Declarative Programming: Handout 7

Yılmaz Kılıçaslan
November 24, 2022

Key Concepts: List, Head / Tail, Recursive Data Structure.

LISTS IN PROLOG

1 INTRODUCTION

Recall from a lecture several weeks ago that a computer program is a combination of data structures and control structures: Data structures organize information, whereas control structures organize algorithms. And, the main emphasis of the preceding two lectures was this: the real power of Prolog comes from recursion. Remember also that Prolog implements a recursive algorithm in a similar way that other programming languages do: by calling a procedure within the body of (at least) one of its rules. Today, we will see a recursive data structure in logic programming / Prolog: the list structure. By grasping and using this structure, we will be equipped with the full power that Prolog provides via recursion (and also, thereby, we will improve our skills to think and program recursively).

Let us explore the syntactic and semantic properties of lists by looking at some examples.

2 EXAMPLES OF LISTS IN PROLOG

2.1 Non-Empty and Empty Lists

Below is a list in Prolog:

(1) [a, b, c, d, e]

The brackets serve to enclose the elements, which are separated by commas.

An empty list (i.e., a list with no elements) is denoted as follows:

(2) []

2.2 Lists versus Sets

The elements of a list, contrary to those of a set, are ordered. Therefore, the following, though containing exactly the same elements, is a list different from that in (1):

(3) [b, c, a, e, d]

Or, we get still a different list if we duplicate an element, even if we keep the same order for the rest, as exemplified below:

(4) [a, a, b, c, d, e]

2.3 Lists as Recursive Structures

As they are ordered, an element of a list can be called by its position, such as ‘the first element’, ‘the second element’, and so on. For the same reason, the whole list can be partitioned as, for instance, ‘the first element and the rest’, ‘the first element, the second element, and the rest’, etc. Note that a list is treated recursively here in virtue of the fact that the rest of a list is another but smaller list. Prolog employs the vertical bar character, |, to separate the element or elements occuring at the beginning of a list from the rest. The beginning component is called the *head* and the rest is called the *tail* of the list. The following are all alternative notations for the list in (1).

(5) [a | [b, c, d, e]]
[a, b | [c, d, e]]
[a, b, c | [d, e]]
[a, b, c, d | [e]]
[a, b, c, d, e | []]

To emphasize, the tail of a list is again a list. Thus, we may have a tail of a tail, a tail of a tail of a tail, and the like. Therefore, the alternative notations for the list in (1) can be easily proliferated:

(6) [a | [b | [c, d, e]]]
[a | [b | [c | [d, e]]]]
[a | [b | [c | [d | e]]]]
[a | [b, c | [d, e]]]
[a | [b, c | [d | e]]]
[a | [b, c, d | [e]]]
[a | [b, c, d, e | []]]
[a, b | [c | [d, e]]]
[a, b | [c | [d | [e]]]]
[a, b | [c | [d, e | []]]]
etc.

2.4 Lists as Heterogenous Structures

An element of a list can be any legitimate prolog term:

(7) [a, 12, john, father(john, mary)]

It can also be a variable:

(8) [a, X, c, Y, e]

It is also possible for another list to occur as an element of a list:

(9) [[a, b, c], [d, e]]

3 RELATIONSHIPS USING LISTS

We can define a great variety of interesting relationhips using lists. Below are some examples (where *List, Prefix, Suffix, Sub, Tsil, Xs, Ys* are all lists):

(10) *% member(Element, List) ← Element is an element of List.*

% prefix(Prefix, List) ← Prefix is a prefix of List.

% suffix(Suffix, List) ← Suffix is a suffix of List.

% sublist(Sub, List) ← Sub is a sublist of List.

% reverse(List, Tsil) ← List is the result of reversing Tsil.

% length(Xs, N) ← Xs has N elements.

% last(X, Xs) ← X is the last element of Xs.

% delete(Xs, X, Ys) ← Ys is the result of removing all occurences of X from Xs.

% select(X, Xs, Ys) ← Ys is the result of removing one occurence of X from Xs.

% insert(X, Xs, Ys) ← Ys is the result of inserting X somewhere in Xs.

% replace(X, Xs, Ys) ← Ys is the result of replacing one element of Xs with X.

The implementations of the procedures are left as exercise.

4 CONCLUSION

Lists are recursively defined data structures in logic programming. A list is either empty or non-empty. A non-empty list can be bi-partitioned as the head that corresponds to the partition falling to the lefthand side and the tail that corresponds to the other partition. The head contains at least one element. The rest is just another list (whether empty or non-empty). Any legitimate term (hence, a list as well) can be an element of a list.

5 EXERCISES

A. Answer the following questions:

1) Are {a, b, c} and {a, a, b, c} the same or different sets? Why?
2) Are [a, b, c] and [a, a, b, c] the same or different lists? Why?

B. How many elements does each of the lists below have?

1) [a, 12, john, father(john, mary)]
2) [a, X, c, Y, e]
3) [[a, b, c], [d, e]]

C. Write the Prolog procedures to implement the relations in (10).

Reference Texts

- Sterling, L. and Shapiro E. (1986) The Art of prolog. Cambridge, Massachusetts: MIT Press.