

---

# Planejamento 1

## Capítulo 11

# Tipos de agentes

---

- **Agente reativo:** seleciona ações baseando-se em percepções ou na representação interna do estado do mundo
- **Agente *problem-solving*:** decide o que fazer procurando uma sequência de ações que o leve a um estado meta (ex.:  $A^*$ )
- **Agente baseado em conhecimento:** seleciona ações baseando-se na representação lógica explícita do estado corrente e dos efeitos de suas ações

# Agente de Planejamento Clássico:

## ***PROBLEMA***

---

Planejamento como um **Modelo de Estados**. Dado:

- um espaço de estados  $S$ , finito e não-vazio
- um estado inicial  $s_0 \in S$
- um conjunto de estados meta  $S_G \subseteq S$
- um conjunto de ações aplicáveis  $A(s) \subseteq A$ , para  $\forall s \in S$
- uma função de transição  $s' = f(a, s)$ , para  $\forall s', s \in S$  e  $\forall a \in A(s)$
- uma função custo  $c(a, s) > 0$ , para  $\forall s \in S$  e  $\forall a \in A(s)$

# Problema de Planejamento Clássico

---

Dado  $\langle s_0, S_G, A, f \rangle$ , encontrar uma sequência de ações aplicáveis (*plano*) que mapeiam  $s_0$  num estado meta  $S_G$ .

Uma **solução ótima** minimiza a soma dos custos das ações.

# Suposições de planejamento clássico

---

- tempo atômico
- existência de um único agente para executar as ações do plano
- determinismo
- onisciência
- mudanças no mundo ocorrem apenas através das ações do agente

# Agente de Planejamento:

## ***ALGORITMO***

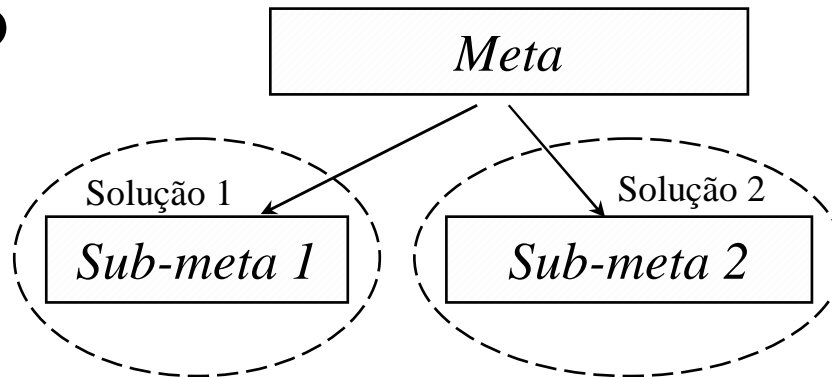
---

- Pode ser modelado como um problema de busca:
  - **agente *problem-solving* (ou *state-space planner*)**: faz busca num espaço de estados. A sequência de ações corresponde a um caminho da árvore de busca.
  - **agente de planejamento (ou *plan-space planner*)**: faz busca num “espaço de planos”. Cada nó da árvore de busca representa uma sequência de ações.
- Pode ser modelado como uma teoria lógica, composta por axiomas descrevendo ações e condições do mundo. Uso de provador de teoremas para inferir um plano.

# Idéias básicas sobre planejamento

---

- Dividir problemas complexos em problemas menores. A combinação das soluções pode ser usada para fornecer uma solução (plano) para o problema completo



- Execução do plano: requer monitoração em ambientes que envolvam o problema de contingência

# Agente simples de planejamento: planejamento e execução

---

## Geração do plano:

dado o **estado inicial** do mundo e um **estado meta**, o agente pode *executar* um algoritmo de planejamento para gerar um plano de ações

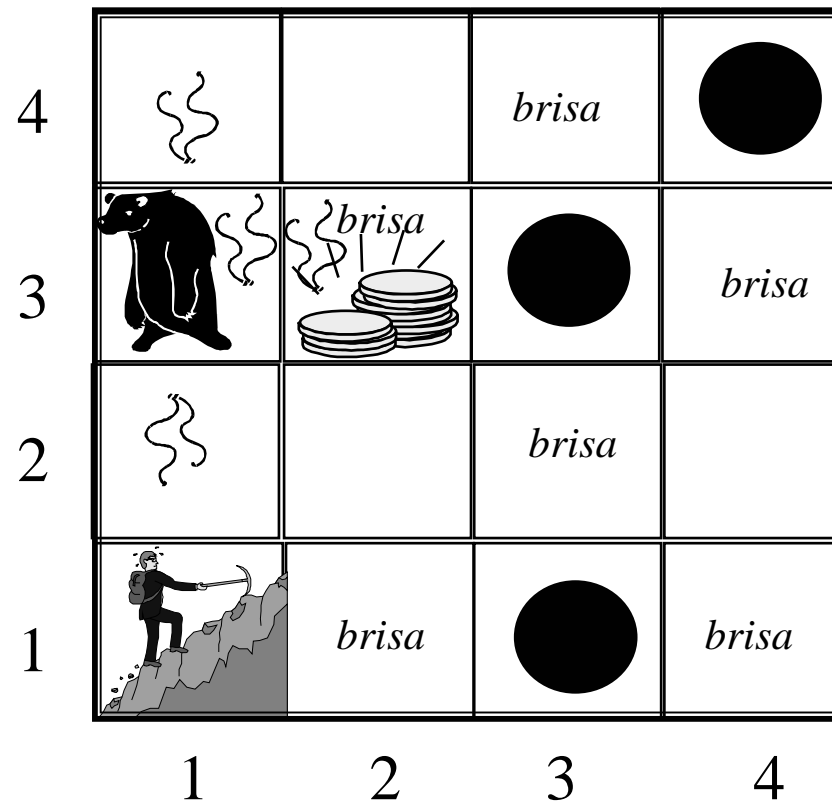
## Execução do plano:

o agente poderá executar uma ação (passo do plano) por vez, intercalando com observações do mundo



# Mundo do Wumpus

---



Objetivo: sair da caverna depois de pegar o ouro

# Mundo do Wumpus: ambiente

---

- uma posição adjacente ao Wumpus *cheira* mal
- uma posição adjacente ao abismo emite uma *brisa*
- uma posição que contém ouro *brilha*
- o agente pode estar direcionado para *N, S, L, O*
- o agente possui somente uma flexa
- a ação *atirar a flexa* mata o Wumpus se o agente estiver de frente para ele
- a ação *pegar o ouro* só é possível ser executada se o agente estiver na mesma posição que o ouro
- a ação *soltar* deixa o ouro na mesma posição que o agente estiver

# Mundo do Wumpus: agente

---

- **percepção:** brisa, cheiro, brilho, escuta, choque (não percebe sua localização)
- **ação:** escala, vira para direita, vira para esquerda, vai para frente, segura, solta, atira
- **metas:** encontrar o ouro e trazê-lo para o início o mais rápido possível sem entrar numa posição com abismo ou com Wumpus
  - O agente morre se entra em uma posição que contém o Wumpus vivo ou um abismo

# Mundo do Wumpus

---

- Determinístico?
- Acessível?
- Estático?
- Discreto?

# Agente simples de planejamento

---

- No mundo do Wumpus o acesso é local
- Se o mundo for totalmente acessível, um agente pode usar a percepção para construir um modelo completo do estado atual (localização do agente, do Wumpus, dos abismos, posições já visitadas, morte do Wumpus, percepções passadas, etc.)
- O mundo muda somente com as ações do agente

# Agente simples de planejamento

---

**Function** Simple-Planning-Agent (*percept*) **returns** an *action*

**static:** *KB* (includes action description)

*p*, a plan, initially *NoPlan*

*t*, a counter to indicate time, initially 0

**local variables:** *G*, a goal, *current-state* is a current state description

TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))

*current-state*  $\leftarrow$  STATE-DESCRIPTION(*KB*, *t*)

**if** *p* = *NoPlan* **then**

$G \leftarrow$  ASK(*KB*, Make-Goal-Query(*t*))

$p \leftarrow$  IDEAL-PLANNER(*current-state*, *G*, *KB*)

**if** *p* = *NoPlan* **or** *p* is empty **then** *action*  $\leftarrow$  *NoOp* ; *p*  $\leftarrow$  *NoPlan*

**else**  $action \leftarrow$  FIRST(*p*)

$p \leftarrow$  REST(*p*)

TELL(*KB*, MAKE-PERCEPT-SENTENCE(*action*, *t*))

