Planeamento

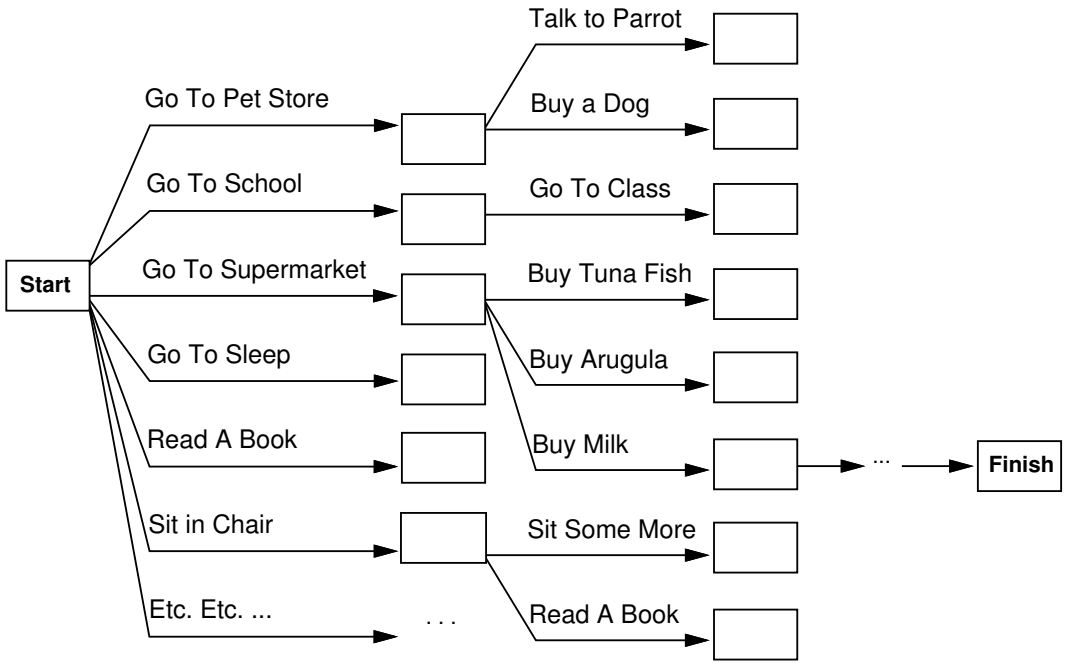
Ano Lectivo 2013/2014 - 1^o Semestre (Cap. 10) – Adaptado de http://aima.eecs.berkeley.edu/slides-tex/

Resumo

- ♢ Procura vs. planeamento
- ♢ Planeamento proposicional
- ♢ Cálculo de Situações
- ♢ Operadores PDDL e STRIPS
- ♢ Planeamento progressivo e regressivo
- ♢ Planeamento com ordem parcial

Procura vs. planeamento

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



Heurística/teste a posteriori é problemático

Procura vs. planeamento

Sistemas de planeamento:

1) abrem a representação das acções e do objectivo para permitir selecção
2) divisão-e-conquista por resolução de subobjectivos
3) não obriga à construção sequencial de soluções

	Procura	**Planeamento**
Estados	estruturas de dados Java	Frases lógicas
Acções	código Java	Precondições/resultados
Objectivo	código Java	Frase Lógica (conjunção)
Plano	Sequência a partir de situação inicial	Restrições sobre as acções

Planeamento Clássico

♢ Consideram-se apenas ambientes:

- Totalmente observáveis
- Deterministas
- Finitos
- Estáticos
- Discretos

Planeamento proposicional

Considere-se novamente o Wumpus e a descrição em lógica proposicional vista anteriormente:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$
$$S_{1,1} \Leftrightarrow (W_{1,2} \vee W_{2,1})$$
$$\vdots$$
$$W_{1,1} \vee W_{1,2} \vee \ldots W_{4,3} \vee W_{4,4}$$
$$\neg W_{1,1} \vee \neg W_{1,2}$$
$$\neg W_{1,1} \vee \neg W_{1,1}$$
$$\vdots$$
$$\neg W_{4,3} \vee \neg W_{4,4}$$

As fórmulas anteriores lidam com os aspectos atemporais do mundo. Como tratar os aspectos dinâmicos?

Fluentes e axiomas de efeito

Associa-se com cada fluente (algo no mundo que se altera com o tempo) um instante de tempo t: $FacingEast^0$, $HaveArrow^0$, $WumpusAlive^1$, etc. . .

$L^t_{x,y} \Rightarrow (Breeze^t \Leftrightarrow B_{x,y})$

$L^t_{x,y} \Rightarrow (Stench^t \Leftrightarrow S_{x,y})$

Para descrever como o mundo se altera necessitamos de axiomas de efeito :

$L^0_{1,1} \wedge FacingEast^0 \wedge Forward^0 \Rightarrow (L^1_{2,1} \wedge \neg L^1_{1,1})$

É necessário axioma destes para cada tempo possível, 16 casas, e cada uma das quatro direcções. Necessitamos ainda de axiomas semelhantes para as restantes acções: $Grab$, $Shoot$, $Climb$, $TurnLeft$ e $TurnRight$.

O "frame problem"

Suponha-se que o agente pretende avançar no instante 0. A partir do axioma anterior:

$L^0_{1,1} \wedge FacingEast^0 \wedge Forward^0 \Rightarrow (L^1_{2,1} \wedge \neg L^1_{1,1})$

obtém-se $ASK(KB, L^1_{2,1}) = true$ como seria de esperar.

Mas $ASK(KB, HaveArrow^1)$?

O "frame problem"

Suponha-se que o agente pretende avançar no instante 0. A partir do axioma anterior:

$L^0_{1,1} \wedge FacingEast^0 \wedge Forward^0 \Rightarrow (L^1_{2,1} \wedge \neg L^1_{1,1})$

obtém-se $ASK(KB, L^1_{2,1}) = true$ como seria de esperar.

Mas $KB \models HaveArrow^1$?

NÃO!

Nem $KB \models \neg HaveArrow^1$!

Os axiomas de efeito não dizem o que fica inalterado!

Os axiomas de "quiescência"

Podemos explicitar o que fica inalterado através de axiomas de quiescência (frame axioms)

$Forward^t \Rightarrow (HaveArrow^t \Leftrightarrow HaveArrow^{t+1})$

$Forward^t \Rightarrow (WumpusAlive^t \Leftrightarrow WumpusAlive^{t+1})$

...

Obriga à escrita de inúmeros axiomas. É preferível pensar em termos de fluentes especificando os axiomas de estado sucessor, genericamente

$F^{t+1} \Leftrightarrow AccaoCausaF^t \vee (F^t \wedge \neg AccaoCausaNaoF^t)$

Por exemplo,

$$HaveArrow^{t+1} \Leftrightarrow (HaveArrow^t \wedge \neg Shoot^t)$$

$$\begin{aligned} L_{1,1}^{t+1} \Leftrightarrow & (L_{1,2}^t \wedge (South^t \wedge Forward^t)) \\ & \vee (L_{2,1}^t \wedge (West^t \wedge Forward^t)) \\ & \vee (L_{1,1}^t \wedge (\neg Forward^t)) \end{aligned}$$

Cálculo de situações

Formalismo em lógica de primeira ordem que permite o raciocínio sobre o resultado das acções.

O cálculo de situações evita lidar explicitamente com o tempo, recorrendo em vez disso a *situações*.

Uma situação denota o estado resultante de se aplicar uma acção.

♢ Acções são denotadas por termos lógicos (e.g. $Forward$, $Move$, etc...)

♢ Situações são denotadas por termos lógicos, construídos a partir da situação inicial (normalmente designada por S_0) e por aplicação de uma acção a uma situação, representado pela expressão funcional $Result(a, s)$.

