Ficha Prática 5

5.1 Objectivos

1. Praticar a escrita de predicados sobre listas.

5.2 Conceitos

5.2.1 Listas

Notação

Em Prolog, as listas representam-se entre parênteses rectos ("`[`", "`]`"), com os elementos separados por vírgulas ("`,`"). Os elementos de uma lista podem ser qualquer tipo de termo.

Alguns exemplos de listas válidas:

```
[1, 2, 3]
[um, dois, tres]
[1, dois, 3]
[ana, [maria, madalena], X]
```

A **lista vazia** é representada por "`[]`". Uma **lista não vazia** pode ser representada utilizando a notação "`[H|T]`", em que "`H`" é a cabeça da lista e "`T`" é a cauda da lista. Assim, a lista "`[1, 2, 3]`" pode também ser representada das seguintes formas:

```
[1 | [2, 3]]
[1, 2 | [3]]
[1, 2, 3 | []]
```

Exemplos

Muitos dos predicados sobre listas que iremos definir serão **predicados recursivos**. Eles serão definidos para:

- o caso base (normalmente a lista vazia "`[]`");
- o caso recursivo (normalmente, para uma lista "`[H|T]`", definir alguma condição sobre a cabeça "`H`" e depois utilizar o predicado, recursivamente, na cauda "`T`").

Por exemplo, o predicado `tamanho/2`, que define o tamanho de uma lista, pode ser escrito da seguinte forma:

```
tamanho([], 0).
tamanho([_|T], N) :- tamanho(T, N1), N is N1+1.
```

Já o predicado `elemento/2` para testar a existência de um elemento numa lista pode ser definido da seguinte forma:

```
elemento(H, [H|_]).
elemento(E, [_|T]) :- elemento(E, T).
```

Note-se que aqui o caso base não está definido sobre a lista vazia: como a lista vazia não tem elementos, tal não faria sentido.

5.3 Exercícios

5.3.1 Percorrer listas

Escreva os seguintes predicados sobre listas (note que para alguns deles já existe equivalente em Prolog — no entanto, vale a pena o exercício):

1. `ultimo/2`:

   `ultimo(E,L) :-` `E` é o último elemento de `L`.

2. `somatorio/2`:

   `somatorio(L,N) :-` `N` é o somatório dos elementos presentes em `L`.

3. `adjacentes/3`:

   `adjacentes(E1,E2,L) :-` `E1` e `E2` são elementos adjacentes na lista `L`.

4. `selecciona/3`:

   `selecciona(E,L1,L2) :-` `E` é um elemento de `L1` e `L2` é o resto da lista.

5. `nesimo/3`:

   `nesimo(L, N, X) :-` `X` é o elemento que se encontra na posição `N` de `L`.

6. `tamanho/2`:

   o predicado tamanho definido na secção 5.2.1 está definido utilizando recursividade à esquerda, redefina-o por forma a utilizar recursividade à direita.

7. `media/2`:

   `media(L,N) :-` `N` é a média dos elementos da lista `L`.

8. `conta/3`:

   `conta(E,L,N) :-` `E` existe `N` vezes em `L`.

9. `maximo/2`:

   `maximo(L,N) :-` `N` é o maior elemento da lista `L`.

10. `posmax/2`:

    `posmax(L,N) :-` `N` é a posição do maior elemento da lista `L`.

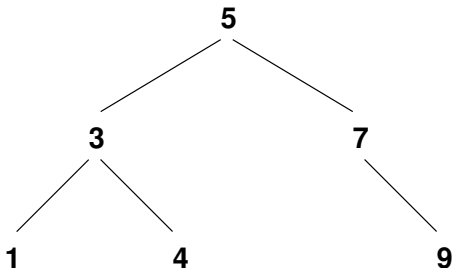

Figura 5.6: Árvore

5.3.2 Problemas com Listas

1. Considere a árvore binária apresentada na figura 5.6. Utilizando termos da forma "`arvore(sub-arv-esq, nodo, sub-arv-dir)`" para representar a árvore (utilize o átomo `nil` para indicar a inexistência de sub-árvores), escreva os seguintes predicados:

   (a) `procura/2`:

   `procura(Arv, Val) :- Val` existe na árvore `Arv`.

   (b) `to_list/2`:

   `to_list(Arv, L) :- L` é a representação sob forma de lista da árvore `Arv`.

2. Considere a figura 5.7. Escreva um predicado que lhe permita obter todos os possíveis

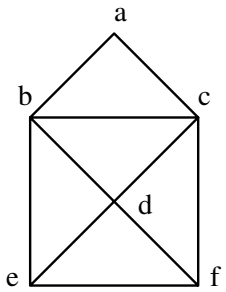

Figura 5.7: Casa

modos de desenhar a figura utilizando uma linha contínua.