

PLANEAMIENTO

CAP 10 (SEC 10.1-10.2)

Parcialmente adaptado de
<http://aima.eecs.berkeley.edu>
e de

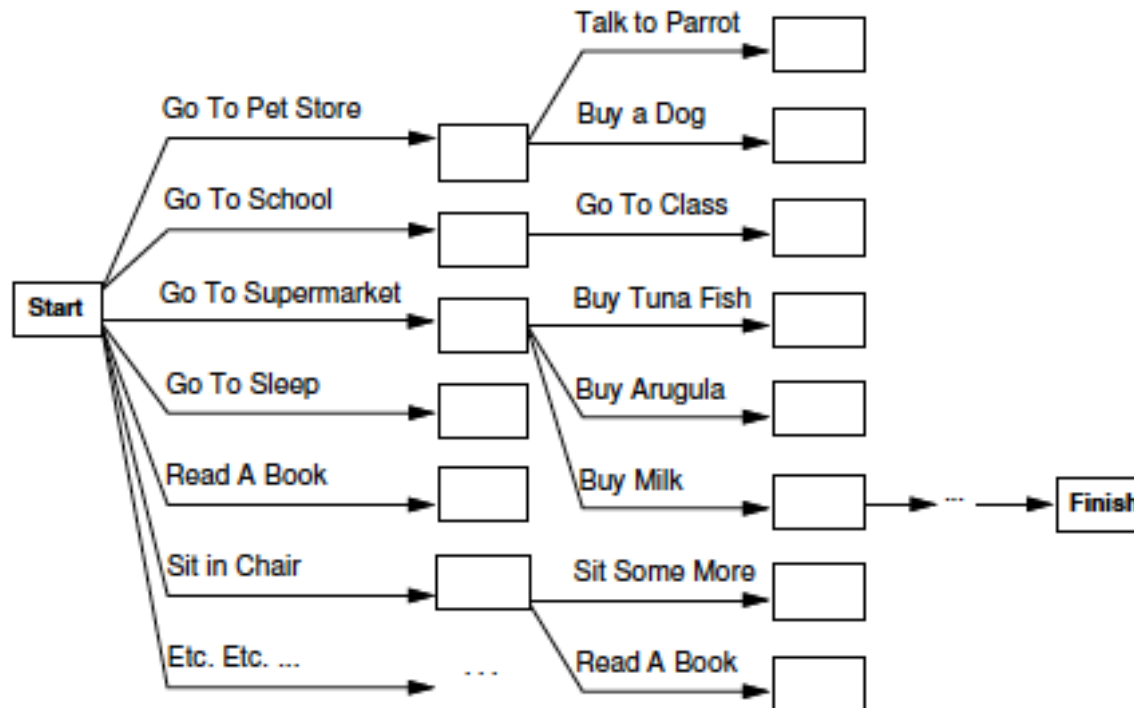
Dana Nau: Lecture slides for Automated Planning

Resumo

- Procura vs. Planeamento
- Planning Domain Description Language (PDDL)
- Algoritmos de Planeamento
 - Pesquisa em espaço de estados
 - Planeamento Progressivo
 - Planeamento Regressivo

Procura vs. Planeamento

- Considere-se a tarefa **comprar leite**, **bananas**, e um **berbequim**. Algoritmos de procura habituais aparentemente parecem desadequados:



- Heurística/teste à posteriori é problemático

Planeamento

- Geração de sequências de acções para efectuar tarefas e atingir objectivos:
 - Estados, acções, objectivos
- Procurar soluções sobre espaço abstracto de planos
- Muitas aplicações práticas:
 - Desenho e manufacturação
 - Jogos
 - Exploração espacial
 - Operações humanitárias
 - Operações militares
 - ...

