Funcons-beta: Lists

The PLanCompS Project

`Funcons-beta/Values/Composite/Lists/Lists.cbs`[*]

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 <: values
```

```
Datatype lists(T) ::= list(_ : (T)*)
```

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

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

The notation $[V_1, \cdots, V_n]$ for list($V_1, \cdots, V_n$) is built-in.

```
Assert [V* : values*] == list(V*)
```

[*] Suggestions for improvement: `plancomps@gmail.com`.
Issues: `https://github.com/plancomps/CBS-beta/issues`.

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

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

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

  $\leadsto [\,]$

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

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

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

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

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

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

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

Rule $\mathsf{list\text{-}head}\ [\,] \leadsto (\,)$

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

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

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

Rule $\mathsf{list\text{-}tail}\ [\,] \leadsto (\,)$

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

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

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

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

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

  Rule $\mathsf{list\text{-}append}(\,) \leadsto [\,]$

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

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