Conteúdo

1 Corte de Alternativas **1**

2 Ordenação **1**

3 Árvores **2**

4 Negação por falha **2**

5 Entrada e Saída **3**

6 Estruturas de Diferenças **3**

1 Corte de Alternativas

Corte de alternativas (!)

Suponhamos que o predicado H tem duas cláusulas:

$$H_1 : -B_1, B_2, \ldots, B_i, !, B_j, \ldots, B_k$$
$$H_2 : -B_m, \ldots, B_n.$$

Suponhamos que temos o objectivo H. Os objectivos entre B_1 e B_i, podem retroceder se falharem e se B_1 falhar a cláusula H_2 é considerada. Mas se B_i suceder, nenhuma outra escolha antes de ! é considerada.

Os objectivos B_j e B_k podem retroceder entre eles mas se B_j falhar o objectivo inicial H falha.

Exercícios:

Define os predicados seguintes:

- `membercheck/2` sem alternativas caso exista um elemento
- `max_int/3` (sem e com !)
- Verificar as chamadas em `drink.pl`.

2 Ordenação

Ordenação

Define os predicados seguintes:

- `slow_sort/2`
- `insert_sort/2`
- `quick_sort/2`

3 Árvores

Árvores

Define os predicados seguintes:

- `bintree/1`
- `inTree/2`
- Travesias duma árvore: pre, in e post ordem `preorder/2, inorder/2, postorder/2`
- inserir um valor num árvore ordenada `insOrdTree/3`
- `maxTree/2`

4 Negação por falha

Negação por falha

`fail` é um predicado que falha sempre. Pode ser usado com o corte !.

O João só não gosta de carne.

```
gosta(joao,X):- carne(X),!, fail.
gosta(joao,X):-comida(X).
```

`not/1`:

- `not(G)` falha se G sucede
- `not(G)` sucede se G não sucede.

NOTA: Não é a negação lógica porque é baseada no sucesso ou não.

Negação por falha

Pode ser assim implementado:

```
not(G):- call(G), !, fail.
not(_).

gosta(joao,X):-not(carne(X)).
```

- `different/2`
- `notmember/2`

NOTA: deve-se usar `not()` com argumentos fechados.

5 Entrada e Saída

Entrada e Saída

- `read/1`: lê um termo e unifica com o argumento. Tem de terminar com '.'
- `write/1`: escreve o termo
- `writeln/1`: escreve o termo e muda de linha (`nl.`).

Sequência de Caracteres:

- `'Ola'` caracteres
- `''Ola''` códigos ASCII dos caracteres

`name/2` converte constantes numa lista de codigos de caracteres.
Baixo nível: `get/1`,`get0/1`,`put/1`

6 Estruturas de Diferenças

Diferença de Listas

Uma lista L pode ser representada como um par $inicio - fim$:`L=L1-L2` onde $L1 = [R|X]$ e $L2 = X$, i.e $L = [R|X] - X$.

```
app(A-B,B-C,A-C).

?- app([1,2,3|X]-X, [4,5,6|Y]-Y,A-Z).
```

- `X-X` lista vazia
- `[R|X]-X` lista que contem os elementos em R.
- Unificando `[R|X]-X` com `Y-[]` corresponde à lista $[R]$.

Podemos considerar listas normais mas associar argumentos extra nos predicados:

`app(A,B,B,C,A,C).`

Diferenças

- `flatten/2` (Ex. $flatten([[1,2,3],[4,[5,6]]],[1,2,3,4,5,6])$
- Transformar uma árvore binária numa com todos os nós igual ao maximo `maxTree/2`
  - acumular o máximo parcial
  - ter uma variavel que vai ser colocada em todos os nós e unificará com o máximo.)

```
maxTree(A,B):-mt(A,B,M,0,M).

mt([],[],_,M,M).
mt(n(N,L,R),n(H,Ls,Rs),H,MX,T):- N<MX, mt(L,Ls,H,MX,MXX),
         mt(R,Rs,H,MXX,T),
mt(n(N,L,R),n(H,Ls,Rs),H,MX,T):- N>=MX, mt(L,Ls,H,N,MXX),
 mt(R,Rs,H,MXX,T),
```