Planejamento em
Inteligência Artificial

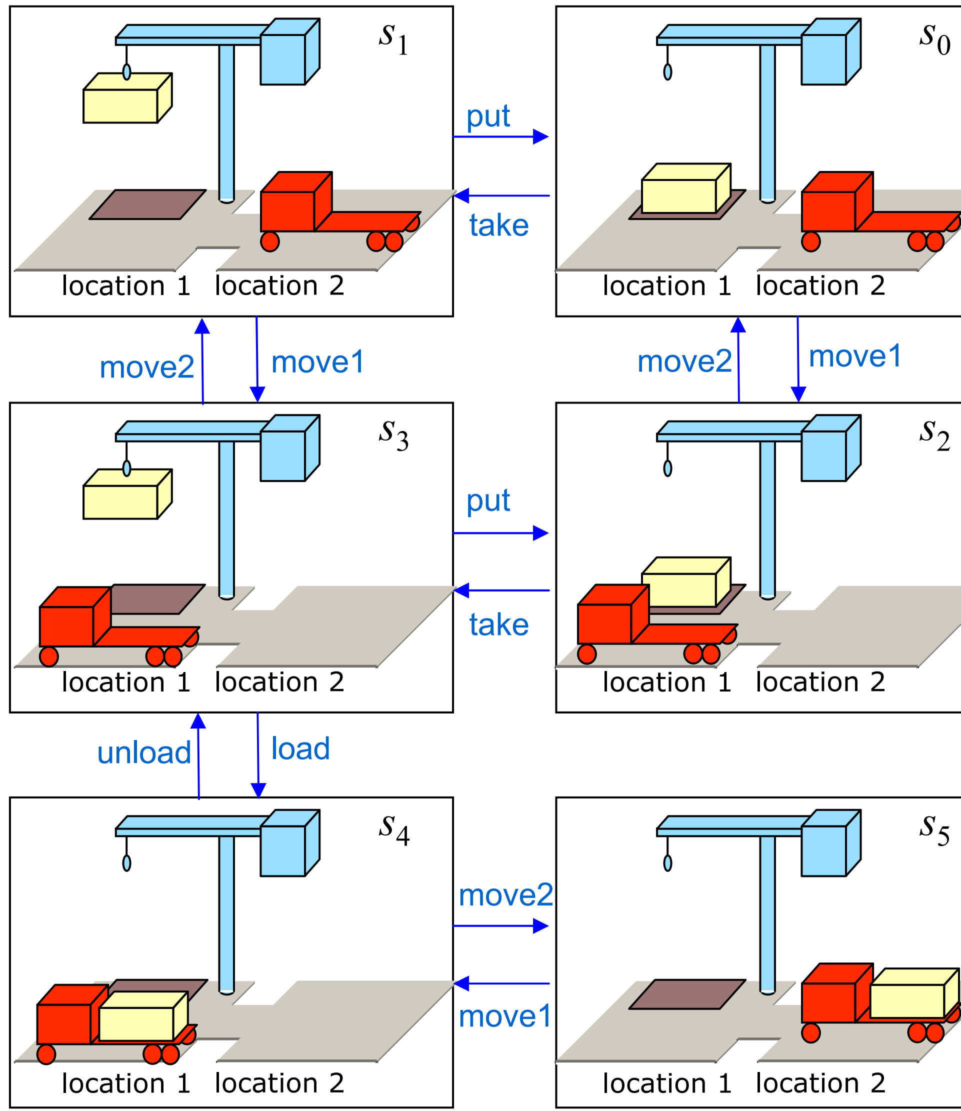
Capítulo 2
Representação de Problemas em Planejamento Clássico

Leliane Nunes de Barros

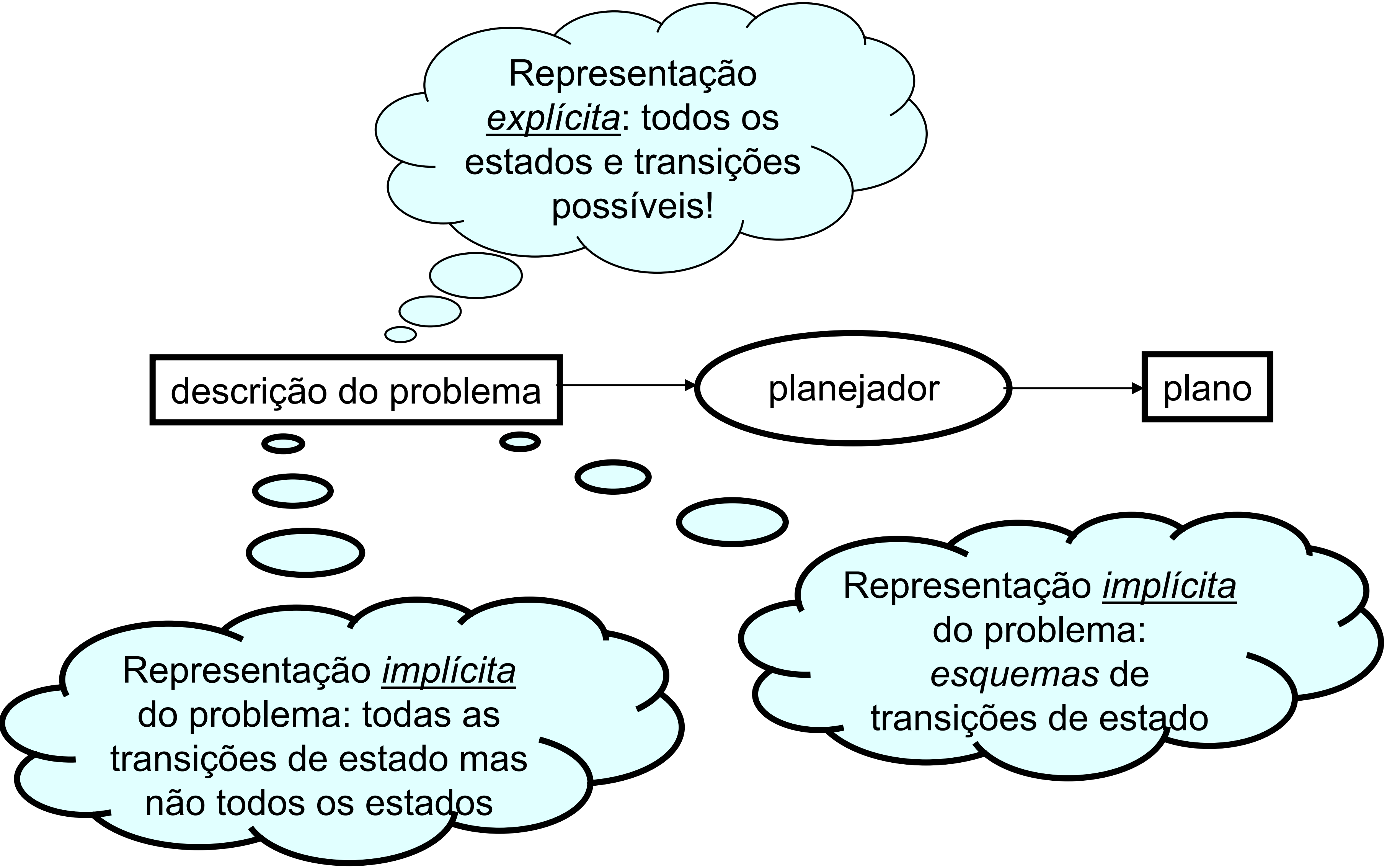
MAC 5788 - IME/USP
segundo semestre de 2005

