Aplicações de Lógica

Lógica como Linguagem de Programação

Mario Benevides
UFRJ

Listas

Lista L=[a,b,c,d,e,f]

- onde a,b,...,f são elementos da lista.

- o elemento a é chamado de **cabeça** e a lista restante [b,c,d,e,f] é chamada de **cauda.**

Notação: l(X,Y) é uma lista onde X é a cabeça e Y é a cauda.

- Tipo de dados tipicamente recursivo: uma lista é um elemento concatenado com uma lista.

Definição: O conjunto de termos denotando listas é definido recursivamente da seguinte forma:

1. nil é um termo denotando a lista de comprimento zero chamada de lista vazia;

2. Se X é um termo e Y é uma lista de comprimento n-1, então l(X,Y) é um termo Prolog denotando uma lista de comprimento n. O termo X é chamado de cabeça e o Y de cauda da lista l(X,Y).

Exemplos:

Nós representamos a lista [a,b,c,d] como l(a,l(b,l(c,l(d,nil)))).

- Vamos definir agora algumas operações sobre lista:

Programa 1: Predicado lista(X) que verifica se X é uma lista;

lista(nil)
lista(Y) $\rightarrow$ lista(l(X,Y))

Exemplo:

Pergunta: lista(l(a,l(b,l(c,nil)))) ?

$$
\begin{array}{c}
\text{lista(l(a,l(b,l(c,nil))))} \\
\downarrow \\
\text{lista(l(b,l(c,nil)))} \\
\downarrow \\
\text{lista(l(c,nil))} \\
\downarrow \\
\text{lista(nil)}
\end{array}
$$

- Outra operação bastante útil é testar quando um dado elemento pertence a uma certa lista.

programa 2: predicado membro(X,Y): o elemento X é membro da lista Y.

membro(X,l(X,Z))
membro(X,Z) → membro(X,l(Y,Z))

- A primeira cláusula testa se X é o cabeça da lista Y e a segunda testa se X está na cauda.

Convensão:	variáveis para elementos:	X,Y,Z,...
	variáveis para listas:	Xs,Ys,Zs,....

- Próxima operação é para testar se uma lista é sublista de outra.

- Exemplo: L=[a,b,c,d] uma lista, [b,c] é uma sublista de L, mas [a,c] não, pois para ser sublista os elementos tem de ser consecutivos em L.

- Exercício: membro(a,[[b,c],d,[a,f]])   membro([a,f],[[b,c],d,[a,f]])

- Vamos primeiro definir dois tipos particulares de sublistas.

programa 3: predicado prefixo(Xs,Ys)

prefixo(nil,Ys)
prefixo(Xs,Ys) → prefixo(l(X,Xs),l(X,Ys))

- Exercício: prefixo([a,b],[a,b,c,d]) é verdade.

programa 4: predicado sufixo(Xs,Ys)

sufixo(Xs,Xs)
sufixo(Xs,Ys) → sufixo(Xs,l(Y,Ys))

- Exercício: sufixo([c,d].[a,b,c,d]) é verdade.

- Existem várias maneiras de se implementar a operação de sublista, a seguir apresentaremos 3:

programa 5: predicado sublista(Xs,Ys)

a. prefixo(Ps,Ys) , sufixo(Xs,Ps) → sublista(Xs,Ys)
ou
b. prefixo(Xs,Ss) , sufixo(Ss,Ys) → sublista(Xs,Ys)
ou
c. prefixo(Xs,Ys) → sublista(Xs,Ys)
prefixo(Xs,Ys) → sublista(Xs,l(Y,Ys))

- O predicado membro pode ser visto como um caso especial de sublista:

sublista(l(X,nil),Xs) → membro(X,Xs)

- Uma outra operação muito útil é concatenar duas listas resultando numa terceira. Por exemplo concatenar [a,b] com [c,d] resultando [a,b,c,d].

programa 6: predicado concat(Xs,Ys,Zs), concatena lista Xs com lista Ys resultando na lista Zs.

concat(nil,Ys,Ys)
concat(Xs,Ys,Zs) → concat(l(X,Xs),Ys,l(X,Zs))

- Pergunta: concat(l(a,l(b,nil)),l(c,l(d,nil)), l(a,l(b,l(c,l(d,nil))))) ?

concat(l(a,l(b,nil)),l(c,l(d,nil)), l(a,l(b,l(c,l(d,nil)))))
↓
concat(l(b,nil),l(c,l(d,nil)), l(b,l(c,l(d,nil))))
↓
concat(nil,l(c,l(d,nil)), l(c,l(d,nil)))

Exercício:
1. definir prefixo, sufixo e membro em função de concat.
2. adjacente(X,Y,Zs) - se elemento X é adjacente a Y na lista Zs.
3. ultimo(X,Ys) - se X é o último elemento da lista Ys.