$t \leftarrow t + 1$

**return** *action*

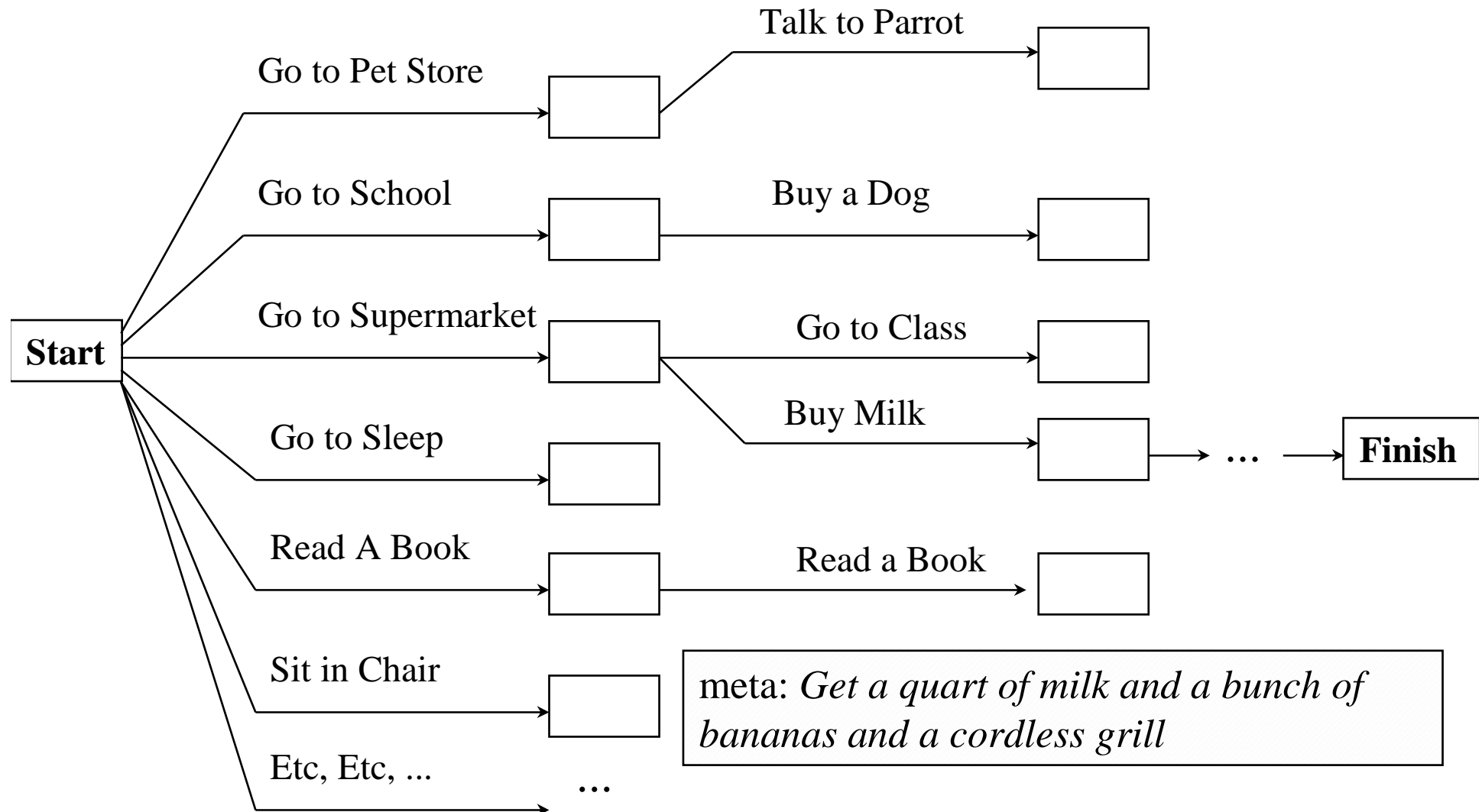
# Busca em um espaço de estados

---

- Busca progressiva
- Busca regressiva
- Estratégias: BFS, ID, A\*, WA\*

# Exemplo: busca progressiva através de um espaço de estados do mundo

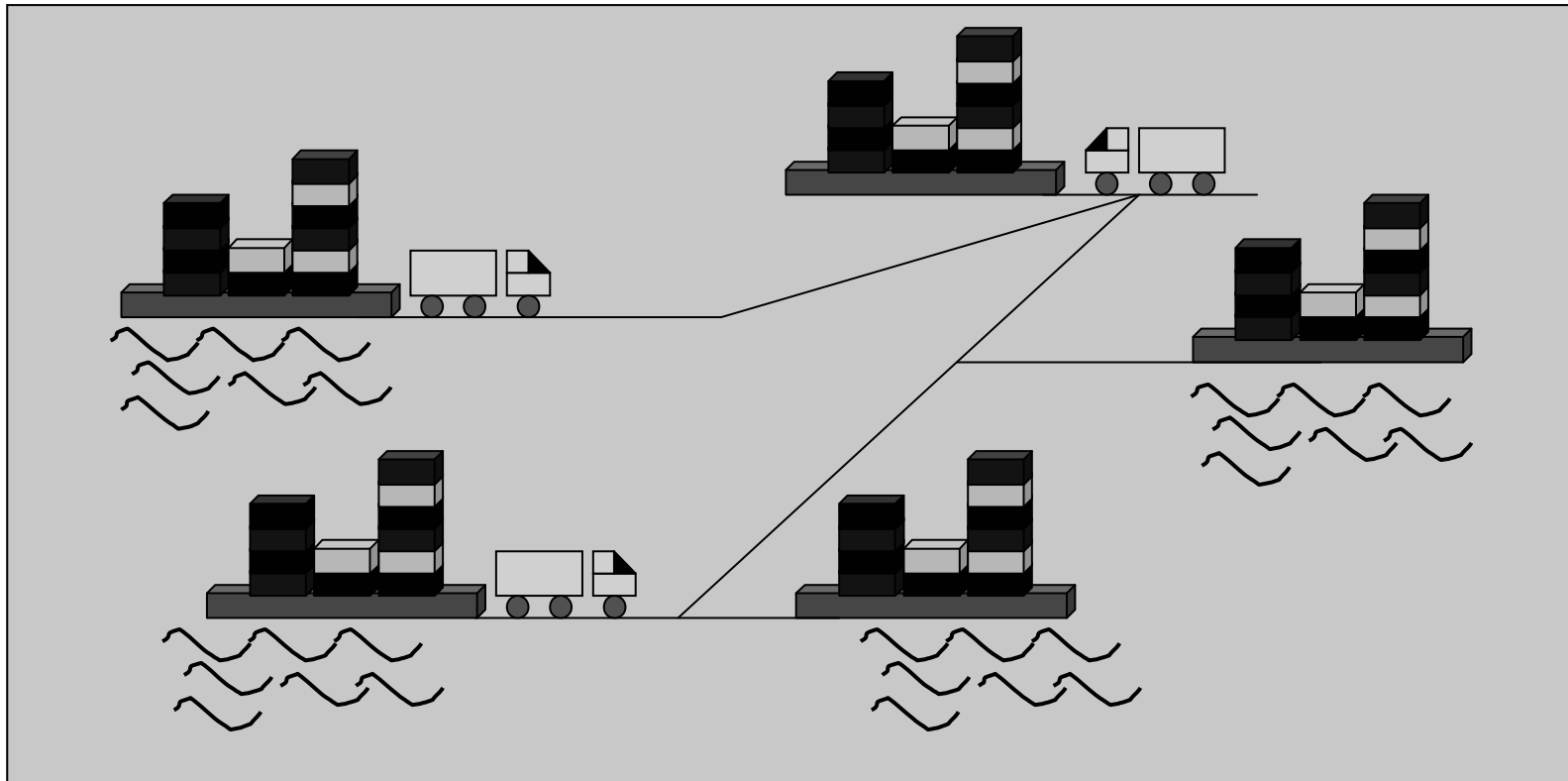
---





# Domínio de Logística

---



5 localizações, 3 pilhas, 100 containers  $\rightarrow 10^{277}$  estados

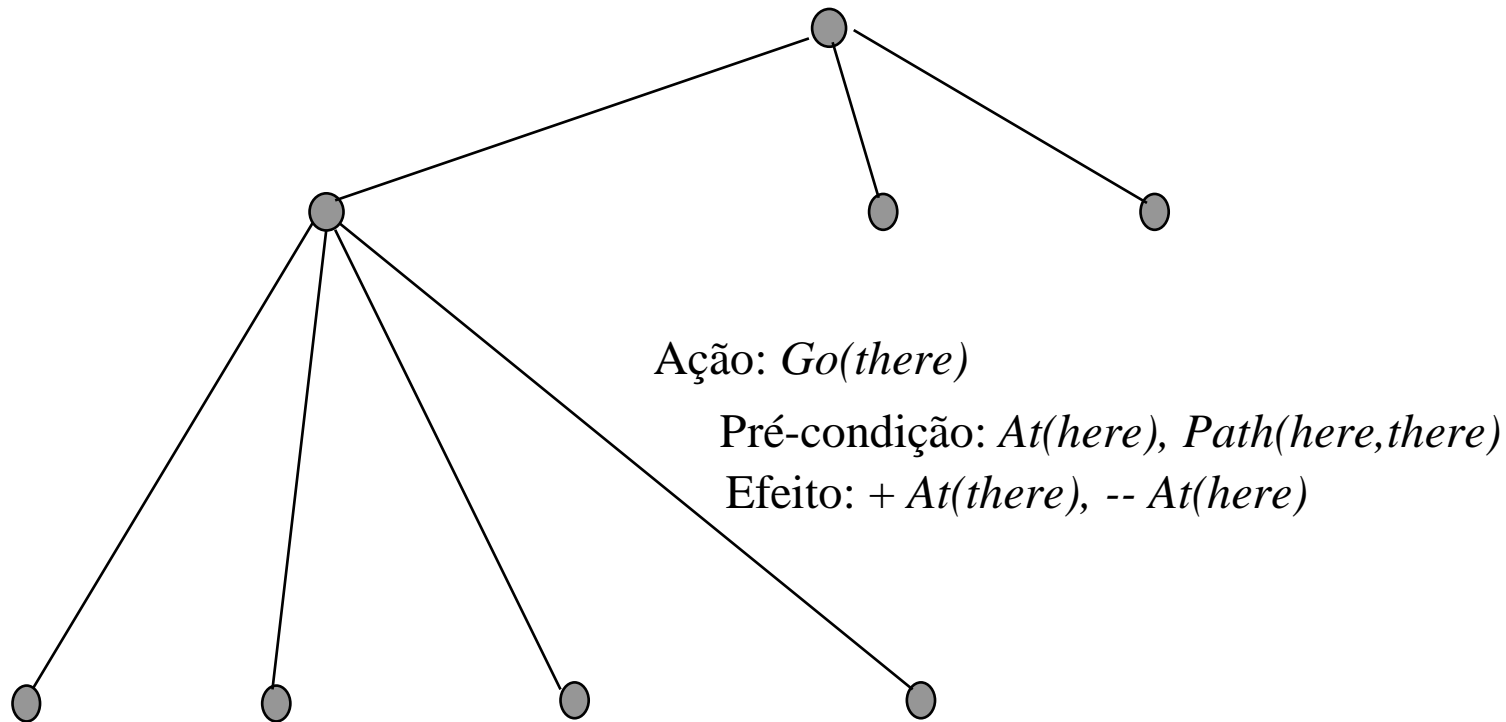
# Como resolver as dificuldades dos agentes que fazem busca?

---

- O agente precisa de uma maneira mais flexível para raciocinar sobre ações
- Solução de planejamento: “**abrir**” [AIMA] a representação de ações e estados:
  - passar da Lógica Proposicional para Lógica de Predicados
  - representar ações através de suas pré-condições e efeitos