Revisão de Planejamento Clássico

- Planejamento clássico faz as 8 suposições restritivas:
  - A0: Finito
  - A1: Totalmente observável
  - A2: Determinístico
  - A3: Estático
  - A4: Satisfação de metas
  - A5: Planos seqüenciais
  - A6: Tempo implícito
  - A7: Planejamento *off-line*



Descrição do Problema de Planejamento



Representações: Motivação

- Na maioria dos problemas, existem muitos estados para representá-los explicitamente (s_0, s_1, s_2, …)
- Podemos representar cada estado como um conjunto de características. Por exemplo:
  - » um vetor de valores para um conjunto de variáveis
  - » um conjunto de (*ground*) átomos em alguma linguagem de primeira ordem *L*
- Também podemos definir um conjunto de *operadores* que podem ser usados para computar as transições de estados
  - Sem fornecer todos os estados explicitamente:
    - » fornecendo apenas o estado inicial
    - » e usando o operador para gerar os outros estados, quando necessário

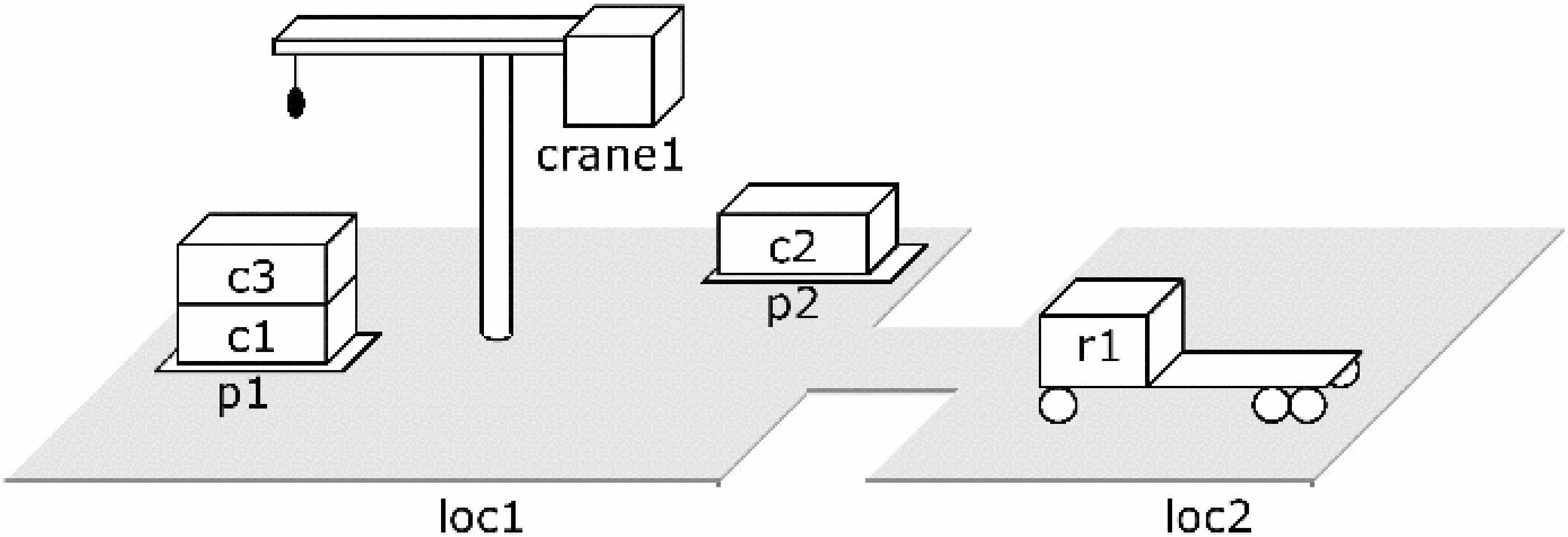
Tópicos dessa aula

- Representação de problemas de planejamento
  - Representação clássica
  - Representação de teoria de conjuntos
  - Representação de variáveis de estado
  - Exemplos: DWR e Mundo dos Blocos
  - Comparações

Representação Clássica

- Começamos com uma linguagem de primeira ordem, livre de funções com:
  - Finitamente muitos símbolos de predicados e símbolos constantes, mas sem símbolos funcionais
  - Expressão (literal *ground*): não contém símbolos variáveis - e.g., on(c1,c3)
  - Expressão (literal u*nground*): com pelo menos uma variável - e.g., on(c1,*X*)
  - *Átomos*: símbolos predicados e termos (ctes ou vars) - e.g., on(c1,c3), on(c1,*X*)
  - *Substituição*: $\theta = \{X_1 \leftarrow v_1,\ X_2 \leftarrow v_2,\ \ldots,\ X_n \leftarrow v_n\}$
    - » Cada X_i é um <u>símbolo variável</u>; cada v_i é um termo
  - *Instância* de *e*: resultado da aplicação de uma substituição θ a *e* por ctes
- *Estado*: um conjunto de (*ground*) átomos
  - Os estados representam as coisas que são verdadeiras em um dos estados do sistema Σ ➔ *suposição do mundo fechado* (CWA)
  - Finitamente muitos átomos, portanto temos "somente" finitamente muitos estados possíveis

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)}.

