

# Amortized Complexity Verified

Tobias Nipkow

May 26, 2024

## Abstract

A framework for the analysis of the amortized complexity of (functional) data structures is formalized in Isabelle/HOL and applied to a number of standard examples and to the following non-trivial ones: skew heaps, splay trees, splay heaps and pairing heaps. This work is described in [4] (except for pairing heaps). An extended version (including pairing heaps) is available online [5].

## Contents

<b>1</b>	<b>Amortized Complexity (Unary Operations)</b>	<b>3</b>
1.1	Binary Counter . . . . .	4
1.2	Dynamic tables: insert only . . . . .	5
1.3	Stack with multipop . . . . .	8
1.4	Queue . . . . .	9
1.5	Dynamic tables: insert and delete . . . . .	10
<b>2</b>	<b>Amortized Complexity Framework</b>	<b>12</b>
<b>3</b>	<b>Simple Examples</b>	<b>14</b>
3.1	Binary Counter . . . . .	14
3.2	Stack with multipop . . . . .	15
3.3	Dynamic tables: insert only . . . . .	16
3.4	Dynamic tables: insert and delete . . . . .	20
3.5	Queue . . . . .	21
<b>4</b>	<b>Skew Heap Analysis</b>	<b>24</b>
<b>5</b>	<b>Splay Tree</b>	<b>30</b>
5.1	Basics . . . . .	30
5.2	Splay Tree Analysis . . . . .	33
5.3	Splay Tree Analysis (Optimal) . . . . .	43
<b>6</b>	<b>Splay Heap</b>	<b>55</b>

<b>7</b>	<b>Pairing Heaps</b>	<b>60</b>
7.1	Binary Tree Representation . . . . .	60
<b>8</b>	<b>Pairing Heaps</b>	<b>66</b>
8.1	Binary Tree Representation . . . . .	66
8.2	Okasaki's Pairing Heap . . . . .	72
8.3	Transfer of Tree Analysis to List Representation . . . . .	80
8.4	Okasaki's Pairing Heap (Modified) . . . . .	83

# 1 Amortized Complexity (Unary Operations)

```
theory Amortized_Framework0
imports Complex_Main
begin
```

This theory provides a simple amortized analysis framework where all operations act on a single data type, i.e. no union-like operations. This is the basis of the ITP 2015 paper by Nipkow. Although it is superseded by the model in *Amortized\_Framework* that allows arbitrarily many parameters, it is still of interest because of its simplicity.

```
locale Amortized =
fixes init :: 's
fixes next :: 'o  $\Rightarrow$  's  $\Rightarrow$  's
fixes inv :: 's  $\Rightarrow$  bool
fixes T :: 'o  $\Rightarrow$  's  $\Rightarrow$  real
fixes  $\Phi$  :: 's  $\Rightarrow$  real
fixes U :: 'o  $\Rightarrow$  's  $\Rightarrow$  real
assumes inv_init: inv init
assumes inv_next: inv s  $\implies$  inv(next f s)
assumes p0s: inv s  $\implies$   $\Phi s \geq 0$ 
assumes p0:  $\Phi init = 0$ 
assumes U: inv s  $\implies$   $T f s + \Phi(next f s) - \Phi s \leq U f s$ 
begin
```

```
fun state :: (nat  $\Rightarrow$  'o)  $\Rightarrow$  nat  $\Rightarrow$  's where
state f 0 = init |
state f (Suc n) = next (f n) (state f n)
```

```
lemma inv_state: inv(state f n)
by(induction n)(simp_all add: inv_init inv_next)
```

```
definition A :: (nat  $\Rightarrow$  'o)  $\Rightarrow$  nat  $\Rightarrow$  real where
A f i =  $T (f i) (state f i) + \Phi(state f (i+1)) - \Phi(state f i)$ 
```

```
lemma aeq:  $(\sum i < n. T (f i) (state f i)) = (\sum i < n. A f i) - \Phi(state f n)$ 
apply(induction n)
apply (simp add: p0)
apply (simp add: A_def)
done
```

```
corollary TA:  $(\sum i < n. T (f i) (state f i)) \leq (\sum i < n. A f i)$ 
by (metis add.commute aeq diff_add_cancel le_add_same_cancel2 p0s[OF inv_state])
```

**lemma aa1:**  $A f i \leq U (f i) (state f i)$   
**by**(simp add: A\_def U inv\_state)

**lemma ub:**  $(\sum i < n. T (f i) (state f i)) \leq (\sum i < n. U (f i) (state f i))$   
**by** (metis (mono\_tags) aa1 order.trans sum\_mono TA)

**end**

## 1.1 Binary Counter

**locale** BinCounter  
**begin**

**fun** incr **where**  
 incr [] = [True] |  
 incr (False#bs) = True # bs |  
 incr (True#bs) = False # incr bs

**fun** T\_incr :: bool list  $\Rightarrow$  real **where**  
 T\_incr [] = 1 |  
 T\_incr (False#bs) = 1 |  
 T\_incr (True#bs) = T\_incr bs + 1

**definition**  $\Phi$  :: bool list  $\Rightarrow$  real **where**  
 $\Phi$  bs = length(filter id bs)

**lemma** A\_incr:  $T\_incr\ bs + \Phi(incr\ bs) - \Phi\ bs = 2$   
**apply**(induction bs rule: incr.induct)  
**apply** (simp\_all add:  $\Phi\_def$ )  
**done**

**interpretation** incr: Amortized  
**where** init = [] **and** nxt = %\_. incr **and** inv =  $\lambda$ \_. True  
**and** T =  $\lambda$ \_. T\_incr **and**  $\Phi$  =  $\Phi$  **and** U =  $\lambda$ \_. 2  
**proof** (standard, goal\_cases)  
**case** 1 **show** ?case **by** simp  
**next**  
**case** 2 **show** ?case **by** simp  
**next**  
**case** 3 **show** ?case **by**(simp add:  $\Phi\_def$ )  
**next**  
**case** 4 **show** ?case **by**(simp add:  $\Phi\_def$ )  
**next**

```

  case 5 show ?case by(simp add: A_incr)
qed

```

```

thm incr_ub

```

```

end

```

## 1.2 Dynamic tables: insert only

```

locale DynTable1

```

```

begin

```

```

fun ins :: nat*nat ⇒ nat*nat where
ins (n,l) = (n+1, if n<l then l else if l=0 then 1 else 2*l)

```

```

fun T_ins :: nat*nat ⇒ real where
T_ins (n,l) = (if n<l then 1 else n+1)

```

```

fun invar :: nat*nat ⇒ bool where
invar (n,l) = (l/2 ≤ n ∧ n ≤ l)

```

```

fun Φ :: nat*nat ⇒ real where
Φ (n,l) = 2*(real n) - l

```

```

interpretation ins: Amortized
where init = (0::nat,0::nat)
and nxt = λ_. ins
and inv = invar
and T = λ_. T_ins and Φ = Φ and U = λ_ _. 3
proof (standard, goal_cases)
  case 1 show ?case by auto
next
  case (2 s) thus ?case by(cases s) auto
next
  case (3 s) thus ?case by(cases s)(simp split: if_splits)
next
  case 4 show ?case by(simp)
next
  case (5 s) thus ?case by(cases s) auto
qed

```

```

end

```

```

locale table_insert = DynTable1 +

```

```

fixes a :: real
fixes c :: real
assumes c1[arith]: c > 1
assumes ac2: a ≥ c/(c - 1)
begin

lemma ac: a ≥ 1/(c - 1)
using ac2 by(simp add: field_simps)

lemma a0[arith]: a > 0
proof-
  have 1/(c - 1) > 0 using ac by simp
  thus ?thesis by (metis ac dual_order.strict_trans1)
qed

definition b = 1/(c - 1)

lemma b0[arith]: b > 0
using ac by (simp add: b_def)

fun ins :: nat * nat ⇒ nat * nat where
ins(n,l) = (n+1, if n<l then l else if l=0 then 1 else nat(ceiling(c*l)))

fun pins :: nat * nat => real where
pins(n,l) = a*n - b*l

interpretation ins: Amortized
where init = (0,0) and next = %_. ins
and inv = λ(n,l). if l=0 then n=0 else n ≤ l ∧ (b/a)*l ≤ n
and T = λ_. T_ins and Φ = pins and U = λ_ _ . a + 1
proof (standard, goal_cases)
  case 1 show ?case by auto
next
  case (2 s)
  show ?case
  proof (cases s)
    case [simp]: (Pair n l)
    show ?thesis
    proof cases
      assume l=0 thus ?thesis using 2 ac
      by (simp add: b_def field_simps)
    next
    assume l≠0
    show ?thesis

```

```

proof cases
  assume  $n < l$ 
  thus ?thesis using 2 by(simp add: algebra_simps)
next
  assume  $\neg n < l$ 
  hence [simp]:  $n = l$  using 2  $\langle l \neq 0 \rangle$  by simp
  have 1:  $(b/a) * \text{ceiling}(c * l) \leq \text{real } l + 1$ 
  proof-
    have  $(b/a) * \text{ceiling}(c * l) = \text{ceiling}(c * l) / (a * (c - 1))$ 
      by(simp add: b_def)
    also have  $\text{ceiling}(c * l) \leq c * l + 1$  by simp
    also have  $\dots \leq c * (\text{real } l + 1)$  by (simp add: algebra_simps)
    also have  $\dots / (a * (c - 1)) = (c / (a * (c - 1))) * (\text{real } l + 1)$  by
simp
  also have  $c / (a * (c - 1)) \leq 1$  using ac2 by (simp add: field_simps)
  finally show ?thesis by (simp add: divide_right_mono)
qed
  have 2:  $\text{real } l + 1 \leq \text{ceiling}(c * \text{real } l)$ 
  proof-
    have  $\text{real } l + 1 = \text{of\_int}(\text{int}(l)) + 1$  by simp
    also have  $\dots \leq \text{ceiling}(c * \text{real } l)$  using  $\langle l \neq 0 \rangle$ 
      by(simp only: int_less_real_le[symmetric] less_ceiling_iff)
      (simp add: mult_less_cancel_right1)
    finally show ?thesis .
  qed
  from  $\langle l \neq 0 \rangle$  1 2 show ?thesis by simp
qed
qed
qed
qed
next
  case (3 s) thus ?case by(cases s)(simp add: field_simps split: if_splits)
next
  case 4 show ?case by(simp)
next
  case (5 s)
  show ?case
  proof (cases s)
    case [simp]: (Pair n l)
    show ?thesis
  proof cases
    assume  $l = 0$  thus ?thesis using 5 by (simp)
  next
    assume [arith]:  $l \neq 0$ 
    show ?thesis

```

```

proof cases
  assume  $n < l$ 
  thus ?thesis using 5 ac by(simp add: algebra_simps b_def)
next
  assume  $\neg n < l$ 
  hence [simp]:  $n = l$  using 5 by simp
  have  $T\_ins\ s + pins\ (ins\ s) - pins\ s = l + a + 1 + (-\ b * ceiling(c * l))$ 
+  $b * l$ 
    using  $\langle l \neq 0 \rangle$ 
    by(simp add: algebra_simps less_trans[of  $-1::real\ 0$ ])
    also have  $-b * ceiling(c * l) \leq -b * (c * l)$  by (simp add: ceiling_correct)
    also have  $l + a + 1 + -b * (c * l) + b * l = a + 1 + l * (1 - b * (c - 1))$ 
    by (simp add: algebra_simps)
    also have  $b * (c - 1) = 1$  by(simp add: b_def)
    also have  $a + 1 + (real\ l) * (1 - 1) = a + 1$  by simp
    finally show ?thesis by simp
  qed
qed
qed
qed

```

**thm** ins\_ub

**end**

### 1.3 Stack with multipop

**datatype** 'a op\_stk = Push 'a | Pop nat

**fun** next\_stk :: 'a op\_stk  $\Rightarrow$  'a list  $\Rightarrow$  'a list **where**  
 next\_stk (Push x) xs = x # xs |  
 next\_stk (Pop n) xs = drop n xs

**fun** T\_stk :: 'a op\_stk  $\Rightarrow$  'a list  $\Rightarrow$  real **where**  
 T\_stk (Push x) xs = 1 |  
 T\_stk (Pop n) xs = min n (length xs)

**interpretation** stack: Amortized

**where** init = [] **and** next = next\_stk **and** inv =  $\lambda\_.$  True  
**and** T = T\_stk **and**  $\Phi$  = length **and** U =  $\lambda f\ \_.$  case f of Push \_  $\Rightarrow$  2 |  
 Pop \_  $\Rightarrow$  0



```

proof (standard, goal_cases)
  case 1 show ?case by auto
next
  case (2 s) thus ?case by(cases s) auto
next
  case 3 thus ?case by simp
next
  case 4 show ?case by(simp)
next
  case (5 _ f) thus ?case by (cases f) auto
qed

```

## 1.4 Queue

See, for example, the book by Okasaki [6].

```

datatype 'a opq = Enq 'a | Deq

```

```

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

```

```

fun nextq :: 'a opq ⇒ 'a queue ⇒ 'a queue where
nextq (Enq x) (xs,ys) = (x#xs,ys) |
nextq Deq (xs,ys) = (if ys = [] then ([], tl(rev xs)) else (xs,tl ys))

```

```

fun Tq :: 'a opq ⇒ 'a queue ⇒ real where
Tq (Enq x) (xs,ys) = 1 |
Tq Deq (xs,ys) = (if ys = [] then length xs else 0)

```

**interpretation** queue: Amortized

**where** *init* = ([],[]) **and** *next* = next<sub>q</sub> **and** *inv* = λ\_. True

**and** *T* = T<sub>q</sub> **and** *Φ* = λ(xs,ys). length xs **and** *U* = λf\_. case f of Enq  
 \_ ⇒ 2 | Deq ⇒ 0

```

proof (standard, goal_cases)
  case 1 show ?case by auto
next
  case (2 s) thus ?case by(cases s) auto
next
  case (3 s) thus ?case by(cases s) auto
next
  case 4 show ?case by(simp)
next
  case (5 s f) thus ?case
    apply(cases s)
    apply(cases f)

```

```

    by auto
qed

```

```

fun balance :: 'a queue  $\Rightarrow$  'a queue where
balance(xs,ys) = (if size xs  $\leq$  size ys then (xs,ys) else ([], ys @ rev xs))

```

```

fun next_q2 :: 'a opq  $\Rightarrow$  'a queue  $\Rightarrow$  'a queue where
next_q2 (Enq a) (xs,ys) = balance (a#xs,ys) |
next_q2 Deq (xs,ys) = balance (xs, tl ys)

```

```

fun T_q2 :: 'a opq  $\Rightarrow$  'a queue  $\Rightarrow$  real where
T_q2 (Enq _) (xs,ys) = 1 + (if size xs + 1  $\leq$  size ys then 0 else size xs
+ 1 + size ys) |
T_q2 Deq (xs,ys) = (if size xs  $\leq$  size ys - 1 then 0 else size xs + (size ys
- 1))

```

**interpretation** queue2: Amortized

**where** *init* = ([],[]) **and** *next* = next\_q2

**and** *inv* =  $\lambda(x,s,y,s). \text{size } xs \leq \text{size } ys$

**and** *T* = T\_q2 **and**  $\Phi = \lambda(x,s,y,s). 2 * \text{size } xs$

**and** *U* =  $\lambda f \_ . \text{case } f \text{ of } \text{Enq } \_ \Rightarrow 3 \mid \text{Deq} \Rightarrow 0$

**proof** (standard, goal\_cases)

  case 1 show ?case by auto

next

  case (2 s f) thus ?case by(cases s) (cases f, auto)

next

  case (3 s) thus ?case by(cases s) auto

next

  case 4 show ?case by(simp)

next

  case (5 s f) thus ?case

    apply(cases s)

    apply(cases f)

    by (auto simp: split: prod.splits)

qed

## 1.5 Dynamic tables: insert and delete

```

datatype optb = Ins | Del

```

```

locale DynTable2 = DynTable1

```

```

begin

```

**fun** *del* :: *nat\*nat* ⇒ *nat\*nat* **where**  
*del* (*n,l*) = (*n - 1*, if *n=1* then 0 else if 4\*(*n - 1*)<*l* then *l div 2* else *l*)

**fun** *T\_del* :: *nat\*nat* ⇒ *real* **where**  
*T\_del* (*n,l*) = (if *n=1* then 1 else if 4\*(*n - 1*)<*l* then *n* else 1)

**fun** *next\_tb* :: *optb* ⇒ *nat\*nat* ⇒ *nat\*nat* **where**  
*next\_tb* *Ins* = *ins* |  
*next\_tb* *Del* = *del*

**fun** *T\_tb* :: *optb* ⇒ *nat\*nat* ⇒ *real* **where**  
*T\_tb* *Ins* = *T\_ins* |  
*T\_tb* *Del* = *T\_del*

**fun** *invar* :: *nat\*nat* ⇒ *bool* **where**  
*invar* (*n,l*) = (*n ≤ l*)

**fun**  $\Phi$  :: *nat\*nat* ⇒ *real* **where**  
 $\Phi$  (*n,l*) = (if *n < l/2* then *l/2 - n* else 2\**n - l*)

**interpretation** *tb*: *Amortized*  
**where** *init* = (0,0) **and** *next* = *next\_tb*  
**and** *inv* = *invar*  
**and** *T* = *T\_tb* **and**  $\Phi$  =  $\Phi$   
**and** *U* =  $\lambda f \_.$  case *f* of *Ins* ⇒ 3 | *Del* ⇒ 2  
**proof** (*standard*, *goal\_cases*)  
  **case** 1 **show** ?*case* **by** *auto*  
**next**  
  **case** (2 *s f*) **thus** ?*case* **by**(*cases s*, *cases f*) (*auto*)  
**next**  
  **case** (3 *s*) **show** ?*case* **by**(*cases s*)(*simp*)  
**next**  
  **case** 4 **show** ?*case* **by**(*simp*)  
**next**  
  **case** (5 *s f*) **thus** ?*case* **apply**(*cases s*) **apply**(*cases f*)  
  **by** (*auto*)  
**qed**  
  
**end**  
  
**end**

## 2 Amortized Complexity Framework

```
theory Amortized_Framework
imports Complex_Main
begin
```

This theory provides a framework for amortized analysis.

```
datatype 'a rose_tree = T 'a 'a rose_tree list
```

```
declare length_Suc_conv [simp]
```

```
locale Amortized =
```

```
fixes arity :: 'op  $\Rightarrow$  nat
```

```
fixes exec :: 'op  $\Rightarrow$  's list  $\Rightarrow$  's
```

```
fixes inv :: 's  $\Rightarrow$  bool
```

```
fixes cost :: 'op  $\Rightarrow$  's list  $\Rightarrow$  nat
```

```
fixes  $\Phi$  :: 's  $\Rightarrow$  real
```

```
fixes U :: 'op  $\Rightarrow$  's list  $\Rightarrow$  real
```

```
assumes inv_exec:  $\llbracket \forall s \in \text{set } ss. \text{inv } s; \text{length } ss = \text{arity } f \rrbracket \Longrightarrow \text{inv}(\text{exec } f \text{ } ss)$ 
```

```
assumes ppos:  $\text{inv } s \Longrightarrow \Phi \text{ } s \geq 0$ 
```

```
assumes U:  $\llbracket \forall s \in \text{set } ss. \text{inv } s; \text{length } ss = \text{arity } f \rrbracket$ 
```

```
 $\Longrightarrow \text{cost } f \text{ } ss + \Phi(\text{exec } f \text{ } ss) - \text{sum\_list } (\text{map } \Phi \text{ } ss) \leq U \text{ } f \text{ } ss$ 
```

```
begin
```

```
fun wf :: 'op rose_tree  $\Rightarrow$  bool where
```

```
wf (T f ts) = (length ts = arity f  $\wedge$  ( $\forall t \in \text{set } ts. \text{wf } t$ ))
```

```
fun state :: 'op rose_tree  $\Rightarrow$  's where
```

```
state (T f ts) = exec f (map state ts)
```

```
lemma inv_state: wf ot  $\Longrightarrow \text{inv}(\text{state } ot)$ 
```

```
by(induction ot)(simp_all add: inv_exec)
```

```
definition acost :: 'op  $\Rightarrow$  's list  $\Rightarrow$  real where
```

```
acost f ss = cost f ss +  $\Phi$  (exec f ss) - sum_list (map  $\Phi$  ss)
```

```
fun acost_sum :: 'op rose_tree  $\Rightarrow$  real where
```

```
acost_sum (T f ts) = acost f (map state ts) + sum_list (map acost_sum ts)
```

```
fun cost_sum :: 'op rose_tree  $\Rightarrow$  real where
```

```
cost_sum (T f ts) = cost f (map state ts) + sum_list (map cost_sum ts)
```

```

fun U_sum :: 'op rose_tree  $\Rightarrow$  real where
  U_sum (T f ts) = U f (map state ts) + sum_list (map U_sum ts)

lemma t_sum_a_sum: wf ot  $\Longrightarrow$  cost_sum ot = acost_sum ot -  $\Phi$ (state
ot)
  by (induction ot) (auto simp: acost_def Let_def sum_list_subtractf cong:
map_cong)

corollary t_sum_le_a_sum: wf ot  $\Longrightarrow$  cost_sum ot  $\leq$  acost_sum ot
by (metis add commute t_sum_a_sum diff_add_cancel le_add_same_cancel2
ppos[OF inv_state])

lemma a_le_U:  $\llbracket \forall s \in \text{set } ss. \text{inv } s; \text{length } ss = \text{arity } f \rrbracket \Longrightarrow \text{acost } f \text{ } ss$ 
 $\leq U f \text{ } ss$ 
by(simp add: acost_def U)

lemma a_sum_le_U_sum: wf ot  $\Longrightarrow$  acost_sum ot  $\leq U\_sum$  ot
proof(induction ot)
  case (T f ts)
    with a_le_U[of map state ts f] sum_list_mono show ?case
    by (force simp: inv_state)
qed

corollary t_sum_le_U_sum: wf ot  $\Longrightarrow$  cost_sum ot  $\leq U\_sum$  ot
by (blast intro: t_sum_le_a_sum a_sum_le_U_sum order.trans)

end

```

**hide\_const** T

*Amortized2* supports the transfer of amortized analysis of one datatype (*Amortized arity exec inv cost  $\Phi$  U* on type '*s*') to an implementation (primed identifiers on type '*t*'). Function *hom* is assumed to be a homomorphism from '*t*' to '*s*', not just w.r.t. *exec* but also *cost* and *U*. The assumptions about *inv'* are weaker than the obvious *inv' = inv  $\circ$  hom*: the latter does not allow *inv* to be weaker than *inv'* (which we need in one application).

```

locale Amortized2 = Amortized arity exec inv cost  $\Phi$  U
  for arity :: 'op  $\Rightarrow$  nat and exec and inv :: 's  $\Rightarrow$  bool and cost  $\Phi$  U +
fixes exec' :: 'op  $\Rightarrow$  't list  $\Rightarrow$  't
fixes inv' :: 't  $\Rightarrow$  bool
fixes cost' :: 'op  $\Rightarrow$  't list  $\Rightarrow$  nat
fixes U' :: 'op  $\Rightarrow$  't list  $\Rightarrow$  real
fixes hom :: 't  $\Rightarrow$  's
assumes exec':  $\llbracket \forall s \in \text{set } ts. \text{inv}' s; \text{length } ts = \text{arity } f \rrbracket$ 

```

```

     $\implies \text{hom}(\text{exec}' f ts) = \text{exec } f (\text{map } \text{hom } ts)$ 
assumes inv_exec':  $\llbracket \forall s \in \text{set } ss. \text{inv}' s; \text{length } ss = \text{arity } f \rrbracket$ 
     $\implies \text{inv}'(\text{exec}' f ss)$ 
assumes inv_hom:  $\text{inv}' t \implies \text{inv } (\text{hom } t)$ 
assumes cost':  $\llbracket \forall s \in \text{set } ts. \text{inv}' s; \text{length } ts = \text{arity } f \rrbracket$ 
     $\implies \text{cost}' f ts = \text{cost } f (\text{map } \text{hom } ts)$ 
assumes U':  $\llbracket \forall s \in \text{set } ts. \text{inv}' s; \text{length } ts = \text{arity } f \rrbracket$ 
     $\implies U' f ts = U f (\text{map } \text{hom } ts)$ 
begin

sublocale A': Amortized arity exec' inv' cost'  $\Phi$  o hom U'
proof (standard, goal_cases)
  case 1 thus ?case by(simp add: exec' inv_exec' inv_exec)
next
  case 2 thus ?case by(simp add: inv_hom ppos)
next
  case 3 thus ?case
    by(simp add: U exec' U' map_map[symmetric] cost' inv_exec inv_hom
del: map_map)
qed

end

end

```

### 3 Simple Examples

```

theory Amortized_Examples
imports Amortized_Framework
begin

```

This theory applies the amortized analysis framework to a number of simple classical examples.

