Planejamento em Inteligência Artificial

planning is the reasoning side of acting

Leliane Nunes de Barros

Planos e Planejamento

- *Planejamento é o processo de escolha e organização de ações através da antecipação (previsão) de seus efeitos. Esse processo de raciocínio tem o objetivo de satisfazer (através da execução de ações), alguns objetivos previamente estabelecidas.*

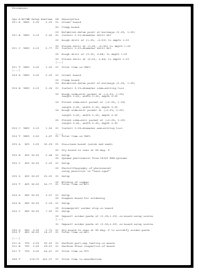
- *Planejamento automático é a sub-área da IA que estuda esse processo de raciocínio, usando o computador.*

***Aplicação**: sistemas que exigem comportamento autônomo (tomada de decisão) e deliberativo.*

Planejamento

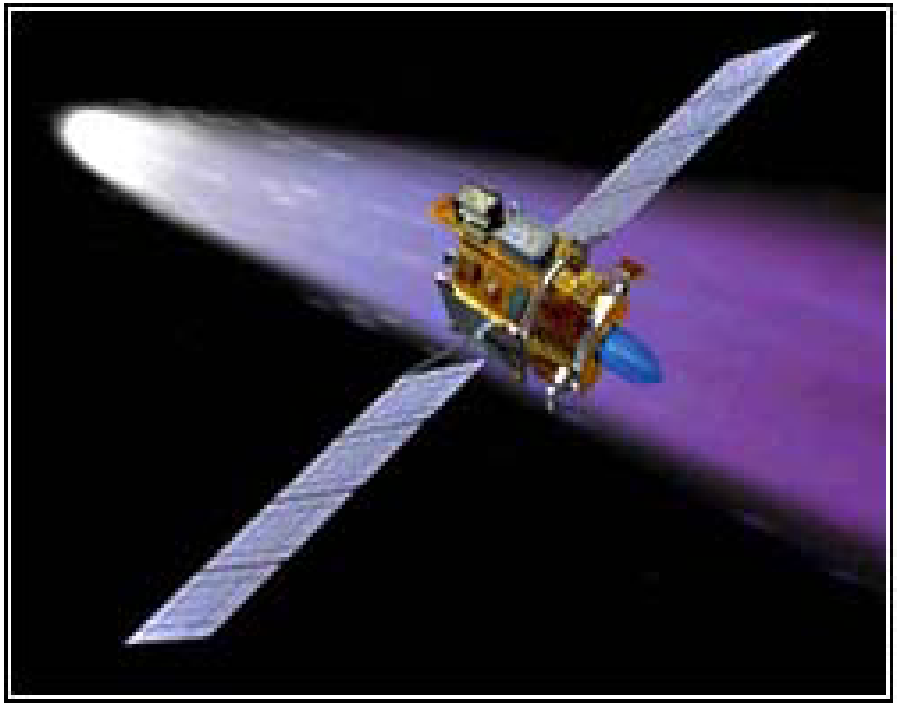
- Existem vários programas para ajudar planejadores humanos
  - Gerenciamento de projeto, armazenamento/recuperação de planos, geração automática de escalonamento com iniciativa mixta

- Geração automática de planos é muito difícil!
  - Existem muitos protótipos de pesquisa, poucos sistemas práticos (usados em aplicações reais)
  - Início da área de pesquisa: ~1970
  - Pesquisa começa a dar retorno: ~1995
    - » Exemplos de sucesso em problemas práticos e difíceis



NASA Unmanned Spacecraft

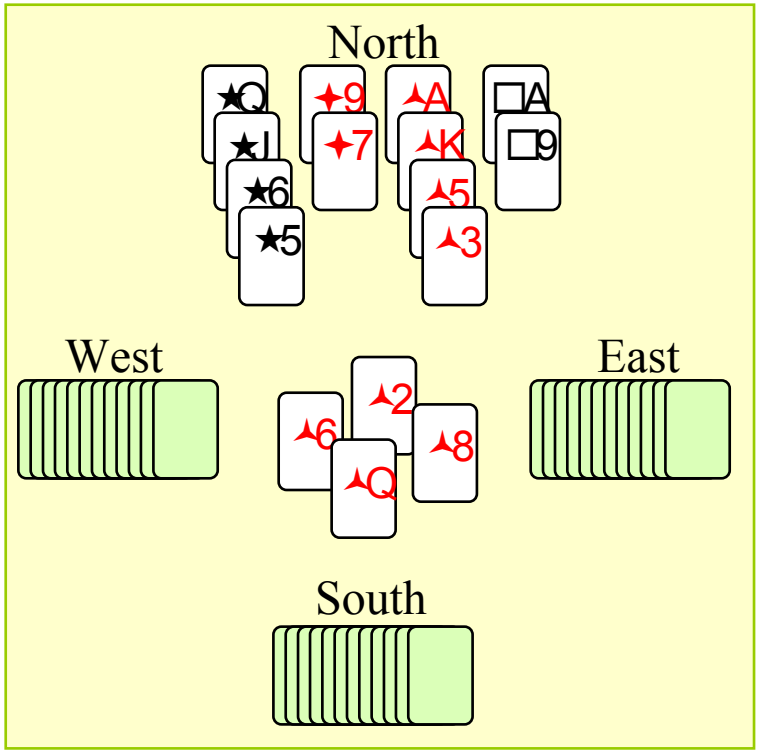
- *Remote Agent eXperiment* (RAX)
  - Software autônomo de planejamento/controle de IA
  - Foi usado na espaçonave DS1 em Maio de 1998
  - A espaçonave foi controlada por vários minutos pelo RAX



- Veículo de exploração (*rover*) em Marte
  - Guiado por um software autônomo de planejamento/controle de Inteligência Artificial

Outros Exemplos

- *Computer bridge*: *Bridge Baron*
  - Usou um sistema de planejamento hierárquico para ganhar o campeonato mundial de 1997 de bridge
  - Software comercial: já vendeu milhares de cópias



- Planejamento de processo de manufatura
  - É usado para planejar operações de moldagem de chapas metálicas (*bending)* na indústria automotiva.

Formas conhecidas de Planejamento

Planejamento de caminho e movimentação:

- definição de um caminho (geométrico) de uma posição inicial a uma posição meta + o controle de um sistema móvel (robôs móveis, veículos, braços mecânicos).

Planejamento de percepção:

- geração de planos envolvendo ações de sensoriamento:
  - » Que informação é necessária? Quando ela é necessária? Qual é o sensor mais adequado para uma dada tarefa? Como usar a informação?

Planejamento de navegação:

- combinação de planejamento de movimentação com percepção (movimentar evitando obstáculos, seguir um caminho até encontrar um marcação)

Planejamento de manipulação:

- problemas de manipulação de objetos para construção e montagem.

Formas conhecidas de Planejamento

Planejamento de manipulação:

- problemas de manipulação de objetos para construção e montagem.

Planejamento para recuperação de informação:

- o ambiente é um banco de dados ou a WWW (construção automática de consultas)

Planejamento de comunicação:

- construção de diálogos em problemas de cooperação entre vários agentes, humanos ou artificiais. Ex: planejamento instrucional em sistemas de educação, tele-atendimento.

Planejamento para jogos:

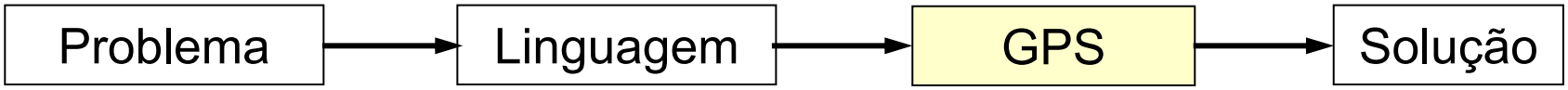
- geração de planos de ações para vencer um jogo (adversarial ou não)

Planejamento em ambientes com incerteza:

- Escolha da melhor ações para cada estado possível do mundo, levando em conta a incerteza dos próximos estados (tomada de decisão sequencial em ambientes não-determinísticos)

Planejamento: motivação

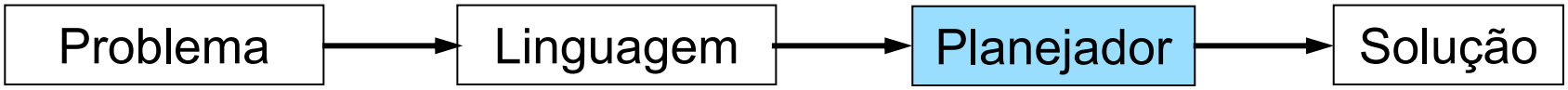
- Um dos principais objetivos da IA foi/é o desenvolvimento de um ***Resolvedor Geral de Problemas (General Problem Solver)*** [Newell & Simon, 1961]



- **Idéia**: problemas são descritos numa linguagem de alto-nível de abstração e são resolvidos automaticamente
- **Objetivo**: facilitar a modelagem de problemas com um prejuízo mínimo em termos de desempenho.

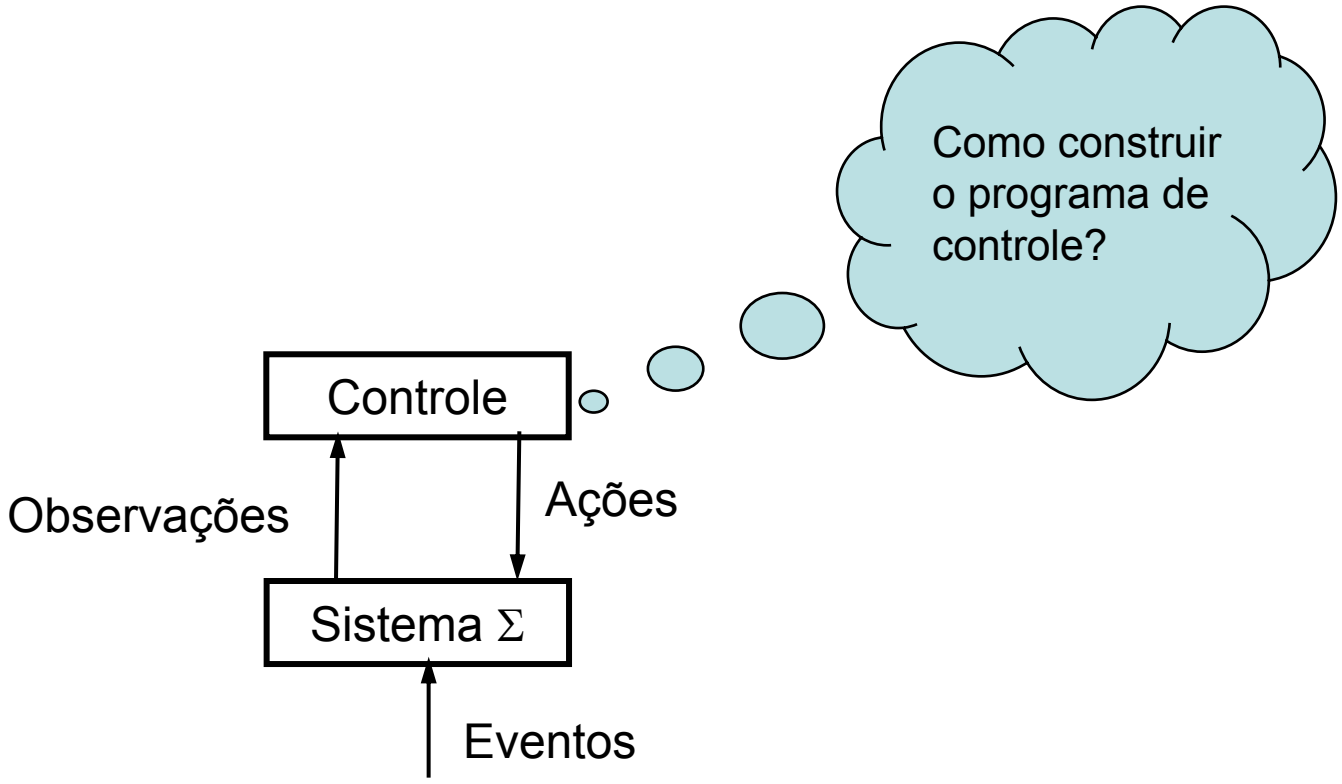
Planejamento: motivação

- Um dos principais objetivos da IA foi/é o desenvolvimento de um ***Resolvedor Geral de Problemas (General Problem Solver)*** [Newell & Simon, 1961]

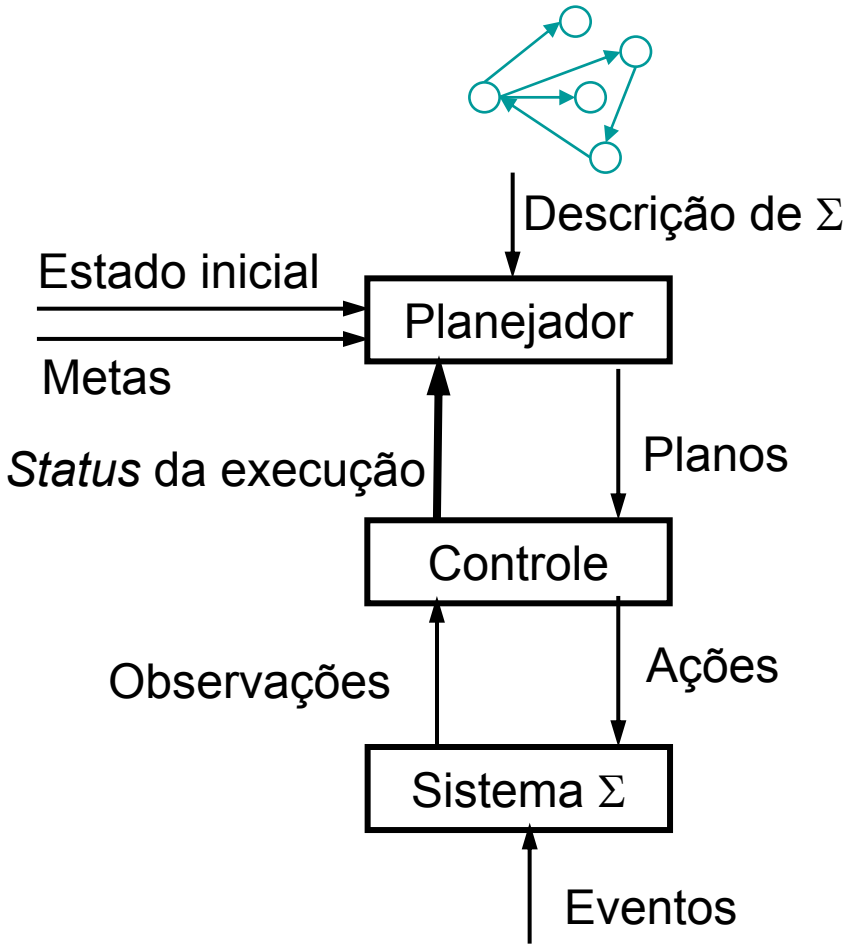


- **Idéia**: problemas são descritos numa linguagem de alto-nível de abstração e são resolvidos automaticamente
- **Objetivo**: facilitar a modelagem de problemas com um prejuízo mínimo em termos de desempenho.

