EI **ETI (diurno e PL)** **IGE (diurno e PL)**

Inteligência Artificial 2012-2013

Teste de SBC. Duração: 1:30H

2013/11/22

Nota: Nos exercícios que se seguem, a programação em lógica, as regras condição-conclusão, e as regras de produção estão expressas e funcionam tal como nos sistemas usados e explicados nas aulas. As respostas devem seguir os mesmos padrões.

I – Lógica de Predicados

(4 Valores) 1 - Considera os predicados *Proposta/1*, *Requisito/2*, *Satisfeito/2* e *Válida/1*, os quais se usam como nos exemplos que se seguem:

Proposta(P1): P1 é uma proposta

Requisito(R1, P1): R1 é um requisito da proposta P1

Satisfeito(R1, P1): R1 é um requisito satisfeito pela proposta P1

Valida(P1): P1 é uma proposta válida

Usando lógica de predicados de primeira ordem, representa o conhecimento *"Se p é uma proposta e p satisfaz todos os seus requisitos, então p é uma proposta válida"*

Sugestão: começa por representar o conhecimento *"Se r é um requisito da proposta concreta P1, então P1 satisfaz r"*. Depois, usa a proposição que o representa (não esquecendo de a modificar para o caso de qualquer proposta e não apenas para a proposta P1) para formar a proposição final.

R:

A sugestão dá origem à seguinte proposição $\forall r$ (Requisito(r, P1) $\Rightarrow$ Satisfeito(r, P1))

Esta pode ser usada quase diretamente como uma das condições do conhecimento que se pretende representar

$\forall p$ {[Proposta(p) $\wedge$ $\forall r$ (Requisito(r, p) $\Rightarrow$ Satisfeito(r, p))] $\Rightarrow$ Válida(p)}

(4 Valores) 2 - Considera a seguinte base de conhecimento em lógica de predicados sobre música japonesa.

i. $\forall m$ [MusicaTradicional(m) $\Rightarrow \neg$ ($\neg$MInstrumental(m) $\wedge$ $\neg$MTeatral(m) $\wedge$ $\neg$MCôrte(m))]
ii. $\neg$MInstrumental(Rosier)
iii. $\neg$MTeatral(Rosier)
iv. $\neg$MCôrte(Rosier)

Prova sem recorrer à forma clausal que existe pelo menos uma obra musical que não é música tradicional japonesa.

R:

Objetivo: $\exists x \neg MusicaTradicional(x)$

Derivação:

i.	$\forall m\,[MusicaTradicional(m) \Rightarrow \neg\,(\neg MInstrumental(m) \wedge \neg MTeatral(m) \wedge \neg MCôrte(m))]$	Δ
ii.	$\neg MInstrumental(Rosier)$	Δ
iii.	$\neg MTeatral(Rosier)$	Δ
iv.	$\neg MCôrte(Rosier)$	Δ
v.	$MusicaTradicional(Rosier) \Rightarrow \neg\,(\neg MInstrumental(Rosier) \wedge \neg MTeatral(Rosier) \wedge \neg MCôrte(Rosier))$	i UI
vi.	$(\neg MInstrumental(Rosier) \wedge \neg MTeatral(Rosier) \wedge \neg MCôrte(Rosier))$	ii, iii, iv 2*AI
vii.	$\neg\,\neg\,(\neg MInstrumental(Rosier) \wedge \neg MTeatral(Rosier) \wedge \neg MCôrte(Rosier))$	vi DNI
viii.	$\neg MusicaTradicional(Rosier)$	vii, v MT
ix.	$\exists x \neg MusicaTradicional(x)$	viii EG

II – Representação computacional de conhecimento

(4 Valores) 3 - Usando Prolog, escreve o predicado *percurso_cumprido/2* tal que `percurso_cumprido(Percurso, Jogador)` significa que o jogador passou por todos os pontos obrigatórios do percurso. Assume que os pontos obrigatórios de um percurso estão especificados através de factos do predicado *ponto_obrigatorio/2*, e que os pontos por onde um jogador passou estão especificados através de factos do predicado *ponto_passado/3*, por exemplo

```
ponto_obrigatorio(p1, torre).     ponto_passado(p1, j1, torre).
ponto_obrigatorio(p1, ponte1).    ponto_passado(p1, j1, ponte1).
ponto_obrigatorio(p1, ponte2).    ponto_passado(p1, j1, ponte2).

ponto_obrigatorio(p2, porta1).    ponto_passado(p1, j2, torre).
                                  ponto_passado(p2, j2, porta1).
```

Exemplo

```
?- percurso_cumprido(p1, Jogador).
Jogador = j1;
False
```

R:

Apresentam-se duas soluções alternativas.

