Golog

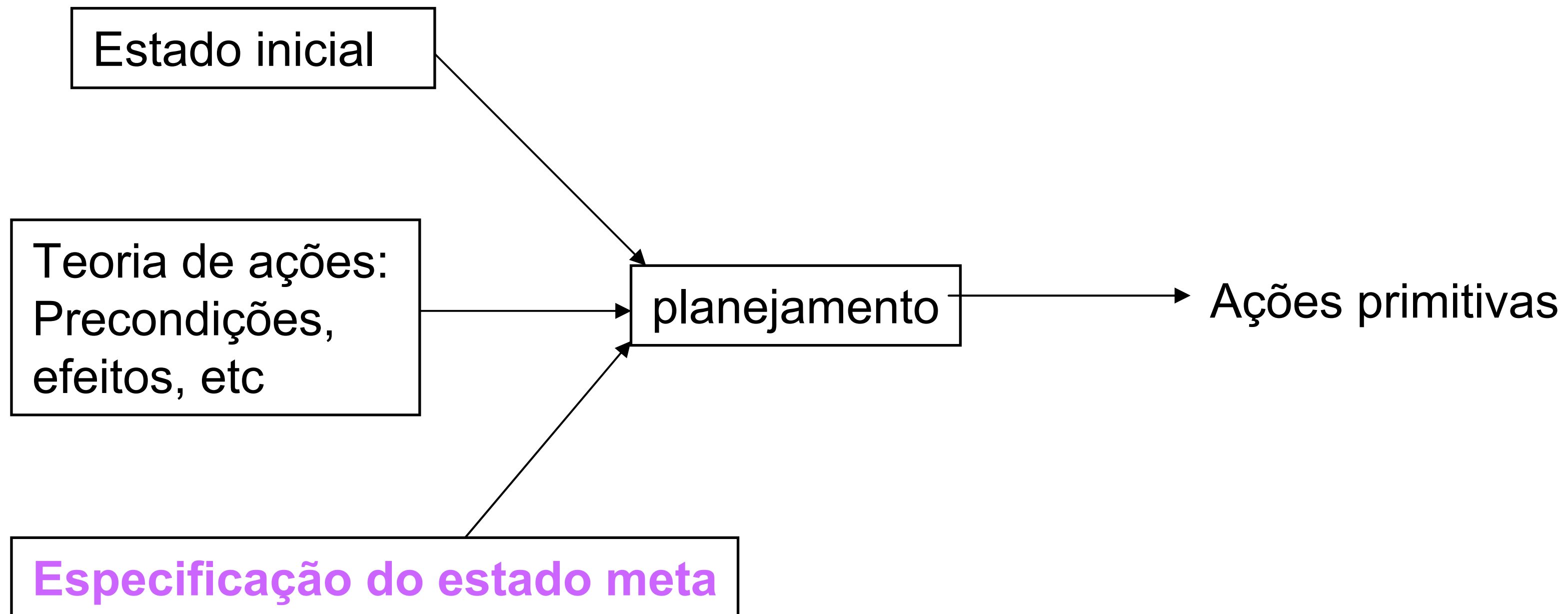
Uma abordagem para Robótica Cognitiva

Baseado

Robótica Cognitiva

- Estudo da representação de conhecimento e raciocínio de um agente autônomo em um ambiente dinâmico num mundo não completamente conhecido
- Trata problemas que envolvam:
  - A especificação de ações do agente
  - Ações exógenas,
  - sensoriamento,
  - informação incompleta,
  - conhecimento e crença do agente,
  - Monitoramento e execução de ãções,
  - ações complexas, concorrentes, etc