Modelo conceitual de um agente planejador



Modelo conceitual de um agente planejador

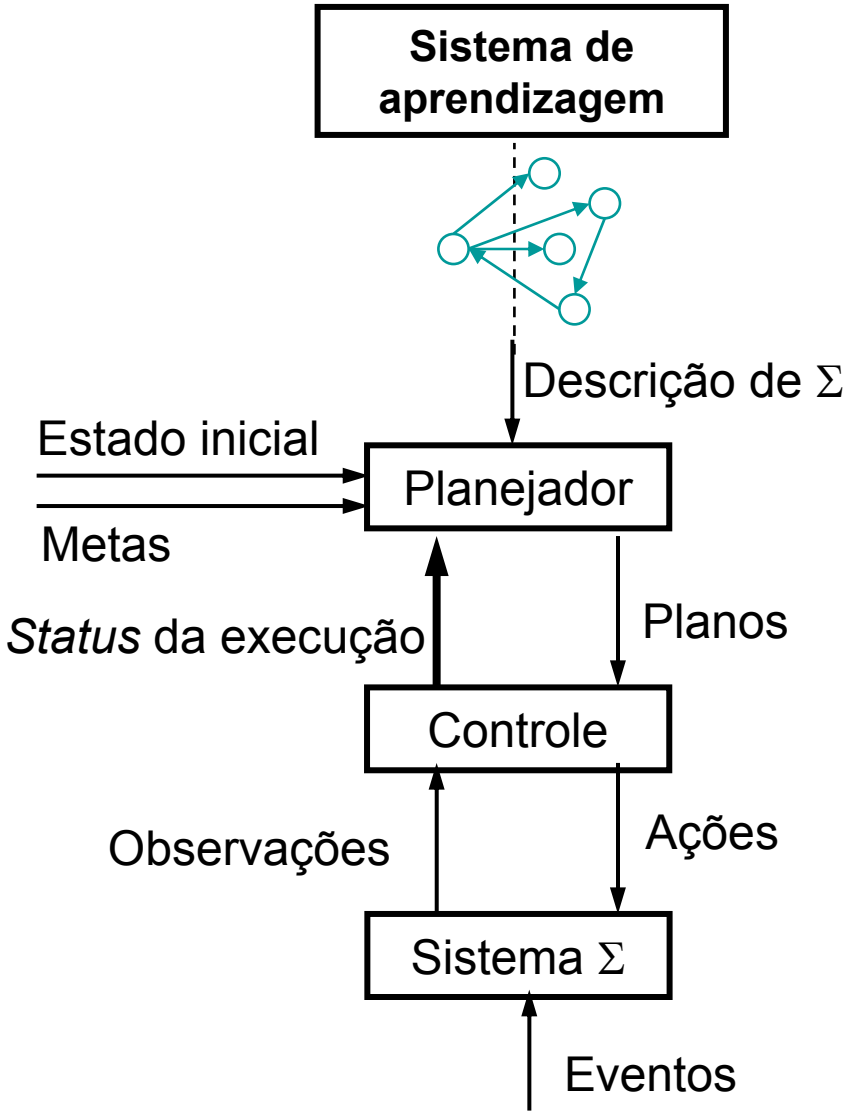


Construção de programas de controle

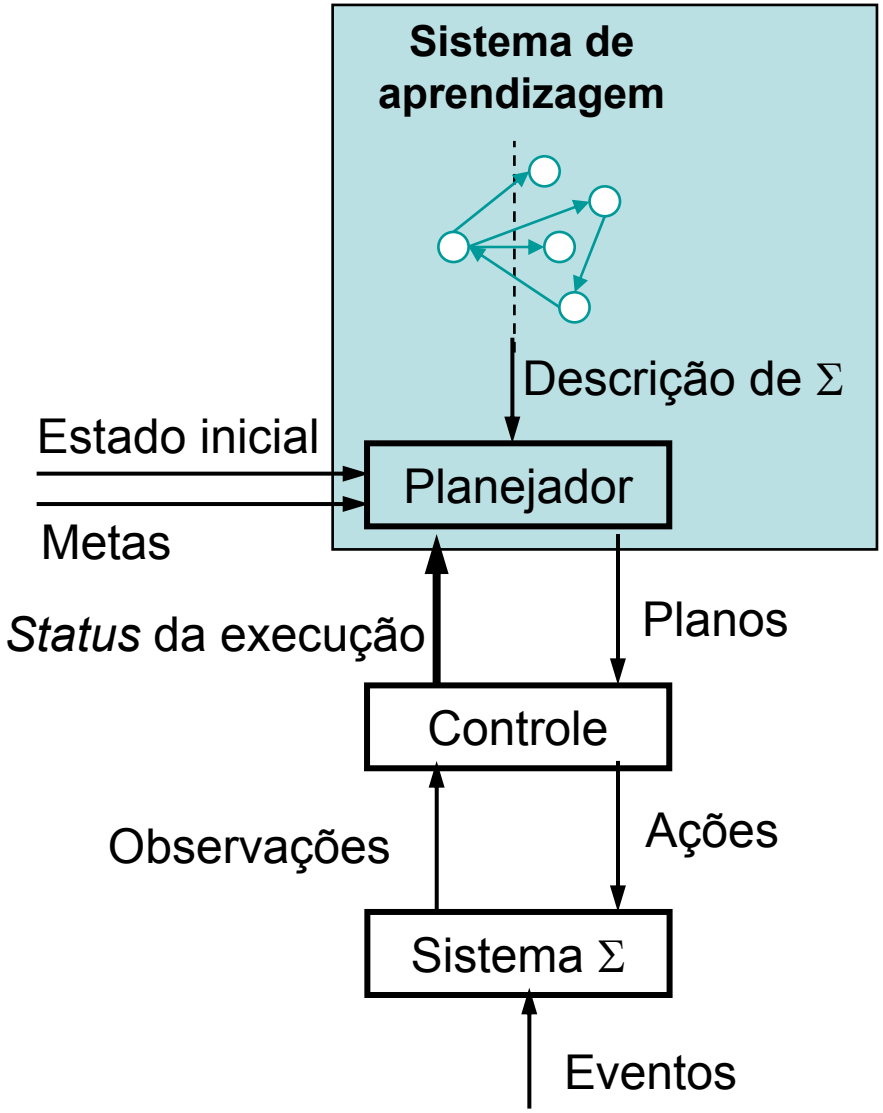
- Existem três abordagens:
  1. Programação: **especificação de controle** pelo projetista. Os problemas não são especificados formalmente mas estão na cabeça do programador (modelo de Σ está implícito no programa).
  2. Planejamento: **especificação do problema** pelo projetista em uma linguagem formal (modelo de Σ é conhecido); o programa de controle é derivado automaticamente (com ou sem sensoriamento).
  3. Aprendizado: o modelo ou o controle é "aprendido" através da experiência (modelo de Σ é desconhecido).

- As três abordagens são consideradas não exclusivas e em geral, se complementam. Possuem seus pontos fortes e fracos.

Aprendizagem sobre a construção de programas



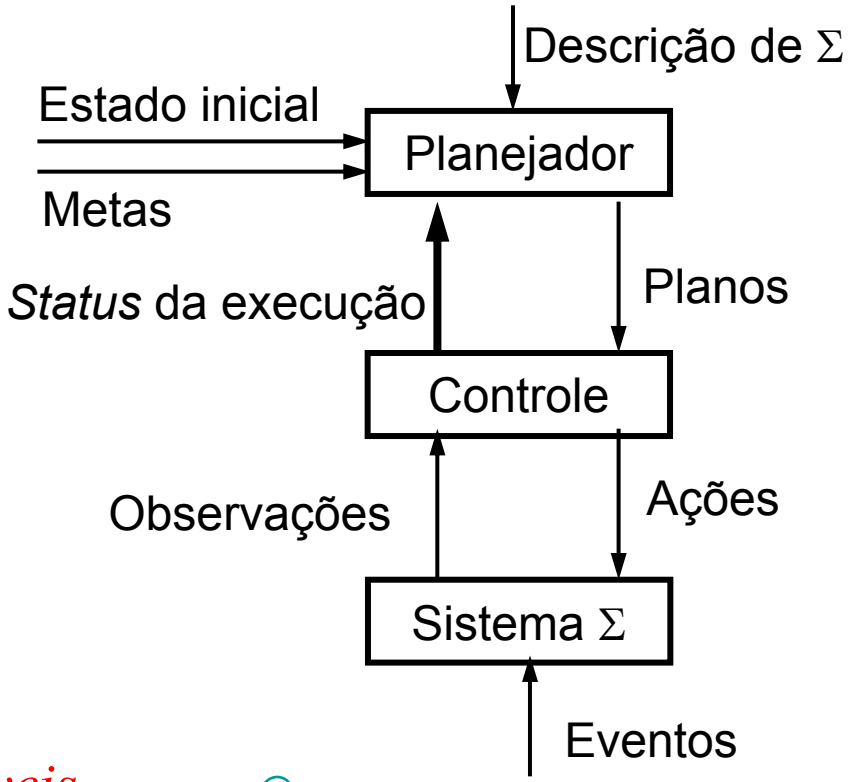
Aprendizagem sobre a construção de programas (visão integrada)



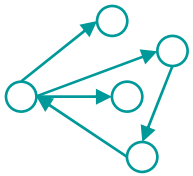
Tópicos da aula

- Modelo conceitual de planejamento
- Suposições restritivas
- Planejamento classico
- Relaxando suposições
- Example: Robôs de cais de porto (carga e descarga de navios)
- Representação de ações
- O algoritmos POP

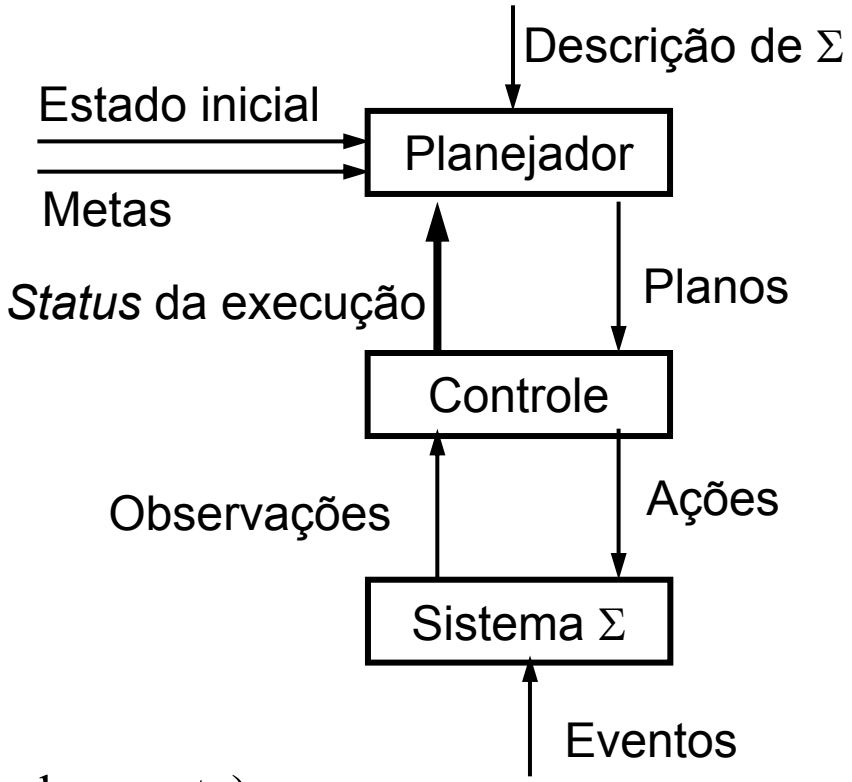
Modelo Conceitual



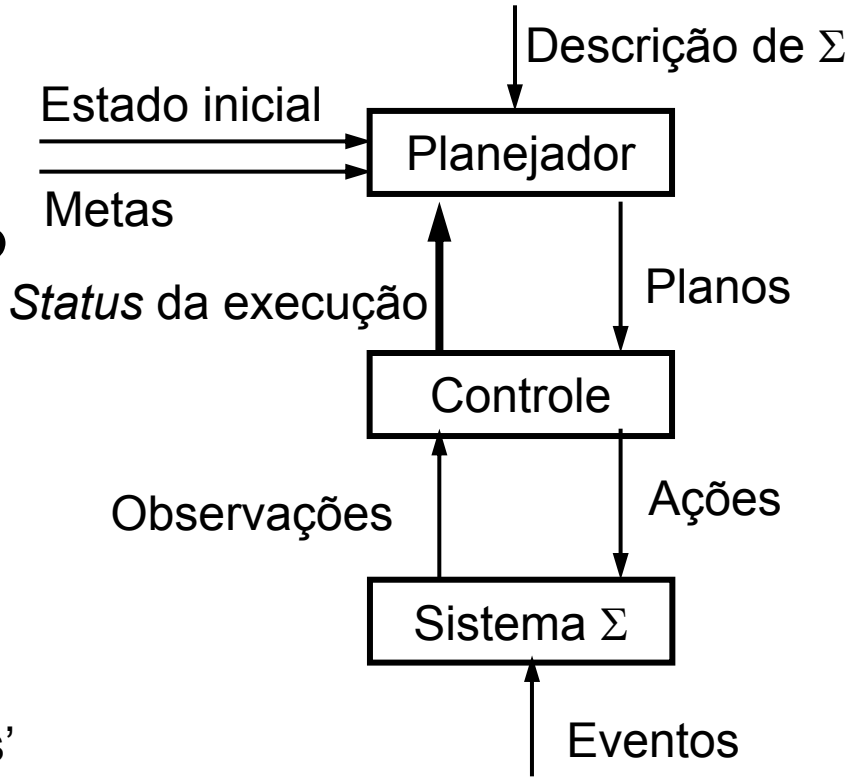
- **Ingredientes:**
  - Modelo do ambiente: *estados possíveis*
  - Modelo de como o ambiente pode mudar: *ações e seus efeitos*



  - Especificação de *condições iniciais* e *metas*
  - *Planos de ações* que são gerados pelo planejador
  - Um modelo de execução de um plano no ambiente
  - Um modelo de observação do ambiente



- Sistema de transição de estado
  $\Sigma = (S,A,E,\gamma)$
  - $S = \{s_1, s_2, \ldots\} = \{\text{estados}\}$
  - $A = \{a_1, a_2, \ldots\} = \{\text{ações}\}$ (controladas pelo agente)
  - $E = \{e_1, e_2, \ldots\} = \{\text{eventos exógenos}\}$ (não controladas pelo agente)
  - Função de transição de estado
    $\gamma: S \times (A \cup E) \rightarrow 2^S$

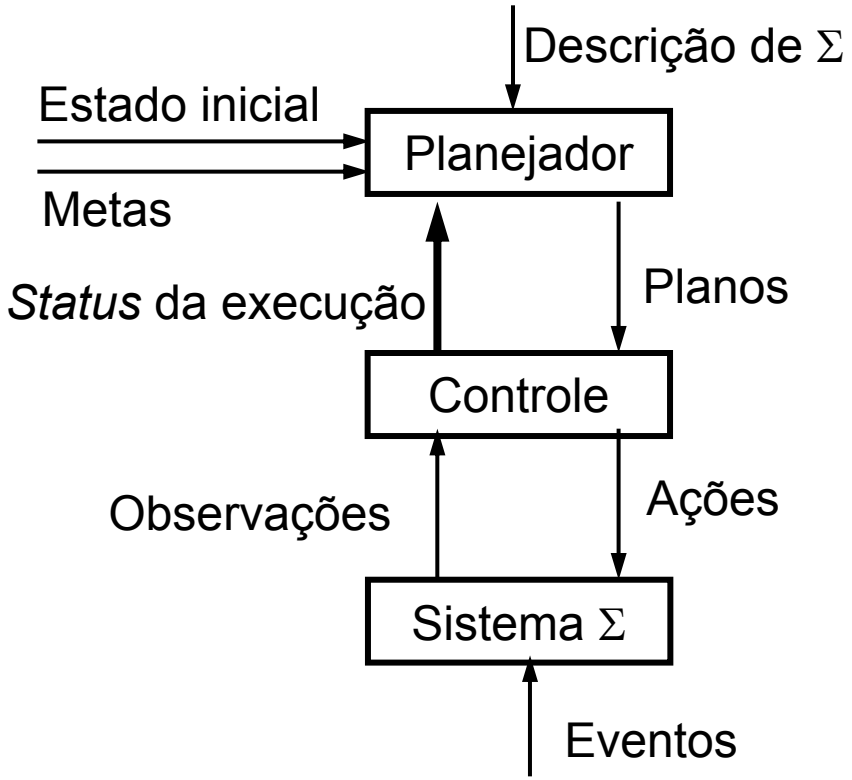


- Um sistema Σ do tipo estado-transição pode ser representado por um grafo dirigido cujos nós são estados em S. Se s' $\in \gamma(s,u)$, em que u é o par (a,e); $a \in A$ e $e \in E$, então o grafo contém um arco (chamado de transição de estado) de s a s', rotulado com u :

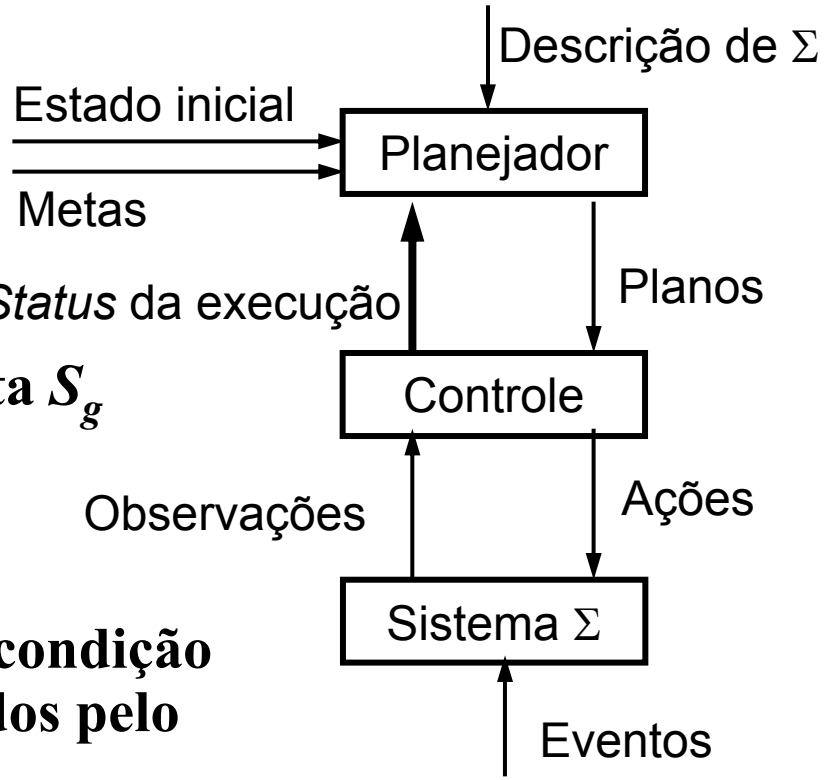
s $\xrightarrow{\text{u}}$ s'

