EI **ETI** **IGE**

Inteligência Artificial 2015-2016

Exame 1. 2016/01/11. Duração: 2H

Notas:

Nos exercícios que se seguem, o Prolog 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.

Responde às perguntas 1 a 4, numa folha de prova; e responde às perguntas 5 a 8, noutra folha de prova.

I – Generalidades

1 – Responde às perguntas, sem apresentar justificações ou outro tipo de explicações. Nos casos indicados, as respostas incorretas descontam, conforme especificado.

(1 Valor) a) As regras de produção são mais adequadas para sistemas de pergunta/resposta ou para sistemas de controlo? **`[desconta 0.5]`**

R: Sistemas de controlo

(1 Valor) b) Os mecanismos de controlo da linguagem Prolog são usados em programação declarativa ou em programação procedimental? **`[desconta 0.5]`**

R: Procedimental

(0.5 Valores) c) Como se chama o operador ∀, da lógica de predicados?

R: Quantificador universal

II – Lógica de Predicados de Primeira Ordem

(2.5 Valores) 2 – Considera a seguinte base de conhecimento

i) $\exists x\ (\text{SenteAnsiedadeAbsoluta}(x) \wedge \neg\text{EstadoInfernal}(x))$
ii) $\forall p\ [\text{Pessoa}(p) \Rightarrow (\text{SenteAnsiedadeAbsoluta}(p) \Rightarrow \text{EstadoInfernal}(p))\ ]$

Prova, por refutação, que existe algo que não é pessoa. Para isso, considera as seguintes conversões para forma clausal:

$\exists x\,(\text{SenteAnsiedadeAbsoluta}(x) \wedge \neg\text{EstadoInfernal}(x))$	{ SenteAnsiedadeAbsoluta(SK) } {¬EstadoInfernal(SK) }
$\neg\exists x\ \neg P(x)$ *[Válido para qualquer proposição com este formato]*	{ P(x) }

R:

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

Objetivo negado: $\neg\exists x \neg Pessoa(x)$

Conversão para forma clausal do objetivo negado: { Pessoa(x) }, de acordo com a tabela

Conversão da segunda proposição da base de conhecimento para forma clausal:

$\forall p\ [Pessoa(p) \Rightarrow (SenteAnsiedadeAbsoluta(p) \Rightarrow EstadoInfernal(p))]$

i) Substituir as implicações $(A \Rightarrow B) \equiv (\neg A \vee B)$
   $\forall p\ [\neg Pessoa(p) \vee \neg SenteAnsiedadeAbsoluta(p) \vee EstadoInfernal(p)]$
ii) Mover as negações para os literais. N/A porque as negações presentes estão ambas nos literais
iii) Skolemização. N/A porque não existem $\exists$s
iv) Remoção dos $\forall$s
   $\neg Pessoa(p) \vee \neg SenteAnsiedadeAbsoluta(p) \vee EstadoInfernal(p)$
v) Conjunção de disjunções de literais. N/A porque já temos uma única disjunção de literais
vi) Escrever as cláusulas
   $\{\neg Pessoa(p), \neg SenteAnsiedadeAbsoluta(p), EstadoInfernal(p)\}$

Mostrar que as cláusulas do objetivo negado juntamente com as da base de conhecimento formam um conjunto contraditório:

i)	$\{ \neg Pessoa(p), \neg SenteAnsiedadeAbsoluta(p), EstadoInfernal(p)\}$	Δ
ii)	$\{ SenteAnsiedadeAbsoluta(SK)\}$	Δ
iii)	$\{\neg EstadoInfernal(SK)\}$	Δ
iv)	$\{Pessoa(x)\}$	$\neg$Obj
v)	$\{ \neg SenteAnsiedadeAbsoluta(p), EstadoInfernal(p)\}$	i, iv
vi)	$\{ \neg SenteAnsiedadeAbsoluta(SK) \}$	iii, v
vii)	$\{\}$	ii, vi

III – Representação computacional de conhecimento

(2.5 Valores) 3 – Considera o seguinte conhecimento expresso informalmente em português:

> *Se X é uma pessoa e X sente ansiedade absoluta, então está num estado infernal*

Usando os predicados da tabela que se segue, representa-o através de uma cláusula Prolog.

pessoa/1	pessoa(P) significa que P é uma pessoa
sente_ansiedade_absoluta/1	sente_ansiedade_absoluta(P) significa que P sente ansiedade absoluta
estado_infernal/1	estado_infernal(P) significa que P está num estado infernal

R:

```
estado_infernal(P):- pessoa(P), sente_ansiedade_absoluta(P).
```

(2.5 Valores) 4 – Para enfrentar demonstrações, dá muito jeito ter uma bolsa de redução ao absurdo. Independentemente do seu restante conteúdo, qualquer bolsa de redução ao absurdo tem obrigatoriamente que conter um toque pessoal. Escreve as regras de um sistema de produção que faz o seguinte: se uma bolsa de redução ao absurdo contiver um toque pessoal e

não estiver fechada, fecha-lhe a boca hermeticamente; se a bolsa não o tiver, acrescenta-lhe um toque pessoal.

