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

Inteligência Artificial 2016/2017

Teste de Sistemas Baseados em Conhecimento

Lê cuidadosamente as instruções desta prova feita em moldes não habituais.

Para passar é forçoso responder a todas as perguntas obrigatórias (Grupo 1). Embora estas perguntas valham 12 valores, existe uma tolerância de 2 valores para pequenos erros. O Grupo 2 não será sequer corrigido em provas sem o mínimo de 10 valores no Grupo 1.

Quem quiser pode responder também às perguntas facultativas do Grupo 2, candidatando-se assim a uma nota de 10 a 20.

Grupo 1 – Perguntas obrigatórias (15 Minutos)

1 – Imagina que tens o predicado *Amigo/2*, tal que Amigo(x, y) significa que *x* e *y* são amigos. Usando a forma habitual da lógica de predicados de primeira ordem, representa o conhecimento

O Lúcifer e a Isabel Alexandre têm pelo menos um amigo comum

R:

$\exists x\ [\text{Amigo}(\text{Lúcifer}, x) \wedge \text{Amigo}(\text{Isabel}, x)]$

2 – Mostra, sem usar a forma clausal, que R se pode derivar da seguinte base de conhecimento

1. $(P \wedge Q) \Rightarrow R$ Δ
2. P Δ
3. Q Δ

R:

1. $(P \wedge Q) \Rightarrow R$ Δ
2. P Δ
3. Q Δ

---

4. $P \wedge Q$ 2, 3 AI
5. R 1, 4 MP

3 - Imagina que tens factos dos predicados *aluno/2* e *cadeira_curso/2*, como os exemplos que se seguem.

```
aluno(ana, ige).
aluno(daria, ei).
aluno(joao, eti).
aluno(inês, ei).
```

```
cadeira_curso(ia, ei).
cadeira_curso(ia, eti).
cadeira_curso(ia, ige).
cadeira_curso(pi, ei).
cadeira_curso(gf, ige).
cadeira_curso(mc, eti).
```

Usando a linguagem *Prolog*, define o predicado *aluno_cadeira/2* que relaciona alunos e cadeiras, recorrendo aos predicados já existentes. aluno_cadeira(joao, ia) significa que o aluno João tem a cadeira Inteligência Artificial. O seguinte seria um exemplo de interação com o predicado:

```
?- aluno_cadeira(joao, X).
X = ia;
X = mc
```

R:

```
aluno_cadeira(Aluno, UC):-
      aluno(Aluno, Curso),
      cadeira_curso(UC, Curso).
```

4 – Num determinado sistema de *software* com sensores para acompanhamento de doentes, sempre que o ritmo cardíaco do paciente é maior que um valor máximo predefinido, é enviado uma mensagem de aviso para o centro hospitalar do paciente.

Imagina que podes usar a ação e os factos que se descrevem seguidamente.

enviar_aviso(T, Ritmo)	O sistema envia, para o centro hospitalar do paciente, o valor do seu ritmo cardíaco, relativo ao instante de tempo T. Além disso cria o facto de controlo aviso_enviado(T).

ritmo(T, Ritmo)	Reflete a leitura do sensor. No instante T, o ritmo cardíaco do paciente tem o valor Ritmo.
aviso_enviado(T)	Resulta da ação enviar_aviso(T, Ritmo). Significa que o sistema enviou um aviso para o centro hospitalar relativo ao ritmo cardíaco do paciente no instante T.
max(Ritmo)	Valor máximo permitido para o ritmo cardíaco

Usando o *PSys*, escreve a regra de produção que envia um aviso para o centro hospitalar, informando sobre o ritmo cardíaco do paciente quando o seu valor é maior que o máximo permitido. A regra só deve enviar aviso relativo ao instante T se ainda não o tiver feito.

R:

```
if   (max(M) and ritmo(T, V) and V > M and \+ aviso_enviado(T))
then enviar_aviso(T, R).
```

Grupo 2 – Perguntas facultativas (15 Minutos)

Responde apenas a uma das perguntas que se apresentam seguidamente.

5 – Considera a seguinte base de conhecimento em lógica de predicados de primeira ordem

1. *Quem tem sorte ao jogo, não tem sorte ao amor*
   $\forall x$ ([$\forall s$ Situação(s, Jogo) $\Rightarrow$TemSorte(x, s)] $\Rightarrow$[$\forall t$ Situação(t, Amor)$\Rightarrow$ $\neg$TemSorte(x, t)])
2. *A Lucrecia tem sorte em pelo menos um caso de amor*
   $\exists s$(Situação(s, Amor) $\wedge$ TemSorte(Lucrecia, s))

Prova, por refutação, que existe pelo menos uma situação de jogo em que a Lucrecia não tem sorte. Durante a resolução usa os resultados que se seguem, sempre que necessitares de converter proposições com o mesmo formato

- A conversão de $\exists s(P(s) \wedge Q(s))$ para forma clausal resulta nas cláusulas
  {P(SK)}
  {Q(SK)}
- A conversão de $\neg\exists s(P(s) \wedge Q(s))$ para forma clausal resulta na cláusula
  $\{\neg P(s), \neg Q(s)\}$

Durante a inferência, poderás ter de te lembrar que o resultado de aplicar uma função ao seu argumento é uma constante.