- Se a é uma ação aplicável no estado s, aplicá-la em s leva o agente a um outro estado $\gamma(s,a)$. **O sistema evolui através de eventos e ações.**

ϵ é um evento neutro => $\gamma(s,a,\epsilon) = \gamma(s,a)$
no-op é uma ação neutra => $\gamma(s,no\text{-}op,e) = \gamma(s,e)$



- Função de observação h: $S \rightarrow O$
  - produz observação o sobre o estado atual s
- Controle: dada a observação $o \in O$, produz ação $a \in A$
- Planejador:
  - entrada: descrição de Σ, estado inicial $s_0 \in S$ e alguma meta
  - saída: produz um plano para guiar o controle



Diferente tipos de metas:

- **Alcançar um conjunto de estados meta S_g**

  Encontre uma seqüência de transição de estados terminando em um estado meta

- **Alcançar metas satisfazendo alguma condição sobre o conjunto de estados percorridos pelo sistema (metas extendidas)**

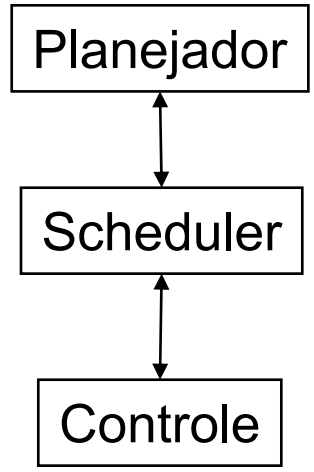
  Atinja S_g , passando por estados que valha r, e permaneça nele

- **Otimização de uma função utilidade (ou recompensa) relacionada aos estados**

  Otimize alguma função utilidade (eventualmente terminando num estado de S_g)

- **Execução de tarefas (planos abstratos), especificadas recursivamente como conjuntos de sub-tarefas e ações**

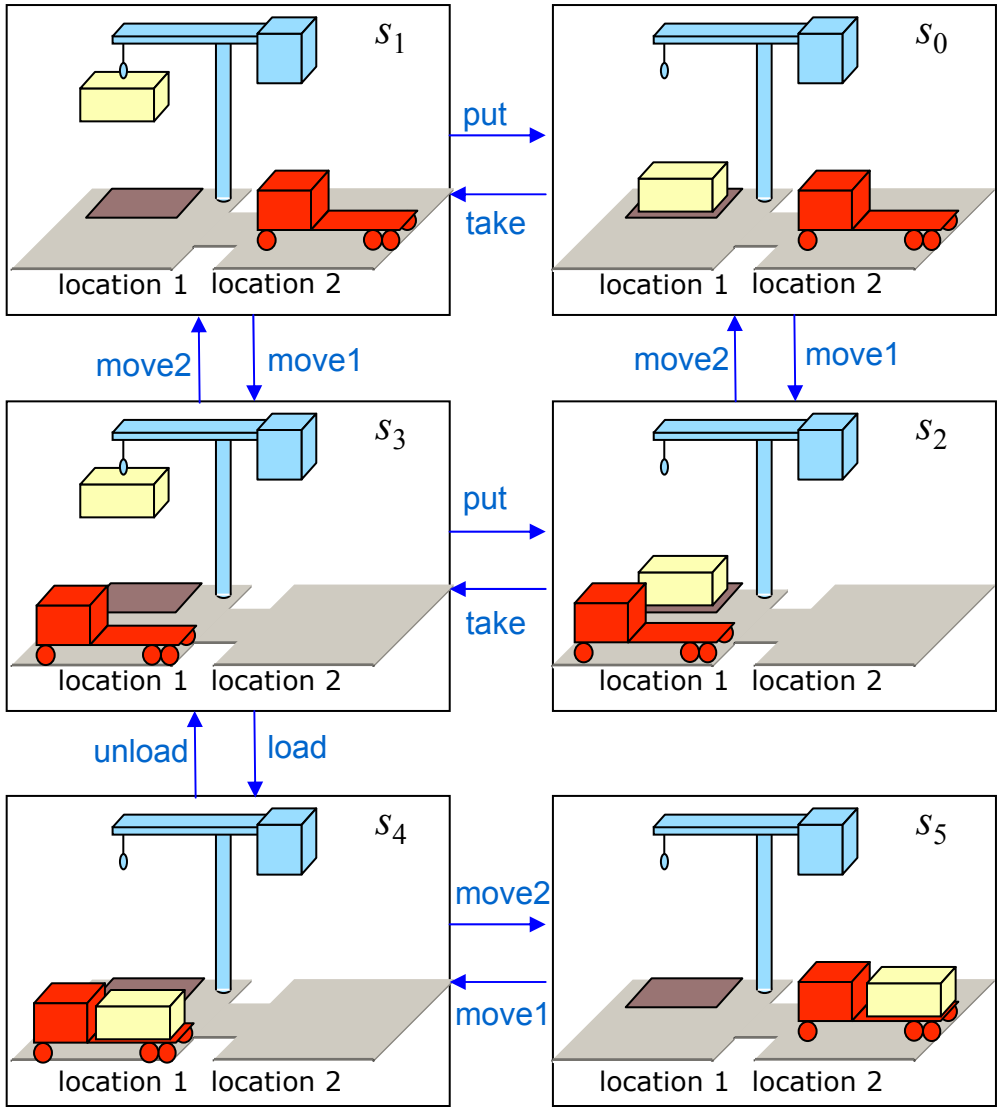
Planejamento Versus Escalonamento (*Scheduling*)



- Escalonamento
  - Decide **como** executar um dado conjunto de ações usando um número limitado de recursos em um intervalo de tempo limitado, com eventual data de término
  - É tipicamente NP-completo
- Planejamento
  - Decide **quais** ações usar para atingir um conjunto de metas
  - Pode ser muito pior que NP-completo
    - » Na maioria dos casos, é não-decidível
    - » Muitas pesquisas fazem um conjunto de suposições restritivas para garantir a decidibilidade
    - » Vamos ver algumas dessas restrições = == ====>

Exemplo do Robô do Cais de Porto

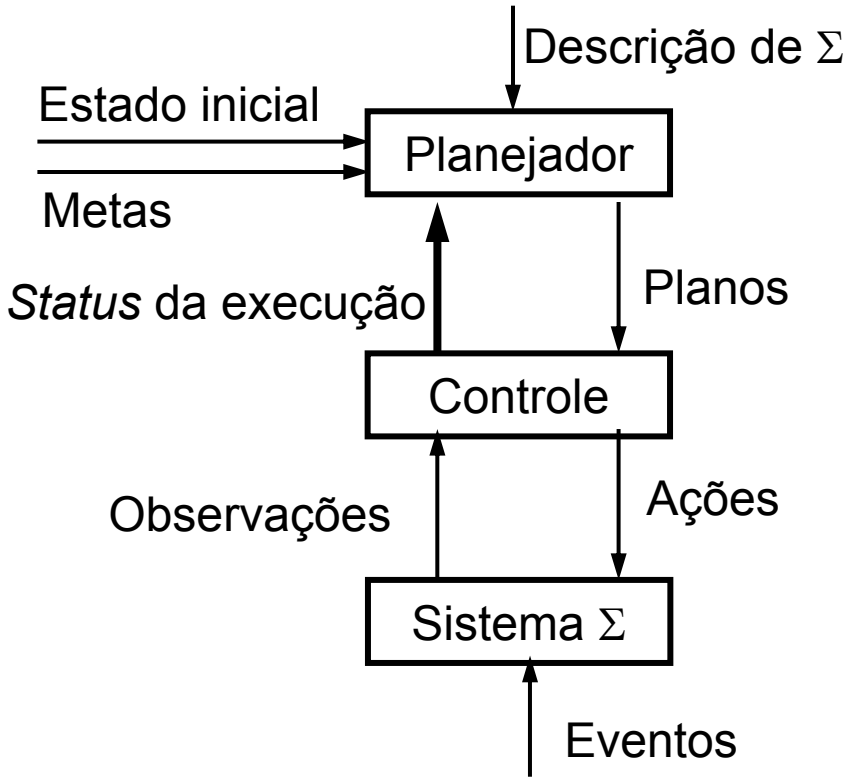
- Sistema de transição de estado $\Sigma = (S,A,E,\gamma)$
  - $S = \{s_0, \ldots, s_5\}$
  - $A = \{$move1, move2, put, take, load, unload$\}$
  - $E = \{\}$
  - γ: como ilustrado na figura
- $h(s) = s$ para todo s
- Entrada do planejador:
  - sistema Σ
  - Estado inicial s_0
  - Estado meta s_5



Suposições restritivas

- **A0 (Σ finito):**
  - O espaço de estados S é finito
  - $S = \{s_0, s_1, s_2, \ldots s_k\}$ para algum k

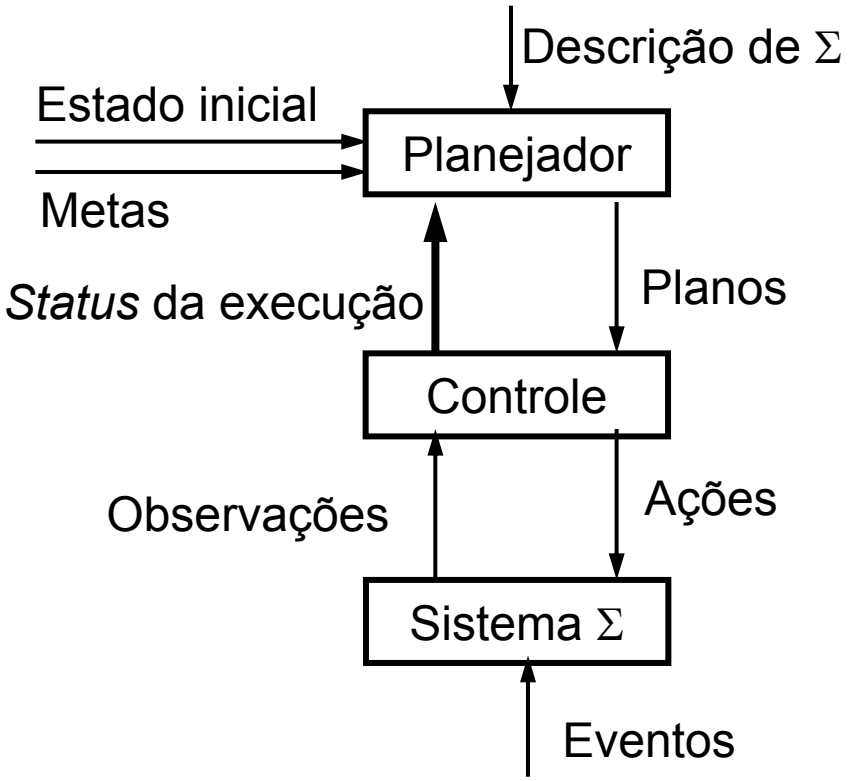
- **A1 (Σ totalmente observável):**
  - A função de observação $h: S \rightarrow O$ é a função identidade
  - i.e., o controle sempre sabe em que estado o sistema está.



$$\Sigma = (S,A,E,\gamma)$$
$$S = \{\text{estados}\}$$
$$A = \{\text{ações}\}$$
$$E = \{\text{eventos}\}$$
$$\gamma: S \times (A \cup E) \rightarrow 2^S$$

Suposições restritivas

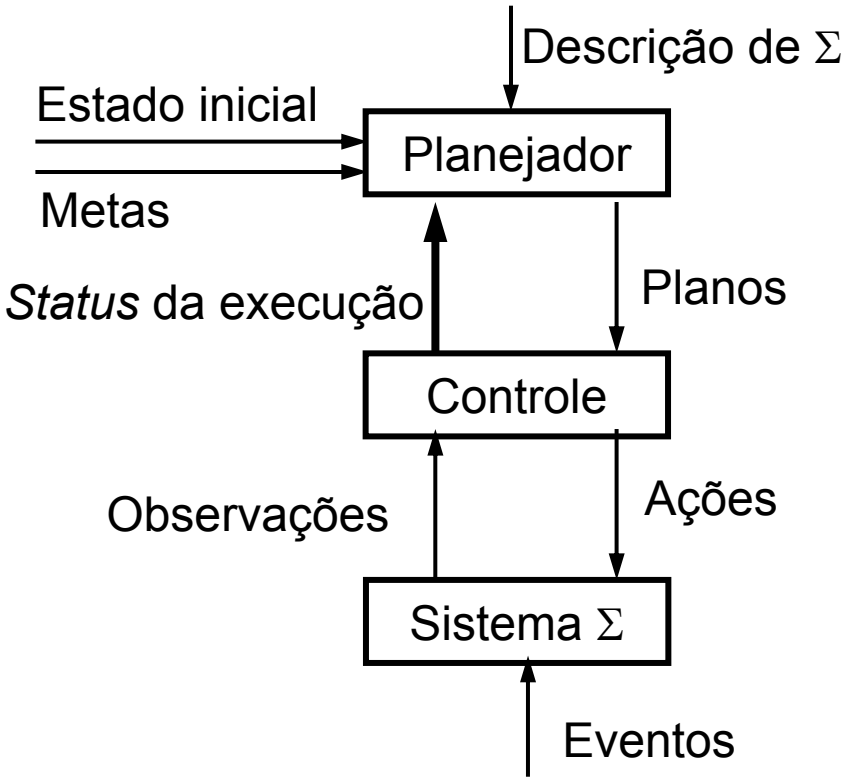
- **A2 (Σ determinístico):**
  - Para todo u em $A \cup E$, $|\gamma(s,u)| = 1$
  - Cada ação ou evento tem apenas um saída possível
- **A3 (Σ estástico):**
  - E é vazio: nenhuma mudança ocorre exceto aquelas efetuadas pelo controle (a $\in$ A)
- **A4 (satisfação de metas = *attainment goals*):**
  - Um estado meta s_g ou um conjunto de estados meta S_g



$$\Sigma = (S,A,E,\gamma)$$
$$S = \{\text{estados}\}$$
$$A = \{\text{ações}\}$$
$$E = \{\text{eventos}\}$$
$$\gamma: S \times (A \cup E) \rightarrow 2^S$$

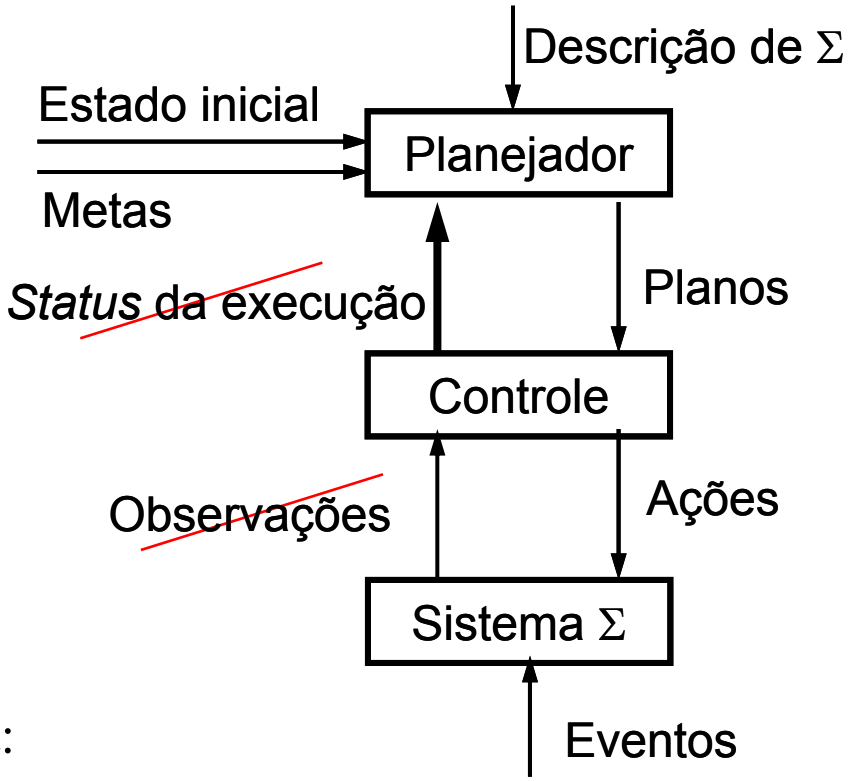
Suposições restritivas

- **A5 (planos sequenciais)**:
  - A solução é uma seqüência de ações totalmente ordenada $(a_1, a_2, \ldots a_n)$
- **A6 (tempo implícito)**:
  - Transições de estados (ações) instantâneas, ações sem duração de tempo
- **A7 (planejamento *off-line*)**:
  - Planejador não conhece o *status* da execução



$$\Sigma = (S,A,E,\gamma)$$
$$S = \{\text{estados}\}$$
$$A = \{\text{ações}\}$$
$$E = \{\text{eventos}\}$$
$$\gamma: S \times (A \cup E) \rightarrow 2^S$$

