Logisch Programmeren en Zoektechnieken

Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam

http://www.illc.uva.nl/~ulle/teaching/prolog/

Lists in Prolog

One of the most useful data structures in Prolog are *lists*. The aim of this lecture is to show you how lists are represented in Prolog and to introduce you to the basic principles of working with lists.

An example for a Prolog list:

```
[elephant, horse, donkey, dog]
```

Lists are enclosed in square brackets. Their elements could be any Prolog terms (including other lists). The *empty list* is `[]`.

Another example:

```
[a, X, [], f(X,y), 47, [a,b,c], bigger(cow,dog)]
```

Internal Representation

Internally, the list

```
[a, b, c]
```

corresponds to the term

```
.(a, .(b, .(c, [])))
```

That means, this is *just a new notation*. Internally, lists are just compound terms with the functor `.` (dot) and the special atom `[]` as an argument on the innermost level.

We can verify this also within Prolog:

```
?- X = .(a, .(b, .(c, []))).
X = [a, b, c]
Yes
```

The Bar Notation

If a bar `|` is put just before the last term in a list, this means that this last term denotes a sub-list. Inserting the elements before the bar at the beginning of the sub-list yields the entire list.

For example, `[a, b, c, d]` is the same as `[a, b | [c, d]]`.

Examples

Extract the second element from a given list:

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

Make sure the first element is a `1` and get the sub-list after the second element:

```
?- MyList = [1, 2, 3, 4, 5], MyList = [1, _ | Rest].
MyList = [1, 2, 3, 4, 5]
Rest = [3, 4, 5]
Yes
```

Head and Tail

The first element of a list is called its *head*. The rest of the list is called its *tail*. (The empty list doesn't have a head.)

A special case of the bar notation—with exactly one element before the bar—is called the *head/tail-pattern*. It can be used to extract head and/or tail from a list. Example:

```
?- [elephant, horse, tiger, dog] = [Head | Tail].
Head = elephant
Tail = [horse, tiger, dog]
Yes
```

Head and Tail (continued)

Another example:

```
?- [elephant] = [X | Y].
X = elephant
Y = []
Yes
```

Note: The tail of a list is always a list itself. The head of a list is an element of that list. The head can also be a list itself (but it usually isn't).

Appending Lists

We want to write a predicate `concat_lists/3` to concatenate (append) two given lists.

It should work like this:

```
?- concat_lists([1, 2, 3, 4], [dog, cow, tiger], L).
L = [1, 2, 3, 4, dog, cow, tiger]
Yes
```

Solution

The predicate `concat_lists/3` is implemented *recursively*. The *base case* is when one of the lists is empty. In every recursion step we take off the head and use the same predicate again, with the (shorter) tail, until we reach the base case.

```
concat_lists([], List, List).

concat_lists([Elem|List1], List2, [Elem|List3]) :-
  concat_lists(List1, List2, List3).
```

Do More

Amongst other things, `concat_lists/3` can also be used for decomposing lists:

```
?- concat_lists(Begin, End, [1, 2, 3]).
Begin = []
End = [1, 2, 3] ;
Begin = [1]
End = [2, 3] ;
Begin = [1, 2]
End = [3] ;
Begin = [1, 2, 3]
End = [] ;
No
```

Built-in Predicates for List Manipulation

`append/3`: Append two lists (same as our `concat_lists/3`).

```
?- append([1, 2, 3], List, [1, 2, 3, 4, 5]).
List = [4, 5]
Yes
```

`length/2`: Get the length of a list.

```
?- length([tiger, donkey, cow, tiger], N).
N = 4
Yes
```

Membership

`member/2`: Test for membership.

```
?- member(tiger, [dog, tiger, elephant, horse]).
Yes
```

Backtracking into `member/2`:

```
?- member(X, [dog, tiger, elephant]).
X = dog ;
X = tiger ;
X = elephant ;
No
```

Example

Consider the following program:

```
show(List) :-
  member(Element, List),
  write(Element),
  nl,
  fail.
```

Note: `fail/0` is a built-in predicate that always fails.

What happens when you submit a query like the following one?

```
?- show([elephant, horse, donkey, dog]).
```

Example (continued)

```
?- show([elephant, horse, donkey, dog]).
elephant
horse
donkey
dog
No
```

The `fail` at the end of the rule causes Prolog to backtrack. The subgoal `member(Element, List)` is the only choicepoint. In every backtracking-cycle a new element of `List` is matched with the variable `Element`. Eventually, the query fails (`No`).

More Built-in Predicates

`reverse/2`: Reverse the order of elements in a list.

```
?- reverse([1, 2, 3, 4, 5], X).
X = [5, 4, 3, 2, 1]
Yes
```

More built-in predicates can be found in the reference manual.

Summary: List Manipulation

- List notation:
  - normal: `[Elem1, Elem2, Elem3]` (empty list: `[]`)
  - internal: `.(Elem1, .(Elem2, .(Elem3, [])))`
  - bar notation: `[Elem1, Elem2 | Rest]`
  - head/tail-pattern: `[Head | Tail]`
- Many predicates can be implemented recursively, exploiting the head/tail-pattern.
- Built-in predicates: `append/3`, `member/2`, `length/2`, ...