Linguagem de planeamento

- Como expressar um problema de planeamento?
- A linguagem deve:
 - Ser suficientemente expressiva para permitir a representação de vários problemas
 - Ser suficientemente restritiva para permitir algoritmos eficientes
 - Permitir que os algoritmos tirem partido da estrutura do problema.
- Alternativas:
 - Lógica proposicional
 - requer demasiadas frases: e.g. no mundo do Wupus, a acção de mover uma casa requer uma frase para cada célula, direcção e tempo!
 - Lógica de primeira ordem
 - Demasiado expressiva. Dificuldade em expressar o que não muda como consequência de uma acção. Foi muito usada no passado.
 - Linguagens específicas para planeamento
 - PDDL, STRIPS, etc...

Linguagem PDDL (estados)

- A linguagem **PDDL** (Planning Domain Definition Language) pretende permitir a combinação da procura em espaço de estados com as representações lógicas.
- **Representação de estados**: O mundo é representado logicamente por intermédio de conjunções de literais positivos concretos (sem variáveis e símbolos de função). Adopta-se a **semântica de bases de dados**:
 - Mundo fechado (CWA): todos os fluentes não mencionados são falsos.
 - Nomes únicos: todas as constantes são diferentes.
 - Domínio fechado: só existem as constantes referidas na representação.
- **Estado inicial**: Conjunção de literais ground positivos.
- **Objectivos**: Conjunção de literais (positivos ou negativos).
- A PDDL é uma generalização da linguagem STRIPS.
 - A linguagem STRIPS não permite literais negativos nas pre-condições nem nos objectivos.

Linguagem PDDL (acções)

- Acção = Pré-condição + Efeitos
- **Acções**: representadas por **esquemas de acção** constituídos por um nome e lista de variáveis (e.g. Go(x,y)), precondições e efeitos, ambos uma conjunção de literais (positivos ou negativos).

Action(Fly(p,from, to),

PRECOND: At(p,from) $\wedge$ Plane(p) $\wedge$ Airport(from) $\wedge$ Airport(to)

EFFECT: $\neg$ AT(p,from) $\wedge$ At(p,to)

)

At(WHI,JFK),Plane(WHI),
Airport(JFK), Airport(SFA)

Fly(WHI,JFK,SFA)

At(WHI,SFA), $\neg$ At(WHI,JFK)

- As variáveis que aparecem em efeitos também devem ocorrer na precondição.
- Uma acção é **aplicável** num estado s se as precondições são satisfeitas por s

$$a \in \text{Actions}(s) \Leftrightarrow s \models \text{Precond}(a)$$

- Um esquema de acção a pode assim ter várias instâncias.

Linguagem PDDL (semântica de conjuntos)

- Normalmente é mais fácil usar a **semântica de conjuntos**:
 - estados são conjuntos de átomos positivos concretos (ground) manipulados por operações de conjuntos.
- O resultado de executar uma instância de uma acção a num estado s é o estado

$$Result(a,s) = (s - Del(a)) \cup Add(a)$$

- em que a lista (na realidade é um conjunto) de remoções $Del(a)$ é formada pelos literais negativos no efeito de a , e a lista de adições $Add(a)$ é o conjunto de literais positivos no efeito de a .
- Um plano é uma sequência de acções PDDL proposicionalizadas que leva do estado inicial ao estado final.

Assunções da Linguagem PDDL

- Estados são conjuntos de literais positivos
- Os literais que não ocorrem num estado assumem-se falsos (princípio do mundo fechado – CWA)
- Um efeito positivo adiciona o literal ao estado. Um efeito negativo remove o literal complementar do estado. O resto mantém-se inalterado para o estado seguinte.
- Fluentes não mencionam explicitamente o tempo. Em PDDL o tempo e o estado é implícito nos esquemas de acções: a precondição refere-se sempre ao tempo t e o efeito ao tempo $t + 1$ imediatamente seguinte.