♢ Fluentes são predicados e funções que variam de uma situação para outra. Por convenção utiliza-se o último argumento do predicado ou função para identificar a situação a que se refere.

♢ Podem-se ainda utilizar predicados ou funções eternas/intemporais.

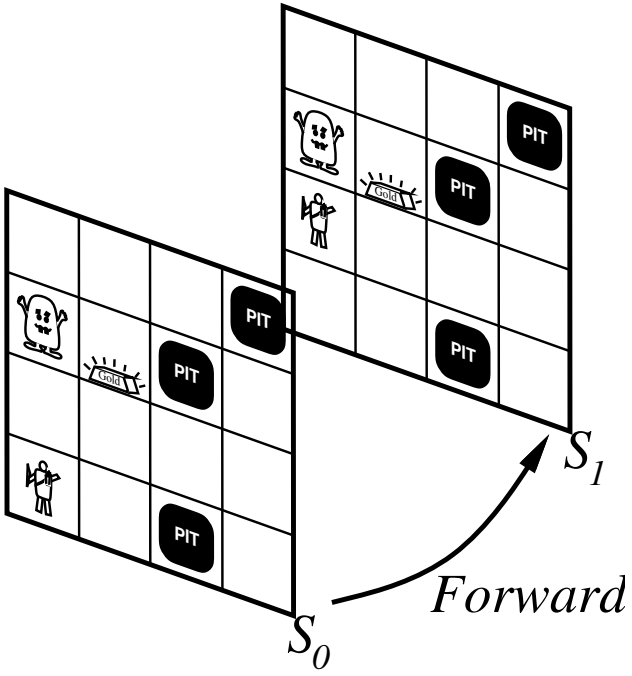
Exemplo

Factos verificam-se em situações, em vez de serem eternos
E.g., $Holding(Gold, Now)$ e não $Holding(Gold)$

Cálculo de Situações uma das formas de representar a mudança em LPO:

Juntar um argumento de situação a cada predicado não-eterno
E.g., Now em $Holding(Gold, Now)$ denota a situação

Situações são ligadas por intermédio da função $Result$
$Result(a, s)$ é a situação resultante de se efectuar a em s (e.g. $Result(Forward, S_0)$).


PIT
PIT
PIT
PIT
PIT
Gold
Gold
S_1
Forward
S_0

Descrevendo acções

Modelo simplificado do mundo do Wumpus em que é ignorada a orientação e em que o agente se pode deslocar para qualquer casa adjacente.

Assuma-se que o agente está na casa $[1,1]$ e o ouro na casa $[1,2]$.

Vamos utilizar os fluentes $At(o,x,s)$ e $Holding(o,s)$ para representar, que o se encontra na casa x na situação s e que o agente possui o na situação s, respectivamente.

Descrição da situação inicial

$$At(o,x,S_0) \equiv [(o = Agent \wedge x = [1,1]) \vee (o = G_1 \wedge x = [1,2])]$$
$$\neg Holding(o,S_0)$$
$$Gold(G_1)$$
$$Adjacent([1,1],[1,2]) \wedge Adjacent([1,2],[1,1])$$

Nota: O conhecimento $At(Agent,[1,1],S_0) \wedge At(Gold,[1,2],S_0)$ não captura a totalidade da situação, pois não especifica conhecimento negativo relativamente ao fluente $At/3$ – Princípio do Mundo Aberto!!!

Tarefas que pretendemos efectuar

♢ Projecção: Deduzir o resultado da aplicação de uma sequência de acções. Por exemplo, demonstrar que:

$$At(G_1, [1,1], Result(Go([1,2],[1,1]), Result(Grab(G_1), Result(Go([1,1],[1,2]), S_0)))$$

♢ Planeamento: Encontrar a sequência de acções que permitem atingir o objectivo.

$$\exists_{s,g}\ Gold(g) \wedge Holding(g,s)$$

Descrevendo acções no cálculo de situações

Pode-se descrever cada acção com dois axiomas: axioma de possibilidade e axioma de efeito.

♢ Axiomas de Possibilidade: $Precondições \Rightarrow Poss(a, s)$

$$
\begin{array}{lcl}
At(Agent, x, s) \wedge Adjacent(x, y) & \Rightarrow & Poss(Go(x, y), s) \\
Gold(g) \wedge At(Agent, x, s) \wedge At(g, x, s) & \Rightarrow & Poss(Grab(g), s) \\
Holding(g, s) & \Rightarrow & Poss(Release(g), s)
\end{array}
$$

♢ Axiomas de Efeito: $Poss(a, s) \Rightarrow$ mudanças devidas à acção

$$
\begin{array}{lcl}
Poss(Go(x, y), s) & \Rightarrow & At(Agent, y, Result(Go(x, y), s)) \\
Poss(Grab(g), s) & \Rightarrow & Holding(g, Result(Grab(g), s)) \\
Poss(Release(g), s) & \Rightarrow & \neg Holding(g, Result(Release(g), s))
\end{array}
$$

Interrogando a teoria

Será que se pode concluir

$$At(Agent, [1,2], Result(Go([1,1],[1,2]), S_0)) \ ?$$

e

$$At(G_1, [1,2], Result(Go([1,1],[1,2]), S_0)) \ ?$$

Interrogando a teoria

Será que se pode concluir

$$At(Agent, [1,2], Result(Go([1,1],[1,2]), S_0)) \ ?$$ Sim

e

$$At(G_1, [1,2], Result(Go([1,1],[1,2]), S_0)) \ ?$$ NÃO!

Problema: Não basta dizer aquilo que se altera com a acção mas também é necessário dizer aquilo que fica inalterado.

Os axiomas de quiesciência (frame axioms) especificam o que se mantém inalterado com a acção

$$At(o,x,s) \wedge (o \neq Agent) \wedge \neg Holding(o,s) \Rightarrow At(o,x,Result(Go(x,y),s))$$

Se existirem F fluentes e A acções necessitaremos de no pior caso de $A \times F$ axiomas.

Problemas a enfrentar

Problema da Quiescência: encontrar uma forma elegante de lidar com o que se mantém inalterável

(a) representação—evitar axiomas de quiescência

(b) inferência—evitar cópias repetidas para manter o estado

Problema da Qualificação: descrições das acções reais obrigam à consideração de uma série de casos excepcionais —o que acontece se o ouro estive escorregadio ou pregado ao chão ou . . .

Problema da Ramificação: acções reais têm muitas consequências secundárias— o que acontece à poeira em cima do ouro . . .

Resolução do problema da quiescência

Axiomas de estado sucessor resolvem o problema da quiescência representacional

Cada axioma é 'acerca' de um predicado (não da acção individualmente):

Acção é possível $\Rightarrow$

(P verdade a seguir $\Leftrightarrow$ (uma acção tornou P verdadeiro

$\vee$ P já verdadeiro e nenhuma acção tornou P falso)]

Axiomas de estado sucessor para o Wumpus

Predicado $Holding/2$:

$$Poss(a, s) \Rightarrow$$
$$(Holding(g, Result(a, s)) \Leftrightarrow (a = Grab(g) \vee Holding(g, s) \wedge a \neq Release(g)))$$

Predicado $At/3$:

$$Poss(a, s) \Rightarrow$$
$$(At(Agent, y, Result(a, s)) \Leftrightarrow (a = Go(x, y) \vee At(Agent, y, s) \wedge a \neq Go(y, z)))$$

E o que acontece ao ouro?

