos Start e Finish

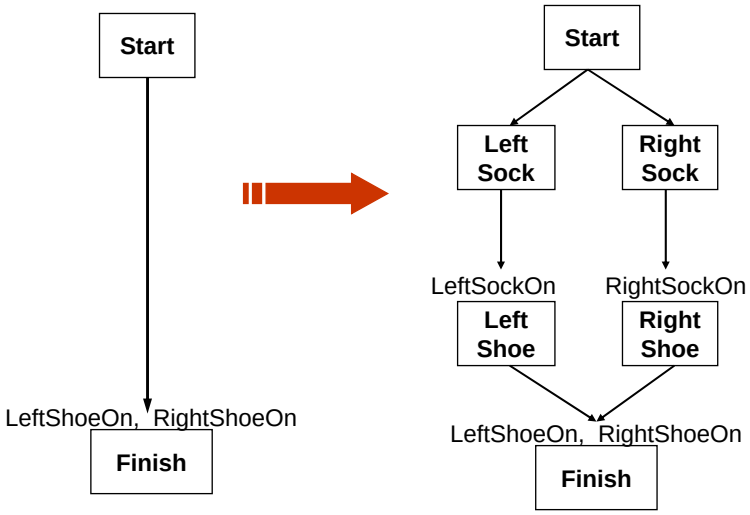
14

Busca no espaço de planos: operadores

- **Condição aberta:** é uma pré-condição de um passo ainda não realizado.
- **Operadores para planos parciais:**
  - Adicionar ligação de uma ação existente para uma condição aberta;
  - Adicionar um passo para preencher uma condição aberta;
  - Ordenar um passo com respeito a outro.
- Gradualmente, evolui de planos incompletos (vagos) para planos completos e corretos

15

Plano (de ordem) parcial



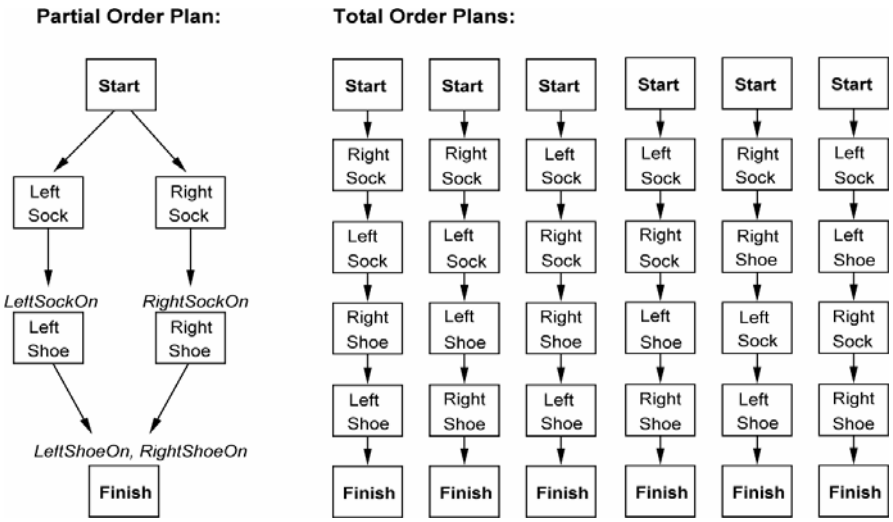
16

Plano final: características

- **Plano final**
  - **Completo** - toda pré-condição é preenchida. Uma pré-condição é preenchida ***iff*** ela for o efeito de um passo prévio e nenhum passo posterior o desfaz.
  - **Consistente** - não há contradições nos ordenamentos ou nas atribuição de variáveis
  - ***mas*** não necessariamente totalmente ordenado e instanciado….
- **Ordem total x Ordem parcial**
  - Ordem total: lista simples com todos os passos, um atrás do outro
  - **Linearizar** um plano é colocá-lo na forma “ordem total”
- **Instanciação completa** de um plano: quando todas variáveis são instanciadas

17

Linearização do exemplo dos sapatos



18

Princípio do menor engajamento

- **Por quê deixar o plano não totalmente ordenado e instanciado?**
- **Princípio do menor engajamento** (least commitment planning)
  - não faça hoje o que você pode fazer amanhã
  - ordem e instanciação parcial são decididas quando necessário
  - evita-se backtracking!
- **Exemplo**
  - para objetivo have(Milk), a ação Buy(item, store) instancia somente o item => Buy (Milk,store)
  - para as meias/sapatos: calçar cada meia antes do sapato, sem dizer por onde começa(esq/dir)

19

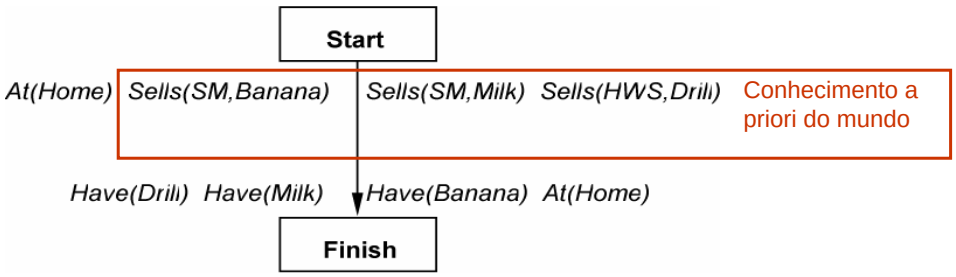
POP (Partial Order Planning)

- **Existindo a linguagem (STRIPS), falta o algoritmo..**
- **Características do POP**
  - Algoritmo não determinístico;
  - A inserção de um passo só é considerada se atender uma pré-condição não preenchida (aberta);
  - Planejador regressivo (do objetivo para o início);
  - É correto e completo, assumindo busca em largura ou em profundidade iterativa.
- **Idéia do algoritmo**
  - identifica passo com pré-condição não satisfeita;
  - introduz passo cujo efeito (causa) é satisfazer esta pré-condição;
  - instancia variáveis e atualiza os links causais;
  - verifica se há ameaças e corrige o plano, se for o caso.

20

Voltando ao exemplo das compras...