Exemplo: transporte aéreo de mercadoria

$Init(At(C1, SFO) \wedge At(C2, JFK) \wedge At(P1, SFO) \wedge At(P2, JFK) \wedge$
 $Cargo(C1) \wedge Cargo(C2) \wedge Plane(P1) \wedge Plane(P2) \wedge Airport(JFK) \wedge$
 $Airport(SFO))$

$Goal(At(C1, JFK) \wedge At(C2, SFO))$

$Action(Load(c, p, a))$

$PRECOND: At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$

$EFFECT: \neg At(c, a) \wedge In(c, p)$

$Action(Unload(c, p, a))$

$PRECOND: In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$

$EFFECT: At(c, a) \wedge \neg In(c, p)$

$Action(Fly(p, from, to))$

$PRECOND: At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

$EFFECT: \neg At(p, from) \wedge At(p, to)$

Exemplo: pneu sobresselente

Init(*At*(Flat, Axle) $\wedge$ *At*(Spare, trunk))

Goal(*At*(Spare, Axle))

Action(*Remove*(Spare, Trunk))

PRECOND: *At*(Spare, Trunk)

EFFECT: $\neg$ *At*(Spare, Trunk) $\wedge$ *At*(Spare, Ground))

Action(*Remove*(Flat, Axle))

PRECOND: *At*(Flat, Axle)

EFFECT: $\neg$ *At*(Flat, Axle) $\wedge$ *At*(Flat, Ground))

Action(*PutOn*(Spare, Axle))

PRECOND: *At*(Spare, Groundp) $\wedge$ $\neg$ *At*(Flat, Axle)

EFFECT: *At*(Spare, Axle) $\wedge$ $\neg$ *At*(Spare, Ground))

Action(*LeaveOvernight*

PRECOND:

EFFECT: $\neg$ *At*(Spare, Ground) $\wedge$ $\neg$ *At*(Spare, Axle) $\wedge$ $\neg$ *At*(Spare, trunk) $\wedge$ $\neg$ *At*(Flat, Ground) $\wedge$ $\neg$ *At*(Flat, Axle))

- Este exemplo vai para além de STRIPS pois tem pré-condições negativas.

Exemplo: mundo dos blocos

$Init(On(A, Table) \wedge On(B, Table) \wedge On(C, A) \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(B) \wedge Clear(C))$

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

$Action(Move(b, x, y))$

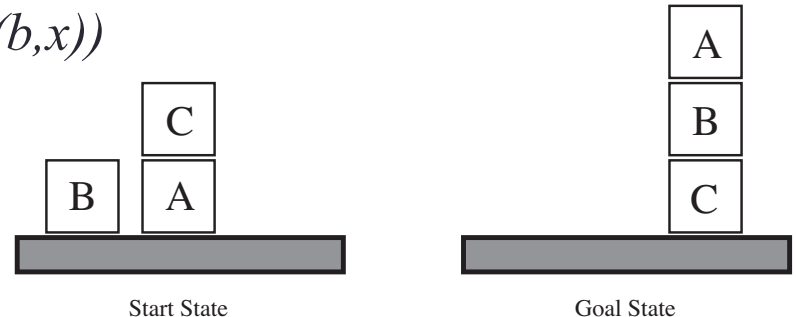
$PRECOND: On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge (b \neq x) \wedge (b \neq y) \wedge (x \neq y)$

$EFFECT: On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y)$

$Action(MoveToTable(b, x))$

$PRECOND: On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x)$

$EFFECT: On(b, Table) \wedge Clear(x) \wedge \neg On(b, x)$



Exemplo: Caparica

Init(
{Em(Fct), Com(Dinheiro), Praia(Caparica), Local(Caparica), Local(Fct)})

Action(Ir(x)
PRECOND: Em(y), Local(x)
EFFECT: Em(x), -Em(y))

Action(BanhoSol
PRECOND: Em(y), Praia(y)
EFFECT: Com(Bronze), Com(Sede), -Sem(Sede))