Planejamento Clássico



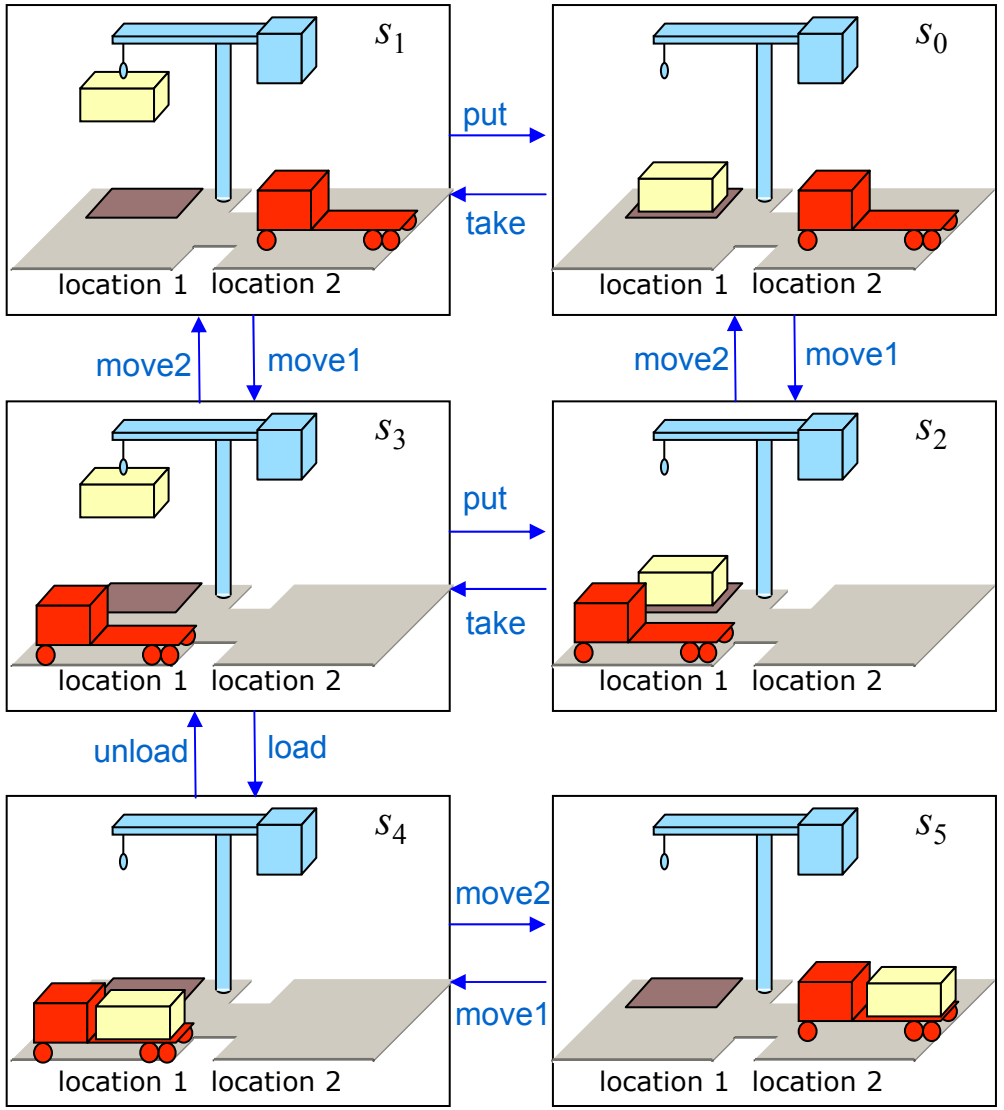
- Planejamento clássico faz as seguintes suposições:
  - *Conhecimento completo sobre um sistema determinístico, estático, estado-finito com satisfação de metas e tempo implícito*
- Planejamento se reduz ao seguinte problema:
  - Dado (Σ, s_0, S_g), encontre uma seqüência de ações $(a_1, a_2, \ldots a_n)$ que produza uma seqüência de transições de estados

$$s_1 = \gamma(s_0, a_1),$$
$$s_2 = \gamma(s_1, a_2),$$
$$\ldots,$$
$$s_n = \gamma(s_{n-1}, a_n)$$

tal que s_n pertença ao conjunto de estados meta S_g.

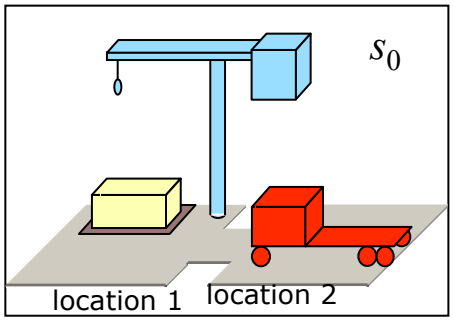
Planejamento Clássico: Exemplo

- Exemplo dos Robôs de Porto:
  - sistema finito, determinístico, estático
  - conhecimento completo
  - metas de satisfação
  - tempo implícito
  - planejamento *offline*
- Planejamento clássico é basicamente uma busca de caminho em um grafo
  - estados são nós
  - ações são arestas
- Esse é um problema trivial?



Planejamento Clássico

- Computacionalmente muito difícil
  - generalização do exemplo dos Robôs de Porto:
    - » 5 localizações, 3 pilhas, 3 robôs, 100 *containers*
  - isso implica em 10^{277} estados
    - » mais do que 10^{190} vezes o número de particulas no universo!
    - » é preciso encontrar formas de resolver o problema sem a enumeração explícita dos estados (grafo)

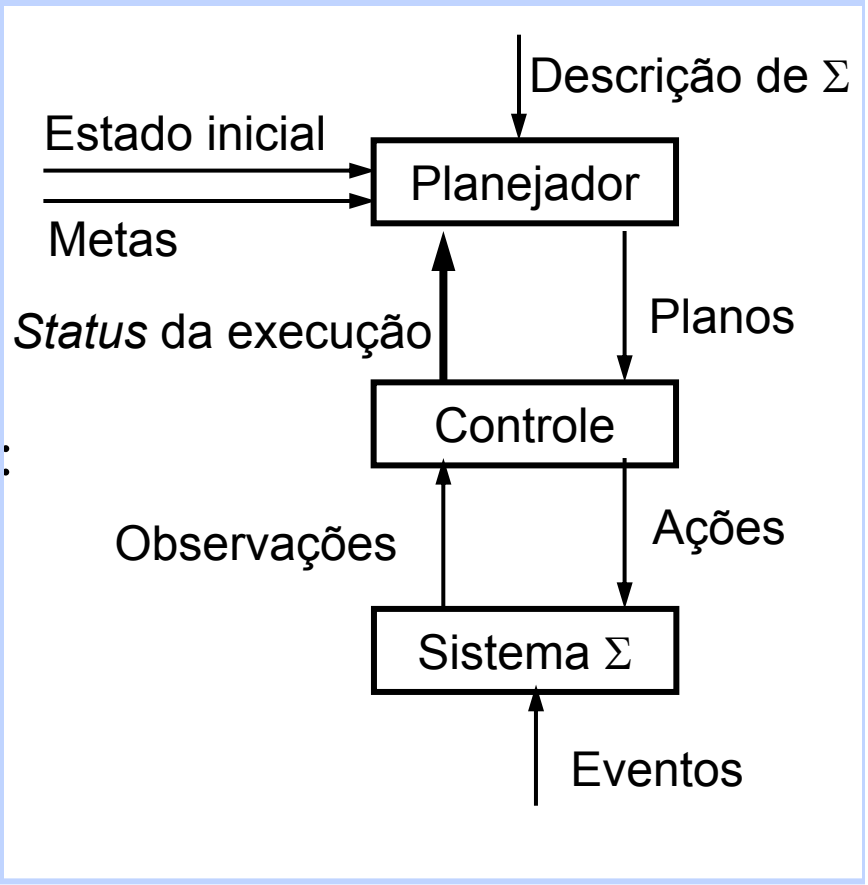


- A grande maioria das pesquisas tem sido sobre planejamento clássico

- Abordagem muito restritiva para tratar a maioria dos problemas de interesse prático
  - *No entanto, muitas das idéias de soluções do planejamento clássico têm sido úteis na resolução de problemas práticos*

Relaxando as Suposições

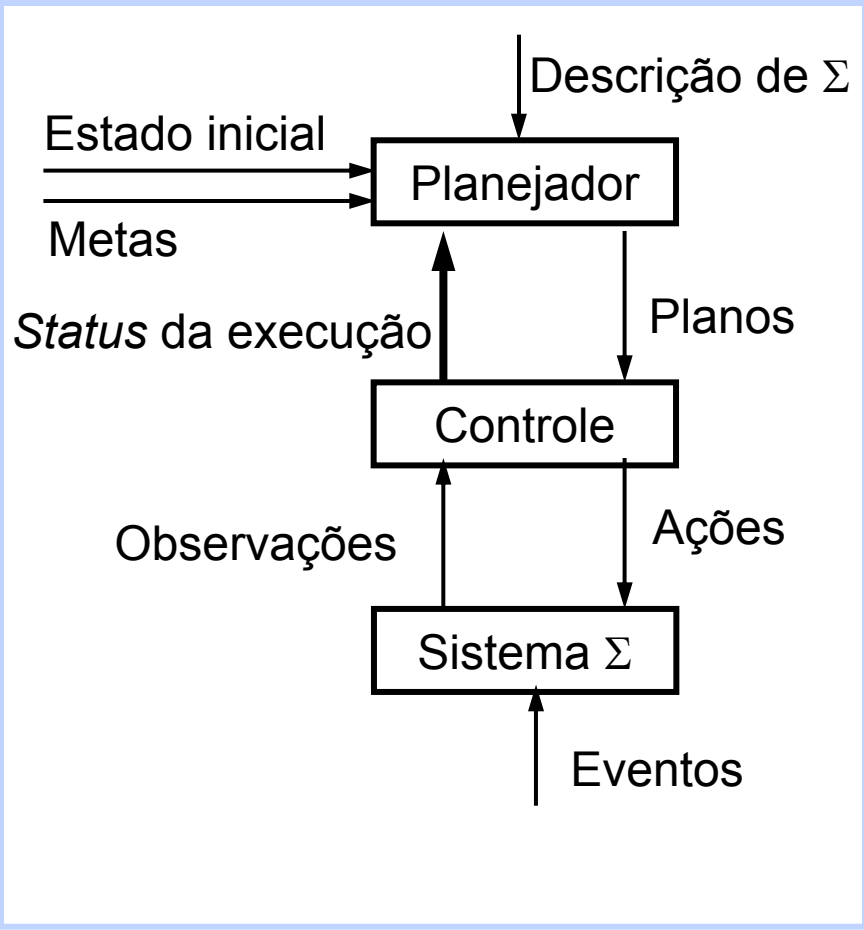
- Relaxar A0 (Σ finito):
  - Contínuo, *e.g.* ações com variáveis numéricas
- Relaxar A1 (Σ totalmente observável):
  - Se não relaxarmos nenhuma outra restrição, então a única incerteza é sobre s_0



A0: Finito
A1: Totalmente observável
A2: Determinístico
A3: Estático
A4: Satisfação de metas
A5: Planos seqüenciais
A6: Tempo implícito
A7: Planejamento *off-line*

Relaxando as Suposições

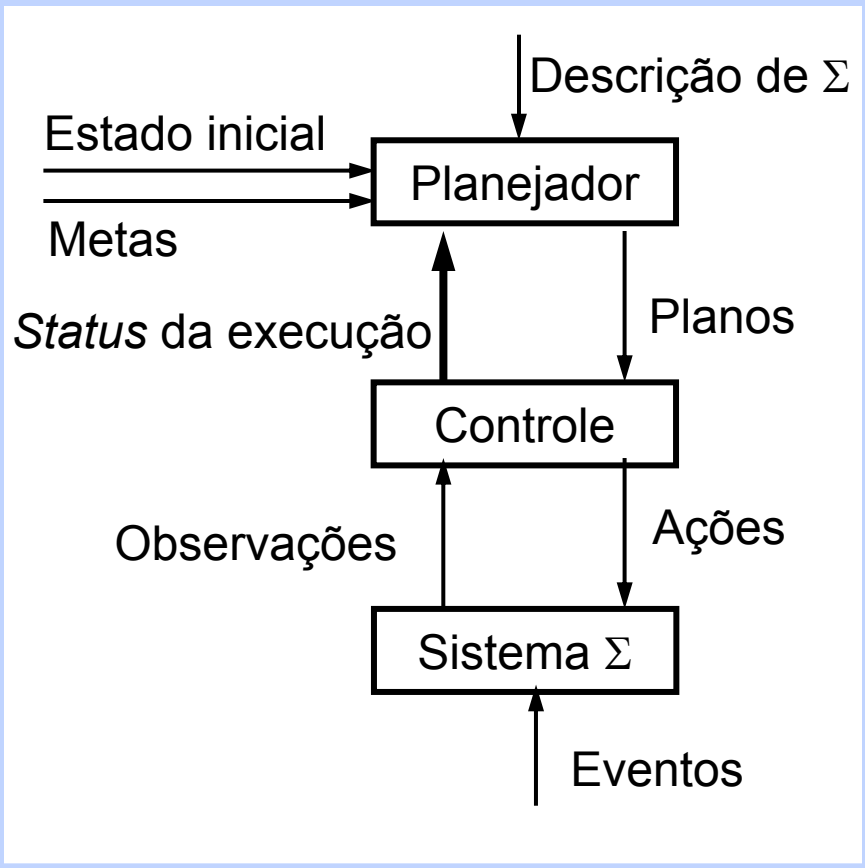
- Relaxar A2 (Σ determinístico):
  - Ações possuem mais do que 1 saída (efeito) possível. É preciso fazer busca por políticas ou planos de contingência
  - Ações probabilísticas:
    - » *Markov Decision Processes* (MDPs)
  - Ações não-determinísticas:
    - » Planejamento como verificação de modelos, ou planejamento condicional
  - Combinação de planejamento probabilístico e não determinístico
    - » MDP-IP e MDP-ST



A0: Finito
A1: Totalmente observável
A2: Determinístico
A3: Estático
A4: Satisfação de metas
A5: Planos seqüenciais
A6: Tempo implícito
A7: Planejamento *off-line*

Relaxando as Suposições

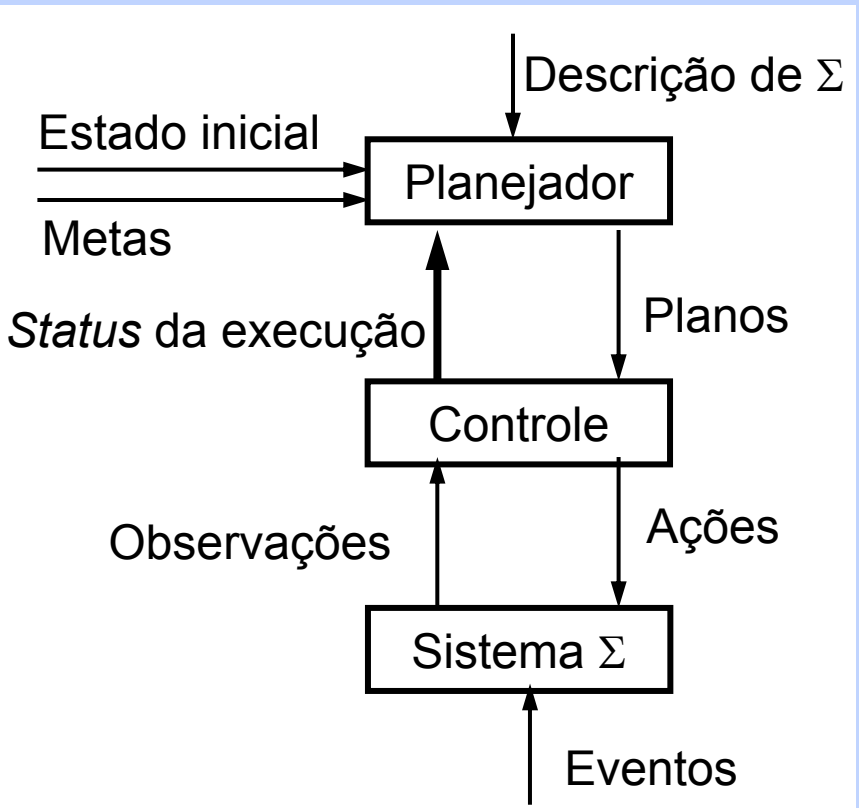
- Relaxar A1 e A2:
  - POMDPs finitos
    - » Planejar sobre *estados de crença*
    - » Tempo e espaço exponenciais
- Relaxar A0 e A2:
  - MDPs contínuos ou híbridos
    - » Teoria de controle da engenharia



