hw04

BY ZIHAN ZHOU

1

2

2.1 2.5

Proposition 1. *For all t: treeS, traverseS (simplify t)≅ traverseS t*

Proof. by structure induction on t

3 cases:

t = emptyS

t = leaf c

t = node(tl,tr)

case 1:

to show traverseS(emptyS) ≅ traverseS(simplify(emptyS))

both right and left side equals [].

case 2:

to show traverseS(leaf c) ≅ traverseS(simplify(leaf c))

both right and left side equals [c]

case 3:

to show: traverseS (simplify node(tl,tr))≅ traverseS node(tl,tr)

IH traverseS (simplify tl)≅ traverseS tl, traverseS (simplify tr)≅ traverseS tr

we have in code:

```
val nodeS(x,y) = nodeS(simplify(tl),simplify(tr))
```

subcase 1: if x is emptyS or y is emtpyS. without loss of generality let us assume x is emptyS

we have in code:

```
        if  not (x = emptyS) andalso not (y=emptyS )
        then nodeS(x,y)
        else simplify(nodeS(x,y))
```

so we have

$$
\begin{array}{rcll}
\text{traverseS}(\text{simplify node}(\text{tl},\text{tr})) & = & \text{traverseS}(\text{simplify node}(\text{simplify tl},\text{simplify tr})) & \text{by code} \\
 & = & \text{traverseS}(\text{simplify}(\text{tr})) & \text{by def of sim} \\
 & = & \text{traverseS}(\text{tr}) & \text{by IH} \\
 & = & \text{traverseS}(\text{nodeS}(\text{emptyS},\text{tr})) & \text{by def of trav} \\
 & = & \text{traverseS}(\text{nodeS}(\text{tl},\text{tr})) & \text{by IH}
\end{array}
$$

subcase 2: if neither x nor y is emptyS.

$$\begin{array}{rcll} \text{traverseS(simplify node(tl, tr))} & = & \text{traverseS(node(simplify tl, simplify tr))} & \text{by code} \\ & = & \text{traverseS(simplify } t1) \circ \text{traverseS(simplify } t2) & \text{by def of tra} \\ & = & \text{traverseS}(t1) \circ \text{traverseS}(t2) & \text{by IH} \\ & = & \text{traverseS(node}(t1, t2)) & \text{by def of trav} \end{array}$$

□

3

3.1 3.3

Lemma 2.

for all treeS ts,

if canonical(ts) and ts != emptyS,

then traverseS(ts)==traverseC(T(convertCan'(ts)))

Proof. by structural induction on ts.

2 cases:

t = leafS c

t = nodeS(tl,tr)

we know that leafS c is always canonical

case 1:

traverseS(leafS c)==traverseC(T(convertCan'(leafS c)))

both right and left side equals [c]

case 2:

to show

if canonical(nodeS(x,y)), traverseS(nodeS(x,y))==traverseC(T(convertCan'(nodeS(x,y))))

since canonical(nodeS(x,y)), so by def of canonical that canonical(x) and canonical(y), and x!=emptyS, y!=emptyS

IH2: traverseS(x)==traverseC(T(convertCan'(x))),

IH1: traverseS(y)==traverseC(T(convertCan'(y)))

$$\begin{array}{cll} & \text{traverseC}(T(\text{convertCan}'(\text{nodeS}(x, y)))) & \text{by} \\ = & \text{traverseC}(T(\text{nodeC}(\text{convertCan}'(x), \text{convertCan}'(y)))) & \text{by} \quad \text{def of conC}' \\ = & \text{traverseC}(T(\text{convertCan}'(x))) \circ \text{traverseC}(T(\text{convertCan}'(y))) & \text{by} \quad \text{def of traC} \\ = & \text{traverseS}(x) \circ \text{traverseS}(y) & \text{by} \quad \text{IH} \\ = & \text{traverseS}(\text{nodeS}(x, y)) & \text{by} \quad \text{def of traS} \end{array}$$

□

Proposition 3.

for all treeS ts, if canonical(ts), then traverseS(ts)==traverseC(convertCan(ts))

Proof.

2 cases:

t = emptyS

t is not emptyS.

we know that emptyS is always canonical

case 1:

to show traverseS(emptyS)==traverseC(convertCan(emptyS))

both right and left side equals [].

case 2:

to show if canonical(t) and t != emtpyS, traverseS(t)==traverseC(convertCan(t))

$$\begin{aligned} \text{traverseC}(\text{convertCan}(t)) &= \text{traverseC}(T(\text{converCan}'(t))) && \text{by def of conC} \\ &= \text{traverseS}(t) && \text{by lemma 2} \end{aligned}$$

□

3.2 3.6

Proposition 4.

for any treeS t, convertSloppy(t) = t', where t' is a treeC, and traverseS(t)=traverseC(t')

Proof.

