5.2 Lists

Montag, 8. Juni 2015  08:30

Representation of lists as terms:

\[ \text{nil} \in \Sigma_0 \quad \text{to represent the empty list} \]

\[ \text{Cons} \in \Sigma_2 \quad \text{to represent list insertion} \]

i.e., \( \text{Cons} (7, \text{Cons} (3, \text{nil})) \) stands for a list with the elements 7 and 3. \(-[7,3]\)

Length-algorithm for user-defined lists:

\[ \text{len} (l, 0). \]

\[ \text{len} (\text{Cons} (X, XS), Z) :- \text{len} (XS, Z'), Z \text{ is } Z' + 1. \]

?- \text{len} (\text{Cons} (7, \text{Cons} (3, \text{nil})), Z).

\[ Z = 2 \]

Prolog has a pre-defined data structure for lists:

\[ [\ ] \in \Sigma_0 \quad \text{for empty list} \]

\[ , \in \Sigma_2 \quad \text{for list insertion} \]
. $(7, .(3, [3]))$ stands for $[7,3].$

```
\text{len } ([3,0]).
```

\text{len } (. (X, XS, Z) : - \text{len } (XS, Z'),
\text{ Z is Z' + 1.}

For lists built with $[3]$ and $\cdot$, Prolog offers alternative notations to improve readability:

- \text{.} (t_1, t_2) = [t_1 | t_2] \quad [7,3] = [7|3]
- \text{.} (t_1, [3]) = [t_1] \quad (7, [3]) = [7]
- \text{.} (t_1, \cdot (t_2, \cdot (t_3, t))) = [t_1, t_2, t_3 | t]
- \text{.} (t_1, \cdot (t_2, \cdot (t_3, [3]))) = [t_1, t_2, t_3]
  = [t_1, t_2 | \text{[t_3 | [3]]}]
  = [t_1 | [t_2, t_3 | [3]]]

\text{? - [1,2] = [1] | [2,3].} \quad \text{short notations are converted to notation with \cdot and [3]}
\text{\Rightarrow these terms are considered}\text{true}.
to be syntactically equal

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

?- \( [X, [\lambda \text{\textbackslash } x \text{\textbackslash }] = [[2], Y] \).
\( X = [2], Y = [1,2] \)

Algorithms on lists:

- `app` should apppend/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], [3,4,5], [1,2,3,4,5])`

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

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