- **Plano inicial**

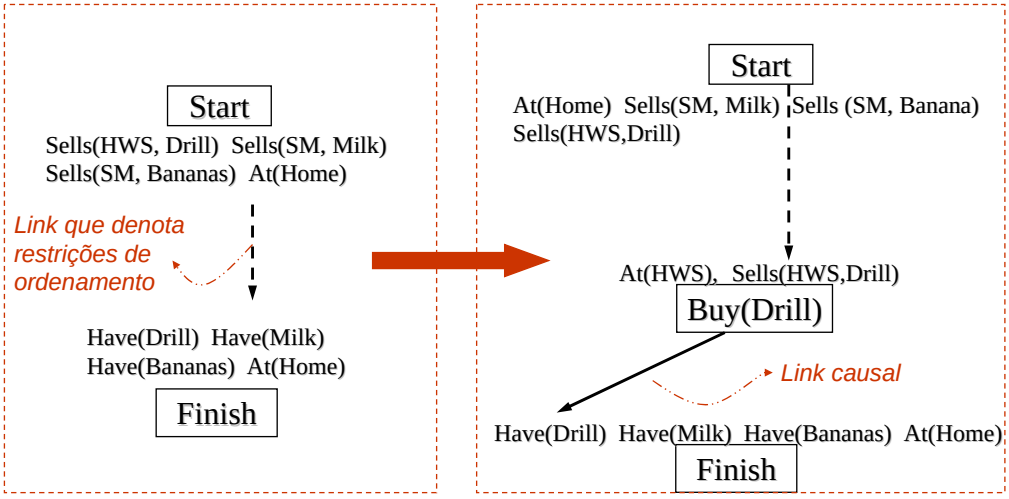


- **Ações**

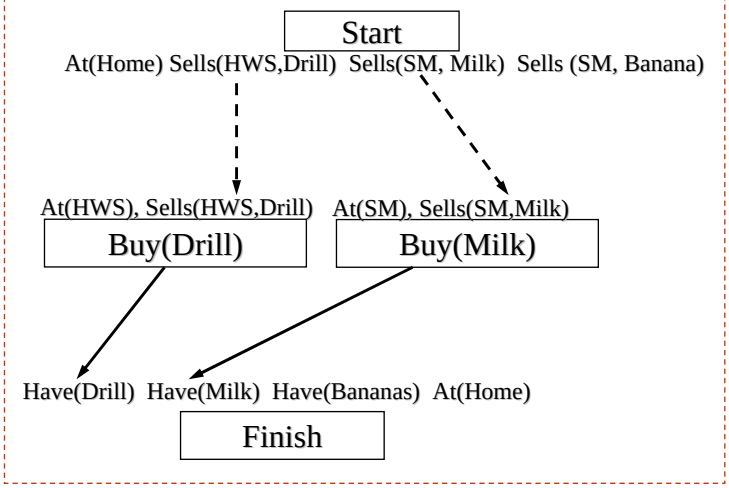
Op(ACTION: **Go**(there), PRECOND: At(here),
EFFECT: At(there) ∧ ¬ At(here))
Op(ACTION: **Buy**(x), PRECOND: At(store) ^ Sells(store, x),
EFFECT: Have(x))

21

Planejamento Parcial – Exemplo (1)

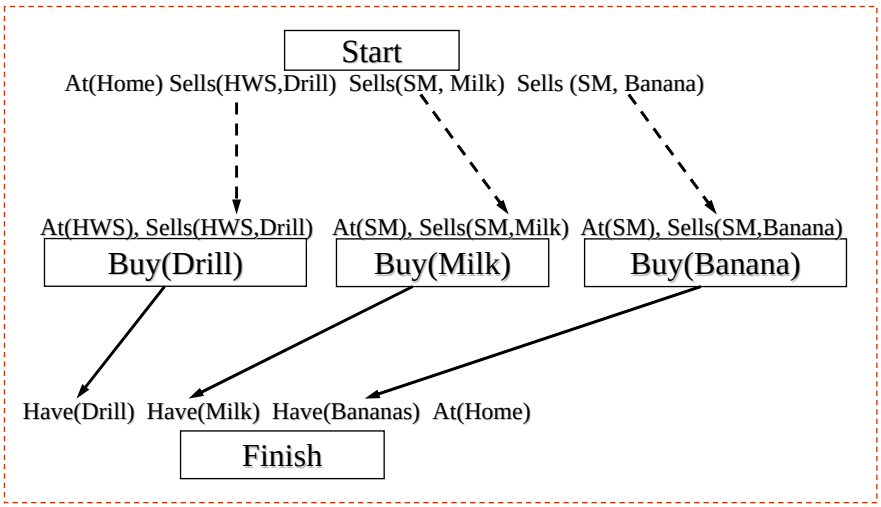


Planejamento Parcial – Exemplo (2)



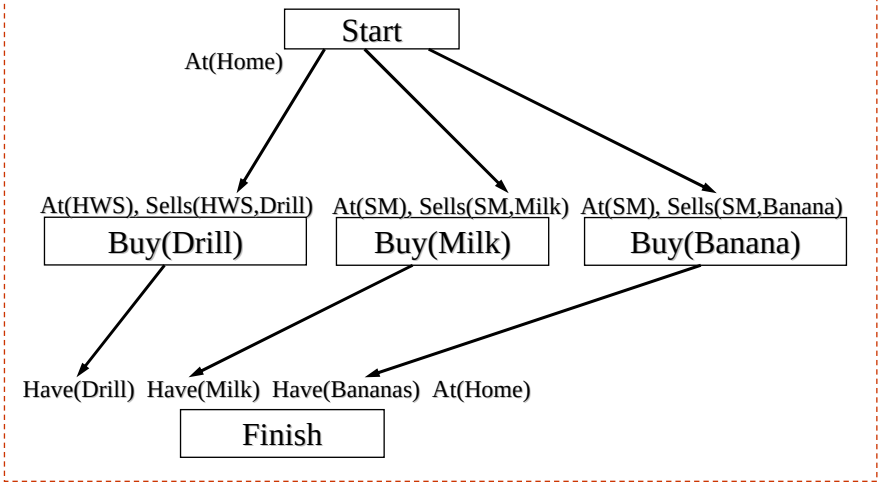
23

Planejamento Parcial – Exemplo (3)



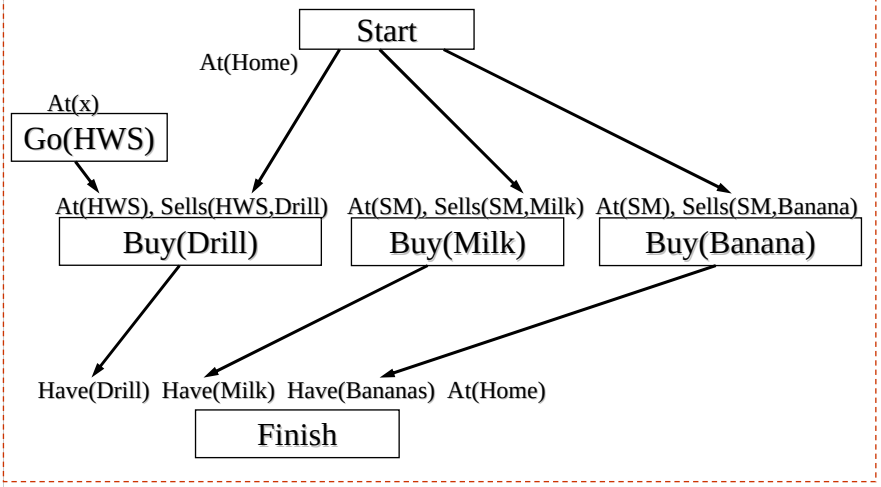
24

Planejamento Parcial – Exemplo (4)



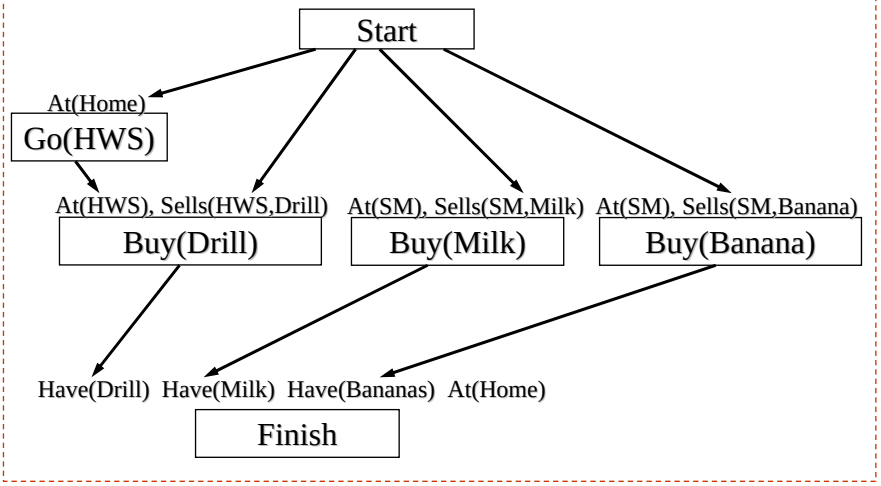
25

Planejamento Parcial – Exemplo (5)



