Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Peter Lammich

SS 2017
14. 7. 2017

Functional Data Structures

Exercise Sheet 12

Exercise 12.1 Balanced Queues

Consider locale *Queue* in file *Thys/Amortized_Examples*. A call of *deq* (*xs*,[]) requires the reversal of *xs*, which may be very long. We can reduce that impact by shifting *xs* over to *ys* whenever *length xs* > *length ys*. This does not improve the amortized complexity (in fact it increases it by 1) but reduces the worst case complexity of individual calls of *deq*. Modify locale *Queue* as follows:

```
locale Queue begin

type_synonym 'a queue = "'a list * 'a list"

definition "init = ([],[])"
fun balance :: "'a queue ⇒ 'a queue" where
"balance(xs,ys) = (if size xs ≤ size ys then (xs,ys) else ([], ys @ rev xs))"
fun enq :: "'a ⇒ 'a queue ⇒ 'a queue" where
"enq a (xs,ys) = balance (a#xs, ys)"
fun deq :: "'a queue ⇒ 'a queue" where
"deq (xs,ys) = balance (xs, tl ys)"
```

Again, we count only the newly allocated list cells.

```
fun t_balance :: "'a queue ⇒ nat" where
  "t_balance (xs,ys) = (if size xs ≤ size ys then 0 else size xs + size ys)"
fun t_enq :: "'a ⇒ 'a queue ⇒ nat" where
"t_enq a (xs,ys) = 1 + t_balance (a#xs, ys)"
fun t_deq :: "'a queue ⇒ nat" where
"t_deq (xs,ys) = t_balance (xs, tl ys)"
```

- Start over with showing functional correctness. Hint: You will require an invariant.

```
fun invar :: "'a queue ⇒ bool"
fun abs :: "'a queue ⇒ 'a list"
lemma [simp]: "invar init"
lemma [simp]: "invar q ⟹ invar (enq x q)"
lemma [simp]: "invar q ⟹ invar (deq q)"
lemma [simp]: "abs init = []"
lemma [simp]: "invar q ⟹ abs (enq x q) = x#abs q"
lemma [simp]: "invar q ⟹ abs (deq q) = butlast (abs q)"
```

- Next, define a suitable potential function Φ, and prove that the amortized complexity of *enq* is ≤ 3 and of *deq* is ≤ 0.

fun Φ :: *"'a queue ⇒ int"*
lemma *Φ_non_neg*: *"Φ t ≥ 0"*
lemma *Φ_init*: *"Φ init = 0"*
lemma *a_enq*: *" t_enq a q + Φ(enq a q) − Φ q ≤ 3"*
lemma *a_deq*: *" t_deq q + Φ(deq q) − Φ q ≤ 0"*

Finally, show that a sequence of enqueue and dequeue operations requires linear cost in its length:

datatype *'a opr = ENQ 'a | DEQ*

fun *execute* :: *"'a queue ⇒ 'a opr list ⇒ 'a queue"*
  where
  "execute q [] = q"
| *"execute q (ENQ x#ops) = execute (enq x q) ops"*
| *"execute q (DEQ#ops) = execute (deq q) ops"*

Count only list cell allocations!

fun *t_execute* :: *"'a queue ⇒ 'a opr list ⇒ nat"*

lemma *t_execute*: *"t_execute init ops ≤ 3∗length ops"*

Homework 12 Dynamic Tables with real-valued Potential

Submission until Friday, 21. 7. 2017, 11:59am.

In file *Thys/Amortized_Examples* in the repository there is a formalization of dynamic tables in locale *Dyn_Tab* with the potential function Φ $(n,l) = 2*n - l$ and a discussion of why this is tricky. The standard definition you find in textbooks does not rely on cut-off subtraction on *nat* but uses standard real numbers:

type_synonym *tab = "nat × nat"*

fun Φ :: *"tab ⇒ real"* **where**
"Φ (n,l) = 2∗(real n) − real l"

Start with the locale *Dyn_Tab* in file *Thys/Amortized_Examples* but use the above definition of Φ :: *tab ⇒ real*. A number of proofs will now fail because the invariant is now too weak. Find a stronger invariant such that all the proofs work again.