PROLOG. Lists in PROLOG. Operations and Predicates. Lists as Sequences, Sets, Bags. Meta Predicates.

Antoni Ligęza

Katedra Automatyki, AGH w Krakowie

2011

[1] Ulf Nilsson, Jan Maluszyński: Logic, Programming and Prolog, John Wiley & Sons Ltd., pdf, http://www.ida.liu.se/ ulfni/lpp

[2] Dennis Merritt: Adventure in Prolog, Amzi, 2004 http://www.amzi.com/AdventureInProlog

[3] Quick Prolog: http://www.dai.ed.ac.uk/groups/ssp/bookpages/quickprolog/quickprolog.html

[4] W. F. Clocksin, C. S. Mellish: Prolog. Programowanie. Helion, 2003

[5] SWI-Prolog's home: http://www.swi-prolog.org

[6] Learn Prolog Now!: http://www.learnprolognow.org

[7] http://home.agh.edu.pl/ ligeza/wiki/prolog

[8] http://www.im.pwr.wroc.pl/ przemko/prolog

Lists - basic concepts

- Lists are one of the most important structures in symbolic languages.
- In most of the implementations of PROLOG lists are standard, build-in structures and there are numerous operations on them provided as routine predicates.
- Lists can be used to represent
  1. sets,
  2. sequences,
  3. multi-sets (bags), and
  4. more complex structures, such as trees, records, nested lists, etc.

Lists - basic notation

A list in PROLOG is a structure of the form

$$[t_1, t_2, \ldots, t_n]$$

The order of elements of a list is important; the direct access is only to the first element called the Head, while the rest forms the list called the Tail.

$$[Head|Tail]$$

where Head is a single element, while Tail is a list.

Lists as Terms

Lists in fact are also terms. A list:

$$[t_1, t_2, \ldots, t_n]$$

is equivalent to a term defined as follows:

$$l(t_1, l(t_2, \ldots l(t_n, nil) \ldots))$$

$l/2$ is the list constructor symbol and nil is symbolic denotation of empty list.

Lists: Head and Tail

In practical programming it is convenient to use the bracket notation. In order to distinguish the head and the tail of a list the following notation is used

$$[H|T].$$

An example of list matching

```
[H|T] = [a,b,c,d,e]
H=a, T = [b,c,d,e]
```

List properties

- A list can have as many elements as necessary.
- A list can be empty; an empty list is denoted as [ ].
- A list can have arguments being of:
  1. mixed types,
  2. complex structures, i.e. terms, lists, etc., and as a consequence
  3. a list can have nested lists (to an arbitrary depth)
- a list of k elements can be matched directly against these elements, i.e.

```
[X,Y,Z,U,V] = [a,b,c,d,e]
X=a, Y=b, Z=c, U=d, V=e
```

- first k elements of any list can be matched directly

```
[X,Y,Z|T] = [a,b,c,d,e]
X=a, Y=b, Z=c, T=[d,e]
```

Single-element list

A single-element list is different from its content-element!

$$foo \neq [foo]$$

First k-elements: $k = 1, 2, 3$

```
[X|_] = [a,b,c,d,e].
X=a

[_,X|_] = [a,b,c,d,e].
X=b

[_,_,X|_] = [a,b,c,d,e].
X=c
```

Take the n-th element

```
take(1,[H|_],H):- !.
take(N,[_|T],X):- N1 is N-1, take(N1,T,X).
```

Propagation of substitutions

```
[X,Y,Z,U] = [a,b,c,d] ?
[X,Y,Z,X] = [a,b,c,d] ?
[X,Y,Y,X] = [a,U,Q,U] ?
```

Applications of Lists: Examples

List understanding: three basic possibilities

- as **sequences**,
- as **sets**,
- as **sets with repeated elements**,

When thinking of lists as sets, the order of elements is (read: must be made) unimportant.

Lists as sets

```
[a,b,c,d,e]
[1,2,3,4,5,6,7,8,9]
[1,a,2,b,f(a),g(b,c)]
```

Lists as multi-sets (bags, collections) or sequences

```
[a,b,c,d,e,a,c,e]
[1,1,2,3,4,5,6,7,8,9,2,7,1]
[1,a,2,b,f(a),g(b,c),b,1,f(a)]
```

Repeated elements can occur.

Member/2 and Select/3 Predicates

Member/2

Checking if an item occurs within a list; deterministic version.

```
member(Element,[Element|_):- !.
member(Element,[_|Tail]):-
      member(Element,Tail).
```

Member/2

Checking if an item occurs within a list; indeterministic version.

```
member(Element,[Element|_).
member(Element,[_|Tail]):-
      member(Element,Tail).
```

Select/3

Selecting and item from a list — indeterministic.

```
select(Element,[Element|Tail],Tail).
select(Element,[Head|Tail],[Head|TaiE]):-
      select(Element,Tail,TaiE).
```

Append/3

The basic use of the append/3 predicate is to concatenate two lists.

```
append([],L,L).
append([H|T],L,[H|TL]) :- append(T,L,TL).
```