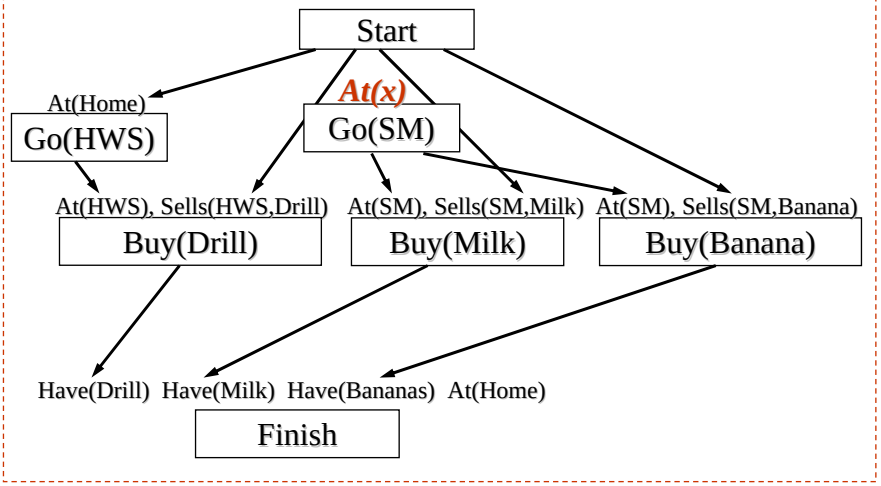
26

Planejamento Parcial – Exemplo (6)



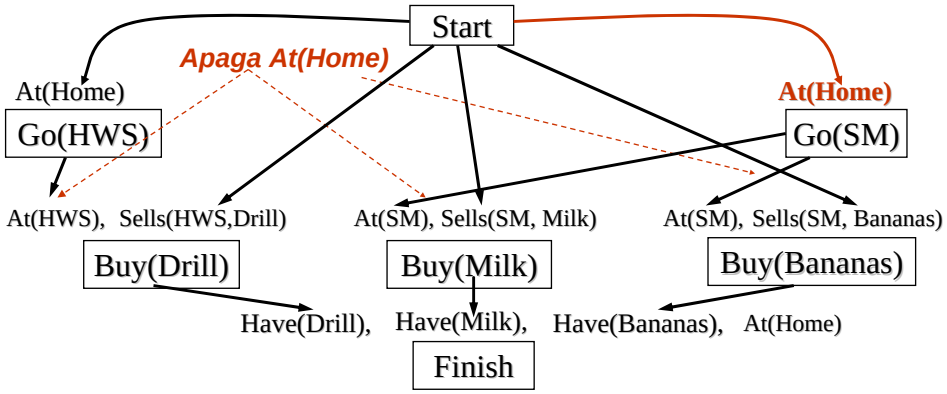
27

Planejamento Parcial – Exemplo (7)



28

Planejamento Parcial – Exemplo (8)



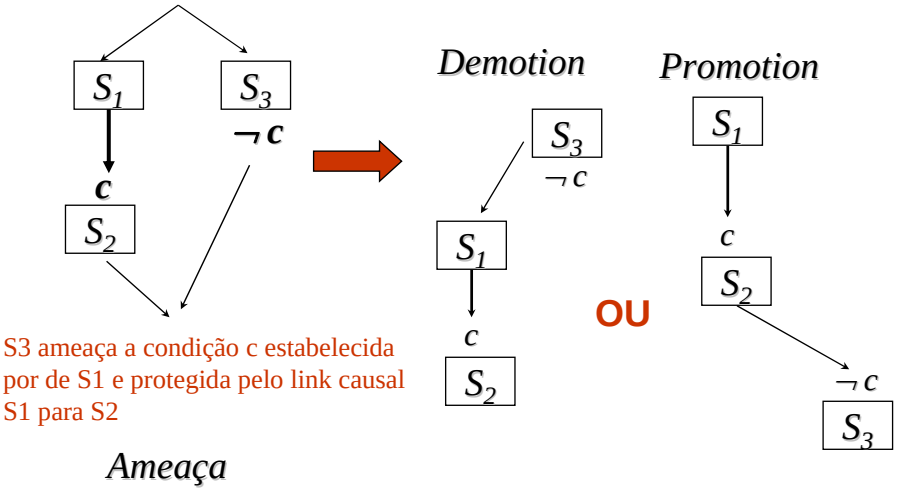
PROBLEMA: Considere que a pré-condição At(x) do Go(SM) foi satisfeita através de uma ligação à condição At(Home) do Start; se o agente decidir ir primeiro à HWS, ele não mais poderá sair de casa para ir ao SM, pois Go(HWS) adiciona At(HWS), mas também remove At(Home)!!! (e vice-versa: indo de casa ao SM, não mais consegue ir de casa à HWS) ➔ AMEAÇA

Problema da ameaça

- **Ameaça**
  - ocorre quando os efeitos de um passo põem em risco as pré-condições de outro
- **Com testar?**
  - O novo passo é inconsistente com condição protegida (link causal)
  - O passo antigo é inconsistente com nova condição protegida

30

Ameaça - soluções



S3 ameaça a condição c estabelecida por de S1 e protegida pelo link causal S1 para S2

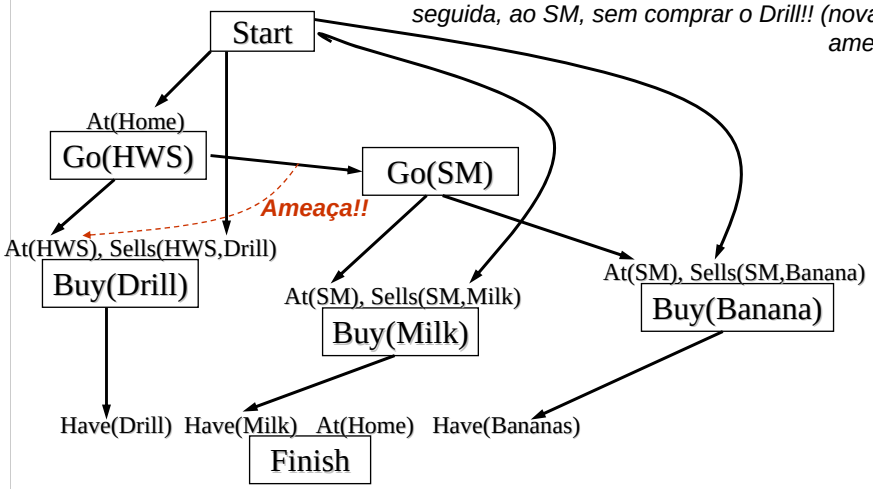
Ameaça

31

Planejamento Parcial – Exemplo (9)

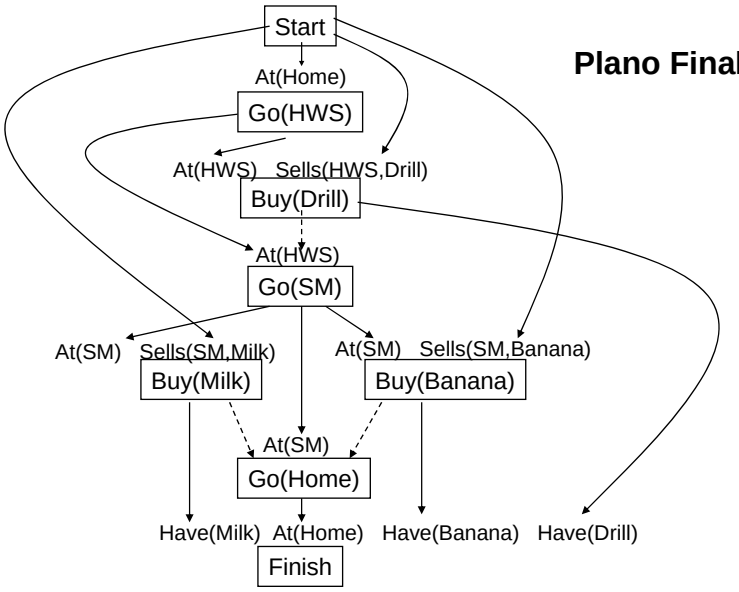
Tentativas de solução:

1. *Demotion* não dá: Go(SM) ficaria antes de Start???
2. *Promotion: sairia de Home, iria à HWS e, logo em seguida, ao SM, sem comprar o Drill!! (nova ameaça)*



32

Planejamento Parcial – Exemplo (10)



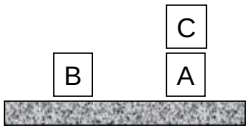
Plano Final

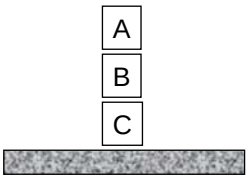
33

Mundo dos blocos

- **Com isto é possível resolver problemas do "mundo dos blocos"...**

34





Estado inicial:

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

Estado final (objetivo):

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

Ações:

PutOn(x,y) ➔ **P:** On(x,z), Clear(y), Clear(x)
Add: On(x,y), Clear(z)
Del: On(x,z), Clear(y)

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

PutOnTable(x) ➔ **P:** On(x,z), Clear(x)
Add: On(x,Table), Clear(z)
Del: On(x,z)

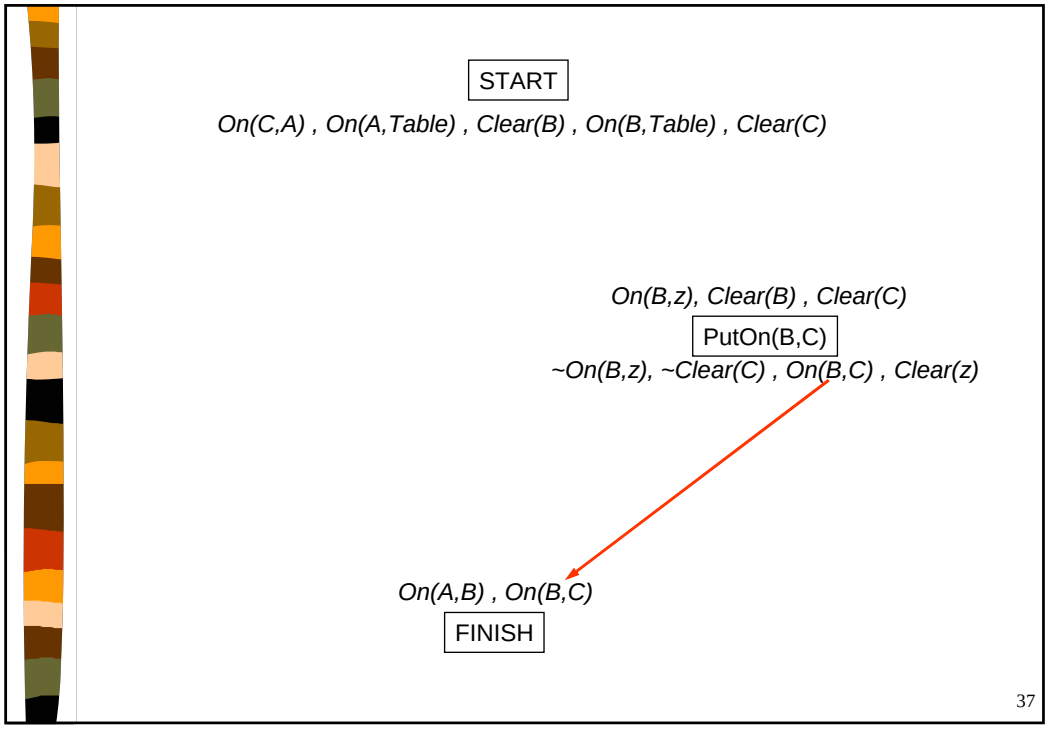
On(x,z), Clear(x)
PutOnTable(x)
On(x,Table), Clear(z), ~On(x,z)

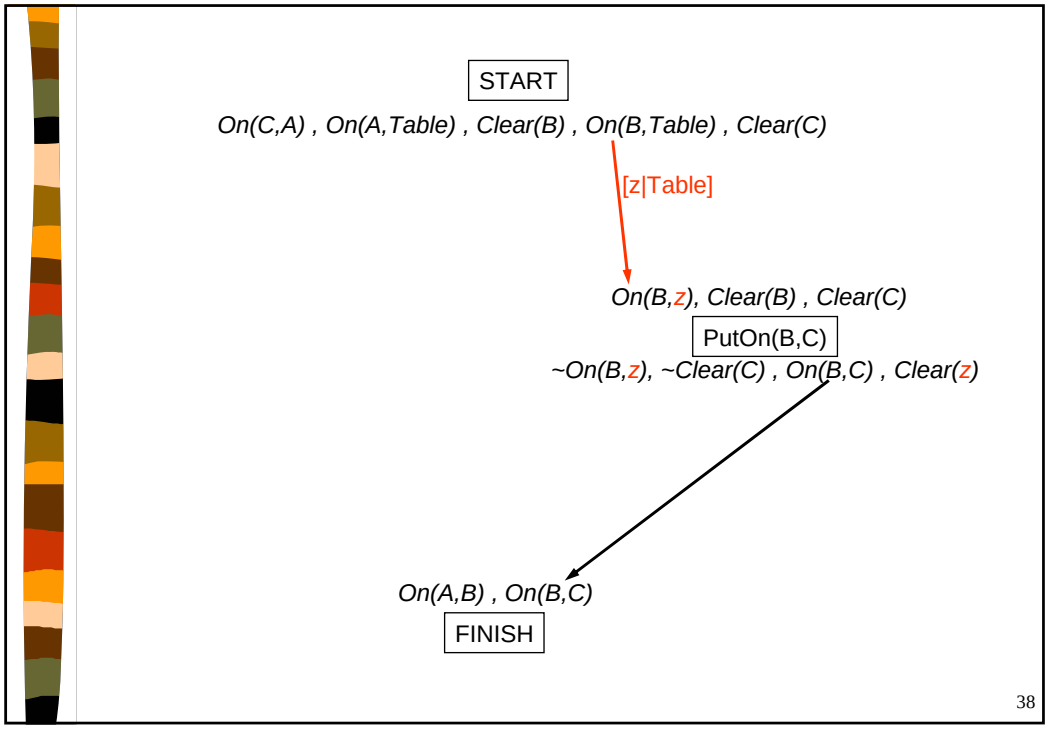
35

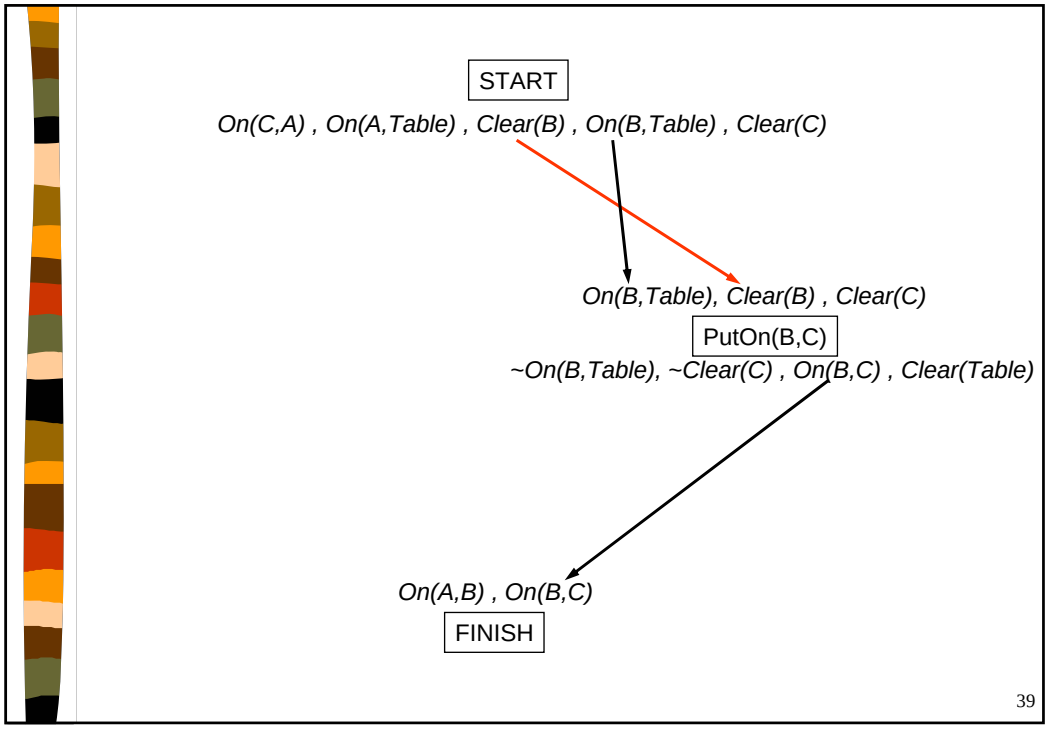
START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)

