5.2 Lists

Montag, 8. Juni 2015 08:30

Representation of lists as terms:

$nil \in \Sigma_0$ to represent the empty list

$cons \in \Sigma_2$ to represent list insertion

(Names from Lisp)

i.e., $cons(7, cons(3, nil))$ stands for a list with the elements 7 and 3. – [7,3]

length-algorithm for user-defined lists:

len(l, z) iff the length of the list l is z

```
len(nil, 0).
len(cons(X, XS), Z) :- len(XS, Z'),
                       Z is Z'+1.
```

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

Prolog has a pre-defined data structure for lists:

$[\,] \in \Sigma_0$ for empty list

$\in \Sigma_2$ for list insertion

.(7, .(3, [])) stands for [7, 3].

```
len([], 0).
len(.(X, XS), Z) :- len(XS, Z'),
                    Z is Z' + 1.
```

For lists built with [] and ., Prolog offers alternative notations to improve readability:

- $.(t_1, t_2) = [t_1 \mid t_2]$ [7, 3] = [7|[3]]
- $.(t_1, []) = [t_1]$.(7, []) = [7]
- $.(t_1, .(t_2, .(t_3, t))) = [t_1, t_2, t_3 \mid t]$
- $.(t_1, .(t_2, .(t_3, []))) = [t_1, t_2, t_3]$
  $= [t_1, t_2 \mid [t_3 \mid []]]$
  $= [t_1 \mid [t_2, t_3 \mid []]]$

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

← short notations are converted to notation with . and [] => these terms are considered

to be syntactically equal

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

X = [2,3]

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

X = [2], Y = [1,2]

Algorithms on lists:

app should append/concatenate lists

app(t_1, t_2, t_3) should hold iff

t_3 is the concatenation of t_1 and t_2

e.g. app([1,2], [3,4,5], [1,2,3,4,5])

(append/3 is pre-defined in Prolog).

```
app([], XS, XS).
app([X|XS], YS, [X|ZS]) :- app(XS, YS, ZS).
```