Problema da ramificação para o Wumpus

É necessário dizer que tudo aquilo que o agente transporta vai com ele, e se o agente não se mexer então tudo o que ele possuir também não se mexe!

$$
\begin{aligned}
& Poss(a, s) \Rightarrow \\
& (At(o, y, Result(a, s)) \Leftrightarrow (a = Go(x, y) \wedge (o = Agent \vee Holding(o, s)) \vee \\
& \qquad At(o, y, s) \wedge \neg(\exists_z\ y \neq z \wedge a = Go(y, z) \wedge \\
& \qquad (o = Agent \vee Holding(o, s)))))
\end{aligned}
$$

Axiomas de nomes únicos

Como se faz uso da igualdade na modelação em cálculo de situações é preciso indicar que constantes e símbolos de função distintos se referem a objectos distintos. Para cada par de constantes temos de dizer que são distintos (Axiomas de nomes únicos). Para o nosso exemplo,

$$Agent \neq Gold \wedge Agent \neq 1 \wedge Agent \neq 2 \wedge Gold \neq 1 \wedge Gold \neq 2 \wedge 1 \neq 2$$

Muitos sistemas assumem implicitamente estes axiomas (e.g. Prolog). Nesse caso dizemos que estamos na presença da assunção de nomes únicos.

Axiomas de acções únicas

Temos ainda de dizer que todas as acções são distintas. Para cada par de nomes de acções A, B é necessário indicar que $A(x_1, \ldots, x_m) \neq B(y_1, \ldots, y_n)$:

$$\forall_{x_1,y_1,y_2}\ Grab(x_1) \neq Go(y_1, y_2)$$
$$\forall_{x_1,y_1}\ Grab(x_1) \neq Release(y_1)$$
$$\forall_{x_1,y_1,y_2}\ Release(x_1) \neq Go(y_1, y_2)$$

É ainda necessário juntar para cada termo de acção:

$$\forall_{x_1,y_1}\ Grab(x_1) = Grab(y_1) \equiv x_1 = y_1$$
$$\forall_{x_1,y_1}\ Release(x_1) = Release(y_1) \equiv x_1 = y_1$$
$$\forall_{x_1,x_2,y_1,y_2}\ Go(x_1, x_2) = Go(y_1, y_2) \equiv x_1 = y_1 \wedge x_2 = y_2$$

Estes axiomas são designados por axiomas de acções únicas.

Com estes axiomas já poderemos resolver o problema de planeamento pretendido!

Linguagem PDDL (estados)

A utilização do cálculo de situações para representar problemas de planeamento obriga a um conhecimento aprofundado das particularidades da lógica, não sendo fácil para o utilizador. A linguagem PDDL (Planning Domain Definition Language) combina a procura em espaços de estados com as representações lógicas.

♢ Representação dos 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).

Linguagem PDDL (acções)

♢ 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).

$$
\begin{array}{l}
Action(Fly(p, from, to), \\
\quad \textsc{Precond} : At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to) \\
\quad \textsc{Effect} : \neg At(p, from) \wedge At(p, to) \\
)
\end{array}
$$

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

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

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 a num estado s é o estado

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

em que a lista (na realidade é um conjunto) de remoções $\textsc{Del}(a)$ é formada pelos literais negativos no efeito de a e a lista de adições $\textsc{Add}(a)$ é o conjunto de literais positivos no efeito de a.

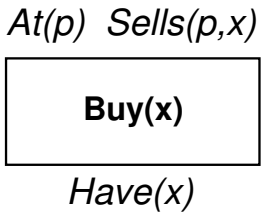
$\diamondsuit$ Um plano é uma sequência de acções PDDL proposicionalizadas que leva do estado inicial ao estado final.

Operadores STRIPS

A aproximação STRIPS (STanford Research Institute Problem Solver) é um subconjunto da PDDL que não permite literais negativos nas precondições e nos objectivos.

Descrição organizada de acções, linguagem restrita

ACÇÃO: $Buy(x)$
PRECONDIÇÃO: $At(p), Sells(p, x)$
EFEITO: $Have(x)$



ACÇÃO: $Go(x)$
PRECONDIÇÃO: $At(y)$
EFEITO: $At(x), -At(y)$

[Nota: abstrai de muitos detalhes importantes!]

Linguagem restrita $\Rightarrow$ algoritmo "eficiente"
- Precondição: conjunção de literais positivos
- Efeito: conjunção de literais

Tradução para Cálculo de Situações

Um conjunto de operadores PDDL pode ser traduzido automaticamente para um conjunto de axiomas de estado-sucessor. Para o exemplo anterior temos:

$$\begin{aligned} \forall_{s,x,p}\ [At(p,s) \wedge Sells(p,x,s) &\Rightarrow Poss(Buy(x),s)] \\ \forall_{s,x,y}\ [At(y,s) &\Rightarrow Poss(Go(x),s)] \end{aligned}$$

$$\forall_{a,s,x}\ Poss(a,s) \Rightarrow [Have(x,Result(a,s)) \equiv (a = Buy(x) \vee Have(x,s))]$$

$$\begin{aligned} \forall_{a,s,x,y}\ Poss(a,s) \Rightarrow [At(x,Result(a,s)) \equiv\ & (a = Go(x) \vee \\ & At(x,s) \wedge \neg(a = Go(y) \wedge x \neq y))] \end{aligned}$$

Assunções da linguagem PDDL

♢ Estados são conjuntos de literais positivos

♢ Os literais que não ocorrem num estado assumem-se falsos (princípio do mundo fechado)

♢ 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

Considere-se o estado inicial e os operadores abaixo:

$\{em(fct), com(dinheiro), praia(caparica), local(caparica), local(fct)\}$

Acção: $ir(?X)$
Precondição: $em(?Y), local(?X)$
Efeito: $em(?X), -em(?Y)$

Acção: $banhoSol$
Precondição: $em(?Y), praia(?Y)$
Efeito: $com(bronze), com(sede), -sem(sede)$

Acção: $beberBijeca$
Precondição: $com(sede), com(dinheiro)$
Efeito: $-com(sede), -com(dinheiro), sem(sede)$

Objectivo: ficar com "bronze" e 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.

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 contrário. Trabalha com estados incompletos.

Exemplo: planeamento progressivo

Plano PDDL: $[ir(caparica), banhoSol, beberBijeca]$

Execução do plano e mudanças de estado:

$$\{em(fct), com(dinheiro), praia(caparica), local(caparica), local(fct)\}$$
$$\Downarrow \quad ir(caparica)$$
$$\{em(caparica), com(dinheiro), praia(caparica), local(caparica), local(fct)\}$$
$$\Downarrow \quad banhoSol$$
$$\{em(caparica), com(dinheiro), com(bronze), com(sede),$$
$$praia(caparica), local(caparica), local(fct)\}$$
$$\Downarrow \quad beberBijeca$$
$$\{em(caparica), com(bronze), sem(sede),$$
$$praia(caparica), local(caparica), local(fct)\}$$

O Plano PDDL $[ir(caparica), banhoSol, banhoSol, beberBijeca]$ também é uma solução ?

E $[ir(caparica), banhoSol, beberBijeca, banhoSol]$ também é solução ?

Exemplo: planeamento regressivo

Seja g o objectivo corrente. O novo objectivo corrente g' é obtido considerando apenas as acções a tal que a tem um efeito positivo em a e nenhum efeito negativo em g. O novo objectivo é obtido $g' = (g - \textsc{Add}(a)) \cup \textsc{Precond}(a)$

- Removendo todo o efeito positivo de a que apareça em g
- Toda a precondição de a é adicionada a g, desde que não ocorra lá.

Construção do plano a partir dos objectivos:

$$\{sem(sede), com(bronze)\}$$
$$\Uparrow \quad beberBijeca$$
$$\{com(dinheiro), com(sede), com(bronze), \}$$
$$\Uparrow \quad banhoSol$$
$$\{com(dinheiro), em(?Y1), praia(?Y1)\}$$
$$\Uparrow \quad ir(caparica)$$
$$\{com(dinheiro), praia(caparica), em(?X1), local(?X1)\}$$

(conclui-se do estado inicial fazendo $?X1 = fct$)

Heurísticas para plan. em espaços 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 plan. em espaços 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 $G_1, \ldots, G_n$ e $P_1, \ldots, P_n$ os planos para os resolver. A heurística $max_i\ COST(P_i)$ é admissível mas $\Sigma_i\ COST(P_i)$ já não. Mas se G_i e G_j forem independentes podemos utilizar $COST(P_i)$+COST(P_j).