Action(BeberCerveja
PRECOND: Com(Sede), Com(Dinheiro)
EFFECT: -Com(Sede), -Com(Dinheiro), Sem(Sede))

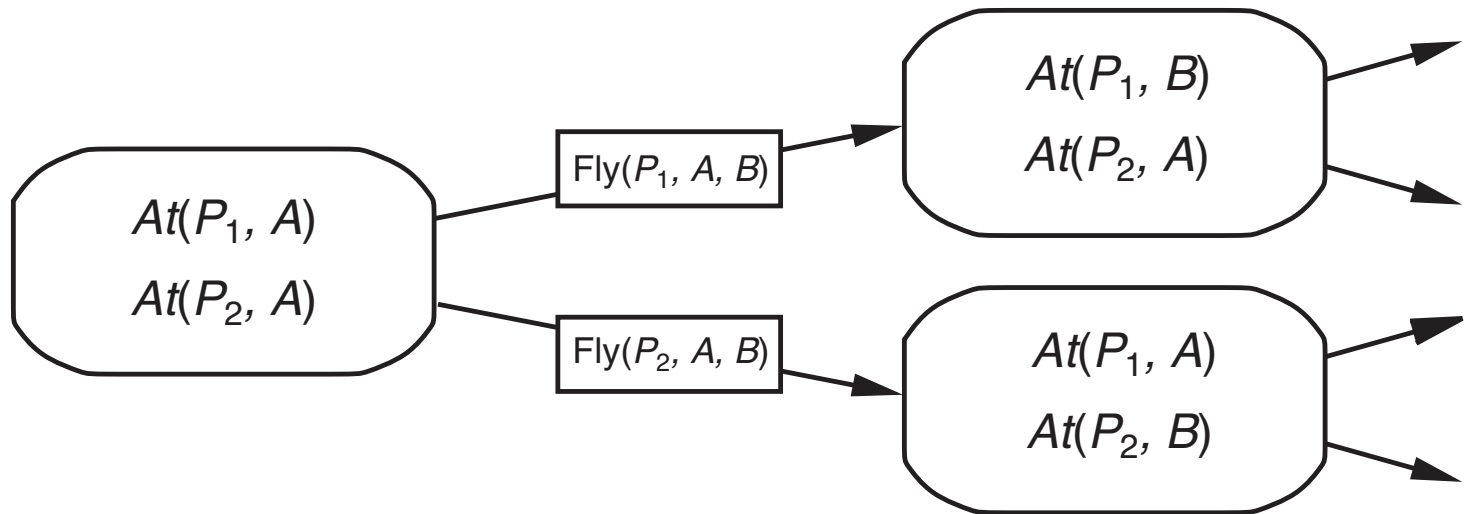
Goal(Com(Bronze), Sem(Sede))

