Lista de Exercícios de Planejamento

Nome: _____

Planejamento Clássico

1. Considerando a ação *move*, mostrada no Listing 1, onde um robô se desloca entre diferentes posições em uma matriz sempre ocupando um espaço 1x1, responda as questões abaixo.

Listing 1: ação `move` em `PDDL`

```
(:action move
  :parameters (?rb ?here ?there)
  :precondition
  (and
    (robot ?rb)
    (adjacent ?here ?there)
    (at ?rb ?here)
    (not (blocked ?there))
  )
  :effect
  (and
    (not (at ?rb ?here))
    (not (blocked ?here))
    (at ?rb ?there)
    (blocked ?there)
  )
)
```

Listing 2: `Problema em PDDL`

```
(define (problem pb1)
  (:domain roboto)
  (:objects roboto px1y1 px2y1 p3y1 p4y1 p5y1)
  (:init
    (robot roboto)
    (adjacent px1y1 px2y1)
    (adjacent px2y1 px3y1)
    (adjacent px3y1 px4y1)
    (adjacent px4y1 px5y1)
    (at roboto px1y1)
  )
  (:goal
    (and (at roboto px5y1))
  )
)
```

(a) Por que é necessário ter efeitos negativos nesta ação?

> **Solution:** Se não for removida a proposição com a posição anterior do robô este ocuparia diversas posições ao longo do trajeto. Se não for removido o bloqueio o robô nunca poderia voltar a ocupar essas posições, outros robôs também não poderiam utilizar essas posições.

(b) O que acontece se removermos a precondição de bloqueio?

> **Solution:** O robô poderia ocupar a mesma posição de outros robôs. Só não afetaria o plano gerado se esse robô fosse o único objeto que tivesse essa característica de bloqueio.

(c) Qual o plano gerado para o problema mostrado no Listing 2 tendo apenas a ação *move*?

> **Solution:**
> move (roboto px1y1 px2y1)
> move (roboto px2y1 px3y1)
> move (roboto px3y1 px4y1)
> move (roboto px4y1 px5y1)

(d) Se pedissemos para o robô fazer o caminho contrário, tendo a posição inicial de *roboto* como *px5y1* e a posição objetivo como *px1y1*, qual seria o plano gerado?

> **Solution:** O plano gerado seria uma falha. O caminho contrário é impossível do ponto de vista do planejador, já que não existem adjacências no sentido inverso. Para resolver isso seria necessário adicionar as adjancências faltantes ou criar uma outra ação com a precondição modificada para *(adjacent ?there ?here)*.

(e) Foi necessário expandir nosso robô, o que significa que agora ele sempre ocupa 2x1 na matriz de posições, ao invés de 1x1 como antigamente. Várias peças foram trocadas, seria necessário modificar a descrição da ação *move*? Se sim, como você faria?

> **Solution:** Sim, é necessário modificar a ação *move*. Seria necessário separar em duas ações para deslocar em cada eixo, vertical ou horizontal. A descrição do problema também precisaria adicionar a informação de adjacência horizontal ou vertical, já que o modelo atual não diferencia a direção. Também seria possível utilizar condicionais e testar em qual eixo o robô se move, sem precisar criar outra ação.

Planejamento Hierárquico

1. Planejamento clássico trabalha com um conjunto de objetivos a serem atingidos, em contraste, HTN planeja de que forma? E o que tenta atingir?

   Solution: Ao invés de chegar em um conjunto de objetivos, HTN tenta atingir um conjunto de tarefas. Um planejador HTN é definido a partir de um conjunto de operadores (similar a planejamento clássico) e métodos que por sua vez decompõe as tarefas definidas pelo problema em sub-tarefas até que estas se tornem tarefas primitivas e possam ser executadas pelos operadores do planejador.

2. Explique por que planejamento HTN é mais expressivo do que planejamento clássico.

   Solution: Através de HTN podemos expressar problemas indecidíveis, enquanto que em planejamento clássico não temos esta liberdade, a desvantagem é que o ganho dessa expressividade também significa que podemos ter planos que não acabam, *i.e.* entram em *loop*.

3. Ainda considerando o cenário do robô na matriz, responda as questões abaixo:
   (a) Como você modificaria o problema mostrado no Listing 2 para ser possível ser resolvido por um HTN? Lembre-se que um HTN decompõe uma tarefa e não procura por um estado objetivo. Não se preocupe com a sintaxe de sua solução.

   Solution: Seria necessário criar uma tarefa *move_to(goal)*, que se decompõe para um conjunto vazio ao alcançar o estado objetivo ou executa uma ação *move* e faz uma chamada recursiva para si. Lembrando que é necessário marcar posições já visitadas para não entrar em *loop*. A posição atual do robô pode sempre ser unificada ou propagada entre os métodos.

   (b) A bateria do nosso robô não é mais a mesma e ele ficou muito pesado para nós mesmos carregarmos ele todo dia até a base com a tomada. Gostaríamos de poder dar uma tarefa onde o robô alcança a base e se conecta na tomada. Nosso robô tem duas ações disponíveis agora: *move* e *plug*. Qual a vantagem de usar HTN para esse problema?

   Solution: Em planejamento clássico nós estaríamos adicionando pelo menos mais uma ação a ser testada a cada novo estado, considerando apenas uma tomada. Em HTN podemos dar um método que só testa a ação *plug* quando alcança a base. Isso se torna mais importante quando temos várias ações que só acontecem em certos contextos, podemos assim dar prioridade para as ações que nos levam a solução.