Lists
Contents

What is a list?

A list is: an ordered collection of values of the same type.

Examples:

⟨1,3,3,1,5⟩
⟨**true**,**false**,**true**⟩

List Type Expressions

- type_expr*
  denotes the type consisting of list values
  $\langle v_1, \ldots, v_n \rangle$
  where $n \geq 0$, v_i : type_expr
- type_expr$^\omega$
  denotes the type consisting of list values
  $\langle v_1, \ldots, v_n \rangle$,
  $\langle v_1, \ldots, v_n, \ldots \rangle$
  where $n \geq 0$, v_i : type_expr

List Type Expressions
Examples

Bool*

denotes the type consisting of the lists:

⟨⟩
⟨**true**⟩, ⟨**false**⟩
⟨**true**,**false**⟩, ⟨**false**,**true**⟩, ⟨**true**,**true**⟩, ⟨**false**,**false**⟩,
⟨**true**,**false**,**true**⟩,
⋮

Bool$^\omega$

denotes the type consisting of the lists:

⟨⟩,
⟨**true**⟩, ⟨**false**⟩,
⟨**true**,**false**⟩, ⟨**false**,**true**⟩, ⟨**true**,**true**⟩, ⟨**false**,**false**⟩,
⟨**true**,**false**,**true**⟩
⋮
⟨**false**,**true**,**true**,**true**,**false**,... ⟩
⋮

List Value Expressions

Enumerated:

⟨1,3,3,1,5⟩
⟨**true**,**false**,**true**⟩

⟨value_expr$_1$,...,value_expr$_n$⟩

Ranged:

⟨3 .. 7⟩ = ⟨3,4,5,6,7⟩
⟨3 .. 3⟩ = ⟨3⟩
⟨3 .. 2⟩ = ⟨⟩

⟨value_expr$_1$ .. value_expr$_2$⟩

Comprehended:

⟨2*n | n **in** ⟨0 .. 3⟩⟩
⟨n | n **in** ⟨0 .. 100⟩ • is_even(n)⟩

⟨value_expr$_1$ | binding **in** value_expr$_2$ • *logical*-value_expr$_3$⟩

List Indexing

Basic form:

list-value_expr(value_expr$_1$)

Example:

$\langle 2,5,3 \rangle(2) = 5$

Derived form:

list-value_expr(value_expr$_1$)...(value_expr$_n$)

Example:

$\langle\langle 2,5,3 \rangle, \langle 3 \rangle\rangle(1) = \langle 2,5,3 \rangle$
$\langle\langle 2,5,3 \rangle, \langle 3 \rangle\rangle(1)(2) = \langle 2,5,3 \rangle(2) = 5$

Built-in List Operators

$\frown : T^* \times T^\omega \to T^\omega$
hd : $T^\omega \xrightarrow{\sim} T$
tl : $T^\omega \xrightarrow{\sim} T^\omega$
len : $T^\omega \xrightarrow{\sim}$ **Nat**
elems : $T^\omega \to$ T**-infset**
inds : $T^\omega \to$ **Nat-infset**

$\langle e_1,\ldots,e_n \rangle \frown \langle e_{n+1},\ldots \rangle = \langle e_1,\ldots,e_n,e_{n+1},\ldots \rangle$

hd $\langle e_1,e_2,\ldots \rangle = e_1$

tl $\langle e_1,e_2,\ldots \rangle = \langle e_2,\ldots \rangle$

len $\langle e_1,\ldots,e_n \rangle = n$
len il $\equiv$ **chaos**

elems $\langle e_1,e_2,\ldots \rangle = \{e_1,e_2,\ldots\}$

inds fl $= \{1$ .. **len** fl$\}$
inds il $= \{$idx | idx : **Nat** $\cdot$ idx $\geq 1\}$

Defining Infinite Lists

value
  all_natural_numbers : **Nat**$^\omega$
axiom
  inds all_natural_numbers $=$ { n | n : **Nat** $\cdot$ n > 0 },
  all_natural_numbers(1) $= 0$,
  $\forall$ idx : **Nat** $\cdot$
    idx $\geq 2 \Rightarrow$
      all_natural_numbers(idx) $=$
        all_natural_numbers(idx $-$ 1) $+ 1$

List Example 1

scheme QUEUE =
  class
    type
      Element,
      Queue = Element*
    value
      empty : Queue $= \langle\rangle$,

      put : Element $\times$ Queue $\to$ Queue
      put(e,q) $\equiv$ q $\frown \langle$e$\rangle$,

      get : Queue $\xrightarrow{\sim}$ Queue $\times$ Element
      get(q) $\equiv$ (**tl** q,**hd** q)
      pre q $\neq$ empty
  end

