Lists in Prolog

Figure 3.1 Tree representation of the list [ann, tennis, tom, skiing]

'.'(ann, '.'(tennis, '.'(tom, '.'(skiing, []))))
Some operations on lists

### 3.2.2 Concatenation

For concatenating lists we will define the relation:

\[
\text{conc}(L_1, L_2, L_3)
\]

Here \(L_1\) and \(L_2\) are lists, and \(L_3\) is their concatenation.

For example,

\[
\text{conc}([a,b], [c,d], [a,b,c,d])
\]

is true, but

\[
\text{conc}([a,b], [c,d], [a,b,a,c,d])
\]

is false.

In the definition of \(\text{conc}\) we will have again two cases, depending on the first argument, \(L_1\):

1. If the first argument is the empty list then the second and the third arguments must be the same list (call it \(L\)); this is expressed by the following Prolog fact:

   \[
   \text{conc}([], L, L).
   \]

2. If the first argument of \(\text{conc}\) is a non-empty list then it has a head and a tail and must look like this:

   \[
   \text{lx lLll}
   \]

   Figure 3.2 illustrates the concatenation of \([X I L_1]\) and some list \(L_2\). The result of the concatenation is the list \([X I L_3]\) where \(L_3\) is the concatenation of \(L_1\) and \(L_2\). In Prolog this is written as:

   \[
   \text{conc}(\text{X I L_1}, L_2, \text{X I L_3}) :- \text{conc}(L_1, L_2, L_3).
   \]

This program can now be used for concatenating given lists, for example:

\[
? \text{-} \text{conc}([a,\text{b}], [c,d], [a,[b,c],d], a, [b], L).
\]

\[
L : [a,b,c,d,a,b]
\]

Figure 3.2 Concatenation of lists.
Chapter 3

Lists, Operators, Arithmetic

Figure 3.4  The member and sublist relations.

Figure 3.4  The member and sublist relations.
3.2 Some operations on lists

member(X, L)
sublist(S, L)

on the same idea as memberl, sublist.

Accordingly, the relation memberl, sublist for decomposing lists. So the nents.
The arguments are two elements.

The intention is to generate permutations of a list: a given list. To this end, we nents.
The corresponding Prolog clauses that correspond to these two cases are:

permutation(tl, []).
permutation([X | L], P):-
permutation(L, L1),
insert(X, L1, P).

Figure 3.5 One way of constructing a permutation of the list [X | L].
The `is` operator

Forces evaluation.

Similar to the assignment statement.

The left argument is a simple object.
+       addition
-       subtraction
*       multiplication
/       division
**      power
//      integer division
mod     modulo, the remainder of integer division

Arithmetic

Some of the predefined operators can be used for basic arithmetic operations. These are:

- +  addition
- -  subtraction
- *  multiplication
- /  division
- ** power
- // integer division
- mod modulo, the remainder of integer division
<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X &gt; Y</td>
<td>X is greater than Y</td>
</tr>
<tr>
<td>X &lt; Y</td>
<td>X is less than Y</td>
</tr>
<tr>
<td>X &gt;= Y</td>
<td>X is greater than or equal to Y</td>
</tr>
<tr>
<td>X &lt;= Y</td>
<td>X is less than or equal to Y</td>
</tr>
<tr>
<td>X :=:= Y</td>
<td>the values of X and Y are equal</td>
</tr>
<tr>
<td>X == Y</td>
<td>the values of X and Y are not equal</td>
</tr>
</tbody>
</table>