1. concat(Xs,Zs,Ys) → prefixo(Xs,Ys)
   concat(Zs,Xs,Ys) → sufixo(Xs,Ys)
   concat(Zs,l(X,Xs),Ys) → membro(Xs,Ys)
2. concat(Ws,l(X,l(Y,Ys)), Zs) → adjacente(X,Y,Zs)

3. concat(Ys,l(X,nil),Xs) → ultimo(X,Xs)

programa 7: predicado inverso(Xs,Ys) - a lista Ys é o inverso da lista Xs.

inverso(nil,nil)
inverso(Xs,Ys) , concat(Ys,l(X,nil),Zs) → inverso(l(X,Xs),Zs)

Exercício: fazer a árvore de derivação para provar inverso([a,b,c,d],[d,c,b,a]) .

programa 8: deleta(L1,X,L2) - lista L2 é obtida da lista L1 deletando-se todas as ocorrências de X em L1.

deleta(Xs,X,Ys) $\rightarrow$ deleta(l(X,Xs),X,Ys)
X$\neq$Z , deleta(Xs,Z,Ys) $\rightarrow$ deleta(l(X,Xs),Z,l(X,Ys))
deleta(nil,X,nil)

Exercício:
1. deleta([a,b,c,b],b,X)?
2. Como fazer para deletar apenas uma ocorrência?

Árvores Binárias

- Vamos representar usando uma função arv(elemento, esquerda, direita).
- Árvore vazia é nil.

```
 a
 /\
b c
```

- arv(a,arv(b,nil,nil),arv(c,nil,nil)).

programa 1: predicado arvbin(X) - testa se X é uma árvore binária.

arvbin(nil).
avrbin(E) , arvbin(D) → arvbin(arv(X,E,D))

programa 2: busca(elemento,arvore)

busca(X,arv(X,E,D)).
busca(X,E) → busca(X,arv(Y,E,D))
busca(X,D) → busca(X,arv(Y,E,D))

programa 3: pre(A,Xs), in(A,Xs) e pos(A,Xs) percorre a árvore A em ordem pré-ordem in-ordem e em pos-ordem respectivamente.

pre(E,Es) , pre(D,Ds) , concat(l(X,Es),Ds,Xs) → pre(arv(X,E,D),Xs)
pre(nil,nil).

in(E,Es) , in(D,Ds) , concat(Es,l(X,Ds),Xs) → in(arv(X,E,D),Xs)
in(nil,nil).

pre(E,Es) , pre(D,Ds) , concat(Ds,l(X,nil), Ds1) ,
concat(Es,Ds1,Xs) → pos(arv(X,E,D),Xs)
pos(nil,nil).

Grafos

Podemos representar os vétices de um um grafo dirigido como constantes e as aresta como um predicado.

$$\begin{array}{cccccc} a & \to & b & & & f \\ \downarrow & & \downarrow & & & \downarrow \\ c & \to & d & \to & e & g \end{array}$$

Aresta(a,b) ←	Aresta(a,c) ←
Aresta(c,d) ←	Aresta(b,d) ←
Aresta(d,e) ←	Aresta(f,g) ←

Definir um predicado Caminho(X,Y) se existe um caminho de X para Y.

1. Caminho(X,X)
2. Aresta(X,Z) , Caminho (Z,Y) → Caminho(X,Y)

Uma definição recursiva sempre tem uma regra que chama a si própria para casos cada vez mais simples e sempre tem uma outra que é o caso base,i.e., o caso mais simples.

- Exercício: Seguir a execução do programa do grafo para a pergunta:

← Conectado(a,e)