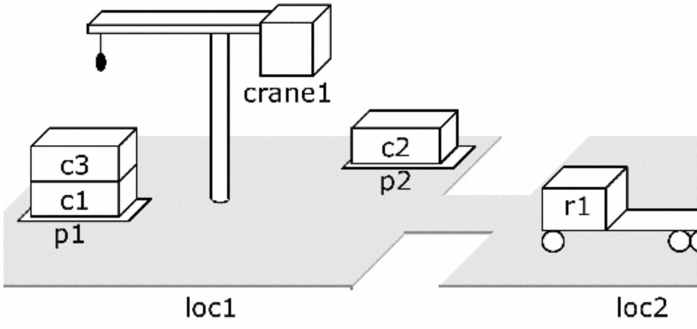
Ações são representadas por operadores

- *Operador*: uma tripla o=(nome(o), precond(o), efeitos(o))
  - *nome(o)*: é uma expressão sintática da forma $n(x_1,\ldots,x_k)$
    - » n: símbolo de *operador* – deve ser único para cada operador
    - » $x_1,\ldots,x_k$: símbolos variáveis (parâmetros)
      - deve incluir cada símbolo de variável em o
  - *precond(o)*: precondições
    - » literais que devem ser verdadeiras para ser possível usar/executar o operador
  - *efeitos(o)*: efeitos
    - » literais que o operador tornará verdadeiros ou falsos

take (*k,l,c,d,p*)
;; guindaste *k* na localização *l* retira *c* de *d* na pilha *p*
precond: belong(*k,l*), attached(*p,l*), empty(*k*), top(*c,p*), on(*c,d*)
efeitos: holding(*k,c*), ¬ empty(*k*), ¬ in(*c,p*), ¬ top(*c,p*), ¬ on(*c,d*), top(*d,p*),

Ações

- *Ações*: (*ground*) instância (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)

Notação

Seja S um conjunto de literais. Então:

» S^+ = {átomos que aparecem positivamente em S}

» S^- = {átomos que aparecem negativamente em S}

Mais especificamente, seja a um operador ou ação. Então

» precond$^+(a)$ = {átomos positivos de precond(a)}

» precond$^-(a)$ = {átomos negativos de precond(a)}

» efeitos$^+(a)$ = {átomos positivos de efeitos(a)}

» efeitos$^-(a)$ = {átomos negativos de efeitos(a)}

ke(k, l, c, d, p)

;; crane k at location l takes c off of d in pile p

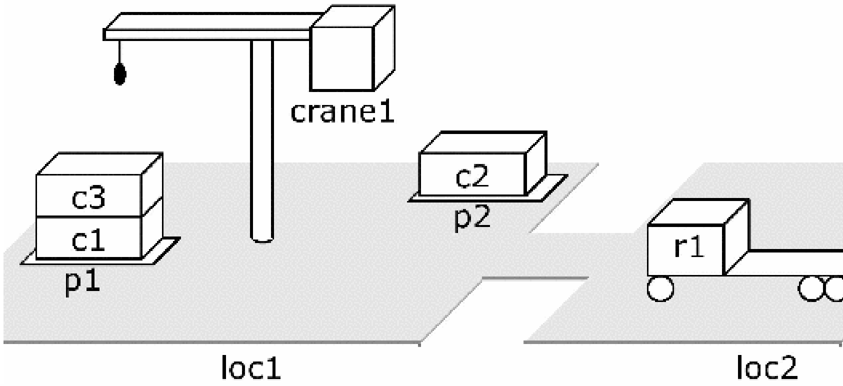
precond: belong(k, l), attached(p, l), empty(k), top(c, p), on(c, d)

effects: holding(k, c), $\neg$ empty(k), $\neg$ in(c, p), $\neg$ top(c, p), $\neg$ on(c, d), top$(d,$

- efeitos$^+$(take(k,l,c,d,p) = {holding(k,c), top(d,p)}
- efeitos$^-$(take(k,l,c,d,p) = {empty(k), in(c,p), top(c,p), on(c,d)}

Aplicabilidade de ações

- Uma ação <u>a</u> é *aplicável* a um estado <u>s</u> se <u>s</u> satisfaz precond(*a*),
  - i.e., se precond$^+$(a) $\subseteq s$ e precond$^-$(a) $\cap\ s = \varnothing$
- Exemplo de um estado e uma ação aplicável:



take(crane1,loc1,c3,c1,p1)

;; crane 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)

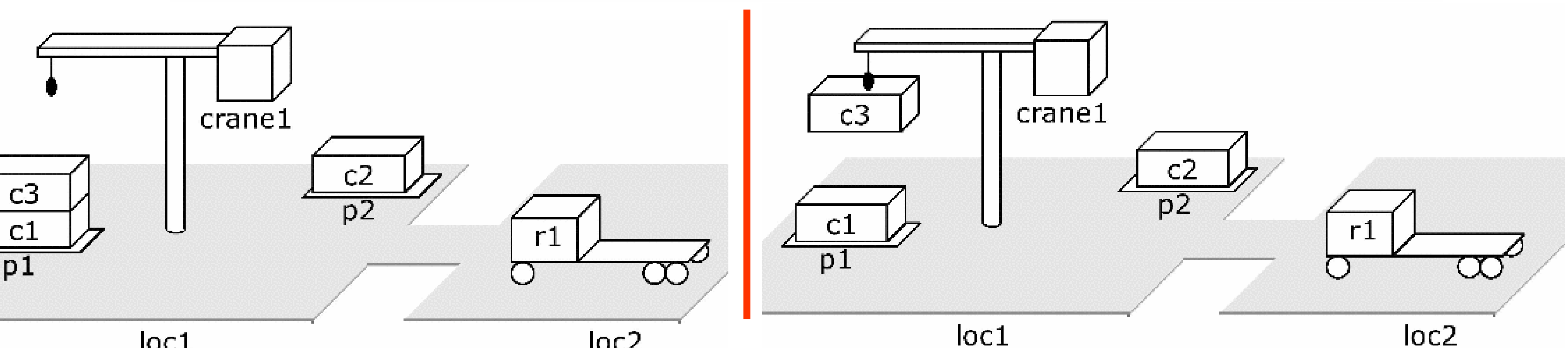
Resultado da execução de uma ação

Se *a* é aplicável a *s*, o resultado de sua execução é:

$$\gamma(s,a) = (s - \text{efeitos}^{-}(a)) \cup \text{efeitos}^{+}(a)$$

- Remover os efeitos negativos, e adicionar os efeitos positivos

take(crane1,loc1,c3,c1,p1)
;; crane 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)



ove(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)

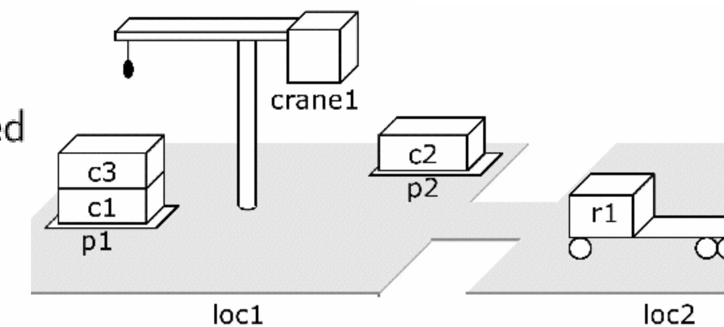
ad(k, l, c, r)
;; crane k at location l loads container c onto robot r
precond: belong(k, l), holding(k, c), at(r, l), unloaded(r)
effects: empty(k), $\neg$ holding(k, c), loaded(r, c), $\neg$ unloaded(r)

load(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

t(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)

ke(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 d 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 $S_g \cap g = g$
- 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

