Chapter 19: Lists in Agda

Zhenjiang Hu, Wei Zhang

School of Computer Science, PKU

November 19, 2025

北京大學 PEKING UNIVERSITY

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 :: []

PEKING UNIVERSITY 北京大學

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
```


PEKING UNIVERSITY

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-++ : ∀{ℓ}{A : Set ℓ}(l1 l2 : 𝕃 A) →
              length (l1 ++ l2) ≡ (length l1) + (length l2)

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

```
map-append : ∀ {ℓ ℓ'} {A : Set ℓ} {B : Set ℓ'} →
             (f : A → B) (l1 l2 : 𝕃 A) →
             map f (l1 ++ l2) ≡ (map f l1) ++ (map f l2)

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


PEKING UNIVERSITY 北京大學

Length of Filtered Lists, and the with Construct

```
length-filter : ∀{ℓ}{A : Set ℓ}(p : A → 𝔹)(l : 𝕃 A) →
                length (filter p l) ≤ length l ≡ 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 =
   ≤-trans{length (filter p l)}
      (length-filter p l)
      (≤-suc (length l))
```

```
postulate
   ≤-trans : ∀ {x y z : ℕ} →
             x ≤ y ≡ tt → y ≤ z ≡ tt → x ≤ z ≡ tt
   ≤-suc : (x : ℕ) → x ≤ suc x ≡ tt
```


北京大學 PEKING UNIVERSITY

Filter Is Idempotent, and the keep Idiom

```
filter-idem : ∀{ℓ}{A : Set ℓ}(p : A → 𝔹)(l : 𝕃 A) →
              (filter p (filter p l)) ≡ (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
```


PEKING UNIVERSITY

Homework

19.1. Define a polymorphic function **takeWhile**, which takes in a predicate on type A (i.e., a function of type A → B), and a list of As, 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.