Recorrendo à forma clausal, representa o conhecimento das proposições 1 e 2 em Prolog. Para isso, é conveniente usar o predicado *TemAzar/2* em vez da negação de *TemSorte/2*.

Como perguntarias qual a situação em que a Lucrécia não tem sorte (i.e., a situação em que tem azar)?

R:

Para provar por refutação, é necessário converter a base de conhecimento para forma clausal, representar o objetivo, nega-lo e convertê-lo para forma clausal, e finalmente aplicar resolução até chegar a uma contradição.

Conversão da base de conhecimento para forma clausal

$\forall x$ ([$\forall s$ (Situação(s, Jogo) $\Rightarrow$TemSorte(x, s))] $\Rightarrow$[$\forall t$ (Situação(t, Amor)$\Rightarrow$ $\neg$TemSorte(x, t))])

1. Substituir implicações por disjunções equivalentes $(A \Rightarrow B) \equiv (\neg A \vee B)$

$\forall x$ ($\neg$[$\forall s$ ($\neg$Situação(s, Jogo) $\vee$ TemSorte(x, s))] $\vee$ [$\forall t$ ($\neg$Situação(t, Amor) $\vee$ $\neg$TemSorte(x, t))])

2. Mover as negações para os literais.

   $(\neg\forall s\ P) \equiv (\exists s\ \neg P)$
   $\forall x$ ([$\exists s$ $\neg$($\neg$Situação(s, Jogo) $\vee$ TemSorte(x, s))] $\vee$ [$\forall t$ ($\neg$Situação(t, Amor) $\vee$ $\neg$TemSorte(x, t))])

   $\neg(a \vee b) \equiv (\neg a \wedge \neg b)$
   $\forall x$ ([$\exists s$ ($\neg\neg$Situação(s, Jogo) $\wedge$ $\neg$TemSorte(x, s))] $\vee$ [$\forall t$ ($\neg$Situação(t, Amor) $\vee$ $\neg$TemSorte(x, t))])

   $(\neg\neg a) \equiv a$
   $\forall x$ ([$\exists s$ (Situação(s, Jogo) $\wedge$ $\neg$TemSorte(x, s))] $\vee$ [$\forall t$ ($\neg$Situação(t, Amor) $\vee$ $\neg$TemSorte(x, t))])

3. Skolemizar. A variável quantificada existencialmente, s, está quantificada no âmbito da quantificação universal da variável x, por isso s é substituída por uma função de Skolem de x

   $\forall x$ ([(Situação(SK(x), Jogo) $\wedge$ $\neg$TemSorte(x, SK(x)))] $\vee$ [$\forall t$ ($\neg$Situação(t, Amor) $\vee$ $\neg$TemSorte(x, t))])

4. Remover os ∀s

   (Situação(SK(x), Jogo) ∧ ¬TemSorte(x, SK(x)) ∨ (¬Situação(t, Amor) ∨ ¬TemSorte(x, t))

5. Converter numa conjunção de disjunções.

   (a ∨ (b∧c)) ≡ ((a∨b) ∧ (a∨c))

   (¬Situação(t, Amor) ∨ ¬TemSorte(x, t) ∨ Situação(SK(x), Jogo)) ∧

   (¬Situação(t, Amor) ∨ ¬TemSorte(x, t) ∨ ¬TemSorte(x, SK(x)))

6. Escrever em forma de cláusulas
   {¬Situação(t, Amor), ¬TemSorte(x, t), Situação(SK(x), Jogo)}
   {¬Situação(t, Amor), ¬TemSorte(x, t), ¬TemSorte(x, SK(x))}

Conversão da base de conhecimento para forma clausal (Continuação)

Sabendo que a conversão de ∃s (P(s) ∧ Q(s)) origina as cláusulas {P(SK)} e {Q(SK)}, a conversão de ∃s (Situação(s, Amor) ∧ TemSorte(Lucrecia, s)) resulta nas cláusulas

{Situação(SK2, Amor)}

{TemSorte(Lucrecia, SK2)}

Usámos SK2 porque já havíamos usado SK anteriormente.