A0: Finito
A1: Totalmente observável
A2: Determinístico
A3: Estático
A4: Satisfação de metas
A5: Planos seqüenciais
A6: Tempo implícito
A7: Planejamento *off-line*

Relaxando as Suposições

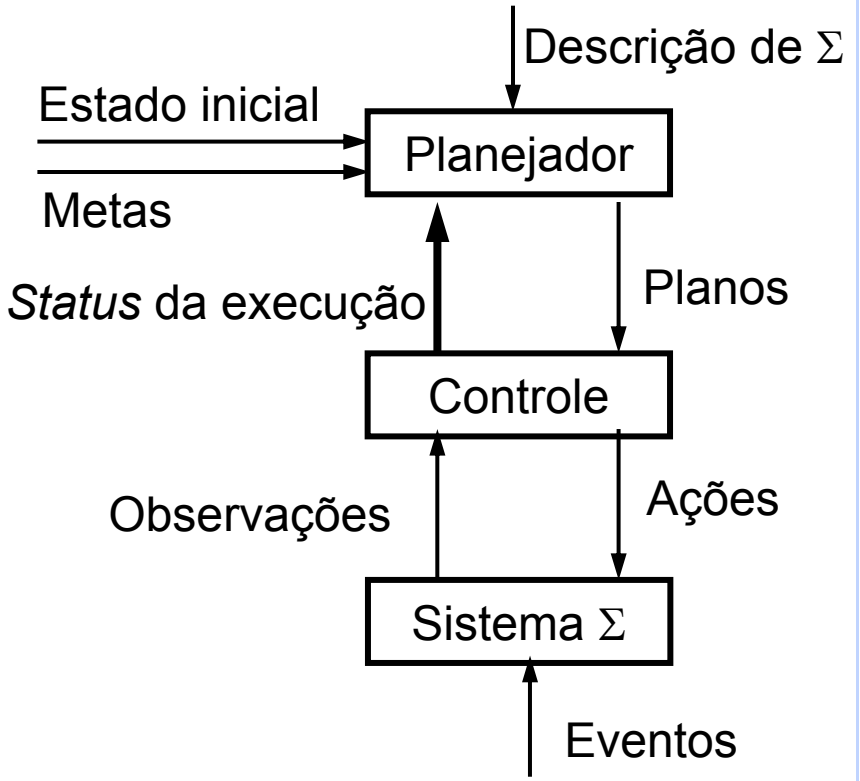
- Relaxar A3 (Σ estático):
  - Outros agentes ou ambientes dinâmicos
    - » Jogos finitos de soma-zero e informação perfeita (cursos introdutórios de IA)
  - Ambientes de comportamento aleatório
    - » Análise de decisão (business, pesquisa operacional)
    - » Algumas vezes pode ser mapeado em MDPs or POMDPs
- Relaxar A1 e A3
  - Jogos com informação inperfeita. Exemplo: bridge



A0: Finito
A1: Totalmente observável
A2: Determinístico
A3: Estático
A4: Satisfação de metas
A5: Planos seqüenciais
A6: Tempo implícito
A7: Planejamento *off-line*

Relaxando as Suposições

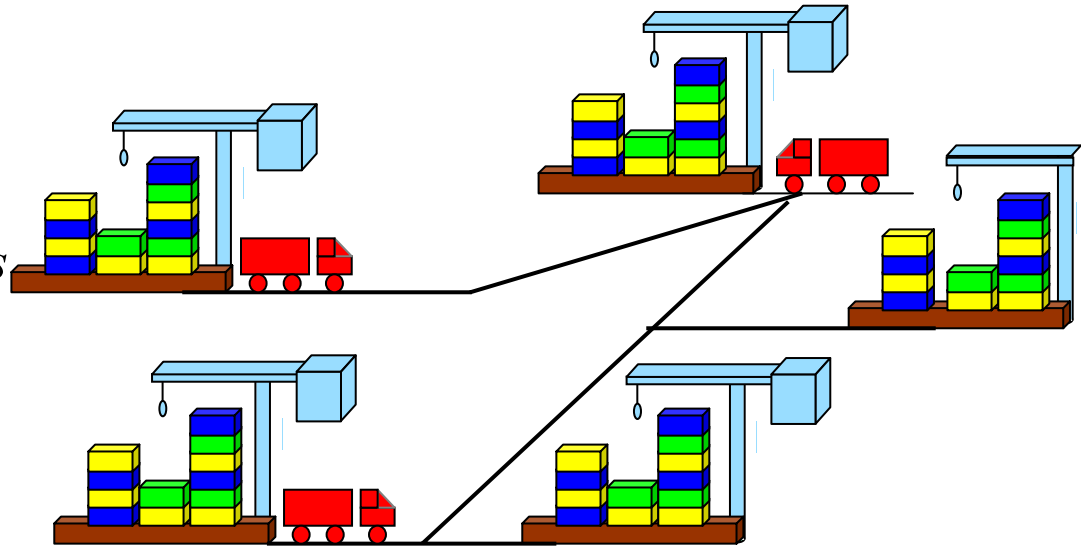
- Relaxar A5 (planos sequenciais) e A6 (tempo implícito):
  - Planos com paralelismo e concorrência
  - Planejamento temporal
- Relaxar A0, A5, A5
  - Planejamento e escalonamento de recursos
- Existem outras 247 combinações …



A0: Finito
A1: Totalmente observável
A2: Determinístico
A3: Estático
A4: Satisfação de metas
A5: Planos seqüenciais
A6: Tempo implícito
A7: Planejamento *off-line*

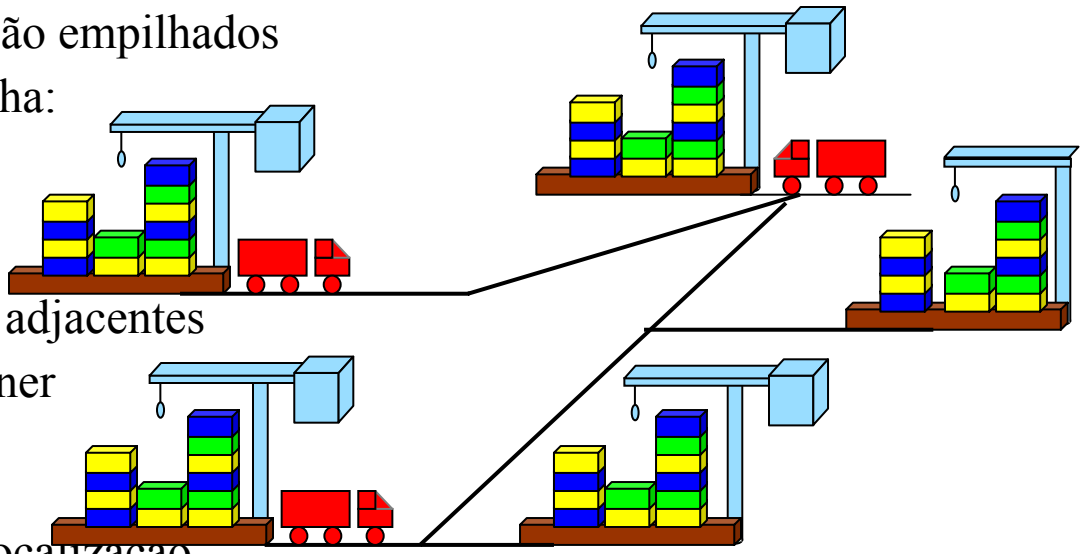
Dock Worker Robots

- Generalização do exemplo anterior
  - Um cais de porto com várias localizações
    - » e.g., docas, navios com docas, áreas de armazenagem, áreas de transferência de carga
  - *Containers*
    - » vão/vêm de navios
  - Carros Robôs
    - » Podem mover *containers*
  - Guindastes
    - » podem carregar ou descarregar *containers*



Um exemplo de execução: Dock Worker Robots

- **Localizações**: l1, l2, …
- **Containers**: c1, c2, …
  - Podem ser empilhados, carregados sobre os robôs, ou carregados pelos guindastes
- **Pilhas**: p1, p2, …
  - Áreas fixas onde os containers são empilhados
  - Plataforma no fundo de cada pilha: *pallet*
- **Carros Robôs**: r1, r2, …
  - Podem mover para localizações adjacentes
  - carregam no máximo um container
- **Guindaste**: k1, k2, …
  - cada um pertence a uma única localização
  - carrega um container de uma pilha para um carro robô e vice-e-versa
  - Se há uma pilha em uma localização então deve haver também um guindaste na mesma localização



A running example: Dock Worker Robots

- Relações fixas: é a mesma em todos os estados

  adjacent(l,l') attached(p,l) belong(k,l)

- Relações dinâmicas (fluentes): diferem de um estado para outro

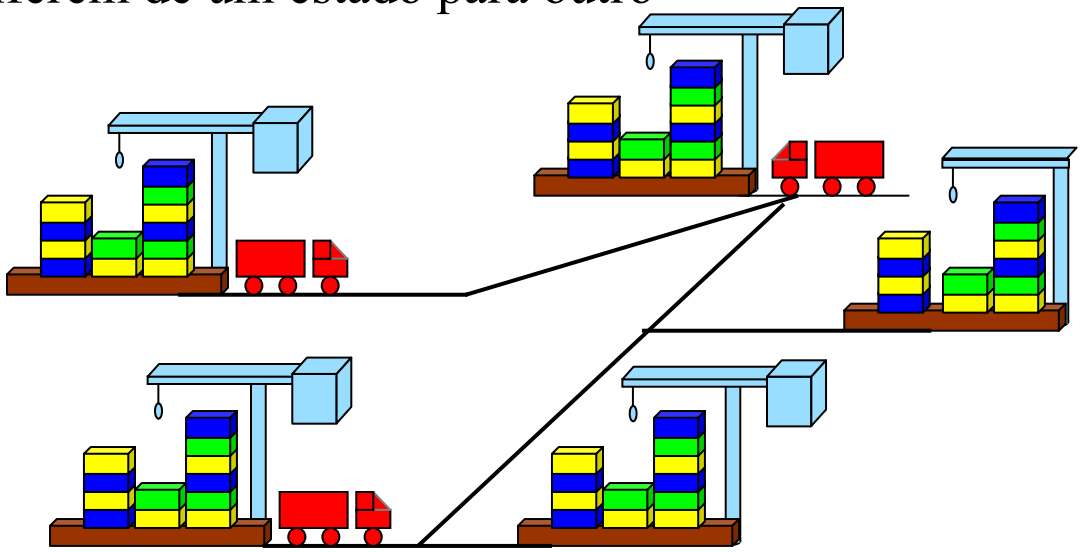
  occupied(l) at(r,l)

  loaded(r,c) unloaded(r)

  holding(k,c) empty(k)

  in(c,p) on(c,c')

  top(c,p) top(pallet,p)

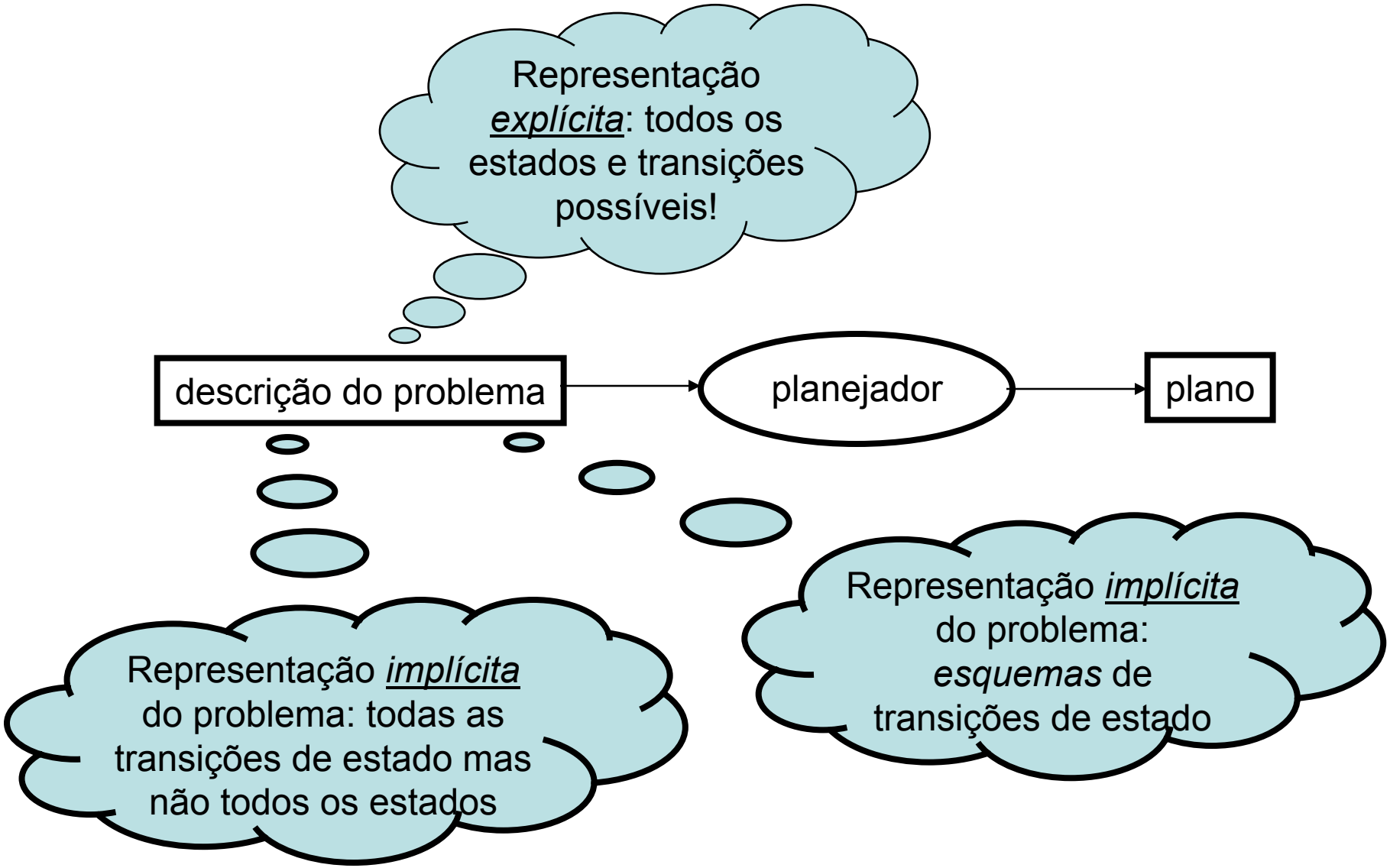


- Ações:

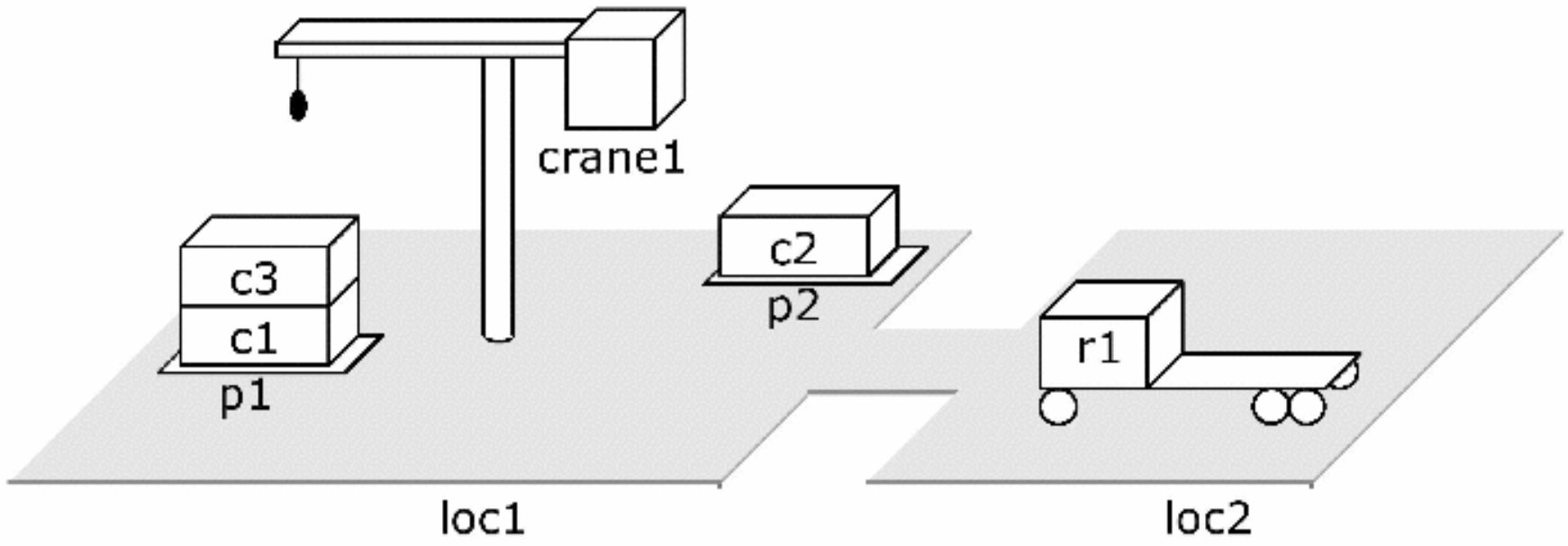
  take(c,k,p) put(c,k,p)

  load(r,c,k) unload(r) move(r,l,l')

Descrição do Problema de Planejamento



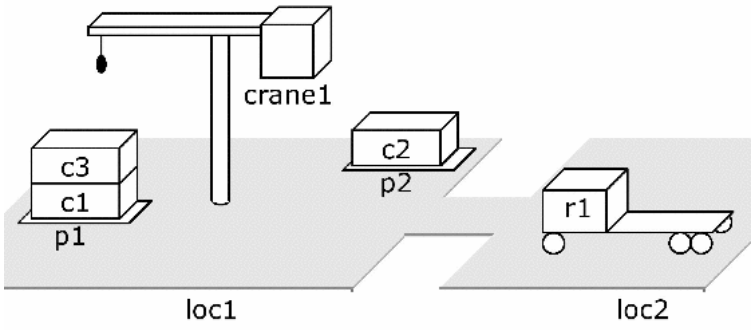
Exemplo de um estado



{attached(p1,loc1), in(c1,p1), in(c3,p1), top(c3,p1), on(c3,c1), on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.

Ações

- *Ações*: instâncias (através de substituições) de um operador



take(k, l, c, d, p)
;; crane k at location l takes c off of d in pile p
precond: belong(k, l), attached(p, l), empty(k), top(c, p), on(c, d)
effects: holding(k, c), $\neg$ empty(k), $\neg$ in(c, p), $\neg$ top(c, p), $\neg$ on(c, d), top(d, p)

take(crane1,loc1,c3,c1,p1)
;; crane1 at location loc1 takes c3 off c1 in pile p1
precond: belong(crane1,loc1), attached(p1,loc1),
empty(crane1), top(c3,p1), on(c3,c1)
effects: holding(crane1,c3), ¬empty(crane1), ¬in(c3,p1),
¬top(c3,p1), ¬on(c3,c1), top(c1,p1)

move(r, l, m)
;; robot r moves from location l to location m
precond: adjacent(l, m), at(r, l), $\neg$ occupied(m)
effects: at(r, m), occupied(m), $\neg$ occupied(l), $\neg$ at(r, l)

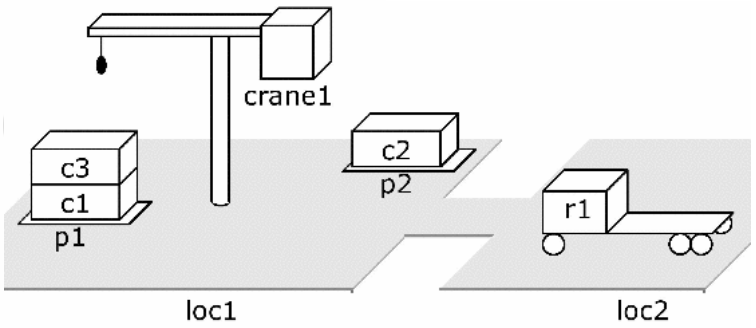
load(k, l, c, r)
;; crane k at location l loads container c onto robot r
precond: belong(k, l), holding(k, c), at(r, l), unloaded(r)
effects: empty(k), $\neg$ holding(k, c), loaded(r, c), $\neg$ unloaded(r)

unload(k, l, c, r)
;; crane k at location l takes container c from robot r
precond: belong(k, l), at(r, l), loaded(r, c), empty(k)
effects: $\neg$ empty(k), holding(k, c), unloaded(r), $\neg$ loaded

put(k, l, c, d, p)
;; crane k at location l puts c onto d in pile p
precond: belong(k, l), attached(p, l), holding(k, c), top(d, p)
effects: $\neg$ holding(k, c), empty(k), in(c, p), top(c, p), on(c, d), $\neg$ top(d, p)

take(k, l, c, d, p)
;; crane k at location l takes c off of d in pile p
precond: belong(k, l), attached(p, l), empty(k), top(c, p), on(c, d)
effects: holding(k, c), $\neg$ empty(k), $\neg$ in(c, p), $\neg$ top(c, p), $\neg$ on(c, d), top(d, p)

- Domínio de planejamento: linguagem + operadores
  - Exemplo: operadores para o domínio DWR
  - Corresponde a um conjunto de sistemas de estado-transição



Problemas de Planejamento

Dado um domínio de planejamento (linguagem L, operadores O)

- *Declaração de um problema de planejamento*: uma tripla $P=(O,s_0,g)$
  - O é uma coleção de operadores
  - s_0 é um estado (o estado inicial)
  - g é um conjunto de literais (a fórmula meta), sendo S_g, o conjunto de estados tal que $S_g \cap g = g$
- Formalmente, o *problema de planejamento* P *é dado pela tripla* (Σ,s_0,S_g)
  - s_0 e S_g (como definido acima)
  - $\Sigma = (S,A,\gamma)$ é um sistema de estado-transição
    - S = {conjuntos de todos *(ground)* átomos em L}
    - A = {todas as *(ground)* instâncias dos operadores em O}
    - γ = a função de transição de estado determinada pelos operadores

- Chamaremos de "problema de planejamento" à declaração de um problema de planejamento

Planos e Soluções

- *Plano*: qualquer seqüência de ações $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ tal que cada a_i é uma *(ground)* instância de um operador em O
- O plano é uma *solução para* $P=(O,s_0,g)$ se ele é executável em s_0 e atinge algum estado de S_g
  - i.e., se há estados $s_0, s_1, \ldots, s_n$ tal que
    - $\gamma(s_0,a_1) = s_1$
    - $\gamma(s_1,a_2) = s_2$
    - …
    - $\gamma(s_{n-1},a_n) = s_n$
    - s_n satisfaz g (ou $s_n \in S_g$)

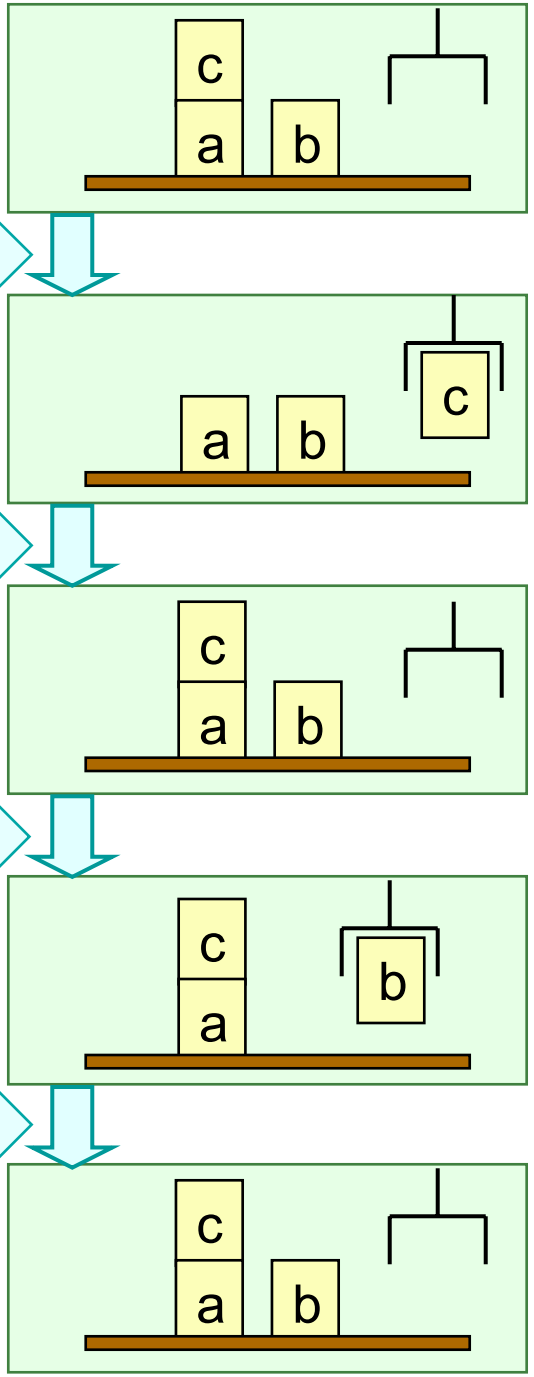
Operadores Clássicos

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



Ações de Teoria de Conjuntos

50 ações diferentes.

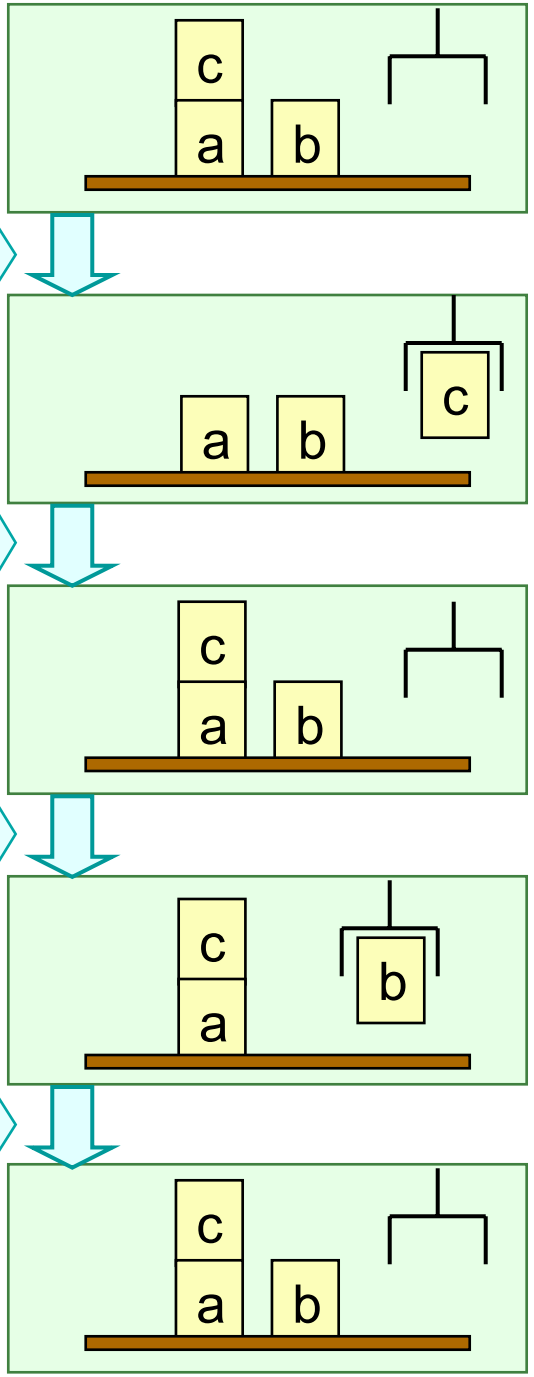
Aqui estão 4 delas:

unstack-c-a
Pre: on-c,a, clear-c, handempty
Del: on-c,a, clear-c, handempty
Add: holding-c, clear-a

stack-c-a
Pre: holding-c, clear-a
Del: holding-c, ~clear-a
Add: on-c-a, clear-c, handempty

pickup-c
Pre: ontable-c, clear-c, handempty
Del: ontable-c, clear-c, handempty
Add: holding-c

putdown-c
Pre: holding-c
Del: holding-c
Add: ontable-c, clear-c, handempty



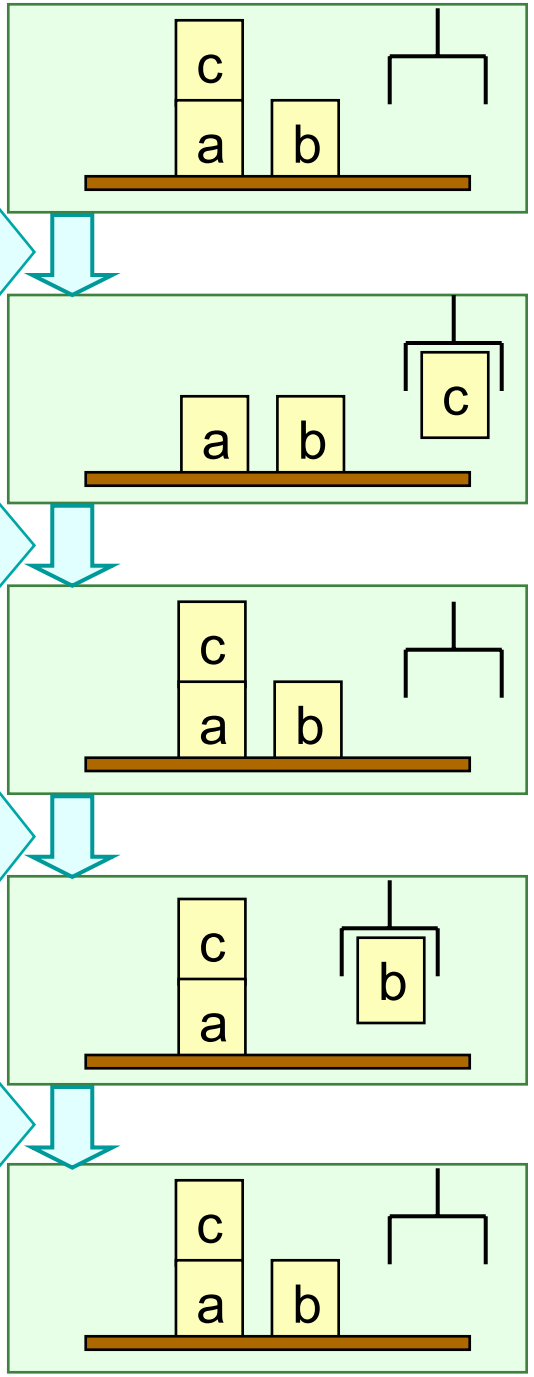
Operadores de Variáveis de Estado

unstack(*x* : block, *y* : block)
Precond: pos(*x*)=*y*, clear(*y*)=0, clear(*x*)=1, holding=nil
Effects: pos(*x*)=nil, clear(*x*)=0, holding=*x*, clear(*y*)=1

stack(*x* : block, *y* : block)
Precond: holding=*x*, clear(*x*)=0, clear(*y*)=1
Effects: holding=nil, clear(*y*)=0, pos(*x*)=y, clear(*x*)=1

pickup(*x* : block)
Precond: pos(*x*)=table, clear(*x*)=1, holding=nil
Effects: pos(*x*)=nil, clear(*x*)=0, holding=*x*

putdown(*x* : block)
Precond: holding=*x*
Effects: holding=nil, pos(*x*)=table, clear(*x*)=1



PDDL

- Linguagem padrão para decrever domínios de planejamento. Permite incluir: tipos, funções, variáveis numéricas, ações durativas, funções de otimização ==>

  planejamento/escalonamento

- Proposta inicial para acompetição de planejamento:
  AIPS 2002 Planning Competition
  http://www.dur.ac.uk/d.p.long/competition.htm

PDDL - ação Strips

```
(:action turn_to
    :parameters (?s - satellite ?d_new - direction ?d_prev - direction)
    :precondition (and (pointing ?s ?d_prev)
                       (not (= ?d_new ?d_prev))
                  )
    :effect (and  (pointing ?s ?d_new)
                       (not (pointing ?s ?d_prev))
            )
 )
```