$$\begin{aligned} \text{traverseC}(t') &= \text{traverseC}(\text{convertSloppy}(t)) \\ &= \text{traverseC}(\text{convertCan.safe}(\text{simplify.safe}(t))) \end{aligned}$$

because we know that simplify and converCan preserve the property of traverse

so

$\text{traverseC}(t') = \text{traverseS}(t)$ □

4

4.1 work

4.1.1 splitN

for function splitN, every step it either terminates or reduce one height,

$W_{\text{splitN}}(d) = W_{\text{splitN}}(d-1) + k_1 + W_{\text{size}}$

$W_{\text{splitN}}(0) = k_o$

$W_{\text{splitN}}(d) \in O(d)$

4.1.2 leftmost

leftmost just call splitN with the same d so

$W_{\text{leftmost}}(d) = W_{\text{splitN}}(d) + k_1 \in O(d)$

4.1.3 halves

for function halves , it called once splitN, once leftmost on the result of that

$W_{\text{halves}}(d) = W_{\text{splitN}}(d) + W_{\text{leftmost}}(d) + k_1 + W_{\text{size}}$

$W_{\text{halves}}(d) \in O(d)$

4.1.4 rebalance

$$\begin{aligned} W_{\text{rebalance}}(0) &= k_0 \\ W_{\text{rebalance}}(d) &= k_1 + W_{\text{halves}}(d) \times k_2 + 2 \times W_{\text{rebalance}}(d-1) \end{aligned}$$

$$\begin{aligned} W_{\text{rebalance}}(d) &= \sum_{i=1}^{d} (k_1 + i \times k_2) \times 2^{d-i} \\ &\leqslant k_3 \times \sum_{i=1}^{d} (2^{d-i} \times i) \\ &= d + 2\,(d-1) + 2^2\,(d-2) + \ldots + 2^{d-2} \times 2 + 2^{d-1} \\ &= (1 + 2 + \ldots + 2^{d-4} + 2^{d-3} + 2^{d-2} + 2^{d-1}) \\ &+ (1 + 2 + \ldots + 2^{d-4} + 2^{d-3} + 2^{d-2}) \\ &+ (1 + 2 + \ldots + 2^{d-4} + 2^{d-3}) \\ &+ (1 + 2 + \ldots + 2^{d-4}) \\ &\vdots \\ &+ 1 \\ &= \sum_{j=1}^{d-1} \sum_{i=0}^{j-1} 2^i \\ &= \sum_{j=1}^{d-1} 2^j \\ &= 2^d - 2 \end{aligned}$$

$W_{\text{rebalance}}(0) = k_0$

$W_{\text{rebalance}}(n) = k_1 + W_{\text{halves}}(\log n) \times k_2 + 2 \times W_{\text{rebalance}}\left(\frac{n}{2}\right) \times k_4$

$$\begin{aligned} W_r(n) &\leqslant k_1 + k_3 \times \log n + 2 \times W_r(n/2) \times k_4 \\ &= \sum_{i=1}^{\log n} (k_1 + i\,k_3) \times 2^{\log n - i} \\ &\leqslant k_5 \times \sum_{i=1}^{\log n} \left(\log \frac{2^{\log n}}{2^{\log n - i}} \times 2^{\log n - i} \right) \\ &= k_5 \times \sum_{j=0}^{\log n} \left(\log \frac{n}{2^j} \times 2^j \right) \\ &= B3 \end{aligned}$$

$O(n)$

4.1.5

if it is roughly balance, then we can regard halves as almost constant time

$W_{\text{rebalance}}(0) = k_0$

$W_{\text{rebalance}}(n) = k_1 + W_{\text{halves}}(k_3) \times k_2 + 2 \times W_{\text{rebalance}}\left(\frac{n}{2}\right) \times k_4$

then

$W_r(n) = k_k + W(n/2) \times 2 = \sum_{i=0}^{\log n} 2^i\,k_k \in O(n)$

4.2 span

4.2.1

$S_{\text{splitN}}(d) = S_{\text{splitN}}(d-1) + k_1 + S_{\text{size}}$

$S_{\text{splitN}}(0) = k_o$

$S_{\text{splitN}}(d) \in O(d)$

4.2.2

leftmost just call splitN with the same d so

$S_{\text{leftmost}}(d) = S_{\text{splitN}}(d) + k_1 \in O(d)$

4.2.3

for function halves , it called once splitN, once leftmost on the result of that

$S_{\text{halves}}(d) = S_{\text{splitN}}(d) + S_{\text{leftmost}}(d) + k_1 + S_{\text{size}}$

$S_{\text{halves}}(d) \in O(d)$

4.2.4

$S_{\text{rebalance}}(0) = k_0$

$S_{\text{rebalance}}(n) = k_1 + S_{\text{halves}}(\log n) \times k_2 + S_{\text{rebalance}}\left(\frac{n}{2}\right) \times k_4$

$S_r(n) = \sum_{i=0}^{\log n} \log n \times k_k \in O((\log n)^2)$

4.2.5

$S_{\text{rebalance}}(0) = k_0$

$S_{\text{rebalance}}(n) = k_1 + S_{\text{halves}}(k_v) \times k_2 + S_{\text{rebalance}}\left(\frac{n}{2}\right) \times k_4$

$S_r(n) = \sum_{i=0}^{\log n} k_k \in O(\log n)$