#### 3.1 Binary Counter

```

locale Bin_Counter
begin

datatype op = Empty | Incr

fun arity :: op  $\Rightarrow$  nat where
  arity Empty = 0 |
  arity Incr = 1

```

```

fun incr :: bool list  $\Rightarrow$  bool list where
  incr [] = [True] |
  incr (False#bs) = True # bs |
  incr (True#bs) = False # incr bs

```

```

fun tincr :: bool list  $\Rightarrow$  nat where
  tincr [] = 1 |
  tincr (False#bs) = 1 |
  tincr (True#bs) = tincr bs + 1

```

```

definition  $\Phi$  :: bool list  $\Rightarrow$  real where
 $\Phi$  bs = length(filter id bs)

```

```

lemma a_incr: tincr bs +  $\Phi$ (incr bs) -  $\Phi$  bs = 2
apply(induction bs rule: incr.induct)
apply (simp_all add:  $\Phi$ _def)
done

```

```

fun exec :: op  $\Rightarrow$  bool list list  $\Rightarrow$  bool list where
  exec Empty [] = [] |
  exec Incr [bs] = incr bs

```

```

fun cost :: op  $\Rightarrow$  bool list list  $\Rightarrow$  nat where
  cost Empty _ = 1 |
  cost Incr [bs] = tincr bs

```

**interpretation** Amortized

```

where exec = exec and arity = arity and inv =  $\lambda$ _. True
and cost = cost and  $\Phi$  =  $\Phi$  and U =  $\lambda$ f_. case f of Empty  $\Rightarrow$  1 | Incr
 $\Rightarrow$  2

```

**proof** (standard, goal\_cases)

case 1 show ?case by simp

next

case 2 show ?case by(simp add:  $\Phi$ \_def)

next

case 3 thus ?case using a\_incr by(auto simp:  $\Phi$ \_def split: op.split)

qed

end

### 3.2 Stack with multipop

locale Multipop

```

begin

datatype 'a op = Empty | Push 'a | Pop nat

fun arity :: 'a op ⇒ nat where
arity Empty = 0 |
arity (Push _) = 1 |
arity (Pop _) = 1

fun exec :: 'a op ⇒ 'a list list ⇒ 'a list where
exec Empty [] = [] |
exec (Push x) [xs] = x # xs |
exec (Pop n) [xs] = drop n xs

fun cost :: 'a op ⇒ 'a list list ⇒ nat where
cost Empty _ = 1 |
cost (Push x) _ = 1 |
cost (Pop n) [xs] = min n (length xs)

interpretation Amortized
where arity = arity and exec = exec and inv = λ_. True
and cost = cost and Φ = length
and U = λf_. case f of Empty ⇒ 1 | Push _ ⇒ 2 | Pop _ ⇒ 0
proof (standard, goal_cases)
  case 1 show ?case by simp
next
  case 2 thus ?case by simp
next
  case 3 thus ?case by (auto split: op.split)
qed

end

```

### 3.3 Dynamic tables: insert only

```

locale Dyn_Tab1
begin

type_synonym tab = nat × nat

datatype op = Empty | Ins

fun arity :: op ⇒ nat where

```



*arity Empty = 0 |*  
*arity Ins = 1*

**fun** *exec* :: *op*  $\Rightarrow$  *tab list*  $\Rightarrow$  *tab* **where**  
*exec Empty [] = (0::nat,0::nat) |*  
*exec Ins [(n,l)] = (n+1, if n<l then l else if l=0 then 1 else 2\*l)*

**fun** *cost* :: *op*  $\Rightarrow$  *tab list*  $\Rightarrow$  *nat* **where**  
*cost Empty \_ = 1 |*  
*cost Ins [(n,l)] = (if n<l then 1 else n+1)*

**interpretation** *Amortized*

**where** *exec* = *exec* **and** *arity* = *arity*  
**and** *inv* =  $\lambda(n,l).$  *if l=0 then n=0 else n  $\leq$  l  $\wedge$  l < 2\*n*  
**and** *cost* = *cost* **and**  $\Phi$  =  $\lambda(n,l).$  *2\*n - l*  
**and** *U* =  $\lambda f \_.$  *case f of Empty  $\Rightarrow$  1 | Ins  $\Rightarrow$  3*  
**proof** (*standard, goal\_cases*)  
  **case** (*1 \_ f*) **thus** ?*case* **by**(*cases f*) (*auto split: if\_splits*)  
**next**  
  **case** *2* **thus** ?*case* **by**(*auto split: prod\_splits*)  
**next**  
  **case** *3* **thus** ?*case* **by** (*auto split: op.split*) *linarith*  
**qed**  
  
**end**

**locale** *Dyn\_Tab2* =  
**fixes** *a* :: *real*  
**fixes** *c* :: *real*  
**assumes** *c1[arith]:* *c > 1*  
**assumes** *ac2:* *a  $\geq$  c/(c - 1)*  
**begin**

**lemma** *ac:* *a  $\geq$  1/(c - 1)*  
**using** *ac2* **by**(*simp add: field\_simps*)

**lemma** *a0[arith]:* *a > 0*  
**proof**–  
  **have** *1/(c - 1) > 0* **using** *ac* **by** *simp*  
  **thus** ?*thesis* **by** (*metis ac dual\_order.strict\_trans1*)  
**qed**

**definition** *b = 1/(c - 1)*

```

lemma b0[arith]:  $b > 0$ 
using ac by (simp add: b_def)

type_synonym tab = nat × nat

datatype op = Empty | Ins

fun arity :: op ⇒ nat where
arity Empty = 0 |
arity Ins = 1

fun ins :: tab ⇒ tab where
ins(n,l) = (n+1, if n<l then l else if l=0 then 1 else nat(ceiling(c*l)))

fun exec :: op ⇒ tab list ⇒ tab where
exec Empty [] = (0::nat,0::nat) |
exec Ins [s] = ins s |
exec _ _ = (0,0)

fun cost :: op ⇒ tab list ⇒ nat where
cost Empty _ = 1 |
cost Ins [(n,l)] = (if n<l then 1 else n+1)

fun Φ :: tab ⇒ real where
Φ(n,l) = a*n - b*l

interpretation Amortized
where exec = exec and arity = arity
and inv = λ(n,l). if l=0 then n=0 else n ≤ l ∧ (b/a)*l ≤ n
and cost = cost and Φ = Φ and U = λf_. case f of Empty ⇒ 1 | Ins ⇒
a + 1
proof (standard, goal_cases)
  case (1 ss f)
  show ?case
  proof (cases f)
    case Empty thus ?thesis using 1 by auto
  next
  case [simp]: Ins
  obtain n l where [simp]: ss = [(n,l)] using 1(2) by (auto)
  show ?thesis
  proof cases
    assume l=0 thus ?thesis using 1 ac
    by (simp add: b_def field_simps)
  next

```

```

assume  $l \neq 0$ 
show ?thesis
proof cases
  assume  $n < l$ 
  thus ?thesis using 1 by(simp add: algebra_simps)
next
  assume  $\neg n < l$ 
  hence [simp]:  $n = l$  using 1  $\langle l \neq 0 \rangle$  by simp
  have 1:  $(b/a) * \text{ceiling}(c * l) \leq \text{real } l + 1$ 
  proof-
    have  $(b/a) * \text{ceiling}(c * l) = \text{ceiling}(c * l) / (a * (c - 1))$ 
    by(simp add: b_def)
    also have  $\text{ceiling}(c * l) \leq c * l + 1$  by simp
    also have  $\dots \leq c * (\text{real } l + 1)$  by (simp add: algebra_simps)
    also have  $\dots / (a * (c - 1)) = (c / (a * (c - 1))) * (\text{real } l + 1)$  by
simp
    also have  $c / (a * (c - 1)) \leq 1$  using ac2 by (simp add: field_simps)
    finally show ?thesis by (simp add: divide_right_mono)
  qed
  have 2:  $\text{real } l + 1 \leq \text{ceiling}(c * \text{real } l)$ 
  by (metis  $\langle l \neq 0 \rangle$  c1 int_less_real_le less_ceiling_iff mult_less_cancel_right1
of_int_of_nat_eq of_nat_le_0_iff)
  from  $\langle l \neq 0 \rangle$  1 2 show ?thesis by simp
  qed
qed
qed
next
  case 2 thus ?case by(auto simp: field_simps split: if_splits prod.splits)
next
  case ( $\exists ss f$ )
  show ?case
  proof (cases f)
    case Empty thus ?thesis using 3(2) by simp
  next
    case [simp]: Ins
    obtain  $n l$  where [simp]:  $ss = [(n, l)]$  using 3(2) by (auto)
    show ?thesis
    proof cases
      assume  $l = 0$  thus ?thesis using 3 by (simp)
    next
      assume [arith]:  $l \neq 0$ 
      show ?thesis
      proof cases
        assume  $n < l$ 

```

```

    thus ?thesis using 3 ac by(simp add: algebra_simps b_def)
next
  assume  $\neg n < l$ 
  hence [simp]:  $n = l$  using 3 by simp
  have cost Ins [(n,l)] +  $\Phi$  (ins (n,l)) -  $\Phi(n,l) = l + a + 1 + (-$ 
 $b * \text{ceiling}(c * l)) + b * l$ 
    using  $\langle l \neq 0 \rangle$ 
    by(simp add: algebra_simps less_trans[of -1::real 0])
  also have  $- b * \text{ceiling}(c * l) \leq - b * (c * l)$  by (simp add: ceil-
ing_correct)
  also have  $l + a + 1 + - b * (c * l) + b * l = a + 1 + l * (1 - b * (c -$ 
 $1))$ 
    by (simp add: algebra_simps)
  also have  $b * (c - 1) = 1$  by(simp add: b_def)
  also have  $a + 1 + (real l) * (1 - 1) = a + 1$  by simp
  finally show ?thesis by simp
qed
qed
qed
qed
end

```

### 3.4 Dynamic tables: insert and delete

```

locale Dyn_Tab3

```

```

begin

```

```

type_synonym tab = nat  $\times$  nat

```

```

datatype op = Empty | Ins | Del

```

```

fun arity :: op  $\Rightarrow$  nat where

```

```

  arity Empty = 0 |

```

```

  arity Ins = 1 |

```

```

  arity Del = 1

```

```

fun exec :: op  $\Rightarrow$  tab list  $\Rightarrow$  tab where

```

```

  exec Empty [] = (0::nat, 0::nat) |

```

```

  exec Ins [(n,l)] = (n+1, if  $n < l$  then l else if  $l = 0$  then 1 else  $2 * l$ ) |

```

```

  exec Del [(n,l)] = (n-1, if  $n \leq 1$  then 0 else if  $4 * (n - 1) < l$  then  $l \text{ div } 2$  else
l)

```

```

fun cost :: op  $\Rightarrow$  tab list  $\Rightarrow$  nat where

```

```

cost Empty _ = 1 |
cost Ins [(n,l)] = (if n<l then 1 else n+1) |
cost Del [(n,l)] = (if n≤1 then 1 else if 4*(n - 1)<l then n else 1)

```

**interpretation** *Amortized*

```

where arity = arity and exec = exec
and inv = λ(n,l). if l=0 then n=0 else n ≤ l ∧ l ≤ 4*n
and cost = cost and Φ = (λ(n,l). if 2*n < l then l/2 - n else 2*n - l)
and U = λf_. case f of Empty ⇒ 1 | Ins ⇒ 3 | Del ⇒ 2
proof (standard, goal_cases)
  case (1 _ f) thus ?case by (cases f) (auto split: if_splits)
next
  case 2 thus ?case by(auto split: prod_splits)
next
  case (3 _ f) thus ?case
    by (cases f)(auto simp: field_simps split: prod_splits)
qed

end

```

### 3.5 Queue

See, for example, the book by Okasaki [6].

```

locale Queue
begin

```

```

datatype 'a op = Empty | Enq 'a | Deq

```

```

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

```

```

fun arity :: 'a op ⇒ nat where
arity Empty = 0 |
arity (Enq _) = 1 |
arity Deq = 1

```

```

fun exec :: 'a op ⇒ 'a queue list ⇒ 'a queue where
exec Empty [] = ([],[]) |
exec (Enq x) [(xs,ys)] = (x#xs,ys) |
exec Deq [(xs,ys)] = (if ys = [] then ([], tl(rev xs)) else (xs,tl ys))

```

```

fun cost :: 'a op ⇒ 'a queue list ⇒ nat where
cost Empty _ = 0 |
cost (Enq x) [(xs,ys)] = 1 |
cost Deq [(xs,ys)] = (if ys = [] then length xs else 0)

```

```

interpretation Amortized
where arity = arity and exec = exec and inv =  $\lambda\_.$  True
and cost = cost and  $\Phi = \lambda(xs,ys).$  length xs
and U =  $\lambda f\_.$  case f of Empty  $\Rightarrow$  0 | Enq _  $\Rightarrow$  2 | Deq  $\Rightarrow$  0
proof (standard, goal_cases)
  case 1 show ?case by simp
next
  case 2 thus ?case by (auto split: prod.splits)
next
  case 3 thus ?case by(auto split: op.split)
qed

end

locale Queue2
begin

datatype 'a op = Empty | Enq 'a | Deq

type__synonym 'a queue = 'a list * 'a list

fun arity :: 'a op  $\Rightarrow$  nat where
arity Empty = 0 |
arity (Enq _) = 1 |
arity Deq = 1

fun adjust :: 'a queue  $\Rightarrow$  'a queue where
adjust(xs,ys) = (if ys = [] then ([], rev xs) else (xs,ys))

fun exec :: 'a op  $\Rightarrow$  'a queue list  $\Rightarrow$  'a queue where
exec Empty [] = ([],[]) |
exec (Enq x) [(xs,ys)] = adjust(x#xs,ys) |
exec Deq [(xs,ys)] = adjust (xs, tl ys)

fun cost :: 'a op  $\Rightarrow$  'a queue list  $\Rightarrow$  nat where
cost Empty _ = 0 |
cost (Enq x) [(xs,ys)] = 1 + (if ys = [] then size xs + 1 else 0) |
cost Deq [(xs,ys)] = (if tl ys = [] then size xs else 0)

interpretation Amortized
where arity = arity and exec = exec
and inv =  $\lambda\_.$  True
and cost = cost and  $\Phi = \lambda(xs,ys).$  size xs

```

```

and  $U = \lambda f \_ . \text{case } f \text{ of } \text{Empty} \Rightarrow 0 \mid \text{Enq } \_ \Rightarrow 2 \mid \text{Deq} \Rightarrow 0$ 
proof (standard, goal_cases)
  case (1 _  $f$ ) thus ?case by (cases  $f$ ) (auto split: if_splits)
next
  case 2 thus ?case by (auto)
next
  case (3 _  $f$ ) thus ?case by(cases  $f$ ) (auto split: if_splits)
qed

```

**end**

```

locale Queue3
begin

```

```

datatype 'a op = Empty | Enq 'a | Deq

```

```

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

```

```

fun arity :: 'a op  $\Rightarrow$  nat where
arity Empty = 0 |
arity (Enq _) = 1 |
arity Deq = 1

```

```

fun balance :: 'a queue  $\Rightarrow$  'a queue where
balance( $xs, ys$ ) = (if size  $xs \leq$  size  $ys$  then ( $xs, ys$ ) else ( $[], ys @ \text{rev } xs$ ))

```

```

fun exec :: 'a op  $\Rightarrow$  'a queue list  $\Rightarrow$  'a queue where
exec Empty [] = ( $[], []$ ) |
exec (Enq  $x$ ) [( $xs, ys$ )] = balance( $x \# xs, ys$ ) |
exec Deq [( $xs, ys$ )] = balance ( $xs, \text{tl } ys$ )

```

```

fun cost :: 'a op  $\Rightarrow$  'a queue list  $\Rightarrow$  nat where
cost Empty _ = 0 |
cost (Enq  $x$ ) [( $xs, ys$ )] = 1 + (if size  $xs + 1 \leq$  size  $ys$  then 0 else size  $xs + 1 +$  size  $ys$ ) |
cost Deq [( $xs, ys$ )] = (if size  $xs \leq$  size  $ys - 1$  then 0 else size  $xs +$  (size  $ys - 1$ ))

```

```

interpretation Amortized

```

```

where arity = arity and exec = exec
and inv =  $\lambda(xs, ys) . \text{size } xs \leq \text{size } ys$ 
and cost = cost and  $\Phi = \lambda(xs, ys) . 2 * \text{size } xs$ 
and  $U = \lambda f \_ . \text{case } f \text{ of } \text{Empty} \Rightarrow 0 \mid \text{Enq } \_ \Rightarrow 3 \mid \text{Deq} \Rightarrow 0$ 
proof (standard, goal_cases)

```

```

    case (1 __ f) thus ?case by (cases f) (auto split: if_splits)
next
    case 2 thus ?case by (auto)
next
    case (3 __ f) thus ?case by (cases f) (auto split: prod_splits)
qed

end

```

```

end
theory Priority_Queue_ops_merge
imports Main
begin

```

```

datatype 'a op = Empty | Insert 'a | Del_min | Merge

```

```

fun arity :: 'a op => nat where
arity Empty = 0 |
arity (Insert _) = 1 |
arity Del_min = 1 |
arity Merge = 2

```

```

end

```

## 4 Skew Heap Analysis

```

theory Skew_Heap_Analysis
imports
    Complex_Main
    Skew_Heap.Skew_Heap
    Amortized_Framework
    HOL-Data_Structures.Define_Time_Function
    Priority_Queue_ops_merge
begin

```

The following proof is a simplified version of the one by Kaldewaij and Schoenmakers [3].

right-heavy:

```

definition rh :: 'a tree => 'a tree => nat where
rh l r = (if size l < size r then 1 else 0)

```

Function  $\Gamma$  in [3]: number of right-heavy nodes on left spine.

```

fun lrh :: 'a tree => nat where

```



```

lrh Leaf = 0 |
lrh (Node l _ r) = rh l r + lrh l

```

Function  $\Delta$  in [3]: number of not-right-heavy nodes on right spine.

```

fun rlh :: 'a tree  $\Rightarrow$  nat where
rlh Leaf = 0 |
rlh (Node l _ r) = (1 - rh l r) + rlh r

```

```

lemma Gexp:  $2^{\wedge} lrh\ t \leq size\ t + 1$ 
by (induction t) (auto simp: rh_def)

```

```

corollary Glog:  $lrh\ t \leq \log\ 2\ (size1\ t)$ 
by (metis Gexp le_log2_of_power size1_size)

```

```

lemma Dexp:  $2^{\wedge} rlh\ t \leq size\ t + 1$ 
by (induction t) (auto simp: rh_def)

```

```

corollary Dlog:  $rlh\ t \leq \log\ 2\ (size1\ t)$ 
by (metis Dexp le_log2_of_power size1_size)

```

```

time_fun merge

```

```

fun  $\Phi$  :: 'a tree  $\Rightarrow$  int where
 $\Phi$  Leaf = 0 |
 $\Phi$  (Node l _ r) =  $\Phi$  l +  $\Phi$  r + rh l r

```

```

lemma  $\Phi$ _nneg:  $\Phi\ t \geq 0$ 
by (induction t) auto

```

```

lemma plus_log_le_2log_plus:  $\llbracket x > 0; y > 0; b > 1 \rrbracket$ 
 $\implies \log\ b\ x + \log\ b\ y \leq 2 * \log\ b\ (x + y)$ 
by(subst mult_2; rule add_mono; auto)

```

```

lemma rh1:  $rh\ l\ r \leq 1$ 
by(simp add: rh_def)

```

```

lemma amor_le_long:
 $T\_merge\ t1\ t2 + \Phi\ (merge\ t1\ t2) - \Phi\ t1 - \Phi\ t2 \leq$ 
 $lrh(merge\ t1\ t2) + rlh\ t1 + rlh\ t2 + 1$ 
proof (induction t1 t2 rule: merge.induct)
case 1 thus ?case by simp
next
case 2 thus ?case by simp
next

```

```

case ( $\exists l1 a1 r1 l2 a2 r2$ )
show ?case
proof (cases  $a1 \leq a2$ )
  case True
    let ?t1 = Node l1 a1 r1 let ?t2 = Node l2 a2 r2 let ?m = merge ?t2
    r1
    have  $T\_merge\ ?t1\ ?t2 + \Phi\ (merge\ ?t1\ ?t2) - \Phi\ ?t1 - \Phi\ ?t2$ 
      =  $T\_merge\ ?t2\ r1 + 1 + \Phi\ ?m + \Phi\ l1 + rh\ ?m\ l1 - \Phi\ ?t1 - \Phi$ 
      ?t2
    using True by (simp)
    also have ... =  $T\_merge\ ?t2\ r1 + 1 + \Phi\ ?m + rh\ ?m\ l1 - \Phi\ r1 -$ 
    rh l1 r1 -  $\Phi\ ?t2$ 
    by simp
    also have ...  $\leq lrh\ ?m + rlh\ ?t2 + rlh\ r1 + rh\ ?m\ l1 + 2 - rh\ l1\ r1$ 
    using 3.IH(1)[OF True] by linarith
    also have ... =  $lrh\ ?m + rlh\ ?t2 + rlh\ r1 + rh\ ?m\ l1 + 1 + (1 - rh$ 
    l1 r1)
    using rh1[of l1 r1] by (simp)
    also have ... =  $lrh\ ?m + rlh\ ?t2 + rlh\ ?t1 + rh\ ?m\ l1 + 1$ 
    by (simp)
    also have ... =  $lrh\ (merge\ ?t1\ ?t2) + rlh\ ?t1 + rlh\ ?t2 + 1$ 
    using True by(simp)
    finally show ?thesis .
  next
    case False with  $\exists$  show ?thesis by auto
qed
qed

```

**lemma** amor\_le:

$$T\_merge\ t1\ t2 + \Phi\ (merge\ t1\ t2) - \Phi\ t1 - \Phi\ t2 \leq$$

$$lrh(merge\ t1\ t2) + rlh\ t1 + rlh\ t2 + 1$$

**by**(induction t1 t2 rule: merge.induct)(auto)

**lemma** a\_merge:

$$T\_merge\ t1\ t2 + \Phi(merge\ t1\ t2) - \Phi\ t1 - \Phi\ t2 \leq$$

$$\exists * \log 2\ (size1\ t1 + size1\ t2) + 1\ (\text{is } ?l \leq \_)$$

**proof** –

**have**  $?l \leq lrh(merge\ t1\ t2) + rlh\ t1 + rlh\ t2 + 1$  **using** amor\_le[of t1 t2] **by** arith

**also have** ... =  $real(lrh(merge\ t1\ t2)) + rlh\ t1 + rlh\ t2 + 1$  **by** simp

**also have** ... =  $real(lrh(merge\ t1\ t2)) + (real(rlh\ t1) + rlh\ t2) + 1$  **by** simp

**also have**  $rlh\ t1 \leq \log 2\ (size1\ t1)$  **by**(rule Dlog)

**also have**  $rlh\ t2 \leq \log 2\ (size1\ t2)$  **by**(rule Dlog)

```

also have  $lrh (merge\ t1\ t2) \leq \log\ 2 (size1 (merge\ t1\ t2))$  by(rule Glog)
also have  $size1 (merge\ t1\ t2) = size1\ t1 + size1\ t2 - 1$  by(simp add:
size1_size size_merge)
also have  $\log\ 2 (size1\ t1 + size1\ t2 - 1) \leq \log\ 2 (size1\ t1 + size1\ t2)$ 
by(simp add: size1_size)
also have  $\log\ 2 (size1\ t1) + \log\ 2 (size1\ t2) \leq 2 * \log\ 2 (real (size1\ t1)
+ (size1\ t2))$ 
by(rule plus_log_le_2log_plus) (auto simp: size1_size)
finally show ?thesis by(simp)
qed

```

Command *time\_fun* does not work for *skew\_heap.insert* and *skew\_heap.del\_min* because they are the result of a locale and not what they seem. However, their manual definition is trivial:

```

definition T_insert :: 'a::linorder  $\Rightarrow$  'a tree  $\Rightarrow$  int where
T_insert a t = T_merge (Node Leaf a Leaf) t

```

```

lemma a_insert:  $T\_insert\ a\ t + \Phi (skew\_heap.insert\ a\ t) - \Phi\ t \leq 3 * \log\ 2 (size1\ t + 2) + 1$ 

```

```

using a_merge[of Node Leaf a Leaf t]
by (simp add: numeral_eq_Suc T_insert_def rh_def)

```

```

definition T_del_min :: ('a::linorder) tree  $\Rightarrow$  int where
T_del_min t = (case t of Leaf  $\Rightarrow$  0 | Node t1 a t2  $\Rightarrow$  T_merge t1 t2)

```

```

lemma a_del_min:  $T\_del\_min\ t + \Phi (skew\_heap.del\_min\ t) - \Phi\ t \leq 3 * \log\ 2 (size1\ t + 2) + 1$ 

```

```

proof (cases t)
  case Leaf thus ?thesis by (simp add: T_del_min_def)
next
  case (Node t1 _ t2)
  have [arith]:  $\log\ 2 (2 + (real (size\ t1) + real (size\ t2))) \leq \log\ 2 (4 + (real (size\ t1) + real (size\ t2)))$  by simp
  from Node show ?thesis using a_merge[of t1 t2]
  by (simp add: size1_size rh_def T_del_min_def)
qed

