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

Antoni Ligęza

Katedra Automatyki, AGH w Krakowie

2011


Introduction to Lists in 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.
Definition of Lists. Lists as Terms

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:

\[ l(t_1, l(t_2, \ldots l(t_n, \text{nil}) \ldots)) \]

\( l/2 \) is the list constructor symbol and \( \text{nil} \) 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

1. \( [H|T] = [a, b, c, d, e] \)
2. \( H=a, \ T = [b, c, d, e] \)
Some Notes on lists. Unification Variants

**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.
  1. \([X, Y, Z, U, V] = [a, b, c, d, e]\)
  2. \(X=a, \ Y=b, \ Z=c, \ U=d, \ V=e\)

- First \( k \) elements of any list can be matched directly
  1. \([X, Y, Z|T] = [a, b, c, d, e]\)
  2. \(X=a, \ Y=b, \ Z=c, \ T=[d,e]\)

**Single-element list**

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

\(foo \neq [foo]\)
First $k$ elements. The $n$-th element. Propagation of Substitutions

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

1. $[X|\_] = [a,b,c,d,e]$.
2. $X=a$
3. $[\_, X|\_] = [a,b,c,d,e]$.
4. $X=b$
5. $[\_, \_, X|\_] = [a,b,c,d,e]$.
6. $X=c$

Take the $n$-th element

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

Propagation of substitutions

1. $[X, Y, Z, U] = [a, b, c, d]$ ?
2. $[X, Y, Z, X] = [a, b, c, d]$ ?
3. $[X, Y, Y, X] = [a, U, Q, U]$ ?
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

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

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

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

Repeated elements can occur.
Member/2

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

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

Member/2

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

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

Select/3

Selecting an item from a list — **indeterministic**.

1. `select(Element,[Element|Tail],Tail).`
2. `select(Element,[Head|Tail],[Head|TaiE]) :- select(Element,Tail,TaiE).`
Lists as Sequences: the Beauty of the Append/3 Predicate

**Append/3**

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

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

**Concatenation Test**

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

**Finding Front List**

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

**Finding Back List**

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

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

**Sum of a list**

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

**Write a list**

1. `writelist([]):- nl.`
2. `writelist([H|T]):-
   write(H), nl,
   writelist(T).`
Putting and Deleting Elements to/form a List

Put X as the first element to L

1  \(XL = [X|L].\)

Put X as the \(k\)-th element to L

1  \(\text{putk}(X,1,L,[X|L]):- !.\)
2  \(\text{putk}(X,K,[F|L],[F|LX]):- K1 \text{ is } K-1, \text{ putk}(X,K1,L,LX).\)

Delete one X from L (indeterministic!)

1  \(\text{del}(X,[X|L],L).\)
2  \(\text{del}(X,[Y|L],[Y|L1]):-\)
3  \(\text{del}(X,L,L1).\)

Delete all X from L

1  \(\text{delall}(\_,[],[]):- !.\)
2  \(\text{delall}(X,[H|L],[H|LL]):- X \ \text{\_}\_ H, !, \text{ delall}(X,L,LL).\)
3  \(\text{delall}(X,[X|L],LL):- \text{ delall}(X,L,LL).\)
### 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


#### A list and a subsequence

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

#### Checking for subsequence

1  \texttt{subseq}([],_):= !.
2  \texttt{subseq}([H|S],L):= \texttt{append}(_,[H|SL],L),!, \texttt{subseq}(S,SL).

#### Nested lists. Flatten a list

\[1,[2,3],4,[5,[6,7],8],9\] \rightarrow \[1,2,3,4,5,6,7,8,9\]
Lists: some small challenges

Think!

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

Think!

\[\times\] Recursion \(\rightarrow\) Iterations,
\[\times\] Recursion \(\rightarrow\) 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

```prolog
1 reverse([],[]).
2 reverse([X|L],R):-
3     reverse(L,RL),
4     append(RL,[X],R).
```

Iterative List Inverting: Accumulator

```prolog
1 inverse(L,R):-
2     do([],L,R).
3     do(L, [],L):-!.
4     do(L, [X|T],S):-
5     do([X|L],T,S).
```

Accumulator

\[ [a, b, c], [d, e, f, g] \rightarrow [d, c, b, a], [e, f, g] \]
Set Algebra Operations

```prolog
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).
```

Antoni Ligęza