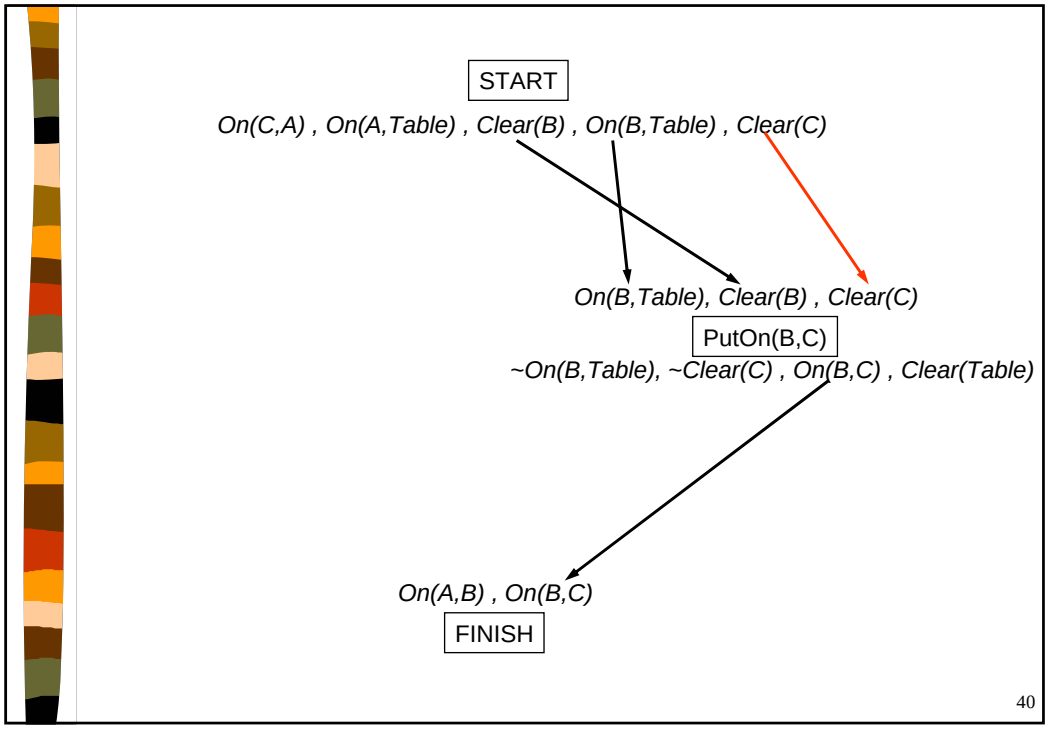
On(A,B) , On(B,C)
FINISH

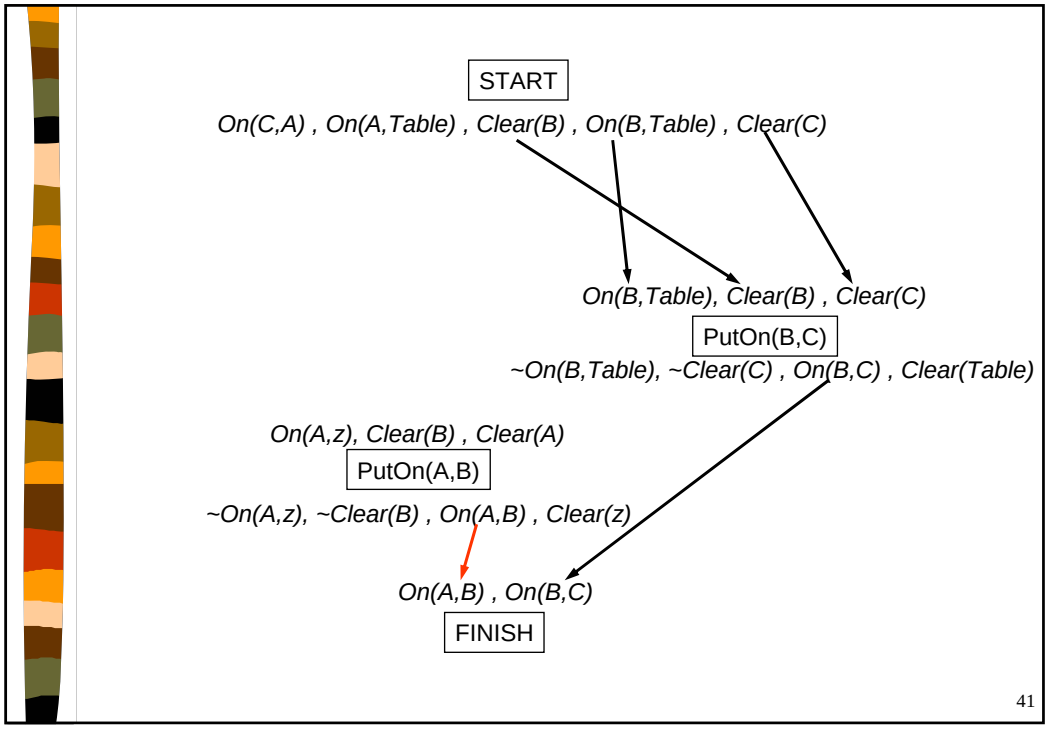
36

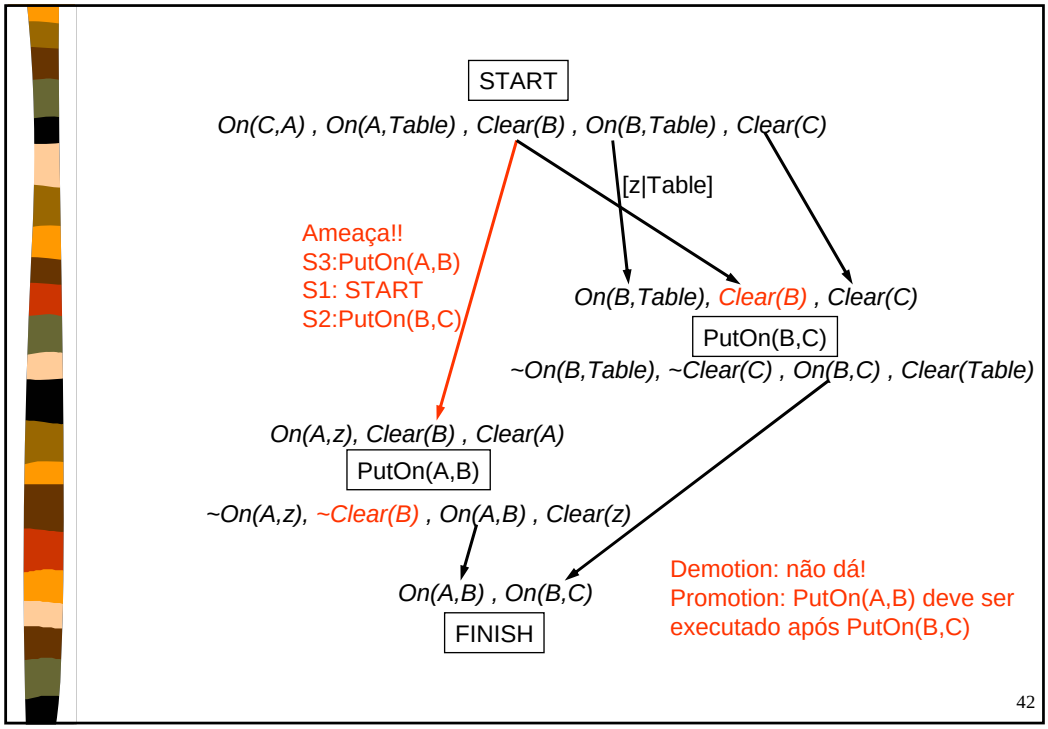

START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
On(B,z), Clear(B) , Clear(C)
PutOn(B,C)
~On(B,z), ~Clear(C) , On(B,C) , Clear(z)
On(A,B) , On(B,C)
FINISH
37