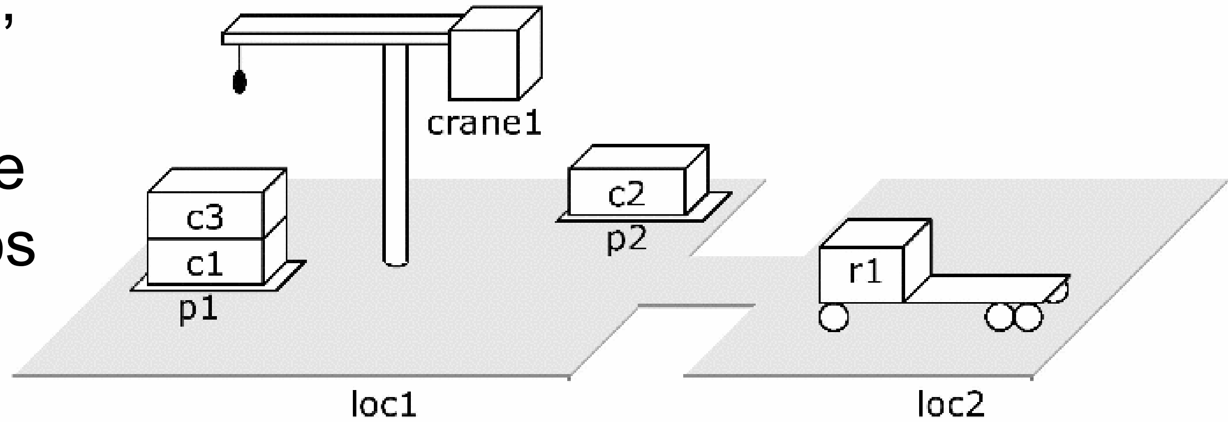
Planos e Soluções

- *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 $s_n \in S_g$)

Exemplo

Seja $P_1 = (O, s_1, g_1)$, onde

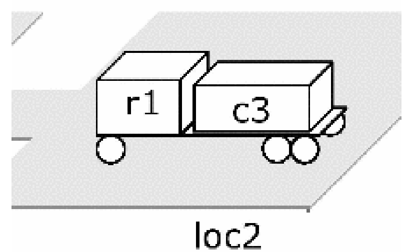
- O é o conjunto de operadores dados anteriormente



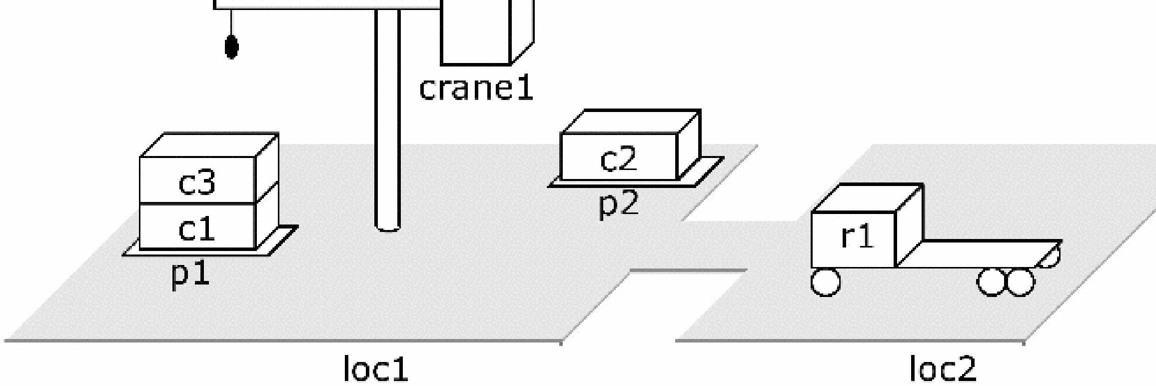
- s_1 é:

{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), adja cent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.

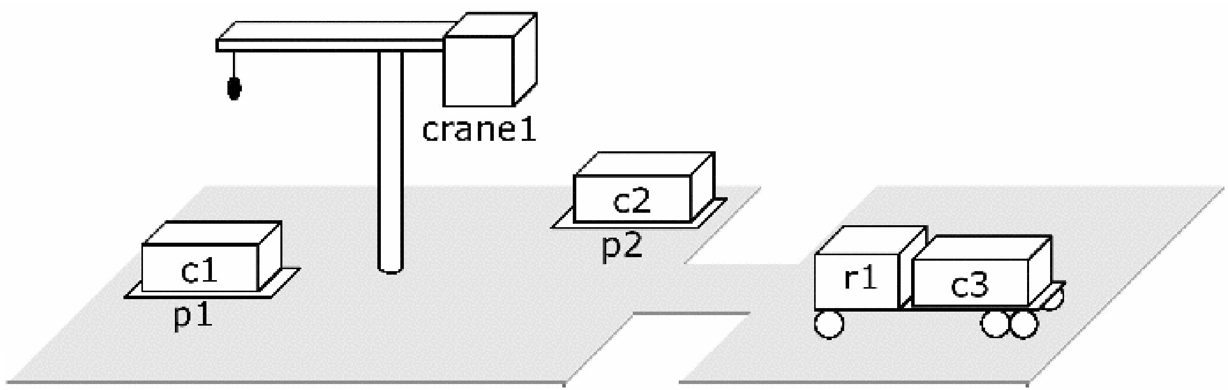
- g_1={loaded(r1,c3), at(r1,loc2)}



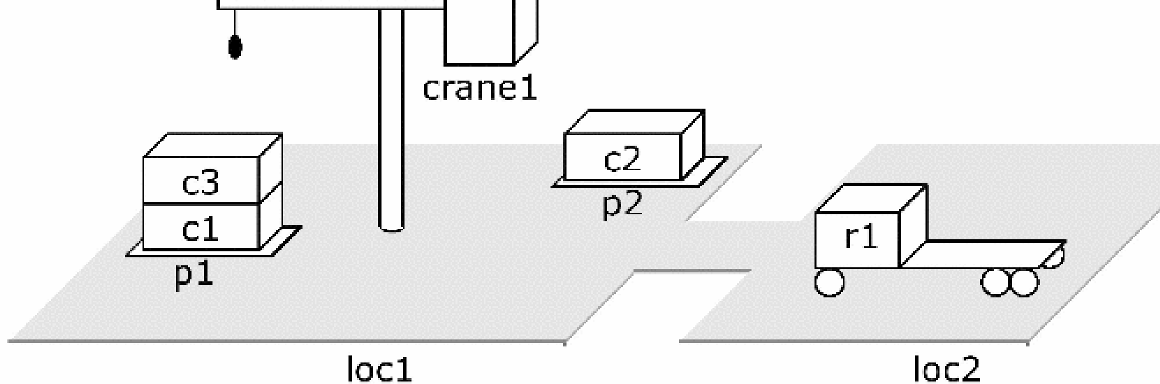
Exemplo (continuação)