Planeamento no espaço de planos parciais

As técnicas anteriores produzem planos lineares totalmente ordenados (sequência de acções), ignorando o facto de que muitos subproblemas são independentes.

♢ Nos anos 1980 e 1990 o planeamento com ordem parcial era a tecnologia dominante pois era capaz de lidar com a independência de subproblemas.

♢ Hoje em dia é ainda muito utilizado em problemas de escalonamento e em situações em que é necessário indicar os planos a seres humanos para os verificarem. Permitem ainda a utilização de bibliotecas de planos.

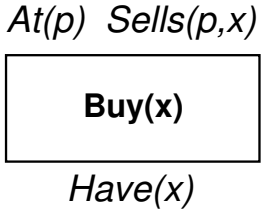
Foram destronados a partir do ano 2000 pelo planeamento progressivo pois foram descobertas excelentes heurísticas capazes de encontrar os subproblemas independentes.

Voltando ao Supermercado

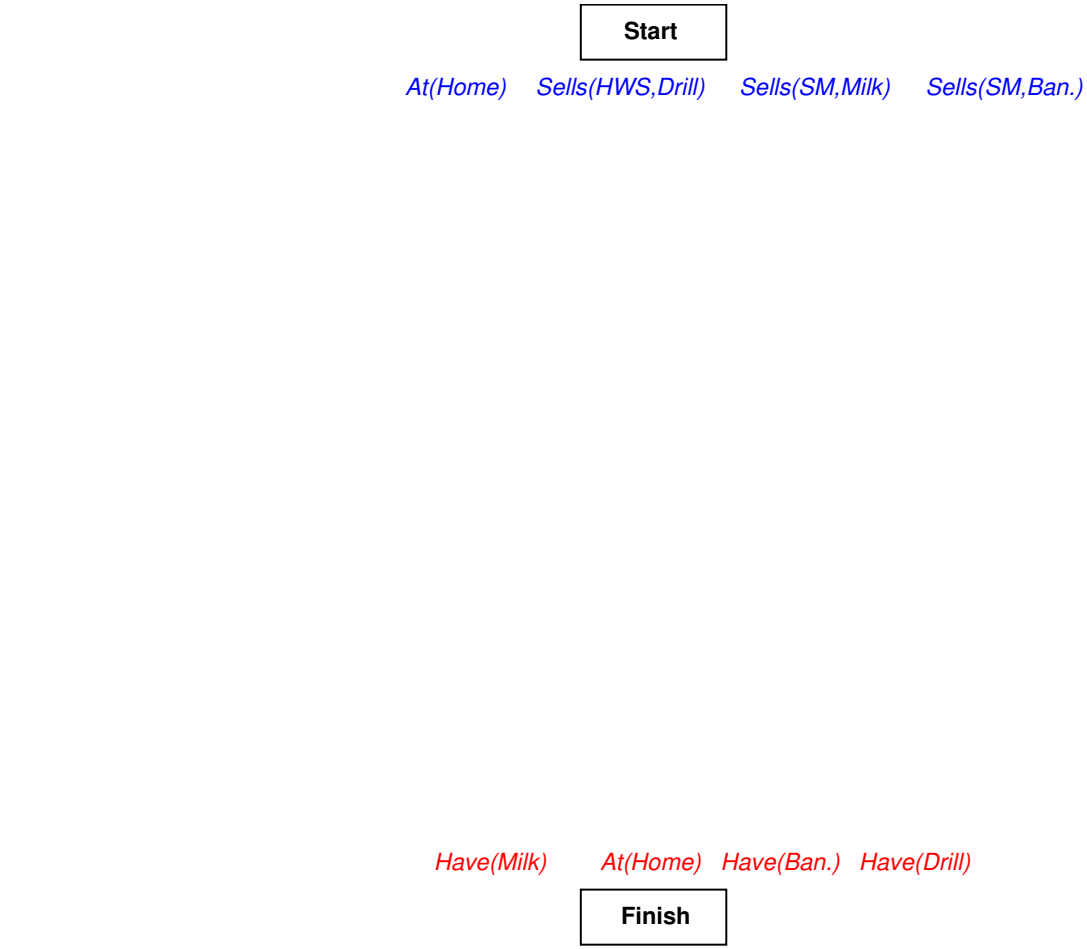
Descrição organizada de acções, linguagem restrita

Acção: $Buy(x)$
Precondição: $At(p), Sells(p, x)$
Efeito: $Have(x)$

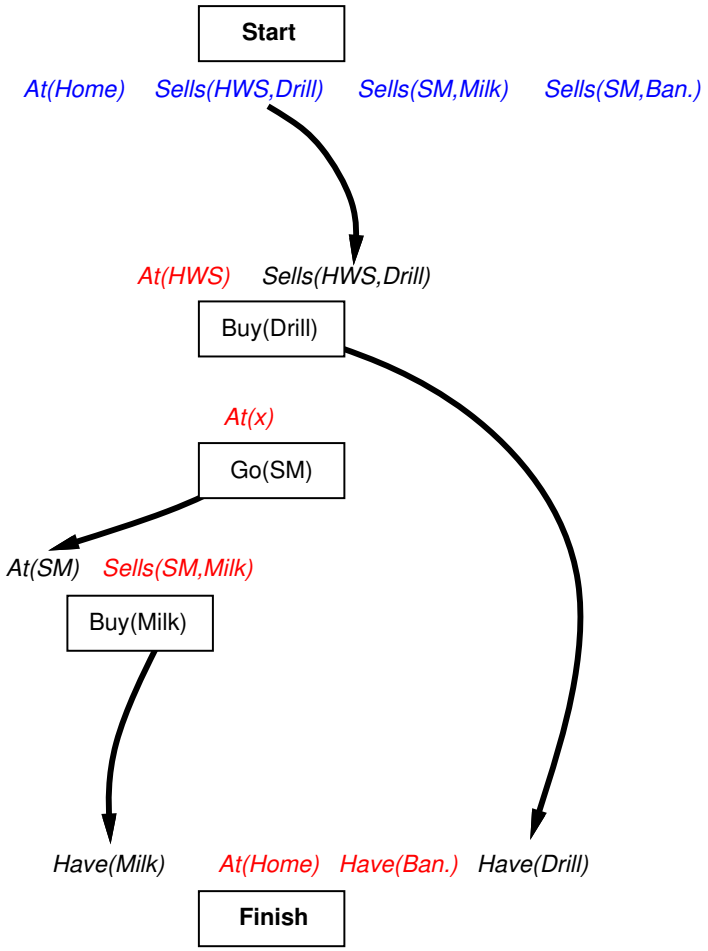
Acção: $Go(x)$
Precondição: $At(y)$
Efeito: $At(x), -At(y)$