Planeamento em espaços de estados

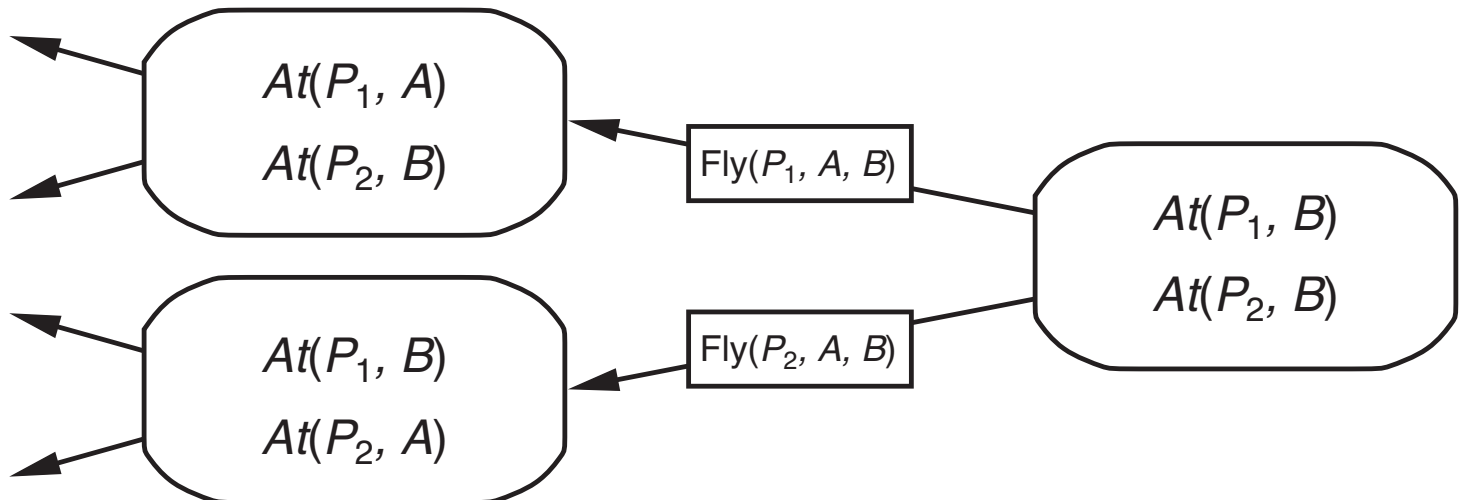
- Podem-se utilizar algoritmos de procura em espaço de estados para efectuar planeamento em dois sentidos:
- **Planeamento Progressivo**: Partir do estado inicial, aplicar os operadores PDDL até atingir um estado que torna os objectivos verdadeiros.
 - Considerar o efeito de todas as acções possíveis num dado estado
 - Utilização imediata dos algoritmos de procura analisados anteriormente, mas pode gerar muitos estados por utilização de acções irrelevantes...
- **Planeamento Regressivo**: Partir dos objectivos e utilizar os “operadores” ao contrario. Trabalha com estados incompletos.
 - Para atingir um objectivo, o que é que tinha que ter sido verdade no estado anterior.

Planeamento Progressivo e Regressivo

(a)



(b)



Algoritmo Progressivo

- Formulação como problema de pesquisa de espaço de estados:
 - **Estado inicial**: estado inicial do problema de planeamento
 - **Acções**: aquelas cujas pré-condições estão satisfeitas
 - Transição: adicionar efeitos positivos e remover negativos
 - **Objectivo**: o estado satisfaz o objectivo?
 - **Custo de acção**: cada tem um custo unitário.
- Sem funções, qualquer algoritmo de pesquisa em grafos completo é um algoritmo de planeamento completo
- Ineficiente:
 - Problema das acções irrelevantes
 - Requer boas heurísticas para ser eficiente.

Exemplo: planeamento progressivo

- Plano PDDL: $[Ir(Caparica), BanhoSol, BeberCerveja]$
- Execução do plano e mudanças de estado:
 - $\{Em(Fct), Com(Dinheiro), Praia(Caparica), Local(Caparica), Local(Fct)\}$
 $\Downarrow Ir(Caparica)$
 $\{Em(Caparica), Com(Dinheiro), Praia(Caparica), Local(Caparica), Local(Fct)\}$
 $\Downarrow BanhoSol$
 $\{Em(Caparica), Com(Dinheiro), Com(Bronze), Com(Sede), Praia(Caparica), Local(Caparica), Local(Fct)\}$
 $\Downarrow BeberCerveja$
 $\{Em(Caparica), Com(Dinheiro), Com(Bronze), Sem(Sede), Praia(Caparica), Local(Caparica), Local(Fct)\}$
- O plano PDDL $[Ir(Caparica), BanhoSol, BanhoSol, BeberCerveja]$ também é solução?
- E o plano PDDL $[Ir(Caparica), BanhoSol, BeberCerveja, BanhoSol]$?