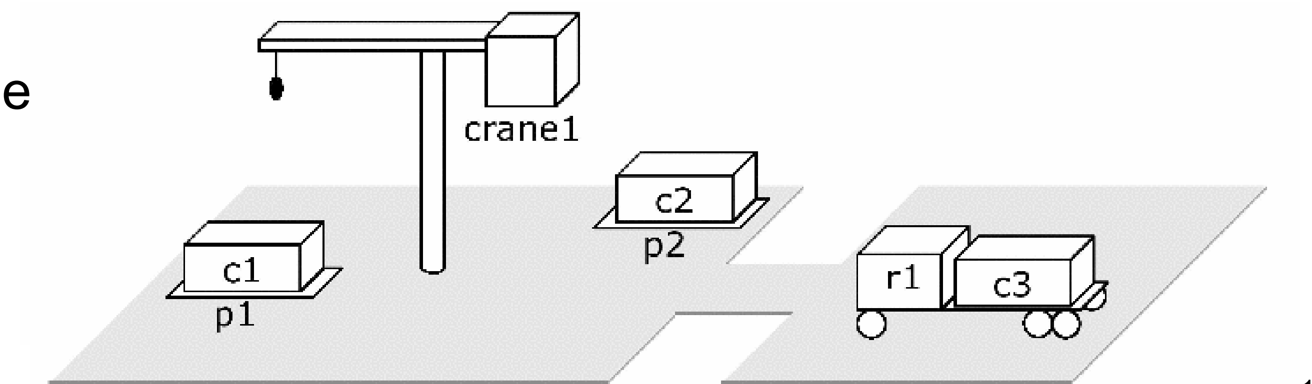
- Três soluções para *P*:
  - ⟨take(crane1,loc1,c3,c1,p1), move(r1,loc2,loc1), move(r1,loc1,loc2), move(r1,loc2,loc1), load(crane1,loc1,c3,r1), move(r1,loc1,loc2)⟩
  - ⟨take(crane1,loc1,c3,c1,p1), move(r1,loc2,loc1), load(crane1,loc1,c3,r1), move(r1,loc1,loc2)⟩
  - ⟨move(r1,loc2,loc1), take(crane1,loc1,c3,c1,p1), load(crane1,loc1,c3,r1), move(r1,loc1,loc2)⟩
- Cada uma delas produz o estado:



Exemplo (continuação)



- A primeira é *redundante*: ações podem ser removidas e ainda teremos uma solução
  - ⟨take(crane1,loc1,c3,c1,p1), move(r1,loc2,loc1), move(r1,loc1,loc2), move(r1,loc2,loc1), load(crane1,loc1,c3,r1), move(r1,loc1,loc2)⟩
  - ⟨take(crane1,loc1,c3,c1,p1), move(r1,loc2,loc1), load(crane1,loc1,c3,r1), move(r1,loc1,loc2)⟩
  - ⟨move(r1,loc2,loc1), take(crane1,loc1,c3,c1,p1), load(crane1,loc1,c3,r1), move(r1,loc1,loc2)⟩
- A 2º e a 3º são *não-redundantes* e são planos mais curtos (*planos minimais*)

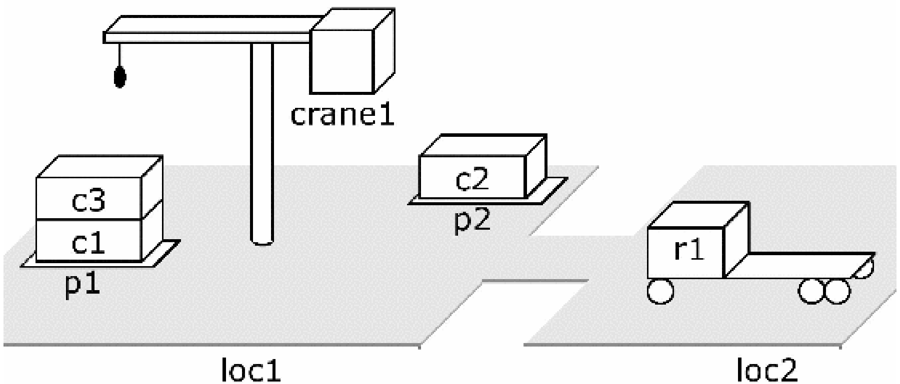


Relevância de ações

- Uma ação _a_ é *relevante* para uma meta *g* se ela for aplicável no estado corrente e:

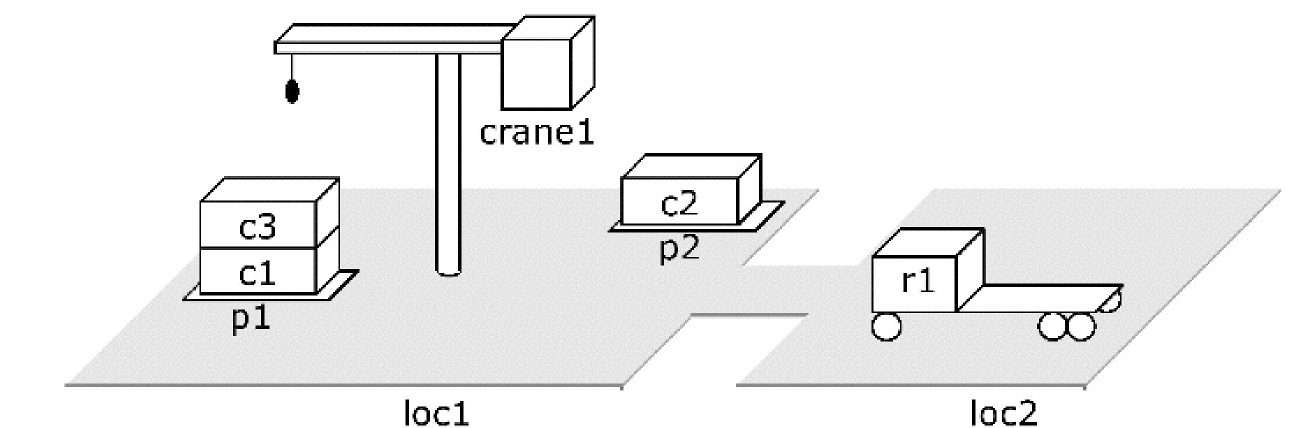
$$g \cap \text{efeitos}^{+}(a) \neq \emptyset \quad \text{e } g \cap \text{efeitos-}(a) = \emptyset$$