Objetivo

∃s (Situação(s, Jogo) ∧ ¬TemSorte(Lucrécia, s))

Objetivo negado

¬∃s (Situação(s, Jogo) ∧ ¬TemSorte(Lucrécia, s))

Conversão do objetivo negado para forma clausal

Sabendo que a conversão de ¬∃s (P(s) ∧ Q(s)) é a cláusula {¬P(s), ¬Q(s))}, a conversão de ¬∃s (Situação(s, Jogo) ∧ ¬TemSorte(Lucrécia, s)) resulta na cláusula

{¬Situação(s, Jogo), TemSorte(Lucrecia, s) }

Encontrar a contradição

1.	{¬Situação(t, Amor), ¬TemSorte(x, t), Situação(SK(x), Jogo)}	Δ
2.	{¬Situação(t, Amor), ¬TemSorte(x, t), ¬TemSorte(x, SK(x))}	Δ
3.	{Situação(SK2, Amor)}	Δ
4.	{TemSorte(Lucrecia, SK2)}	Δ
5.	{¬Situação(s, Jogo), TemSorte(Lucrecia, s) }	¬Objetivo

6.	{ ¬TemSorte(x, SK2), ¬TemSorte(x, SK(x)) }	3, 2
7.	{¬TemSorte(Lucrecia, SK(Lucrecia)) }	6, 4
8.	{ }	4, 7

Nota que o passo 8 é possível, apenas se SK(Lucrécia)=SK2.

Representar a base de conhecimento em Prlog

Para representar o conhecimento em Prolog, partindo da forma clausal, é conveniente usar o predicado *TemAzar/2* em vez da negação de *TemSorte/2* (numa das ocorrências da negação de *TemSorte/2*), porque a segunda cláusula só tem literais negativos.

```
situacao(sk(X), jogo):- situacao(T, amor), tem_sorte(X, T).

tem_azar(X, sk(X)):- situacao(T, amor), tem_sorte(X, T).

situacao(sk2, amor).

tem_sorte(lucrecia, sk2).
```

Interação

Para saber qual a situação em que a Lucrécia tem azar, seria efetuada a seguinte interação (a qual mostra que a resposta é sk2(lucrecia)):

```
?- tem_azar(lucrecia, Situacao).

Situacao = sk2(lucrecia)

?-
```

6 – Usando o *PSys*, escreve as regras de um sistema de regras de produção para calcular a média dos valores introduzidos pelo utilizador, através do teclado. Para terminar, o utilizador introduz a palavra *end*. As regras têm de assegurar que nenhum *lixo* criado pelas computações é deixado em memória após a média ter sido calculada. Admite que o utilizador só introduz valores corretos (números ou *end*).

O seguinte é o programa que serve de interface com o sistema de produção referido:

```
compute_average :-
      assert(goal(average_computed)),  % Se não existir, as regras não trabalham
      psys.
```

Exemplo de interação

```
?- compute_average.
Enter a number or end: 5.
Enter a number or end: 7.
Enter a number or end: end.
Average: 6
true.
```

As regras do sistema têm de prever os seguintes casos

1. Não existe registo de *input*: ler e registar o *input*
2. Existe registo de um *input* numérico (i.e., diferente de *end*): atualizar a contagem, atualizar a soma, remover o registo de *input*.
3. Existe registo de que a palavra *end* foi lida: calcular e escrever a média no monitor, apagar os factos criados pelo programa.

Na realidade, como não se pode apagar factos que não existem, a situação 3 divide-se em duas: *(i)* já foram contabilizados *inputs*; e (ii) a palavra *end* foi o único *input* introduzido (i.e., não foram contabilizados inputs). Nas tuas regras, deves considerar estas duas situações.

Para além dos recursos do *PSys*, já teus conhecidos (e.g., *write/1*, *nl/0*, *assert/1*, e *retract/1*), dispões também

das ações *read_number/1*, *update_count/0* e *update_sum/1*, cuja descrição se apresenta seguidamente, e dos factos de controlo criados pelo programa de interface, pelas ações e pelas próprias regras que desenhares:

`read_number(X)`	Lê um valor numérico ou a palavra *end*, introduzidos pelo utilizador, através do teclado. Não cria nenhum facto de controlo.
`update_count`	Remove o facto `count(N)` e substitui-o pelo facto `count(`N_1`)` em que $N_1 = N+1$. Funciona mesmo que ainda não exista um facto do predicado *count/1*. Nesse caso cria o facto `count(1)`
`update_sum(X)`	Remove o facto `sum(S)` e substitui-o pelo facto `sum(`S_1`)` em que $S_1 = S+X$. Funciona mesmo que ainda não exista um facto do predicado *sum/1*. Nesse caso cria o facto `sum(X)`

R:

```
% Não existe nenhum input
if   (goal(average_computed) and \+ acquired_input(_))
then (read_number(X), assert(acquired_input(X))).


% Foi lido um input numérico que ainda não foi processado
if   (goal(average_computed) and acquired_input(X) and X \= end)
then (update_count, update_sum(X), retract(acquired_input(_))).


% Foi lida a palavra end
% Já foram lidos e processados outros inputs
if   (goal(average_computed) and acquired_input(end) and
      count(N) and sum(S) and Av is S/N)
then (retract(goal(average_computed)),
      retract(count(_)),
      retract(sum(_)),
      retract(acquired_input(_)),
      write('Average: '), write(Av), nl).


% Foi lida a palavra end, a qual foi o primeiro input inserido
if   (goal(average_computed) and acquired_input(end) and
      \+ count(_))
then (retract(goal(average_computed)),
      retract(acquired_input(_)),
      write('No number has been entered'), nl).
```