Functional Programming / Lean

More about lists

Prof. Dr. Peter Thiemann

Albert-Ludwigs-Universität Freiburg, Germany

WS 2026/27

Lists recap

Zero or more values

```
[]    [1]    [true, false]    ["a", "bunch", "of", "flowers"]
```

All elements have the same type

```
[true, false] -- good
[1, "two", false] -- bad, type error
```

Order matters

```
[1,2,3] != [3,2,1]
```

List syntax

```
(1 :: (2 :: (3 :: [])))
==
1 :: 2 :: 3 :: []
==
[1,2,3]
```

Alternatively use `List.cons` and `List.nil` as list constructors.

Strings are distinct from characters, but conversion is possible.

```
"Hearts".toList == ['H','e','a','r','t','s']
String.ofList ['H','e','a','r','t','s'] == "Hearts"
```

Defining a list datatype

The values of type [a] are ...

- either `[]`, the empty list
- or `x::xs` where `x` has type `a` and `xs` has type `[a]`
  "`::`" is pronounced "cons"

Defining a list datatype

The values of type [a] are ...

- either `[]`, the empty list
- or `x::xs` where `x` has type `a` and `xs` has type `[a]`
  "`::`" is pronounced "cons"

Corresponding definition

```
1 inductive List (α : Type) where
2   | nil : List α
3   | cons : α → List α → List α
```

- New: `List` is a parametric datatype with type parameter α
- Many functions on lists are also parametric (i.e., **polymorphic**)

Predefined polymorphic functions on lists

```
1 List.length  : List α -> Int
2 List.append  : List α -> List α -> List α
3 List.flatten : List (List α) -> List α
4 List.take    : Nat -> List α -> List α
5 List.drop    : Nat -> List α -> List α
6 List.zip     : List α → List β → List (α × β)
7
8 List.map     : (α → β) → List α → List β
9 List.filter  : (α → Bool) → List α → List α
```

Std functions on lists

Functions on specific lists

```
List.and  : List Bool -> Bool
List.or   : List Bool -> Bool

def words (s : String) : List String :=
  ((s.split Char.isWhitespace).toStringList).filter (· != "")

def lines (s : String) : List String :=
  s.lines.toStringList

def unwords (ss : List String) : String :=
  " ".intercalate ss

def unlines (ss : List String) : String :=
  String.join (ss.map (fun s => s ++ "\n"))
```

Std functions on lists

Overloaded functions on lists

```
#eval ([1, 2, 3, 4] : List Nat).sum      -- 10
#eval ([1, 2, 3, 4] : List Nat).prod     -- 24
#eval ([1, 2, 3, 4] : List Nat).elem 3   -- true
#eval ([3, 1, 4, 2] : List Nat).mergeSort -- [1, 2, 3, 4]
```

Quicksort!

```
1 def qsort {α : Type} [LT α] [LE α] [DecidableLT α] [DecidableLE α] : List α →
      List α
2   | [] => []
3   | x :: xs =>
4     let smaller := xs.filter (fun y ↦ decide (y ≤ x))
5     let larger := xs.filter (fun y ↦ decide (y > x))
6     qsort smaller ++ [x] ++ qsort larger
```

. . . is not accepted as is . . .

Quicksort terminates . . .

We have to explain Lean that it terminates (details later)

```
termination_by xs => xs.length
decreasing_by
  all_goals
    simp_wf
    apply Nat.lt_succ_of_le
    calc
      _ ≤ xs.attach.length := List.length_filter_le ..
      _ = xs.length := by simp
```

Break Time — Questions?