```

#### 4.0.1 Instantiation of Amortized Framework

```

lemma T_merge_nneg:  $T\_merge\ t1\ t2 \geq 0$ 
by(induction t1 t2 rule: T_merge.induct) auto

```

```

fun exec :: 'a::linorder op  $\Rightarrow$  'a tree list  $\Rightarrow$  'a tree where
exec Empty [] = Leaf |

```

```

exec (Insert a) [t] = skew_heap.insert a t |
exec Del_min [t] = skew_heap.del_min t |
exec Merge [t1,t2] = merge t1 t2

```

```

fun cost :: 'a::linorder op ⇒ 'a tree list ⇒ nat where
cost Empty [] = 1 |
cost (Insert a) [t] = T_merge (Node Leaf a Leaf) t + 1 |
cost Del_min [t] = (case t of Leaf ⇒ 1 | Node t1 a t2 ⇒ T_merge t1 t2
+ 1) |
cost Merge [t1,t2] = T_merge t1 t2

```

```

fun U where
U Empty [] = 1 |
U (Insert _) [t] = 3 * log 2 (size1 t + 2) + 2 |
U Del_min [t] = 3 * log 2 (size1 t + 2) + 2 |
U Merge [t1,t2] = 3 * log 2 (size1 t1 + size1 t2) + 1

```

**interpretation** Amortized

**where** arity = arity **and** exec = exec **and** inv = λ\_. True  
**and** cost = cost **and** Φ = Φ **and** U = U

**proof** (standard, goal\_cases)

**case** 1 **show** ?case **by** simp

**next**

**case** (2 t) **show** ?case **using** Φ\_nneg[of t] **by** linarith

**next**

**case** (3 ss f)

**show** ?case

**proof** (cases f)

**case** Empty **thus** ?thesis **using** 3(2) **by** (auto)

**next**

**case** [simp]: (Insert a)

**obtain** t **where** [simp]: ss = [t] **using** 3(2) **by** (auto)

**thus** ?thesis **using** a\_merge[of Node Leaf a Leaf t]

**by** (simp add: numeral\_eq\_Suc insert\_def rh\_def T\_merge\_nneg)

**next**

**case** [simp]: Del\_min

**obtain** t **where** [simp]: ss = [t] **using** 3(2) **by** (auto)

**thus** ?thesis

**proof** (cases t)

**case** Leaf **with** Del\_min **show** ?thesis **by** simp

**next**

**case** (Node t1 \_ t2)

**have** [arith]: log 2 (2 + (real (size t1) + real (size t2))) ≤  
      log 2 (4 + (real (size t1) + real (size t2))) **by** simp

```

    from Del_min Node show ?thesis using a_merge[of t1 t2]
      by (simp add: size1_size T_merge_nneg)
  qed
next
  case [simp]: Merge
  obtain t1 t2 where ss = [t1,t2] using 3(2) by (auto simp: nu-
meral_eq_Suc)
  thus ?thesis using a_merge[of t1 t2] by (simp add: T_merge_nneg)
  qed
qed

```

```

end
theory Lemmas_log
imports Complex_Main
begin

```

```

lemma ld_sum_inequality:
  assumes  $x > 0$   $y > 0$ 
  shows  $\log 2 x + \log 2 y + 2 \leq 2 * \log 2 (x + y)$ 
proof -
  have 0:  $0 \leq (x-y)^2$  using assms by(simp)
  have 2 powr  $(2 + \log 2 x + \log 2 y) = 4 * x * y$  using assms
    by(simp add: powr_add)
  also have  $4*x*y \leq (x+y)^2$  using 0 by(simp add: algebra_simps nu-
meral_eq_Suc)
  also have  $\dots = 2 \text{ powr } (\log 2 (x + y) * 2)$  using assms
    by(simp add: powr_powr[symmetric] powr_numeral)
  finally show ?thesis by (simp add: mult_ac)
qed

```

```

lemma ld_ld_1_less:
   $\llbracket x > 0; y > 0 \rrbracket \implies 1 + \log 2 x + \log 2 y < 2 * \log 2 (x+y)$ 
using ld_sum_inequality[of x y] by linarith

```

```

lemma ld_le_2ld:
  assumes  $x \geq 0$   $y \geq 0$  shows  $\log 2 (1+x+y) \leq 1 + \log 2 (1+x) + \log$ 
 $2 (1+y)$ 
proof -
  have 1:  $1+x+y \leq (x+1)*(y+1)$  using assms
    by(simp add: algebra_simps)
  show ?thesis
  apply(rule powr_le_cancel_iff[of 2, THEN iffD1])
  apply simp

```

```

    using assms 1 by(simp add: powr_add algebra_simps)
qed

lemma ld_ld_less2: assumes  $x \geq 2 \ y \geq 2$ 
  shows  $1 + \log 2 \ x + \log 2 \ y \leq 2 * \log 2 \ (x + y - 1)$ 
proof-
  from assms have  $2*x \leq x*x \ 2*y \leq y*y$  by simp_all
  hence  $1: 2 * x * y \leq (x + y - 1)^2$ 
  by(simp add: numeral_eq_Suc algebra_simps)
  show ?thesis
  apply(rule powr_le_cancel_iff[of 2, THEN iffD1])
  apply simp
  using assms 1 by(simp add: powr_add log_powr[symmetric] powr_numeral)
qed

end

```

## 5 Splay Tree

### 5.1 Basics

```

theory Splay_Tree_Analysis_Base
imports
  Lemmas_log
  Splay_Tree.Splay_Tree
  HOL-Data_Structures.Define_Time_Function
begin

declare size1_size[simp]

abbreviation  $\varphi \ t == \log 2 \ (size1 \ t)$ 

fun  $\Phi :: 'a \ tree \Rightarrow \text{real}$  where
 $\Phi \ Leaf = 0 \ |$ 
 $\Phi \ (Node \ l \ a \ r) = \varphi \ (Node \ l \ a \ r) + \Phi \ l + \Phi \ r$ 

time_fun cmp
time_fun splay equations splay_simps(1) splay_code

lemma T_splay_simps[simp]:
   $T\_splay \ a \ (Node \ l \ a \ r) = 1$ 
   $x < b \Longrightarrow T\_splay \ x \ (Node \ Leaf \ b \ CD) = 1$ 
   $a < b \Longrightarrow T\_splay \ a \ (Node \ (Node \ A \ a \ B) \ b \ CD) = 1$ 
   $x < a \Longrightarrow x < b \Longrightarrow T\_splay \ x \ (Node \ (Node \ A \ a \ B) \ b \ CD) =$ 

```

```

    (if A = Leaf then 1 else T_splay x A + 1)
  x < b  $\implies$  a < x  $\implies$  T_splay x (Node (Node A a B) b CD) =
    (if B = Leaf then 1 else T_splay x B + 1)
  b < x  $\implies$  T_splay x (Node AB b Leaf) = 1
  b < a  $\implies$  T_splay a (Node AB b (Node C a D)) = 1
  b < x  $\implies$  x < c  $\implies$  T_splay x (Node AB b (Node C c D)) =
    (if C = Leaf then 1 else T_splay x C + 1)
  b < x  $\implies$  c < x  $\implies$  T_splay x (Node AB b (Node C c D)) =
    (if D = Leaf then 1 else T_splay x D + 1)
by (auto simp add: tree.case_eq_if)

declare T_splay.simps(2)[simp del]

time_fun insert

lemma T_insert_simp: T_insert x t = (if t = Leaf then 0 else T_splay x t)
by(auto split: tree.split)

time_fun splay_max

time_fun delete

lemma ex_in_set_tree: t  $\neq$  Leaf  $\implies$  bst t  $\implies$ 
   $\exists x' \in$  set_tree t. splay x' t = splay x t  $\wedge$  T_splay x' t = T_splay x t
proof(induction x t rule: splay.induct)
  case (6 x b c A)
  hence splay x A  $\neq$  Leaf by simp
  then obtain A1 u A2 where [simp]: splay x A = Node A1 u A2
    by (metis tree.exhaust)
  have b < c bst A using 6.prem1 by auto
  from 6.IH[OF  $\langle$ A  $\neq$  Leaf $\rangle$   $\langle$ bst A $\rangle$ ]
  obtain x' where x'  $\in$  set_tree A splay x' A = splay x A T_splay x' A =
    T_splay x A
    by blast
  moreover hence x' < b using 6.prem2(2) by auto
  ultimately show ?case using  $\langle$ x < c $\rangle$   $\langle$ x < b $\rangle$   $\langle$ b < c $\rangle$   $\langle$ bst A $\rangle$  by force
next
  case (8 a x c B)
  hence splay x B  $\neq$  Leaf by simp
  then obtain B1 u B2 where [simp]: splay x B = Node B1 u B2
    by (metis tree.exhaust)
  have a < c bst B using 8.prem1 by auto
  from 8.IH[OF  $\langle$ B  $\neq$  Leaf $\rangle$   $\langle$ bst B $\rangle$ ]

```

**obtain**  $x'$  **where**  $x' \in \text{set\_tree } B$   $\text{splay } x' B = \text{splay } x B$   $T\_splay\ x' B = T\_splay\ x B$   
**by** *blast*  
**moreover hence**  $a < x' \ \& \ x' < c$  **using**  $8.\text{prems}(2)$  **by** *simp*  
**ultimately show**  $?case$  **using**  $\langle x < c \rangle \langle a < x \rangle \langle a < c \rangle \langle \text{bst } B \rangle$  **by** *force*  
**next**  
**case**  $(11\ b\ x\ c\ C)$   
**hence**  $\text{splay } x\ C \neq \text{Leaf}$  **by** *simp*  
**then obtain**  $C1\ u\ C2$  **where**  $[\text{simp}]: \text{splay } x\ C = \text{Node } C1\ u\ C2$   
**by**  $(\text{metis tree.exhaust})$   
**have**  $b < c$   $\text{bst } C$  **using**  $11.\text{prems}$  **by** *auto*  
**from**  $11.IH[OF\ \langle C \neq \text{Leaf} \rangle \langle \text{bst } C \rangle]$   
**obtain**  $x'$  **where**  $x' \in \text{set\_tree } C$   $\text{splay } x' C = \text{splay } x C$   $T\_splay\ x' C = T\_splay\ x C$   
**by** *blast*  
**moreover hence**  $b < x' \ \& \ x' < c$  **using**  $11.\text{prems}$  **by** *simp*  
**ultimately show**  $?case$  **using**  $\langle b < x \rangle \langle x < c \rangle \langle b < c \rangle \langle \text{bst } C \rangle$  **by** *force*  
**next**  
**case**  $(14\ a\ x\ c\ D)$   
**hence**  $\text{splay } x\ D \neq \text{Leaf}$  **by** *simp*  
**then obtain**  $D1\ u\ D2$  **where**  $[\text{simp}]: \text{splay } x\ D = \text{Node } D1\ u\ D2$   
**by**  $(\text{metis tree.exhaust})$   
**have**  $a < c$   $\text{bst } D$  **using**  $14.\text{prems}$  **by** *auto*  
**from**  $14.IH[OF\ \langle D \neq \text{Leaf} \rangle \langle \text{bst } D \rangle]$   
**obtain**  $x'$  **where**  $x' \in \text{set\_tree } D$   $\text{splay } x' D = \text{splay } x D$   $T\_splay\ x' D = T\_splay\ x D$   
**by** *blast*  
**moreover hence**  $c < x'$  **using**  $14.\text{prems}(2)$  **by** *simp*  
**ultimately show**  $?case$  **using**  $\langle a < x \rangle \langle c < x \rangle \langle a < c \rangle \langle \text{bst } D \rangle$  **by** *force*  
**qed**  $(\text{auto simp: le\_less})$

**datatype**  $'a\ op = \text{Empty} \mid \text{Splay } 'a \mid \text{Insert } 'a \mid \text{Delete } 'a$

**fun**  $\text{arity} :: 'a::\text{linorder } op \Rightarrow \text{nat}$  **where**

$\text{arity } \text{Empty} = 0 \mid$   
 $\text{arity } (\text{Splay } x) = 1 \mid$   
 $\text{arity } (\text{Insert } x) = 1 \mid$   
 $\text{arity } (\text{Delete } x) = 1$

**fun**  $\text{exec} :: 'a::\text{linorder } op \Rightarrow 'a\ \text{tree } \text{list} \Rightarrow 'a\ \text{tree}$  **where**

$\text{exec } \text{Empty } [] = \text{Leaf} \mid$   
 $\text{exec } (\text{Splay } x) [t] = \text{splay } x\ t \mid$   
 $\text{exec } (\text{Insert } x) [t] = \text{Splay\_Tree.insert } x\ t \mid$



*exec (Delete x) [t] = Splay\_Tree.delete x t*

```
fun cost :: 'a::linorder op ⇒ 'a tree list ⇒ nat where
cost Empty [] = 1 |
cost (Splay x) [t] = T_splay x t |
cost (Insert x) [t] = T_insert x t |
cost (Delete x) [t] = T_delete x t
```

**end**

## 5.2 Splay Tree Analysis

**theory** *Splay\_Tree\_Analysis*

**imports**

*Splay\_Tree\_Analysis\_Base*

*Amortized\_Framework*

**begin**

### 5.2.1 Analysis of splay

**definition** *A\_splay* :: 'a::linorder ⇒ 'a tree ⇒ real **where**

*A\_splay a t = T\_splay a t + Φ(splay a t) - Φ t*

The following lemma is an attempt to prove a generic lemma that covers both zig-zig cases. However, the lemma is not as nice as one would like. Hence it is used only once, as a demo. Ideally the lemma would involve function *A\_splay*, but that is impossible because this involves *splay* and thus depends on the ordering. We would need a truly symmetric version of *splay* that takes the ordering as an explicit argument. Then we could define all the symmetric cases by one final equation *splay2* ( $<$ )  $t = \text{splay2 } (\lambda x y. \neg x < y)$  (*mirror t*). This would simplify the code and the proofs.

**lemma** *zig\_zig*: **fixes**  $lx\ x\ rx\ lb\ b\ rb\ a\ ra\ u\ lb1\ lb2$

**defines** [*simp*]:  $X == \text{Node } lx\ (x)\ rx$  **defines** [*simp*]:  $B == \text{Node } lb\ b\ rb$

**defines** [*simp*]:  $t == \text{Node } B\ a\ ra$  **defines** [*simp*]:  $A' == \text{Node } rb\ a\ ra$

**defines** [*simp*]:  $t' == \text{Node } lb1\ u\ (\text{Node } lb2\ b\ A')$

**assumes** *hyps*:  $lb \neq \langle \rangle$  **and** *IH*:  $T\_splay\ x\ lb + \Phi\ lb1 + \Phi\ lb2 - \Phi\ lb \leq 2 * \varphi\ lb - 3 * \varphi\ X + 1$  **and**

*prems*:  $size\ lb = size\ lb1 + size\ lb2 + 1$   $X \in subtrees\ lb$

**shows**  $T\_splay\ x\ lb + \Phi\ t' - \Phi\ t \leq 3 * (\varphi\ t - \varphi\ X)$

**proof** -

**define**  $B'$  **where** [*simp*]:  $B' = \text{Node } lb2\ b\ A'$

**have**  $T\_splay\ x\ lb + \Phi\ t' - \Phi\ t = T\_splay\ x\ lb + \Phi\ lb1 + \Phi\ lb2 - \Phi\ lb + \varphi\ B' + \varphi\ A' - \varphi\ B$

**using** *prems*

```

  by(auto simp: A_splay_def size_if_splay algebra_simps in_set_tree_if
split: tree.split)
  also have ... ≤ 2 * φ lb + φ B' + φ A' - φ B - 3 * φ X + 1
    using IH prems(2) by(auto simp: algebra_simps)
  also have ... ≤ φ lb + φ B' + φ A' - 3 * φ X + 1 by(simp)
  also have ... ≤ φ B' + 2 * φ t - 3 * φ X
    using prems ld_ld_1_less[of size1 lb size1 A']
    by(simp add: size_if_splay)
  also have ... ≤ 3 * φ t - 3 * φ X
    using prems by(simp add: size_if_splay)
  finally show ?thesis by simp
qed

```

```

lemma A_splay_ub: [[ bst t; Node l x r : subtrees t ]
  ⇒ A_splay x t ≤ 3 * (φ t - φ(Node l x r)) + 1

```

```

proof(induction x t rule: splay.induct)

```

```

  case 1 thus ?case by simp

```

```

next

```

```

  case 2 thus ?case by (auto simp: A_splay_def)

```

```

next

```

```

  case 4 hence False by(fastforce dest: in_set_tree_if) thus ?case ..

```

```

next

```

```

  case 5 hence False by(fastforce dest: in_set_tree_if) thus ?case ..

```

```

next

```

```

  case 7 hence False by(fastforce dest: in_set_tree_if) thus ?case ..

```

```

next

```

```

  case 10 hence False by(fastforce dest: in_set_tree_if) thus ?case ..

```

```

next

```

```

  case 12 hence False by(fastforce dest: in_set_tree_if) thus ?case ..

```

```

next

```

```

  case 13 hence False by(fastforce dest: in_set_tree_if) thus ?case ..

```

```

next

```

```

  case (3 x b A B CD)

```

```

  let ?t = ⟨⟨A, x, B⟩, b, CD⟩

```

```

  let ?t' = ⟨A, x, ⟨B, b, CD⟩⟩

```

```

  have *: l = A ∧ r = B using 3.premis by(fastforce dest: in_set_tree_if)

```

```

  have A_splay x ?t = 1 + Φ ?t' - Φ ?t using 3.hyps by (simp add:
A_splay_def)

```

```

  also have ... = 1 + φ ?t' + φ ⟨B, b, CD⟩ - φ ?t - φ ⟨A, x, B⟩
by(simp)

```

```

  also have ... = 1 + φ ⟨B, b, CD⟩ - φ ⟨A, x, B⟩ by(simp)

```

```

  also have ... ≤ 1 + φ ?t - φ(Node A x B)

```

```

    using log_le_cancel_iff[of 2 size1(Node B b CD) size1 ?t] by (simp)

```

**also have**  $\dots \leq 1 + 3 * (\varphi ?t - \varphi(\text{Node } A \ x \ B))$   
**using**  $\text{log\_le\_cancel\_iff}[of \ 2 \ \text{size1}(\text{Node } A \ x \ B) \ \text{size1 } ?t]$  **by** (*simp*)  
**finally show**  $?case$  **using**  $*$  **by** *simp*  
**next**  
**case** ( $9 \ b \ x \ AB \ C \ D$ )  
  
**let**  $?A = \langle AB, b, \langle C, x, D \rangle \rangle$   
**have**  $x \notin \text{set\_tree } AB$  **using**  $9.\text{prems}(1)$  **by** *auto*  
**with**  $9$  **show**  $?case$  **using**  
 $\text{log\_le\_cancel\_iff}[of \ 2 \ \text{size1}(\text{Node } AB \ b \ C) \ \text{size1 } ?A]$   
 $\text{log\_le\_cancel\_iff}[of \ 2 \ \text{size1}(\text{Node } C \ x \ D) \ \text{size1 } ?A]$   
**by** (*auto simp: A\_splay\_def algebra\_simps simp del:log\_le\_cancel\_iff*)  
**next**  
**case** ( $6 \ x \ a \ b \ A \ B \ C$ )  
**hence**  $*, \langle l, x, r \rangle \in \text{subtrees } A$  **by**(*fastforce dest: in\_set\_tree\_if*)  
**obtain**  $A1 \ a' \ A2$  **where**  $sp: \text{splay } x \ A = \text{Node } A1 \ a' \ A2$   
**using**  $\text{splay\_not\_Leaf}[OF \ \langle A \neq \text{Leaf} \rangle]$  **by** *blast*  
**let**  $?X = \text{Node } l \ x \ r$  **let**  $?AB = \text{Node } A \ a \ B$  **let**  $?ABC = \text{Node } ?AB \ b \ C$   
**let**  $?A' = \text{Node } A1 \ a' \ A2$   
**let**  $?BC = \text{Node } B \ b \ C$  **let**  $?A2BC = \text{Node } A2 \ a \ ?BC$  **let**  $?A1A2BC =$   
 $\text{Node } A1 \ a' \ ?A2BC$   
**have**  $0: \varphi ?A1A2BC = \varphi ?ABC$  **using**  $sp$  **by**(*simp add: size\_if\_splay*)  
**have**  $1: \Phi ?A1A2BC - \Phi ?ABC = \Phi A1 + \Phi A2 + \varphi ?A2BC + \varphi ?BC$   
 $- \Phi A - \varphi ?AB$   
**using**  $0$  **by** (*simp*)  
**have**  $A\_splay \ x \ ?ABC = T\_splay \ x \ A + 1 + \Phi ?A1A2BC - \Phi ?ABC$   
**using**  $6.\text{hyps } sp$  **by**(*simp add: A\_splay\_def*)  
**also have**  $\dots = T\_splay \ x \ A + 1 + \Phi A1 + \Phi A2 + \varphi ?A2BC + \varphi$   
 $?BC - \Phi A - \varphi ?AB$   
**using**  $1$  **by** *simp*  
**also have**  $\dots = T\_splay \ x \ A + \Phi ?A' - \varphi ?A' - \Phi A + \varphi ?A2BC + \varphi$   
 $?BC - \varphi ?AB + 1$   
**by**(*simp*)  
**also have**  $\dots = A\_splay \ x \ A + \varphi ?A2BC + \varphi ?BC - \varphi ?AB - \varphi ?A'$   
 $+ 1$   
**using**  $sp$  **by**(*simp add: A\_splay\_def*)  
**also have**  $\dots \leq 3 * \varphi A + \varphi ?A2BC + \varphi ?BC - \varphi ?AB - \varphi ?A' - 3$   
 $* \varphi ?X + 2$   
**using**  $6.IH \ 6.\text{prems}(1) *$  **by**(*simp*)  
**also have**  $\dots = 2 * \varphi A + \varphi ?A2BC + \varphi ?BC - \varphi ?AB - 3 * \varphi ?X$   
 $+ 2$   
**using**  $sp$  **by**(*simp add: size\_if\_splay*)  
**also have**  $\dots < \varphi A + \varphi ?A2BC + \varphi ?BC - 3 * \varphi ?X + 2$  **by**(*simp*)  
**also have**  $\dots < \varphi ?A2BC + 2 * \varphi ?ABC - 3 * \varphi ?X + 1$

```

    using sp ld_ld_1_less[of size1 A size1 ?BC]
    by(simp add: size_if_splay)
  also have ... < 3 * φ ?ABC - 3 * φ ?X + 1
    using sp by(simp add: size_if_splay)
  finally show ?case by simp
next
case (8 a x b B A C)
hence *: ⟨l, x, r⟩ ∈ subtrees B by(fastforce dest: in_set_tree_if)
obtain B1 b' B2 where sp: splay x B = Node B1 b' B2
  using splay_not_Leaf[OF ⟨B ≠ Leaf⟩] by blast
let ?X = Node l x r let ?AB = Node A a B let ?ABC = Node ?AB b C
let ?B' = Node B1 b' B2
let ?AB1 = Node A a B1 let ?B2C = Node B2 b C let ?AB1B2C =
Node ?AB1 b' ?B2C
  have 0: φ ?AB1B2C = φ ?ABC using sp by(simp add: size_if_splay)
  have 1: Φ ?AB1B2C - Φ ?ABC = Φ B1 + Φ B2 + φ ?AB1 + φ ?B2C
- Φ B - φ ?AB
    using 0 by (simp)
  have A_splay x ?ABC = T_splay x B + 1 + Φ ?AB1B2C - Φ ?ABC
    using 8.hyps sp by(simp add: A_splay_def)
  also have ... = T_splay x B + 1 + Φ B1 + Φ B2 + φ ?AB1 + φ ?B2C
- Φ B - φ ?AB
    using 1 by simp
  also have ... = T_splay x B + Φ ?B' - φ ?B' - Φ B + φ ?AB1 + φ
?B2C - φ ?AB + 1
    by simp
  also have ... = A_splay x B + φ ?AB1 + φ ?B2C - φ ?AB - φ ?B'
+ 1
    using sp by (simp add: A_splay_def)
  also have ... ≤ 3 * φ B + φ ?AB1 + φ ?B2C - φ ?AB - φ ?B' - 3
* φ ?X + 2
    using 8.IH 8.prem1(1) * by(simp)
  also have ... = 2 * φ B + φ ?AB1 + φ ?B2C - φ ?AB - 3 * φ ?X +
2
    using sp by(simp add: size_if_splay)
  also have ... < φ B + φ ?AB1 + φ ?B2C - 3 * φ ?X + 2 by(simp)
  also have ... < φ B + 2 * φ ?ABC - 3 * φ ?X + 1
    using sp ld_ld_1_less[of size1 ?AB1 size1 ?B2C]
    by(simp add: size_if_splay)
  also have ... < 3 * φ ?ABC - 3 * φ ?X + 1 by(simp)
  finally show ?case by simp
next
case (11 b x c C A D)
hence *: ⟨l, x, r⟩ ∈ subtrees C by(fastforce dest: in_set_tree_if)

