hw03

BY ZIHAN ZHOU

1

2

2.1

$$\begin{cases} \mathrm{multTail}'(\mathrm{nil}, x) = x \\ \mathrm{multTail}'(n :: l, x) = (\mathrm{multTail}'(l, n \times x)) \end{cases}$$

$$\mathrm{multTail}(l) = \mathrm{multTail}'(l, 1)$$

2.2

2.3

2.3.1

$$\begin{cases} W_m([\,]) = k_0 \\ W_m(a :: l) = W_m(l) + k_1 \end{cases}$$

closed:

$$W_m(l) = \mathrm{len}(l) \times k_1 + k_0$$

it is $O(n)$

2.3.2

$W_T(l) = W_{T'}(l, 1)$

$$\begin{cases} W_{T'}([\,], 1) = k_0 \\ W_{T'}(a :: l, x) = W_{T'}(l, x - 1) + k_1 \end{cases}$$

$W_T(l) = \mathrm{len}(l) \times k_1 + k_0$

it is $O(n)$

2.3.3

tail is more effcient

2.4

Lemma 1. $\mathrm{multTail}'(l, x) \times a = \mathrm{multTail}'(l, x \times a)$

Proof. by list indution on l

2 case l is nil, l is not

bc: l=nil

ts: multTail'(nil,x)×a=multTail'(nil,x×a)

$\mathrm{multTail}(\mathrm{nil}, x) \times a = x \times a = \mathrm{multTail}'(\mathrm{nil}, x \times a)$

Ind ca: $l = b :: l'$

ts multTail'(b::l',x)×a=multTail'(b::l',x×a)

ih: $\mathrm{multTail}'(l', x') \times a' = \mathrm{multTail}'(l', x' \times a')$

$$\begin{array}{rcll} \mathrm{multTail}'(b :: l', x) \times a & = & \mathrm{multTail}'(l', b \times x) \times a & \text{by def} \\ & = & \mathrm{multTail}'(l', b \times x \times a) & \text{by ih} \\ & = & \mathrm{multTail}'(b :: l', x \times a) & \text{by def} \end{array}$$

□

Proposition 2.

for all $l \in \mathbb{L}_z$,

mult(l)=multTail(l)

Proof. by list induction on l.

there are two cases : l is nil or l is not.

base case: l = nil

to show : mult(nil)=multiTail(nil)

mult(nil)=nil=mutilTail'(nil,1)=mutilTail(nil) by def

Inductive case $l = a :: l'$

to show: mult(a::l')=multTail(a::l')

IH: mult(l')=multTail(l')

$$\begin{array}{rcll} \mathrm{mult}(a :: l') & = & \mathrm{mult}(l') \times a & \text{by def} \\ & = & \mathrm{multTail}(l') \times a & \text{by ih} \\ & = & \mathrm{multTail}(l', 1) \times a & \text{by def} \\ & = & \mathrm{multTail}'(l', a) & \text{by lemma 1} \\ & = & \mathrm{multTail}(a :: l') & \text{by def} \end{array}$$

□

2.5

$$\begin{cases} \mathrm{even}(\mathrm{nil}) = \mathrm{nil} \\ \mathrm{even}(a :: l) = (a \bmod 2) :: \mathrm{even}(l) \end{cases}$$

2.6

2.7

2.7.1

$$\begin{cases} W_{\mathrm{eT}'}(\mathrm{nil}, x) = k_0 \\ W_{\mathrm{eT}'}(a :: l, x) = W_{\mathrm{eT}'}(l, x \circ [a]) + (1 + \mathrm{len}(x))\, k_2 + k_1 \end{cases}$$

$$W_{\mathrm{eT}}(l) = W_{\mathrm{eT}'}(l, [])$$

close:

$$W_{\mathrm{eT}}(l) \quad = \quad k_0 + \mathrm{len}(l) \times k_1 + \frac{\mathrm{len}(l) \times (\mathrm{len}(l) + 1)}{2} \times k_2$$

$O(n^2)$

2.7.2

$$\begin{cases} W_e(\text{nil}) = k_0 \\ W_e(a :: l) = W_e(l) + k_1 \end{cases}$$

close

$$W_e(l) = k_0 + \text{len}(l) \times k_1$$

$O(n)$

2.8

2.9

Proposition 3.

for all l: int list

even l $\cong$ evenTail l

which is a special case k=nil for the following line

k $\circ$ even l $\cong$ evenTail'(l,k)

Proof. by list induction on l

thre are two cases: l is nil, l is a::l'.

base case l=nil:

ts: $k \circ \text{even nil} \cong \text{evenTail}'(\text{nil}, k)$

$k \circ \text{even nil} = k = \text{evenTail}\prime(\text{nil}, k)$

induitive case: $l = a :: l'$

ts: $k \circ$ even (a::l') = evenTail'(a::l',k)

IH: $k \circ$ even(l')=evenTail(l',k)

$$\begin{array}{lll} \text{evenTail}'(a :: l', k) & = & \text{evenTail}'(l', k \circ [(a \bmod 2)]) & \text{by def} \\ & = & k \circ [(a \bmod 2)] \circ \text{even}(l') & \text{by ih} \\ & = & k \circ ((a \bmod 2) :: \text{even}(l')) & \text{by list function lema} \\ & = & k \circ \text{even}(a :: l') & \text{by def} \end{array}$$

$\square$

2.10

we do not worry about mathematical property but only the definiton of the structure, in this case, list.

It is still induction, but with less concern.

2.11

$$\begin{cases} \text{treepod}(\text{Leaf}(z)) = z \\ \text{treepod}(\text{node}(x, y)) = \text{treepod}(x) \times \text{treepod}(y) \end{cases}$$

2.12

2.13

no because it has two recursive calls, before reaching leaf, there are nothing to accumulate for node.

3

3.1 ~ 3.7
3.8

`smSum,smSum_cert,smSum_check`

$$\begin{cases} W_s(i,[\,]) & = & k_0 \\ W_s(i,a::l) & = & W_s(i-a,l)+W_s(i,l)+k_1 \end{cases}$$

let n denote the length of input list. in the worst case,

$$W_s = 2^n\,k_1 + k_0$$

$$\begin{cases} S_s(i,[\,]) & = & k_0 \\ S_s(i,a::l) & = & \max\left(W_s(i-a,l),W_s(i,l)\right)+k_1 \end{cases}$$

let n denote the length of input list. in the worst case,

$$S_s = n\,k_1 + k_0$$

for `smSum_cert`, it just has larger k_1 , but essestially the same.

for `smSum_check`, it calls addtional `sum` function which is completely sequential in my implementation, in worst case it adds $n\,k_2$ to both work and span

$$W_s = 2^n\,k_1 + k_0 + n\,k_2$$

$$S_{\text{check}}(i,l) = n\,(k_1+k_2)+k_0$$

in general $O(S)=O(n)$, and $O(W)=2^n$

4