Capítulo 7

Prolog

7.17 O Prolog como linguagem de programação

Na resolução dos exercícios desta aula não pode utilizar funcionais sobre listas, excepto no último exercício.

7.17.1. Implemente o predicado `substitui_f/3`, tal que `substitui_f(T_c, Novo_F, Novo_T_c)`, em que `T_c` é um termo composto e `Novo_f` é um átomo, significa que `Novo_T_c` é um termo composto de functor `Novo_f` e com os mesmos argumentos que `T_c`. Por exemplo,

```
?- substitui_f(f(X, a), g, Novo_T_c).
Novo_T_c = g(X, a).
```

7.17.2. Implemente o predicado `substitui_arg/4`, tal que `substitui_arg(T_c, Arg, Novo_Arg, Novo_T_c)`, em que `T_c` é um termo composto, significa que `Novo_T_c` é um termo composto obtido de `T_c` substituindo os argumentos que sejam iguais a `Arg` por `Novo_Arg`. Por exemplo,

```
?- substitui_arg(f([a,b,c],9,9), [a,b,c], abc, Novo_T_c).
Novo_T_c = f(abc, 9, 9).

?- substitui_arg(f([a,b,c],9,9), 9, 99, Novo_T_c).
Novo_T_c = f([a, b, c], 99, 99).
```

```
?- substitui_arg(f([a,b,c],9,9), 1, 99, Novo_T_c).
Novo_T_c = f([a, b, c], 9, 9).
```

Sugestão: Defina o predicado `substitui_el_lst(Lst1, El, Novo_El, Lst2)` que substitui todas as ocorrências de `El` em `Lst1` por `Novo_El`, para obter `Lst2`.

7.17.3. Implemente o predicado `todos/2`, tal que `todos(Pred, Lst)`, em que `Pred` é o nome de um predicado unário e `Lst` é uma lista, significa que todos os elementos de `Lst` satisfazem `Pred`. Por exemplo, supondo definido o predicado `par(X)`, significando que `X` é um inteiro par,

```
?- todos(par, [1,2,3]).
false.

?- todos(par, [4,2,32]).
true.

?- todos(par, []).
true.
```

7.17.4. Implemente o predicado `algum/2`, tal que `algum(Pred, Lst)`, em que `Pred` é o nome de um predicado unário e `Lst` é uma lista, significa que existe um elemento de `Lst` que satisfaz `Pred`. Por exemplo, supondo definido o predicado `par(X)`, significando que `X` é um inteiro par,

```
?- algum(par, []).
false.

?- algum(par, [1,3,5]).
false.

?- algum(par, [1,4,5]).
true.
```

7.17.5. Implemente o predicado `quantos/3`, tal que `quantos(Pred, Lst, N)`, em que `Pred` é o nome de um predicado unário e `Lst` é uma lista, significa que existem `N` elementos de `Lst` que satisfazem `Pred`. Por exemplo, supondo definido o predicado `par(X)`, significando que `X` é um inteiro par,

```
?- quantos(par, [1,4,5], N).
N = 1.

?- quantos(par, [], N).
N = 0.

?- quantos(par, [2,4,6], N).
N = 3.
```

7.17.6. Implemente o predicado `transforma/3`, tal que `transforma(Tr, Lst1, Lst2)`, em que `Tr` é o nome de um predicado binário e `Lst1` é uma lista, significa que `Lst2` é o resultado de "aplicar" `Tr` a cada elemento de `Lst1`. Por exemplo, supondo definido o predicado `soma_1(X, Y)`, significando que `Y` é o resultado de somar um a `X`,

```
?- transforma(soma_1, [2,4,6], Lst2).
Lst2 = [3, 5, 7].
```

7.17.7. Implemente o predicado `filtra_inc/3`, tal que `filtra_inc(Lst1, Tst, Lst2)`, em que `Tst` é o nome de um predicado unário e `Lst1` é uma lista, significa que `Lst2` é a lista formada pelos elementos de `Lst1` que satisfazem `Tst`. Por exemplo, supondo definido o predicado `par(X)`, significando que `X` é um inteiro par,

```
?- filtra_inc([1,2,3,4,5,6], par, Lst2).
Lst2 = [2, 4, 6].
```

7.17.8. Implemente o predicado `filtra_exc/3`, tal que `filtra_exc(Lst1, Tst, Lst2)`, em que `Tst` é o nome de um predicado unário e `Lst1` é uma lista, significa que `Lst2` é a lista formada pelos elementos de `Lst1` que não satisfazem `Tst`. Por exemplo, supondo definido o predicado `par(X)`, significando que `X` é um inteiro par,

```
?- filtra_exc([1,2,3,4,5,6], par, Lst2).
Lst2 = [1, 3, 5].
```

7.17.9. Implemente o predicado `acumula/3`, tal que `acumula(Lst, Op, Res)`, em que `Lst` é uma lista não vazia e `Op` é o nome de um predicado ternário aplicável aos elementos de `Lst`, significa que `Res` é o resultado de "aplicar" sucessivamente `Op` aos elementos de `Lst`. Por exemplo, supondo definidos os predicados `mais(X, Y, Res)` e `menos(X, Y, Res)`, significando, respectivamente, que `Res` é a soma e a diferença de `X` e `Y`,

```
?- acumula([1,2,3], mais, Res).
Res = 6.

?- acumula([6,1,2,3], menos, Res).
Res = 0.

?- acumula([[1,2], [4,5], [6,7]], append, Res).
Res = [1, 2, 4, 5, 6, 7].

?- acumula([], mais, Res).
false.
```

7.17.10. Implemente o predicado `executa_lst/1`, tal que `executa_lst(Lst)`, em que `Lst` é uma lista de objectivos, tem como efeito executar sucessivamente os elementos de `Lst`. Por exemplo, supondo definidos os predicados `par` e `filtra_inc`, descritos anteriormente,

```
?- executa_lst([L = [1,2,3,4,5,6], filtra_inc(L, par, Pares),
                write('**'), write(Pares), write('**')]).
**[2,4,6]**
L = [1, 2, 3, 4, 5, 6],
Pares = [2, 4, 6].
```