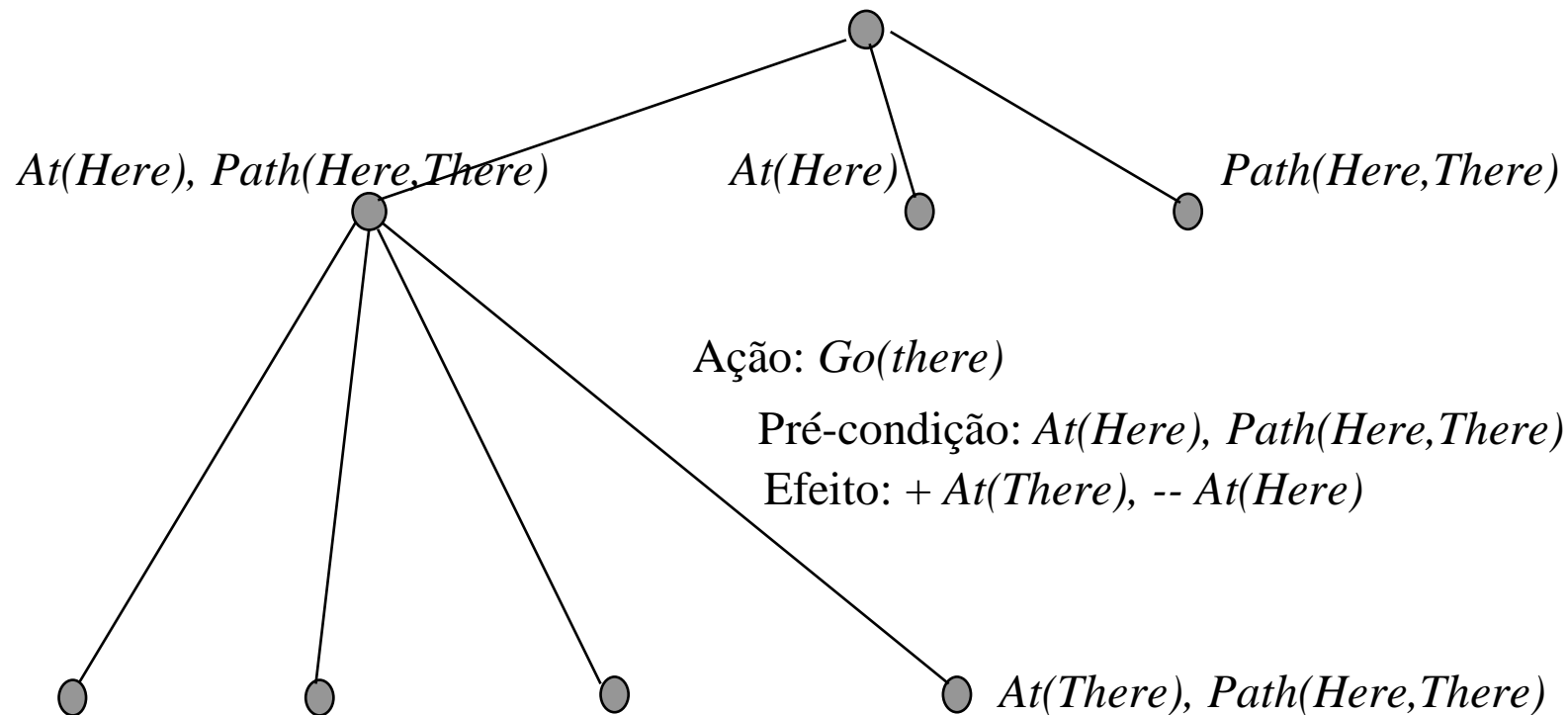
# Busca com ações estruturadas

---



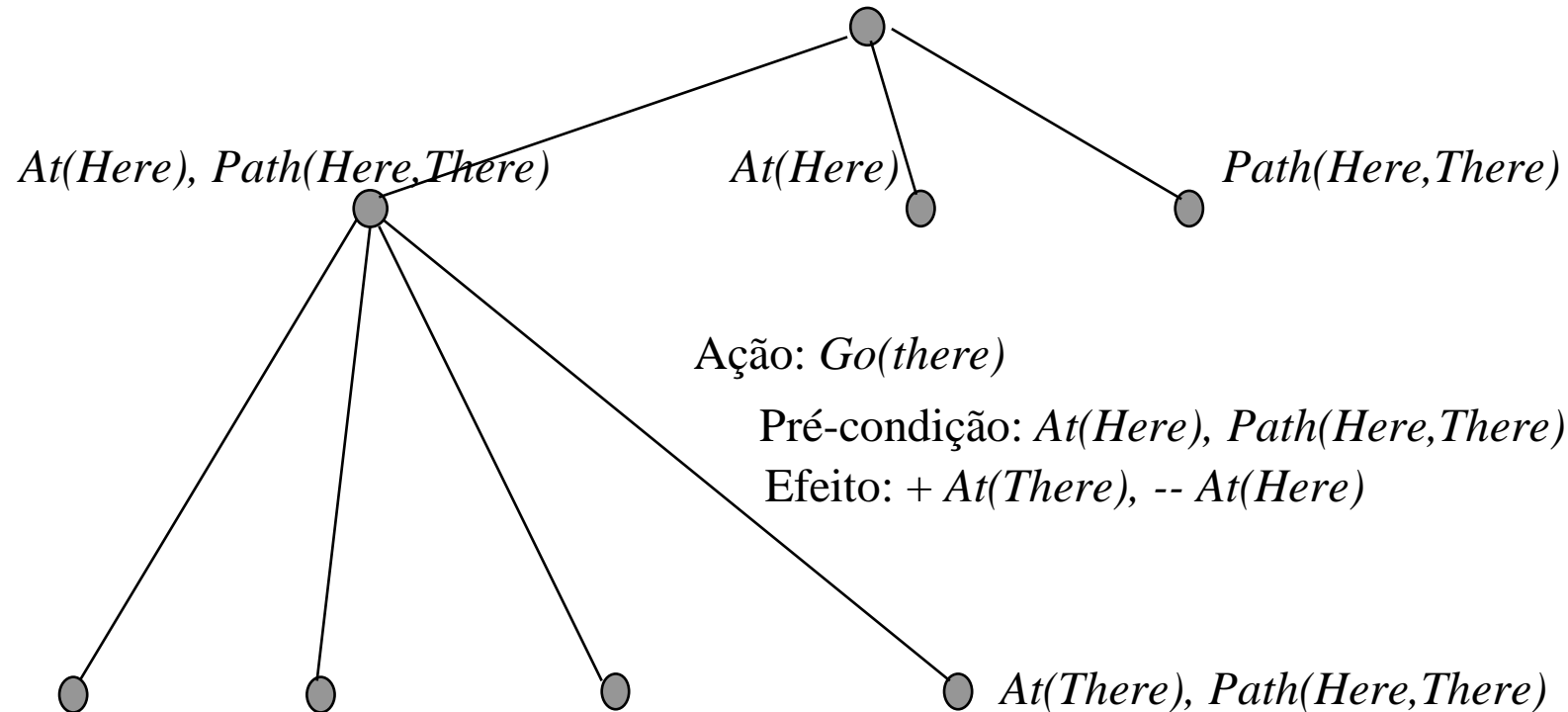
# Busca com estados estruturados

---



# Busca com planos estruturados

---



*Plano* (Passos do plano: { $P_1$ : *Op*(Ação: *Start* Pré-condição: { } ),  
 $P_2$ : *Op*(Ação: *Finish* Pré-condição: *At(There)* and *Path(Here, There)* } ,  
Ordenações: { $P_1 < P_2$  } ,  
Unificações: { } ,  
Vínculos Causais: { } )

# Representação de Conhecimento estruturada

---

Algoritmos de planejamento devem usar alguma **linguagem formal** para descrever estados, metas ou ações (LPO ou um sub-conjunto de LPO):

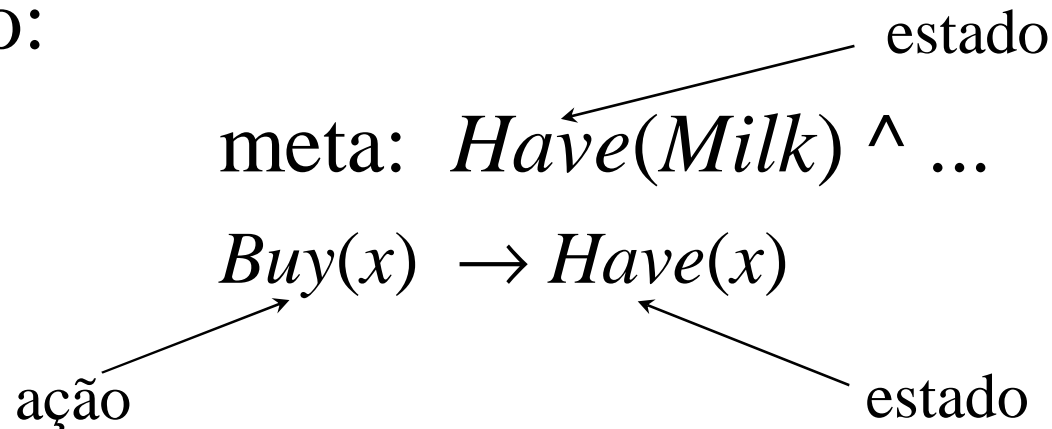
- estados e metas: conjunto de sentenças da lógica
- ações: implicações lógicas (precondições e efeitos)

# Inferência em Planejamento

---