O conteúdo realmente existente em cada bolsa está especificado em factos do predicado *content/2*, por exemplo

```
% content(Bag, Content)
content(bag001, [converter, resolution]).
content(bag002, [mirror, personal_touch]).
```

As bolsas existentes estão registadas em factos do predicado *bag/1*, por exemplo

```
bag(bag001).
bag(bag002).
```

O sistema pode recorrer apenas às ações que se descrevem na seguinte tabela:

add_personal_touch(Bag)	O sistema adiciona um toque pessoal à bolsa de redução ao absurdo especificada. Altera o facto do predicado *content/2*.
close_bag(Bag)	O sistema fecha hermeticamente a bolsa de redução ao absurdo especificada. Cria o facto `closed(Bag).`

As ações descritas encarregam-se de gerir os factos de controlo necessários. As regras do sistema de produção não têm de se ocupar disso; apenas têm de usar os predicados *content/2* e *closed/1* para testar o estado atual da bolsa de redução ao absurdo. Além disso, as ações descritas escrevem mensagens de notificação no ecrã, para se saber o que está a acontecer.

Nas condições das regras deste exercício, podem usar-se todos os predicados do Prolog dados nas aulas.

R:

```
if   (bag(Bag) and \+ closed(Bag) and
      content(Bag,Content) and member(personal_touch, Content))
then close_bag(Bag).

if   (bag(Bag) and content(Bag, Content) and
      \+ member(personal_touch, Content))
then add_personal_touch(Bag).
```

(2.5 Valores) 5 – Como sabes, as demonstrações são animais perigosíssimos que requerem cuidados aturados. No entanto podemos abordá-las de forma relativamente certeira:

> *Se a demonstração vem de cornos baixos e cabeça balançante, o melhor é pegá-la de cernelha.*
>
> *Se a demonstração vem de cornos levantados e cabeça direita, o melhor é pegá-la de caras.*
>
> *Para uma pega de caras, o matemático tem de estar posicionado em frente da demonstração, virado para ela (se não estiver na posição certa, tem de se pôr primeiro na posição certa).*
>
> *Para uma pega de cernelha, o matemático tem de estar posicionado lateralmente à demonstração (se não estiver na posição certa, tem de se pôr primeiro na posição certa).*

Escreve as regras de produção que representam estes conhecimentos práticos para lidar com demonstrações. Para tal usa os predicados e ações que se descrevem seguidamente (sem as implementar) e ainda quaisquer outros recursos de tecnologias baseadas no Prolog, que tenham sido dados nas aulas, por exemplo o *assert/1* e o *retract/1*.

Predicados	
demo(D)	D é uma demonstração
cornos_baixos(D)	A demonstração vem de cornos baixos
cornos_levantados(D)	A demonstração vem de cornos levantados
cabeca_balancante(D)	A demonstração vem de cabeça balançante
cabeca_direita(D)	A demostração vem de cabeça direita
de_lado_para(D)	O matemático está de lado para a demonstração
de_frente_para(D)	O matemático está de frente para a demonstração
objetivo(P)	O matemático tem o objetivo P

Ações	
posicionar_de_lado_para(D)	Ação em que o matemático se posiciona de lado para a demonstração D
posicionar_de_frente_para(D)	Ação em que o matemático se posiciona de frente para a demonstração D
pegar_cernelha(D)	Ação em que o matemático pega a demostração D, de cernelha
pegar_caras(D)	Ação em que o matemático pega a demostração D, de caras

Depois de pegada a demonstração, o sistema não deve voltar a pegá-la. Para isso, o sistema de produção recorre a factos com a especificação do objetivo atual do sistema. Inicialmente, o programa de interface com o sistema de produção cria o facto objetivo(pega(D)) em que D é o nome da demonstração.

Os únicos factos a criar, alterar ou apagar explicitamente pelas regras são os factos do predicado *objetivo/1*. As ações descritas escrevem mensagens no ecrã para que se saiba o que está a acontecer.

R:

Alternativa 1

```
if    (demo(D) and objetivo(pega(D)) and
       cornos_baixos(D) and cabeca_balancante(D))
then  (assert(objetivo(pega_cernelha(D))),
       retract(objetivo(pega(D)))).

if    (demo(D) and objetivo(pega(D)) and
       cornos_levantados(D) and cabeca_direita(D))
then  (assert(objetivo(pega_caras(D))),
       retract(objetivo(pega(D)))).

if    (demo(D) and objetivo(pega_cernelha(D)) and
       \+ de_lado_para(D))
then  posicionar_de_lado_para(D).

if    (demo(D) and objetivo(pega_caras(D)) and
       \+ de_frente_para(D))
then  posicionar_de_frente_para(D).

if    (demo(D) and objetivo(pega_cernelha(D)) and
       de_lado_para(D))
then  (pegar_cernelha(D), retract(objetivo(pega_cernelha(D)))).
```

```
if    (demo(D) and objetivo(pega_caras(D)) and de_frente_para(D))
then  (pegar_caras(D), retract(objetivo(pega_caras(D)))).
```

