Planejamento em
Inteligência Artificial

Capítulo 12
Estratégias de Controle em Planejamento Dedutivo

Leliane Nunes de Barros

MAC 5788 - IME/USP

segundo semestre de 2005

Planejamento Dedutivo

- O problema de planejamento é visto como um problema de prova de teoremas.
- Esse capítulo do livro, *Automated Planning,* só fala de dedução, considerada uma inferência lógica "natural" para planejamento no espaço de estados.
- No entanto, faltou abordar o Planejamento Abdutivo como foi proposto por Shanahan como sendo a inferência lógica "natural" para planejamento no espaço de planos.

Planejamento Clássico Versus Planejamento Dedutivo

Planejamento Clássico (não-hierárquico)	Planejamento Dedutivo
Ações são especificadas por triplas de precondições, efeitos positivos e efeitos negativos	Ações são especificadas por fórmulas lógicas
Transições de estados são computadas adicionando-se os efeitos positivos e eliminando os efeitos negativos	Transições de estados são computadas usando dedução
Geração de planos é feita através de busca no espaço de estados ou de planos	Geração de planos é feita através de dedução
Vantagem: técnicas específicas de busca (e.g., Graphplan) são mais eficientes que planejamento dedutivo	Vantagem: expressividade da lógica ➔dependendo da lógica usada muitas das restrições do planejamento clássico podem ser relaxadas

Planejamento Dedutivo: gargalo

- Faltam procedimentos automáticos para a geração de planos. Algumas soluções para esse problema são:
  - limitar a expressividade de lógica para casos especiais em que existem procedimentos eficientes, por exemplo, *planejamento como satisfazibilidade*;
  - manter a expressividade da lógica mas permitir que o usuário escreva estratégias de controle que reduzam o espaço de busca da prova. Nesse caso, existem dois paradigmas importantes que vermos nessa aula:
    - » **Planos como Programas**
    - » **Táticas**

Planos como Programas

- O usuário não especifica um estado meta mas escreve um programa em uma linguagem lógica que é uma especificação incompleta possível de um plano
- O usuário pede para um provador de teoremas encontrar um plano por dedução
- O provador de teoremas deve encontrar uma prova apenas para as partes do plano não totalmente especificadas, o que reduz a busca por uma prova

Estado Meta	Planos (programas) meta
o robô deve estar carregado na localização de destino	**mova o robô para uma localização intermediária, encontre um plano para carregar o robô e mova o robô para a localização de destino**

Táticas

- Táticas são programas definidos pelo usuário que especificam quais regras de inferência devem ser usadas e como elas devem ser combinadas para gerar uma prova
- O usuário especifica um estado meta e então escreve uma *tática* para guiar o processo de busca para uma prova para aquela meta

Planejamento como Programas

Cálculo de Situações

Cálculo de Situações Versus Planejamento Clássico

	Estado s_1	Estado s_2
Planejamento Clássico	at(r1,l1) loaded(r1,c1) in(c1,l1)	at(r1,l2) loaded(r1,c1) in(c1,l2) ¬at(r1,l1) ¬in(c1,l1)

Cálculo de Situações

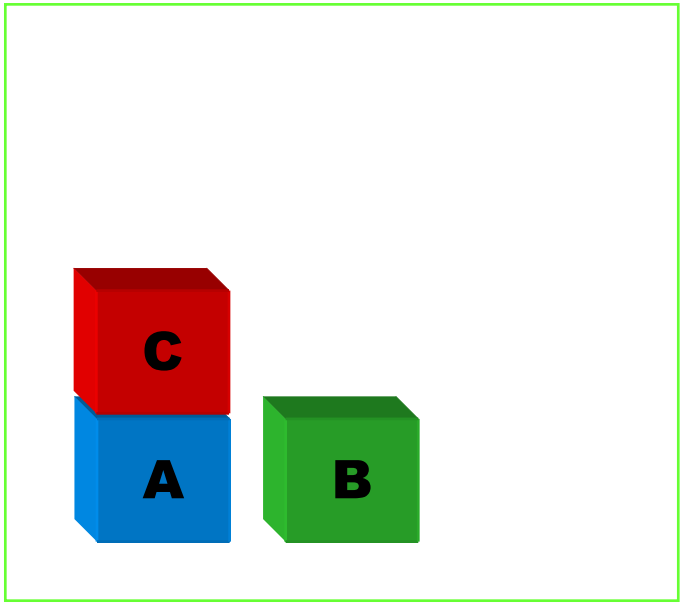
$$at(r1,l1,s1) \wedge loaded(r1,c1,s1) \wedge in(c1,l1,s1) \wedge$$
$$at(r1,l2,s2) \wedge loaded(r1,c1,s2) \wedge in(c1,l2,s1) \wedge$$
$$\neg at(r1,l1,s2) \wedge \neg in(c1,l1,s2)$$

