EI **ETI** **IGE**

Inteligência Artificial 2014-2015

Segundo Exame. Duração: 2:00H

2015/01/26

Nota: Nos exercícios que se seguem, o Prolog, 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. Exceto quando algo em contrário for especificado, nos exercícios em Prolog ou usando uma tecnologia nele baseada podes usar todos os recursos existentes na linguagem (e.g., append/3).

I – Generalidades

1 – Responde, sem justificar, às seguintes perguntas. Nos casos assinalados, as respostas erradas descontam de acordo com o indicado.

(1 Valor) a) Conceptualmente, o *assert/1* do Prolog é um predicado, uma ação ou uma função? **`[desconta 0.3]`**

(1 Valor) b) Escreve a regra da eliminação da disjunção (OR)? **`[não desconta]`**

(0.5 Valores) c) Um sistema de representação de conhecimento usando a Programação em Lógica é um sistema simbólico ou não simbólico? **`[desconta 0.5]`**

R:

a) Ação

b)

$$\frac{P \vee Q \quad \neg P}{Q}$$

c) Simbólico

II – Lógica de predicados

Considera a seguinte base de conhecimento

1. $\neg\exists x$ Gambuzino(x)
2. $\forall x[(\text{Animal}(x) \wedge \text{Noturno}(x) \wedge \text{Estranho}(x)) \Rightarrow \text{Gambuzino}(x)]$
3. $\exists x$ (Peixe(x) $\wedge$ Pássaro(x) $\wedge$ Noturno(x))
4. $\forall x[(\text{Peixe}(x) \wedge \text{Passaro}(x)) \Rightarrow (\text{Animal}(x) \wedge \text{Estranho}(x))]$

(2.5 Valores) 2 – Recorrendo à forma clausal, mostra que a base de conhecimento apresentada é contraditória. Para isso, assume as seguintes correspondências:

$\forall x[(P_1(x) \wedge \ldots \wedge P_N(x)) \Rightarrow Q(x)]$ corresponde à cláusula $\{ \neg P_1(x), \ldots, \neg P_N(x), Q(x) \}$

$\exists x(P_1(x) \wedge \ldots \wedge P_M(x))$ corresponde às cláusulas $\{P_1(SK)\}, \ldots, \{P_M(SK)\}$

$\neg\exists x$ Gambuzino(x) corresponde à cláusula { $\neg$Gambuzino(x) }

R:

Para mostrar que a base de conhecimento é contraditória, é necessário convertê-la para forma clausal e depois aplicar resolução até produzir a contradição (i.e., uma cláusula vazia).

Conversão para forma clausal:

As proposições 1), 2) e 3) convertem-se diretamente, usando a informação do enunciado:

$\neg\exists x$ Gambuzino(x) corresponde à cláusula { $\neg$Gambuzino(x) }

$\forall x[(\text{Animal}(x) \wedge \text{Noturno}(x) \wedge \text{Estranho}(x)) \Rightarrow \text{Gambuzino}(x)]$ corresponde a { $\neg$Animal(x), $\neg$Noturno(x) , $\neg$Estranho (x), Gambuzino (x) }

$\exists x$ (Peixe(x) $\wedge$ Pássaro(x) $\wedge$ Noturno(x)) corresponde às cláusulas { Peixe(SK)}, {Pássaro(SK)}, { Noturno(SK)}

Conversão da proposição 4:

$\forall x[(\text{Peixe}(x) \wedge \text{Passaro}(x)) \Rightarrow (\text{Animal}(x) \wedge \text{Estranho}(x))]$

a) $(A \Rightarrow B) \equiv (\neg A \vee B)$
   $\forall x[\neg(\text{Peixe}(x) \wedge \text{Passaro}(x)) \vee (\text{Animal}(x) \wedge \text{Estranho}(x))]$
b) Mover as negações para os literais: $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$
   $\forall x[(\neg\text{Peixe}(x) \vee \neg\text{Passaro}(x)) \vee (\text{Animal}(x) \wedge \text{Estranho}(x))]$
c) Skolemização: N/A porque não há variáveis existenciais
d) Remoção dos quantificadores universais
   $(\neg\text{Peixe}(x) \vee \neg\text{Passaro}(x)) \vee (\text{Animal}(x) \wedge \text{Estranho}(x))$
e) Produzir uma conjunção de disjunções: $(C \vee (A \wedge B)) \equiv ((C \vee A) \wedge (C \vee B))$
   $(\neg\text{Peixe}(x) \vee \neg\text{Passaro}(x) \vee \text{Animal}(x)) \wedge (\neg\text{Peixe}(x) \vee \neg\text{Passaro}(x) \vee \text{Estranho}(x))$
f) Escrever em forma de cláusulas
   {$\neg$Peixe(x), $\neg$Passaro(x), Animal(x)}
   {$\neg$Peixe(x), $\neg$Passaro(x), Estranho(x)}