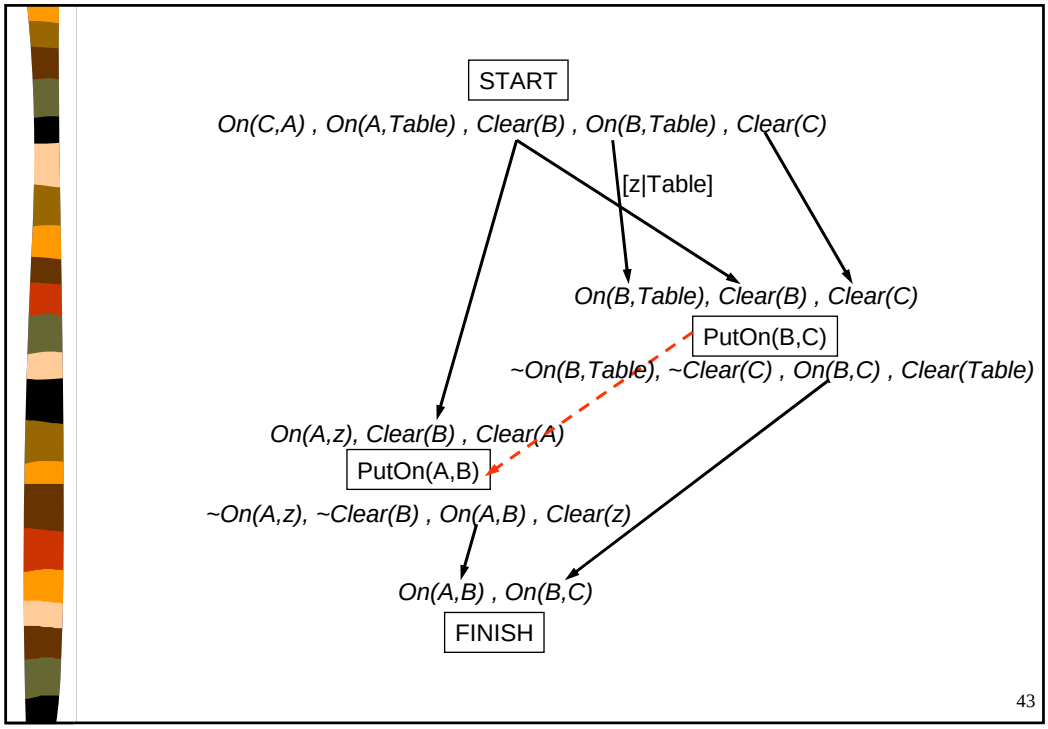

START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
[z|Table]
On(B,z), Clear(B) , Clear(C)
PutOn(B,C)
~On(B,z), ~Clear(C) , On(B,C) , Clear(z)
On(A,B) , On(B,C)
FINISH
38

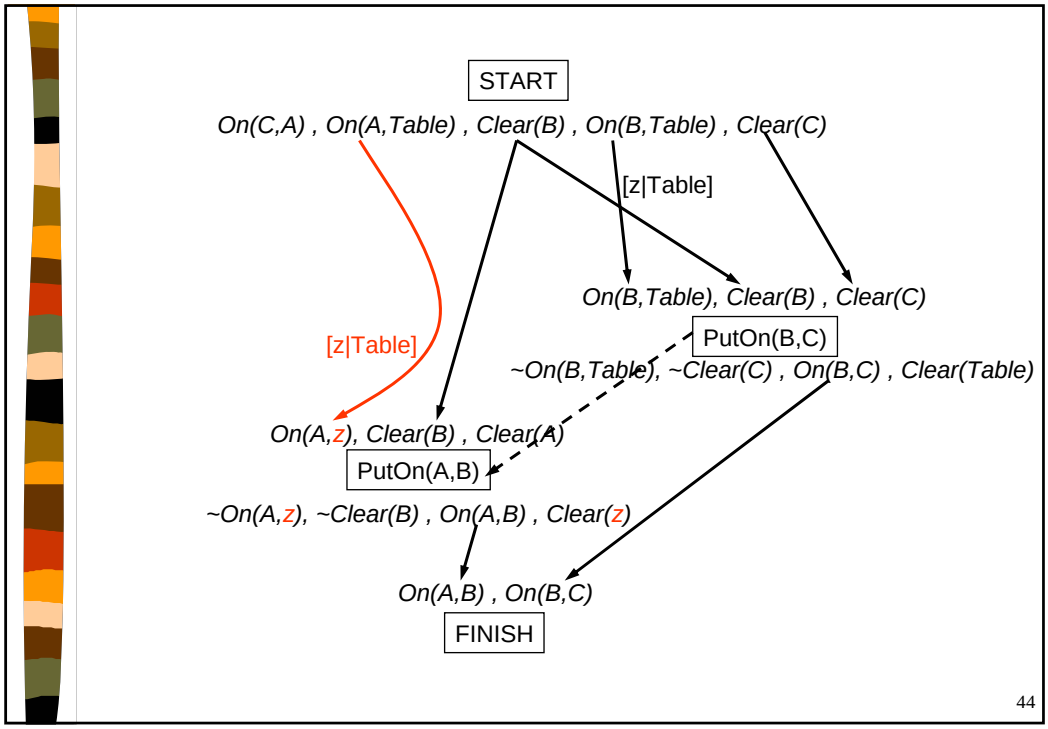

START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
On(B,Table), Clear(B) , Clear(C)
PutOn(B,C)
~On(B,Table), ~Clear(C) , On(B,C) , Clear(Table)
On(A,B) , On(B,C)
FINISH
39

