Modelação de problemas de Planeamento

O tópico destas aulas é modelação de problemas de Planeamento em PDDL. Poderá usar o editor/planeador disponível em http://editor.planning.domains.

Exemplo

I - Dentro de uma sala, onde se encontra um macaco, está um cacho de bananas pendurado no tecto, demasiado alto para o macaco lhe chegar, existindo também uma fonte de água, uma faca e um copo. Num canto da sala está um caixote, que não está por baixo das bananas. O caixote é suficientemente forte para suportar o macaco se ele lhe subir para cima, e suficientemente leve para por ele ser empurrado entre quaisquer dois locais. Se o macaco, na posse da faca, subir para cima do caixote e este estiver junto às bananas, o macaco consegue apanhar as bananas. Noutro local da sala, afastado da fonte de água, está um copo. Se o macaco tiver o copo consigo, e estiver junto à fonte de água, pode encher o copo de água. O macaco consegue executar as seguintes acções:

- Mover-se entre quaisquer dois pontos da sala, desde que não esteja em cima do caixote;
- Subir para cima do caixote, assumindo que o macaco está no local do caixote;
- Empurrar o caixote entre quaisquer dois pontos, assumindo que o macaco está no local do caixote, e não está em cima dele;
- Apanhar a faca, assumindo que o macaco está no local da faca;
- Apanhar o copo, assumindo que o macaco está no local do copo;
- Apanhar as bananas, assumindo que o macaco está em cima do caixote, no local das bananas, e tem a faca.
- Encher o copo de água, assumindo que o macaco está no chão, tem o copo e está no local da fonte de água.

Para modelar este exemplo em PDDL, iremos usar as constantes (com significado óbvio) monkey, box, knife, bananas, glass, water e waterfountain, bem como os seguintes predicados:

- `location/1` - predicado de domínio indicando os locais da sala;
- `on-floor/0` - proposição indicando que o macaco está no chão;
- `at/2` - predicado binário indicando que o objecto referido pelo primeiro termo está no local referido pelo segundo termo.
- `onbox/0` - proposição indicando que o macaco está em cima do caixote;
- `hasknife/0` - proposição indicando que o macaco tem a faca;
- `hasbananas/0` - proposição indicando que o macaco tem as bananas;
- `hasglass/0` - proposição indicando que o macaco tem o copo;
- `haswater/0` - proposição indicando que o macaco tem o copo cheio de água;

Em PDDL, isto é declarado através de:

```
(:constants monkey box knife bananas glass water waterfountain)

(:predicates (on-floor) (at ?x ?y) (onbox) (hasknife)
     (hasbananas) (hasglass) (haswater) (location ?x))
```

As acções são definidas da seguinte forma:

```
(:action go-to
     :parameters (?x ?y)
     :precondition (and (not (= ?y ?x)) (on-floor)
          (at monkey ?y))
     :effect (and (at monkey ?x) (not (at monkey ?y))))
(:action climb
     :parameters (?x)
     :precondition (and (at box ?x) (at monkey ?x))
     :effect (and (onbox) (not (on-floor))))
(:action push-box
     :parameters (?x ?y)
     :precondition (and (not (= ?y ?x)) (at box ?y)
          (at monkey ?y) (on-floor))
     :effect (and (at monkey ?x) (not (at monkey ?y))
          (at box ?x) (not (at box ?y))))
(:action get-knife
     :parameters (?y)
     :precondition (and (at knife ?y) (at monkey ?y))
     :effect (and (hasknife) (not (at knife ?y))))
(:action grab-bananas
     :parameters (?y)
     :precondition (and (at monkey ?y) (hasknife)
          (at bananas ?y) (onbox))
     :effect (hasbananas))
(:action pickglass
     :parameters (?y)
     :precondition (and (at glass ?y) (at monkey ?y))
     :effect (and (hasglass) (not (at glass ?y))))
(:action getwater
     :parameters (?y)
     :precondition (and (hasglass) (at waterfountain ?y)
          (at monkey ?y) (on-floor))
     :effect (haswater))
```

Devido à utilização da igualdade, é necessário declarar

```
(:requirements :equality)
```

A definição completa do domínio pode ser encontrada aqui.