Planejamento clássico

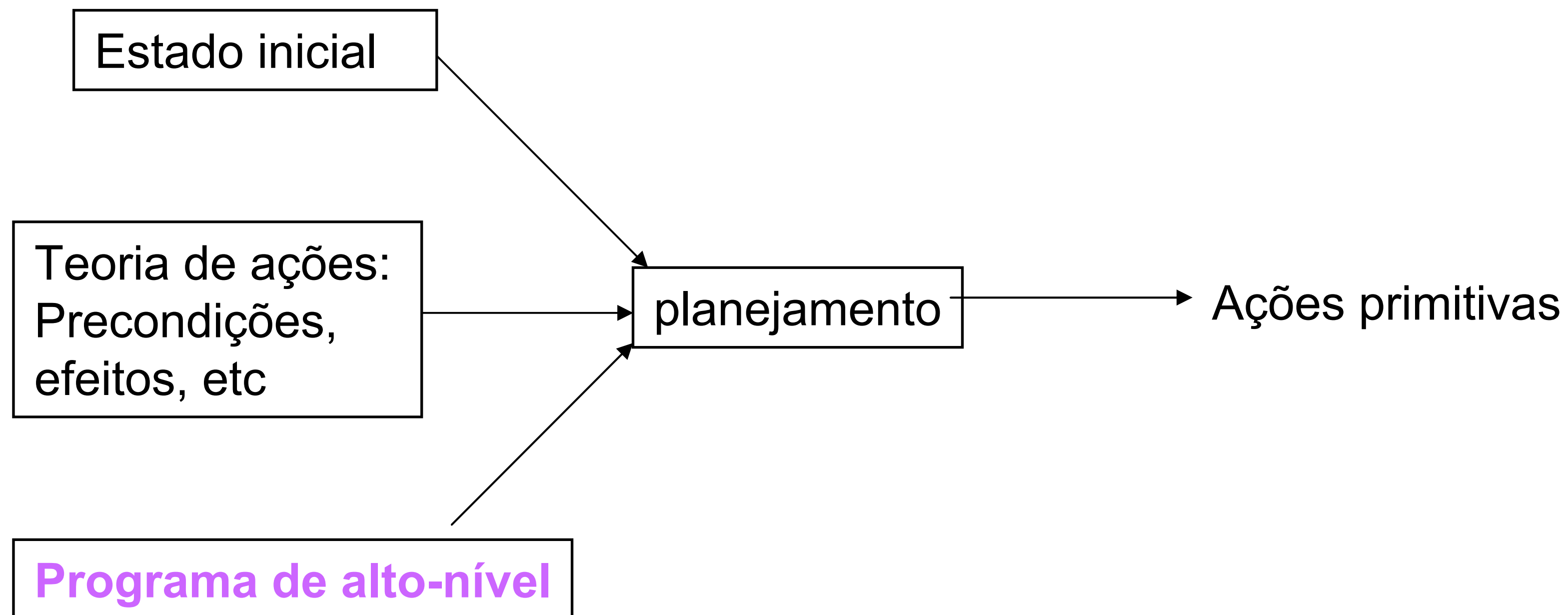


Solução: determinar uma sequencia de ações (primitivas) que pode
Ser executada no estado inicial produzindo o estado meta

Cálculo de Situações

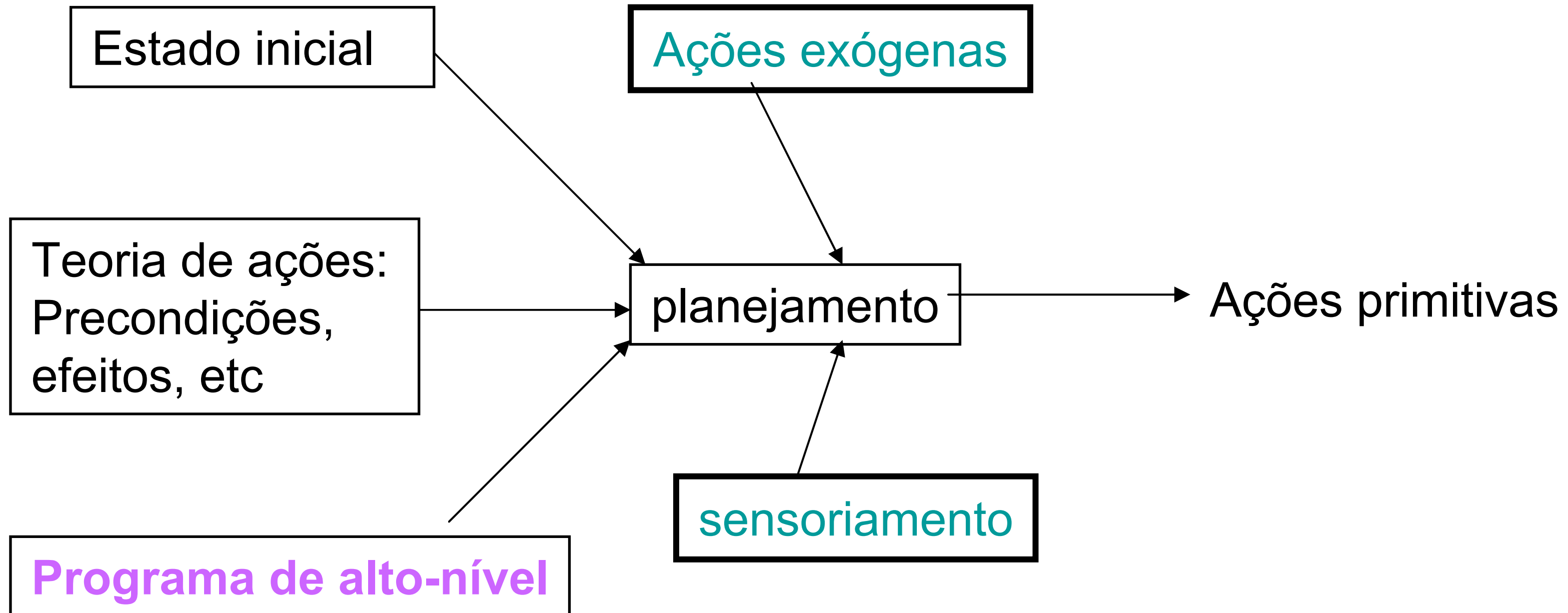
- Linguagem para descrever mudanças na Lógica de Primeira Ordem
- O que é representado:
  - Situações ou estados do mundo
  - Ações que mudam o estado do mundo, quando é possível que elas ocorram e quais são seus efeitos
  - Problemas de raciocínio sobre mudanças: *frame*, *ramification and qualification problem*

