Introduction to AI with Prolog

Lists in Prolog

Alexey Sery
a.seryj@g.nsu.ru

Novosibirsk State University
*THE REAL SCIENCE

НГУ ФИТ

Department of Information Technologies
Novosibirsk State University

September 29, 2020

Overview

Goals of the lesson

- To introduce lists, an important recursive data structure widely used in computational linguistics.
- To define predicates Prolog provides to work with lists.
- To introduce the idea of recursing down lists.

Functional notation

Lists are Prolog terms for which there are dedicated syntactic provisions.

Lists are defined inductively:

- The atom `[]` (nil) is a list. It denotes an empty list.
- The compound term `.(H, T)` is a list iff `T` is a list. `H` is a first element or **head** of the list, and `T` is a **tail** of the list.

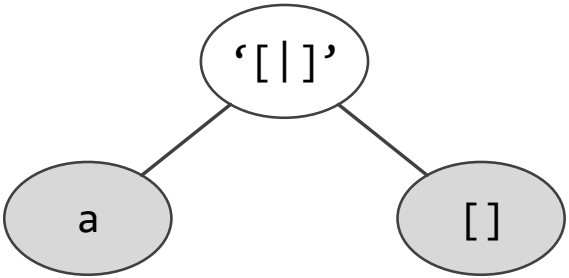
As if to make canonical definition of lists more painful in SWI Prolog predicate `'[|]'` is used instead of `'.'`.

Functional notation

[]	An empty list

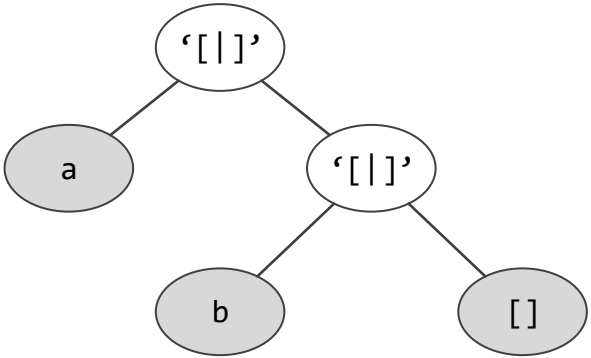
Functional notation

```
'[|]'(a,[])
```



Functional notation

```
'[|]'(a, '[|]'(b, []))
```



List notation

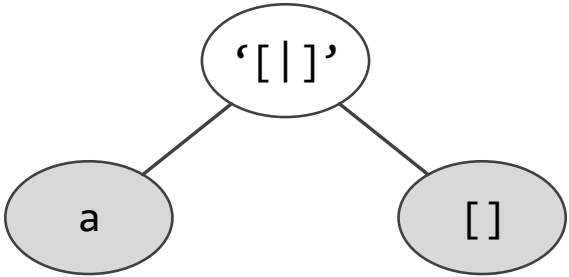
- List is a finite sequence of elements.
- Lists in Prolog are specified by enclosing the elements in square brackets [...].
- The elements are separated by commas: `[one, two, three, 1, 2]`. The length of a list is a number of its elements. Any term could be an element of a list.
- Any non-empty list can be thought of as consisting of the head and the tail. The head is simply the first item in the list; the tail is everything else. SWI Prolog has a special inbuilt operator | which can be used to decompose a list into its head and tail.
- The empty list contains no internal structure.

