Programming in Prolog

Lists

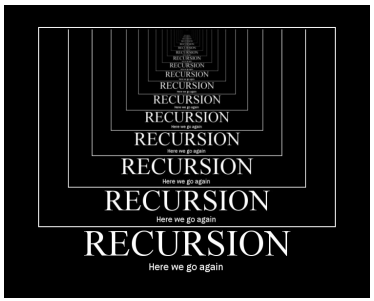
Claudia Schulz

Logic and AI Programming

(Course 518)

What you will learn in this lecture

What are lists in Prolog?



+

How is recursion used in lists?

Some Prolog lists

- `[january, march, september, december]`
- `[7, john, March, X, 'marry']`
- `[ ]`
- `[march, 9, [september, X], 'marry]`
- `[march, 9 | [[september, X], 'marry]]`
- `.(march, .(9, .(.(september, .(X, [])), .('marry', []))))`

What is a Prolog list?

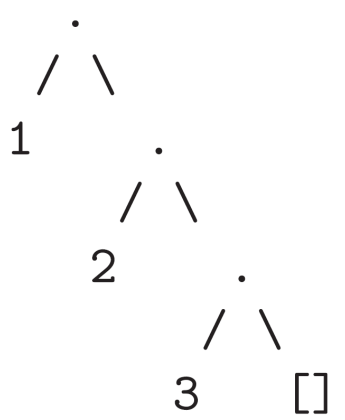
A list is...

a collection of elements in a certain order

- `[ ... ]` denotes a list
- `[ ]` is the empty list

But: `[ ... ]` is just a convenient shorthand notation

`[1,2,3]`



Head and tail

A list is...

some **head** element(s) and a list as the **tail**: `[H | T]`

- `[1|[2,3]]` with `H = 1` and `T = [2,3]`
- same as `[1|[2|[3]]]`
- same as `[1|[2|[3|[]]]]`
- same as `[1,2,3]`

there can be more than one head element: `[H1, H2 | T]`

- `[1,2|[3]]` with `H1 = 1`, `H2 = 2`, `T = [3]`
- `[1,2,3|[]]` with `H1 = 1`, `H2 = 2`, `H3 = [3]`, `T = []`
- same as `[1,2,3]`

$\Rightarrow$ the head cannot be empty, but the tail can be `[]`

Head and tail

A list really is...

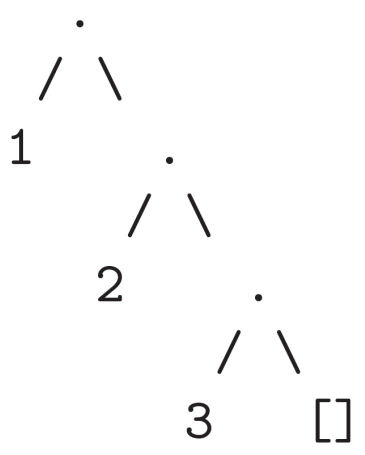
a "pair" made of a first element and a list
$\Rightarrow$ the list is again a "pair" made of a first element and a list...
$\Rightarrow$ the last list in the last pair is []

`[1,2,3]`

really is

`[1|[2|[3|[]]]]`

really is



Unifying lists

- `[H1,H2]` – unifies with any list of 2 elements
  $\Rightarrow$ `[a,b]`: `H1 = a`, `H2 = b`
- `[H1|T]` – unifies with any list with at least 1 element
  $\Rightarrow$ `[a,b,c]`: `H1 = a`, `T = [b,c]`
- `[[H|T1]|T2]` – unifies with any list where the first element is a non-empty list
  $\Rightarrow$ `[[a,b,c],b,d]`: `H = a`, `T1 = [b,c]`, `T2 = [b,d]`

Unifying lists – Try it yourself

Do these lists unify? If so, what is the variable binding?

- `[X,Y,Z]` and `[a,b,c,d]`
- `[X|Y]` and `[a]`
- `[X,Y|Z]` and `[a,b,c,d]`
- `[X|Y]` and `[[a,b],[c,d]]`
- `[X|Y]` and `[[a,b],c,d]`
- `[[X|Y]|Z]` and `[[a,b],c]`
- `[X,Y]` and `[a,[b,c]]`
- `[X,Y]` and `[a]`

Special lists – strings

A **string** is...

a sequence of characters in double quotes
$\Rightarrow$ ‘‘HeLLo m9?@’’

- shorthand for a list of ASCII code decimals
- ‘‘HeLLo m9?@’’ stands for [ASCII-for(H), ASCII-for(e), ..., ASCII-for(@)]
- ‘‘HeLLo m9?@’’ = [72,101,76,76,111,32,109,57,63,64]

`[X,Y|Z] = "HeLLo m9?@"` succeeds with
`X = 72, Y = 101, Z = [76,76,111,32,109,57,63,64]`

Recursion on lists

`is_member_of(X,L)`: `X` is a member of the list `L`

`is_member_of(X,L)`

- base case: `X` is the head of `L`

```
is_member_of(X,L) :-
    L = [X|_].
```

```
is_member_of(X,[X|_]).
```

- recursive definition: `X` is a member of the tail of `L`

```
is_member_of(X,L) :-
    L = [H|T],
    is_member_of(X,T).
```

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

Recursion on lists

`is_member_of(X,L)`: `X` is a member of the list `L`

`is_member_of(X,L)`

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

- Is `X` a member of `L`: `?- is_member_of(3,[1,2,3]).`
- What elements are in `L`: `?- is_member_of(X,[1,2,3]).`
- Generate a list containing `X`: `?- is_member_of(3,L).`

More recursion on lists - Try yourself

`a2b(L1,L2)`: every `a` occurring in a list `L1` occurs as `b` in `L2`; everything else is the same in `L1` and `L2`

- compare every element: either `a - b`, or the same
- base case: easiest possible lists satisfying the condition
- recursion: more complex lists – which elements can be easily compared in two lists?

More recursion on lists - a solution

a2b(L1,L2): every a occurring in a list L1 occurs as b in L2; everything else is the same in L1 and L2

a2b(L1,L2)

```
a2b([],[]).

a2b([a|T1], [b|T2]) :-
    a2b(T1, T2).

a2b([H1|T1], [H1|T2]) :-
    H1 \= a,
    H1 \= b,
    a2b(T1, T2).
```

Built-in predicates for lists

`member(?X,?L)`

`X` is contained in `L`

$\Rightarrow$ same as our `is_member_of`

`nonmember(+X,+L)`

`X` is not contained in `L`

`length(?L,?N)`

the length of list `L` is `N`

Built-in predicates for lists

`append(?L1,?L2,?L3)`

concatenate lists `L1` (first) and `L2` (second) to yield list `L3`

`sort(+L1,-L2)`

sorting the elements of list `L1` (according to the standard order) yields list `L2`

1. variables (in order of occurrence)
2. floats
3. integers
4. constants
5. compound terms (by arity, by function name)

List Exercise

Define predicates for

- `nonmember`
- `length`
- `append`
  Note: define `my_nonmember`, `my_length`, and `my_append`
  $\Rightarrow$ otherwise you will get an error: "Permission error: cannot redefine built_in"

What you should know now

- What are lists in Prolog?
  - What are head and tail?
  - How does a list represent a list of lists?
  - Which built-in predicates can be used for list manipulation?
- How is recursion used for lists?
  - How does Prolog search when performing recursion on a list?
  - How can additional arguments be used to achieve tail-recursion?