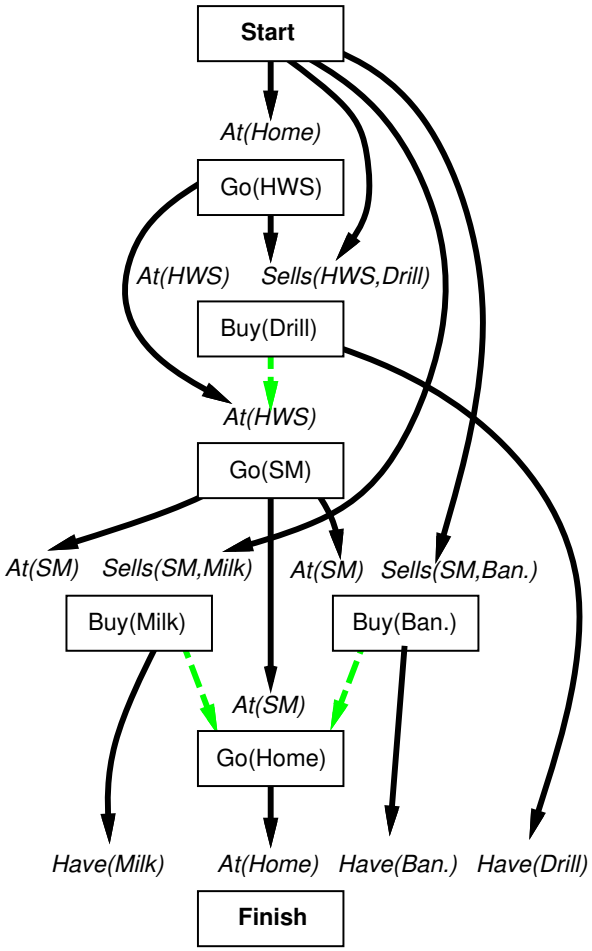
Exemplo


Start
At(Home)
Sells(HWS,Drill)
Sells(SM,Milk)
Sells(SM,Ban.)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish

Exemplo


Start
At(Home)
Sells(HWS,Drill)
Sells(SM,Milk)
Sells(SM,Ban.)
At(HWS)
Sells(HWS,Drill)
Buy(Drill)
At(x)
Go(SM)
At(SM)
Sells(SM,Milk)
Buy(Milk)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish

Exemplo


Start
At(Home)
Go(HWS)
At(HWS)
Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM)
Sells(SM,Milk)
At(SM)
Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish

Planos parcialmente ordenados

Colecção de passos *parcialmente ordenados* tal que

- passo $Start$ descreve o estado inicial como os seus efeitos
- passo $Finish$ descreve o objectivo como as suas precondições
- ligações causais do efeito de um passo para a precondição de outro
- ordenação temporal entre pares de passos

Condição aberta = precondição de um passo que ainda não está ligada causalmente

Um plano está completo sse toda a precondição foi alcançada

Uma precondição encontra-se alcançada sse é o efeito de um passo anterior
e nenhum passo possivelmente interveniente a contraria

NOTA: Consideramos apenas operadores STRIPS (sem precondições negativas!).

Processo de Planeamento

Operadores em planos parciais:

adiciona uma ligação de uma acção existente para uma condição aberta
adiciona um passo para garantir uma condição aberta
ordenar um passo relativamente a outro para eliminar possíveis conflitos

Evoluem gradualmente de planos incompletos/com falhas para planos correctos e completos

Retrocede se uma condição não é alcançável ou se um conflito não se puder resolver.

Algoritmo POP

function POP(*initial, goal, operators*) **returns** *plan*

plan ← MAKE-INITIAL-PLAN(*initial, goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*
  $S_{need},\ c \leftarrow$ SELECT-SUBGOAL(*plan*)
  CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)
  RESOLVE-THREATS(*plan*)
end

---

function SELECT-SUBGOAL(*plan*) **returns** $S_{need},\ c$

pick a plan step S_{need} from STEPS(*plan*)
  with a precondition c that has not been achieved
return $S_{need},\ c$

Algoritmo POP

procedure CHOOSE-OPERATOR($plan$, $operators$, S_{need}, c)

choose a step S_{add} from $operators$ or STEPS($plan$) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS($plan$)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS($plan$)
if S_{add} is a newly added step from $operators$ **then**
    add S_{add} to STEPS($plan$)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS($plan$)

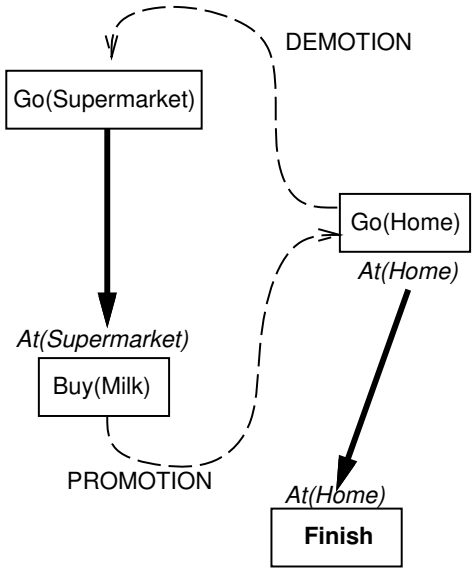
---

procedure RESOLVE-THREATS($plan$)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS($plan$) **do**
    choose either
        Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS($plan$)
        Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS($plan$)
    if not CONSISTENT($plan$) **then fail**
end

Ameaças e promoção/despromoção

Um passo é uma ameaça (conflituante) quando potencialmente destrói uma condição já alcançada por uma ligação causual. E.g., $Go(Home)$ é uma ameaça a $At(Supermarket)$:



Despromoção: colocar antes de $Go(Supermarket)$

Promoção: colocar depois de $Buy(Milk)$

O teste de consistência assegura que não foram introduzidos ciclos nas relações de ordem temporais.

Propriedades do POP

Algoritmo não-determinista: retrocede no caso de falha para pontos de escolha:
- escolha de S_{add} para alcançar S_{need}
- escolha de promoção ou despromoção
- selecção de S_{need} é irrevogável

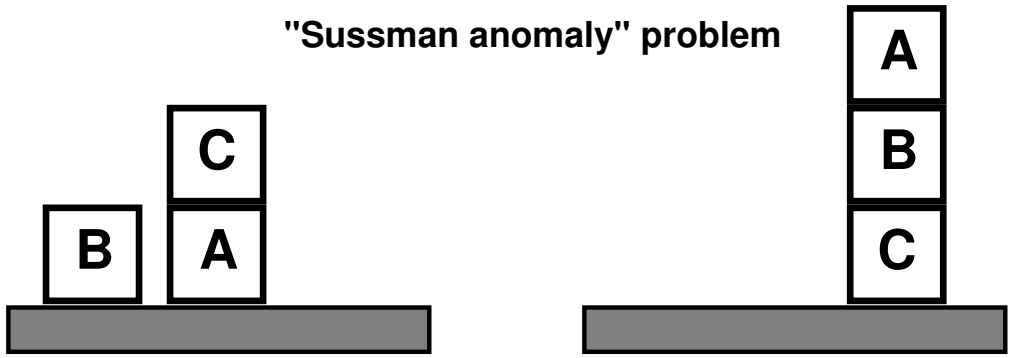
POP é sólido, completo, e sistemático (sem repetições)

Extensões para disjunção, universais, negação e condicionais na definição dos operadores (linguagem ADL).