[Alternativa 1]

```
percurso_cumprido(Percurso, Jogador):-
      \+( (ponto_obrigatorio(Percurso, Ponto),
           \+ponto_passado(Percurso, Jogador, Ponto) ).
```

[Alternativa 2]

Define-se primeiro o predicado *percurso_incumprido/2*:

```
percurso_incumprido(Percurso, Jogador):-
      ponto_obrigatorio(Percurso, Ponto),
      \+ ponto_passado(Percurso, Jogador, Ponto).

percurso_cumprido(Percurso, Jogador):-
      \+ percurso_incumprido(Percurso, Jogador),
```

(4 Valores) 4 - Considera a base de conhecimento que se segue, a qual foi representada usando o método de representação da ferramenta *BRules*.

```
if   (apresentacao(A) and
      conteudo(A, Cont) and comunicacao(A, Com) and
      minConteudo(MinCont) and Cont >= MinCont and
      minComunicacao(MinCom) and Com >= MinCom and
      pesoCont(PCont) and pesoCom(PCom) and
      Aval is PCont * Cont + PCom * Com)
then avaliacao(A, Aval).

if   (apresentacao(A) and conteudo(A, Cont)and
      minConteudo(MinCont) and Cont < MinCont)
then avaliacao(A, 1).

if   (apresentacao(A) and comunicacao(A, Com) and
      minComunicacao(MinCom) and Com < MinCom)
then avaliacao(A, 1).
```

```
% Configuração: Valores mínimos

fact(minComunicacao(2)).
fact(minConteudo(3)).

% Configuração: Pesos

fact(pesoCont(0.4)).
fact(pesoCom(0.6)).
```

```
% Factos

fact(apresentacao(robot)).
fact(apresentacao(deep_blue)).
fact(apresentacao(quinta)).

% O que o sistema pode perguntar

askable(conteudo(_, _)).
askable(comunicacao(_, _)).
```

Imagina agora a seguinte interrogação:

```
?- solve(avaliacao(A, Aval)).
```

O sistema tenta dar quantas soluções, em reação à interrogação apresentada? No processo de determinar a primeira resposta, que perguntas faz o sistema? Especifica a ordem pela qual as perguntas são feitas.

R:

É feita a tentativa de apresentar três respostas. No processo de determinar a primeira resposta, o sistema faz as seguintes perguntas, pela ordem especificada:

i. conteudo(robot, Cont) *[Que valor tem o conteúdo?]*

ii. comunicacao(robot, Com) *[Que valor tem a comunicação?]*

(4 Valores) 5 - Escreve as regras de um sistema de produção para controlar um robot que arruma blocos – cada um na sua posição desejada - e, quando TODOS OS BLOCOS ESTIVEREM NA POSIÇÃO DESEJADA, pinta os blocos. No final, o robot deve dizer que os blocos estão todos pintados. *paint_blocks/0* é o programa principal que põe o sistema de produção em funcionamento:

```
paint_blocks :-
      assert(pos(a, 2)),
      assert(pos(b, 5)),
      assert(pos(c, 18)),
      assert(painting),
      psys.
```

O sistema de produção poderá usar as ações que se descrevem seguidamente, as quais já estão implementadas.

move(Block, Pos1, Pos2)	O robot move o bloco *Block* da posição *Pos1* para a posição *Pos2*. Em termos internos, o facto com a posição original do bloco movido é substituído pelo facto com a sua posição final.
paint(Block)	O robot pinta o bloco *Block*. Em termos internos, é criado o facto `painted(Block).`
say_blocks_painted	O robot diz que já pintou todos os blocos. Esta ação deve ser executada quando todos os blocos foram pintados.
assert(Fact)	O robot cria o facto *Fact* (insere-o na memória do sistema)
retract(Fact)	O robot remove o facto *Fact* da memória do sistema

Durante a operação do sistema, são criados, removidos ou alterados factos dos seguintes predicados:

painting	Facto criado no início para que as regras se usem apenas quando este facto existe. No fim, deve ser apagado
all_blocks_ready	Facto criado pelas regras do sistema de produção quando todos os blocos estão nas posições desejadas onde podem ser pintados. No final, este facto deve ser apagado.
painted(Block)	Facto criado cada vez que um bloco é pintado.
pos(Block, Pos)	Facto que reflete a posição de um bloco (num dado instante).

Configuração inicial

Da base de conhecimento do sistema, têm de constar os factos do predicado *block/1* que especifica todos os blocos existentes, por exemplo `block(a)`, e os factos do predicado *desired_position/2* que estabelece a posição desejada de cada bloco, por exemplo `desired_position(a, 1)`. As posições iniciais dos blocos são especificadas através de factos do predicado *pos/2* pelo programa principal do sistema, *paint_blocks/0*. Para simplificar o problema, garante-se que nenhum bloco é inicialmente posicionado na posição desejada de outro bloco.

Além dos predicados já mencionados, o sistema poderá recorrer a qualquer predicado da linguagem Prolog, por exemplo \= (diferente).

R:

```
if   (painting and block(Block) and
      pos(Block, Pos1) and desired_pos(Block, Pos2) and
      Pos1 \= Pos2)
then move(Block, Pos1, Pos2).

if   (painting and \+ all_blocks_ready and
      \+( (block(Block) and
          pos(Block, Pos1) and desired_pos(Block, Pos2) and
          Pos1 \= Pos2) ))
then assert(all_blocks_ready).

if   (painting and all_blocks_ready and
      block(Block) and \+ painted(Block))
then paint(Block).

if   (painting and all_blocks_ready and
      \+( (block(Block) and \+ painted(Block)) ))
then (retract(painting),
      retract(all_blocks_ready),
      say_blocks_painted).
```