PDDL - ação Strips-numérico

```
(:action turn_to
    :parameters (?s - satellite ?d_new - direction ?d_prev - direction)
    :precondition (and  (pointing ?s ?d_prev)
                        (not (= ?d_new ?d_prev))
                        (>= (fuel ?s) (slew_time ?d_new ?d_prev))
                  )
    :effect (and      (pointing ?s ?d_new)
                      (not (pointing ?s ?d_prev))
                      (decrease (fuel ?s) (slew_time ?d_new ?d_prev))
                      (increase (fuel-used) (slew_time ?d_new ?d_prev))
            )
 )
```

PDDL - ação Strips-temporal

```
(:durative-action turn_to
    :parameters (?s - satellite ?d_new - direction ?d_prev - direction)
    :duration (= ?duration 5)
    :condition (and  (at start (pointing ?s ?d_prev))
                     (over all (not (= ?d_new ?d_prev)))
                )
    :effect (and     (at end (pointing ?s ?d_new))
                     (at start (not (pointing ?s ?d_prev)))
            )
)
```

PDDL - ação Strips-temporal*

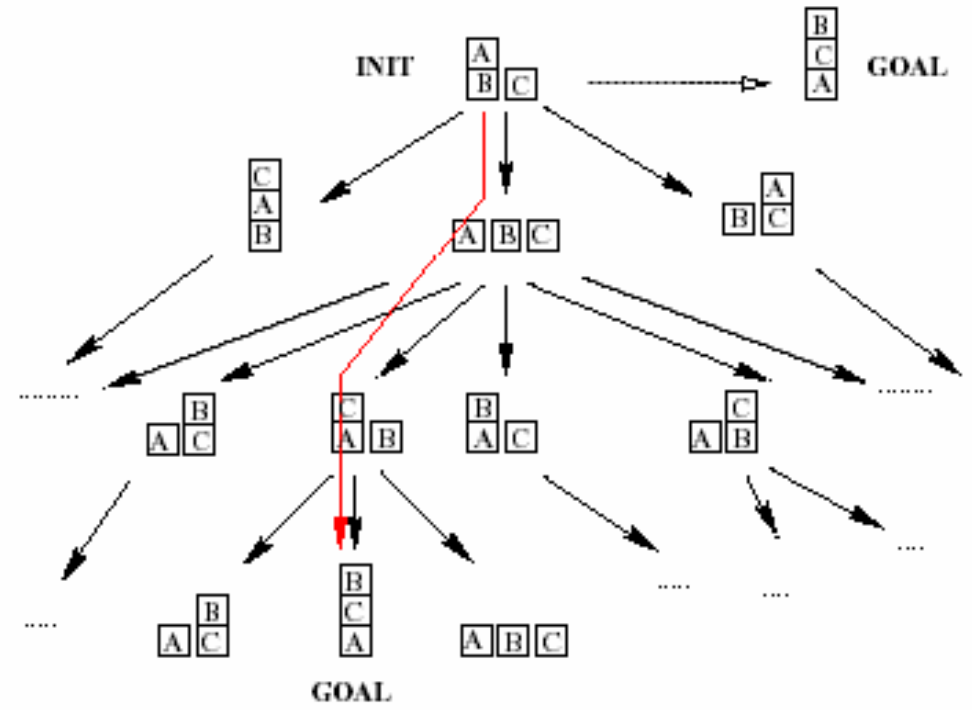
```
(:durative-action turn_to
    :parameters (?s - satellite ?d_new - direction ?d_prev - direction)
    :duration (= ?duration (slew_time ?d_prev ?d_new))
    :condition (and  (at start (pointing ?s ?d_prev))
                     (over all (not (= ?d_new ?d_prev)))
                )
    :effect (and     (at end (pointing ?s ?d_new))
                     (at start (not (pointing ?s ?d_prev)))
            )
)
```

PDDL - ação Strips-temporal*

```
(:durative-action take_image
    :parameters (?s - satellite ?d - direction ?i - instrument ?m - mode)
    :duration (= ?duration 7)
    :condition (and  (over all (calibrated ?i))
                     (over all (on_board ?i ?s))
                     (over all (supports ?i ?m) )
                     (over all (power_on ?i))
                     (over all (pointing ?s ?d))
                     (at end (power_on ?i))
                     (at start (>= (data_capacity ?s) (data ?d ?m)))
                )
    :effect (and     (at start (decrease (data_capacity ?s) (data ?d ?m)))
                     (at end (have_image ?d ?m))
                     (at end (increase (data-stored) (data ?d ?m)))       )
```

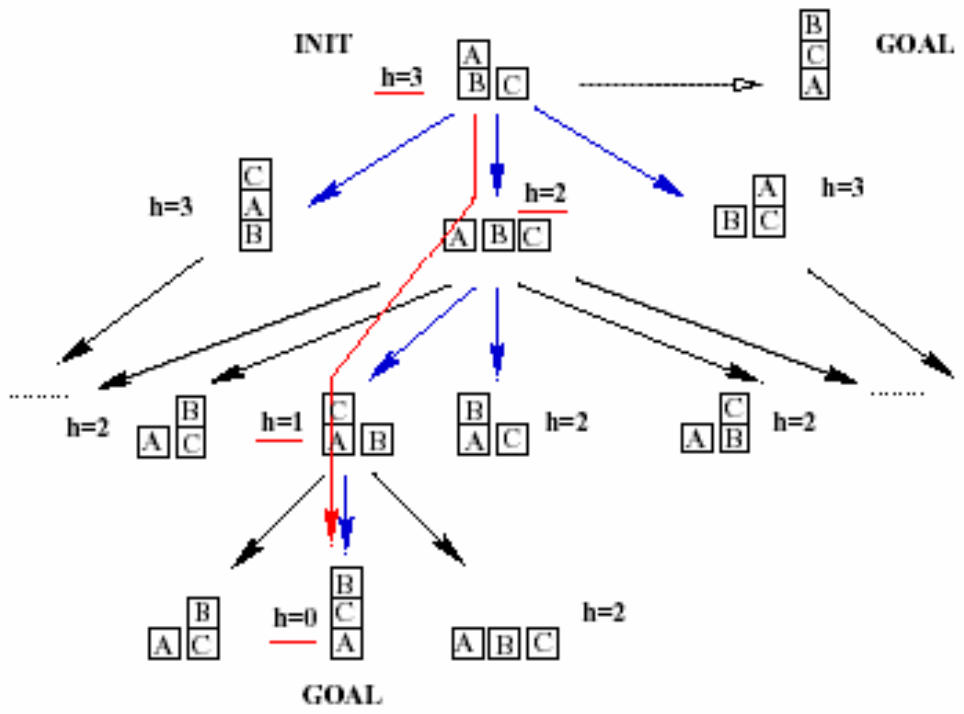
Planejamento como busca no espaço de estados

- Motivação:
  - Planejadores independentes de domínio possuem complexidade combinatória (intratável no pior-caso)
  - Planejamento heurístico: define formas de controle de busca



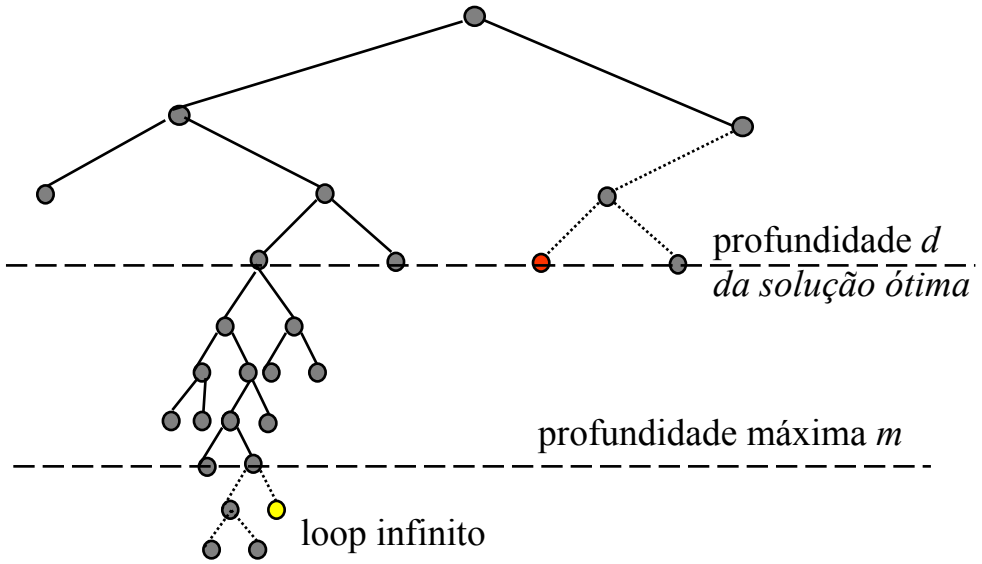
Planejamento cmo busca no espaço de estados

- Motivação:
  - Planejadores independentes de domínio possuem complexidade combinatória (intratável no pior-caso)
  - Planejamento heurístico: define formas de controle de busca



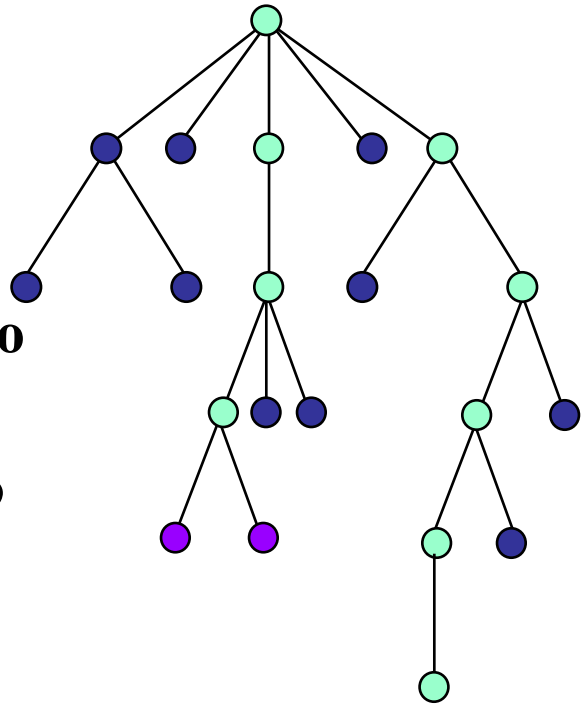
Planejamento como busca no espaço de estados

- Algoritmos de busca exaustiva:
  - » busca em largura: algoritmo completo mas $O(b^d)$ em tempo e memória
  - » busca em profundidade: não é completo mas é $O(b^d)$ em tempo e $O(bm)$ em memória
  - » busca iterativa em profundidade: algoritmo completo, $O(b^d)$ em tempo e $O(bm)$ em memória



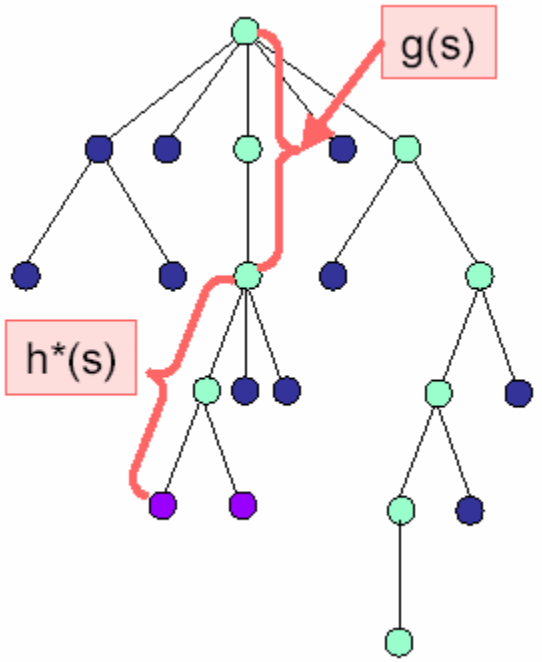
Heurística de seleção de nós

- Suponha que para cada estado *s*, temos uma estimativa *h(s)* da distância de *s* para uma solução
  - *h(s)* é usada para escolher qual será o **próximo nó a ser expandido**
  - quanto mais informativa for *h(s)* menor será o número de escolhas erradas
  - *h(s)* deve ainda ser facilmente computável
    - » heurísticas são geralmente baseadas no princípio de relaxação de problemas



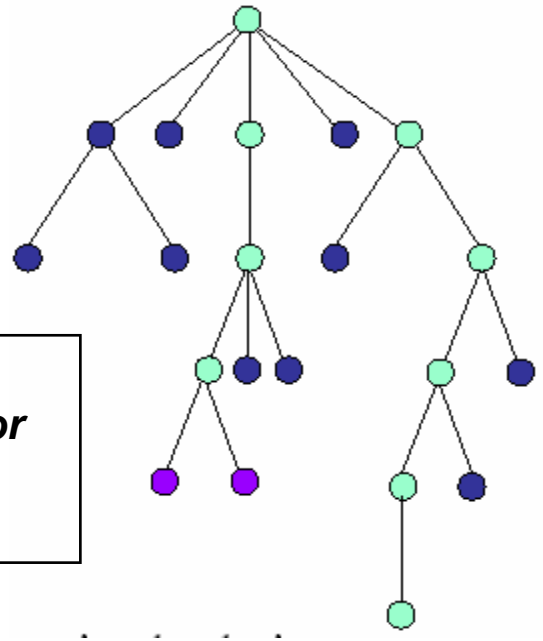
Heurística de seleção de nós

- Para cada estado s, seja
  - $g(s)$ = custo do caminho de s_0 à s
  - $h^*(s)$ = o menor custo de todos os caminhos de s para os nós meta
  - $f^*(s) = g(s) + h^*(s)$ = o menor custo de todos os caminhos de $s0$ aos nós meta que passam por s



- Seja $h(s)$ a função que estima $h^*(s)$
  - Seja $f(s) = g(s) + h(s)$
    - » $f(s)$ is uma estimativa de $f^*(s)$
  - h é *admissível* se para todo estado s, $0 \leq h(s) \leq h^*(s)$
  - Se h é admissível então f é um limitante inferior de f^*

O algoritmo A*



Laço

escolha um nó folha s tal que f(s) seja o menor valor
Se s é uma solução então devolva a solução
expanda s (gere os filhos de s)

- Se $h(s)$ é admissível então A* garante encontrar uma solução ótima
- Quanto mais "informativa" for a heurística (isto é, o quão próxima ela for de h^*) menor será o número de nós expandidos por A*
- Se $h(s)$ for admissível a menos de c, então A* garante encontrar uma solução ótima a menos da constante c

h(s): relaxando o problema de planejamento

- Desconsiderar efeitos^{-}(a) e precond^{-}(a). Assim, temos que:
  - $\gamma(s,a)$ cresce monotonicamente com o número de proposições de s para , $\gamma(s,a)$
  - É mais fácil computar a distância para a meta
  - Vamos ver heurísticas com base nessa idéia de relaxação.

Família de Funções Heurísticas para Planejamento

Sejam *s* e *s'* estados, *p* uma proposição e *g* um conjunto de proposições.