Consegue-se eficiência com boas heurísticas derivadas da descrição do problema

Particularmente bom para problemas com muitos sub-objectivos fracamente interligados

Exemplo: Mundo dos blocos



Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)

PutOnTable(x)

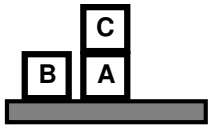
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Exemplo: Mundo dos blocos

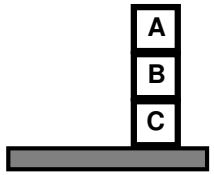
START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

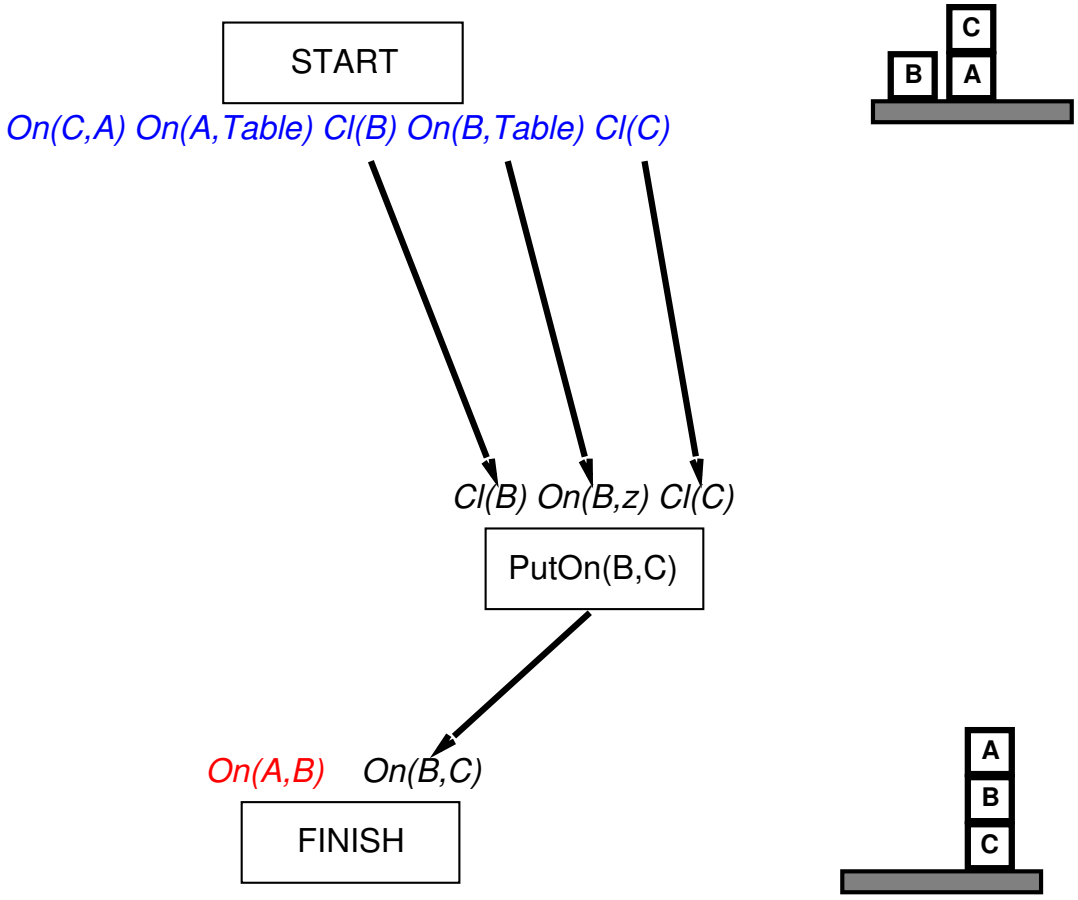


On(A,B) On(B,C)

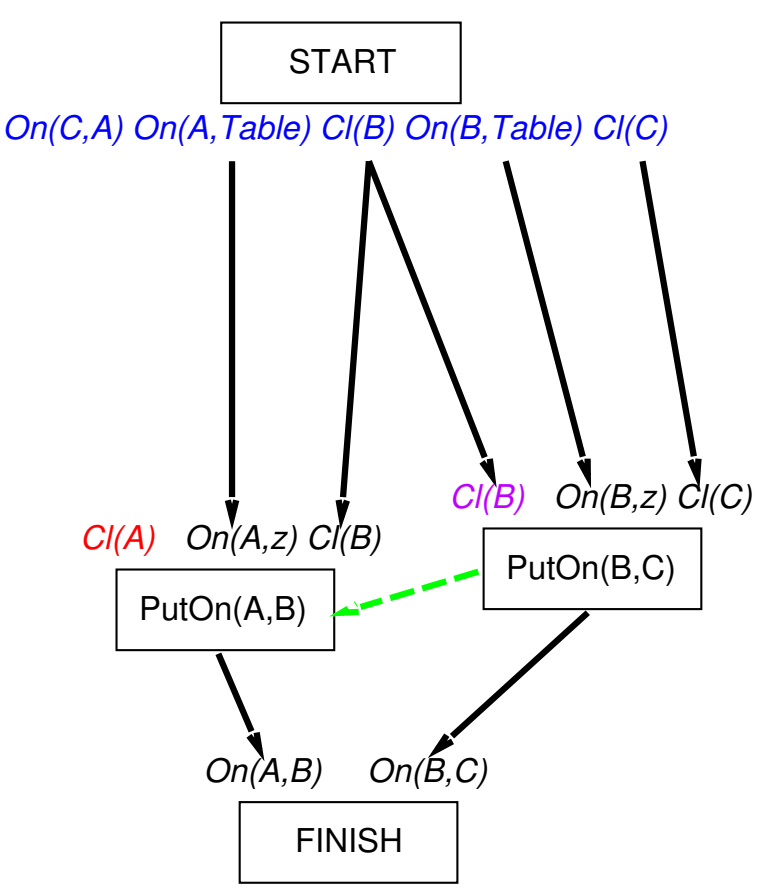
FINISH

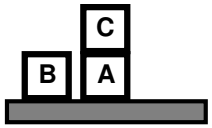


Exemplo: Mundo dos blocos

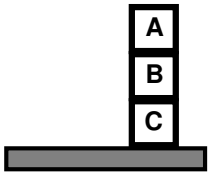


Exemplo: Mundo dos blocos

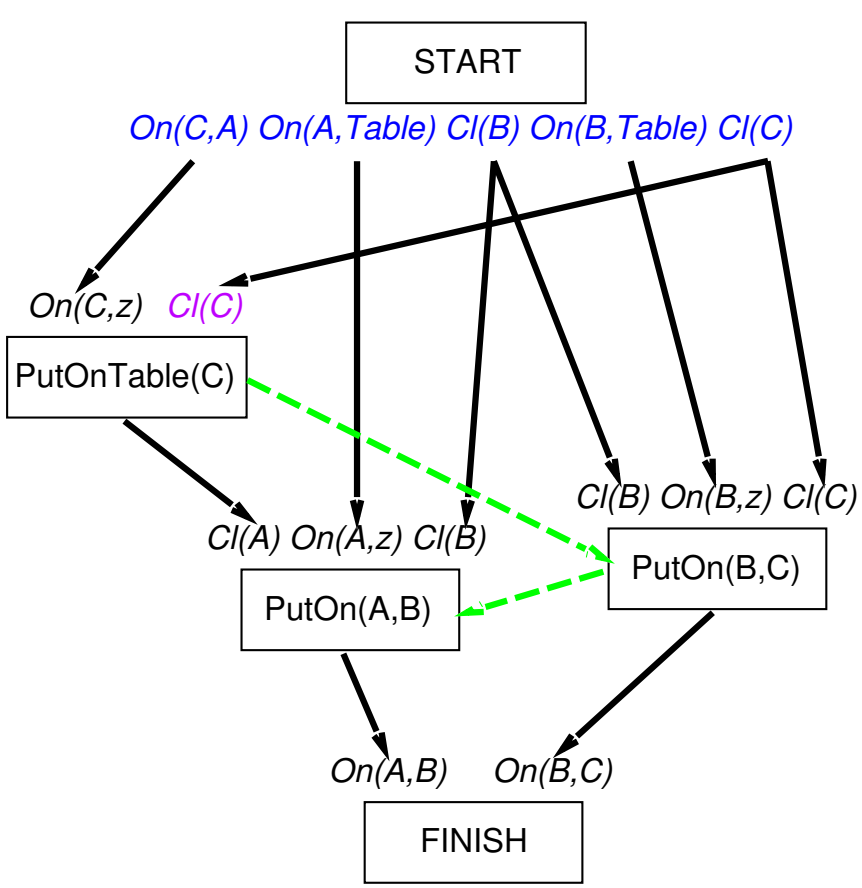


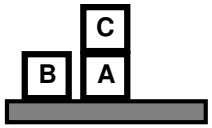


PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)



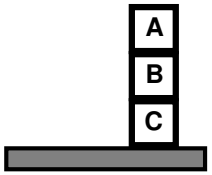
Exemplo: Mundo dos blocos





PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)

PutOn(B,C)
clobbers Cl(C)
=> order after
PutOnTable(C)



Casa & Descasa

A agência de matrimónios Casa&Descasa decidiu investir num sistema de Inteligência Artificial para melhorar o atendimento aos seus clientes. O sistema mantém informação sobre os homens e mulheres registados no sistema, os casais (heterossexuais) e indivíduos solteiros. O objectivo do sistema é aconselhar casamentos e divórcios sugerindo ainda "falecimentos"de pessoas. A poligamia não é permitida.

