Prolog programming: a do-it-yourself course for beginners

Day 3

Kristina Striegnitz

Department of Computational Linguistics

Saarland University, Saarbrücken, Germany

kris@coli.uni-sb.de

http://www.coli.uni-sb.de/~kris

Day 3: Lists

Today:
- calculating with numbers
- processing collections of objects

Reader: Lectures 4, 5, and 6 of *Learn Prolog Now!*

Arithmetic in Prolog

```
?- 3+5 = +(3,5).
yes
?- 3+5 = +(5,3).
no
?- 3+5 = 8.
no
```

- `3+5` is a normal complex term.
- Prolog has to be told explicitly to evaluate it as an arithmetic expressions.
- This done using certain built-in predicates, such as `is/2`, `=:=/2`, `>/2`, etc.

The built-in predicate `is`

```
?- X is 3+5.
X = 8 ;
no
?- X is 30-4.
X = 26 ;
no
?- X is 3*5.
X = 15 ;
no
?- X is 9/4.
X = 2.25 ;
no
```

Attention:

```
?- 3+5 is 8.
no
?- 8 is 3+X.
ERROR: Arguments are not sufficiently
instantiated
?- X = 4, 8 is 3+X.
no
```

Built-in predicates for comparing values of arithmetic expressions

```
?- 8 > 3.
yes
?- 8+2 > 9-2.
yes
?- 8 < 3.
no
?- 8 =:= 3.
no
?- 8 =\= 3.
yes
```

```
?- 8 >= 3.
yes
?- 8 =< 3.
no
```

Lists

- Intuitively: sequences or enumerations of things
- In Prolog: a special kind of data structure, i.e., special kinds of Prolog terms

Prolog lists

Prolog lists either look like this:

the empty list: `[]`

or like this:

non-empty lists: `.(`*`Head`*`,`*`Tail`*`)`

a Prolog term;

i.e., an atom (`dobbey`), a variable (X), a complex term (`house_elf(dobbey)`), a list, a number

a list;

i.e., the empty list `[]` or a non-empty list of the form `.(`*`Head`*`,`*`Tail`*`)`

Examples of lists and non-lists

Lists:

`[]`

`.(a,[])`: this list has one element

`.(a,.(b,[]))`: this list has two elements

`.(b,.(a,[]))`: this list also has two elements

`.(.(a,[]), .(b,[]))`: this list also has two elements. The first element is the list `.(a,[])`, and the second element is the atom `b`.

No lists:

`.([])`

`.(a,b)`

Another way of writing lists in Prolog

- `.(a,Tail) = [a|Tail]`
- `.(a,.(b,Tail)) = [a,b|Tail]`
- `.(a,.(b,.(c,[]))) = [a,b,c]`

Working with lists (1)

`trans_a_b/2`: a predicate for "translating" a list of as into a list of bs.

`trans_a_b(X,Y)` should be true if `X`, the 'input', is a list of as and `Y`, the 'output', is a list of bs which has just as many bs as the input has as.

```
trans_a_b([],[]).
```

If the input is empty, then the output is empty as well.

```
trans_a_b(.(a,InputTail), .(b,OutputTail)) :-
                    trans_a_b(InputTail, OutputTail).
```

Otherwise the first a in the input list has to correspond to a b in the output list. The tail of the output list has to be the "translation" of the input list.

Working with lists (1)

`trans_a_b/2`: a predicate for "translating" a list of as into a list of bs.

`trans_a_b(X,Y)` should be true if `X`, the 'input', is a list of as and `Y`, the 'output', is a list of bs which has just as many bs as the input has as.

```
trans_a_b([],[]).
```

If the input is empty, then the output is empty as well.

```
trans_a_b([a|InputTail], [b|OutputTail]) :-
                    trans_a_b(InputTail, OutputTail).
```

Otherwise the first a in the input list has to correspond to a b in the output list. The tail of the output list has to be the "translation" of the input list.

Working with lists (2)

`element_of/2:` a predicate for testing whether a list contains a given Prolog term.