```

```

obtain  $C1\ c'\ C2$  where  $sp: splay\ x\ C = Node\ C1\ c'\ C2$ 
  using  $splay\_not\_Leaf[OF\ \langle C \neq Leaf \rangle]$  by  $blast$ 
let  $?X = Node\ l\ x\ r$  let  $?CD = Node\ C\ c\ D$  let  $?ACD = Node\ A\ b\ ?CD$ 
let  $?C' = Node\ C1\ c'\ C2$ 
let  $?C2D = Node\ C2\ c\ D$  let  $?AC1 = Node\ A\ b\ C1$ 
have  $A\_splay\ x\ ?ACD = A\_splay\ x\ C + \varphi\ ?C2D + \varphi\ ?AC1 - \varphi\ ?CD$ 
 $- \varphi\ ?C' + 1$ 
  using  $11.hyps\ sp$ 
  by( $auto\ simp: A\_splay\_def\ size\_if\_splay\ algebra\_simps\ split: tree.split$ )
also have  $\dots \leq 3 * \varphi\ C + \varphi\ ?C2D + \varphi\ ?AC1 - \varphi\ ?CD - \varphi\ ?C' - 3$ 
 $* \varphi\ ?X + 2$ 
  using  $11.IH\ 11.prem1 * by(auto\ simp: algebra\_simps)$ 
also have  $\dots = 2 * \varphi\ C + \varphi\ ?C2D + \varphi\ ?AC1 - \varphi\ ?CD - 3 * \varphi\ ?X$ 
 $+ 2$ 
  using  $sp\ by(simp\ add: size\_if\_splay)$ 
also have  $\dots \leq \varphi\ C + \varphi\ ?C2D + \varphi\ ?AC1 - 3 * \varphi\ ?X + 2$  by( $simp$ )
also have  $\dots \leq \varphi\ C + 2 * \varphi\ ?ACD - 3 * \varphi\ ?X + 1$ 
  using  $sp\ ld\_ld\_1\_less[of\ size1\ ?C2D\ size1\ ?AC1]$ 
  by( $simp\ add: size\_if\_splay\ algebra\_simps$ )
also have  $\dots \leq 3 * \varphi\ ?ACD - 3 * \varphi\ ?X + 1$  by( $simp$ )
finally show  $?case$  by  $simp$ 
next
case ( $14\ a\ x\ b\ CD\ A\ B$ )
hence  $0: x \notin set\_tree\ B \wedge x \notin set\_tree\ A$ 
  using  $14.prem1\ \langle b < x \rangle$  by( $auto$ )
hence  $1: x \in set\_tree\ CD$  using  $14.prem1\ \langle b < x \rangle\ \langle a < x \rangle$  by ( $auto$ )
obtain  $C\ c\ D$  where  $sp: splay\ x\ CD = Node\ C\ c\ D$ 
  using  $splay\_not\_Leaf[OF\ \langle CD \neq Leaf \rangle]$  by  $blast$ 
from  $zig\_zig[of\ CD\ x\ D\ C\ l\ r\_b\ B\ a\ A]$   $14\ sp\ 0$ 
show  $?case$  by( $auto\ simp: A\_splay\_def\ size\_if\_splay\ algebra\_simps$ )

```

qed

**lemma**  $A\_splay\_ub2$ : **assumes**  $bst\ t\ x : set\_tree\ t$   
**shows**  $A\_splay\ x\ t \leq 3 * (\varphi\ t - 1) + 1$

**proof** –

```

from  $assms(2)$  obtain  $l\ r$  where  $N: Node\ l\ x\ r : subtrees\ t$ 
  by ( $metis\ set\_treeE$ )
have  $A\_splay\ x\ t \leq 3 * (\varphi\ t - \varphi(Node\ l\ x\ r)) + 1$  by( $rule\ A\_splay\_ub[OF\$ 
 $assms(1)\ N]$ )
also have  $\dots \leq 3 * (\varphi\ t - 1) + 1$  by( $simp\ add: field\_simps$ )
finally show  $?thesis$  .

```

qed

**lemma**  $A\_splay\_ub3$ : **assumes**  $bst\ t$  **shows**  $A\_splay\ x\ t \leq 3 * \varphi\ t + 1$   
**proof** *cases*  
**assume**  $t = Leaf$  **thus**  $?thesis$  **by**(*simp add: A\_splay\_def*)  
**next**  
**assume**  $t \neq Leaf$   
**from**  $ex\_in\_set\_tree$ [*OF this assms*] **obtain**  $x'$  **where**  
 $a': x' \in set\_tree\ t\ splay\ x'\ t = splay\ x\ t\ T\_splay\ x'\ t = T\_splay\ x\ t$   
**by** *blast*  
**show**  $?thesis$  **using**  $A\_splay\_ub2$ [*OF assms a'(1)*] **by**(*simp add: A\_splay\_def a'*)  
**qed**

### 5.2.2 Analysis of insert

**lemma**  $amor\_insert$ : **assumes**  $bst\ t$   
**shows**  $T\_insert\ x\ t + \Phi(Splay\_Tree.insert\ x\ t) - \Phi\ t \leq 4 * \log\ 2\ (size1\ t) + 2$  (**is**  $?l \leq ?r$ )  
**proof** *cases*  
**assume**  $t = Leaf$  **thus**  $?thesis$  **by**(*simp*)  
**next**  
**assume**  $t \neq Leaf$   
**then obtain**  $l\ e\ r$  **where** [*simp*]:  $splay\ x\ t = Node\ l\ e\ r$   
**by** (*metis tree.exhaust splay\_Leaf\_iff*)  
**let**  $?t = real(T\_splay\ x\ t)$   
**let**  $?Plr = \Phi\ l + \Phi\ r$  **let**  $?Ps = \Phi\ t$   
**let**  $?slr = real(size1\ l) + real(size1\ r)$  **let**  $?LR = \log\ 2\ (1 + ?slr)$   
**have**  $opt: ?t + \Phi\ (splay\ x\ t) - ?Ps \leq 3 * \log\ 2\ (real\ (size1\ t)) + 1$   
**using**  $A\_splay\_ub3$ [*OF <bst t>, simplified A\_splay\_def, of x*] **by** (*simp*)  
**from**  $less\_linear$ [*of e x*]  
**show**  $?thesis$   
**proof** (*elim disjE*)  
**assume**  $e=x$   
**have**  $nneg: \log\ 2\ (1 + real\ (size1\ t)) \geq 0$  **by** *simp*  
**thus**  $?thesis$  **using**  $<t \neq Leaf>$   $opt$   $<e=x>$   
**apply**(*simp add: algebra\_simps*) **using**  $nneg$  **by** *arith*  
**next**  
**let**  $?L = \log\ 2\ (real(size1\ l) + 1)$   
**assume**  $e < x$  **hence**  $e \neq x$  **by** *simp*  
**hence**  $?l = (?t + ?Plr - ?Ps) + ?L + ?LR$   
**using**  $<t \neq Leaf>$   $<e < x>$  **by**(*simp*)  
**also have**  $?t + ?Plr - ?Ps \leq 2 * \log\ 2\ ?slr + 1$   
**using**  $opt\ size\_splay$ [*of x t, symmetric*] **by**(*simp*)  
**also have**  $?L \leq \log\ 2\ ?slr$  **by**(*simp*)  
**also have**  $?LR \leq \log\ 2\ ?slr + 1$

```

proof -
  have  $?LR \leq \log 2 (2 * ?slr)$  by (simp add:)
  also have  $\dots \leq \log 2 ?slr + 1$ 
    by (simp add: log_mult del:distrib_left_numeral)
  finally show ?thesis .
qed
finally show ?thesis using size_splay[of x t,symmetric] by (simp)
next
let  $?R = \log 2 (2 + \text{real}(\text{size } r))$ 
assume  $x < e$  hence  $e \neq x$  by simp
hence  $?l = (?t + ?Plr - ?Ps) + ?R + ?LR$ 
  using  $\langle t \neq \text{Leaf} \rangle \langle x < e \rangle$  by(simp)
also have  $?t + ?Plr - ?Ps \leq 2 * \log 2 ?slr + 1$ 
  using opt_size_splay[of x t,symmetric] by(simp)
also have  $?R \leq \log 2 ?slr$  by(simp)
also have  $?LR \leq \log 2 ?slr + 1$ 
proof -
  have  $?LR \leq \log 2 (2 * ?slr)$  by (simp add:)
  also have  $\dots \leq \log 2 ?slr + 1$ 
    by (simp add: log_mult del:distrib_left_numeral)
  finally show ?thesis .
qed
finally show ?thesis using size_splay[of x t, symmetric] by simp
qed
qed

```

### 5.2.3 Analysis of delete

**definition**  $A\_splay\_max :: 'a::\text{linorder } \text{tree} \Rightarrow \text{real}$  **where**  
 $A\_splay\_max t = T\_splay\_max t + \Phi(\text{splay\_max } t) - \Phi t$

**lemma**  $A\_splay\_max\_ub: t \neq \text{Leaf} \implies A\_splay\_max t \leq 3 * (\varphi t - 1) + 1$

```

proof(induction t rule: splay_max.induct)
  case 1 thus ?case by (simp)
next
  case (2 A)
  thus ?case using one_le_log_cancel_iff[of 2 size1 A + 1]
    by (simp add: A_splay_max_def del: one_le_log_cancel_iff)
next
  case (3 l b rl c rr)
  show ?case
  proof cases
    assume  $rr = \text{Leaf}$ 

```

```

thus ?thesis
  using one_le_log_cancel_iff[of 2 1 + size1 rl]
  one_le_log_cancel_iff[of 2 1 + size1 l + size1 rl]
  log_le_cancel_iff[of 2 size1 l + size1 rl 1 + size1 l + size1 rl]
by (auto simp: A_splay_max_def field_simps
      simp del: log_le_cancel_iff one_le_log_cancel_iff)
next
assume rr ≠ Leaf
then obtain l' u r' where sp: splay_max rr = Node l' u r'
  using splay_max_Leaf_iff tree.exhaust by blast
hence 1: size rr = size l' + size r' + 1
  using size_splay_max[of rr, symmetric] by (simp)
let ?C = Node rl c rr let ?B = Node l b ?C
let ?B' = Node l b rl let ?C' = Node ?B' c l'
have A_splay_max ?B = A_splay_max rr + φ ?B' + φ ?C' - φ rr
- φ ?C + 1 using 3.prem1 sp 1
  by (auto simp add: A_splay_max_def)
also have ... ≤ 3 * (φ rr - 1) + φ ?B' + φ ?C' - φ rr - φ ?C + 2
  using 3 ⟨rr ≠ Leaf⟩ by auto
also have ... = 2 * φ rr + φ ?B' + φ ?C' - φ ?C - 1 by simp
also have ... ≤ φ rr + φ ?B' + φ ?C' - 1 by simp
also have ... ≤ 2 * φ ?B + φ ?C' - 2
  using ld_ld_1_less[of size1 ?B' size1 rr] by (simp add:)
also have ... ≤ 3 * φ ?B - 2 using 1 by simp
finally show ?case by simp
qed
qed

```

**lemma** A\_splay\_max\_ub3: A\_splay\_max t ≤ 3 \* φ t + 1

**proof** *cases*

**assume** t = Leaf **thus** ?thesis **by** (simp add: A\_splay\_max\_def)

**next**

**assume** t ≠ Leaf

**show** ?thesis **using** A\_splay\_max\_ub[OF ⟨t ≠ Leaf⟩] **by** (simp)

**qed**

**lemma** amor\_delete: **assumes** bst t

**shows** T\_delete a t + Φ(Splay\_Tree.delete a t) - Φ t ≤ 6 \* log 2 (size1 t) + 2

**proof** (*cases* t)

**case** Leaf **thus** ?thesis **by** (simp add: Splay\_Tree.delete\_def)

**next**

**case** [simp]: (Node ls x rs)

**then obtain** l e r **where** sp[simp]: splay a (Node ls x rs) = Node l e r



```

  by (metis tree.exhaust splay_Leaf_iff)
let ?t = real(T_splay a t)
let ?Plr =  $\Phi$  l +  $\Phi$  r let ?Ps =  $\Phi$  t
let ?slr = real(size1 l) + real(size1 r) let ?LR = log 2 (1 + ?slr)
let ?lslr = log 2 (real (size ls) + (real (size rs) + 2))
have ?lslr  $\geq$  0 by simp
have opt: ?t +  $\Phi$  (splay a t) - ?Ps  $\leq$  3 * log 2 (real (size1 t)) + 1
  using A_splay_ub3[OF  $\langle$ bst t $\rangle$ , simplified A_splay_def, of a]
  by (simp add: field_simps)
show ?thesis
proof (cases e=a)
  case False thus ?thesis
    using opt apply (simp add: Splay_Tree.delete_def field_simps)
    using  $\langle$ ?lslr  $\geq$  0 $\rangle$  by arith
next
  case [simp]: True
  show ?thesis
  proof (cases l)
    case Leaf
    have 1: log 2 (real (size r) + 2)  $\geq$  0 by (simp)
    show ?thesis
      using Leaf opt apply (simp add: Splay_Tree.delete_def field_simps)
      using 1  $\langle$ ?lslr  $\geq$  0 $\rangle$  by arith
  next
    case (Node ll y lr)
    then obtain l' y' r' where [simp]: splay_max (Node ll y lr) = Node
l' y' r'
    using splay_max_Leaf_iff tree.exhaust by blast
    have bst l using bst_splay[OF  $\langle$ bst t $\rangle$ , of a] by simp
    have  $\Phi$  r'  $\geq$  0 apply (induction r') by (auto)
    have optm: real(T_splay_max l) +  $\Phi$  (splay_max l) -  $\Phi$  l  $\leq$  3 *  $\varphi$ 
l + 1
      using A_splay_max_ub3[of l, simplified A_splay_max_def] by
(simp add: field_simps Node)
    have 1: log 2 (2+(real(size l')+real(size r)))  $\leq$  log 2 (2+(real(size
l)+real(size r)))
      using size_splay_max[of l] Node by simp
    have 2: log 2 (2 + (real (size l') + real (size r')))  $\geq$  0 by simp
    have 3: log 2 (size1 l' + size1 r)  $\leq$  log 2 (size1 l' + size1 r') + log 2
?slr
      apply simp using 1 2 by arith
    have 4: log 2 (real(size ll) + (real(size lr) + 2))  $\leq$  ?lslr
      using size_if_splay[OF sp] Node by simp
    show ?thesis using add_mono[OF opt optm] Node 3

```

```

    apply(simp add: Splay_Tree.delete_def field_simps)
    using 4 ⟨ $\Phi$   $r' \geq 0$ ⟩ by arith
  qed
qed
qed

```

#### 5.2.4 Overall analysis

**fun**  $U$  **where**

```

 $U$  Empty [] = 1 |
 $U$  (Splay _) [t] = 3 * log 2 (size1 t) + 1 |
 $U$  (Insert _) [t] = 4 * log 2 (size1 t) + 3 |
 $U$  (Delete _) [t] = 6 * log 2 (size1 t) + 3

```

**interpretation** *Amortized*

**where**  $arity = arity$  **and**  $exec = exec$  **and**  $inv = bst$

**and**  $cost = cost$  **and**  $\Phi = \Phi$  **and**  $U = U$

**proof** (*standard, goal\_cases*)

**case** (1  $ss$   $f$ ) **show** ?*case*

**proof** (*cases f*)

**case** *Empty* **thus** ?*thesis* **using** 1 **by** *auto*

**next**

**case** (*Splay a*)

**then obtain**  $t$  **where**  $ss = [t]$  *bst t* **using** 1 **by** *auto*

**with** *Splay bst\_splay[OF <bst t>, of a]* **show** ?*thesis*

**by** (*auto split: tree.split*)

**next**

**case** (*Insert a*)

**then obtain**  $t$  **where**  $ss = [t]$  *bst t* **using** 1 **by** *auto*

**with** *bst\_splay[OF <bst t>, of a] Insert* **show** ?*thesis*

**by** (*auto simp: splay\_bstL[OF <bst t>] splay\_bstR[OF <bst t>] split: tree.split*)

**next**

**case** (*Delete a*)

**then obtain**  $t$  **where**  $ss = [t]$  *bst t* **using** 1 **by** *auto*

**with** 1 *Delete* **show** ?*thesis* **by** (*simp add: bst\_delete*)

**qed**

**next**

**case** (2  $t$ ) **thus** ?*case* **by** (*induction t*) *auto*

**next**

**case** (3  $ss$   $f$ )

**show** ?*case* (**is** ? $l \leq ?r$ )

**proof**(*cases f*)

**case** *Empty* **thus** ?*thesis* **using** 3(2) **by**(*simp add: A\_splay\_def*)

```

next
  case (Splay a)
  then obtain t where ss = [t] bst t using  $\mathcal{B}$  by auto
  thus ?thesis using Splay A_splay_ub $\mathcal{B}$ [OF ‹bst t›] by (simp add: A_splay_def)
next
  case [simp]: (Insert a)
  then obtain t where [simp]: ss = [t] and bst t using  $\mathcal{B}$  by auto
  thus ?thesis using amor_insert[of t a] by auto
next
  case [simp]: (Delete a)
  then obtain t where [simp]: ss = [t] and bst t using  $\mathcal{B}$  by auto
  thus ?thesis using amor_delete[of t a] by auto
qed
qed

end

```

### 5.3 Splay Tree Analysis (Optimal)

```

theory Splay_Tree_Analysis_Optimal
imports
  Splay_Tree_Analysis_Base
  Amortized_Framework
  HOL-Library.Sum_of_Squares
begin

```

This analysis follows Schoenmakers [7].

#### 5.3.1 Analysis of splay

```

locale Splay_Analysis =
fixes  $\alpha :: real$  and  $\beta :: real$ 
assumes a1[arith]:  $\alpha > 1$ 
assumes A1:  $\llbracket 1 \leq x; 1 \leq y; 1 \leq z \rrbracket \implies$ 
   $(x+y) * (y+z) \text{ powr } \beta \leq (x+y) \text{ powr } \beta * (x+y+z)$ 
assumes A2:  $\llbracket 1 \leq l'; 1 \leq r'; 1 \leq lr; 1 \leq r \rrbracket \implies$ 
   $\alpha * (l'+r') * (lr+r) \text{ powr } \beta * (lr+r'+r) \text{ powr } \beta$ 
   $\leq (l'+r') \text{ powr } \beta * (l'+lr+r') \text{ powr } \beta * (l'+lr+r'+r)$ 
assumes A3:  $\llbracket 1 \leq l'; 1 \leq r'; 1 \leq ll; 1 \leq r \rrbracket \implies$ 
   $\alpha * (l'+r') * (l'+ll) \text{ powr } \beta * (r'+r) \text{ powr } \beta$ 
   $\leq (l'+r') \text{ powr } \beta * (l'+ll+r') \text{ powr } \beta * (l'+ll+r'+r)$ 
begin

```

```

lemma nl2:  $\llbracket ll \geq 1; lr \geq 1; r \geq 1 \rrbracket \implies$ 
   $\log \alpha (ll + lr) + \beta * \log \alpha (lr + r)$ 

```

$\leq \beta * \log \alpha (ll + lr) + \log \alpha (ll + lr + r)$   
**apply**(rule powr\_le\_cancel\_iff[THEN iffD1, OF a1])  
**apply**(simp add: powr\_add mult.commute[of  $\beta$ ] powr\_powr[symmetric] A1)  
**done**

**definition**  $\varphi :: 'a \text{ tree} \Rightarrow 'a \text{ tree} \Rightarrow \text{real}$  **where**  
 $\varphi \ t1 \ t2 = \beta * \log \alpha (size1 \ t1 + size1 \ t2)$

**fun**  $\Phi :: 'a \text{ tree} \Rightarrow \text{real}$  **where**  
 $\Phi \ Leaf = 0 \mid$   
 $\Phi \ (Node \ l \ _ \ r) = \Phi \ l + \Phi \ r + \varphi \ l \ r$

**definition**  $A :: 'a::\text{linorder} \Rightarrow 'a \text{ tree} \Rightarrow \text{real}$  **where**  
 $A \ a \ t = T\_splay \ a \ t + \Phi(splay \ a \ t) - \Phi \ t$

**lemma**  $A\_simps[simp]$ :  $A \ a \ (Node \ l \ a \ r) = 1$   
 $a < b \implies A \ a \ (Node \ (Node \ ll \ a \ lr) \ b \ r) = \varphi \ lr \ r - \varphi \ lr \ ll + 1$   
 $b < a \implies A \ a \ (Node \ l \ b \ (Node \ rl \ a \ rr)) = \varphi \ rl \ l - \varphi \ rr \ rl + 1$   
**by**(auto simp add: A\_def  $\varphi\_def$  algebra\_simps)

**lemma**  $A\_ub$ :  $\llbracket \text{bst } t; \text{Node } la \ a \ ra : \text{subtrees } t \rrbracket$   
 $\implies A \ a \ t \leq \log \alpha ((size1 \ t)/(size1 \ la + size1 \ ra)) + 1$   
**proof**(induction a t rule: splay.induct)  
**case 1 thus ?case by simp**  
**next**  
**case 2 thus ?case by auto**  
**next**  
**case 4 hence False by(fastforce dest: in\_set\_tree\_if) thus ?case ..**  
**next**  
**case 5 hence False by(fastforce dest: in\_set\_tree\_if) thus ?case ..**  
**next**  
**case 7 hence False by(fastforce dest: in\_set\_tree\_if) thus ?case ..**  
**next**  
**case 10 hence False by(fastforce dest: in\_set\_tree\_if) thus ?case ..**  
**next**  
**case 12 hence False by(fastforce dest: in\_set\_tree\_if) thus ?case ..**  
**next**  
**case 13 hence False by(fastforce dest: in\_set\_tree\_if) thus ?case ..**  
**next**  
**case** ( $\exists \ b \ a \ lb \ rb \ ra$ )  
**have**  $b \notin \text{set\_tree } ra$  **using** 3.prem1 **by auto**  
**thus ?case using** 3.prem1,2  $nl2$ [of size1 lb size1 rb size1 ra]

```

    by (auto simp:  $\varphi\_def$  log_divide algebra_simps)
next
case (9 a b la lb rb)
have  $b \notin \text{set\_tree } la$  using 9.prem1 by auto
thus ?case using 9.prem1,2 nl2[of size1 rb size1 lb size1 la]
  by (auto simp add:  $\varphi\_def$  log_divide algebra_simps)
next
case (6 x b a lb rb ra)
hence 0:  $x \notin \text{set\_tree } rb \wedge x \notin \text{set\_tree } ra$  using 6.prem1 by auto
hence 1:  $x \in \text{set\_tree } lb$  using 6.prem2  $\langle x < b \rangle$  by (auto)
then obtain lu u ru where sp: splay x lb = Node lu u ru
  using 6.prem1,2 by (cases splay x lb) auto
have  $b < a$  using 6.prem1,2 by (auto)
let ?lu = real (size1 lu) let ?ru = real (size1 ru)
let ?rb = real (size1 rb) let ?r = real (size1 ra)
have  $1 + \log \alpha (?lu + ?ru) + \beta * \log \alpha (?rb + ?r) + \beta * \log \alpha (?rb + ?ru + ?r) \leq$ 
 $\beta * \log \alpha (?lu + ?ru) + \beta * \log \alpha (?lu + ?rb + ?ru) + \log \alpha (?lu + ?rb + ?ru + ?r)$  (is ?L ≤ ?R)
proof (rule powr_le_cancel_iff [THEN iffD1, OF a1])
  show  $\alpha \text{ powr } ?L \leq \alpha \text{ powr } ?R$  using A2 [of ?lu ?ru ?rb ?r]
  by (simp add: powr_add add_ac mult.commute [of  $\beta$ ] powr_powr [symmetric])
qed
thus ?case using 6 0 1 sp
  by (auto simp: A_def  $\varphi\_def$  size_if_splay algebra_simps log_divide)
next
case (8 b x a rb lb ra)
hence 0:  $x \notin \text{set\_tree } lb \wedge x \notin \text{set\_tree } ra$  using 8.prem1 by (auto)
hence 1:  $x \in \text{set\_tree } rb$  using 8.prem2  $\langle b < x \rangle \langle x < a \rangle$  by (auto)
then obtain lu u ru where sp: splay x rb = Node lu u ru
  using 8.prem1,2 by (cases splay x rb) auto
let ?lu = real (size1 lu) let ?ru = real (size1 ru)
let ?lb = real (size1 lb) let ?r = real (size1 ra)
have  $1 + \log \alpha (?lu + ?ru) + \beta * \log \alpha (?lu + ?lb) + \beta * \log \alpha (?ru + ?r) \leq$ 
 $\beta * \log \alpha (?lu + ?ru) + \beta * \log \alpha (?lu + ?lb + ?ru) + \log \alpha (?lu + ?lb + ?ru + ?r)$  (is ?L ≤ ?R)
proof (rule powr_le_cancel_iff [THEN iffD1, OF a1])
  show  $\alpha \text{ powr } ?L \leq \alpha \text{ powr } ?R$  using A3 [of ?lu ?ru ?lb ?r]
  by (simp add: powr_add mult.commute [of  $\beta$ ] powr_powr [symmetric])
qed
thus ?case using 8 0 1 sp
  by (auto simp add: A_def size_if_splay  $\varphi\_def$  log_divide algebra_simps)
next