Planejamento permite que o agente faça uma correlação direta entre estados, ações e metas.

Exemplo:



O agente conclue que vale a pena incluir *Buy(Milk)* no plano (note que esse raciocínio não é uma dedução lógica mas uma abdução lógica!)

# Cálculo de situações

---

- **Ontologia:** situações, fluentes e ações
- **Linguagem:**
  - $s_0$
  - $do(\alpha, \sigma)$
  - $poss(\alpha, \sigma)$
  - $holds(\phi, \sigma)$



# Mundo dos Blocos

---

- **fluentes:**

*clear(X)*

*ontable(X)*

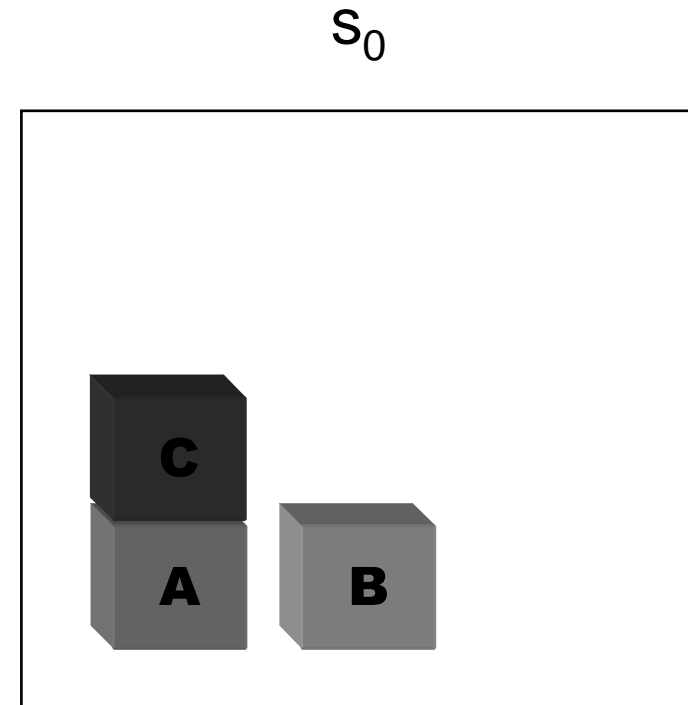
*on(X,Y)*

- **ações:**

*stack(X,Y)*

*unstack(X,Y)*

*move(X,Y,Z)*



# Especificação lógica do domínio

---

- axiomas de observação:

$holds(clear(c), s_0)$

$holds(on(c,a), s_0)$

...

- axiomas de efeito:

$holds(clear(Y), do(move(X,Y,Z), S))$

$holds(on(X,Z), do(move(X,Y,Z), S))$

...

- axiomas de precondições:

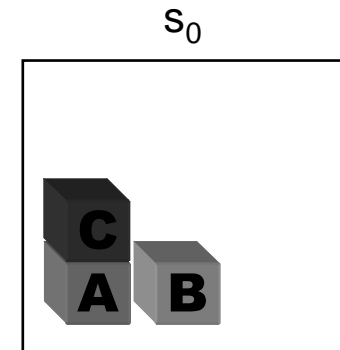
$poss(move(X,Y,Z), S) \leftarrow$

$holds(clear(X), S) \wedge holds(clear(Z), S) \wedge holds(on(X,Y), S)$

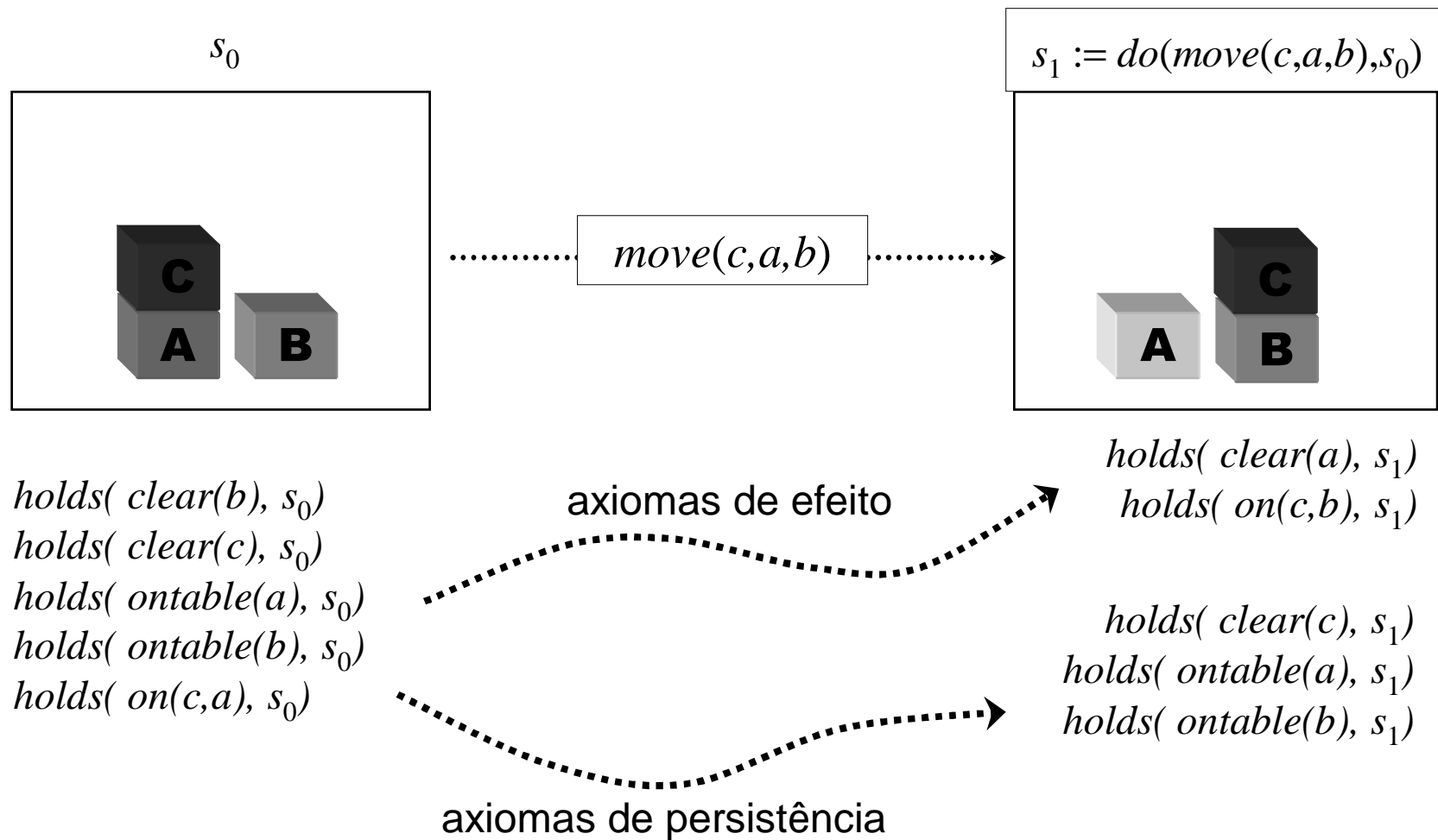
...

- axioma de persistência (frame axiom):

$holds(F, do(A, S)) :- poss(A, S), holds(F, S), \text{ not affects}(A, F).$



# O problema da persistência



# Descrição de meta em Planejamento

---

- Dada uma meta, o agente planejador *pergunta* por uma sequência de ações (plano) que quando executada torna o estado meta verdadeiro, a partir do estado inicial
- Provador de teoremas: *pergunta* se a sentença da meta é verdadeira em uma dada Base de Conhecimento (KB)

# Planejamento dedutivo

---

Dados:

$\mathcal{A}$  : axiomatização do domínio

$I$  : situação inicial

$\mathcal{G}$  : meta de planejamento

O planejamento consiste em provar que

$$\mathcal{A} \wedge I \models \exists S[exec(S) \wedge \mathcal{G}(S)],$$

sendo executabilidade definida indutivamente por:

$$exec(s_0)$$

$$exec(do(A,S)) \leftarrow poss(A,S) \wedge exec(S)$$

# Um planejador em PROLOG

---

holds(clear(b),s0).

holds(clear(c),s0).

holds(ontable(a),s0).

holds(ontable(b),s0).

holds(on(c,a),s0).

holds(on(X,Y),do(stack(X,Y),S)).

holds(clear(Y),do(unstack(X,Y),S)).

holds(ontable(X),do(unstack(X,Y),S)).

poss(stack(X,Y),S) :- holds(ontable(X),S), holds(clear(X),S), holds(clear(Y),S),  
X\=Y.

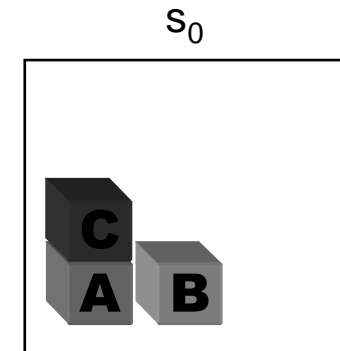
poss(unstack(X,Y),S) :- holds(clear(X),S), holds(on(X,Y),S).

holds(F,do(A,S)) :- poss(A,S), holds(F,S), not affects(A,F).

affects(stack(X,Y),clear(Y)).

affects(stack(X,Y),ontable(X)).

affects(unstack(X,Y),on(X,Y)).



# Um planejador em PROLOG

---

Para busca em profundidade iterativa, adicionar:

```
exec(s0).  
exec(do(A,S)) :- poss(A,S), exec(S).  
  
plan(s0).  
plan(do(A,S)) :- plan(S).
```

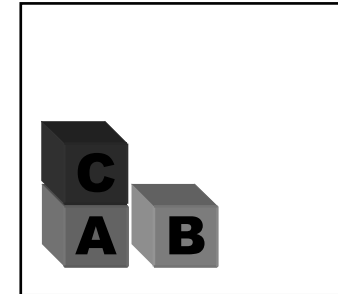
# Consultando o planejador

$s_0$

?- plan(S), exec(S), holds(on(a,c),S).

S = do(stack(a,c),do(unstack(c,a),s0))

yes



?- plan(S), exec(S), holds(on(a,b),S), holds(on(b,c),S).

S = do(stack(a,b),do(stack(b,c),do(unstack(c,a),s0)))

yes

?- holds(F, do(stack(a,b),do(stack(b,c),do(unstack(c,a),s0)))).

F = on(a,b) ;

F = on(b,c) ;

F = clear(a) ;

F = ontable(c) ;

no



# A linguagem Strips (Fikes e Nilsson, 1971)

---

- Strips (*Stanford Research Institute Problem-Solving*) é a mais antiga, simples e usada linguagem de planejamento (sistema Strips)
- Evolução: Strips  $\rightarrow$  ADL  $\rightarrow$  PDDL
- Um problema em Strips é uma tupla  $\langle A, O, I, G \rangle$ 
  - $A$  é o conjunto de todos os átomos (variáveis booleanas descritores de estados),
  - $O$  é um conjunto de todos os operadores (ações proposicionais), e
  - $I \subseteq A$  representa a situação inicial (descrição completa de estado)
  - $G \subseteq A$  representa as situações meta (descrição parcial de estados)

# Operador Strips

---

Ação:  $Buy(x)$

Pré-condição:  $At(p), Sells(p,x)$  //conjunção de átomos(literais positivos)

Efeito:  $Have(x)$  //conjunção de literais (positivos ou negativos)  
(a versao original de Strips divide os efeitos em **add list** e **delete list**)

Linguagem restrita  $\implies$  algoritmo eficiente

Representação gráfica:

$At(p) \quad Sells(p,x)$

$Buy(x)$

$Have(x)$

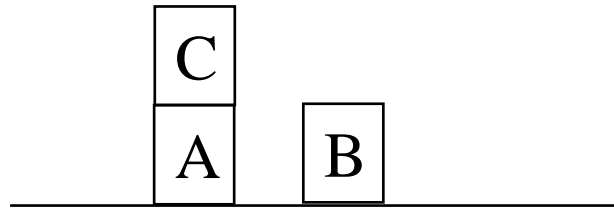
# Mundo dos Blocos

---

- $Move(x,y,z)$  move bloco x de cima de y para cima de z e z pode ser a mesa ou outro bloco
- $Move$  é aplicável somente se x e z estiverem livres (*clear*) e x estiver sobre y.
- Um bloco está livre (*clear*) se não possui nenhum outro bloco em cima dele
- A mesa possui sempre espaço livre