- Exemplo de um estado e uma ação relevante:



Representação baseada em Teoria de Conjuntos

Como a representação clássica, mas restrita à lógica proposicional



Estados:

- Ao invés de uma coleção de *ground* átomos …

  {on(c1,pallet), on(c1,r1), on(c1,c2), …, at(r1,l1), at(r1,l2), …}

… usa uma coleção de proposições (variáveis boleanas):

  {on-c1-pallet, on-c1-r1, on-c1-c2, …, at-r1-l1, at-r1-l2, …}

Representação baseada em Teoria de Conjuntos

- Ao invés de um operador como esse:

take(crane1,loc1,c3,c1,p1)
;; crane 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)

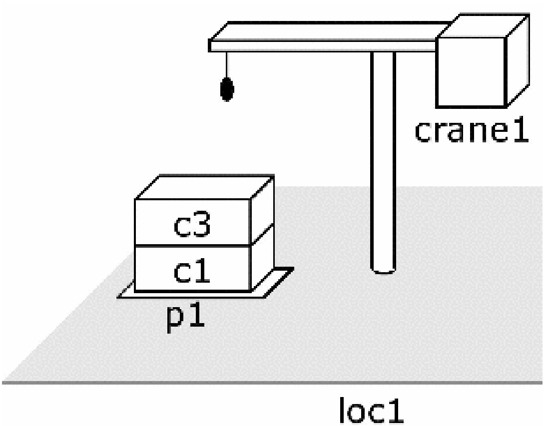
- … temos várias ações como essa:

take-crane1-loc1-c3-c1-p1
precond: belong-crane1-loc1, attached-p1-loc1,
empty-crane1, top-c3-p1, on-c3-c1
delete: empty-crane1, in-c3-p1, top-c3-p1, on-c3-p1
add: holding-crane1-c3, top-c1-p1

- Explosão exponential
  - Se um operador clássico contém *n* átomos, cada um com aridade *k*, então ele corresponde a c^{nk} ações onde c = |{símbolos constantes}|

Representação de Variáveis de Estado

- Uma variável de estado é como um campo em uma estrutura de registros



load(c, r, l)
;; robot r loads container c at location l
precond: rloc$(r) = l$, cpos$(c) = l$, rload$(r) =$
effects: rload$(r) \leftarrow c$, cpos$(c) \leftarrow r$

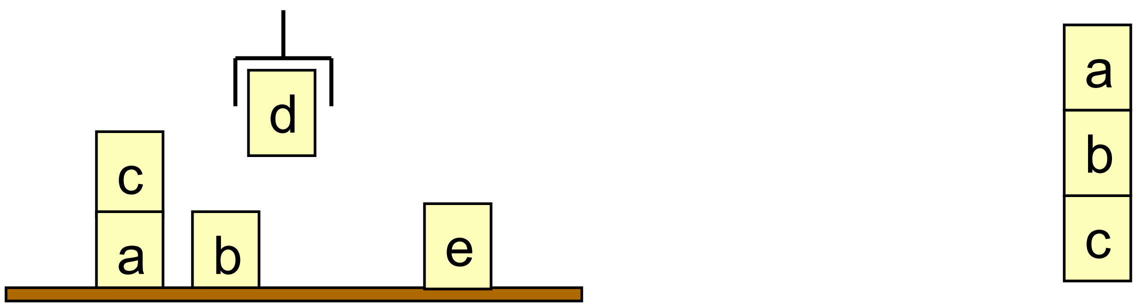
unload(c, r, l)
;; robot r unloads container c at location l
precond: rloc$(r) = l$, rload$(r) = c$
effects: rload$(r) \leftarrow$ nil, cpos$(c) \leftarrow l$

{top(p1)=c3, cpos(c3)=c1,
cpos(c1)=pallet, …}

- Representações clássica e de variáveis de estado consomem espaços similares
  - Cada uma pode ser traduzida para a outra em tempo polinomial de baixa ordem

Exemplo: O Mundo dos Blocos

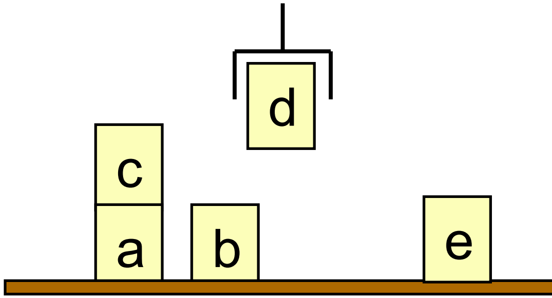
- Mesa infinitamente larga, número finito de blocos de criança
- Ignora a posição em que um bloco está sobre a mesa
- Um bloco pode estar sobre a mesa ou sobre um outro bloco
- Os blocos devem ser movidos de uma configuração para outra
  - e.g.,



estado inicial   estado meta

- Pode ser expresso como um caso especial de DWR, porém sua formulação é mais simples
- Daremos as formulações: clássica, teoria de conjuntos e variáveis de estado, para o caso de existirem 5 blocos.

Representação Clássica: Símbolos



- Símbolos constantes:
  - Os blocos: a, b, c, d, e
- Predicados:
  - ontable(*x*) - bloco *x* está sobre a mesa
  - on(*x*,*y*) - bloco *x* está sobre o bloco *y*
  - clear(*x*) - bloco *x* não tem nada sobre ele
  - holding(*x*) - a garra do robô está segurando o bloco *x*
  - handempty - a garra do robô não está segurando nada

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*)