Planejamento como uma execução de um programa de alto-nível



Solução: determinar uma execução legal do programa a partir do estado inicial

Planejamento como uma execução de um programa de alto-nível



Solução: determinar uma execução legal do programa a partir do estado inicial, lenvando em conta as ações exógenas e o sensoriamento.

Definição de planos como programas de controle

- Para definir plano é introduzido um símbolo de função ; na linguagem de Cálculo de Situações, usado na notação infixa ***x* ; *y*** que significa composição seqüêncial de ações (*total-order*)
  - qualquer termo denotando uma ação é um plano
  - se π_1 é um plano e π_2 é um plano então $\pi_1 ; \pi_2$ é um plano

Definição de planos como programas de controle

- A execução de um plano é definida através de um predicado *trans*
- trans(π, s, s') é verdade se um plano π leva da situação s para a situação s'
- Seja α uma variável representando uma ação e π_1 e π_2 variáveis que representam planos. Então, podemos usar as seguintes abreviações:

  trans(α, s, s') significa Poss(α,s) $\wedge$ s' **=** do(α,s)

  trans(π_1;π_2, s, s') significa $\exists$ s''(*trans*(π_1, s, s'') $\wedge$ *trans*(π_2, s'', s')

Fórmulas *trans*: definição lógica para programas de controle

- *α é uma ação primitiva:*

  $trans(\alpha, s, s') = Poss(\alpha, s) \wedge s' = do(\alpha, s)$

- *δ é uma seqüência*

  $trans([\delta_1; \delta_2], s, s') = \exists s''. trans([\delta_1], s, s'') \wedge trans([\delta_2], s'', s)$

- δ é uma sentença condicional

  $trans([\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2], s, s') =$

  $\phi(s) \wedge trans(\delta_1, s, s') \vee \neg\phi(s) \wedge trans(\delta_1, s, s')$

-

Golog (Al**go**l in **logi**c)

- Golog é uma linguagem baseada no Cálculo de Situações que permite descrever um ***programa de controle de um robô***, ou seja, um comportamento de alto-nível pré-definido por um programador, como por exemplo: controle de processos ou automação industrial
- Criada pelo Grupo de Robótica Cognitiva da University of Toronto como um meta-interpretador Prolog
- Um programa Golog, quando executado, decompõe-se em uma seqüência de ações primitivas

Golog

- Trabalhos em Golog envolvem:
  - *Execução e sensoriamento* [Reiter, 2001][De Giacomo, 1999]
  - *Conhecimento incompleto do mundo* [De Giacomo, 1998]
  - *Raciocínio temporal* [Reiter, 1998]
  - *Sistemas multi-agentes* [Shapiro, 1997]

Programa Golog

Um programa Golog é composto por:

- *declarações de procedimentos* descritos na linguagem Golog
- *domínio* descrito como axiomas do Cálculo de Situações

Execução de um programa Golog

O que significa executar um programa GOLOG ?

- Encontrar uma seqüência de ações primitivas cuja execução em alguma situação inicial s resultando na situação s’ tornando a fórmula *trans*(δ, *s*, *s’*) verdadeira
- Gerar uma seqüência de ações para ser executada por um robô ou um simulador
- Para encontrar essa seqüência é necessário raciocinar sobre as ações primitivas (no Cálculo de Situações)

Execução de um programa Golog

Exemplo:

Seja a ação complexa:

δ ; **if** *holding(x,s)* **then** *B* **else** *C*

Para decidir entre executar B e C é preciso testar se o fluente *segura(x,s)* é verdadeiro após executar a ação δ