Concatenation Test

```
append([a,b],[c,d,e],[a,b,c,d,e]).
```

Finding Front List

```
append(FL,[c,d,e],[a,b,c,d,e]).
FL = [a,b]
```

Finding Back List

```
append([a,b],BL,[a,b,c,d,e]).
BL = [c,d,e]
```

Indeterministic List Decomposition

```
append(FL,BL,[a,b,c,d,e])

FL = [],
BL = [a,b,c,d,e];

FL = [a],
BL = [b,c,d,e];

FL = [a,b],
BL = [c,d,e];

FL = [a,b,c],
BL = [d,e];

FL = [a,b,c,d],
BL = [e];

FL = [a,b,c,d,e],
BL = [];
false.
```

Basic Recurrent Operations: length, sum, writing a list

Length of a list

```
len([],0).
len([_|T],L):-
   len(T,LT),
   L is LT+1.
```

Sum of a list

```
sum([],0).
sum([H|T],S):-
   sum(T,ST),
   S is ST+H.
```

Write a list

```
writelist([]):- nl.
writelist([H|T]):-
       write(H),nl,
       writelist(T).
```

Put X as the first element to L

```
XL = [X|L].
```

Put X as the *k*-th element to L

```
putk(X,1,L,[X|L]):- !.
putk(X,K,[F|L],[F|LX]):- K1 is K-1, putk(X,K1,L,LX).
```

Delete one X from L (indeterministic!)

```
del(X,[X|L],L).
del(X,[Y|L],[Y|L1]):-
        del(X,L,L1).
```

Delete all X from L

```
delall(_,[],[]):- !.
delall(X,[H|L],[H|LL]):- X \= H,!, delall(X,L,LL).
delall(X,[X|L],LL):- delall(X,L,LL).
```

Lists and sublists. Nested Lists. Flatten List

A list and a sublist

[1,2,3,4,5,6,7,8,9]
[3,4,5,6]

Checking for a sublist

```
sublist(S,FSL,F,L):- append(F,SL,FSL),append(S,L,SL).
```

A list and a subsequence

[1,2,3,4,5,6,7,8,9]
[3,5,8]

Checking for subsequence

```
subseq([],_):- !.
subseq([H|S],L):- append(_,[H|SL],L),!, subseq(S,SL).
```

Nested lists. Flatten a list

[1,[2,3],4,[5,[6,7],8],9] $\longrightarrow$ [1,2,3,4,5,6,7,8,9]

Think!

1. N $\longrightarrow$ [1,2,3,...,N-1,N],
2. List: [1,2,3,4,5,6,7] $\longrightarrow$ all permutations,
3. K, [1,2,3,4,5,6,7] $\longrightarrow$ K-element comobinations,
4. Set: [1,2,3,4,5,6,7] $\longrightarrow$ all subsets,
5. ExchangeFL: [1,2,3,4,5,6,7] $\longrightarrow$ [7,2,3,4,5,6,1],
6. ShiftLCircular: [1,2,3,4,5,6,7] $\longrightarrow$ [2,3,4,5,6,7,1],
7. ShiftRCircular: [1,2,3,4,5,6,7] $\longrightarrow$ [7,1,2,3,4,5,6,7],
8. Split: [1,2,3,4,5,6,7] $\longrightarrow$ [1,3,5,7], [2,4,6],
9. Merge: [1,3,5,7], [2,4,6] $\longrightarrow$ [1,2,3,4,5,6,7],
10. Split C=4: [1,2,3,4,5,6,7] $\longrightarrow$ [1,2,3],[4],[5,6,7],
11. p1. p2. ... pK. $\longrightarrow$ [p1,p2,...,pK].

Think!

- Recursion $\longrightarrow$ Iterations,
- Recursion $\longrightarrow$ repeat-fail.

Inserting List Element. Permutations.

Insert (indeterministic!). Permutations: insert

```
insert(X,L,LX):-  del(X,LX,L).

perm([],[]).
perm([H|T],P):-
   perm(T,T1),
   insert(H,T1,P).
```

Sorted List Definition

```
sorted([]):- !. sorted([_]):- !.
sorted([X,Y|T]) :- X =< Y, sorted([Y|T]).
```

Slow Sort

```
slowsort(L,S):-
   perm(L,S),
   sorted(S).
```

Reverse List. Inverse List

Naive List Reverse

```
reverse([],[]).
reverse([X|L],R):-
        reverse(L,RL),
        append(RL,[X],R).
```

Iterative List Inverting: Accumulator

```
inverse(L,R):-
   do([],L,R).
do(L,[],L):-!.
do(L,[X|T],S):-
   do([X|L],T,S).
```

Accumulator

[a,b,c], [d,e,f,g] $\longrightarrow$ [d,c,b,a], [e,f,g]

Set Algebra Operations

```
subset([],_).
subset([X|L],Set):-
        member(X,Set),
        subset(L,Set).
intersect([],_,[]).
intersect([X|L],Set,[X|Z]):-
        member(X,Set),!,
        intersect(L,Set,Z).
intersect([X|L],Set,Z):-
        not(member(X,Set)),
        intersect(L,Set,Z).
union([],Set,Set).
union([X|L],Set,Z):-
        member(X,Set),!,
        union(L,Set,Z).
union([X|L],Set,[X|Z]):-
        not(member(X,Set)),!,
        union(L,Set,Z).
difference([],_,[]).
difference([X|L],Set,[X|Z]):-
        not(member(X,Set)),!,
        difference(L,Set,Z).
difference([_|L],Set,Z):- difference(L,Set,Z).
```