Dados compostos

Marco A L Barbosa malbarbo.pro.br

Prefira usar restrições sobre inteiros ao invés das operações aritméticas tradicionais do Prolog. Veja na documentação da biblioteca `clpfd` os tipos de expressões que podem ser usadas nas restrições.

Tente projetar os predicados mais genéricos possíveis, isto é, só restrinja algum argumento para instanciado se o predicado não fizer sentido de outra forma.

Você pode usar os predicados pré-definidos de lista: `length/2`, `member/2`, `append/3`, `nth0/3`.

1) [pp99 1.01] Projete um predicado `ultimo(L, X)` que é verdadeiro se `X` é o último elemento da lista `L`.

2) [p99 1.14] Projete um predicado `duplicada(L, D)` que é verdadeiro se `D` tem os elementos de `L` duplicados. Exemplo

   ```
   ?- duplicada([a, b, c, c, d], D).
   D = [a, a, b, b, c, c, c, c, d, d].
   ```

3) [lpn 6.1] Uma lista é dobrada se ele é constituída de dois blocos consecutivos de elementos iguais. Projete um predicado `dobrada(L)` que é verdadeiro se `L` é uma lista dobrada.

   ```
   ?- dobrada([a, b, c, a, b, c]).
   true.
   ?- dobrada([a, b, a]).
   false.
   ```

4) Projete um predicado `soma_zero(L, A, B)` que é verdadeiro `A` e `B` são elementos distintos da lista `L` e a soma de `A` e `B` é zero.

5) [pp99 1.06] Projete um predicado `palindromo(L)` que é verdadeiro se a lista `L` é palíndromo. Dica: projete um predicado auxiliar `invertida(A, B)` que é verdadeiro se `A` é a lista `B` invertida.

6) Projete um predicado `mergesort(A, S)` que é verdeiro `S` é `A` ordenada. Implemente a ordenação usando o algoritmo de ordenação mergesort.

   ```
   ?- mergesort([7, 3, 6, 1, 2, 5, 4], S).
   S = [1, 2, 3, 4, 5, 6, 7].
   ```

7) Projete um predicado `meio(L, X)` que é verdadeiro se `X` é o elemento do meio da lista `L`.

8) Projete um predicado `maximo(XS, M)` que é verdadeiro se `M` é o valor máximo da lista `XS`.

9) Projete um predicado `pares(L, P)` que é verdadeiro se `P` é uma lista com os números pares de `L` (na mesma ordem).

10) Projete um predicado `sequencia(L, N)` que é verdadeiro se `L` é uma lista com os primeiros `N` números naturais em sequência.

11) Projete um predicado `lista_soma(XS, A, YS)` que é verdadeiro se a lista `YS` é a lista `XS + A` (cada elemento de `XS` somado com `A`). Exemplo

    ```
    ?- lista_soma([1, 4, 23], -3, YS).
    YS = [-2, 1, 20].
    ```

12) [p99 1.20] Projete um predicado `removido_em(L, X, I, R)` que é verdadeiro se `R` é a lista `L` com o elemento `X` removido da posição `I`.

```
?- removido_em([a, b, c, d], X, 2, R).
X = c,
R = [a, b, d].
```

13) [p99 1.21] Projete um predicado `inserido_em(L, X, I, R)` que é verdadeiro se `R` é a lista `L` com o elemento `X` inserido da posição `I`.

```
?- inserido_em([a, b, d], c, 2, R).
R = [a, b, c, d].
```

14) [p99 1.18] Projete um predicado `sub_lista(L, I, J, S)` que é verdadeiro se `S` é uma sub lista de `L` com os elementos das posições de `I` a `J` (inclusive). Exemplo

```
?- sub_lista([a, b, c, d, e, f, g, h, i, k], 3, 7, S).
S = [d, e, f, g, h].
```

15) [p99 1.19] Projete um predicado `rotacionada(L, N, R)` que é verdadeiro se `R` contém os elementos de `L` rotacionados `N` posições a esquerda. Exemplo

```
?- rotacionada([a, b, c, d, e, f, g, h], 3, R).
R = [d, e, f, g, h, a, b, c].
```

16) [p99 2.02] Projete um predicado `fatores_primo(N, F)` que é verdadeiro se `F` é uma lista com os fatores primos de `N`.

```
?- fatores_primos(315, F).
F = [3, 3, 5, 7].
```

17) Projete um predicado `arvore(T)` que é verdadeiro se `T` é uma árvore binária (de acordo com a definição das notas de aula).

18) Projete um predicado `num_folhas(T, S)` que é verdadeiro se `S` é o número de folhas da árvore binária `T`.

19) Analise os exercícios anteriores (inclusive da lista de fundamentos) e reescreva os predicados (que obtiverem algum melhora no desempenho) utilizando acumuladores.

20) Analise os exercícios anteriores e reescreva os predicados (que obtiverem algum melhora no desempenho) utilizando diferença de listas.

Referências

- [lpn]. Lear Prolog Now
- [pip]. Programming in Prolog.
- [pp99]. 99 problemas para resolver em (Prolog)