EI **ETI** **IGE**

Inteligência Artificial 2015-2016

Teste de Prolog

2015/12/18

I – 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.

A não ser se for explicitamente dito o contrário, podem usar-se predicados declarativos existentes na linguagem, por exemplo *member/2*, *append/3* e *select/3*.

(4 Valores) 1 – Escreve uma definição recursiva do predicado *oddElementsList/2*, tal que oddElementsList(L1, L2) significa que a lista *L2* é formada pelos elementos de ordem ímpar da lista *L1*, assumindo que o primeiro elemento de uma lista é o elemento de ordem 1. Não faças validações dos argumentos do predicado.

Exemplos

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

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

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

Sugestão: Na definição, representa a lista de entrada como sendo constituída por dois elementos seguidos do resto.

R:

```
oddElementsList([X, _|Rest1], [X|Rest2]):-
      oddElementsList(Rest1, Rest2).

oddElementsList([X], [X]).

oddElementsList([], []).
```

(4 Valores) 2 – Escreve uma definição recursiva do predicado *interleavedList/3*, tal que interleavedList(L1, L2, L3) significa que a lista *L3* é formada intercaladamente por elementos de *L1* e de *L2*. Garante que *L1* e *L2* são listas de números ímpares e pares, respetivamente, e que *L3* é uma variável ou uma lista. Para tal, define os predicados *even/1* (par) e *odd/1* (ímpar) que verificam se um termo é um número par ou ímpar, respetivamente.

Exemplos

```
?- interleavedList([a, b, c], [2, 4, 6], L).
false

?- interleavedList([1, 5, 9], [2, 6, 10], L).
L = [1, 2, 5, 6, 9, 10]
```

Informação: A função aritmética *mod/2*, existente no Prolog, devolve o resto da divisão inteira do primeiro argumento pelo segundo, por exemplo mod(4, 2) =:= 0. A função *mod* gera uma exceção se receber argumentos não numéricos.

R:

```
interleavedList(L1, L2, L3):-
      is_list(L1), is_list(L2), (var(L3); is_list(L3)),
      iList(L1, L2, L3).

iList([X|L1], [Y|L2], [X, Y | L3]):-
      odd(X), even(Y),
      iList(L1, L2, L3).
iList([], [], []).

odd(X):-
      integer(X),
      X mod 2 =:= 1.

even(X):-
      integer(X),
      X mod 2 =:= 0.
```

(4 Valores) 3 – Escreve o predicado *splitList/3* tal que splitList(L1, L2, L3) significa que *L2* é a lista dos elementos de *L1* que são vogais, e *L3* é a lista dos elementos de *L1* que não são vogais.

Por exemplo

```
?- splitList([a, b, c, d, e, f, g, h, i], V, C).
V = [a, e, i]   C = [b, c, d, f, g, h]

?- splitList([b, c, d], V, C).
V = []     C = [b, c, d]
```

Não faças validações de dados.

R:

```
splitList([X|L1], [X|L2], L3):-
      vowel(X),
      splitList(L1, L2, L3).
splitList([X|L1], L2, [X|L3]):-
      \+ vowel(X),
      splitList(L1, L2, L3).
splitList([], [], []).

vowel(X):- member(X, [a, e, i, o, u]).
```

II – Prolog Procedimental

Neste grupo podem usar mecanismos de controlo, em Prolog, por exemplo *assert/1*, *retract/1*, *repeat/0*, *cut*, e *fail/0*. Podem usar também usar mecanismos de input/output (*read/1* e *write/1)*.

(4 Valores) 4 – Recorrendo a um mecanismo de repetição por falha, escreve o procedimento *printConjugation/1*, para imprimir, no ecrã do computador, todas as formas verbais do presente do indicativo do verbo especificado, por exemplo

```
?- printConjugation(amar).
   amo
   amas
   ama
   amamos
   amais
   amam
true
```

Para realizar o exercício admite que existe o predicado *verbConjugation/4*, tal que verbConjugation(Verb, Person, Number, V) significa que *V* é a forma verbal especificada, do presente do indicativo, do verbo especificado, por exemplo:

```
?- verbConjugation(amar, 1, singular, V).
V = amo
```

Sabendo que existem as pessoas (*Person*) *1*, *2* e *3*, e os números (*Number*) *singular* e *plural*, cria os predicados v*person/1* e v*number/1* que geram as três pessoas e os dois números da conjugação verbal:

```
?- vperson(P).
P = 1;
P = 2;
P = 3;
false
```

```
?- vnumber(N).
N = singular;
N = plural;
false
```

Estes dois predicados, além de *verbConjugation/4*, são necessários para a implementação do procedimento *printConjugation/1*.

Não faças validações.

R:

```
printConjugation(Verb):-
      vnumber(N),
      vperson(P),
      verbConjugation(Verb, P, N, V),
      write(V), nl,
      fail.
printConjugation(_).
```

```
vnumber(singular).
vnumber(plural).
```

```
vperson(P):- member(P, [1, 2, 3]).
```

(4 Valores) 5 – Recorrendo a um mecanismo de repetição por falha, escreve o predicado *iPrintConjugation/0* (*interactive printConjugation*) que repete o seguinte processo: *(i)* lê um verbo do utilizador, e *(ii)* imprime a sua conjugação no presente do indicativo. A repetição termina quando o utilizador introduz a palavra *end*, em vez de introduzir um verbo. Ao terminar, o procedimento deve escrever, no ecrã, o número de verbos conjugados.

Exemplo:

```
?- iPrintConjugation.
 Verb (or end to finish): amar.
 amo
 amas
 ama
 amamos
 amais
 amam
 Verb (or end to finish): correr.
 corro
 corres
 corre
 corremos
 correis
 correm
 Verb (or end to finish): end.
 Thanks for your preference. 2 verbs were conjugated.
true.
```

Para realizares o exercício, admite que tens o procedimento *printConjugation/1*, do exercício anterior. Não faças validações.

R:

```
iPrintConjugation:-
   assert(vcounter(0)),
   repeat,
       read_verb(Verb),
       process_verb(Verb),
   !,
   retract(vcounter(N)),
   writelist([N, ' verbs were conjugated.']), nl.

read_verb(Verb):-
   write('Verb (or end to finish): '),
   read(Verb).

process_verb(end):- !.
process_verb(Verb):-
   printConjugation(Verb),
   increase_counter,
   fail.

increase_counter :-
   retract(vcounter(N)),
   N1 is N + 1,
   assert(vcounter(N1)),
   !.
```