As acções são: casar, divorciar, matar_homem_casado e matar_mulher_casada.

Por exemplo, o sistema deverá ser capaz de indicar os passos que levam o indivíduo Aníbal, homem e solteiro, a ficar casado com Belarmina que inicialmente está casada com César

♢ Existe um número infinito de planos ?

♢ Porque é que não é possível poligamia ?

Casa & Descasa

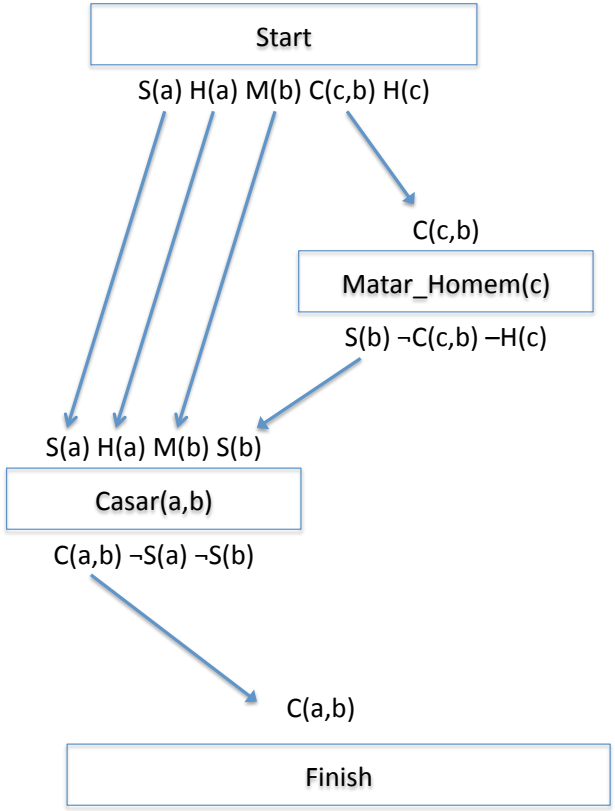
Acção: $Casar(H, M)$
Precondição: $solteiro(H), homem(H), solteiro(M), mulher(M)$
Efeito: $casados(H, M), -solteiro(H), -solteiro(M)$

Acção: $Divorciar(H, M)$
Precondição: $casados(H, M)$
Efeito: $solteiro(H), solteiro(M), -casados(H, M)$

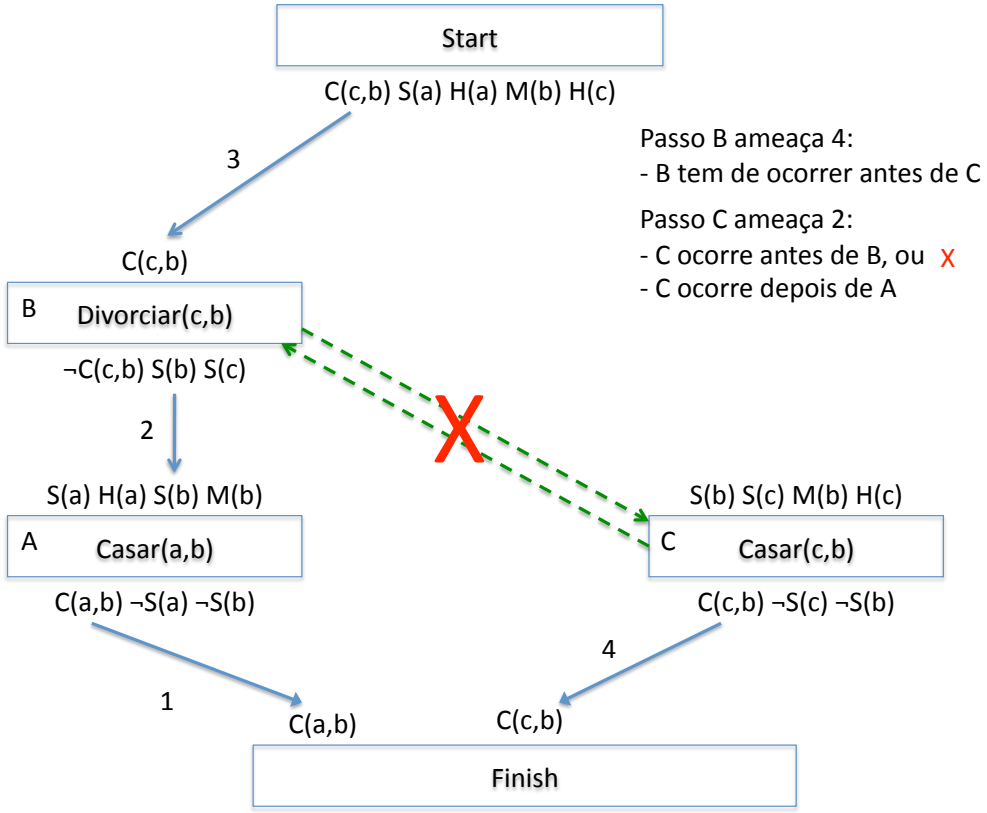
Acção: $Matar\_Homem\_Casado(H)$
Precondição: $casados(H, M)$
Efeito: $solteiro(M), -casados(H, M), -homem(H)$

Acção: $Matar\_Mulher\_Casada(M)$
Precondição: $casados(H, M)$
Efeito: $solteiro(H), -casados(H, M), -mulher(M)$