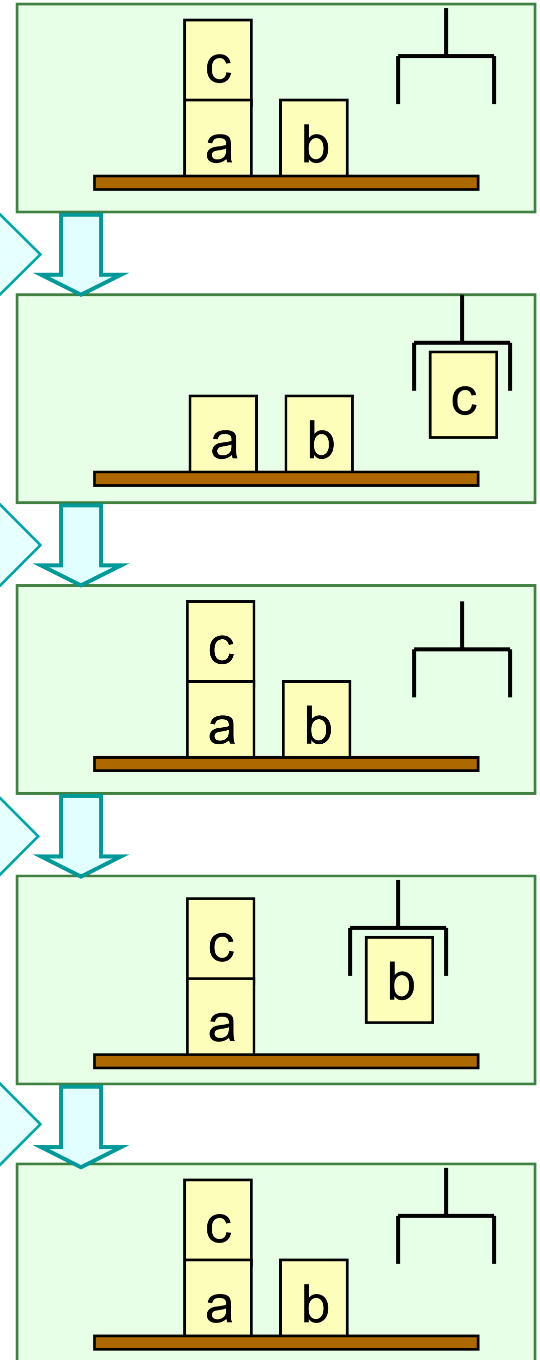
Precond: ontable(*x*), clear(*x*), handempty

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

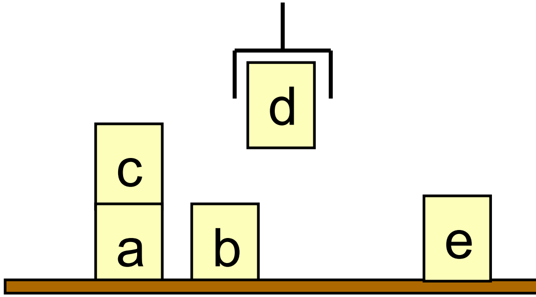
putdown(*x*)

Precond: holding(*x*)

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



Representação baseada em Teoria de Conjuntos: Símbolos



- Para 5 blocos, há 36 proposições
- Aqui estão 5 delas:

ontable-a	- o bloco a está na mesa
on-c-a	- o bloco c está sobre o bloco a
clear-c	- o bloco c não possue nada sobre ele
holding-d	- a garra do robô está segurando o bloco d
handempty	- a garra do robô não está segurando nada

Ações de Teoria de Conjuntos

50 ações
diferentes.

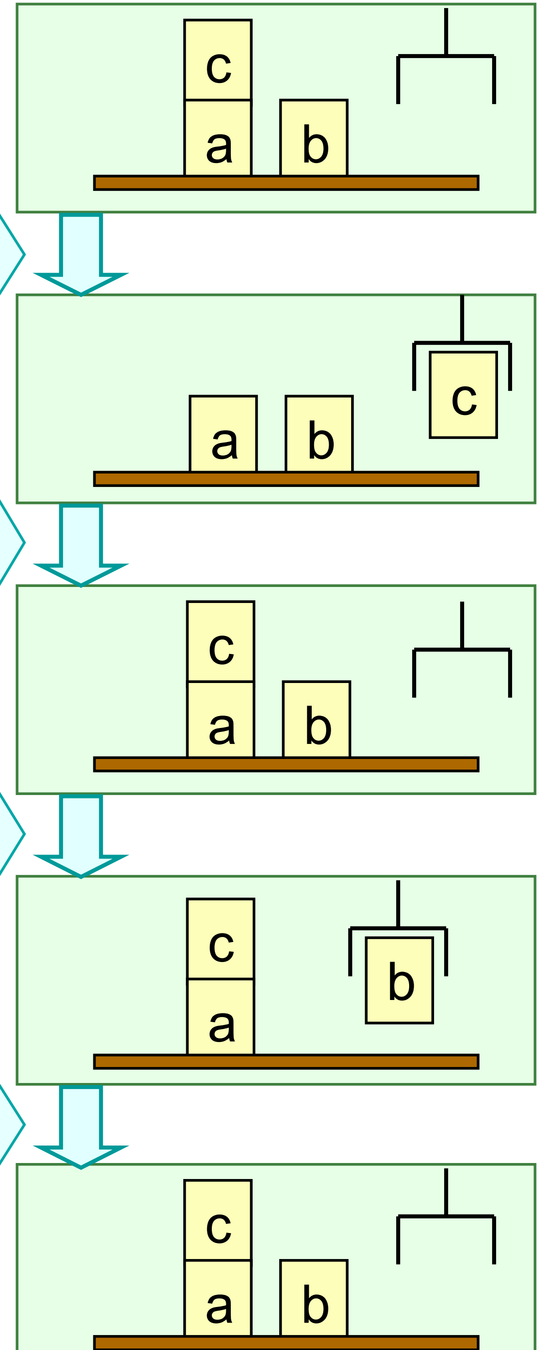
Aqui estão
4 delas:

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

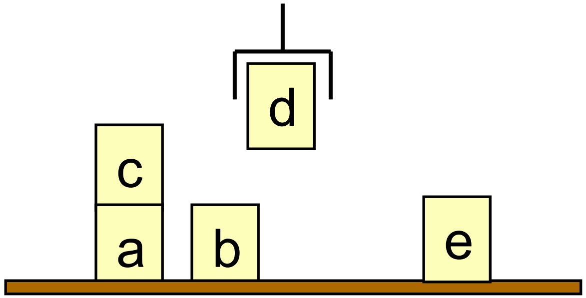
stack-c-a
Pre: holding-c, clear-a
Del: holding-c, ~clear-a
Add: on-c-a, clear-c, handempty

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



Representação de variáveis de estado: Símbolos



- Símbolos constantes:
  - a, b, c, d, e — do tipo bloco
  - 0, 1, table, nil — de outros tipos
- Variáveis de estado:
  - pos(*x*) = y — se bloco *x* está sobre o bloco *y*
  - pos(*x*) = table — se bloco *x* está sobre a mesa
  - pos(*x*) = nil — se bloco *x* está na garra do robô
  - clear(*x*) = 1 — se bloco *x* não tem nada sobre ele
  - clear(*x*) = 0 — se bloco *x* está na garra ou se tem outro bloco sobre ele
  - holding = *x* — se a garra do robô está segurando o bloco *x*
  - holding = nil — a garra do robô não está segurando nada

