Final Exam

Functional Data Structures

27. 7. 2020

Proof Guidelines: We expect valid Isabelle proof scripts to be submitted as a solution to the questions of this exam. Major proof steps, especially inductions, need to be stated explicitly. The use of "sorry" may lead to the deduction of points but is preferable to spending a lot of time on individual proof steps.

Please submit your solution via the submission system and ALSO via email to mansour@in.tum.de. You have to include your final answer to all questions in one email. If you do not submit the exam by the deadline (10 minutes after the official end of the exam) either via the submission system or via email you will have failed the exam (X = no show = 5,0). The solutions you send by email are primarily intended as a backup in case of technical problems. Unless you state explicitly that we should grade the email submission (and it was submitted in time), we will grade the submission on the submission system.

In the unlikely event that you discover a mistake in one of the questions you should communicate this to nipkow@in.tum.de and mansour@in.tum.de. We will communicate any corrections to the exam questions by email to you. Thus you do need to watch your email at regular intervals.

Also, if you encounter unforeseen technical problems during the exam, you can send an e-mail to nipkow@in.tum.de and mansour@in.tum.de.

1 Induction

1.1 Solution

abbreviation *a* **where**
"a ≡ 2::nat"

abbreviation *b* **where**
"b ≡ 1::nat"

lemma *"lvs t = a * sz t + b"*
by (*induction t*) *simp+*

1.1.1 Solution

lemma *"lvs t = a * sz t + b"*
proof (*induction t*)
case *Lf*
have *"lvs Lf = 1"* **by** (*simp only*: *lvs.simps*)
also have *"... = 2 * sz Lf + 1"* **by** (*simp only*: *sz.simps*)
finally show *?case* **.**
next
case (*Nd r s t*)
have *"lvs (Nd r s t) = lvs r + lvs s + lvs t"*
by (*simp only*: *lvs.simps*)
also have *"... = 2 * sz r + 1 + 2 * sz s + 1 + 2 * sz t + 1"*
by (*simp only*: *Nd.IH*)
also have *"... = 2 * (sz r + sz s + sz t + 1) + 1"*
by (*simp only*: *algebra_simps*)
also have *"... = 2 * sz (Nd r s t) + 1"*
by (*simp only*: *sz.simps*)
finally show *?case* **.**
qed

2 Trees with Same Structure

2.1 Solution

fun *same* :: *"'a tree ⇒ 'b tree ⇒ bool"* **where**
"same Leaf Leaf = True" |
"same (Node l a r) (Node l' a' r') = (same l l' ∧ same r r')" |
"same _ _ = False"

lemma *"same t t'*
⟹ same (insert_pq x t) (insert_pq y t')"
proof (*induction t t' arbitrary*: *x y rule*: *same.induct*)
case (*2 l a r l' a' r'*)

Induction hypothesis, generalized over x,y!

have
IH1: *"same (insert_pq x l) (insert_pq y l')"* **and**
IH2: *"same (insert_pq x r) (insert_pq y r')"* **for** *x y*
using *2* **by** *auto*

assume *PREM*: *"same (Node l a r) (Node l' a' r')"*

show *"same (insert_pq x (Node l a r)) (insert_pq y (Node l' a' r'))"*
proof −

We get four cases

consider ‹*x*<*a* ∧ *y*<*a'*› | ‹*x*<*a* ∧ *y*≥*a'*› | ‹*x*≥*a* ∧ *y*<*a'*› | ‹*x*≥*a* ∧ *y*≥*a'*›
by *force*
then show *?thesis* **proof** *cases*
case *1*
hence
"insert_pq x (Node l a r) = Node (insert_pq a r) x l"
"(insert_pq y (Node l' a' r')) = Node (insert_pq a' r') y l'"
by *simp_all*
moreover from *PREM* **have** *"same l l'"* **by** *simp*
moreover note *IH2*
ultimately show *?thesis* **by** *simp*

The other cases are analogous

qed (*insert IH1 IH2 PREM*; *auto*)+
qed
qed *auto*

The other cases are simple

3 2-3-4 Trees

3.1 Solution

datatype *'a upI = TI "'a tree234" | OF "'a tree234" 'a "'a tree234"*

fun *height_upI :: "'a upI ⇒ nat"* **where**
"height (TI t) = height t" |
"height (OF l a r) = height l"

fun *treeI :: "'a upI ⇒ 'a tree234"* **where**
"treeI (TI t) = t" |
"treeI (OF l a r) = Node2 l a r"