```

```

case (11 a x b lb la rb)
hence 0:  $x \notin \text{set\_tree } rb \wedge x \notin \text{set\_tree } la$  using 11.prems(1) by (auto)
hence 1:  $x \in \text{set\_tree } lb$  using 11.prems ⟨a<x⟩ ⟨x<b⟩ by (auto)
then obtain lu u ru where sp: splay x lb = Node lu u ru
  using 11.prems(1,2) by(cases splay x lb) auto
let ?lu = real (size1 lu) let ?ru = real (size1 ru)
let ?l = real(size1 la) let ?rb = real(size1 rb)
have  $1 + \log \alpha (?lu + ?ru) + \beta * \log \alpha (?lu + ?l) + \beta * \log \alpha (?ru + ?rb) \leq$ 
   $\beta * \log \alpha (?lu + ?ru) + \beta * \log \alpha (?lu + ?ru + ?rb) + \log \alpha (?lu + ?l + ?ru + ?rb)$  (is ?L≤?R)
proof(rule powr_le_cancel_iff[THEN iffD1, OF a1])
  show  $\alpha \text{ powr } ?L \leq \alpha \text{ powr } ?R$  using A3[of ?ru ?lu ?rb ?l]
  by(simp add: powr_add mult.commute[of β] powr_powr[symmetric])
  (simp add: algebra_simps)
qed
thus ?case using 11 0 1 sp
  by(auto simp add: A_def size_if_splay φ_def log_divide algebra_simps)
next
case (14 a x b rb la lb)
hence 0:  $x \notin \text{set\_tree } lb \wedge x \notin \text{set\_tree } la$  using 14.prems(1) by(auto)
hence 1:  $x \in \text{set\_tree } rb$  using 14.prems ⟨a<x⟩ ⟨b<x⟩ by (auto)
then obtain lu u ru where sp: splay x rb = Node lu u ru
  using 14.prems(1,2) by(cases splay x rb) auto
let ?la = real(size1 la) let ?lb = real(size1 lb)
let ?lu = real (size1 lu) let ?ru = real (size1 ru)
have  $1 + \log \alpha (?lu + ?ru) + \beta * \log \alpha (?la + ?lb) + \beta * \log \alpha (?lu + ?la + ?lb) \leq$ 
   $\beta * \log \alpha (?lu + ?ru) + \beta * \log \alpha (?lu + ?lb + ?ru) + \log \alpha (?lu + ?lb + ?ru + ?la)$  (is ?L≤?R)
proof(rule powr_le_cancel_iff[THEN iffD1, OF a1])
  show  $\alpha \text{ powr } ?L \leq \alpha \text{ powr } ?R$  using A2[of ?ru ?lu ?lb ?la]
  by(simp add: powr_add add_ac mult.commute[of β] powr_powr[symmetric])
qed
thus ?case using 14 0 1 sp
  by(auto simp add: A_def size_if_splay φ_def log_divide algebra_simps)
qed

```

**lemma** A\_ub2: **assumes** bst t a : set\_tree t

**shows**  $A a t \leq \log \alpha ((\text{size1 } t)/2) + 1$

**proof** –

**from** assms(2) **obtain** la ra **where** N: Node la a ra : subtrees t

**by** (metis set\_treeE)

**have**  $A a t \leq \log \alpha ((\text{size1 } t)/(\text{size1 } la + \text{size1 } ra)) + 1$

**by**(rule  $A\_ub$ [ $OF$   $assms(1)$   $N$ ])  
**also have**  $\dots \leq \log \alpha ((size1\ t)/2) + 1$  **by**(simp add: field\_simps)  
**finally show**  $?thesis$  **by** simp  
**qed**

**lemma**  $A\_ub3$ : **assumes**  $bst\ t$  **shows**  $A\ a\ t \leq \log \alpha (size1\ t) + 1$   
**proof** cases

**assume**  $t = Leaf$  **thus**  $?thesis$  **by**(simp add:  $A\_def$ )  
**next**  
**assume**  $t \neq Leaf$   
**from**  $ex\_in\_set\_tree$ [ $OF$   $this\ assms$ ] **obtain**  $a'$  **where**  
 $a': a' \in set\_tree\ t$   $splay\ a'\ t = splay\ a\ t$   $T\_splay\ a'\ t = T\_splay\ a\ t$   
**by** blast  
**have** [ $arith$ ]:  $\log \alpha\ 2 > 0$  **by** simp  
**show**  $?thesis$  **using**  $A\_ub2$ [ $OF$   $assms\ a'(1)$ ] **by**(simp add:  $A\_def\ a'$   
 $log\_divide$ )  
**qed**

**definition**  $Am :: 'a::linorder\ tree \Rightarrow real$  **where**  
 $Am\ t = T\_splay\_max\ t + \Phi(splay\_max\ t) - \Phi\ t$

**lemma**  $Am\_simp3'$ :  $\llbracket c < b; bst\ rr; rr \neq Leaf \rrbracket \Longrightarrow$   
 $Am\ (Node\ l\ c\ (Node\ rl\ b\ rr)) =$   
 $(case\ splay\_max\ rr\ of\ Node\ rrl\_ rrr \Rightarrow$   
 $Am\ rr + \varphi\ rrl\ (Node\ l\ c\ rl) + \varphi\ l\ rl - \varphi\ rl\ rr - \varphi\ rrl\ rrr + 1)$   
**by**(auto simp:  $Am\_def\ \varphi\_def\ size\_if\_splay\_max\ algebra\_simps\ neq\_Leaf\_iff$   
 $split: tree.split$ )

**lemma**  $Am\_ub$ :  $\llbracket bst\ t; t \neq Leaf \rrbracket \Longrightarrow Am\ t \leq \log \alpha ((size1\ t)/2) + 1$

**proof**(induction  $t$  rule:  $splay\_max.induct$ )  
**case** 1 **thus**  $?case$  **by** (simp)  
**next**  
**case** 2 **thus**  $?case$  **by** (simp add:  $Am\_def$ )  
**next**  
**case** ( $\exists\ l\ b\ rl\ c\ rr$ )  
**show**  $?case$   
**proof** cases  
**assume**  $rr = Leaf$   
**thus**  $?thesis$   
**using**  $nl2$ [ $of\ 1\ size1\ rl\ size1\ l$ ]  $log\_le\_cancel\_iff$ [ $of\ \alpha\ 2\ 2 + real(size$   
 $rl)$ ]  
**by**(auto simp:  $Am\_def\ \varphi\_def\ log\_divide\ field\_simps$   
 $simp\ del: log\_le\_cancel\_iff$ )

**next**  
**assume**  $rr \neq \text{Leaf}$   
**then obtain**  $l' u r'$  **where**  $sp: \text{splay\_max } rr = \text{Node } l' u r'$   
**using**  $\text{splay\_max\_Leaf\_iff } tree.exhaust$  **by**  $blast$   
**hence**  $1: \text{size } rr = \text{size } l' + \text{size } r' + 1$   
**using**  $\text{size\_splay\_max}[of\ rr]$  **by**  $(simp)$   
**let**  $?l = \text{real } (size1\ l)$  **let**  $?rl = \text{real } (size1\ rl)$   
**let**  $?l' = \text{real } (size1\ l')$  **let**  $?r' = \text{real } (size1\ r')$   
**have**  $1 + \log \alpha (?l' + ?r') + \beta * \log \alpha (?l + ?rl) + \beta * \log \alpha (?l' + ?l + ?rl) \leq$   
 $\beta * \log \alpha (?l' + ?r') + \beta * \log \alpha (?l' + ?rl + ?r') + \log \alpha (?l' + ?rl + ?r' + ?l)$  **(is**  $?L \leq ?R$ **)**  
**proof**  $(rule\ \text{powr\_le\_cancel\_iff}[THEN\ \text{iffD1},\ OF\ a1])$   
**show**  $\alpha\ ?L \leq \alpha\ \text{powr } ?R$  **using**  $A2[of\ ?r'\ ?l'\ ?rl\ ?l]$   
**by**  $(simp\ add: \text{powr\_add } add.commute\ add.left\_commute\ mult.commute[of\ \beta]\ \text{powr\_powr}[symmetric])$   
**qed**  
**thus**  $?case$  **using**  $\exists\ sp\ 1\ \langle rr \neq \text{Leaf} \rangle$   
**by**  $(auto\ simp\ add: Am\_simp3'\ \varphi\_def\ \log\_divide\ algebra\_simps)$   
**qed**  
**qed**

**lemma**  $Am\_ub3$ : **assumes**  $bst\ t$  **shows**  $Am\ t \leq \log \alpha (size1\ t) + 1$

**proof**  $cases$

**assume**  $t = \text{Leaf}$  **thus**  $?thesis$  **by**  $(simp\ add: Am\_def)$

**next**

**assume**  $t \neq \text{Leaf}$

**have**  $[arith]: \log \alpha\ 2 > 0$  **by**  $simp$

**show**  $?thesis$  **using**  $Am\_ub[OF\ \text{assms } \langle t \neq \text{Leaf} \rangle]$  **by**  $(simp\ add: Am\_def\ \log\_divide)$

**qed**

**end**

### 5.3.2 Optimal Interpretation

**lemma**  $mult\_root\_eq\_root$ :

$n > 0 \implies y \geq 0 \implies \text{root } n\ x * y = \text{root } n\ (x * (y \wedge n))$

**by**  $(simp\ add: \text{real\_root\_mult } \text{real\_root\_pos2})$

**lemma**  $mult\_root\_eq\_root2$ :

$n > 0 \implies y \geq 0 \implies y * \text{root } n\ x = \text{root } n\ ((y \wedge n) * x)$

**by**  $(simp\ add: \text{real\_root\_mult } \text{real\_root\_pos2})$



**lemma** *powr\_inverse\_numeral*:

$0 < x \implies x \text{ powr } (1 / \text{numeral } n) = \text{root } (\text{numeral } n) x$   
**by** (*simp add: root\_powr\_inverse*)

**lemmas** *root\_simps = mult\_root\_eq\_root mult\_root\_eq\_root2 powr\_inverse\_numeral*

**lemma** *nl31*:  $\llbracket (l'::\text{real}) \geq 1; r' \geq 1; lr \geq 1; r \geq 1 \rrbracket \implies$

$4 * (l' + r') * (lr + r) \leq (l' + lr + r' + r)^2$   
**by** (*sos* ((( $A < 0 * R < 1$ ) + ( $R < 1 * (R < 1 * [r + \sim 1 * l' + lr + \sim 1 * r']^2$ ))))))

**lemma** *nl32*: **assumes**  $(l'::\text{real}) \geq 1 r' \geq 1 lr \geq 1 r \geq 1$

**shows**  $4 * (l' + r') * (lr + r) * (lr + r' + r) \leq (l' + lr + r' + r)^3$

**proof** –

**have** 1:  $lr + r' + r \leq l' + lr + r' + r$  **using** *assms* **by** *arith*

**have** 2:  $0 \leq (l' + lr + r' + r)^2$  **by** (*rule zero\_le\_power2*)

**have** 3:  $0 \leq lr + r' + r$  **using** *assms* **by** *arith*

**from** *mult\_mono*[*OF nl31*[*OF assms*] 1 2 3] **show** *?thesis*

**by**(*simp add: ac\_simps numeral\_eq\_Suc*)

**qed**

**lemma** *nl3*: **assumes**  $(l'::\text{real}) \geq 1 r' \geq 1 lr \geq 1 r \geq 1$

**shows**  $4 * (l' + r')^2 * (lr + r) * (lr + r' + r)$

$\leq (l' + lr + r') * (l' + lr + r' + r)^3$

**proof**–

**have** 1:  $l' + r' \leq l' + lr + r'$  **using** *assms* **by** *arith*

**have** 2:  $0 \leq (l' + lr + r' + r)^3$  **using** *assms* **by** *simp*

**have** 3:  $0 \leq l' + r'$  **using** *assms* **by** *arith*

**from** *mult\_mono*[*OF nl32*[*OF assms*] 1 2 3] **show** *?thesis*

**unfolding** *power2\_eq\_square* **by** (*simp only: ac\_simps*)

**qed**

**lemma** *nl41*: **assumes**  $(l'::\text{real}) \geq 1 r' \geq 1 ll \geq 1 r \geq 1$

**shows**  $4 * (l' + ll) * (r' + r) \leq (l' + ll + r' + r)^2$

**using** *assms* **by** (*sos* ((( $A < 0 * R < 1$ ) + ( $R < 1 * (R < 1 * [r + \sim 1 * l' + \sim 1 * ll + r']^2$ ))))))

**lemma** *nl42*: **assumes**  $(l'::\text{real}) \geq 1 r' \geq 1 ll \geq 1 r \geq 1$

**shows**  $4 * (l' + r') * (l' + ll) * (r' + r) \leq (l' + ll + r' + r)^3$

**proof** –

**have** 1:  $l' + r' \leq l' + ll + r' + r$  **using** *assms* **by** *arith*

**have** 2:  $0 \leq (l' + ll + r' + r)^2$  **by** (*rule zero\_le\_power2*)

**have** 3:  $0 \leq l' + r'$  **using** *assms* **by** *arith*

**from** *mult\_mono*[*OF nl41*[*OF assms*] 1 2 3] **show** *?thesis*  
**by**(*simp add: ac\_simps numeral\_eq\_Suc del: distrib\_right\_numeral*)  
**qed**

**lemma** *nl4*: **assumes**  $(l'::\text{real}) \geq 1 \ r' \geq 1 \ ll \geq 1 \ r \geq 1$   
**shows**  $4 * (l' + r')^2 * (l' + ll) * (r' + r)$   
 $\leq (l' + ll + r') * (l' + ll + r' + r)^3$

**proof** –

**have** 1:  $l' + r' \leq l' + ll + r'$  **using** *assms* **by** *arith*

**have** 2:  $0 \leq (l' + ll + r' + r)^3$  **using** *assms* **by** *simp*

**have** 3:  $0 \leq l' + r'$  **using** *assms* **by** *arith*

**from** *mult\_mono*[*OF nl42*[*OF assms*] 1 2 3] **show** *?thesis*

**unfolding** *power2\_eq\_square* **by** (*simp only: ac\_simps*)

**qed**

**lemma** *cancel*:  $x > (0::\text{real}) \implies c * x^2 * y * z \leq u * v \implies c * x^3 * y * z \leq x * u * v$

**by**(*simp add: power2\_eq\_square power3\_eq\_cube*)

**interpretation** *S34*: *Splay\_Analysis root 3 4 1/3*

**proof** (*standard, goal\_cases*)

**case** 2 **thus** *?case*

**by**(*simp add: root\_simps*)

(*auto simp: numeral\_eq\_Suc split\_mult\_pos\_le intro!: mult\_mono*)

**next**

**case** 3 **thus** *?case* **by**(*simp add: root\_simps cancel nl3*)

**next**

**case** 4 **thus** *?case* **by**(*simp add: root\_simps cancel nl4*)

**qed** *simp*

**lemma** *log4\_log2*:  $\log 4 x = \log 2 x / 2$

**proof** –

**have**  $\log 4 x = \log (2^2) x$  **by** *simp*

**also have**  $\dots = \log 2 x / 2$  **by**(*simp only: log\_base\_pow*)

**finally show** *?thesis* .

**qed**

**declare** *log\_base\_root*[*simp*]

**lemma** *A34\_ub*: **assumes** *bst t*

**shows** *S34.A a t*  $\leq (3/2) * \log 2 (\text{size1 } t) + 1$

**proof** –

**have** *S34.A a t*  $\leq \log (\text{root } 3 \ 4) (\text{size1 } t) + 1$  **by**(*rule S34.A\_ub3*[*OF*

*assms*])  
**also have**  $\dots = (3/2) * \log 2 (size\ t + 1) + 1$  **by**(*simp add: log4\_log2*)  
**finally show** *?thesis* **by** *simp*  
**qed**

**lemma** *Am34\_ub*: **assumes** *bst t*  
**shows** *S34.Am t*  $\leq (3/2) * \log 2 (size1\ t) + 1$   
**proof** –  
**have** *S34.Am t*  $\leq \log (root\ 3\ 4) (size1\ t) + 1$  **by**(*rule S34.Am\_ub3[OF assms]*)  
**also have**  $\dots = (3/2) * \log 2 (size1\ t) + 1$  **by**(*simp add: log4\_log2*)  
**finally show** *?thesis* **by** *simp*  
**qed**

### 5.3.3 Overall analysis

**fun** *U* **where**  
*U Empty [] = 1* |  
*U (Splay \_) [t] = (3/2) \* log 2 (size1 t) + 1* |  
*U (Insert \_) [t] = 2 \* log 2 (size1 t) + 3/2* |  
*U (Delete \_) [t] = 3 \* log 2 (size1 t) + 2*

**interpretation** *Amortized*  
**where** *arity = arity* **and** *exec = exec* **and** *inv = bst*  
**and** *cost = cost* **and**  $\Phi = S34.\Phi$  **and** *U = U*  
**proof** (*standard, goal\_cases*)  
**case** (*1 ss f*) **show** *?case*  
**proof** (*cases f*)  
**case** *Empty* **thus** *?thesis* **using** *1* **by** *auto*  
**next**  
**case** (*Splay a*)  
**then obtain** *t* **where** *ss = [t] bst t* **using** *1* **by** *auto*  
**with** *Splay bst\_splay[OF <bst t>, of a]* **show** *?thesis*  
**by** (*auto split: tree.split*)  
**next**  
**case** (*Insert a*)  
**then obtain** *t* **where** *ss = [t] bst t* **using** *1* **by** *auto*  
**with** *bst\_splay[OF <bst t>, of a] Insert* **show** *?thesis*  
**by** (*auto simp: splay\_bstL[OF <bst t>] splay\_bstR[OF <bst t>] split: tree.split*)  
**next**  
**case** (*Delete a*)  
**then obtain** *t* **where** *ss = [t] bst t* **using** *1* **by** *auto*  
**with** *1 Delete* **show** *?thesis* **by**(*simp add: bst\_delete*)

```

qed
next
  case (2 t) show ?case by(induction t)(simp_all add: S34.φ_def)
next
  case (3 ss f)
  show ?case (is ?l ≤ ?r)
  proof(cases f)
    case Empty thus ?thesis using 3(2) by(simp add: S34.A_def)
  next
    case (Splay a)
    then obtain t where ss = [t] bst t using 3 by auto
    thus ?thesis using S34.A_ub3[OF ‹bst t›] Splay
      by(simp add: S34.A_def log4_log2)
  next
    case [simp]: (Insert a)
    obtain t where [simp]: ss = [t] and bst t using 3 by auto
    show ?thesis
    proof cases
      assume t = Leaf thus ?thesis by(simp add: S34.φ_def log4_log2)
    next
      assume t ≠ Leaf
      then obtain l e r where [simp]: splay a t = Node l e r
        by (metis tree.exhaust splay_Leaf_iff)
      let ?t = real(T_splay a t)
      let ?Plr = S34.Φ l + S34.Φ r let ?Ps = S34.Φ t
      let ?slr = real(size1 l) + real(size1 r) let ?LR = log 2 (1 + ?slr)
      have opt: ?t + S34.Φ (splay a t) - ?Ps ≤ 3/2 * log 2 (real (size1
t)) + 1
        using S34.A_ub3[OF ‹bst t›, simplified S34.A_def, of a]
        by (simp add: log4_log2)
      from less_linear[of e a]
      show ?thesis
      proof (elim disjE)
        assume e=a
        have nneg: log 2 (1 + real (size t)) ≥ 0 by simp
        thus ?thesis using ‹t ≠ Leaf› opt ‹e=a›
          apply(simp add: field_simps) using nneg by arith
      next
        let ?L = log 2 (real(size1 l) + 1)
        assume e<a hence e ≠ a by simp
        hence ?l = (?t + ?Plr - ?Ps) + ?L / 2 + ?LR / 2
          using ‹t ≠ Leaf› ‹e<a› by(simp add: S34.φ_def log4_log2)
        also have ?t + ?Plr - ?Ps ≤ log 2 ?slr + 1
          using opt size_splay[of a t,symmetric]

```

```

    by(simp add: S34.φ_def log4_log2)
  also have  $?L/2 \leq \log 2 \ ?slr / 2$  by(simp)
  also have  $?LR/2 \leq \log 2 \ ?slr / 2 + 1/2$ 
  proof -
    have  $?LR/2 \leq \log 2 (2 * ?slr) / 2$  by simp
    also have  $\dots \leq \log 2 \ ?slr / 2 + 1/2$ 
      by (simp add: log_mult del:distrib_left_numeral)
    finally show ?thesis .
  qed
  finally show ?thesis using size_splay[of a t,symmetric] by simp
next
let  $?R = \log 2 (2 + \text{real}(\text{size } r))$ 
assume  $a < e$  hence  $e \neq a$  by simp
hence  $?l = (?t + ?Plr - ?Ps) + ?R / 2 + ?LR / 2$ 
  using  $\langle t \neq \text{Leaf} \rangle \langle a < e \rangle$  by(simp add: S34.φ_def log4_log2)
also have  $?t + ?Plr - ?Ps \leq \log 2 \ ?slr + 1$ 
  using opt_size_splay[of a t,symmetric]
  by(simp add: S34.φ_def log4_log2)
also have  $?R/2 \leq \log 2 \ ?slr / 2$  by(simp)
also have  $?LR/2 \leq \log 2 \ ?slr / 2 + 1/2$ 
  proof -
    have  $?LR/2 \leq \log 2 (2 * ?slr) / 2$  by simp
    also have  $\dots \leq \log 2 \ ?slr / 2 + 1/2$ 
      by (simp add: log_mult del:distrib_left_numeral)
    finally show ?thesis .
  qed
  finally show ?thesis using size_splay[of a t,symmetric] by simp
qed
qed
next
case [simp]: (Delete a)
obtain  $t$  where [simp]:  $ss = [t]$  and  $\text{bst } t$  using 3 by auto
show ?thesis
proof (cases  $t$ )
  case Leaf thus ?thesis
    by(simp add: Splay_Tree.delete_def S34.φ_def log4_log2)
next
case [simp]: (Node  $ls\ x\ rs$ )
then obtain  $l\ e\ r$  where  $sp[simp]: \text{splay } a (\text{Node } ls\ x\ rs) = \text{Node } l\ e\ r$ 
  by (metis tree.exhaust splay_Leaf_iff)
let  $?t = \text{real}(T\_splay\ a\ t)$ 
let  $?Plr = S34.\Phi\ l + S34.\Phi\ r$  let  $?Ps = S34.\Phi\ t$ 
let  $?slr = \text{real}(\text{size1 } l) + \text{real}(\text{size1 } r)$  let  $?LR = \log 2 (1 + ?slr)$ 
let  $?lslr = \log 2 (\text{real}(\text{size } ls) + (\text{real}(\text{size } rs) + 2))$ 

```

```

have ?lslr ≥ 0 by simp
have opt: ?t + S34.Φ (splay a t) - ?Ps ≤ 3/2 * log 2 (real (size1
t)) + 1
  using S34.A_ub3[OF ‹bst t›, simplified S34.A_def, of a]
  by (simp add: log4_log2 field_simps)
show ?thesis
proof (cases e=a)
  case False thus ?thesis using opt
    apply(simp add: Splay_Tree.delete_def field_simps)
    using ‹?lslr ≥ 0› by arith
next
  case [simp]: True
  show ?thesis
  proof (cases l)
    case Leaf
      have S34.φ Leaf r ≥ 0 by(simp add: S34.φ_def)
      thus ?thesis using Leaf opt
        apply(simp add: Splay_Tree.delete_def field_simps)
        using ‹?lslr ≥ 0› by arith
    next
      case (Node ll y lr)
      then obtain l' y' r' where [simp]:
        splay_max (Node ll y lr) = Node l' y' r'
        using splay_max_Leaf_iff tree.exhaust by blast
      have bst l using bst_splay[OF ‹bst t›, of a] by simp
        have S34.Φ r' ≥ 0 apply (induction r') by (auto simp add:
S34.φ_def)
      have optm: real(T_splay_max l) + S34.Φ (splay_max l) - S34.Φ
l
        ≤ 3/2 * log 2 (real (size1 l)) + 1
        using S34.Am_ub3[OF ‹bst l›, simplified S34.Am_def]
        by (simp add: log4_log2 field_simps Node)
      have 1: log 4 (2+(real(size l')+real(size r))) ≤
log 4 (2+(real(size l)+real(size r)))
        using size_splay_max[of l] Node by simp
      have 2: log 4 (2 + (real (size l') + real (size r'))) ≥ 0 by simp
      have 3: S34.φ l' r ≤ S34.φ l' r' + S34.φ l r
        apply(simp add: S34.φ_def) using 1 2 by arith
      have 4: log 2 (real(size ll) + (real(size lr) + 2)) ≤ ?lslr
        using size_if_splay[OF sp] Node by simp
      show ?thesis using add_mono[OF opt optm] Node 3
        apply(simp add: Splay_Tree.delete_def field_simps)
        using 4 ‹S34.Φ r' ≥ 0› by arith
  qed

