

Funcons-beta: Lists *

The P_LanCompS Project

Lists.cbs | PLAIN | PRETTY

Lists

```
[ Datatype  lists
  Funcon    list
  Funcon    list-elements
  Funcon    list-nil
  Alias      nil
  Funcon    list-cons
  Alias      cons
  Funcon    list-head
  Alias      head
  Funcon    list-tail
  Alias      tail
  Funcon    list-length
  Funcon    list-append ]
```

Meta-variables $T <: \text{values}$

Datatype $\text{lists}(T) ::= \text{list}(_ : (T)^*)$

$\text{lists}(T)$ is the type of possibly-empty finite lists $[V_1, \dots, V_n]$ where $V_1 : T, \dots, V_n : T$.

N.B. $[T]$ is always a single list value, and *not* interpreted as the type $\text{lists}(T)$.

The notation $[V_1, \dots, V_n]$ for $\text{list}(V_1, \dots, V_n)$ is built-in.

Assert $[V^* : \text{values}^*] == \text{list}(V^*)$

Funcon $\text{list-elements}(_ : \text{lists}(T)) : \Rightarrow (T)^*$

Rule $\text{list-elements}(\text{list}(V^* : \text{values}^*)) \rightsquigarrow V^*$

Funcon $\text{list-nil} : \Rightarrow \text{lists}(_)$

$\rightsquigarrow []$

Alias $\text{nil} = \text{list-nil}$

*Suggestions for improvement: plancomps@gmail.com.
Reports of issues: <https://github.com/plancomps/CBS-beta/issues>.

Funcon $\text{list-cons}(_ : T, _ : \text{lists}(T)) : \Rightarrow \text{lists}(T)$

Alias $\text{cons} = \text{list-cons}$

Rule $\text{list-cons}(V : \text{values}, [V^* : \text{values}^*]) \rightsquigarrow [V, V^*]$

Funcon $\text{list-head}(_ : \text{lists}(T)) : \Rightarrow (T)?$

Alias $\text{head} = \text{list-head}$

Rule $\text{list-head}[V : \text{values}, _ : \text{values}^*] \rightsquigarrow V$

Rule $\text{list-head}[] \rightsquigarrow ()$

Funcon $\text{list-tail}(_ : \text{lists}(T)) : \Rightarrow (\text{lists}(T))?$

Alias $\text{tail} = \text{list-tail}$

Rule $\text{list-tail}[_ : \text{values}, V^* : \text{values}^*] \rightsquigarrow [V^*]$

Rule $\text{list-tail}[] \rightsquigarrow ()$

Funcon $\text{list-length}(_ : \text{lists}(T)) : \Rightarrow \text{natural-numbers}$

Rule $\text{list-length}[V^* : \text{values}^*] \rightsquigarrow \text{length}(V^*)$

Funcon $\text{list-append}(_ : (\text{lists}(T))^*) : \Rightarrow \text{lists}(T)$

Rule $\text{list-append}([V_1^* : \text{values}^*], [V_2^* : \text{values}^*]) \rightsquigarrow [V_1^*, V_2^*]$

Rule $\text{list-append}(L_1 : \text{lists}(_), L_2 : \text{lists}(_), L_3 : \text{lists}(_), L^* : (\text{lists}(_))^*) \rightsquigarrow$
 $\text{list-append}(L_1, \text{list-append}(L_2, L_3, L^*))$

Rule $\text{list-append}() \rightsquigarrow []$

Rule $\text{list-append}(L : \text{lists}(_)) \rightsquigarrow L$

Datatypes of infinite and possibly-infinite lists can be specified as algebraic datatypes using abstractions.