fun *joinL* **where**
"joinL l x Leaf = OF l x tree234.Leaf" |
"joinL l x (Node2 t1 v t2) =
(if height l = height (Node2 t1 v t2) then
OF l x (Node2 t1 v t2)
else case joinL l x t1 of
TI t ⇒ TI (Node2 t v t2)
| OF l' x' r' ⇒ TI (Node3 l' x' r' v t2))" |
"joinL l x (Node3 t1 v1 t2 v2 t3) =
(if height l = height (Node3 t1 v1 t2 v2 t3) then
OF l x (Node3 t1 v1 t2 v2 t3)
else case joinL l x t1 of
TI t ⇒ TI (Node3 t v1 t2 v2 t3)
| OF l' x' r' ⇒ TI (Node4 l' x' r' v1 t2 v2 t3))" |
"joinL l x (Node4 t1 v1 t2 v2 t3 v3 t4) =
(if height l = height (Node4 t1 v1 t2 v2 t3 v3 t4) then
OF l x (Node4 t1 v1 t2 v2 t3 v3 t4)
else case joinL l x t1 of
TI t ⇒ TI (Node4 t v1 t2 v2 t3 v3 t4)
| OF l' x' r' ⇒ OF (Node2 l' x' r') v1 (Node3 t2 v2 t3 v3 t4))"

fun *joinR* **where**
"joinR Leaf x r = OF tree234.Leaf x r" |
"joinR (Node2 t1 v t2) x r =
(if height (Node2 t1 v t2) = height r then
OF (Node2 t1 v t2) x r
else case joinR t2 x r of
TI t ⇒ TI (Node2 t1 v t)
| OF l' x' r' ⇒ TI (Node3 t1 v l' x' r'))" |
"joinR (Node3 t1 v1 t2 v2 t3) x r =
(if height (Node3 t1 v1 t2 v2 t3) = height r then

```
                                OF (Node3 t1 v1 t2 v2 t3) x r
                               else case joinR t3 x r of
                                 TI t ⇒ TI (Node3 t1 v1 t2 v2 t)
                               | OF l' x' r' ⇒ TI (Node4 t1 v1 t2 v2 l' x' r'))" |
  "joinR (Node4 t1 v1 t2 v2 t3 v3 t4) x r=
                              (if height (Node4 t1 v1 t2 v2 t3 v3 t4) = height r then
                                 OF (Node4 t1 v1 t2 v2 t3 v3 t4) x r
                               else case joinR t4 x r of
                                 TI t ⇒ TI (Node4 t1 v1 t2 v2 t3 v3 t)
                               | OF l' x' r' ⇒ OF (Node3 t1 v1 t2 v2 t3) v3 (Node2 l' x' r'))"

fun join::"'a tree234 ⇒ 'a ⇒ 'a tree234 ⇒ 'a tree234" where
  "join l x r =
     (if height l ≤ height r then
        treeI (joinL l x r)
      else treeI (joinR l x r))"

fun split :: "'a::linorder tree234 ⇒ 'a ⇒ 'a tree234 × bool × 'a tree234" where
"split Leaf k = (Leaf, False, Leaf)" |
"split (Node2 l a r) k =
  (if k < a then let (l1,b,l2) = split l k in (l1, b, join l2 a r) else
   if a < k then let (r1,b,r2) = split r k in (join l a r1, b, r2)
   else (l, True, r))" |
"split (Node3 t1 v1 t2 v2 t3) k =
  (if k < v1 then let (l1,b,l2) = split t1 k in (l1, b, join l2 v1 (Node2 t2 v2 t3)) else
   if k = v1 then (t1, True, Node2 t2 v2 t3) else
   if k < v2 then let (l1,b,l2) = split t2 k in (join t1 v1 l1, b, join l2 v2 t3) else
   if v2 < k then let (l1,b,l2) = split t3 k in (join (Node2 t1 v1 t2) v2 l1, b, l2) else
   (Node2 t1 v1 t2, True, t3))" |
"split (Node4 t1 v1 t2 v2 t3 v3 t4) k =
  (if k < v1 then let (l1,b,l2) = split t1 k in (l1, b, join l2 v1 (Node3 t2 v2 t3 v3 t4)) else
   if k = v1 then (t1, True, Node3 t2 v2 t3 v3 t4) else
   if k < v2 then let (l1,b,l2) = split t2 k in (join t1 v1 l1, b, join l2 v2 (Node2 t3 v3 t4)) else
   if k = v2 then (Node2 t1 v1 t2, True, Node2 t3 v3 t4) else
   if k < v3 then let (l1,b,l2) = split t3 k in (join (Node2 t1 v1 t2) v2 l1, b, join l2 v3 t4) else
   if v3 < k then let (l1,b,l2) = split t4 k in (join (Node3 t1 v1 t2 v2 t3) v3 l1, b, l2) else
   (Node3 t1 v1 t2 v2 t3, True, t4))"

definition insert :: "'a::linorder ⇒ 'a tree234 ⇒ 'a tree234" where
  "insert k t = (let (l,_,r) = split t k in join l k r)"
```

4 Amortized Complexity

4.1 Solution

fun Φ :: *"'a stk* $\Rightarrow$ *nat"* **where**
"Φ *(xs,ys) = length xs + length ys"*

lemma *"t_push x (xs,ys) +* Φ*(push x (xs,ys)) −* Φ *(xs,ys) = 2"*
by *simp*

lemma *"t_pop n (xs,ys) +* Φ*(pop n (xs,ys)) −* Φ *(xs,ys) = 0"*
by *simp*