# Mundo dos Blocos

---



Estado Inicial:

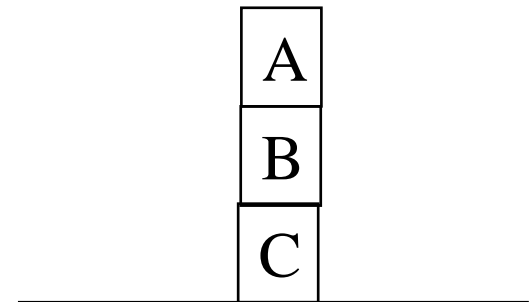
$On(B, table)$

$On(A, table)$

$On(C, A)$

$Clear(C)$

$Clear(B)$



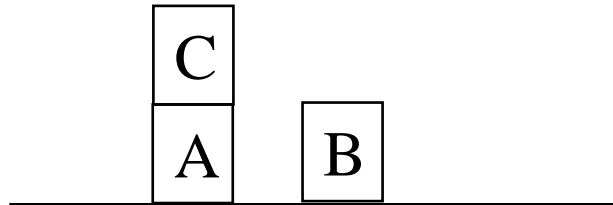
Estado Meta:

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

# Mundo dos Blocos

---

Problema: *Anomalia de Sussman*



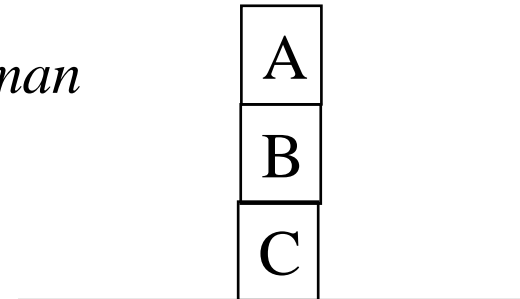
Estado Inicial

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

$PutOn(x,y)$

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

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



Estado Meta

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

$PutOnTable(x)$

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

$On(x,Table),$

# Busca no espaço de estados - dificuldades



---

- Existem **muitas ações e estados** para serem considerados (fator de ramificação e profundidade da solução)
- **Informações sobre a meta:** a função de avaliação heurística só pode escolher qual estado está mais próximo da meta mas não pode descartar ações
  - uma meta não pode ser decomposta e não está diretamente relacionada à descrição de ações
- Considerar a sequência completa a partir do estado inicial não permite que o agente trate *interações entre ações*
  - *Anomalia de Susman*

# Planejamento de ordem parcial ou planejamento como busca no espaço de planos

---

Permite:

-  que o agente adicione ações em qualquer lugar do plano, relaxando o requisito de se construir um plano sequencialmente, como na busca no espaço de estados.
-  não fazer compromissos prematuros entre a ordem das ações (geração do plano) e unificação de variáveis

# Planejamento

---

- ✉ Muitas partes do mundo são independentes de outras partes.

Dada a meta: (*get a quart of milk*) and (*get a bunch of bananas*) and (*get a cordless grill*)

- ✉ Planejamento permite que se selecione uma sub-meta de uma sentença conjuntiva

↑ estratégia de resolução de problemas do tipo *divide-and-conquer*



# Descrição de estado

---

- Conjunções de literais instanciados sem funções, ou seja, predicados aplicados a símbolos constantes, possivelmente negados

$Estar-em(Casa) \wedge \neg Ter(Leite) \wedge \neg Ter(Bananas) \wedge \neg Ter(Grelha) \wedge \dots$

- Uma descrição de estado não precisa ser completa: um estado pode corresponder a um conjunto estados (*possible completions*), ou então ...
- *Closed World Assumption*: tudo que for incluído na descrição de estado é verdade; o que não for declarado é falso!

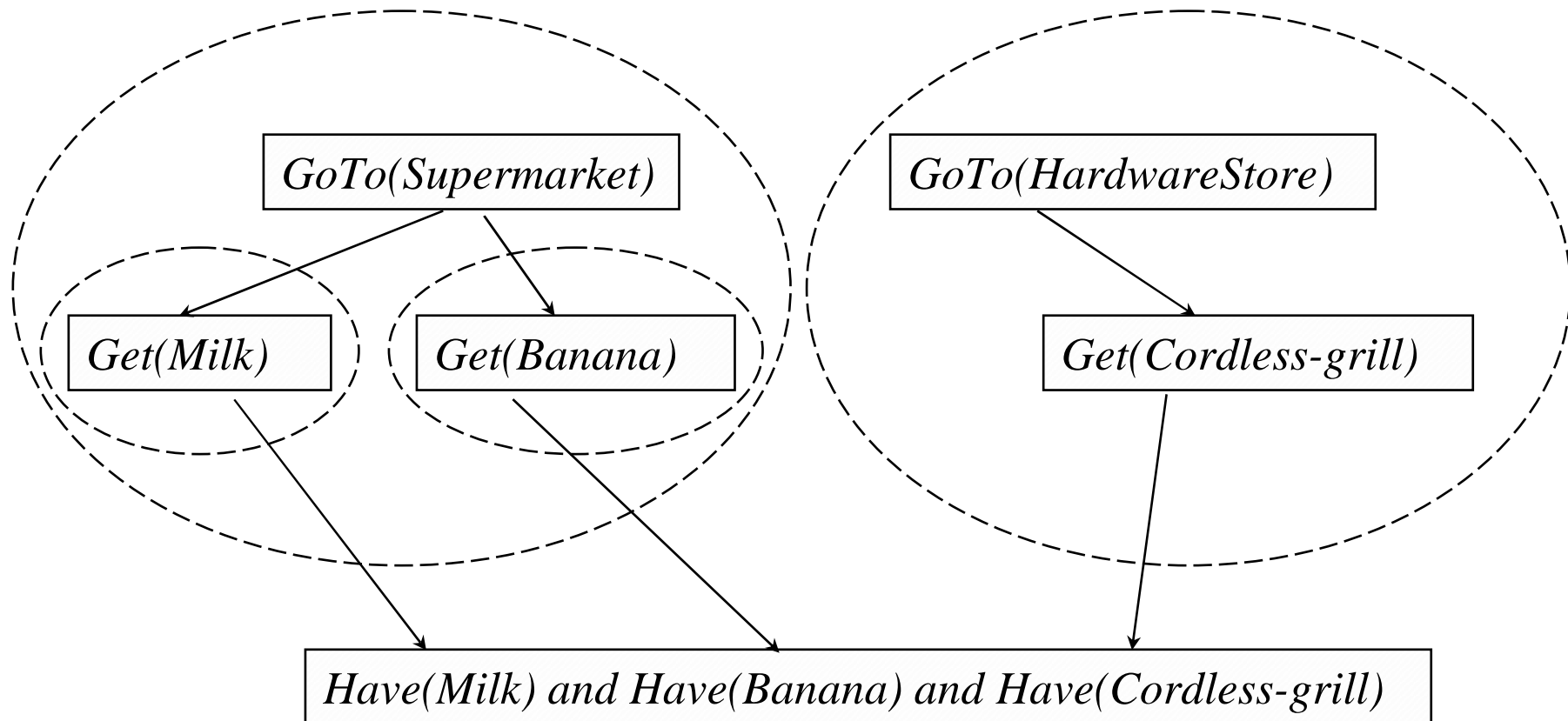
$Estar-em(Casa)$

- metas: também descritos como conjunções de literais, mas podem conter variáveis

$Estar-em(X) \wedge Ter(Leite) \wedge Ter(Bananas) \wedge Ter(Grelha)$

# Planejamento de ordem parcial

---



# Algoritmo POP

---

**function** POP(*initial*, *goal*, *operators*) **returns** *plan*

*plan*  $\leftarrow$  Make-Minimal-Plan(*initial*, *goal*)

**loop do**

**if** Solution? (*plan*) **then return** *plan*

$S_{need}$ , *c*  $\leftarrow$  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*

# POP algorithm (cont.)

---

**procedure** Chose-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  $S_{add} < S_{need}$  to  $Orderings(plan)$

**if**  $S_{add}$  is a newly added step from  $operators$  **then**

add  $S_{add}$  to  $Steps(plan)$

add  $Start < S_{add} < Finish$  to  $Orderings(plan)$

# POP algorithm (cont.)

---

```
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  $S_{threat} < S_i$  to Orderings(plan)  
      Promotion: Add  $S_i < S_{threat}$  to Orderings(plan)  
    if not consistent(plan) then fail  
end
```