Cálculo de Situações

- Linguagem de primeira ordem para representar estados e ações.
- Ontologia: situações, fluentes e ações
- Linguagem:
  - s_0
  - $do(\alpha , s)$
  - $poss(\alpha ,s)$
  - $holds(f , s)$ (no livro ➔ $f(\vec{x}, s)$)

Exemplo: Mundo dos Blocos

- fluentes:
  $clear(X)$
  $ontable(X)$
  $on(X,Y)$
- ações:
  $stack(X,Y)$
  $unstack(X,Y)$
  $move(X,Y,Z)$



Axiomas do Cálculo de Situações

- 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:

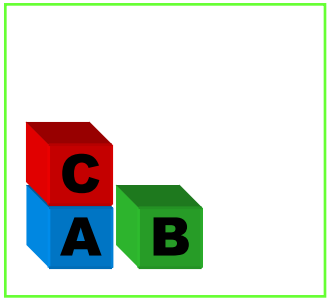
$poss(move(X,Y,Z),S) \leftarrow$

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

...

- axioma de persistência (*frame axiom*):

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



Planejamento Dedutivo

- Dados:
  - A : axiomatização do domínio
  - I : situação inicial
  - G : meta de planejamento

- O planejamento consiste em provar que

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

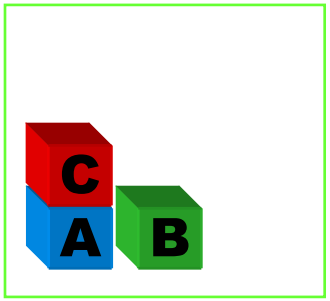
- sendo executabilidade definida indutivamente por:

$$exec(s0).$$

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

Planejador Dedutivo 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)).

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

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



Consultas ao planejador

?- 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

Definição clássica de planos em Cálculo de Situações

- Um planos é uma situação (!) composta por um aninhamento do símbolo de função *do*. Exemplo:

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

  Assim, planejamento consiste em:

$$A \wedge I \models \exists S[exec(S) \wedge G(S)]$$

Definição de planos em Cálculo de Situações (livro Automated Planning)

- 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
- A execução de um plano é definida através de um predicado Exec
- Exec(π, s, s') é verdade se um plano πleva 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:

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

  Exec($\pi_1;\pi_2$, s, s') significa $\exists$ s''(Exec(π_1, s, s'') $\wedge$ Exec(π_2, s'', s')

Definição de Problema em Cálculo de Situações

- Um problema de planejamento é dado pela tripla $(D, Ax_0, \Phi_g(s))$
  - sendo D é um domínio de planejamento dado pelos axiomas de Cáculo de Situações
  - Ax_0 um conjunto de axiomas de estado inicial s_0
  - $\Phi_g(s)$ uma fórmula meta
- Uma solução para um problema de planejamento é um plano π, tal que:

$$\exists\, s\, (Exec(\pi, s_0, s) \wedge \Phi_g(s)))$$

Generalizações da definição de plano

- | é o operador de ***escolha não-determinística***
- * é o operador de ***iteração não-determinística***

if ϕ *then* π_1 *else* π_2 significa $\text{Exec}((\phi?; \pi_1)|(\neg \phi?; \pi_1), s, s')$

while ϕ *do* π significa $\text{Exec}((\phi?; \pi_1)^*;\neg \phi?, s, s')$

- Essa lógica de planos como programas é vista como uma linguagem de programação de ações de alto nível, usada na área de Robótica Cognitiva
- A linguagem Golog é parecida com essa linguagem mas adiciona outros aspectos, como por exemplo, concorrência, procedimentos e chamadas a procedimentos, execução de ações e ações de sensoriamento.

