Lists in Prolog

A list is a sequence of any number of items. Lists can be written in Prolog as:

    [ann, tennis, tom, skiing]

A list can be either empty or non-empty.

The empty list is simply written as a Prolog atom [].

A non-empty list can be viewed as consisted of two parts:

1. the first item, called the head of the list
2. the remaining part of the list, called the tail

For out example list (above) the head is ann and the tail is the list:[tennis, tom, skiing].

In general, the head can be anything but the tail is always a list. For example, the list:

    [skiing]

has the head *skiing* and the empty list [] as its tail.

The | 'bar' Operator

Prolog also has a special facility to split the first part of the list (the head) away from the rest of the list (the tail). We can place a special symbol | (pronounced 'bar') in the list to distinguish between the first item in the list and the remaining list. For example, consider the following.

[first, second, third] = [A|B]

where A = first and B=[second, third]. The unification here succeeds. A is bound to the first item in the list, and B to the remaining list.

The vertical bar notation is in fact more general than this general. For example, we can list any number of elements followed by | and the list of remaining items. Thus the following lists are all equivalent:

[first, second, third, fourth] = [A,B|C]

this unification will succeed with A = first, B=second and C=[third,fourth].

Or:

[first, second, third] = [A,B,C|D]

this unification will succeed with A = first, B=second, C=third and D=[].

List examples

Consider the following fact.

```
p([H|T], H, T).
```

Let’s see what happens when we ask some simple queries with this fact in the knowledge base.

```
?- p([a,b,c], X, Y).
X=a
Y=[b,c]
yes
```

```
?- p([], X, Y).
no
```

```
?- p([], X, Y).
no
```

Exercises: List processing

Exercise 1: all_as/1

Define a predicate all_as(+List) which succeeds if List is a list containing only a's as elements (if it contains any elements at all).

Hint:

The general strategy for solving this task is to first make sure that the first element of the list (the head) is an 'a'. If that is the case, then you have to deal with the tail of the list. The tail of a list is again a list and you have to make sure that it also is a list which contains only ‘a’s as elements (an all_as list) or is empty.

There is also a more declarative way of describing the predicates that you have to define. Ask yourself: *what are the properties that an all_as list should have?* One possibility is that it is empty. If it is not empty, then its first element should be an a and the tail should be an all_as list.)

Remark: Maybe you wonder what the + in all_as(+List) is all about. It's a notation that is commonly used in specifications of what particular Prolog predicates do (or should do). The plus expresses that the argument that it is attached to should be instantiated when calling the predicate. Similarly, arguments can be prefixed by a minus to express that the argument should not be instantiated or by a question mark to express that the argument may or may not be instantiated.

Exercise 2: replace_a_b_c/2
Define a predicate replace_a_b_c(+InList,?OutList) where OutListis obtained from InList by replacing all a's in InList with b's, all b's with c's, and all c's with a's.

Hint:

There are again two ways of describing what the predicate has to do. The declarative way states what properties two lists have to have in order for the predicate replace_a_b_c to be true. The declarative way says what has to be done to the input list in order to transform it into the output list. These are, of course, just two ways of describing the same predicate definition.

Under the declarative point of view, the predicate should be true when both lists are empty. It should also be true in the following four cases:

- The first element of the input list is an *a*, the first element of the output list is a *b* and the predicate replace_a_b_c is true when applied to the tails of the input list and the output list.
- The first element of the input list is an *b*, the first element of the output list is a *c* and the predicate replace_a_b_c is true when applied to the tails of the input list and the output list.
- The first element of the input list is an *c*, the first element of the output list is a *a* and the predicate replace_a_b_c is true when applied to the tails of the input list and the output list.
- The first element of the input list and the first element of the output list are the same and the predicate replace_a_b_c is true when applied to the tails of the input list and the output list.

The procedural description goes as follows: if the input is an empty list, then there is nothing to replace; output the empty list. If the input is a non-empty list, then there are four cases that you have to deal with. The first element can be an *a*, *b*, or *c*, or neither of them. In the first three cases, specify that the first element of output list is a b, c, or a, respectively and treat the tail of the input list in the same way. The tail of the output list has to be what you get when replacing the *a*'s, *b*'s, and *c*'s of the tail of the input list. If the fourth case (the first element is neither an *a*, nor a *b*, nor a *c*), specify that the first element of the output list is the same as the first element of the input list. And replace all *a*'s, *b*'s, and *c*'s of the tail of the input list.)