Algoritmo Regressivo

- Como determinar os predecessores?
 - Quais os estados a partir dos quais executar uma dada acção leva ao objectivo?
 - Objectivo = $\{At(C1, B), At(C2, B), \dots, At(C20, B)\}$
 - Acção relevante para o primeiro elemento: $Unload(C1, p, B)$
 - Apenas funciona se as pré-condições estiverem satisfeitas
 - Estado anterior = $\{In(C1, p), At(p, B), At(C2, B), \dots, At(C20, B)\}$
 - Sub-objectivo $At(C1, B)$ não deve estar presente neste estado.
- Acções têm que ser consistentes: não desfazer literais desejados.
- Principal vantagem: apenas as acções relevantes são consideradas.

Algoritmo Regressivo

- Processo genérico de construção do predecessor:
 - Dada uma descrição do objectivo G
 - Seja A uma acção que é relevante e consistente
 - O predecessor é construído da seguinte forma:
 - Efeitos positivos de A são eliminados de G
 - Cada pre-condição de A é adicionada, se ainda lá não estiver
 - O predecessor torna-se no novo objectivo.
- Qualquer algoritmo de pesquisa standard pode ser usado para efectuar a pesquisa
- Termina quando o predecessor for satisfeito pelo estado inicial.
 - Poderá requerer unificação.

Exemplo: planeamento regressivo

- Construção do plano a partir dos objectivos:

$\{Com(Bronze), Sem(Sede)\}$

$\uparrow$ *BeberCerveja*

$\{Com(Dinheiro), Com(Bronze), Com(Sede)\}$

$\uparrow$ *BanhoSol*

$\{Em(y), Com(Dinheiro), Praia(y)\}$

$\uparrow$ *Ir(Caparica)*

$\{Em(x), Com(Dinheiro), Praia(Caparica), Local(x)\}$

(conclui-se do estado inicial fazendo $x=Fct$)

Heurísticas para planeamento em espaço de estados

- Quer o planeamento progressivo quer o planeamento regressivo requerem heurísticas apropriadas. Utiliza-se a técnica de relaxação de problemas.
- **Ignorar precondições**: toda a acção é aplicável em qualquer estado. Atenção que o número de passos para alcançar o objectivo não é o número de objectivos por satisfazer porque:
 1. uma acção pode satisfazer vários objectivos,
 2. uma acção pode desfazer o efeito de outra.
 - Para muitos problemas uma boa heurística considera 1 e ignora 2 relaxando as acções removendo das precondições e efeitos todos os literais que não estejam no estado objectivo. Contudo obter o número mínimo de acções para resolver o problema relaxado é o problema de SET-COVER que é NP-difícil...
- **ignorar lista de remoções**: removem-se todos os efeitos negativos aos operadores e utiliza-se um algoritmo de planeamento (mais simples...) para obter o número mínimo de acções necessárias. NP-difícil mas pode ser aproximado por hill-climbing.

Heurísticas para planeamento em espaço de estados

- As técnicas anteriores podem não ser suficientes, logo poderá ser necessário efectuar **abstracção dos estados**, sendo a mais simples ignorar alguns fluentes.
- Outras heurísticas utilizam **decomposição** para dividir o problema em partes, resolver cada parte independentemente e combinar as partes.
- **independência de subobjectivos**: assume que o custo de resolver uma conjunção de subobjectivos é aproximado pela soma do custo de resolver cada um dos subobjectivos independentemente.
 - Pode ser optimista (logo admissível) quando existem interacções negativas entre os subplanos.
 - Pode ser pessimista (logo não admissível) quando os subplanos têm acções redundantes.
- Seja G um conjunto de fluentes particionado em conjuntos disjuntos $G_1, \dots, G_n$ e $P_1, \dots, P_n$ os planos para os resolver. A heurística $\max_i \text{COST}(P_i)$ é admissível mas $\sum_i \text{COST}(P_i)$ já não. Mas se G_i e G_j forem independentes podemos utilizar $\text{COST}(P_i) + \text{COST}(P_j)$.