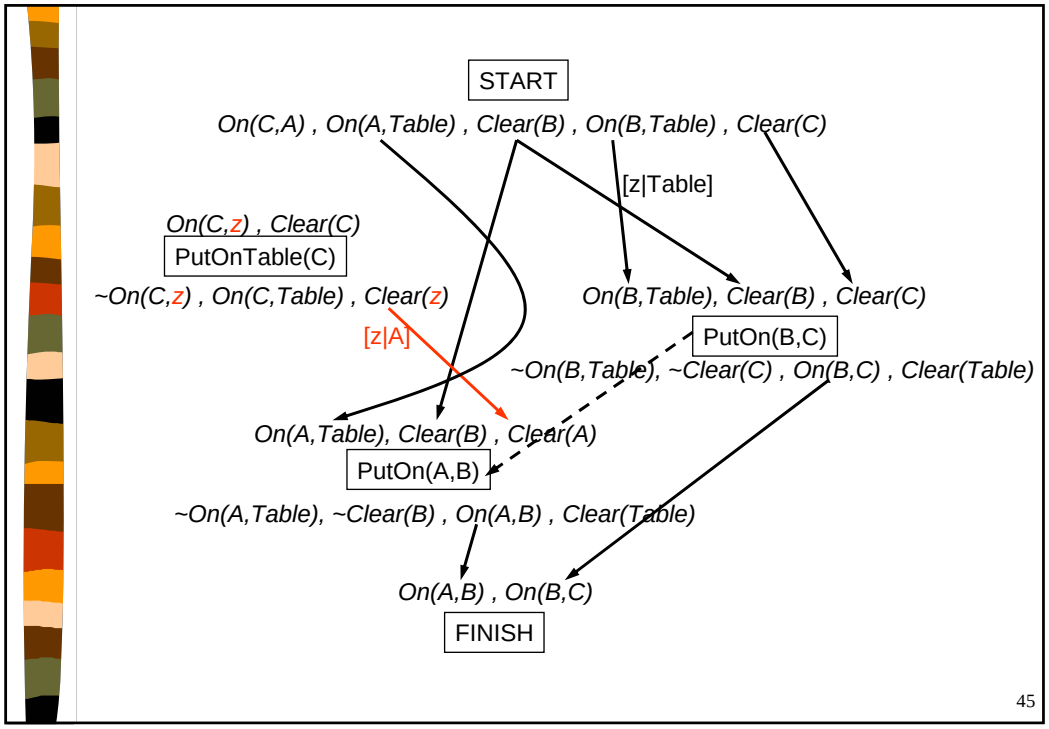

START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
On(B,Table), Clear(B) , Clear(C)
PutOn(B,C)
~On(B,Table), ~Clear(C) , On(B,C) , Clear(Table)
On(A,B) , On(B,C)
FINISH
40

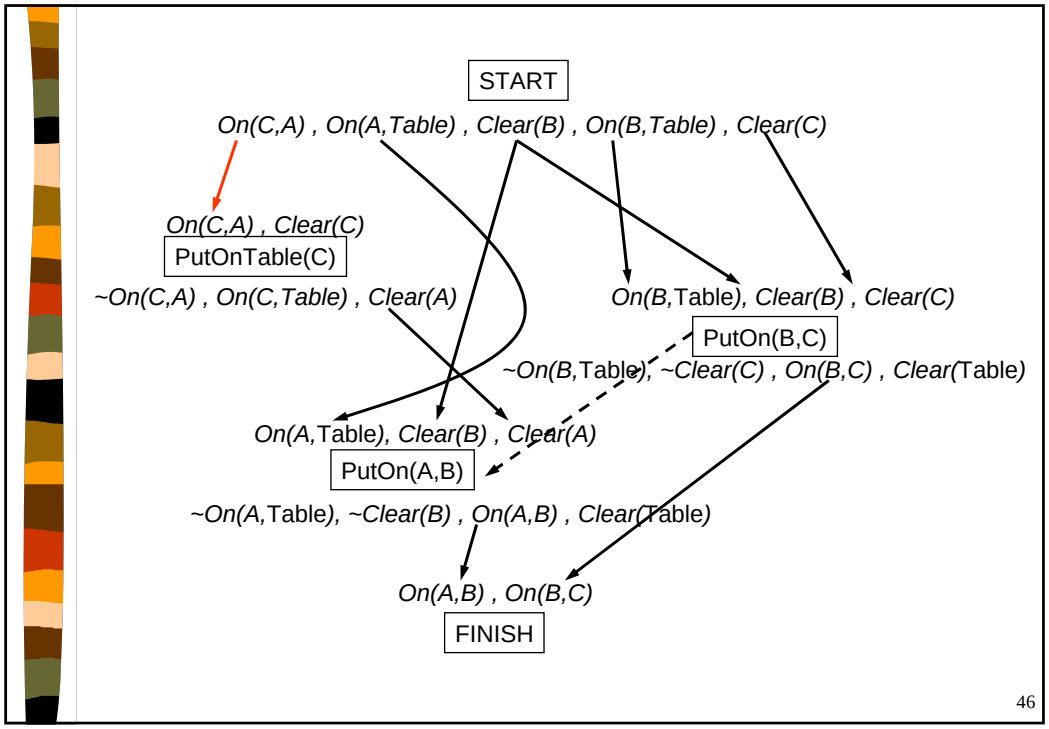

START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
On(B,Table), Clear(B) , Clear(C)
PutOn(B,C)
~On(B,Table), ~Clear(C) , On(B,C) , Clear(Table)
On(A,z), Clear(B) , Clear(A)
PutOn(A,B)
~On(A,z), ~Clear(B) , On(A,B) , Clear(z)
On(A,B) , On(B,C)
FINISH
41

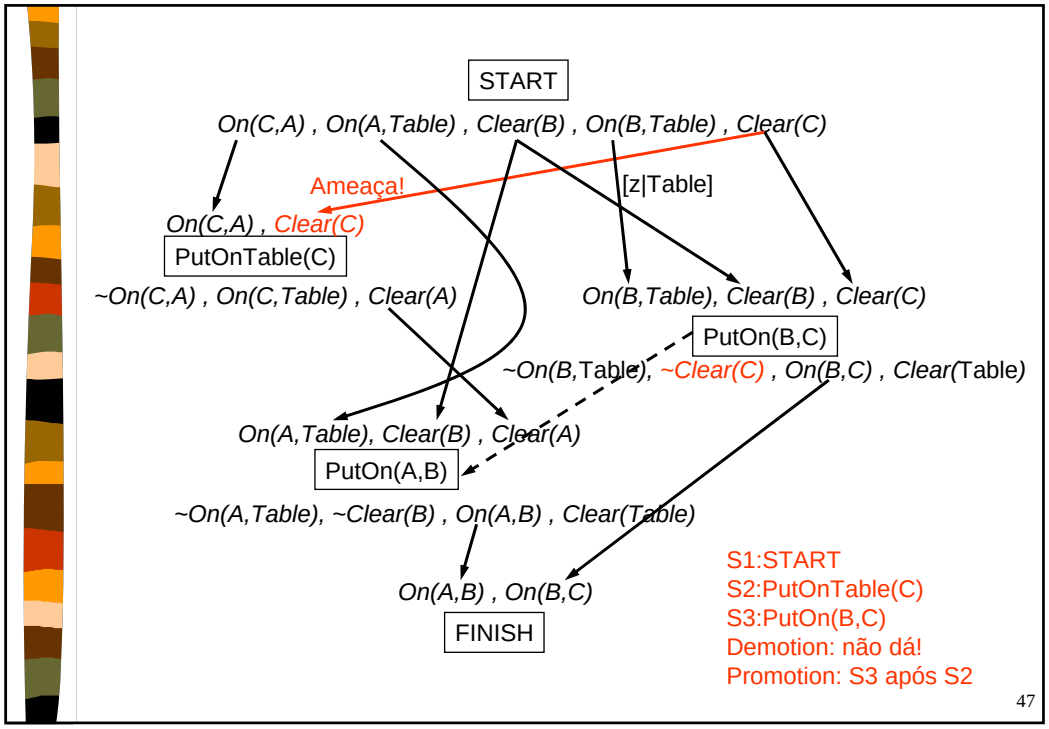

START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
[z|Table]
Ameaça!!
S3:PutOn(A,B)
S1: START
S2:PutOn(B,C)
On(B,Table), Clear(B) , Clear(C)
PutOn(B,C)
~On(B,Table), ~Clear(C) , On(B,C) , Clear(Table)
On(A,z), Clear(B) , Clear(A)
PutOn(A,B)
~On(A,z), ~Clear(B) , On(A,B) , Clear(z)
On(A,B) , On(B,C)
FINISH
Demotion: não dá!
Promotion: PutOn(A,B) deve ser executado após PutOn(B,C)
42