Alternativa 2

```
if    (demo(D) and objetivo(pega(D)) and cornos_baixos(D) and
       cabeca_balancante(D) and de_lado_para(D))
then  (pegar_cernelha(D), retract(objetivo(pega(D)))).

if    (demo(D) and objetivo(pega(D)) and cornos_baixos(D) and
       cabeca_balancante(D) and \+ de_lado_para(D))
then  (posicionar_de_lado_para(D), pegar_cernelha(D),
       retract(objetivo(pega(D)))).

if    (demo(D) and objetivo(pega(D)) and cornos_levantados(D) and
       cabeca_direita(D) and de_frente_para(D))
then  (pegar_caras(D), retract(objetivo(pega(D)))).

if    (demo(D) and objetivo(pega(D)) and cornos_levantados(D) and
       cabeca_direita(D) and \+ de_frente_para(D))
then  (posicionar_de_frente_para(D), pegar_caras(D),
       retract(objetivo(pega(D)))).
```

IV – Linguagem de Programação Prolog (Prolog Declarativo)

Neste grupo, apenas se pode usar Prolog declarativo. Mecanismos de controlo da linguagem, por exemplo *assert/1*, *retract/1*, *repeat/0*, *cut*, *read/1* e *write/1* não podem ser usados.

(2.5 Valores) 6 – Escreve o predicado *mirror/2*, tal que mirror(L1, L2) significa que *L2* é a concatenação de *L1* com a inversão de *L1*.

Exemplos

```
?- mirror([a, b, c], L).
L = [a, b, c, c, b, a]
```

```
?- mirror([], L).
L = []
```

Se te der jeito, podes usar os predicados do Prolog dados nas aulas, sem os implementar. Não faças validações.

R:

```
mirror(L1, L2):-
      reverse(L1, R1),
      append(L1, R1, L2).
```

(2.5 Valores) 7– Usando recursividade, escreve o predicado declarativo *folded/2*, tal que folded(L1, L2) significa que *L2* é a lista *L1* sujeita a *"dobras"*: (enquanto for possível) os dois primeiros elementos de *L2* são o primeiro e o último de *L1*, os dois elementos seguintes de *L2*, são o segundo e o penúltimo de *L1*, etc. Por exemplo,

```
?- folded([1, 2, 3, 4, 5, 6], L).
L = [1, 6, 2, 5, 3, 4]

?- folded([a], L).
L = [a]

?- folded([], L).
L = []
```

Se te der jeito, usa o predicado *last/3*, sem o definir. last(L, F, X) significa que *F* é a lista de todos os elementos de *L*, exceto o último; e *X* é o último elemento de *L*. A tua solução tem que verificar que *L1* é uma lista de dados atómicos, e que *L2* é uma lista ou uma variável.

Exemplo

```
?- last([a, b, c], F, X).
F = [a, b] X = c
```

R:

```
folded(L1, L2):-
     is_list(L1), (var(L2); is_list(L2)),
     aux_folded(L1, L2).

aux_folded([X|L1], [X, Y | L2]):-
     atomic(X),
     last(L1, F1, Y),
     aux_folded(F1, L2).
aux_folded([X], [X]).
aux_folded([], []).
```

V – Linguagem de Programação Prolog (Prolog Procedimental)

Exceto quando se indicar o contrário, neste grupo podem usar-se todos os recursos da linguagem Prolog, desde que tenham sido dados nas aulas.

(2.5 Valores) 8 – Admite que tens um conjunto de factos do predicado *compra/3* que registam as tuas compras (produto comprado, categoria da compra, e preço):

```
compra(blusão, senhora, 200).
compra(saia, senhora, 25).
compra(calças, criança, 25).
compra(cachecol, homem, 20).
compra(sapatos, homem, 150).
```

Sem recorrer aos predicados da família *findall*, escreve o procedimento *soma_compras/1* que imprime, no ecrã do computador, o valor da soma das compras feitas, da categoria especificada, por exemplo:

```
?- soma_compras(senhora).
Valor gasto em compras de senhora: 225
true.
```

O procedimento deve funcionar bem para qualquer número de compras. Não faças validações. Se te der jeito, podes usar o *writelist/1*, sem o implementar. Não faças validações.

R:

```
soma_compras(Cat):-
      assert(soma(0)),
      compra(_, Cat, Valor),
      atualiza_soma(Valor),
      fail.
soma_compras(Cat):-
      retract(soma(Val)),
      writelist(['Valor gasto em compras de ', Cat, ': ', Val]), nl.

atualiza_soma(Valor):-
      retract(soma(S)),
      S1 is S + Valor,
      assert(soma(S1)),
      !.
```