List Example 2

scheme LIST_PROPERTIES =
  class
    value
      is_permutation : **Int*** × **Int*** → **Bool**,
      is_permutation(l1,l2) ≡
        (∀ i : **Int** •
          card {idx | idx : **Nat** •
                    idx ∈ **inds** l1 ∧ l1(idx) = i}
          =
          card {idx | idx : **Nat** •
                    idx ∈ **inds** l2 ∧ l2(idx) = i}),

      is_sorted : **Int*** → **Bool**
      is_sorted(l) ≡
        (∀ idx1,idx2 : **Nat** •
          {idx1,idx2} ⊆ **inds** l ∧ idx1 < idx2 ⇒
            l(idx1) ≤ l(idx2))
  end

List Example 2 – continued

scheme SORTING =
  extend LIST_PROPERTIES **with**
  class
    value
      sort : **Int*** → **Int***
      sort(l) **as** l1 **post** is_permutation(l1,l) ∧ is_sorted(l1)
  end

List Example 2 – continued

scheme SORTING =
  class
    value
      is_permutation : **Int*** × **Int*** → **Bool**,
      is_permutation(l1,l2) ≡
        (∀ i : **Int** •
          card {idx | idx : **Nat** •
                    idx ∈ **inds** l1 ∧ l1(idx) = i}
          =
          card {idx | idx : **Nat** •
                    idx ∈ **inds** l2 ∧ l2(idx) = i}),

      is_sorted : **Int*** → **Bool**
      is_sorted(l) ≡
        (∀ idx1,idx2 : **Nat** •
          {idx1,idx2} ⊆ **inds** l ∧ idx1 < idx2 ⇒
            l(idx1) ≤ l(idx2))

    value
      sort : **Int*** → **Int***
      sort(l) **as** l1 **post** is_permutation(l1,l) ∧ is_sorted(l1)
  end

Example 3: Database Representations

- **type** Database = (Key × Data)**-set**

  {}
  {(k_1,d_1), (k_2,d_2)}

- **type** Database = (Key × Data)*

  ⟨⟩
  ⟨(k_1,d_1), (k_2,d_2)⟩

Example 3: Database Keys & Data

```
scheme KEY =
  class
    type
      Key
    value
      less_than : Key × Key → Bool
    axiom
      [anti_reflexive]
        ∀ k : Key • ~less_than(k,k),
      [transitive]
        ∀ k1,k2,k3 : Key •
          less_than(k1,k2) ∧ less_than(k2,k3) ⇒ less_than(k1,k3),
      [total_order]
        ∀ k1,k2,k3 : Key •
          less_than(k1,k2) ∨ less_than(k2,k1) ∨ k1 = k2
  end

scheme DATA =
  class
    type Data
    value not_found : Data
  end
```


Example 3: Database Records

```
scheme RECORD =
  extend KEY with extend DATA with
  class
    type
      Record = Key × Data
    value
      new_record : Key × Data → Record
      new_record(k,d) ≡ (k,d),

      key_of : Record → Key
      key_of(k,d) ≡ k,

      data_of : Record → Data
      data_of(k,d) ≡ d
  end
```


Example 3: Database Modeled as Lists

```
scheme LIST_DATABASE =
  extend RECORD with
  class
    type
      Database = {| rl : Record* • is_wf_Database(rl) |}
    value
      is_wf_Database : Record* → Bool
      is_wf_Database(rl) ≡
        (∀ r1,r2 : Record, left,right : Record* •
          rl = left ^ ⟨r1,r2⟩ ^ right ⇒
            less_than(key_of(r1),key_of(r2))),
```


Example 3: LIST_DATABASE continued

```
      empty : Database = ⟨⟩,

      insert : Key × Data × Database → Database
      insert(k,d,db) as db1
        post elems db1 =
          (elems remove(k,db)) ∪ {new_record(k,d)},

      remove : Key × Database → Database
      remove(k,db) ≡ ⟨r | r in db • key_of(r) ≠ k⟩,

      defined : Key × Database → Bool
      defined(k,db) ≡
        if db = ⟨⟩ ∨ less_than(k,key_of(hd db)) then false
        else key_of(hd db) = k ∨ defined(k,tl db)
        end,

      lookup : Key × Database → Data
      lookup(k,db) ≡
        if db = ⟨⟩ ∨ less_than(k,key_of(hd db))
          then not_found
        elsif key_of(hd db) = k then data_of(hd db)
        else lookup(k,tl db) end
  end
```

Sets, Lists and Products

	number of elements	type of elements	duplicates	ordered
T-**set**	varying	same	no	no
T-**infset**	varying	same	no	no
T^*	varying	same	yes	yes
T^ω	varying	same	yes	yes
$T_1 \times ... \times T_n$	fixed (n)	varying	yes	yes