Planejamento como Programas:

Lógica Dinâmica

Lógica Dinâmica

- Como no Cálculo de Situações, estados são representados como uma única teoria lógica.
- Diferente do Cálculo de Situações, estados não são representados explicitamente na linguagem (isto é, fluentes com termo situação).
- Os átomos que valem em diferentes estados são representados por operadores modais que são *parametrizados* com ações ou planos.
- Se *a* é uma ação, então [*a*] e <*a*> são operadores modais que declaram o que é verdade depois de uma ação *a*.
- Exemplos:
  - <move(r1,l1,l2)> at(r1,l2) significa que at(r1,l2) é verdade no estado resultante da execução da ação move(r1,l1,l2);
  - [move(r1,l1,l2)] at(r1,l2) significa que at(r1,l2) é possivelmente verdade no estado resultante da execução da ação move(r1,l1,l2).

Planejamento Clássico, Cálculo de Situações e Lógica Dinâmica

	Estado s_1	Estado s_2
Planejamento Clássico	at(r1,l1) loaded(r1,c1) in(c1,l1)	at(r1,l2) loaded(r1,c1) in(c1,l2) ¬at(r1,l1) ¬in(c1,l1)

Cálculo de Situações

at(r1,l1,s1) ∧ loaded(r1,c1,s1) ∧ in(c1,l1,s1) ∧
at(r1,l2,s2) ∧ loaded(r1,c1,s2) ∧ in(c1,l2,s1) ∧
¬at(r1,l1,s2) ∧ ¬in(c1,l1,s2)

Lógica Dinâmica

at(r1,l1) ∧ loaded(r1,c1) ∧ in(c1,l1) ∧
<move(r1,l1,l2)>(at(r1,l2) ∧ in(c1,l2,s1)) ∧
loaded(r1,c1)

Lógica Dinâmica

- Descreveremos a um fragmento simples da Lógica Dinâmica
  - Linguagem
  - Semântica
  - Axiomas e regras de inferência (dedutiva)
  - Domínios de planejamento, problemas e soluções
  - Extensões de planos totalmente ordenados para programas

Lógica Dinâmica: a linguagem

- O conjunto de planos Π inclue o conjunto de ações básicas Π_0 que correspondem a planos com uma única ação
- Para cada par de planos de π_1 e π_2 , o plano $\pi_1 ; \pi_2$ é também um plano em Π
- O conjunto de fórmulas Φ é definido indutivamente começando com um conjunto de proposições atômicas Φ_0 e o conjunto de planos Π
  - True e False são fórmulas em Φ
  - $\Phi_0 \subseteq \Phi$
  - Se p e q são fórmulas em Φ então $\neg p$ e $p \wedge q$ são fórmulas em Φ
  - Se p é uma fórmula em Φ e π é um plano em Π , então $< \pi > p$ é uma fórmula em Φ

Um plano como parâmetro do operador modal <>

Lógica Dinâmica: a linguagem

- A fórmula $< \pi > p$ declara que p é verdade em pelo menos um dos estados ***possíveis*** que resultam da execução de π . Exemplo:

  <move(r1,l1,l2);load(r1)>(at(r1,l2) $\wedge$ loaded(r1))

- Seja $[\ \pi]\ p$ a abreviação de $\neg < \pi > \neg\ p$ então $[\ \pi]\ p$ significa que p vale ***necessariamente*** em todos os estados resultantes da execução de π

- Dessa forma, Lógica Dinâmica pode representar ações não-determinísticas. Veremos como ações e planos podem ser não determinísticos e ainda assim, ser possível fazer planejamento.

