5.2 Lists

Dienstag, 20. Juni 2017 09:00

Lists are also supported in Prolog.

Up to now, we had to represent lists by suitable terms: $nil \in \Sigma_0$ $cons \in \Sigma_2$

```
len(nil, 0).
len(cons(X, Xs), Y) :- len(Xs, Y1), Y is Y1+1.
```

cons stands for the list with first element X and Xs is the rest of the list

```
?- len(cons(3, cons(7, nil)), Y).
Y = 2
```

Instead of nil and cons, Prolog has pre-defined lists built with $[] \in \Sigma_0$, $. \in \Sigma_2$

```
len([], 0).
len(.(X, Xs), Y) :- len(Xs, Y1), Y is Y1 + 1.
```

Prolog offers the following alternative ways to write lists built with [] and .

= $.(t_1, t_2)$ = $[t_1 \mid t_2]$

= $.(t_1, [])$ = $[t_1]$

= $.(t_1, .(t_2, .(t_3, t)))$ = $[t_1, t_2, t_3 \mid t]$

$= .(t_1, .(t_2, .(t_3, [\,]))) = [t_1, t_2, t_3]$

$= [t_1, t_2 \mid [t_3 \mid [\,]]] = \ldots$

The shorthand notations are considered to be syntactically identical to the corresponding term built with [ ] and .

```
?- [1,2] == [1|[2]].
true
```

← .(1, .(2, .(3, [])))

```
?- .(1,X) = [1,2,3].
X = [2,3]

?- [X,[1|X]] = [[2], Y].
X = [2], Y = [1,2]
```

Ex: List Concatenation (append)

```
app([], Ys, Ys).
app([X|Xs], Ys, [X|Zs]) :- app(Xs, Ys, Zs).

?- app([1,2], [3,4,5], Xs).
Xs = [1,2,3,4,5]
```

?- app(Xs, Ys, [1,2,3]).

Xs = [], Ys = [1,2,3] ;

Xs = [1], Ys = [2,3] ; ...