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)

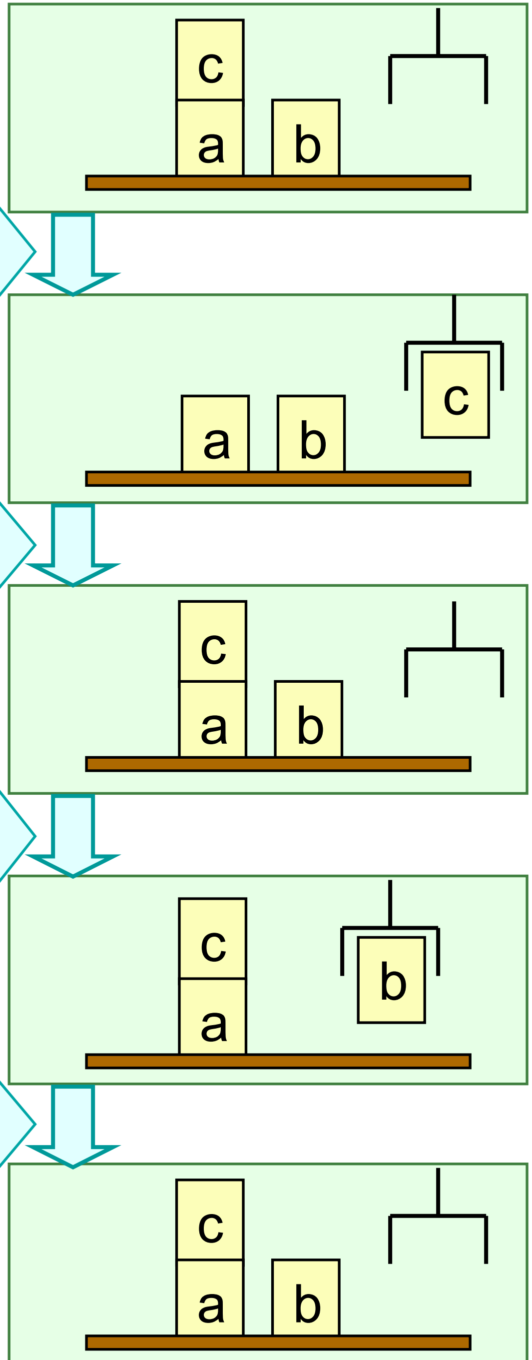
Precond: pos(*x*)=table, clear(*x*)=1, holding=nil

Effects: pos(*x*)=nil, clear(*x*)=0, holding=*x*

putdown(*x* : block)

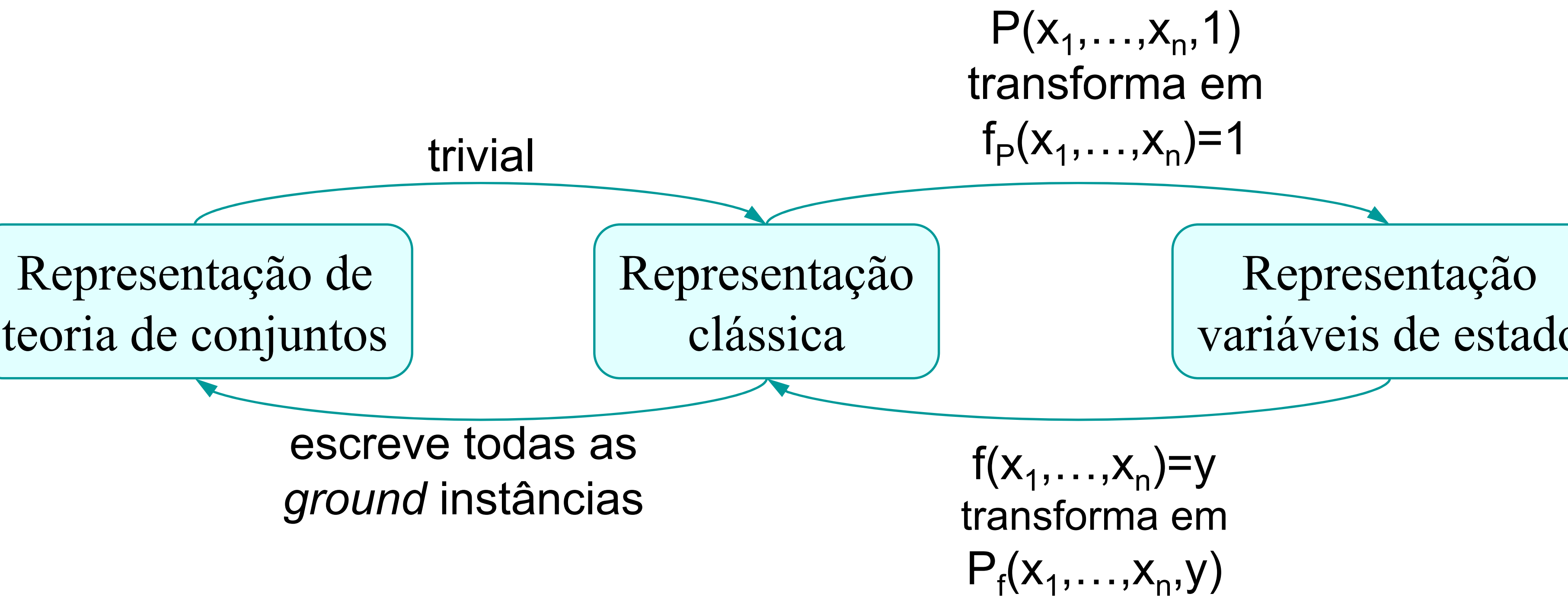
Precond: holding=*x*

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



Poder de expressão

- Qualquer problema que pode ser representado em uma representação pode ser representado nas outras duas
- Conversão em tempo e espaço linear, exceto para:
  - Conversão para teoria de conjuntos das outras duas representações: pode causar uma explosão combinatória



Comparação

- Representação clássica
  - A mais popular para planejamento clássico, em parte por razões históricas
- Representação de teoria de conjuntos
  - Consome muito mais espaço do que a representação clássica
  - Útil em algoritmos que manipulam diretamente *ground* átomos
    - » e.g., grafos de planejamento (Capítulo 6), satisfazibilidade (Capítulo 7)
  - Útil também para certos tipos de estudos teóricos
- Representação de variável de estado
  - Menos natural para os lógicos, mais natural para os engenheiros
  - Útil em problemas de planejamento não-clássicos como uma maneira de tratar números, funções e tempo
  - Útil na representação de MDPs fatorados para planejamento probabilístico (Capítulo 16)

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)))         )
```