EI **ETI** **IGE**

Inteligência Artificial 2014-2015

Primeiro Exame. Duração: 2:00H

2015/01/07

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 *write/1* do Prolog é um predicado, uma ação ou uma função? **`[desconta 0.3]`**

(1 Valor) b) Quantas regras de inferência se usam na forma clausal? **`[não desconta]`**

(0.5 Valores) c) Numa regra de produção *if … then …*, o que representam as expressões após o *then*? **`[não desconta]`**

R:

a) Ação

b) 1

c) Ações

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) $\wedge$ Passaro(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 – Mostra, sem recorrer a forma clausal, que a base de conhecimento apresentada é contraditória. Para isso, baste que mostres que, a partir dela, consegues derivar uma proposição e a sua negação.

R:

Para mostrar que a base de conhecimento é contraditória, mostra-se que dela se deriva $\exists x$ Gambuzino(x) e $\neg\exists x$ Gambuzino(x)

5. Peixe(SK) $\wedge$ Passaro(SK) $\wedge$ Noturno(SK) 3 EI
6. Peixe(SK) $\wedge$ Passaro(SK) 5 AE
7. Noturno(SK) 5 AE
8. (Peixe(SK) $\wedge$ Passaro(SK)) $\Rightarrow$ (Animal(SK) $\wedge$ Estranho(SK)) 4 UI
9. Animal(SK) $\wedge$ Estranho(SK) 6, 8 MP
10. Animal(SK) $\wedge$ Estranho(SK) $\wedge$ Noturno(SK) 9, 7 AI
11. (Animal(SK) $\wedge$ Noturno(SK) $\wedge$ Estranho(SK)) $\Rightarrow$ Gambuzino(SK) 4 UI
12. Gambuzino(SK) 11, 10 MP
13. $\exists$x Gambuzino(x) 12 EG

As proposições 1 e 13 são a negação uma da outra, pelo que a base de conhecimentos é contraditória.

III – Representação computacional de conhecimento

Considera os seguintes predicados

bebida /2	bebida(X, Bebida) significa que *Bebida* (e.g., champanhe ou caipirinha) é a bebida de *X*.
estado_barba/2	estado_barba(X, Estado) significa que *Estado* (e.g, por_fazer, mal_feita) é o estado da barba de *X*.
estado_pele/2	estado_pele(X, Estado) significa que *Estado* (e.g., tostada, gretada) é o estado da pele de *X*.
galinha_com_pele/1	galinha_com_pele(X) significa que *X* é uma galinha com pele.
javali/1	javali(X) significa que *X* é um javali
sentado/2	sentado(X, Local) significa que X está sentado no local especificado (e.g, esplanada)
tem_alma/1	tem_alma(X) significa que X tem alma
tipo_de_oculos/2	tipo_de_oculos(X, Tipo) significa que os óculos de X são do tipo Tipo (e.g, ver_ao_perto, escuros)
tipo_de_veu/2	tipo_de_veu(X, Tipo) significa que X tem um véu do tipo Tipo (e.g, negro, transparente)

Considera o seguinte conhecimento

> *Para teres alma, se fores um javali, tens de estar sentado numa esplanada, com barba por fazer, óculos escuros e champanhe.*
>
> *Para teres alma, se fores uma galinha com pele, tens de estar tostada, com caipirinha e véu negro.*

(2.5 Valores) 3 – Escreve um conjunto de regras que represente o conhecimento informalmente apresentado. As regras devem especificar as condições para se poder concluir que o indivíduo tem alma. O sistema deve funcionar, como no seguinte exemplo

```
?- solve(tem_alma(X)).
X = abrenuncia_imaculada;
X = ze_pumba
```

R:

```
if   (javali(X) and
      sentado(X, esplanada) and
      estado_barba(X, por_fazer) and
      tipo_de_oculos(X, escuros) and
      bebida(X, champanhe)
then  tem_alma(X).

if   (galinha_com_pele(X) and
      estado_pele(X, tostada) and
      tipo_de_veu(X, negro) and
      bebida(X, caipirinha)
then  tem_alma(X).
```

(2.5 Valores) 4 – Escreve o conjunto de regras de produção embebido na mente do javali e da galinha com pele para pedir a bebida que lhes permite ter alma. As regras propostas têm de controlar tanto o javali como a galinha. pedir_bebida(X, Tipo) é a ação executada por X (javali ou galinha) para pedir uma bebida do tipo Tipo (e.g, cerveja, champanhe).

