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

Inteligência Artificial 2013-2014

Teste de Prolog

2013/12/20

As resoluções das perguntas deste teste devem funcionar para casos gerais; e não apenas para casos específicos. Por exemplo, não devem funcionar apenas para listas de 4 elementos.

I – Prolog Declarativo

Neste grupo não podem ser usados os mecanismos de controlo "cut", fail e repeat, nem os meta-predicados assert/1, retract/1, findall/3, bagof/3 e setof/3.

Neste grupo, além de todos os predicados da linguagem Prolog, considera os predicados *nth/3*, *modified_list/3*, e *changed_list_element/4*. nth(L, N, X) significa que X é o elemento número N da lista L. modified_list(L1, Changes, L2) significa que L2 resulta de L1, depois de aplicar as alterações especificadas em Changes. Cada especificação de alteração é um par (N, X) em que N especifica a posição do elemento a alterar e X especifica o novo valor do elemento a alterar. changed_list_element(L1, N, X, L2) significa que L2 é o resultado de L1, depois de substituir o elemento número N de L1 por X.

Exemplos

```
?- nth([a, b, c], 2, X).
X = b

?- modified_list([a, b, c, d], [(2, 100), (4, -100)], L).
L = [a, 100, c, -100]

?- changed_list_element([a, b, c, d], 3, 100, L).
L = [a, b, 100, d]
```

(4 Valores) 1 - Este exercício é simples, nem recursividade tem; basta usar predicados que se assume existirem, sem ter de os definir. Admitindo a existência dos predicados *nth/3* e *modified_list/3* já descritos, escreve o predicado *list_with_error/2* que recebe uma lista e produz uma versão modificada dessa lista. A modificação consiste em trocar dois elementos, em posições aleatórias, da lista original. Sendo aleatória, a "troca" pode resultar numa lista igual à original. *list_with_errors/2* tem de se certificar que o seu primeiro argumento está instanciado com uma lista e que o seu segundo argumento é uma variável.

Exemplos

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

Sugestões: usa o predicado *length/2*, já existente no Prolog, para determinar o comprimento de uma lista; usa o predicado *random/3*, já existente no Prolog, para gerar dois inteiros entre 1 e o comprimento da lista. Esses inteiros são os índices dos elementos da lista a trocar.

Exemplos

```
?- length([a, b, c], N).
N = 3

?- random(1, 3, X). % gera um inteiro aleatório entre 1 e 2
X = 2
```

R:

```
list_with_error(L1, L2):-
      is_list(L1),
      var(L2),
      length(L1, Len1), N is Len1 + 1,
      random(1, N, Pos1),
      random(1, N, Pos2),
      nth(L1, Pos1, X1),
      nth(L1, Pos2, X2),
      modified_list(L1, [(Pos1, X2), (Pos2, X1)], L2).
```

(4 Valores) 2 - Admitindo que existe o predicado *changed_list_element/4*, já descrito, escreve o predicado *modified_list/3* igualmente descrito e exemplificado. O predicado não faz validações dos seus argumentos.

R:

```
modified_list(L, [], L).

modified_list(L1, [(N, X)|Changes], L2):-
      changed_list_element(L1, N, X, L),
      modified_list(L, Changes, L2).
```

(4 Valores) 3 - Escreve o predicado *changed_list_element/4* descrito e exemplificado no texto introdutório deste grupo. O predicado não faz validações dos seus argumentos.

R:

```
changed_list_element([_|L], 1, X, [X|L]).

changed_list_element([H|L1], N, X, [H|L2]):-
      N > 1,
      N1 is N - 1,
      changed_list_element(L1, N1, X, L2).
```

II – Prolog Procedimental

Cada processo repetitivo a usar neste grupo tem de recorrer a um dos mecanismos de repetição por falha.

(4 Valores) 4 - Escreve o programa *print_lists_with_errors/1* que lê um ficheiro de listas, cada uma numa linha, terminada por um ponto. Por cada lista lida, chama o predicado *list_with_error/2* (definido na pergunta 1) e imprime a lista original e a lista com erro. O procedimento não faz validações. Mesmo que não tenhas definido o predicado *list_with_errors/2*, usa-o como se ele já existisse.

Exemplo

```
?- print_lists_with_errors('Lists.txt').
[a, b, c, d]      error: [a, b, c, d] % por acaso, a lista ficou igual
[1, 2, 3, 4]      error: [1, 4, 3, 2]
[n, l, k, r]      error: [n, k, l, r]
true
```

R:

```
print_lists_with_errors(InFile):-
      see(InFile),
      repeat,
            read(List),
            process_input_list(List), !,
      seen.
process_input_list(end_of_file).
process_input_list(List1):-
      list_with_error(List1, List2),
      write(List1), tab(4), write('error: '), write(List2), nl,
      !,
      fail.
```

(4 Valores) 5 - Admite que tens uma série de factos *list/1* que armazenam listas, por exemplo `list([a, b, c, d, e])`. Escreve o predicado *print_lists_with_errors/0* que, para cada lista armazenada no predicado *list/1*, chama o predicado *list_with_error/2* (definido na pergunta 1) e imprime a lista original e a lista com erro. O procedimento não faz validações. Mesmo que não tenhas definido o predicado *list_with_errors/2*, usa-o como se ele já existisse.

Exemplo:

```
?- print_lists_with_errors.
[a, b, c, d]      error: [d, b, c, a]
[1, 2, 3, 4]      error: [1, 4, 3, 2]
[n, l, k, r]      error: [n, k, l, r]
true
```

R:

```
print_lists_with_errors :-
     list(L1),
     list_with_error(L1, L2),
     write(L1), tab(4), write('error: '), write(L2), nl,
     fail.
print_lists_with_errors.
```