Produzir a cláusula vazia:

1.	{ $\neg$Gambuzino(x) }	Δ
2.	{ $\neg$Animal(x), $\neg$Noturno(x) , $\neg$Estranho (x), Gambuzino (x) }	Δ
3.	{ Peixe(SK)}	Δ
4.	{Pássaro(SK)}	Δ
5.	{ Noturno(SK)}	Δ
6.	{$\neg$Peixe(x), $\neg$Passaro(x), Animal(x)}	Δ
7.	{$\neg$Peixe(x), $\neg$Passaro(x), Estranho(x)}	Δ
8.	{$\neg$Passaro(SK), Animal(SK)}	3, 6
9.	{Animal(SK)}	8, 4
10.	{$\neg$Passaro(SK), Estranho(SK)}	3, 7
11.	{Estranho(SK)}	10, 4
12.	{$\neg$Noturno(SK) , $\neg$Estranho(SK), Gambuzino(SK) }	9, 2
13.	{$\neg$Estranho(SK), Gambuzino(SK) }	12, 5
14.	{Gambuzino(SK) }	13, 11
15.	{}	14, 1

III – Representação computacional de conhecimento

(2.5 Valores) 3 – Considera o seguinte conhecimento

> *Para ser um brinquedo, um objeto tem de ter menos lâminas cortantes que o máximo estipulado.*
>
> *Para que um explosivo seja um brinquedo tem de ter um detonador.*
>
> *Os sapos pegajosos e cegos são brinquedos.*

Escreve um conjunto de regras que represente o conhecimento informalmente apresentado. As regras devem especificar as condições para se poder concluir que um objeto (no sentido lato) é um brinquedo. O sistema deve funcionar, como no seguinte exemplo

```
?- solve(brinquedo(X)).
X = obj1;
X = obj23
```

A representação pode usar apenas os predicados da tabela e ainda todos os predicados do BRules, o qual herda os da linguagem Prolog (e.g., *member* e >=).

brinquedo/1	brinquedo(X): X é um brinquedo
cego/1	cego(X): X é cego
explosivo/1	explosivo(X): X é um explosivo
numero_maximo_laminas/1	numero_maximo_laminas(Max): Max é o máximo número de lâminas cortantes que um objeto pode ter para ser um brinquedo
pegajoso/1	pegajoso(X): X é pegajoso
sapo/1	sapo(X): X é um sapo
tem_detonador/1	tem_detonador(X): X tem detonador
tem_lamina_cortante/2	tem_lamina_cortante(X, N): X tem N lâminas cortantes

R:

```
if   (tem_lamina_cortante(X, N) and numero_maximo_laminas(Max) and
      N < Max)
then  brinquedo(X).

if   (explosivo(X) and tem_detonador(X))
then  brinquedo(X).

if   (sapo(X) and pegajoso(X)and cego(X))
then  brinquedo(X).
```

(2.5 Valores) 4 – Imagina que tens um *robot* parvo e feio que só faz o que lhe mandam (e mal), se comer Bolas de Berlim. Escreve o conjunto de regras de produção que controlam o *robot* de modo que este coma a (única) Bola de Berlim, apenas depois de ter arrumado todos os brinquedos. O sistema de controlo do robot pode usar os recursos que se descrevem.

Predicados	
objetivo(brinquedos_arrumados)	Facto de controlo. A tarefa do robot consiste em arrumar os brinquedos. No fim, come uma Bola de Berlim
bola_berlim(X)	X é uma Bola de Berlim
brinquedo(X)	X é um brinquedo
arrumado(X)	X está arrumado

Ações	
comer(X)	O robot come a Bola de Berlim X
arrumar(X)	O robot arruma o brinquedo X. A ação cria o facto arrumado(X)

Além das ações descritas, o mecanismo de controlo do robot pode usar as ações *assert/1* e *retract/1* se necessário.