Linguagem Golog

$\phi?$	Condição de teste	testa se ϕ é verdadeiro
$\delta_1 : \delta_2$	Seqüência de ações	executa δ_1 e depois executa δ_2
$\delta_1 \# \delta_2$	Escolha não-determinística de ações	ou executa δ_1 ou executa δ_2
$\delta_1 \parallel \delta_2$	Execução concorrente de ações	executa $\delta_1 : \phi_2$ ou executa $\delta_2 : \delta_1$

If ϕ *then* δ_1 *else* δ_2

While ϕ *do* δ

Proc $P(x)$ δ *endProc* — definição de procedimentos

(Procedimentos da Aplicação

Linguagem Golog

- Como definir uma semântica para a linguagem baseada em lógica?
  - Fórmulas *Trans* (alguns trabalhos usam *Exec* ou *Do)*

Fórmulas *trans*

- Execução baseada em relações de transição de situações *trans(P, S, Pr, Sr).*

- *trans(P, S, Pr, Sr)* significa que dado um programa *P* e uma situação *S*, a execução de um passo do programa *P* a partir de *S*, leva a uma situação *Sr* com *P* passando à configuração *Pr* .

- Exemplo:

***P* = { *a1 : a2 : a3 : ... : an* }** **S = s0**

***Pr* = {*a2 : a3 : ... : an* }** ***Sr* = do(*a1, s0*)**

Fórmulas *trans*

- Nós queremos ser capazes de usar ações complexas compostas de ações primitivas e ainda "herdar" a solução para o problema da persistência ou quadro (*frame problem*)
- Para uma ação complexa δ podemos definir uma fórmula *trans*(δ, *s*, *s'*) que significa:
  - "A ação δ pode ser executada na situação *s* gerando a nova situação *s'* "

Meta-interpretador Prolog

- Quando o conhecimento sobre as ações e o estado inicial pode ser expresso através de cláusulas de Horn, essa avaliação pode ser fita diretamente em Prolog
- O interpretador GOLOG em Prolog possui cláusulas como essas

```
do(A,S1,do(A,S1)) :-
  prim_action(A), poss(A,S1).

do(seq(A,B),S1,S2) :-
  do(A,S1,S3), do(B,S3,S2).
```

Exemplo I

- Primitive actions: $pickup(x)$, $putonfloor(x)$, $putontable(x)$
- Fluents: $Holding(x,s)$, $OnTable(x,s)$, $OnFloor(x,s)$
- Action preconditions:

  $$Poss(pickup(x),s) \equiv \forall z.\neg Holding(z,s)$$
  $$Poss(putonfloor(x),s) \equiv Holding(x,s)$$
  $$Poss(putontable(x),s) \equiv Holding(x,s)$$

- Successor state axioms:

  $$Holding(x,do(a,s)) \equiv a = pickup(x) \vee$$
  $$Holding(x,s) \wedge a \neq putontable(x) \wedge a \neq putonfloor(x).$$
  $$OnTable(x,do(a,s)) \equiv a = putontable(x) \vee$$
  $$OnTable(x,s) \wedge a \neq pickup(x).$$
  $$OnFloor(x,do(a,s)) \equiv a = putonfloor(x)$$
  $$OnFloor(x,s) \wedge a \neq pickup(x).$$

Exemplo I (cont.)

- Situação Inicial:

  $\forall x, \neg holding(x, s_0)$

  $ontable(x, s_0) \equiv x = a \vee x = b$

- *Ações complexas:*

  proc *cleartable:* **while** $\exists y$. ontable(y) **do** πy*[ontable(y*)?; *removeblocck(y*)] **endProc**

  proc *removeblock(x):* *pickup(x);putonfloor(x)* **endProc**

Exemplo I (cont.)

- Para encontrar uma seqüência de ações que corresponda uma execução legal de um programa GOLOG, usamos a técnica de *resolução*.
- Por exemplo:

  KB |= ∃*S*. trans(*cleartable,* s_0, S)

O resultado dessa prova seria:

S= do(*putonfloor*(b), do(*pickup*(b), do(*putonfloor*(a), do(*pickup*(a), s_0)

Exemplo I (cont.)

- Que geraria a seguinte seqüência de ações:

pickup(a), *putonfloor*(a), *pickup*(b), *putonfloor*(b)

Exemplo II [Levesque *et al.* 1997]

proc *Control*
[**while** ($\exists n$) on(n) **do** *Serve_a_floor* **endWhile**]; *Park* **endProc**

proc *Serve(n)*
Go_floor(n); *Turnoff(n)*; *open*; *close* **endProc**

proc *Go_floor(n)*
(current_floor = n)? | *up(n)* | *down(n)* **endProc**

proc *Serve_a_floor*
(π *n*) [Next_floor(n)?; *Serve(n)*] **endProc**

proc *Park*
if current_floor = 0 **then** *open* **else** *down(0)*; *open* **endIf endProc**