R:

```
if   (javali(X) and
      \+ bebida(X, champanhe)
then  pedir_bebida(X, champanhe).

if   (galinha_com_pele(X) and
      \+ bebida(X, caipirinha)
then  pedir_bebida(X, caipirinha).
```

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 - Escreve 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
```

Deve garantir-se que o primeiro argumento é uma lista, e que o segundo pode ser uma lista ou uma variável.

R:

```
last(L1, L2, T):-
      is_list(L1),
      (var(L2); is_list(L2)),
      aux_last(L1, L2, T).

aux_last([X], [], X).
aux_last([X|L1], [X|L2], Last):-
      aux_last(L1, L2, Last).
```

(2.5 Valores) 6 – Usando o predicado *type/2*, que devolve o tipo de dados de um termo, define o predicado *types/2*, tal que types(L, Types) significa que *Types* é a lista, sem repetições, dos tipos

dos elementos da lista *L*, por exemplo

```
? - types([1, b, [], c, d, [x, y, z], 3.5], L).
L = [number, atom, list];
False
```

Não podes usar o *nodups/2*, mas podes inspirar-te na sua definição.

R:

Apresentam-se duas soluções: a da equipa docente e a do aluno Tommaso d'Orsi por ser mais simples (pelo menos, do nosso ponto de vista).

Solução da equipa docente:

```
types([X|L], Types):-
      type(X, T),
      member(Y, L), type(Y, T),
      types(L, Types).
types([X|L], [T|Types]):-
      type(X, T),
      \+( (member(Y, L), type(Y, T)) ),
      types(L, Types).
types([], []).
```

Solução pelo menos mais elegante (Tommaso d'Orsi):

```
types([X|Terms], [T|Types]):-
      type(X, T),
      types(Terms, Types),
      \+ member(T, Types).
types([X|Terms], Types):-
      type(X, T),
      types(Terms, Types),
      member(T, Types).
types([], []).
```

V – Prolog Procedimental

Os exercícios deste grupo podem recorrer a todo e qualquer mecanismo de controlo, por exemplo, o cut, o repeat, o fail, o assert e o retract.

(2.5 Valores) 7 – Imagina que tens um conjunto de listas de termos Prolog, mantidas em factos do predicado *terms_list/1*, por exemplo

```
terms_list([1, b, [], c, d, [x, y, z], 3.5]).

terms_list([a, b]).
```

Escreve o procedimento *print_types/0* que imprime, no ecrã do computador, as listas de tipos de dados dos elementos de cada uma dessas listas, por exemplo:

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

Para isso, admite que dispões do predicado *types/2* (pergunta 6) já implementado e que podes e deves usar. Podes usar qualquer outro predicado existente em Prolog.

R:

```
print_types:-
      terms_list(Data),
      write(Data), nl,
      types(Data, Types),
      member(T, Types),
      tab(5), write(T), nl,
      fail.
print_types.
```

(2.5 Valores) 8 – Imagina que tens um conjunto de listas de termos Prolog, mantidas num ficheiro, cada lista numa linha terminada por um ponto, de modo que possam ser lidas com o *read/1*.

Escreve o procedimento *print_types/1* que imprime, no ecrã do computador, as listas de tipos de dados dos elementos de cada uma dessas listas, por exemplo:

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

Para isso, admite que dispões do predicado *types/2* (pergunta 6) já implementado e que podes e deves usar. Podes usar qualquer outro predicado existente em Prolog.

R:

```
print_types(InFile):-
      see(InFile),
      repeat,
            read(Data),
            process_data(Data), !,
      seen.

process_data(end_of_file):- !.
process_data(Data):-
      write(Data),nl,
      types(Data, Types),
      member(T, Types),
      tab(5), write(T), nl,
      fail.
```