#LECTURE 9: Amortized Incrementing

"Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk"

("God made the integers, all else is the work of man")

— Leopold Kronecker, 1886

Given operations op_i between states xs_i:

$$xs_0 \overset{op_0}{\rightsquigarrow} xs_1 \overset{op_1}{\rightsquigarrow} xs_2 \overset{op_2}{\rightsquigarrow} \cdots \overset{op_{n-1}}{\rightsquigarrow} xs_n$$

Our goal is to establish:

$$\sum_{i=0}^{n-1} C_{op_i}(xs_i) \leq A_{op_i}(xs_i) \qquad (*)$$

In the previous lecture, we defined:

```
> tail :: Deque a -> Deque a
> tail (Deque [] sy)  = empty
> tail (Deque [x] sy) = Deque us sv
>   where
>     ys       = reverse sy
>     (us, vs) = splitAt (n `div` 2) ys
>     sv       = reverse vs
>     n        = length ys
```

What is the cost of:

$$xs_0 \overset{tail}{\rightsquigarrow} xs_1 \overset{tail}{\rightsquigarrow} \cdots \overset{tail}{\rightsquigarrow} xs_n$$

We can establish (*) by proving that:

$$C_{op_i}(xs_i) \leq A_{op_i}(xs_i) + S(xs_i) - S(xs_{i+1})$$

We apply the 3 steps for amortized analysis:

1. $C_{cons}(xs) = 1$ $\quad C_{snoc}(xs) = 1$

   $C_{head}(xs) = 1$ $\quad C_{tail}(xs) = 1$

   $C_{tail}(\text{Deque } us\ sv) = \text{length } sv$

2. $A_{op}(xs) = 2$

ie

$A_{cons}(xs) = 2$
$A_{head}(xs) = 2$
$A_{tail}(xs) = 2$

3. $S(\text{Deque } us\ sv) = |\text{length } us - \text{length } sv|$

$$C_{op_i}(xs_i) \leq A_{op_i}(xs_i) + S(xs_i) - S(xs_{i+1})$$

Consider op_i = tail in the worst case:

xs_i = Deque [x] sy where lenth sy = k

let xs_{i+1} = Deque xs' sy' = tail (Deque [x] sy)

$S(xs_i) = S(\text{Deque } [x]\ sy) = k-1$

$S(xs_{i+1}) = S(\text{Deque } xs'\ sy') \leq 1$

$$C_{tail}(\text{Deque } xs\ sy) \leq A_{tail}(\text{Deque } xs\ sy) + S(\text{Deque } xs\ sy) - S(\text{Deque } xs'\ sy')$$

$$k \leq 2 + (k-1) - 1$$

```haskell
> data Peano = Zero | Succ Peano.

> incr :: Peano -> Peano
> incr n = Succ n

> decr :: Peano -> Peano
> decr (Succ n) = n

> add :: Peano -> Peano -> Peano
> add Zero n = n
> add (Succ m) n = Succ (add m n)
```

Compare these functions with lists:

```
> data [a] = [] | a : [a]

> cons :: a -> [a] -> [a]
> cons x xs = x:xs

> tail :: [a] -> [a]
> tail (x:xs) = xs

> (++) :: [a] -> [a] -> [a]
> [] ++ ys = ys
> (x:xs) ++ ys = x : (xs ++ ys)
```

What is the cost of incr in binary?

```
> type Binary = [Bit]
```

(least significant bit first)

```
> data Bit = I | O

> incr :: Binary -> Binary.
> incr [] = [I]
> incr (O : bs) = I : bs
> incr (I : bs) = O : incr bs
```

What is the complexity?

Let's do amortized analysis:

1. $C_{incr}(bs) = t + 1$
   where
   $t = \text{length}\ (\text{takeWhile}\ (== I)\ bs)$

2. $A_{incr}(bs) = 2$

3. $S(bs) = \text{length}\ (\text{filter}\ (== I)\ bs)$

Given $bs' = \text{incr}\ bs$

$$C_{incr}(bs) \le A_{incr}(bs) + S(bs) - S(bs')$$

$$t + 1 \le 2 + b - b'$$

where $b' = b - t + 1$

$$\Leftrightarrow$$

$$\cancel{t} + \cancel{1} \le \cancel{2} + \cancel{b} - (\cancel{b} - \cancel{t} + \cancel{1})$$