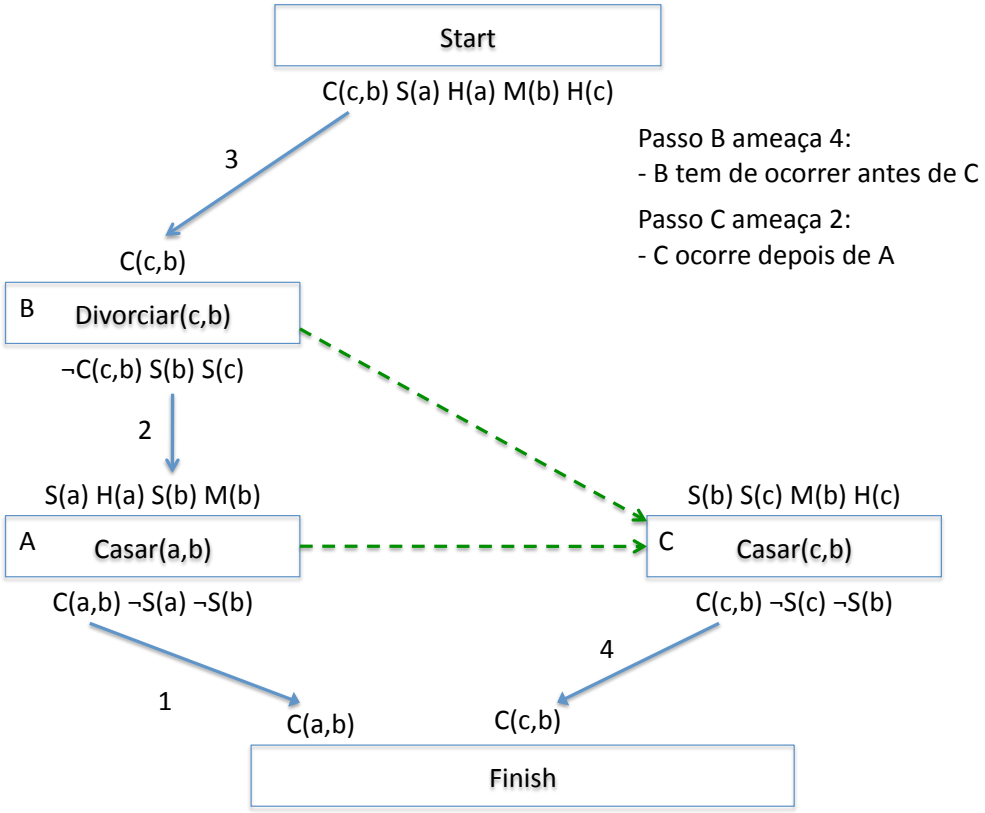
Plano solução



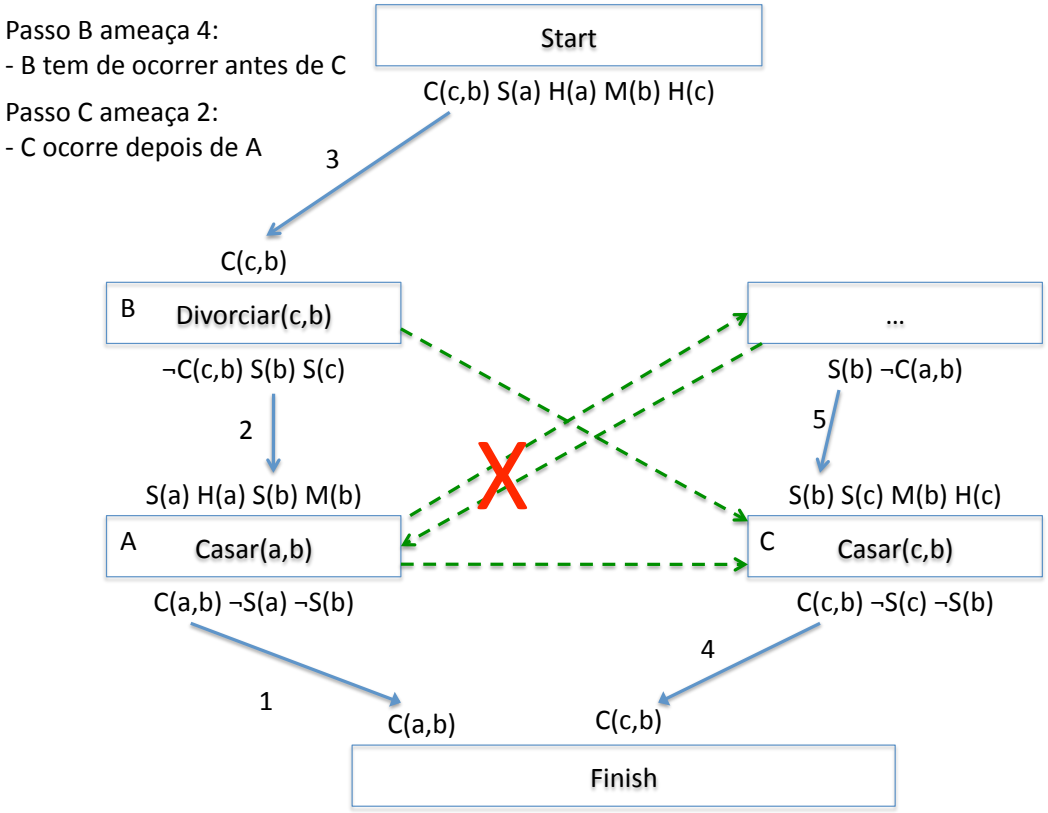
Poligamia impossível (um exemplo)



Poligamia impossível (exemplos)



Poligamia impossível (exemplos)



Zé Tuga

O Zé Tuga, com tanto campeonato Europeu de Futebol, não tem dedicado muito tempo à sua Maria. Para acalmar os ânimos decidiu organizar um jantar a dois, em que ele põe a mesa! Dada a sua manifesta ignorância na organização destes eventos pediu conselhos à Maria, que o instruiu:

- Podes pôr uma toalha (ou não), mas só o podes fazer se não estiver nada em cima da mesa.

- Para a mesa ficar arranjadinha só deves colocar os talheres e os copos depois dos pratos.

- A ordem de colocação dos talheres e dos copos é indiferente, mas só depois dos pratos!

♢ Podem existir planos com acções repetidas ?

Zé Tuga

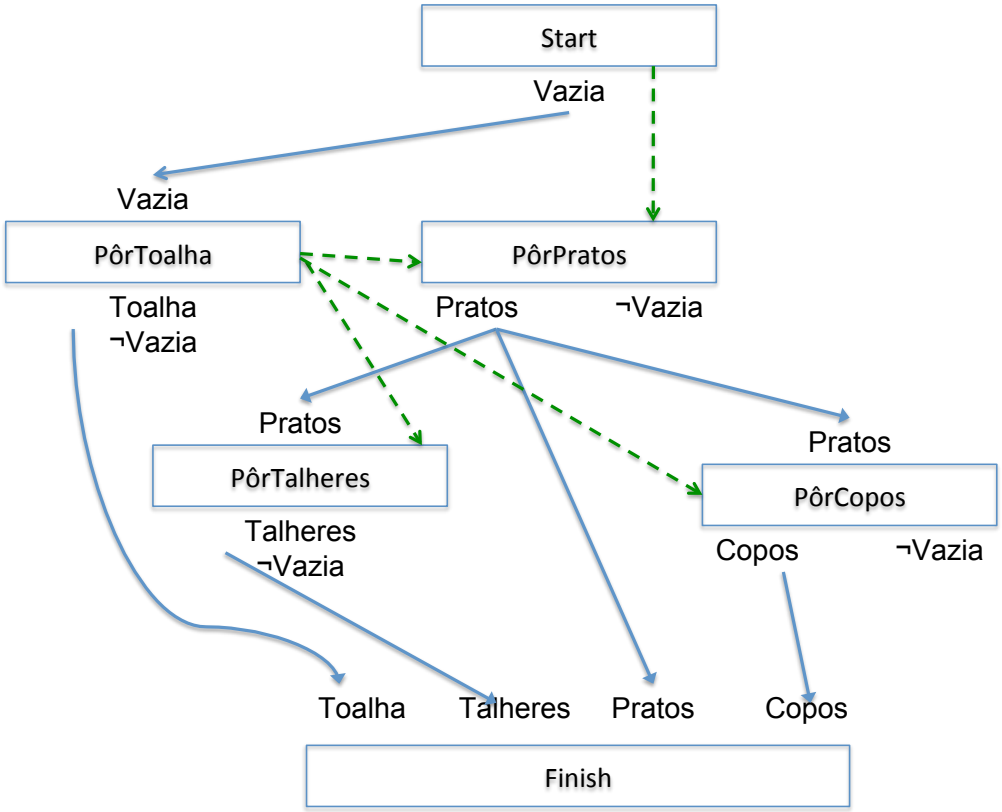
Acção: PôrToalha
Precondição: $Vazia(Mesa)$
Efeito: $Toalha, -Vazia(Mesa)$

Acção: PôrPratos
Precondição:
Efeito: $Pratos, -Vazia(Mesa)$

Acção: PôrCopos
Precondição: $Pratos$
Efeito: $Copos, -Vazia(Mesa)$

Acção: PôrTalheres
Precondição: $Pratos$
Efeito: $Talheres, -Vazia(Mesa)$

Plano para o Zé Tuga



Outro plano para o Zé Tuga