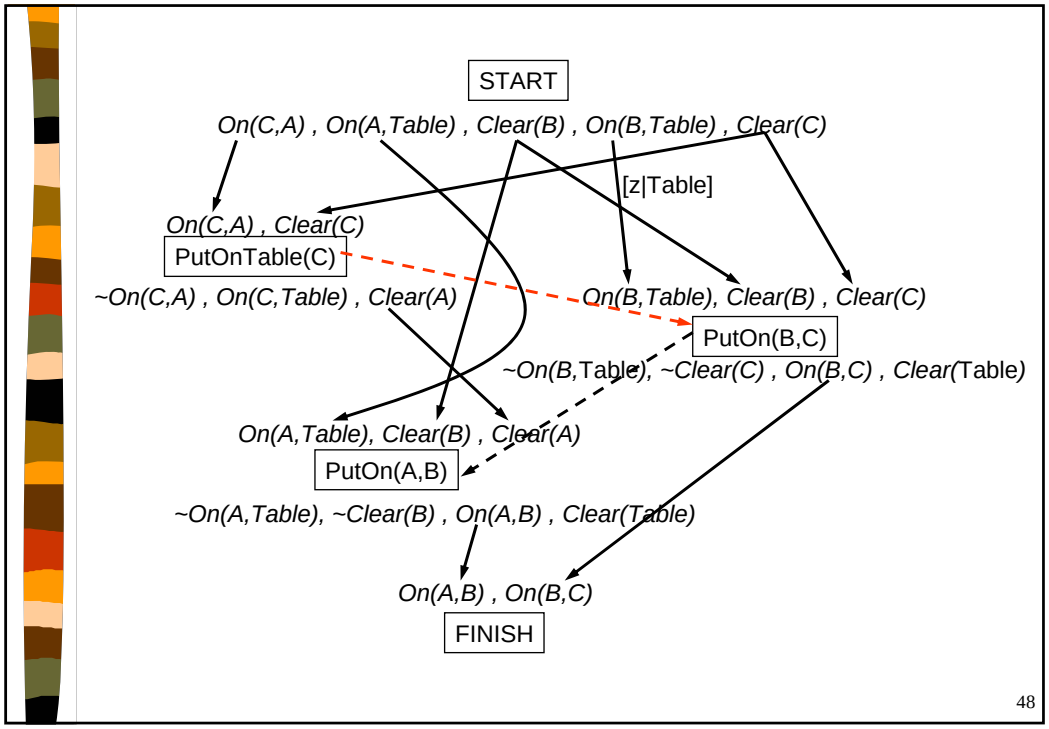

START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
[z|Table]
On(B,Table), Clear(B) , Clear(C)
PutOn(B,C)
~On(B,Table), ~Clear(C) , On(B,C) , Clear(Table)
On(A,z), Clear(B) , Clear(A)
PutOn(A,B)
~On(A,z), ~Clear(B) , On(A,B) , Clear(z)
On(A,B) , On(B,C)
FINISH
43

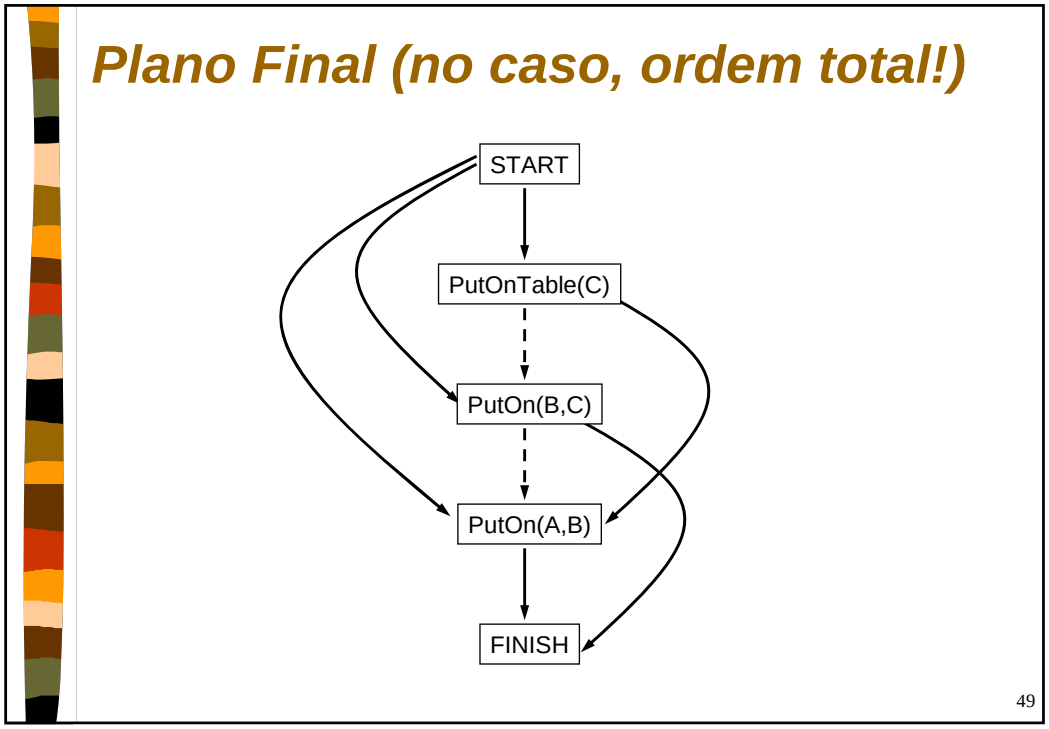

START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
[z|Table]
On(B,Table), Clear(B) , Clear(C)
PutOn(B,C)
[z|Table]
~On(B,Table), ~Clear(C) , On(B,C) , Clear(Table)
On(A,z), Clear(B) , Clear(A)
PutOn(A,B)
~On(A,z), ~Clear(B) , On(A,B) , Clear(z)
On(A,B) , On(B,C)
FINISH
44


START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
[z|Table]
On(C,z) , Clear(C)
PutOnTable(C)
~On(C,z) , On(C,Table) , Clear(z)
On(B,Table), Clear(B) , Clear(C)
[z|A]
PutOn(B,C)
~On(B,Table), ~Clear(C) , On(B,C) , Clear(Table)
On(A,Table), Clear(B) , Clear(A)
PutOn(A,B)
~On(A,Table), ~Clear(B) , On(A,B) , Clear(Table)
On(A,B) , On(B,C)
FINISH
45


START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
On(C,A) , Clear(C)
PutOnTable(C)
~On(C,A) , On(C,Table) , Clear(A)
On(B,Table), Clear(B) , Clear(C)
PutOn(B,C)
~On(B,Table), ~Clear(C) , On(B,C) , Clear(Table)
On(A,Table), Clear(B) , Clear(A)
PutOn(A,B)
~On(A,Table), ~Clear(B) , On(A,B) , Clear(Table)
On(A,B) , On(B,C)
FINISH
46


START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
Ameaça!
[z|Table]
On(C,A) , Clear(C)
PutOnTable(C)
~On(C,A) , On(C,Table) , Clear(A)
On(B,Table), Clear(B) , Clear(C)
PutOn(B,C)
~On(B,Table), ~Clear(C) , On(B,C) , Clear(Table)
On(A,Table), Clear(B) , Clear(A)
PutOn(A,B)
~On(A,Table), ~Clear(B) , On(A,B) , Clear(Table)
On(A,B) , On(B,C)
FINISH
S1:START
S2:PutOnTable(C)
S3:PutOn(B,C)
Demotion: não dá!
Promotion: S3 após S2
47


START
On(C,A) , On(A,Table) , Clear(B) , On(B,Table) , Clear(C)
[z|Table]
On(C,A) , Clear(C)
PutOnTable(C)
~On(C,A) , On(C,Table) , Clear(A)
On(B,Table), Clear(B) , Clear(C)
PutOn(B,C)
~On(B,Table), ~Clear(C) , On(B,C) , Clear(Table)
On(A,Table), Clear(B) , Clear(A)
PutOn(A,B)
~On(A,Table), ~Clear(B) , On(A,B) , Clear(Table)
On(A,B) , On(B,C)
FINISH
48


Plano Final (no caso, ordem total!)
START
PutOnTable(C)
PutOn(B,C)
PutOn(A,B)
FINISH
49