Lógica Dinâmica: a semântica

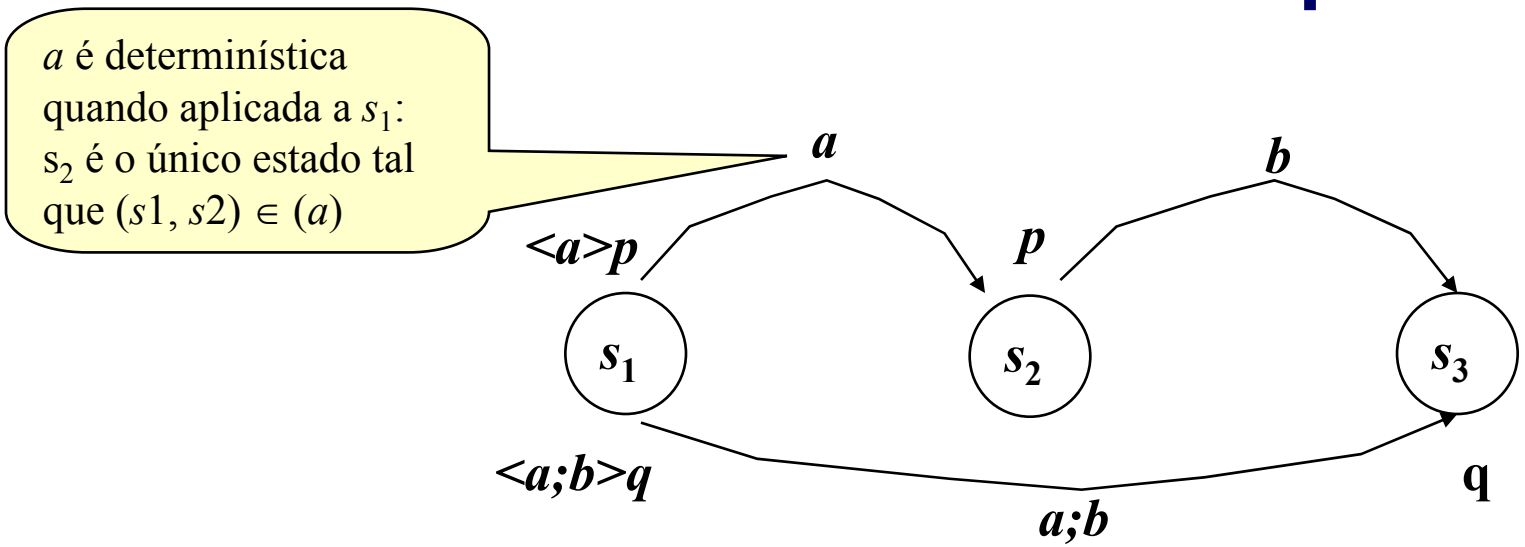
- A semântica de planos é dada pela função ρ: $\Pi \rightarrow 2^{SxS}$, que mapeia um plano π em um conjunto de pares de estados (s, s').
- Uma ação a é dita ser *aplicável* a um estado s se existe um estado s' tal que (s, s') $\in \rho(a)$. A concatenação seqüêncial de duas ações, a1;a2 , é aplicável no estado s se (s, s') $\in \rho(a1)$ e (s', s'') $\in \rho(a2)$.
- Um plano π é dito ser *aplicável* a um estado s se existe um estado s' tal que (s, s') $\in \rho(\pi)$. Intuitivamente:

$$\rho_s(\pi 1;\ \pi 2) = \{(s,t) \in\ SxS \mid \exists\ s' \in S.\ (s,s') \in \rho(\pi 1) \wedge (s',t) \in \rho(\pi 2)\}$$

- Seja a proposição p vedadeira no estado s ➔ $p \in s$
- O conjunto de estados em que $< \pi > p$ é verdadeiro é dado por:

$$\tau(< \pi > p\ ) = \{s \in\ S \mid \exists\ s'.(s,s') \in \rho(\pi) \wedge p \in s'\}$$

Semântica das fórmulas e planos



- Seja $<a>p$ verdadeiro no estado s_1, isto é, é verdade que a proposição p é verdadeira em pelo menos um estado resultante da execução da ação a, nesse caso, o estado s_2 então p é verdadeiro no estado s_2
  - se $s_1 \in \tau(<a> p)$ e a ação a leva do estado s_1 para o estado s_2, isto é, $(s_1, s_2) \in \rho(a)$ então $p \in s_2$ (se existissem mais estados $(s_1, s_2) \in \rho(a)$ então $p \in$ a pelo menos um deles
- O mesmo para planos:
  - Se $<a;b>q$ é verdadeiro no estado s_1, isto é, é verdade que a proposição q é verdadeira em pelo menos um estado resultante da execução da ação $a;b$, nesse caso, o estado s_3. Se a leva do estado s_1 para o estado s_2 e b leva do estado s_2 para o estado s_3 então p é verdadeiro no estado s_3

Interpretação de uma fórmula lógica

- Uma *interpretação* é uma tripla $M = (S, \tau, \rho)$.
- A fórmula $p \in \Phi$ é válida em uma interpretação $M = (S, \tau, \rho)$ (i.e., $M \models p$) sse $\tau(p) = S$; p é válido (i.e., $\models p$) se for é válido em todas as interpretações de p.