Pode testar a implementação usando o seguinte problema (disponível aqui):

```
(define (problem monkey-test3)
     (:domain monkey-domain)
     (:objects p1 p2 p3 p4 p5 p6)
     (:init (location p1) (location p2)
          (location p3) (location p4)
          (location p5) (location p6)
          (at monkey p1) (on-floor)
          (at box p2)
          (at bananas p3)
          (at knife p4)
          (at waterfountain p5) (at glass p6))
     (:goal (and (hasbananas) (haswater))))
```

Poderá também criar outros problemas diferentes para testar a definição do domínio.

Exercícios

I - Considere uma generalização do problema das Torres de Hanói, permitindo a existência de um número arbitrário de pinos e de discos. Assumindo que o domínio está definido em PDDL com o nome "hanoi",uma instância da versão clássica com 3 torres e 3 discos poderia ser representada da seguinte forma em PDDL:

```
(define (problem hanoi-3)
     (:domain hanoi)
     (:objects p1 p2 p3 d1 d2 d3)
     (:init (disk d1) (disk d2) (disk d3) (on d1 d2)
          (on d2 d3) (on d3 p3)
          (smaller d1 p1) (smaller d2 p1) (smaller d3 p1)
          (smaller d1 p2) (smaller d2 p2) (smaller d3 p2)
          (smaller d1 p3) (smaller d2 p3) (smaller d3 p3)
          (smaller d1 d2) (smaller d1 d3) (smaller d2 d3)
          (clear p1) (clear p2) (clear d1))
     (:goal (and (on d1 d2) (on d2 d3) (on d3 p1))))
```

Modele o domínio das Torres de Hanói em PDDL de modo a que possa resolver problemas arbitrários do tipo do problema descrito i.e. considerando várias torres, discos, configurações iniciais e finais (dica: basta definir uma acção `move-disk/3`). Pode testar a sua implementação neste problema.

II - Um robô tem vários braços, podendo apanhar e largar caixas, uma com cada um dos braços de forma independente, e transportá-las entre as várias divisões de uma casa. Claro que o robô só se pode deslocar entre divisões que estejam ligadas. O objectivo é colocar as caixas nas salas indicadas.

Assumindo que o domínio está definido em PDDL com o nome "robot", segue a descrição de um problema concreto envolvendo uma casa com duas divisões, A e B, ligadas entre sí, um robô inicialmente localizado na divisao A, com dois braços, inicialmente vazios, duas caixas, 1 e 2, e o objectivo de colocar a caixa 1 na divisão B.

```
(define (problem duasSalas)
     (:domain robot)
     (:objects divisaoA divisaoB caixa1 caixa2
          bracoEsquerdo bracoDireito)
     (:init (divisao divisaoA)
          (divisao divisaoB)
          (caixa caixa1)
          (caixa caixa2)
          (braco bracoEsquerdo)
          (braco bracoDireito)
          (em-robot divisaoA)
          (livre bracoEsquerdo)
          (livre bracoDireito)
          (em caixa1 divisaoA)
          (em caixa2 divisaoA)
          (ligado divisaoA divisaoB)
          (ligado divisaoB divisaoA))
     (:goal (em caixa1 divisaoB)))
```

Um plano para este problema consistiria na execução da acção apanhar da caixa1 com um dos braços, deslocar para a divisão b, e largar a caixa1 com o braço usado para a apanhar, na divisão b.

Modele este domínio em PDDL de modo a que possa resolver problemas arbitrários do tipo do problema descrito i.e. considerando vários braços, várias caixas, varias casas com várias configurações, e objectivos que prevejam localizações específicas para várias caixas. Pode testar a sua implementação neste problema.

III - Existem várias cidades, cada uma contendo vários locais, alguns dos quais são aeroportos. Há também camiões, que podem ser conduzidos dentro de uma única cidade, e aviões, que podem voar entre os aeroportos. O objetivo é transportar alguns objectos de vários locais para vários novos locais.

Modele este domínio em PDDL usando 9 predicados: pacote/1, camiao/1, aviao/1, local/1, aeroporto/1, cidade/1, em_local/2, em_veiculo/2 e em_cidade/2. Pode testar a sua implementação neste problema.