```

```

    qed
  qed
  qed
  qed

end
theory Priority_Queue_ops
imports Main
begin

datatype 'a op = Empty | Insert 'a | Del_min

fun arity :: 'a op  $\Rightarrow$  nat where
arity Empty = 0 |
arity (Insert _) = 1 |
arity Del_min = 1

end

```

## 6 Splay Heap

```

theory Splay_Heap_Analysis
imports
  Splay_Tree.Splay_Heap
  Amortized_Framework
  Priority_Queue_ops
  Lemmas_log
begin

```

Timing functions must be kept in sync with the corresponding functions on splay heaps.

```

fun T_part :: 'a::linorder  $\Rightarrow$  'a tree  $\Rightarrow$  nat where
T_part p Leaf = 1 |
T_part p (Node l a r) =
  (if a  $\leq$  p then
    case r of
      Leaf  $\Rightarrow$  1 |
      Node rl b rr  $\Rightarrow$  if b  $\leq$  p then T_part p rr + 1 else T_part p rl + 1
    else case l of
      Leaf  $\Rightarrow$  1 |
      Node ll b lr  $\Rightarrow$  if b  $\leq$  p then T_part p lr + 1 else T_part p ll + 1)

```

```

definition T_in :: 'a::linorder  $\Rightarrow$  'a tree  $\Rightarrow$  nat where
T_in x h = T_part x h

```

```

fun T_dm :: 'a::linorder tree  $\Rightarrow$  nat where
  T_dm Leaf = 1 |
  T_dm (Node Leaf _ r) = 1 |
  T_dm (Node (Node ll a lr) b r) = (if ll=Leaf then 1 else T_dm ll + 1)

```

**abbreviation**  $\varphi$  t == log 2 (size1 t)

```

fun  $\Phi$  :: 'a tree  $\Rightarrow$  real where
   $\Phi$  Leaf = 0 |
   $\Phi$  (Node l a r) =  $\Phi$  l +  $\Phi$  r +  $\varphi$  (Node l a r)

```

**lemma** amor\_del\_min: T\_dm t +  $\Phi$  (del\_min t) -  $\Phi$  t  $\leq$  2 \*  $\varphi$  t + 1

**proof**(induction t rule: T\_dm.induct)

**case** ( $\exists$  ll a lr b r)

**let** ?t = Node (Node ll a lr) b r

**show** ?case

**proof** cases

**assume** [simp]: ll = Leaf

**have** 1: log 2 (real (size1 lr) + real (size1 r))

$\leq$  3 \* log 2 (1 + (real (size1 lr) + real (size1 r))) (**is** ?l  $\leq$  3 \* ?r)

**proof** -

**have** ?l  $\leq$  ?r **by**(simp add: size1\_size)

**also have** ...  $\leq$  3 \* ?r **by**(simp)

**finally show** ?thesis .

**qed**

**have** 2: log 2 (1 + real (size1 lr))  $\geq$  0 **by** simp

**thus** ?case **apply** simp **using** 1 2 **by** linarith

**next**

**assume** ll[simp]:  $\neg$  ll = Leaf

**let** ?l' = del\_min ll

**let** ?s = Node ll a lr **let** ?t = Node ?s b r

**let** ?s' = Node lr b r **let** ?t' = Node ?l' a ?s'

**have** 0:  $\varphi$  ?t'  $\leq$   $\varphi$  ?t **by**(simp add: size1\_size)

**have** 1:  $\varphi$  ll  $<$   $\varphi$  ?s **by**(simp add: size1\_size)

**have** 2: log 2 (size1 ll + size1 ?s')  $\leq$  log 2 (size1 ?t) **by**(simp add: size1\_size)

**have** T\_dm ?t +  $\Phi$  (del\_min ?t) -  $\Phi$  ?t

= 1 + T\_dm ll +  $\Phi$  (del\_min ?t) -  $\Phi$  ?t **by** simp

**also have** ...  $\leq$  2 + 2 \*  $\varphi$  ll +  $\Phi$  ll -  $\Phi$  ?l' +  $\Phi$  (del\_min ?t) -  $\Phi$  ?t

**using** 3 ll **by** linarith

**also have** ... = 2 + 2 \*  $\varphi$  ll +  $\varphi$  ?t' +  $\varphi$  ?s' -  $\varphi$  ?t -  $\varphi$  ?s **by**(simp)

**also have** ...  $\leq$  2 +  $\varphi$  ll +  $\varphi$  ?s' **using** 0 1 **by** linarith

**also have** ...  $<$  2 \*  $\varphi$  ?t + 1 **using** 2 ld\_ld\_1\_less[of size1 ll size1



```

?<math>s</math>]
  by (simp add: size1_size)
  finally show ?case by simp
qed
qed auto

```

**lemma zig\_zig:**

**fixes**  $s\ u\ r\ r1'\ r2'\ T\ a\ b$

**defines**  $t == \text{Node } s\ a\ (\text{Node } u\ b\ r)$  **and**  $t' == \text{Node } (\text{Node } s\ a\ u)\ b\ r1'$

**assumes**  $\text{size } r1' \leq \text{size } r$

$$T\_part\ p\ r + \Phi\ r1' + \Phi\ r2' - \Phi\ r \leq 2 * \varphi\ r + 1$$

**shows**  $T\_part\ p\ r + 1 + \Phi\ t' + \Phi\ r2' - \Phi\ t \leq 2 * \varphi\ t + 1$

**proof** –

**have** 1:  $\varphi\ r \leq \varphi\ (\text{Node } u\ b\ r)$  **by** (simp add: size1\_size)

**have** 2:  $\log\ 2\ (\text{real } (\text{size1 } s + \text{size1 } u + \text{size1 } r1')) \leq \varphi\ t$

**using** *assms*(3) **by** (simp add: t\_def size1\_size)

**from** *ld\_ld\_1\_less*[of size1 s + size1 u size1 r]

**have**  $1 + \varphi\ r + \log\ 2\ (\text{size1 } s + \text{size1 } u) \leq 2 * \log\ 2\ (\text{size1 } s + \text{size1 } u + \text{size1 } r)$

**by**(simp add: size1\_size)

**thus** ?thesis **using** *assms* 1 2 **by** (simp add: algebra\_simps)

**qed**

**lemma zig\_zag:**

**fixes**  $s\ u\ r\ r1'\ r2'\ a\ b$

**defines**  $t \equiv \text{Node } s\ a\ (\text{Node } r\ b\ u)$  **and**  $t1' \equiv \text{Node } s\ a\ r1'$  **and**  $t2' \equiv \text{Node } u\ b\ r2'$

**assumes**  $\text{size } r = \text{size } r1' + \text{size } r2'$

$$T\_part\ p\ r + \Phi\ r1' + \Phi\ r2' - \Phi\ r \leq 2 * \varphi\ r + 1$$

**shows**  $T\_part\ p\ r + 1 + \Phi\ t1' + \Phi\ t2' - \Phi\ t \leq 2 * \varphi\ t + 1$

**proof** –

**have** 1:  $\varphi\ r \leq \varphi\ (\text{Node } u\ b\ r)$  **by** (simp add: size1\_size)

**have** 2:  $\varphi\ r \leq \varphi\ t$  **by** (simp add: t\_def size1\_size)

**from** *ld\_ld\_less2*[of size1 s + size1 r1' size1 u + size1 r2']

**have**  $1 + \log\ 2\ (\text{size1 } s + \text{size1 } r1') + \log\ 2\ (\text{size1 } u + \text{size1 } r2') \leq 2 * \varphi\ t$

**by**(simp add: *assms*(4) size1\_size t\_def ac\_simps)

**thus** ?thesis **using** *assms* 1 2 **by** (simp add: algebra\_simps)

**qed**

**lemma amor\_partition:**  $\text{bst\_wrt } (\leq)\ t \implies \text{partition } p\ t = (l', r')$

$$\implies T\_part\ p\ t + \Phi\ l' + \Phi\ r' - \Phi\ t \leq 2 * \log\ 2\ (\text{size1 } t) + 1$$

**proof**(*induction* p t *arbitrary:* l' r' *rule:* partition.induct)

**case** 1 **thus** ?case **by** simp

```

next
  case (2 p l a r)
  show ?case
  proof cases
    assume a ≤ p
    show ?thesis
    proof (cases r)
      case Leaf thus ?thesis using ⟨a ≤ p⟩ 2.prem1 by fastforce
    next
      case [simp]: (Node rl b rr)
      let ?t = Node l a r
      show ?thesis
      proof cases
        assume b ≤ p
        with ⟨a ≤ p⟩ 2.prem1 obtain rrl
          where 0: partition p rr = (rrl, r') l' = Node (Node l a rl) b rrl
          by (auto split: tree.splits prod.splits)
        have size rrl ≤ size rr
          using size_partition[OF 0(1)] by (simp add: size1_size)
        with 0 ⟨a ≤ p⟩ ⟨b ≤ p⟩ 2.prem1(1) 2.IH(1)[OF _ Node, of rrl r']
          zig_zig[where s=l and u=rl and r=rr and r1'=rrl and r2'=r']
and p=p, of a b]
          show ?thesis by (simp add: algebra_simps)
        next
          assume ¬ b ≤ p
          with ⟨a ≤ p⟩ 2.prem1 obtain rll rlr
            where 0: partition p rl = (rll, rlr) l' = Node l a rll r' = Node rlr
b rr
          by (auto split: tree.splits prod.splits)
          from 0 ⟨a ≤ p⟩ ⟨¬ b ≤ p⟩ 2.prem1(1) 2.IH(2)[OF _ Node, of rll
rlr]
            size_partition[OF 0(1)]
            zig_zag[where s=l and u=rr and r=rl and r1'=rll and r2'=rlr
and p=p, of a b]
            show ?thesis by (simp add: algebra_simps)
          qed
        qed
      next
        assume ¬ a ≤ p
        show ?thesis
        proof (cases l)
          case Leaf thus ?thesis using ⟨¬ a ≤ p⟩ 2.prem1 by fastforce
        next
          case [simp]: (Node ll b lr)

```

```

let ?t = Node l a r
show ?thesis
proof cases
  assume b ≤ p
  with ⟨¬ a ≤ p⟩ 2.prem1 obtain lrl lrr
    where 0: partition p lr = (lrl, lrr) l' = Node ll b lrl r' = Node lrr
a r
    by (auto split: tree.splits prod.splits)
  from 0 ⟨¬ a ≤ p⟩ ⟨b ≤ p⟩ 2.prem1 2.IH(3)[OF _ Node, of lrl
lrr]
    size_partition[OF 0(1)]
    zig_zag[where s=r and u=ll and r=lr and r1'=lrr and r2'=lrl
and p=p, of a b]
    show ?thesis by (auto simp: algebra_simps)
  next
  assume ¬ b ≤ p
  with ⟨¬ a ≤ p⟩ 2.prem1 obtain llr
    where 0: partition p ll = (l',llr) r' = Node llr b (Node lr a r)
    by (auto split: tree.splits prod.splits)
  have size llr ≤ size ll
    using size_partition[OF 0(1)] by (simp add: size1_size)
  with 0 ⟨¬ a ≤ p⟩ ⟨¬ b ≤ p⟩ 2.prem1 2.IH(4)[OF _ Node, of l'
lrr]
    zig_zig[where s=r and u=lr and r=ll and r1'=llr and r2'=l'
and p=p, of a b]
    show ?thesis by (auto simp: algebra_simps)
  qed
qed
qed
qed

```

```

fun exec :: 'a::linorder op ⇒ 'a tree list ⇒ 'a tree where
exec Empty [] = Leaf |
exec (Insert a) [t] = insert a t |
exec Del_min [t] = del_min t

```

```

fun cost :: 'a::linorder op ⇒ 'a tree list ⇒ nat where
cost Empty [] = 1 |
cost (Insert a) [t] = T_in a t |
cost Del_min [t] = T_dm t

```

```

fun U where
U Empty [] = 1 |
U (Insert _) [t] = 3 * log 2 (size1 t + 1) + 1 |

```

$U \text{ Del\_min } [t] = 2 * \varphi t + 1$

```

interpretation Amortized
where arity = arity and exec = exec and inv = bst_wrt ( $\leq$ )
and cost = cost and  $\Phi$  =  $\Phi$  and U = U
proof (standard, goal_cases)
  case (1 _ f) thus ?case
    by(cases f)
      (auto simp: insert_def bst_del_min dest!: bst_partition split: prod.splits)
next
  case (2 h) thus ?case by(induction h) (auto simp: size1_size)
next
  case (3 s f)
  show ?case
  proof (cases f)
    case Empty with 3 show ?thesis by(auto)
  next
    case Del_min with 3 show ?thesis by(auto simp: amor_del_min)
  next
    case [simp]: (Insert x)
    then obtain t where [simp]: s = [t] bst_wrt ( $\leq$ ) t using 3 by auto
    { fix l r assume 1: partition x t = (l,r)
      have  $\log 2 (1 + \text{size } t) < \log 2 (2 + \text{size } t)$  by simp
      with 1 amor_partition[OF  $\langle$ bst_wrt ( $\leq$ ) t $\rangle$  1] size_partition[OF 1]
    have ?thesis
      by(simp add: T_in_def insert_def algebra_simps size1_size
        del: log_less_cancel_iff) }
    thus ?thesis by(simp add: insert_def split: prod.split)
  qed
qed

end

```

## 7 Pairing Heaps

### 7.1 Binary Tree Representation

```

theory Pairing_Heap_Tree_Analysis
imports
  Pairing_Heap.Pairing_Heap_Tree
  Amortized_Framework
  Priority_Queue_ops_merge
  Lemmas_log
begin

```

Verification of logarithmic bounds on the amortized complexity of pairing heaps [2, 1].

```
fun len :: 'a tree  $\Rightarrow$  nat where
  len Leaf = 0
| len (Node _ _ r) = 1 + len r
```

```
fun  $\Phi$  :: 'a tree  $\Rightarrow$  real where
   $\Phi$  Leaf = 0
|  $\Phi$  (Node l x r) = log 2 (size (Node l x r)) +  $\Phi$  l +  $\Phi$  r
```

```
lemma link_size[simp]: size (link hp) = size hp
by (cases hp rule: link.cases) simp_all
```

```
lemma size_pass1: size (pass1 hp) = size hp
by (induct hp rule: pass1.induct) simp_all
```

```
lemma size_pass2: size (pass2 hp) = size hp
by (induct hp rule: pass2.induct) simp_all
```

```
lemma size_merge:
  is_root h1  $\implies$  is_root h2  $\implies$  size (merge h1 h2) = size h1 + size h2
by (simp split: tree.splits)
```

```
lemma  $\Delta\Phi$ _insert: is_root hp  $\implies$   $\Phi$  (insert x hp) -  $\Phi$  hp  $\leq$  log 2 (size
hp + 1)
by (simp split: tree.splits)
```

```
lemma  $\Delta\Phi$ _merge:
  assumes h1 = Node hs1 x1 Leaf h2 = Node hs2 x2 Leaf
  shows  $\Phi$  (merge h1 h2) -  $\Phi$  h1 -  $\Phi$  h2  $\leq$  log 2 (size h1 + size h2) + 1
proof -
  let ?hs = Node hs1 x1 (Node hs2 x2 Leaf)
  have  $\Phi$  (merge h1 h2) =  $\Phi$  (link ?hs) using assms by simp
  also have ... =  $\Phi$  hs1 +  $\Phi$  hs2 + log 2 (size hs1 + size hs2 + 1) + log
2 (size hs1 + size hs2 + 2)
  by (simp add: algebra_simps)
  also have ... =  $\Phi$  hs1 +  $\Phi$  hs2 + log 2 (size hs1 + size hs2 + 1) + log
2 (size h1 + size h2)
  using assms by simp
  finally have  $\Phi$  (merge h1 h2) = ... .
  have  $\Phi$  (merge h1 h2) -  $\Phi$  h1 -  $\Phi$  h2 =
  log 2 (size hs1 + size hs2 + 1) + log 2 (size h1 + size h2)
  - log 2 (size hs1 + 1) - log 2 (size hs2 + 1)
```

using *assms* by (*simp add: algebra\_simps*)  
 also have ...  $\leq \log 2 (\text{size } h1 + \text{size } h2) + 1$   
 using *ld\_le\_2ld*[of *size hs1 size hs2*] *assms* by (*simp add: algebra\_simps*)  
 finally show *?thesis* .  
 qed

**fun** *ub\_pass1* :: 'a tree  $\Rightarrow$  real **where**  
*ub\_pass1* (Node \_ \_ Leaf) = 0  
| *ub\_pass1* (Node *hs1* \_ (Node *hs2* \_ Leaf)) =  $2 * \log 2 (\text{size } hs1 + \text{size } hs2 + 2)$   
| *ub\_pass1* (Node *hs1* \_ (Node *hs2* \_ *hs*)) =  $2 * \log 2 (\text{size } hs1 + \text{size } hs2 + \text{size } hs + 2)$   
-  $2 * \log 2 (\text{size } hs) - 2 + \text{ub\_pass1 } hs$

**lemma**  $\Delta \Phi\_pass1\_ub\_pass1$ : *hs*  $\neq$  Leaf  $\implies \Phi (\text{pass1 } hs) - \Phi hs \leq \text{ub\_pass1 } hs$

**proof** (*induction hs rule: ub\_pass1.induct*)  
**case** ( $2 \text{ } lx \text{ } x \text{ } ly \text{ } y$ )  
**have**  $\log 2 (\text{size } lx + \text{size } ly + 1) - \log 2 (\text{size } ly + 1)$   
 $\leq \log 2 (\text{size } lx + \text{size } ly + 1)$  **by** *simp*  
**also have** ...  $\leq \log 2 (\text{size } lx + \text{size } ly + 2)$  **by** *simp*  
**also have** ...  $\leq 2 * \dots$  **by** *simp*  
**finally show** *?case* **by** (*simp add: algebra\_simps*)

**next**

**case** ( $3 \text{ } lx \text{ } x \text{ } ly \text{ } y \text{ } lz \text{ } z \text{ } rz$ )  
**let** *?ry* = Node *lz z rz*  
**let** *?rx* = Node *ly y ?ry*  
**let** *?h* = Node *lx x ?rx*  
**let** *?t* =  $\log 2 (\text{size } lx + \text{size } ly + 1) - \log 2 (\text{size } ly + \text{size } ?ry + 1)$   
**have**  $\Phi (\text{pass1 } ?h) - \Phi ?h \leq ?t + \text{ub\_pass1 } ?ry$   
**using** *3.IH* **by** (*simp add: size\_pass1 algebra\_simps*)  
**moreover have**  $\log 2 (\text{size } lx + \text{size } ly + 1) + 2 * \log 2 (\text{size } ?ry) + 2$   
 $\leq 2 * \log 2 (\text{size } ?h) + \log 2 (\text{size } ly + \text{size } ?ry + 1)$  (**is**  $?l \leq ?r$ )  
**proof** -  
**have**  $?l \leq 2 * \log 2 (\text{size } lx + \text{size } ly + \text{size } ?ry + 1) + \log 2 (\text{size } ?ry)$   
**using** *ld\_sum\_inequality* [of *size lx + size ly + 1 size ?ry*] **by** *simp*  
**also have** ...  $\leq 2 * \log 2 (\text{size } lx + \text{size } ly + \text{size } ?ry + 2) + \log 2$   
(*size ?ry*)  
**by** *simp*  
**also have** ...  $\leq ?r$  **by** *simp*  
**finally show** *?thesis* .

qed

ultimately show *?case* **by** *simp*

qed *simp\_all*

**lemma**  $\Delta\Phi_{pass1}$ : **assumes**  $hs \neq Leaf$   
**shows**  $\Phi (pass_1 hs) - \Phi hs \leq 2 * \log 2 (size hs) - len hs + 2$   
**proof** –  
**from** *assms* **have**  $ub_{pass_1} hs \leq 2 * \log 2 (size hs) - len hs + 2$   
**by** (*induct hs rule: ub\_pass1.induct*) (*simp\_all add: algebra\_simps*)  
**thus** *?thesis* **using**  $\Delta\Phi_{pass1\_ub\_pass1}$  [*OF assms*] *order\_trans* **by** *blast*  
**qed**

**lemma**  $\Delta\Phi_{pass2}$ :  $hs \neq Leaf \implies \Phi (pass_2 hs) - \Phi hs \leq \log 2 (size hs)$   
**proof** (*induction hs*)  
**case** (*Node lx x rx*)  
**thus** *?case*  
**proof** (*cases rx*)  
**case** 1: (*Node ly y ry*)  
**let** *?h* = *Node lx x rx*  
**obtain** *la a* **where** 2:  $pass_2 rx = Node\ la\ a\ Leaf$   
**using** *pass2\_struct 1* **by** *force*  
**hence** 3:  $size\ rx = size\ \dots$  **using** *size\_pass2* **by** *metis*  
**have** *link*:  $\Phi(link(Node\ lx\ x\ (pass_2\ rx))) - \Phi\ lx - \Phi (pass_2\ rx) =$   
 $\log 2 (size\ lx + size\ rx + 1) + \log 2 (size\ lx + size\ rx) - \log 2$   
 $(size\ rx)$   
**using** 2 3 **by** (*simp add: algebra\_simps*)  
**have**  $\Phi (pass_2\ ?h) - \Phi\ ?h =$   
 $\Phi (link (Node\ lx\ x\ (pass_2\ rx))) - \Phi\ lx - \Phi\ rx - \log 2 (size\ lx + size$   
 $rx + 1)$   
**by** (*simp*)  
**also have**  $\dots = \Phi (pass_2\ rx) - \Phi\ rx + \log 2 (size\ lx + size\ rx) - \log$   
 $2 (size\ rx)$   
**using** *link* **by** *linarith*  
**also have**  $\dots \leq \log 2 (size\ lx + size\ rx)$   
**using** *Node.IH 1* **by** *simp*  
**also have**  $\dots \leq \log 2 (size\ ?h)$  **using** 1 **by** *simp*  
**finally show** *?thesis* .  
**qed** *simp*  
**qed** *simp*

**lemma**  $\Delta\Phi_{del\_min}$ : **assumes**  $hs \neq Leaf$   
**shows**  $\Phi (del\_min (Node\ hs\ x\ Leaf)) - \Phi (Node\ hs\ x\ Leaf)$   
 $\leq 3 * \log 2 (size\ hs) - len\ hs + 2$   
**proof** –  
**let** *?h* = *Node hs x Leaf*  
**let**  $? \Delta\Phi_1 = \Phi\ hs - \Phi\ ?h$   
**let**  $? \Delta\Phi_2 = \Phi(pass_2(pass_1\ hs)) - \Phi\ hs$

```

let ? $\Delta\Phi$  =  $\Phi$  (del_min ?h) -  $\Phi$  ?h
have  $\Phi$ (pass2(pass1 hs)) -  $\Phi$  (pass1 hs)  $\leq$  log 2 (size hs)
  using  $\Delta\Phi$ _pass2 [of pass1 hs] assms by (metis eq_size_0 size_pass1)
moreover have  $\Phi$  (pass1 hs) -  $\Phi$  hs  $\leq$  2*... - len hs + 2
  by (rule  $\Delta\Phi$ _pass1 [OF assms])
moreover have ? $\Delta\Phi$   $\leq$  ? $\Delta\Phi_2$  by simp
ultimately show ?thesis using assms by linarith
qed

```

```

lemma is_root_merge:
  is_root h1  $\implies$  is_root h2  $\implies$  is_root (merge h1 h2)
by (simp split: tree.splits)

```

```

lemma is_root_insert: is_root h  $\implies$  is_root (insert x h)
by (simp split: tree.splits)

```

```

lemma is_root_del_min:
  assumes is_root h shows is_root (del_min h)
proof (cases h)
  case [simp]: (Node lx x rx)
  have rx = Leaf using assms by simp
  thus ?thesis
  proof (cases lx)
    case (Node ly y ry)
    then obtain la a ra where pass1 lx = Node a la ra
      using pass1_struct by blast
    moreover obtain lb b where pass2 ... = Node b lb Leaf
      using pass2_struct by blast
    ultimately show ?thesis using assms by simp
  qed simp
qed simp

```

```

lemma pass1_len: len (pass1 h)  $\leq$  len h
by (induct h rule: pass1.induct) simp_all

```

```

fun exec :: 'a :: linorder op  $\implies$  'a tree list  $\implies$  'a tree where
exec Empty [] = Leaf |
exec Del_min [h] = del_min h |
exec (Insert x) [h] = insert x h |
exec Merge [h1,h2] = merge h1 h2

```

```

fun T_pass1 :: 'a tree  $\implies$  nat where
T_pass1 Leaf = 1
| T_pass1 (Node _ _ Leaf) = 1