POP é correto, completo e sistemático (busca sem repetição)

# Planejamento Heurístico

---

- Heurística  $h(s)$  é computada resolvendo um problema relaxado. Exemplo:
  - distância de Manhattan
  - comprimento do plano solução de um problema relaxado em que são removidas todas as listas de eliminação das ações
- Heurística informativa e admissível ... mas ainda recai num problema intratável

# Descrição de ações do tipo Strips

---

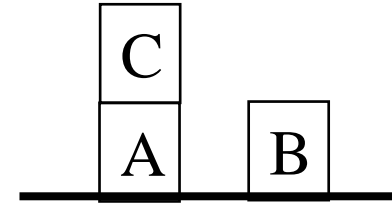
- STRIPS ( *STanford Research Instituto Problem Solver* )
- Desenvolvido em 1970 por Nils Nilsson (originou do GPS)
- *Strips-like language*: linguagem usada pela maioria dos planejadores clássicos
- PDDL: extensão da linguagem Strips, usada em competições de planejamento

# Exemplo: Anomalia de Sussman

---

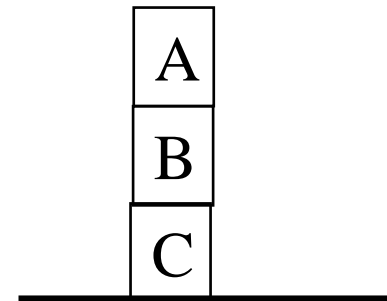
Start

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



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

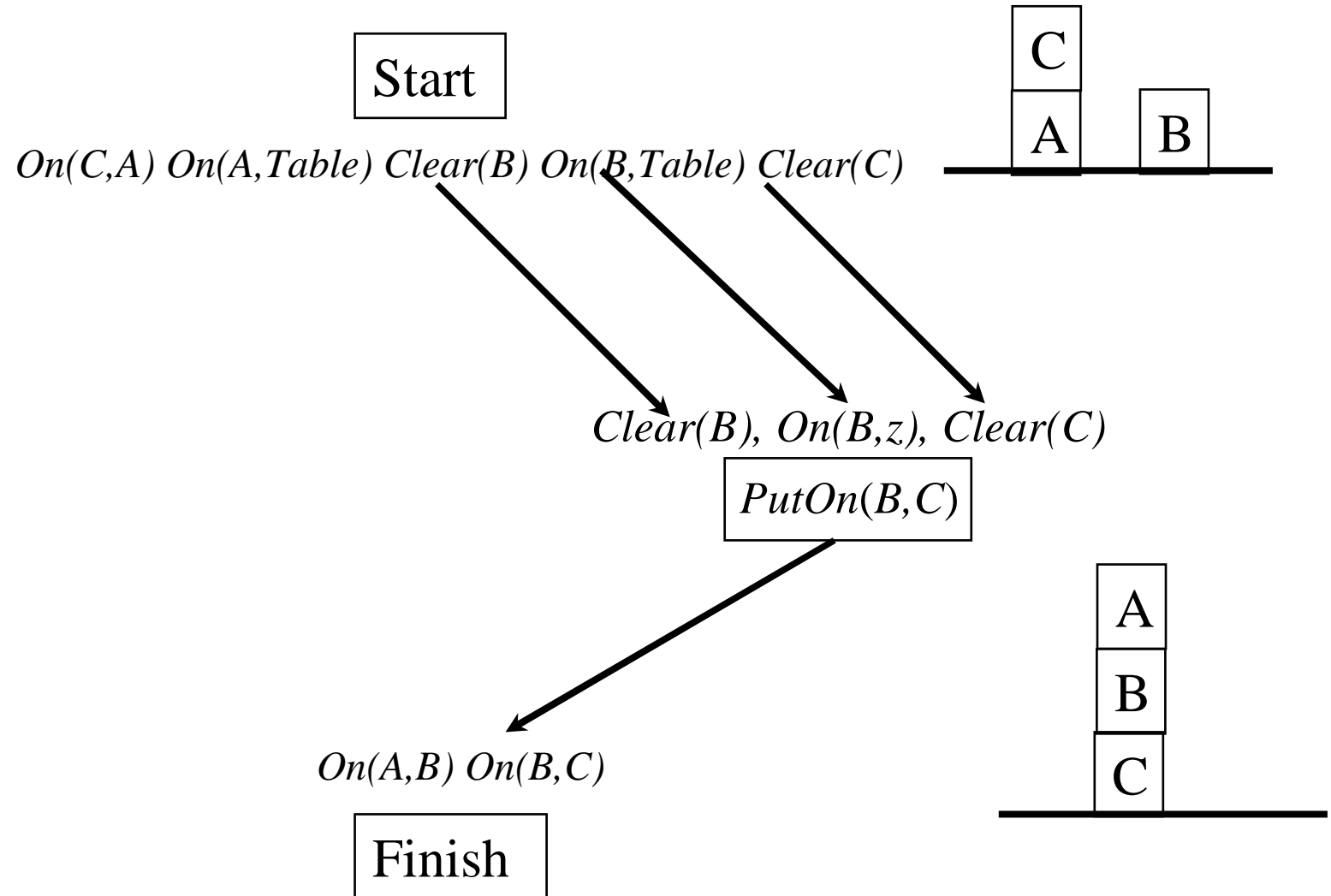
Finish





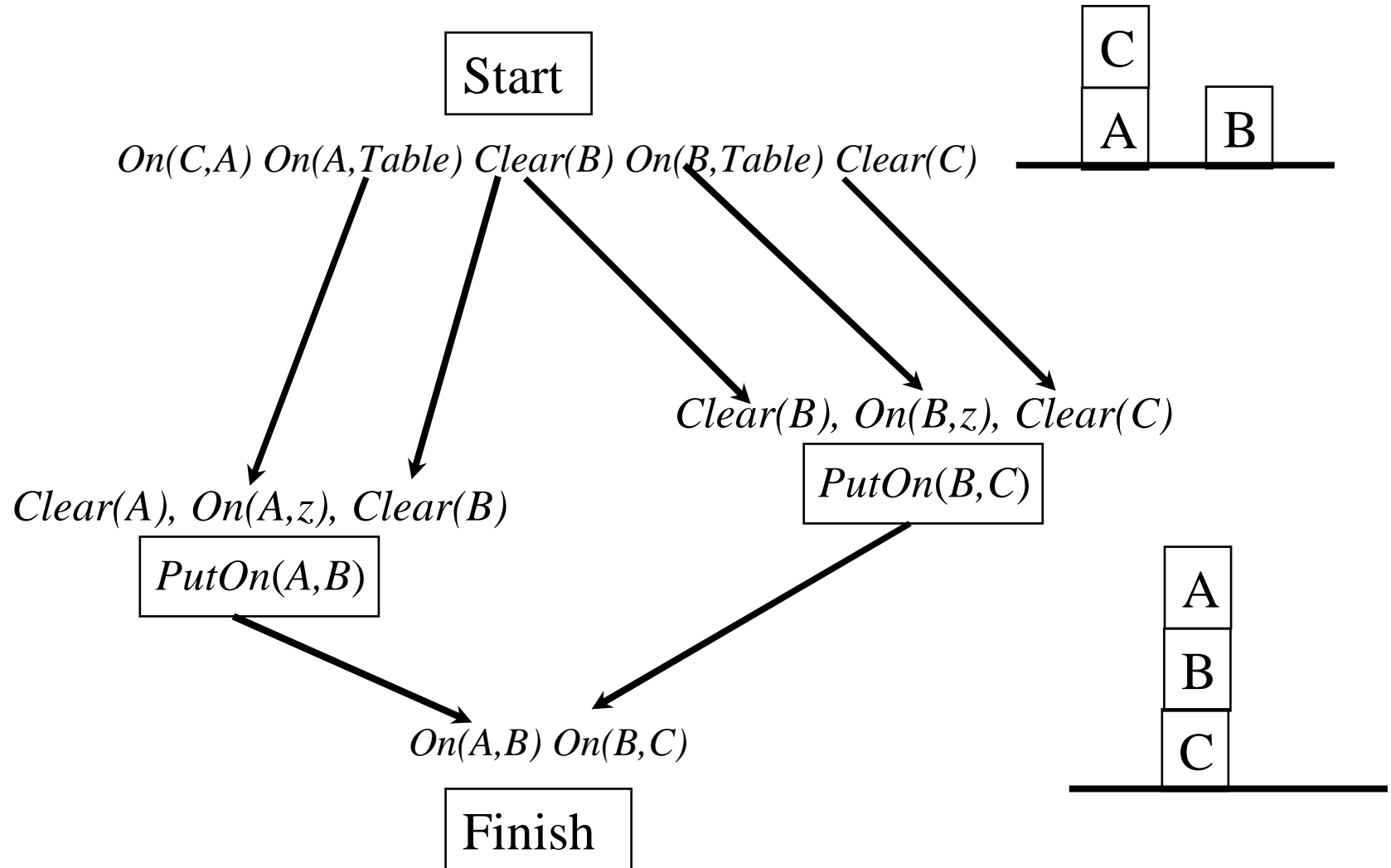
# Anomalia de Sussman

---

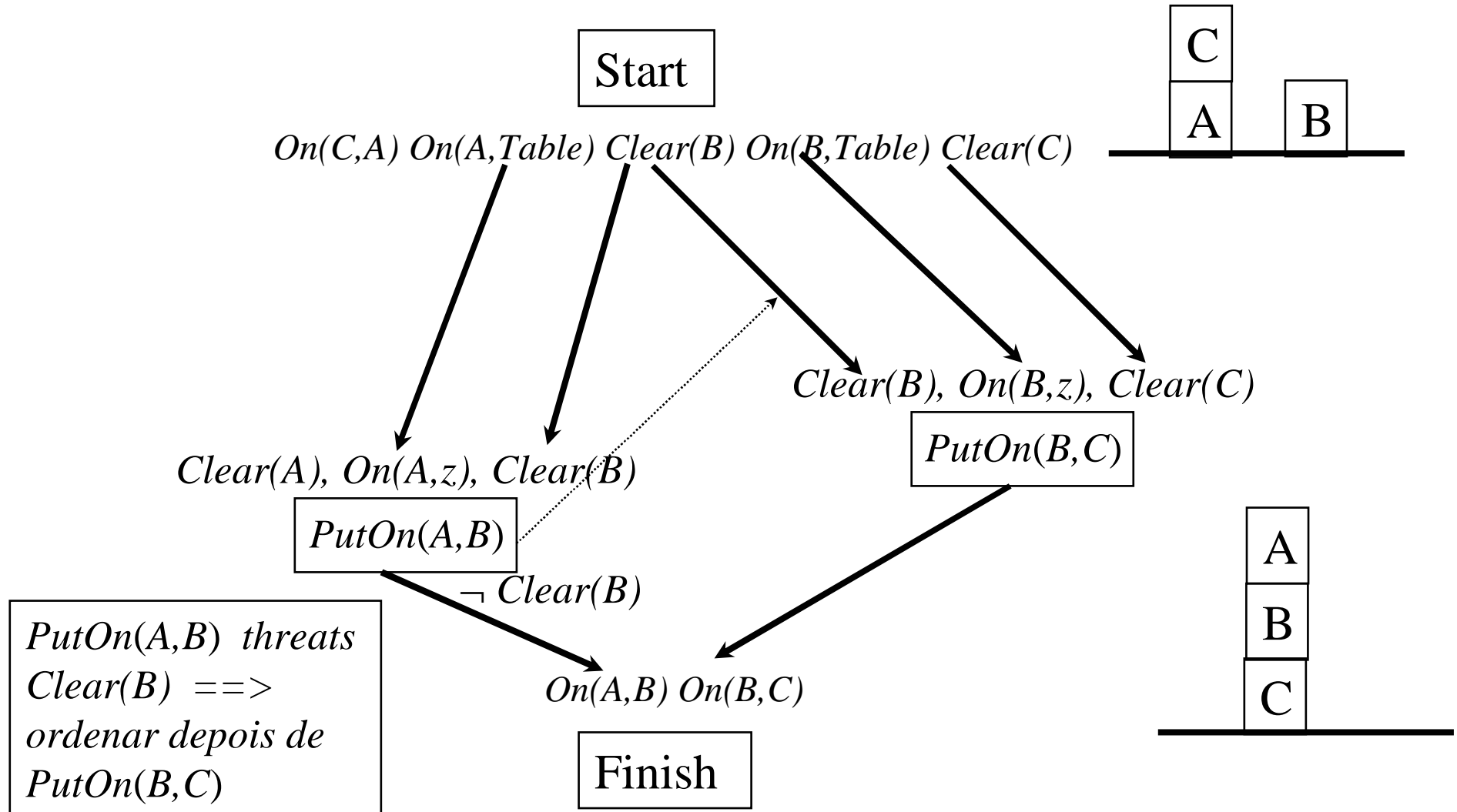


# Anomalia de Sussman

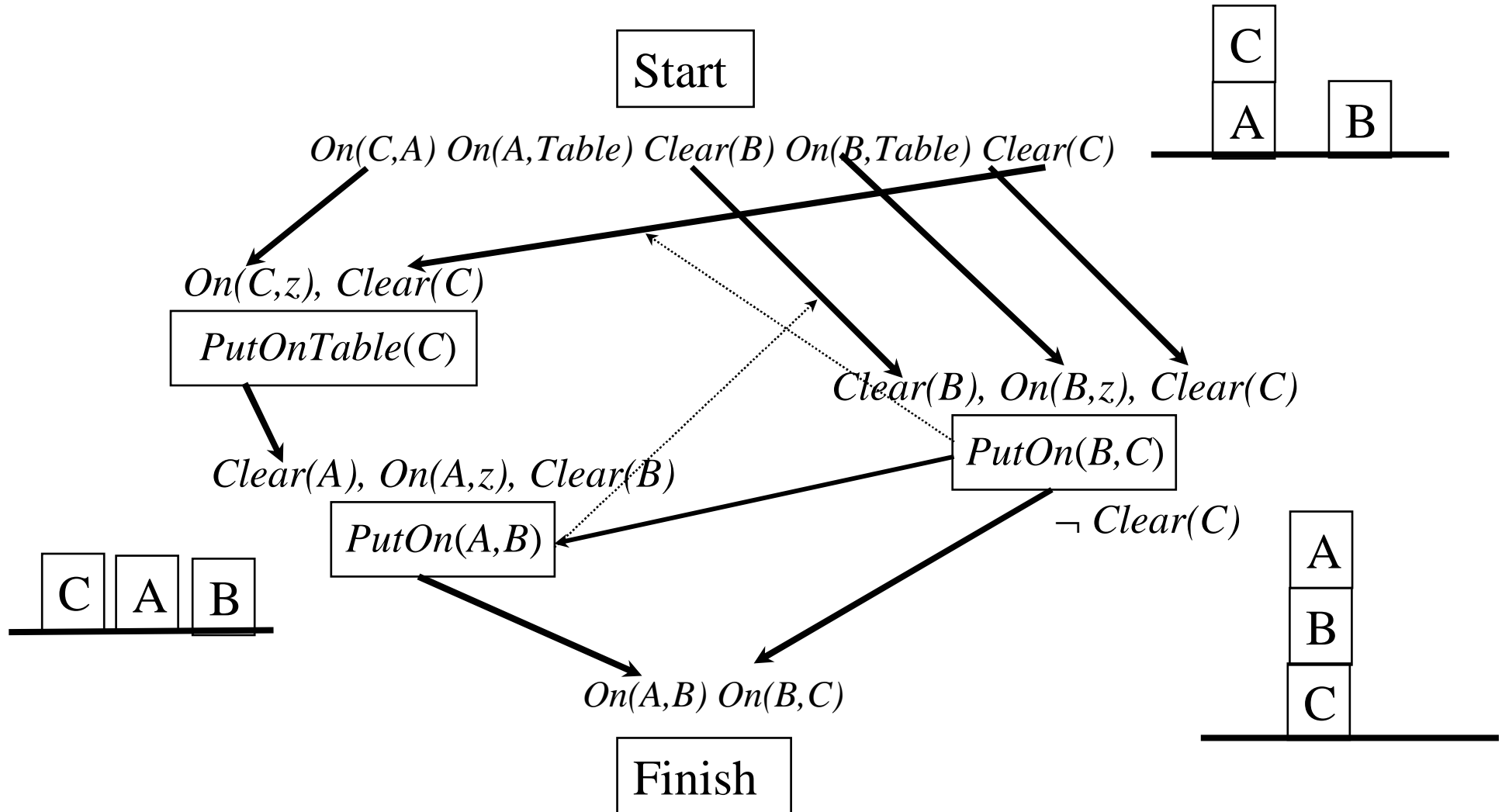
---



# Anomalia de Sussman



# Anomalia de Sussman



# Anomalia de Sussman