Maquina de dedução

- Dedução em Lógica Dinâmica é baseada em um conjunto de axiomas e um conjunto de regras de inferências.
- Axiomas:
  - Qualquer axioma do cálculo de proposicional é um axioma da Lógica Dinâmica
  - Além disso, incluimos os seguintes axiomas:

$$<\pi_1 ; \pi_2> p \leftrightarrow \ < \pi_1> (< \pi_2> p)$$

$$<\pi> (p \vee q) \leftrightarrow (<\pi> p \vee <\pi> q)$$

- Regras de inferência:
  - De p, $p \rightarrow q$, podemos inferir q (modus ponens)
  - De p podemos inferir $\neg < a> \neg p$. Essa regra, chamada de *necessária*, declara que se p é verdadeira em todos os estados então a negação de p não pode valer em qualquer estado alcançável pela ação a.
  - Se uma fórmula p é inferida dos axiomas através as regras de inferência, dizemos que ela é provável (i.e., $\vdash p$). Se p é derivada da fórmula q escrevemos : $p \vdash q$

Domínios de planejamento, problemas e soluções

- Um domínio de planejamento é uma tripa (Φ_0 , Π_0, $Ax_{\Pi 0}$), onde Φ_0 é o conjunto das proposições atômicas, Π_0 é o conjunto das ações básicas e $Ax_{\Pi 0}$ é a união dos três conjuntos disjuntos:
  1. Um conjunto de fórmulas proposicionais chamadas de *axiomas de domínio*
  2. Um conjunto de fórmulas chamadas de *axiomas de ações* que descrevem as precondições e efeitos de ações, do tipo $p \rightarrow [a]q$
     - a é qualquer ação básica de Π_0 , p e q são fórmulas proposicionais. Chamamos de p de precondição e q de efeitos da ação a
  3. *Axiomas de Frame*, que descrevem quais ações não mudam a proposições

Extenções

- Assim como Golog PDL (Propositional Dynamic Logic) pode expressar planos com estruturas de controle como condicionais e laços, definidas com base em escolhas não-determinísticas, testes e repetições

Extenções

- Planos e fórmulas são definidos indutivamente como:
  - True e False são fórmulas em Φ, $\Phi_0 \subseteq \Phi$
  - Se p e $q \in \Phi$, então $\neg p \in \Phi$ e $p \wedge q \in \Phi$
  - Se π_1 e π_2 são planos em Π , então
    - » $\pi_1; \pi_2$ é um plano em Π (seqüência)
    - » $\pi_1 \cup \pi_2$ é um plano em Π (escolha não determinística)
    - » π^* é um plano em Π (repetição não determinística)
  - Se $p \in \Phi$, então $p? \in \Phi$ (teste)
  - Se $\alpha \in \Pi$, então $<\alpha>p \in \Phi$
- Definição de condicionais e laços:

$$\textit{if } p \textit{ then } \pi_1 \textit{ else } \pi_2 \quad \equiv \quad p?; \pi_1 \cup (\neg p)?; \pi_2$$

$$\textit{while } p \textit{ do } \pi \quad \equiv \quad (p?; \pi)^* ; (\neg p)?$$

Tactical Theorem Proving

- Programas definidos pelo usuário sobre quais regras de inferência devem ser usada, e em que ordem
  - **Táticas primitivas, exemplo**:

```
Modus-ponens-tac(Φ1, Φ2)          ;; primitive tactic for modus ponens
   if premise(Φ1, Φ2)             ;; rule precondition
      then return conclusion(Φ2)  ;; rule application
      else exit with "failure"    ;; failure generation
end
```

  - **Táticas compostas:**
    - » then(tac1,tac2). Aplique tac1. Se tac1 falha, então tac2 falha, caso contrário, aplique tac2.
    - » orelse(tac1,tac2). Aplique tac1. Se tac1 falha, então aplique tac2.
    - » Try(tac1). Aplique tac1. Se tac1 falha, então devolva a fórmula original.
    - » Repeat(tac1). Aplique ta1 até falhar.