`element_of(X,Y):` should be true if X is an element of Y.

```
element_of(X, [X|Tail]).
```

If the first element of the list is the one that we are looking for, we are done.

```
element_of(X, [_|Tail]) :- element_of(X,Tail).
```

Otherwise, check whether the term we are looking for is in the tail of the list.

In SWI-Prolog `element_of/2` is predefined under the name `member`.

Working with lists (3)

`concatenate/3:` a predicate for concatenating two lists.

`concatenate(X,Y,Z)` should be true if `Z` is the concatenation of `X` and `Y`; for example, concatenating `[a]` with `[b,c]` yields `[a,b,c]`.

```
concatenate([],L,L).
```

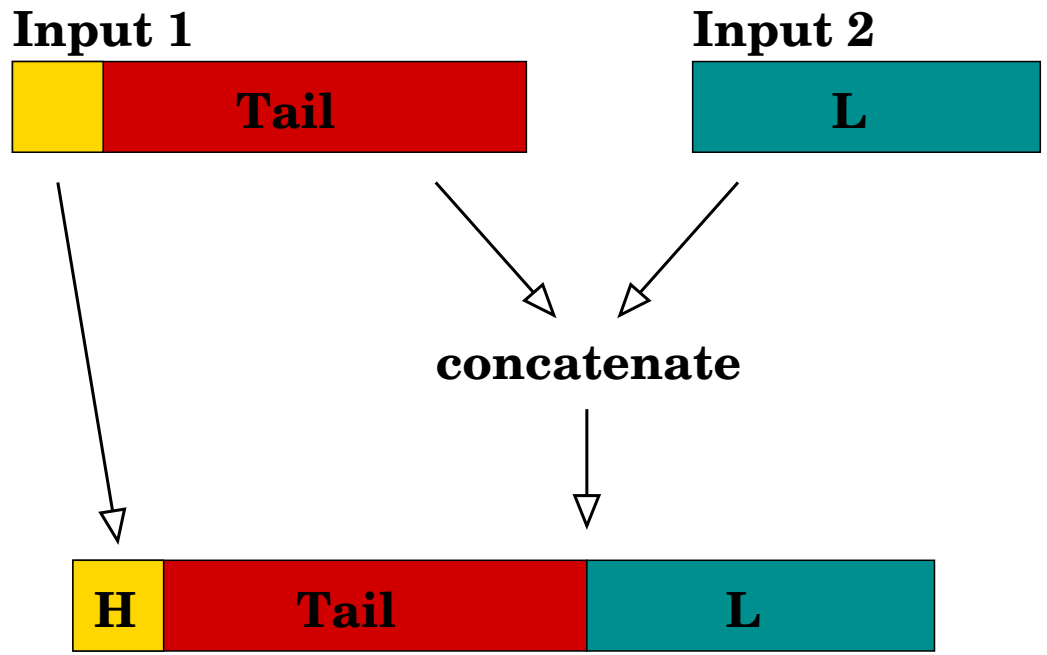
Concatenating the empty list with any other list L yields L.

```
concatenate([Head|Tail],L,[Head|NewTail]) :-
                          concatenate(Tail,L,NewTail).
```

Otherwise, the first element of the output list has to be the same as the first element of the first input list. And the tail of the output list is the concatenation of the tail of the first input list with the second input list.

Concatenating lists

```
concatenate(.(Head,Tail),L,.(Head,NewTail)) :-
                                concatenate(Tail,L,NewTail).
```



In SWI-Prolog `element_of/2` is predefined under the name `member`.

Prolog predicates can be used in many ways

```
?- trans_a_b([a,a,a],L).
L = [b,b,b] ;
no

?- trans_a_b([a,a,a],[b]).
no

?- trans_a_b(L,[b,b]).
L=[a,a] ;
no
```

```
?- member(a,[a,b,c]).
yes

?- member(X,[a,b,c]).
X = a ;
X = b ;
X = c ;
no

?- member(a,L).
L = [a|_G280] ;
L = [_G279, a|_G283] ;
L = [_G279, _G282, a|_G286] ;
L = [_G279, _G282, _G285, a|_G289]
Yes
```

Practical Session

Have fun!

`http://www.coli.uni-sb.de/~kris/esslli04prolog`