

Chapter 19: Lists in Agda

Zhenjiang Hu, Wei Zhang
School of Computer Science, PKU
November 9, 2022

The List Datatype and Type Parameters

```
data L {l} (A : Set l) : Set l where
  [] : L A
  _::_ : (x : A) (xs : L A) → L A
```

[]

1 :: 2 :: 3 :: []

tt :: tt :: ff :: ff :: []

Basic Operations on Lists

```
[_] : ∀ {ℓ} {A : Set ℓ} → A → ℒ A  
[ x ] = x :: []
```

```
is-empty : ∀ {ℓ} {A : Set ℓ} → ℒ A → ℬ  
is-empty [] = tt  
is-empty (_ :: _) = ff
```

```
head : ∀ {ℓ} {A : Set ℓ} → (ℓ : ℒ A) → is-empty ℓ ≡ ff → A  
head [] ()  
head (x :: xs) _ = x
```

```
head2 : ∀ {ℓ} {A : Set ℓ} → (ℓ : ℒ A) → maybe A  
head2 [] = nothing  
head2 (a :: _) = just a
```

Basic Operations on Lists

```
length : ∀{ℓ}{A : Set ℓ} → ℒ A → ℕ
length [] = 0
length (x :: xs) = suc (length xs)

_++_ : ∀ {ℓ} {A : Set ℓ} → ℒ A → ℒ A → ℒ A
[] ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)

map : ∀ {ℓ ℓ'} {A : Set ℓ} {B : Set ℓ'} → (A → B) → ℒ A → ℒ B
map f [] = []
map f (x :: xs) = f x :: map f xs

filter : ∀{ℓ}{A : Set ℓ} → (A → B) → ℒ A → ℒ A
filter p [] = []
filter p (x :: xs) = let r = filter p xs in
                      if p x then x :: r else r

foldr : ∀{ℓ ℓ'}{A : Set ℓ}{B : Set ℓ'} → (A → B → B) → B → ℒ A → B
foldr f b [] = b
foldr f b (a :: as) = f a (foldr f b as)
```

Reasoning about List Operations

```
length-++ :  $\forall \{l\} \{A : \text{Set } l\} (l1\ l2 : \mathbb{L}\ A) \rightarrow$   
            $\text{length } (l1\ ++\ l2) \equiv (\text{length } l1) + (\text{length } l2)$ 
```

```
length-++ [] l2 = refl
```

```
length-++ (h :: t) l2 rewrite length-++ t l2 = refl
```

```
map-append :  $\forall \{l\ l'\} \{A : \text{Set } l\} \{B : \text{Set } l'\} \rightarrow$   
             $(f : A \rightarrow B) (l1\ l2 : \mathbb{L}\ A) \rightarrow$   
             $\text{map } f (l1\ ++\ l2) \equiv (\text{map } f\ l1) ++ (\text{map } f\ l2)$ 
```

```
map-append f [] l2 = refl
```

```
map-append f (x :: xs) l2 rewrite map-append f xs l2 = refl
```

Length of Filtered Lists, and the with Construct

```
length-filter :  $\forall \{l\} \{A : \text{Set } l\} (p : A \rightarrow \mathbb{B}) (l : \mathbb{L} A) \rightarrow$   
               length (filter p l)  $\leq$  length l  $\equiv$  tt
```

```
length-filter p [] = refl  
length-filter p (x :: l) with p x  
length-filter p (x :: l) | tt = length-filter p l  
length-filter p (x :: l) | ff =  
   $\leq$ -trans{length (filter p l)}  
    (length-filter p l)  
    ( $\leq$ -suc (length l))
```

postulate

```
 $\leq$ -trans :  $\forall \{x y z : \mathbb{N}\} \rightarrow$   
            $x \leq y \equiv$  tt  $\rightarrow y \leq z \equiv$  tt  $\rightarrow x \leq z \equiv$  tt  
 $\leq$ -suc : (x :  $\mathbb{N}$ )  $\rightarrow x \leq$  suc x  $\equiv$  tt
```

Filter Is Idempotent, and the keep Idiom

```
filter-idem :  $\forall \{l\} \{A : \text{Set } l\} (p : A \rightarrow \mathbb{B}) (l : \mathbb{L} A) \rightarrow$   
              (filter p (filter p l))  $\equiv$  (filter p l)  
filter-idem p [] = refl  
filter-idem p (x :: l) with keep (p x)  
filter-idem p (x :: l) | tt , p'  
    rewrite p' | p' | filter-idem p l = refl  
filter-idem p (x :: l) | ff , p'  
    rewrite p' = filter-idem p l
```

Homework

- 19.1. Define a polymorphic function `takeWhile`, which takes in a predicate on type A (i.e., a function of type $A \rightarrow B$), and a list of A s, and returns the longest prefix of the list that satisfies the predicate.
- 19.2. Define a function `repeat` function that takes a number n and an element a , and constructs a list of length n where all elements are just a .
- 19.3. Prove that if value a satisfies predicate p , then `takeWhile p (repeat n a)` is equal to `repeat n a`, where `takeWhile` is the function you defined in the previous problem.