```



|  $T_{pass1} (Node \_ \_ (Node \_ \_ ry)) = T_{pass1} ry + 1$

**fun**  $T_{pass2} :: 'a tree \Rightarrow nat$  **where**

$T_{pass2} Leaf = 1$

|  $T_{pass2} (Node \_ \_ rx) = T_{pass2} rx + 1$

**fun**  $cost :: 'a :: linorder op \Rightarrow 'a tree list \Rightarrow nat$  **where**

$cost Empty [] = 1$

|  $cost Del\_min [Leaf] = 1$

|  $cost Del\_min [Node lx \_ \_] = T_{pass2} (pass1 lx) + T_{pass1} lx$

|  $cost (Insert a) \_ = 1$

|  $cost Merge \_ = 1$

**fun**  $U :: 'a :: linorder op \Rightarrow 'a tree list \Rightarrow real$  **where**

$U Empty [] = 1$

|  $U (Insert a) [h] = \log 2 (size h + 1) + 1$

|  $U Del\_min [h] = 3 * \log 2 (size h + 1) + 4$

|  $U Merge [h1, h2] = \log 2 (size h1 + size h2 + 1) + 2$

**interpretation** *Amortized*

**where**  $arity = arity$  **and**  $exec = exec$  **and**  $cost = cost$  **and**  $inv = is\_root$

**and**  $\Phi = \Phi$  **and**  $U = U$

**proof** (*standard, goal\_cases*)

**case** ( $1\_f$ ) **thus**  $?case$  **using**  $is\_root$   $insert$   $is\_root\_del\_min$   $is\_root\_merge$   
**by** (*cases f*) (*auto simp: numeral\_eq\_Suc*)

**next**

**case** ( $2\ s$ ) **show**  $?case$  **by** (*induct s*) *simp\_all*

**next**

**case** ( $3\ ss\ f$ ) **show**  $?case$

**proof** (*cases f*)

**case**  $Empty$  **with**  $3$  **show**  $?thesis$  **by** (*auto*)

**next**

**case** ( $Insert\ x$ )

**then obtain**  $h$  **where**  $ss = [h]$   $is\_root\ h$  **using**  $3$  **by** *auto*

**thus**  $?thesis$  **using**  $Insert\ \Delta\Phi\_insert\ 3$  **by** *auto*

**next**

**case** [*simp*]: ( $Del\_min$ )

**then obtain**  $h$  **where** [*simp*]:  $ss = [h]$  **using**  $3$  **by** *auto*

**show**  $?thesis$

**proof** (*cases h*)

**case** [*simp*]: ( $Node\ lx\ x\ rx$ )

**have**  $T_{pass2} (pass1\ lx) + T_{pass1}\ lx \leq len\ lx + 2$

**by** (*induct lx rule: pass1.induct*) *simp\_all*

**hence**  $cost\ f\ ss \leq \dots$  **by** *simp*

```

    moreover have  $\Phi (\text{del\_min } h) - \Phi h \leq 3 * \log 2 (\text{size } h + 1) - \text{len } lx + 2$ 
  proof (cases lx)
    case [simp]: (Node ly y ry)
      have  $\Phi (\text{del\_min } h) - \Phi h \leq 3 * \log 2 (\text{size } lx) - \text{len } lx + 2$ 
        using  $\Delta\Phi\_del\_min[\text{of } lx \ x] \ 3$  by simp
      also have  $\dots \leq 3 * \log 2 (\text{size } h + 1) - \text{len } lx + 2$  by fastforce
      finally show ?thesis by blast
    qed (insert 3, simp)
  ultimately show ?thesis by auto
qed simp
next
case [simp]: Merge
  then obtain h1 h2 where [simp]:  $ss = [h1, h2]$  and 1: is_root h1
is_root h2
  using 3 by (auto simp: numeral_eq_Suc)
show ?thesis
proof (cases h1)
  case Leaf thus ?thesis by (cases h2) auto
next
  case h1: Node
  show ?thesis
  proof (cases h2)
    case Leaf thus ?thesis using h1 by simp
  next
    case h2: Node
    have  $\Phi (\text{merge } h1 \ h2) - \Phi h1 - \Phi h2 \leq \log 2 (\text{real } (\text{size } h1 + \text{size } h2)) + 1$ 
      apply (rule  $\Delta\Phi\_merge$ ) using h1 h2 1 by auto
    also have  $\dots \leq \log 2 (\text{size } h1 + \text{size } h2 + 1) + 1$  by (simp add: h1)
    finally show ?thesis by (simp add: algebra_simps)
  qed
qed
qed
qed
end

```

## 8 Pairing Heaps

### 8.1 Binary Tree Representation

theory *Pairing\_Heap\_Tree\_Analysis2*

**imports**

*Pairing\_Heap.Pairing\_Heap\_Tree*

*Amortized\_Framework*

*Priority\_Queue\_ops\_merge*

*Lemmas\_log*

**begin**

Verification of logarithmic bounds on the amortized complexity of pairing heaps. As in [2, 1], except that the treatment of *pass*<sub>1</sub> is simplified. TODO: convert the other Pairing Heap analyses to this one.

**fun** *len* :: 'a tree  $\Rightarrow$  nat **where**

*len* Leaf = 0

| *len* (Node \_ \_ r) = 1 + *len* r

**fun**  $\Phi$  :: 'a tree  $\Rightarrow$  real **where**

$\Phi$  Leaf = 0

|  $\Phi$  (Node l x r) = log 2 (size (Node l x r)) +  $\Phi$  l +  $\Phi$  r

**lemma** *link\_size[simp]*: size (link hp) = size hp

**by** (cases hp rule: link.cases) simp\_all

**lemma** *size\_pass1*: size (pass<sub>1</sub> hp) = size hp

**by** (induct hp rule: pass<sub>1</sub>.induct) simp\_all

**lemma** *size\_pass2*: size (pass<sub>2</sub> hp) = size hp

**by** (induct hp rule: pass<sub>2</sub>.induct) simp\_all

**lemma** *size\_merge*:

*is\_root* h1  $\implies$  *is\_root* h2  $\implies$  size (merge h1 h2) = size h1 + size h2

**by** (simp split: tree.splits)

**lemma**  $\Delta\Phi$ \_insert: *is\_root* hp  $\implies$   $\Phi$  (insert x hp) -  $\Phi$  hp  $\leq$  log 2 (size hp + 1)

**by** (simp split: tree.splits)

**lemma**  $\Delta\Phi$ \_merge:

**assumes** h1 = Node hs1 x1 Leaf h2 = Node hs2 x2 Leaf

**shows**  $\Phi$  (merge h1 h2) -  $\Phi$  h1 -  $\Phi$  h2  $\leq$  log 2 (size h1 + size h2) + 1

**proof** -

**let** ?hs = Node hs1 x1 (Node hs2 x2 Leaf)

**have**  $\Phi$  (merge h1 h2) =  $\Phi$  (link ?hs) **using** assms **by** simp

**also have** ... =  $\Phi$  hs1 +  $\Phi$  hs2 + log 2 (size hs1 + size hs2 + 1) + log 2 (size hs1 + size hs2 + 2)

**by** (simp add: algebra\_simps)

**also have**  $\dots = \Phi \text{ hs1} + \Phi \text{ hs2} + \log 2 (\text{size hs1} + \text{size hs2} + 1) + \log 2 (\text{size h1} + \text{size h2})$   
**using** *assms by simp*  
**finally have**  $\Phi (\text{merge h1 h2}) = \dots$   
**have**  $\Phi (\text{merge h1 h2}) - \Phi \text{ h1} - \Phi \text{ h2} =$   
 $\log 2 (\text{size hs1} + \text{size hs2} + 1) + \log 2 (\text{size h1} + \text{size h2})$   
 $- \log 2 (\text{size hs1} + 1) - \log 2 (\text{size hs2} + 1)$   
**using** *assms by (simp add: algebra\_simps)*  
**also have**  $\dots \leq \log 2 (\text{size h1} + \text{size h2}) + 1$   
**using** *ld\_le\_2ld[of size hs1 size hs2] assms by (simp add: algebra\_simps)*  
**finally show** *?thesis* .  
**qed**

**lemma**  $\Delta\Phi\_pass1: \Phi (\text{pass}_1 \text{ hs}) - \Phi \text{ hs} \leq 2 * \log 2 (\text{size hs} + 1) - \text{len hs} + 2$   
**proof** (*induction hs rule: pass1.induct*)  
**case**  $(1 \text{ hs1 } x \text{ hs2 } y \text{ hs})$   
**let**  $?h = \text{Node hs1 } x (\text{Node hs2 } y \text{ hs})$   
**let**  $?n1 = \text{size hs1}$   
**let**  $?n2 = \text{size hs2}$  **let**  $?m = \text{size hs}$   
**have**  $\Phi (\text{pass}_1 ?h) - \Phi ?h = \Phi (\text{pass}_1 \text{ hs}) + \log 2 (?n1 + ?n2 + 1) - \Phi \text{ hs}$   
 $- \log 2 (?n2 + ?m + 1)$   
**by** (*simp add: size\_pass1 algebra\_simps*)  
**also have**  $\dots \leq \log 2 (?n1 + ?n2 + 1) - \log 2 (?n2 + ?m + 1) + 2 * \log 2$   
 $(?m + 1) - \text{len hs} + 2$   
**using** *1.IH by (simp)*  
**also have**  $\dots \leq 2 * \log 2 (?n1 + ?n2 + ?m + 2) - \log 2 (?n2 + ?m + 1) +$   
 $\log 2 (?m + 1) - \text{len hs}$   
**using** *ld\_sum\_inequality [of ?n1 + ?n2 + 1 ?m + 1] by (simp)*  
**also have**  $\dots \leq 2 * \log 2 (?n1 + ?n2 + ?m + 2) - \text{len hs}$  **by** *simp*  
**also have**  $\dots = 2 * \log 2 (\text{size ?h}) - \text{len ?h} + 2$  **by** *simp*  
**also have**  $\dots \leq 2 * \log 2 (\text{size ?h} + 1) - \text{len ?h} + 2$  **by** *simp*  
**finally show** *?case* .  
**qed** *simp\_all*

**lemma**  $\Delta\Phi\_pass2: \text{hs} \neq \text{Leaf} \implies \Phi (\text{pass}_2 \text{ hs}) - \Phi \text{ hs} \leq \log 2 (\text{size hs})$   
**proof** (*induction hs*)  
**case**  $(\text{Node hs1 } x \text{ hs})$   
**thus** *?case*  
**proof** (*cases hs*)  
**case** *1: (Node hs2 y r)*  
**let**  $?h = \text{Node hs1 } x \text{ hs}$   
**obtain**  $\text{hs3 } a$  **where** *2: pass2 hs = Node hs3 a Leaf*  
**using** *pass2\_struct 1 by force*

**hence**  $\exists$ :  $\text{size } hs = \text{size } \dots$  **using**  $\text{size\_pass}_2$  **by**  $\text{metis}$   
**have**  $\text{link}$ :  $\Phi(\text{link}(\text{Node } hs1 \ x \ (\text{pass}_2 \ hs))) - \Phi \ hs1 - \Phi (\text{pass}_2 \ hs) =$   
 $\log 2 (\text{size } hs1 + \text{size } hs + 1) + \log 2 (\text{size } hs1 + \text{size } hs) - \log 2$   
 $(\text{size } hs)$   
**using**  $2 \ 3$  **by**  $(\text{simp add: algebra\_simps})$   
**have**  $\Phi (\text{pass}_2 \ ?h) - \Phi \ ?h =$   
 $\Phi (\text{link} (\text{Node } hs1 \ x \ (\text{pass}_2 \ hs))) - \Phi \ hs1 - \Phi \ hs - \log 2 (\text{size } hs1$   
 $+ \text{size } hs + 1)$   
**by**  $(\text{simp})$   
**also have**  $\dots = \Phi (\text{pass}_2 \ hs) - \Phi \ hs + \log 2 (\text{size } hs1 + \text{size } hs) - \log$   
 $2 (\text{size } hs)$   
**using**  $\text{link}$  **by**  $\text{linarith}$   
**also have**  $\dots \leq \log 2 (\text{size } hs1 + \text{size } hs)$   
**using**  $\text{Node.IH}(2) \ 1$  **by**  $\text{simp}$   
**also have**  $\dots \leq \log 2 (\text{size } ?h)$  **using**  $1$  **by**  $\text{simp}$   
**finally show**  $?thesis$  .  
**qed**  $\text{simp}$   
**qed**  $\text{simp}$

**corollary**  $\Delta\Phi_{\text{pass}_2}$ :  $\Phi (\text{pass}_2 \ hs) - \Phi \ hs \leq \log 2 (\text{size } hs + 1)$

**proof**  $\text{cases}$

**assume**  $hs = \text{Leaf}$  **thus**  $?thesis$  **by**  $\text{simp}$

**next**

**assume**  $hs \neq \text{Leaf}$

**hence**  $\log 2 (\text{size } hs) \leq \log 2 (\text{size } hs + 1)$  **using**  $\text{eq\_size}_0$  **by**  $\text{fastforce}$

**then show**  $?thesis$  **using**  $\Delta\Phi_{\text{pass}_2}[\text{OF } \langle hs \neq \text{Leaf} \rangle]$  **by**  $\text{linarith}$

**qed**

**lemma**  $\Delta\Phi_{\text{del\_min}}$ :

$\Phi (\text{del\_min} (\text{Node } hs \ x \ \text{Leaf})) - \Phi (\text{Node } hs \ x \ \text{Leaf})$

$\leq 2 * \log 2 (\text{size } hs + 1) - \text{len } hs + 2$

**proof** –

**have**  $\Phi (\text{del\_min} (\text{Node } hs \ x \ \text{Leaf})) - \Phi (\text{Node } hs \ x \ \text{Leaf}) =$

$\Phi (\text{pass}_2 (\text{pass}_1 \ hs)) - (\log 2 (1 + \text{real } (\text{size } hs)) + \Phi \ hs)$  **by**  $\text{simp}$

**also have**  $\dots \leq \Phi (\text{pass}_1 \ hs) - \Phi \ hs$

**using**  $\Delta\Phi_{\text{pass}_2}' [\text{of } \text{pass}_1 \ hs]$  **by**  $(\text{simp add: size\_pass}_1)$

**also have**  $\dots \leq 2 * \log 2 (\text{size } hs + 1) - \text{len } hs + 2$  **by**  $(\text{rule } \Delta\Phi_{\text{pass}_1})$

**finally show**  $?thesis$  .

**qed**

**lemma**  $\text{is\_root\_merge}$ :

$\text{is\_root } h1 \implies \text{is\_root } h2 \implies \text{is\_root } (\text{merge } h1 \ h2)$

**by**  $(\text{simp split: tree.splits})$

**lemma** *is\_root\_insert*: *is\_root h*  $\implies$  *is\_root (insert x h)*  
**by** (*simp split: tree.splits*)

**lemma** *is\_root\_del\_min*:  
**assumes** *is\_root h* **shows** *is\_root (del\_min h)*  
**proof** (*cases h*)  
**case** [*simp*]: (*Node hs1 x hs*)  
**have** *hs = Leaf* **using** *assms* **by** *simp*  
**thus** *?thesis*  
**proof** (*cases hs1*)  
**case** (*Node hs2 y hs'*)  
**then obtain** *la a ra* **where** *pass<sub>1</sub> hs1 = Node a la ra*  
**using** *pass<sub>1</sub>\_struct* **by** *blast*  
**moreover obtain** *lb b* **where** *pass<sub>2</sub> ... = Node b lb Leaf*  
**using** *pass<sub>2</sub>\_struct* **by** *blast*  
**ultimately show** *?thesis* **using** *assms* **by** *simp*  
**qed** *simp*  
**qed** *simp*

**lemma** *pass<sub>1</sub>\_len*: *len (pass<sub>1</sub> h)*  $\leq$  *len h*  
**by** (*induct h rule: pass<sub>1</sub>.induct*) *simp\_all*

**fun** *exec* :: '*a* :: *linorder* *op*  $\Rightarrow$  '*a* *tree list*  $\Rightarrow$  '*a* *tree* **where**  
*exec Empty [] = Leaf* |  
*exec Del\_min [h] = del\_min h* |  
*exec (Insert x) [h] = insert x h* |  
*exec Merge [h1,h2] = merge h1 h2*

**fun** *T\_pass<sub>1</sub>* :: '*a* *tree*  $\Rightarrow$  *nat* **where**  
*T\_pass<sub>1</sub> (Node \_ \_ (Node \_ \_ hs')) = T\_pass<sub>1</sub> hs' + 1* |  
*T\_pass<sub>1</sub> h = 1*

**fun** *T\_pass<sub>2</sub>* :: '*a* *tree*  $\Rightarrow$  *nat* **where**  
*T\_pass<sub>2</sub> Leaf = 1*  
| *T\_pass<sub>2</sub> (Node \_ \_ hs) = T\_pass<sub>2</sub> hs + 1*

**fun** *T\_del\_min* :: ('*a*::*linorder*) *tree*  $\Rightarrow$  *nat* **where**  
*T\_del\_min Leaf = 1* |  
*T\_del\_min (Node hs \_ \_) = T\_pass<sub>2</sub> (pass<sub>1</sub> hs) + T\_pass<sub>1</sub> hs + 1*

**fun** *T\_insert* :: '*a*  $\Rightarrow$  '*a* *tree*  $\Rightarrow$  *nat* **where**  
*T\_insert a h = 1*

**fun** *T\_merge* :: '*a* *tree*  $\Rightarrow$  '*a* *tree*  $\Rightarrow$  *nat* **where**

$T\_merge\ h1\ h2 = 1$

**lemma**  $A\_del\_min$ : **assumes**  $is\_root\ h$   
**shows**  $T\_del\_min\ h + \Phi(del\_min\ h) - \Phi\ h \leq 2 * \log\ 2\ (size\ h + 1) + 5$   
**proof** ( $cases\ h$ )  
  **case** [ $simp$ ]: ( $Node\ hs1\ x\ hs$ )  
  **have**  $T\_pass2\ (pass1\ hs1) + real(T\_pass1\ hs1) \leq real(len\ hs1) + 2$   
  **by** ( $induct\ hs1\ rule: pass1.induct$ )  $simp\_all$   
  **moreover have**  $\Phi\ (del\_min\ h) - \Phi\ h \leq 2 * \log\ 2\ (size\ h + 1) - len\ hs1 + 2$   
  **proof** -  
  **have**  $\Phi\ (del\_min\ h) - \Phi\ h \leq 2 * \log\ 2\ (size\ hs1 + 1) - len\ hs1 + 2$   
  **using**  $\Delta\Phi\_del\_min[of\ hs1\ x]$  **assms** **by**  $simp$   
  **also have**  $\dots \leq 2 * \log\ 2\ (size\ h + 1) - len\ hs1 + 2$  **by**  $fastforce$   
  **finally show**  $?thesis$  .  
  **qed**  
  **ultimately show**  $?thesis$  **by**( $simp$ )  
**qed**  $simp$

**lemma**  $A\_insert$ :  $is\_root\ h \implies T\_insert\ a\ h + \Phi(insert\ a\ h) - \Phi\ h \leq \log\ 2\ (size\ h + 1) + 1$   
**by**( $drule\ \Delta\Phi\_insert$ )  $simp$

**lemma**  $A\_merge$ : **assumes**  $is\_root\ h1\ is\_root\ h2$   
**shows**  $T\_merge\ h1\ h2 + \Phi(merge\ h1\ h2) - \Phi\ h1 - \Phi\ h2 \leq \log\ 2\ (size\ h1 + size\ h2 + 1) + 2$   
**proof** ( $cases\ h1$ )  
  **case**  $Leaf$  **thus**  $?thesis$  **by** ( $cases\ h2$ )  $auto$   
**next**  
  **case**  $h1: Node$   
  **show**  $?thesis$   
  **proof** ( $cases\ h2$ )  
    **case**  $Leaf$  **thus**  $?thesis$  **using**  $h1$  **by**  $simp$   
  **next**  
    **case**  $h2: Node$   
    **have**  $\Phi\ (merge\ h1\ h2) - \Phi\ h1 - \Phi\ h2 \leq \log\ 2\ (real\ (size\ h1 + size\ h2)) + 1$   
    **apply**( $rule\ \Delta\Phi\_merge$ ) **using**  $h1\ h2$  **assms** **by**  $auto$   
    **also have**  $\dots \leq \log\ 2\ (size\ h1 + size\ h2 + 1) + 1$  **by** ( $simp\ add: h1$ )  
    **finally show**  $?thesis$  **by**( $simp\ add: algebra\_simps$ )  
  **qed**  
**qed**

**fun**  $cost :: 'a :: linorder\ op \Rightarrow 'a\ tree\ list \Rightarrow nat$  **where**

```

  cost Empty [] = 1
| cost Del_min [h] = T_del_min h
| cost (Insert a) [h] = T_insert a h
| cost Merge [h1,h2] = T_merge h1 h2

```

**fun**  $U :: 'a :: \text{linorder } \text{op} \Rightarrow 'a \text{ tree list} \Rightarrow \text{real}$  **where**

```

  U Empty [] = 1
| U (Insert a) [h] = log 2 (size h + 1) + 1
| U Del_min [h] = 2 * log 2 (size h + 1) + 5
| U Merge [h1,h2] = log 2 (size h1 + size h2 + 1) + 2

```

**interpretation** *Amortized*

**where**  $\text{arity} = \text{arity}$  **and**  $\text{exec} = \text{exec}$  **and**  $\text{cost} = \text{cost}$  **and**  $\text{inv} = \text{is\_root}$

**and**  $\Phi = \Phi$  **and**  $U = U$

**proof** (*standard, goal\_cases*)

**case** ( $1\_f$ ) **thus** *?case using is\_root insert is\_root\_del\_min is\_root\_merge*  
**by** (*cases f*) (*auto simp: numeral\_eq\_Suc*)

**next**

**case** ( $2\ s$ ) **show** *?case by (induct s) simp\_all*

**next**

**case** ( $3\ ss\ f$ ) **show** *?case*

**proof** (*cases f*)

**case** *Empty with 3* **show** *?thesis by(auto)*

**next**

**case** *Insert*

**then obtain**  $h$  **where**  $ss = [h]$  *is\_root h using 3 by auto*

**thus** *?thesis using Insert  $\Delta\Phi_{\text{insert}}$  3 by auto*

**next**

**case** *Del\_min*

**then obtain**  $h$  **where** [*simp*]:  $ss = [h]$  *using 3 by auto*

**show** *?thesis using A\_del\_min[of h] 3 Del\_min by simp*

**next**

**case** *Merge*

**then obtain**  $h1\ h2$  **where**  $ss = [h1,h2]$  *is\_root h1 is\_root h2*

**using**  $3$  **by** (*auto simp: numeral\_eq\_Suc*)

**with**  $A_{\text{merge}}$ [of  $h1\ h2$ ] *Merge* **show** *?thesis by simp*

**qed**

**qed**

**end**

## 8.2 Okasaki's Pairing Heap

**theory** *Pairing\_Heap\_List1\_Analysis*



**imports**

*Pairing\_Heap.Pairing\_Heap\_List1*

*Amortized\_Framework*

*Priority\_Queue\_ops\_merge*

*Lemmas\_log*

**begin**

Amortized analysis of pairing heaps as defined by Okasaki [6].