- **Δ*(*s*,*p*):** distância mínima entre *s* e *p*, é o número mínimo de ações necessárias para ir de *s* a um estado que contém *p*
- **Δ*(*s*,C):** distância mínima entre *s* e s', é o número mínimo de ações necessárias para ir de *s* a um estado *s'* que contém um conjunto C de proposições
- **Δ*(*s*,*g*)=h*(*s*):** distância mínima entre *s* e *g*, é o número mínimo de ações necessárias para ir de *s a* um estado contendo todas as proposições em *g* (meta).
- Seja Δ(*s*,*p*) e Δ(*s*,*g*) estimativas de Δ*(*s*,*p*) e Δ*(*s*,*g*), respectivamente
- Família de funções heurísticas de planejamento:
  - $\Delta_i(s,p)$, $\Delta_i(s,g)$, onde i = 0, 1, 2, …

  Definem diferentes estimativas de Δ*(*s*,*p*) e Δ*(*s*,g) que se baseiam em diferentes relaxações do problema de planejamento

Função Heurística Δ_0

- Δ_0 ignora os efeitos e as pré-condições negativas das ações e aproxima a distância para g como a soma das distâncias para as proposições em g

$$\begin{aligned}
&\Delta_0(s,p) = 0 && \text{if } p \in s,\\
&\Delta_0(s,p) = \infty && \text{if } \forall a \in A, p \notin \text{effects}^+(a), \text{ and } p \notin s,\\
&\Delta_0(s,g) = 0 && \text{if } g \subseteq s,
\end{aligned} \tag{9.1}$$

caso contrário:

$$\Delta_0(s,p) = \min_a\{1 + \Delta_0(s, \text{precond}^+(a)) \mid p \in \text{effects}^+(a)\}$$
$$\Delta_0(s,g) = \sum_{p \in g} \Delta_0(s,p)$$

- $h_0(s) = \Delta_0(s,g)$, onde g é a meta dada por um conjunto de proposições
- … *heurística aditiva,* faz a suposição de independência: cada proposição $p \in g$ pode ser alcançada de maneira independente.
- $\Delta_0(s,g)$ não é admissível (mas nem sempre importa)

Uma heurística admissível

$$\Delta_1(s,p) = 0 \qquad \text{if } p \in s,$$
$$\Delta_1(s,p) = \infty \qquad \text{if } \forall a \in A, p \notin \text{effects}^+(a), \text{ and } p \notin s,$$
$$\Delta_1(s,g) = 0 \qquad \text{if } g \subseteq s,$$
otherwise:
$$\Delta_1(s,p) = \min_a\{1 + \Delta_0(s, \text{precond}^+(a)) \mid p \in \text{effects}^+(a)\}$$
$$\Delta_1(s, g) = \max_{p \in g} \Delta_1(s,p) = h_1(s)$$

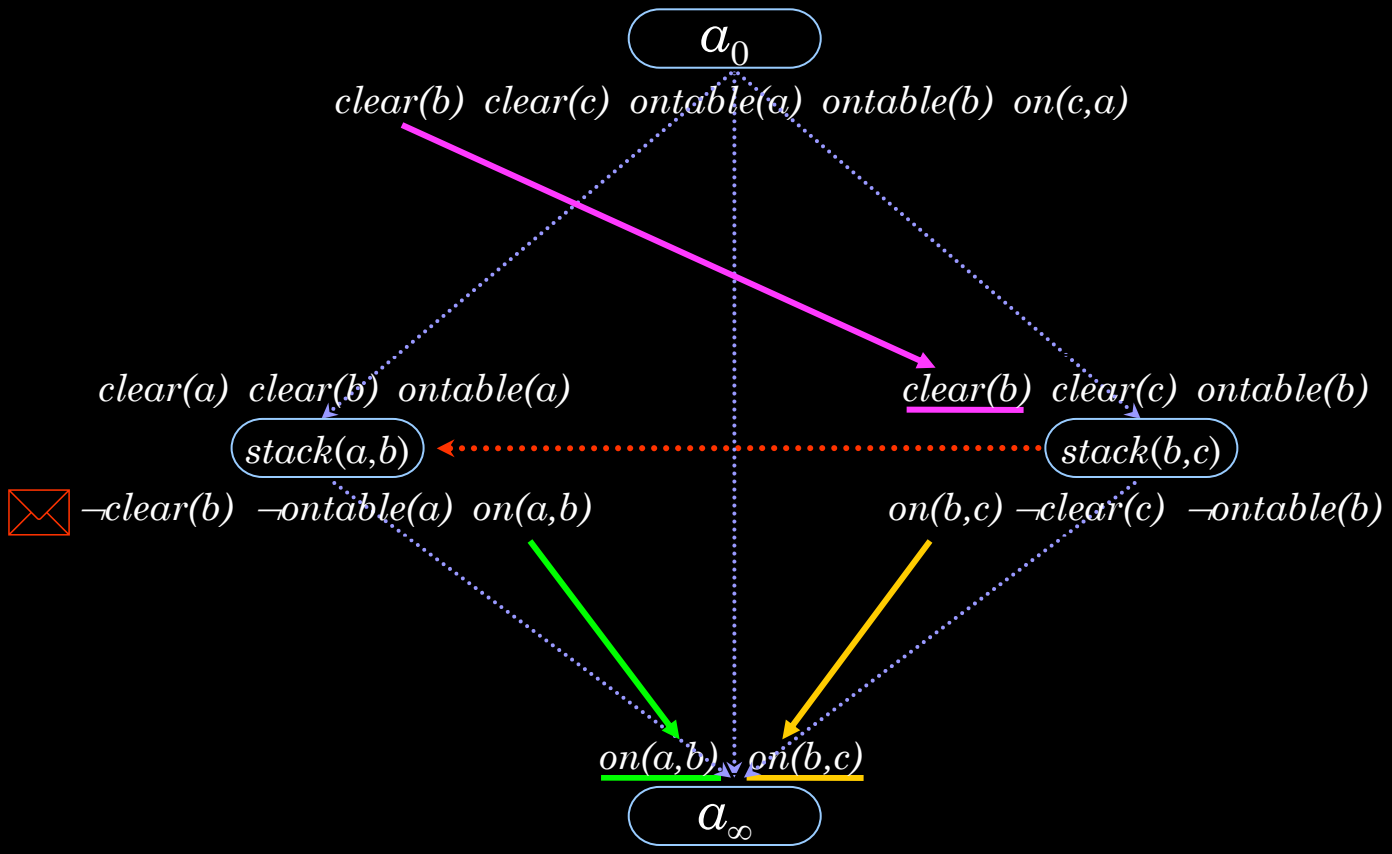
- $h_1(s)$ = é uma heurística admissível (mas não muito informativa)
  - Também ignora os efeitos e as pré-condições negativas das ações
  - O custo de atingir um conjunto de pré-condições $\{p_1, \ldots, p_n\}$ é o **max** dos custos de se alcançar cada p_i separadamente
- Cálculo de Δ_1: igual a Δ_0 , exceto que $\Delta_1(s,g) = \max_{p \in g} \Delta_0(s,p)$

The PSP Procedure

PSP(π)
  $flaws \leftarrow$ OpenGoals(π) $\cup$ Threats(π)
  if $flaws = \emptyset$ then return(π)
  select any flaw $\phi \in flaws$
  $resolvers \leftarrow$ Resolve(ϕ, π)
  if $resolvers = \emptyset$ then return(failure)
  nondeterministically choose a resolver $\rho \in resolvers$
  $\pi' \leftarrow$ Refine(ρ, π)
  return(PSP(π'))
end

- POP (PSP) é correto e completo

Plano encontrado pelo POP



S = {$stack(b,c)$, $stack(a,b)$, a_0, a_∞}

O = {$stack(b,c) < stack(a,b)$, $a_0 < stack(b,c) < a_\infty$, $a_0 < stack(a,b) < a_\infty$, $a_0 < a_\infty$}

$\mathcal{L}$ = {$a_0 \rightarrow clear(b)@stack(b,c)$, $stack(b,c) \rightarrow on(b,c)@a_\infty$, $stack(a,b) \rightarrow on(a,b)@a_\infty$}

Markov Decision Process (MDP)

- **S**: A set of states
- **A**: A set of actions
- **Pr**(s'|s,a): transition model
- **C**(s,a,s'): cost model
- **G**: set of goals
- s_0: start state
- γ: discount factor
- **R**(s,a,s'): reward model

Value function (V(s): expected long term reward from the state

Q values: Expected long term reward of doing a in s

$V(s) = \max Q(s,a)$

Greedy Policy w.r.t. a value function

Value of a policy

Optimal value function

Bellman Equations for infinite horizon discounted reward maximization MDP

- <**S**, **A**, **P**r, **R**, s_0, γ>
- Define **V*(s)** {optimal **value**} as the **maximum** expected **discounted reward** from this state.
- V* should satisfy the following equation:

$$V^*(s) = \max_{a \in Ap(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma V^*(s') \right]$$

Examples of MDPs

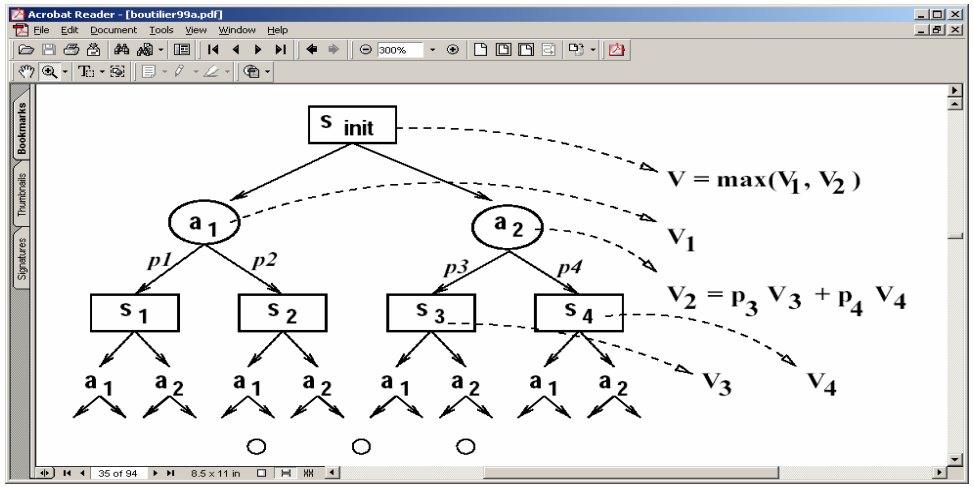
- Goal-directed, Indefinite Horizon, Cost Minimization MDP
  - <**S**, **A**, **P**r, **C**, **G**, s_0>
  - Most often studied in planning community
- Infinite Horizon, Discounted Reward Maximization MDP
  - <**S**, **A**, **P**r, **R**, γ>
  - Most often studied in reinforcement learning
- Goal-directed, Finite Horizon, Prob. Maximization MDP
  - <**S**, **A**, **P**r, **G**, s_0, T>
  - Also studied in planning community
- Oversubscription Planning: Non absorbing goals, Reward Max. MDP
  - <**S**, **A**, **P**r, **G**, **R**, s_0>
  - Relatively recent model

Ideas for Efficient Algorithms..

- Use heuristic search (and reachability information)
- Use execution and/or Simulation
  - "Actual Execution" Reinforcement learning

    (Main motivation for RL is to "learn" the model)
  - "Simulation" –simulate the given model to sample possible futures

- Use "factored" representations
  - Factored representations for Actions, Reward Functions, Values and Policies
  - Directly manipulating factored representations during the Bellman update



Real Time Dynamic Programming
[Barto, Bradtke, Singh'95]

- Trial: simulate greedy policy starting from start state;
  perform Bellman backup on visited states

- RTDP: repeat Trials until cost function converges

Notice that you can also do the "Trial" above by *executing* rather than "simulating". In that case, we will be doing reinforcement learning. (In fact, RTDP was originally developed for reinforcement learning)

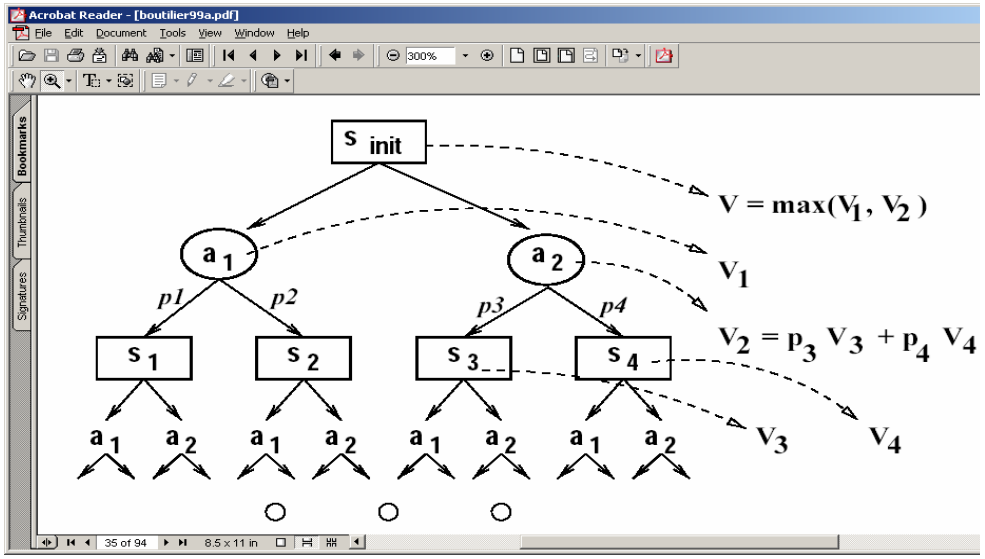
RTDP Approach: Interleave Planning & Execution (Simulation)

Start from the current state S. Expand the tree (either uniformly to k-levels, or non-uniformly—going deeper in some branches)

Evaluate the leaf nodes; back-up the values to S. Update the stored value of S.

Pick the action that leads to best value

Do it {or simulate it}. Loop back.



Leaf nodes evaluated by

Using their "cached" values

→If this node has been evaluated using RTDP analysis in the past, you use its remembered value else use the heuristic value

→If not use heuristics to estimate

a. Immediate reward values

b. Reachability heuristics

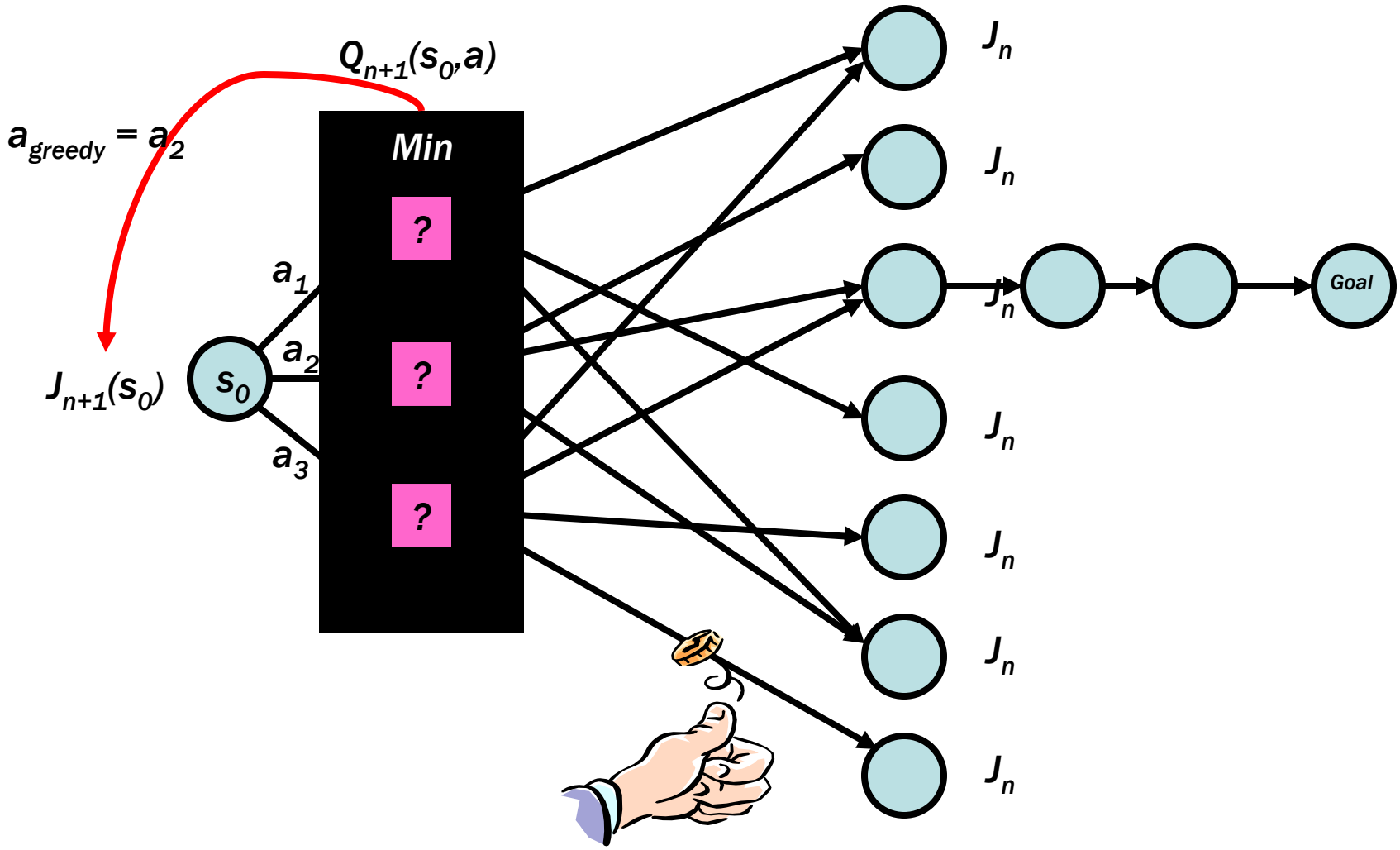
Sort of like depth-limited game-playing (expectimax)

--Who is the game against?

Can also do "reinforcement learning" this way

→ The M_{ij} are not known correctly in RL

RTDP Trial



Livro recomendado

- M. Ghallab, D. Nau e P. Traverso
  Automated Planning: Theory and Practice
  Morgan Kaufmann Publishers, 2004

- S. Russel e P. Norvig
  Artificial Intelligence: a Modern Approach