List notation

`[]`	An empty list

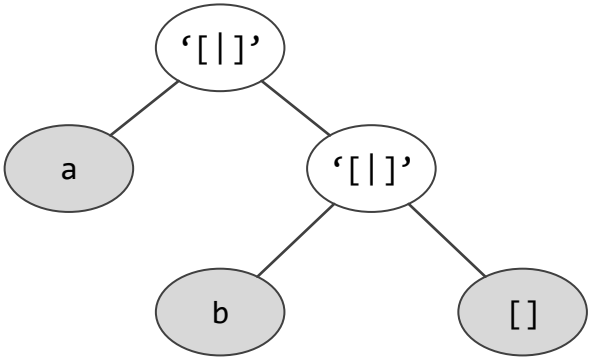
List notation

`[a]`



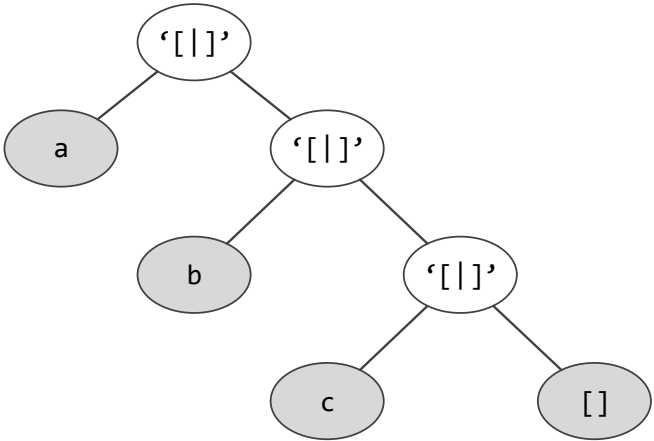
List notation

`[a, b]`



List notation

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



List notation

```
[]
[first, second, third, fourth, fifth]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
[first, 2, color(cornie, black), F, fifth, F]
[first, second, [third, fourth], [fifth, color(cornie, black)]]
[[], [], car(volkswagen), F, 1, 2, [1, F, car(bmw), [1, 2, 4]], X]
```

```
[abyssian, bobtail, [bengal, birman]]
```

Elements and operations

```
[abyssian, bobtail, [bengal, birman]]
```

- `[H|T] = [abyssian, bobtail, [bengal, birman]]`.

Elements and operations

```
[abyssian, bobtail, [bengal, birman]]
```

- `[H|T] = [abyssian, bobtail, [bengal, birman]].`
- `[F,S|T] = [abyssian, bobtail, [bengal, birman]].`

Elements and operations

```
[abyssian, bobtail, [bengal, birman]]
```

- `[H|T] = [abyssian, bobtail, [bengal, birman]].`
- `[F,S|T] = [abyssian, bobtail, [bengal, birman]].`
- `[_,S|T] = [abyssian, bobtail, [bengal, birman]].`

Elements and operations

```
[abyssian, bobtail, [bengal, birman]]
```

- `[H|T] = [abyssian, bobtail, [bengal, birman]].`
- `[F,S|T] = [abyssian, bobtail, [bengal, birman]].`
- `[_,S|T] = [abyssian, bobtail, [bengal, birman]].`
- `[First,_,_,Fourth|_] = [abyssian, bobtail, [bengal, birman]].`

```
[abyssian, bobtail, [bengal, birman]]
```

- `[H|T] = [abyssian, bobtail, [bengal, birman]].`
- `[F,S|T] = [abyssian, bobtail, [bengal, birman]].`
- `[_,S|T] = [abyssian, bobtail, [bengal, birman]].`
- `[First,_,_,Fourth|_] = [abyssian, bobtail, [bengal, birman]].`
- `[_,_,[_|T]|_] = [abyssian, bobtail, [bengal, birman]].`

Elements and operations

The inbuilt predicate `member/2 = member(?Elem, ?List)` is True if `Elem` is a member of `List`, and False otherwise.

`member(a, []).`	`false.`
`member(a, [b, a, 1, [], f(a, b, c)])`	`true`

Elements and operations

The inbuilt predicate `member/2 = member(?Elem, ?List)` is True if `Elem` is a member of `List`, and False otherwise.

`member(a, []).`	`false.`
`member(a, [b, a, 1, [], f(a, b, c)])`	`true`

```
member(X, [X|T]).
```

Elements and operations

The inbuilt predicate `member/2 = member(?Elem, ?List)` is True if `Elem` is a member of `List`, and False otherwise.

`member(a, []).`	`false.`
`member(a, [b, a, 1, [], f(a, b, c)])`	`true`

```
member(X, [X|T]).

member(X, [H|T]) :- member(X,T).
```

Elements and operations

The inbuilt predicate `member/2 = member(?Elem, ?List)` is True if `Elem` is a member of `List`, and False otherwise.

`member(a, []).`	`false.`
`member(a, [b, a, 1, [], f(a, b, c)])`	`true`

```
member(X, [X|T]).

member(X, [H|T]) :- member(X,T).


member(X, [X|_]).

member(X, [_|T]) :- member(X,T).
```

Other operations on lists:

1. Length of a list.
2. Concatenation.
3. Prefix, suffix and a sublist of a list.
4. Get last element.
5. List reversal.
6. Sorting.