Assume que o programa que controla o *robot* tem a seguinte definição:

```
controlar_robot(Bola, Brinquedos) :-
      assert(objetivo(brinquedos_arrumados)),
      assert(bola_berlim(Bola)),
      regista_brinquedos(Brinquedos),
      psys.

regista_brinquedos([B|Brinquedos]):-
      assert(brinquedo(B)),
      regista_brinquedos(Brinquedos).
regista_brinquedos([]).
```

R:

```
if   (objetivo(brinquedos_arrumados) and
      bola_berlim(X) and
      \+( (brinquedo(Y) and \+ arrumado(Y)) ))
then  (comer(X), retract(objetivo(brinquedos_arrumados))).

if   (objetivo(brinquedos_arrumados) and
      brinquedo(X) and
      \+ arrumado(X)
then  arrumar(X).
```

IV – Prolog declarativo

A resolução dos exercícios deste grupo não pode recorrer a mecanismos de controlo, por exemplo, o cut, o repeat, o fail, o assert e o retract.

(2.5 Valores) 5 – Imagina que tens o predicado *last/3*, tal que last(L1, L2, X) significa que L2 é a lista de todos os elementos de L1, exceto o último, o qual é X, por exemplo

```
? - last([], L, X).
false

? - last([a, b, c], L, X).
L = [a, b]  X = c;
false
```

Usando obrigatoriamente o predicado *last/3*, escreve uma definição recursiva do predicado *reverse/2*, tal que reverse(L, R) significa que R é a lista dos elementos de L, mas por ordem inversa, por exemplo

```
? - reverse([a, b, c], L).
L = [c, b, a];
false
```

Não faças validações

R:

```
reverse([], []).
reverse(L, [X|Reverse]):-
      last(L, L1, X),
      reverse(L1, Reverse).
```

(2.5 Valores) 6 – Escreve uma definição recursiva do predicado *cartesian_product/3*, tal que cartesian_product(L1, L2, P) significa que a lista P é o produto cartesiano das listas L1 e L2, por exemplo

```
?- cartesian_product([1, 2, 3], [4, 5, 6], L).
L = [4, 10, 18]
```

Um produto cartesiano de duas listas obtém-se multiplicando os elementos correspondentes das duas listas. No exemplo apresentado, o resultado [4, 10, 18] obtém-se de multiplicar 1 por 4, 2 por 5 e 3 por 6. O predicado deve garantir que os seus dois primeiros argumentos são listas e que o terceiro é uma variável ou uma lista.

R:

```
cartesian_product(L1, L2, P):-
      is_list(L1), is_list(L2), (var(P); is_list(P)),
      aux_cartesian_product(L1, L2, P).

aux_cartesian_product([],[], []).
aux_cartesian_product([X|L1],[Y|L2], [Z|L3]):-
      Z is X*Y,
      aux_cartesian_product(L1, L2, L3).
```

(2.5 Valores) 7 – Escreve uma definição recursiva do predicado *t_group/3*, tal que t_group(Type, L1, L2) significa que *L2* é a lista dos elementos de *L1* que têm o tipo de dados *Type*, por exemplo

```
? - t_group(number, [1, b, [], c, d, [x, y, z], 3.5], L).
L = [1, 3.5];
false

? - t_group(list, [1, b, [], c, d, [x, y, z], 3.5], L).
L = [[],[x, y, z]];
false
```

Para isto, assume que existe o predicado *type/2*, tal que type(Term, Type) significa que o termo *Term* é do tipo de dados *Type*. Não faças validações.

R:

```
t_group(T, [X|L1], [X|L2]):-
      type(X, T),
      t_group(T, L1, L2).
t_group(T, [X|L1], L2):-
      \+ type(X, T),
      t_group(T, L1, L2).
t_group(_, [], []).
```

V – Prolog Procedimental

O exercício deste grupo pode recorrer a todo e qualquer mecanismo de controlo, por exemplo, o cut, o repeat, o fail, o assert e o retract.

(2.5 Valores) 8 – Imagina que tens uma lista de possíveis tipos de dados, mantidos num facto do predicado *types_list/1*, por exemplo

```
types_list([number, atom, list]).
```

Recorrendo a um mecanismo de repetição não recursivo, escreve o procedimento *print_types/1* que imprime, no ecrã do computador, as listas de elementos de cada tipo da lista de elementos recebida como argumento, por exemplo:

```
?- print_types([1, b, [], c, d, [x, y, z], 3.5]).
number :  [1, 3.5]
atom : [b, c, d]
list : [[],[x, y, z]]
true
```

Para isso, admite que dispões do predicado *t_group/3* (pergunta 7) já implementado e que podes e deves usar. Não faças validações.

R:

```
print_types(L):-
      types_list(Types),
      member(T, Types),
      t_group(T, L, TL),
      write(T:TL), nl,
      fail.
print_types(_).
```