**fun** *hps* **where**

*hps* (*Hp* \_ *hs*) = *hs*

**lemma** *merge\_Empty[simp]*: *merge heap.Empty h = h*

**by**(*cases h*) *auto*

**lemma** *merge2*: *merge (Hp x lx) h = (case h of heap.Empty  $\Rightarrow$  Hp x lx | (Hp y ly)  $\Rightarrow$*

*(if x < y then Hp x (Hp y ly # lx) else Hp y (Hp x lx # ly)))*

**by**(*auto split: heap.split*)

**lemma** *pass1\_Nil\_iff*: *pass<sub>1</sub> hs = []  $\longleftrightarrow$  hs = []*

**by**(*cases hs rule: pass<sub>1</sub>.cases*) *auto*

### 8.2.1 Invariant

**fun** *no\_Empty* :: '*a* :: *linorder heap*  $\Rightarrow$  *bool* **where**

*no\_Empty heap.Empty = False* |

*no\_Empty (Hp x hs) = ( $\forall h \in \text{set } hs. \text{no\_Empty } h$ )*

**abbreviation** *no\_Emptys* :: '*a* :: *linorder heap list*  $\Rightarrow$  *bool* **where**

*no\_Emptys hs  $\equiv$   $\forall h \in \text{set } hs. \text{no\_Empty } h$*

**fun** *is\_root* :: '*a* :: *linorder heap*  $\Rightarrow$  *bool* **where**

*is\_root heap.Empty = True* |

*is\_root (Hp x hs) = no\_Emptys hs*

**lemma** *is\_root\_if\_no\_Empty*: *no\_Empty h  $\Longrightarrow$  is\_root h*

**by**(*cases h*) *auto*

**lemma** *no\_Emptys\_hps*: *no\_Empty h  $\Longrightarrow$  no\_Emptys(hps h)*

**by**(*induction h*) *auto*

**lemma** *no\_Empty\_merge*:  $\llbracket \text{no\_Empty } h1; \text{no\_Empty } h2 \rrbracket \Longrightarrow \text{no\_Empty } (\text{merge } h1 \ h2)$

**by** (cases (h1,h2) rule: merge.cases) auto

**lemma** *is\_root\_merge*:  $\llbracket \text{is\_root } h1; \text{is\_root } h2 \rrbracket \implies \text{is\_root } (\text{merge } h1 \text{ } h2)$

**by** (cases (h1,h2) rule: merge.cases) auto

**lemma** *no\_Emptyys\_pass1*:

$\text{no\_Emptyys } hs \implies \text{no\_Emptyys } (\text{pass}_1 \text{ } hs)$

**by**(induction hs rule: pass<sub>1</sub>.induct)(auto simp: no\_Empty\_merge)

**lemma** *is\_root\_pass2*:  $\text{no\_Emptyys } hs \implies \text{is\_root}(\text{pass}_2 \text{ } hs)$

**proof**(induction hs)

**case** (Cons \_ hs)

**show** ?case

**proof** cases

**assume**  $hs = []$  **thus** ?thesis **using** Cons **by** (auto simp: is\_root\_if\_no\_Empty)

**next**

**assume**  $hs \neq []$  **thus** ?thesis **using** Cons **by**(auto simp: is\_root\_merge is\_root\_if\_no\_Empty)

**qed**

**qed** simp

## 8.2.2 Complexity

**fun** *size\_hp* :: 'a heap  $\Rightarrow$  nat **where**

*size\_hp* heap.Empty = 0 |

*size\_hp* (Hp x hs) = sum\_list(map *size\_hp* hs) + 1

**abbreviation** *size\_hps* **where**

*size\_hps* hs  $\equiv$  sum\_list(map *size\_hp* hs)

**fun**  $\Phi$ \_hps :: 'a heap list  $\Rightarrow$  real **where**

$\Phi$ \_hps [] = 0 |

$\Phi$ \_hps (heap.Empty # hs) =  $\Phi$ \_hps hs |

$\Phi$ \_hps (Hp x hsl # hsr) =

$\Phi$ \_hps hsl +  $\Phi$ \_hps hsr + log 2 (size\_hps hsl + size\_hps hsr + 1)

**fun**  $\Phi$  :: 'a heap  $\Rightarrow$  real **where**

$\Phi$  heap.Empty = 0 |

$\Phi$  (Hp \_ hs) =  $\Phi$ \_hps hs + log 2 (size\_hps hs + 1)

**lemma**  $\Phi$ \_hps\_ge0:  $\Phi$ \_hps hs  $\geq$  0

**by** (induction hs rule:  $\Phi$ \_hps.induct) auto

**lemma** *no\_Empty\_ge0*: *no\_Empty h*  $\implies$  *size\_hp h* > 0  
**by**(*cases h*) *auto*

**declare** *algebra\_simps*[*simp*]

**lemma**  $\Phi\_hps1$ :  $\Phi\_hps [h] = \Phi h$   
**by**(*cases h*) *auto*

**lemma** *size\_hp\_merge*: *size\_hp*(*merge h1 h2*) = *size\_hp h1* + *size\_hp h2*

**by** (*induction h1 h2 rule: merge.induct*) *simp\_all*

**lemma** *pass1\_size*[*simp*]: *size\_hps* (*pass1 hs*) = *size\_hps hs*  
**by** (*induct hs rule: pass1.induct*) (*simp\_all add: size\_hp\_merge*)

**lemma**  $\Delta\Phi\_insert$ :

$\Phi$  (*Pairing\_Heap\_List1.insert x h*) -  $\Phi h \leq \log 2$  (*size\_hp h* + 1)  
**by**(*cases h*)(*auto simp: size\_hp\_merge*)

**lemma**  $\Delta\Phi\_merge$ :

$\Phi$  (*merge h1 h2*) -  $\Phi h1$  -  $\Phi h2$   
 $\leq \log 2$  (*size\_hp h1* + *size\_hp h2* + 1) + 1  
**proof**(*induction h1 h2 rule: merge.induct*)  
**case** ( $\exists x lx y ly$ )  
**thus** *?case*  
**using** *ld\_le\_2ld*[*of size\_hps lx size\_hps ly*]  
*log\_le\_cancel\_iff*[*of 2 size\_hps lx + size\_hps ly + 2 size\_hps lx +*  
*size\_hps ly + 3*]  
**by** (*auto simp del: log\_le\_cancel\_iff*)  
**qed** *auto*

**fun** *sum\_ub* :: 'a *heap list*  $\Rightarrow$  *real* **where**

*sum\_ub* [] = 0  
| *sum\_ub* [\_] = 0  
| *sum\_ub* [*h1*, *h2*] =  $2 * \log 2$  (*size\_hp h1* + *size\_hp h2*)  
| *sum\_ub* (*h1* # *h2* # *hs*) =  $2 * \log 2$  (*size\_hp h1* + *size\_hp h2* + *size\_hps*  
*hs*)  
-  $2 * \log 2$  (*size\_hps hs*) - 2 + *sum\_ub hs*

**lemma**  $\Delta\Phi\_pass1\_sum\_ub$ : *no\_Emptys hs*  $\implies$

$\Phi\_hps$  (*pass1 hs*) -  $\Phi\_hps hs \leq$  *sum\_ub hs* (**is** \_  $\implies$  *?P hs*)

**proof** (*induction hs rule: sum\_ub.induct*)

**case** ( $\exists h1 h2$ )

**then obtain** *x hsx y hsy* **where** [*simp*]: *h1* = *Hp x hsx h2* = *Hp y hsy*

```

    by simp (meson no_Empty.elims(2))
  have 0:  $\bigwedge x y :: \text{real}. 0 \leq x \implies x \leq y \implies x \leq 2 * y$  by linarith
  show ?case using 3 by (auto simp add: add_increasing 0)
next
  case (4 h1 h2 h3 hs)
  hence IH: ?P(h3#hs) by auto
  from 4.prem1 obtain x hsx y hsy where [simp]: h1 = Hp x hsx h2 = Hp
y hsy
    by simp (meson no_Empty.elims(2))
  from 4.prem2 have s3: size_hp h3 > 0
    apply auto using size_hp.elims by force
  let ?ry = h3 # hs
  let ?rx = Hp y hsy # ?ry
  let ?h = Hp x hsx # ?rx
  have  $\Phi\_hps(\text{pass}_1 \ ?h) - \Phi\_hps \ ?h$ 
     $\leq \log 2 (1 + \text{size\_hps } hsx + \text{size\_hps } hsy) - \log 2 (1 + \text{size\_hps } hsy$ 
+  $\text{size\_hps } ?ry) + \text{sum\_ub } ?ry$ 
    using IH by simp
  also have  $\log 2 (1 + \text{size\_hps } hsx + \text{size\_hps } hsy) - \log 2 (1 + \text{size\_hps } hsy$ 
+  $\text{size\_hps } ?ry)$ 
     $\leq 2 * \log 2 (\text{size\_hps } ?h) - 2 * \log 2 (\text{size\_hps } ?ry) - 2$ 
  proof -
    have  $\log 2 (1 + \text{size\_hps } hsx + \text{size\_hps } hsy) + \log 2 (\text{size\_hps } ?ry)$ 
-  $2 * \log 2 (\text{size\_hps } ?h)$ 
    =  $\log 2 ((1 + \text{size\_hps } hsx + \text{size\_hps } hsy) / (\text{size\_hps } ?h)) + \log 2$ 
( $\text{size\_hps } ?ry / \text{size\_hps } ?h$ )
    using s3 by (simp add: log_divide)
    also have ...  $\leq -2$ 
  proof -
    have  $2 + \dots$ 
     $\leq 2 * \log 2 ((1 + \text{size\_hps } hsx + \text{size\_hps } hsy) / \text{size\_hps } ?h +$ 
 $\text{size\_hps } ?ry / \text{size\_hps } ?h)$ 
    using ld_sum_inequality [of  $(1 + \text{size\_hps } hsx + \text{size\_hps } hsy) /$ 
 $\text{size\_hps } ?h$   $(\text{size\_hps } ?ry / \text{size\_hps } ?h)$ ] using s3 by simp
    also have ...  $\leq 0$  by (simp add: field_simps log_divide add_pos_nonneg)
    finally show ?thesis by linarith
  qed
  finally have  $\log 2 (1 + \text{size\_hps } hsx + \text{size\_hps } hsy) + \log 2 (\text{size\_hps } ?ry) + 2$ 
     $\leq 2 * \log 2 (\text{size\_hps } ?h)$  by simp
  moreover have  $\log 2 (\text{size\_hps } ?ry) \leq \log 2 (\text{size\_hps } ?rx)$  using s3
by simp
  ultimately have  $\log 2 (1 + \text{size\_hps } hsx + \text{size\_hps } hsy) - \dots$ 
     $\leq 2 * \log 2 (\text{size\_hps } ?h) - 2 * \log 2 (\text{size\_hps } ?ry) - 2$  by linarith

```

```

    thus ?thesis by simp
  qed
  finally show ?case by (simp)
qed simp_all

lemma  $\Delta\Phi\_pass1$ : assumes  $hs \neq []$  no_Emptys  $hs$ 
  shows  $\Phi\_hps (pass_1\ hs) - \Phi\_hps\ hs \leq 2 * \log 2 (size\_hps\ hs) - length\ hs + 2$ 
proof -
  have  $sum\_ub\ hs \leq 2 * \log 2 (size\_hps\ hs) - length\ hs + 2$ 
  using assms by (induct  $hs$  rule: sum_ub.induct) (auto dest: no_Empty_ge0)
  thus ?thesis using  $\Delta\Phi\_pass1\_sum\_ub$ [OF assms(2)] by linarith
qed

lemma size_hps_pass2:  $hs \neq [] \implies no\_Empty\_s\ hs \implies$ 
   $no\_Empty(pass_2\ hs) \ \&\ \ size\_hps\ hs = size\_hps(hps(pass_2\ hs))+1$ 
apply (induction  $hs$  rule:  $\Phi\_hps.induct$ )
  apply (fastforce simp: merge2 split: heap.split)+
done

lemma  $\Delta\Phi\_pass2$ :  $hs \neq [] \implies no\_Empty\_s\ hs \implies$ 
   $\Phi (pass_2\ hs) - \Phi\_hps\ hs \leq \log 2 (size\_hps\ hs)$ 
proof (induction  $hs$ )
  case (Cons  $h\ hs$ )
  thus ?case
  proof cases
    assume  $hs = []$ 
    thus ?thesis using Cons by (auto simp add:  $\Phi\_hps1$  dest: no_Empty_ge0)
  next
    assume *:  $hs \neq []$ 
    obtain  $x\ hs2$  where [simp]:  $h = Hp\ x\ hs2$ 
      using Cons.prems(2) by simp (meson no_Empty.elims(2))
    show ?thesis
    proof (cases  $pass_2\ hs$ )
      case Empty thus ?thesis using  $\Phi\_hps\_ge0$ [of  $hs$ ]
        by (simp add: add_increasing xt1(3)[OF mult_2, add_increasing])
    next
      case (Hp  $y\ hs3$ )
        with Cons * size_hps_pass2[of  $hs$ ] show ?thesis by (auto simp:
add_mono)
    qed
  qed
qed
qed simp

```

**lemma**  $\Delta\Phi\_del\_min$ : **assumes**  $hps\ h \neq []\ no\_Empty\ h$   
**shows**  $\Phi\ (del\_min\ h) - \Phi\ h$   
 $\leq 3 * \log\ 2\ (size\_hps(hps\ h)) - length(hps\ h) + 2$   
**proof** –  
**obtain**  $x\ hs$  **where**  $[simp]: h = Hp\ x\ hs$  **using**  $assms(2)$  **by**  $(cases\ h)$   
*auto*  
**let**  $?\Delta\Phi_1 = \Phi\_hps(hps\ h) - \Phi\ h$   
**let**  $?\Delta\Phi_2 = \Phi(pass_2(pass_1(hps\ h))) - \Phi\_hps(hps\ h)$   
**let**  $?\Delta\Phi = \Phi\ (del\_min\ h) - \Phi\ h$   
**have**  $\Phi(pass_2(pass_1(hps\ h))) - \Phi\_hps(pass_1(hps\ h)) \leq \log\ 2\ (size\_hps(hps\ h))$   
**using**  $\Delta\Phi\_pass2[of\ pass_1(hps\ h)]$  **using**  $assms$   
**by**  $(auto\ simp: pass1\_Nil\_iff\ no\_Emptyys\_pass1\ dest: no\_Emptyys\_hps)$   
**moreover have**  $\Phi\_hps(pass_1(hps\ h)) - \Phi\_hps(hps\ h) \leq 2 * \dots - length(hps\ h) + 2$   
**using**  $\Delta\Phi\_pass1[OF\ assms(1)\ no\_Emptyys\_hps[OF\ assms(2)]]$  **by**  $blast$   
**moreover have**  $?\Delta\Phi_1 \leq 0$  **by**  $simp$   
**moreover have**  $?\Delta\Phi = ?\Delta\Phi_1 + ?\Delta\Phi_2$  **by**  $simp$   
**ultimately show**  $?thesis$  **by**  $linarith$   
**qed**

**fun**  $exec :: 'a :: linorder\ op \Rightarrow 'a\ heap\ list \Rightarrow 'a\ heap$  **where**  
 $exec\ Empty\ [] = heap.Empty\ |$   
 $exec\ Del\_min\ [h] = del\_min\ h\ |$   
 $exec\ (Insert\ x)\ [h] = Pairing\_Heap\_List1.insert\ x\ h\ |$   
 $exec\ Merge\ [h1,h2] = merge\ h1\ h2$

**fun**  $T_{pass1} :: 'a\ heap\ list \Rightarrow nat$  **where**  
 $T_{pass1}\ [] = 1$   
 $| T_{pass1}\ [_] = 1$   
 $| T_{pass1}\ (\_ \# \_ \# hs) = 1 + T_{pass1}\ hs$

**fun**  $T_{pass2} :: 'a\ heap\ list \Rightarrow nat$  **where**  
 $T_{pass2}\ [] = 1$   
 $| T_{pass2}\ (\_ \# hs) = 1 + T_{pass2}\ hs$

**fun**  $cost :: 'a :: linorder\ op \Rightarrow 'a\ heap\ list \Rightarrow nat$  **where**  
 $cost\ Empty\ \_ = 1\ |$   
 $cost\ Del\_min\ [heap.Empty] = 1\ |$   
 $cost\ Del\_min\ [Hp\ x\ hs] = T_{pass2}\ (pass_1\ hs) + T_{pass1}\ hs\ |$   
 $cost\ (Insert\ a)\ \_ = 1\ |$   
 $cost\ Merge\ \_ = 1$

```

fun U :: 'a :: linorder op  $\Rightarrow$  'a heap list  $\Rightarrow$  real where
  U Empty _ = 1 |
  U (Insert a) [h] = log 2 (size_hp h + 1) + 1 |
  U Del_min [h] = 3*log 2 (size_hp h + 1) + 4 |
  U Merge [h1,h2] = log 2 (size_hp h1 + size_hp h2 + 1) + 2

```

**interpretation** *pairing: Amortized*

**where** *arity* = *arity* **and** *exec* = *exec* **and** *cost* = *cost* **and** *inv* = *is\_root*

**and**  $\Phi = \Phi$  **and**  $U = U$

**proof** (*standard, goal\_cases*)

**case** (1 *ss f*) **show** ?*case*

**proof** (*cases f*)

**case** *Empty* **with** 1 **show** ?*thesis* **by** *simp*

**next**

**case** *Insert* **thus** ?*thesis* **using** 1 **by**(*auto simp: is\_root\_merge*)

**next**

**case** *Merge*

**thus** ?*thesis* **using** 1 **by**(*auto simp: is\_root\_merge numeral\_eq\_Suc*)

**next**

**case** [*simp*]: *Del\_min*

**then obtain** *h* **where** [*simp*]: *ss* = [*h*] **using** 1 **by** *auto*

**show** ?*thesis*

**proof** (*cases h*)

**case** [*simp*]: (*Hp \_ hs*)

**show** ?*thesis*

**proof** *cases*

**assume** *hs* = [] **thus** ?*thesis* **by** *simp*

**next**

**assume** *hs*  $\neq$  [] **thus** ?*thesis*

**using** 1(1) *no\_Emptys\_pass1* **by** (*auto intro: is\_root\_pass2*)

**qed**

**qed** *simp*

**qed**

**next**

**case** (2 *s*) **show** ?*case* **by** (*cases s*) (*auto simp:  $\Phi$ \_hps\_ge0*)

**next**

**case** (3 *ss f*) **show** ?*case*

**proof** (*cases f*)

**case** *Empty* **with** 3 **show** ?*thesis* **by**(*auto*)

**next**

**case** *Insert* **thus** ?*thesis* **using**  $\Delta\Phi$ \_insert 3 **by** *auto*

**next**

**case** [*simp*]: *Del\_min*

**then obtain** *h* **where** [*simp*]: *ss* = [*h*] **using** 3 **by** *auto*

```

show ?thesis
proof (cases h)
  case [simp]: (Hp x hs)
    have  $T_{pass2} (pass1\ hs) + T_{pass1}\ hs \leq 2 + length\ hs$ 
      by (induct hs rule: pass1.induct) simp_all
    hence  $cost\ f\ ss \leq \dots$  by simp
    moreover have  $\Phi (del\_min\ h) - \Phi\ h \leq 3 * log\ 2 (size\_hp\ h + 1) -$ 
length hs + 2
    proof (cases hs = [])
      case False
        hence  $\Phi (del\_min\ h) - \Phi\ h \leq 3 * log\ 2 (size\_hps\ hs) - length\ hs +$ 
2
        using  $\Delta\Phi\_del\_min[of\ h]\ 3(1)$  by simp
        also have  $\dots \leq 3 * log\ 2 (size\_hp\ h + 1) - length\ hs + 2$ 
        using False 3(1) size_hps_pass2 by fastforce
        finally show ?thesis .
    qed simp
    ultimately show ?thesis by simp
  qed simp
next
  case [simp]: Merge
  then obtain h1 h2 where [simp]: ss = [h1, h2]
    using 3 by(auto simp: numeral_eq_Suc)
  show ?thesis
  proof (cases h1 = heap.Empty  $\vee$  h2 = heap.Empty)
    case True thus ?thesis by auto
  next
    case False
      then obtain x1 x2 hs1 hs2 where [simp]: h1 = Hp x1 hs1 h2 = Hp
x2 hs2
        by (meson hps.cases)
      have  $\Phi (merge\ h1\ h2) - \Phi\ h1 - \Phi\ h2 \leq log\ 2 (size\_hp\ h1 + size\_hp$ 
h2 + 1) + 1
        using  $\Delta\Phi\_merge[of\ h1\ h2]$  by simp
        thus ?thesis by(simp)
    qed
  qed
qed
end

```

### 8.3 Transfer of Tree Analysis to List Representation

**theory** Pairing\_Heap\_List1\_Analysis2



**imports**

*Pairing\_Heap\_List1\_Analysis*

*Pairing\_Heap\_Tree\_Analysis*

**begin**

This theory transfers the amortized analysis of the tree-based pairing heaps to Okasaki's pairing heaps.

**abbreviation** *is\_root'* == *Pairing\_Heap\_List1\_Analysis.is\_root*

**abbreviation** *del\_min'* == *Pairing\_Heap\_List1.del\_min*

**abbreviation** *insert'* == *Pairing\_Heap\_List1.insert*

**abbreviation** *merge'* == *Pairing\_Heap\_List1.merge*

**abbreviation** *pass1'* == *Pairing\_Heap\_List1.pass1*

**abbreviation** *pass2'* == *Pairing\_Heap\_List1.pass2*

**abbreviation** *T<sub>pass1</sub>'* == *Pairing\_Heap\_List1\_Analysis.T<sub>pass1</sub>*

**abbreviation** *T<sub>pass2</sub>'* == *Pairing\_Heap\_List1\_Analysis.T<sub>pass2</sub>*

**fun** *homs* :: 'a heap list  $\Rightarrow$  'a tree **where**

*homs* [] = *Leaf* |

*homs* (*Hp* *x lhs* # *rhs*) = *Node* (*homs lhs*) *x* (*homs rhs*)

**fun** *hom* :: 'a heap  $\Rightarrow$  'a tree **where**

*hom* *heap.Empty* = *Leaf* |

*hom* (*Hp* *x hs*) = (*Node* (*homs hs*) *x Leaf*)

**lemma** *homs\_pass1'*: *no\_Emptys hs*  $\implies$  *homs*(*pass1'* *hs*) = *pass1* (*homs* *hs*)

**apply**(*induction* *hs* *rule: Pairing\_Heap\_List1.pass1.induct*)

**subgoal** **for** *h1 h2*

**apply**(*case\_tac* *h1*)

**apply** *simp*

**apply**(*case\_tac* *h2*)

**apply** (*auto*)

**done**

**apply** *simp*

**subgoal** **for** *h*

**apply**(*case\_tac* *h*)

**apply** (*auto*)

**done**

**done**

**lemma** *hom\_merge'*:  $\llbracket$  *no\_Emptys lhs*; *Pairing\_Heap\_List1\_Analysis.is\_root* *h* $\rrbracket$

$\implies$  *hom* (*merge'* (*Hp* *x lhs*) *h*) = *link*  $\langle$  *homs lhs*, *x*, *hom h* $\rangle$

**by**(*cases* *h*) *auto*

**lemma** *hom\_pass2'*:  $no\_Emptyys\ hs \implies hom(pass2'\ hs) = pass2\ (homs\ hs)$   
**by**(*induction hs rule: homs.induct*) (*auto simp: hom\_merge' is\_root\_pass2*)

**lemma** *del\_min'*:  $is\_root'\ h \implies hom(del\_min'\ h) = del\_min\ (hom\ h)$   
**by**(*cases h*)  
(*auto simp: homs\_pass1' hom\_pass2' no\_Emptyys\_pass1 is\_root\_pass2*)

**lemma** *insert'*:  $is\_root'\ h \implies hom(insert'\ x\ h) = insert\ x\ (hom\ h)$   
**by**(*cases h*)(*auto*)

**lemma** *merge'*:  
 $\llbracket is\_root'\ h1; is\_root'\ h2 \rrbracket \implies hom(merge'\ h1\ h2) = merge\ (hom\ h1)$   
(*hom h2*)  
**apply**(*cases h1*)  
**apply**(*simp*)  
**apply**(*cases h2*)  
**apply**(*auto*)  
**done**

**lemma** *T\_pass1'*:  $no\_Emptyys\ hs \implies T_{pass1}'\ hs = T_{pass1}(homs\ hs)$   
**apply**(*induction hs rule: Pairing\_Heap\_List1.pass1.induct*)  
**subgoal for** *h1 h2*  
**apply**(*case\_tac h1*)  
**apply** *simp*  
**apply**(*case\_tac h2*)  
**apply** (*auto*)  
**done**  
**apply** *simp*  
**subgoal for** *h*  
**apply**(*case\_tac h*)  
**apply** (*auto*)  
**done**  
**done**

**lemma** *T\_pass2'*:  $no\_Emptyys\ hs \implies T_{pass2}'\ hs = T_{pass2}(homs\ hs)$   
**by**(*induction hs rule: homs.induct*) (*auto simp: hom\_merge' is\_root\_pass2*)

**lemma** *size\_hp*:  $is\_root'\ h \implies size\_hp\ h = size\ (hom\ h)$   
**proof**(*induction h*)  
**case** (*Hp \_ hs*) **thus** *?case*  
**apply**(*induction hs rule: homs.induct*)  
**apply** *simp*  
**apply** *force*

```

    apply simp
  done
qed simp

interpretation Amortized2
where arity = arity and exec = exec and inv = is_root
and cost = cost and  $\Phi = \Phi$  and  $U = U$ 
and hom = hom
and exec' = Pairing_Heap_List1_Analysis.exec
and cost' = Pairing_Heap_List1_Analysis.cost and inv' = is_root'
and U' = Pairing_Heap_List1_Analysis.U
proof (standard, goal_cases)
  case (1 f) thus ?case
    by (cases f)(auto simp: merge' del_min' numeral_eq_Suc)
next
  case (2 ts f)
  show ?case
  proof(cases f)
    case [simp]: Del_min
    then obtain h where [simp]: ts = [h] using 2 by auto
    show ?thesis using 2
      by(cases h) (auto simp: is_root_pass2 no_Empty_pass1)
  qed (insert 2,
    auto simp: Pairing_Heap_List1_Analysis.is_root_merge numeral_eq_Suc)
next
  case (3 t) thus ?case by (cases t) (auto)
next
  case (4 ts f) show ?case
  proof (cases f)
    case [simp]: Del_min
    then obtain h where [simp]: ts = [h] using 4 by auto
    show ?thesis using 4
      by (cases h)(auto simp: T_pass1' T_pass2' no_Empty_pass1 homs_pass1')
  qed (insert 4, auto)
next
  case (5 f) thus ?case by(cases f) (auto simp: size_hp numeral_eq_Suc)
qed

end

```

#### 8.4 Okasaki's Pairing Heap (Modified)

```

theory Pairing_Heap_List2_Analysis
imports

```

```

    Pairing_Heap.Pairing_Heap_List2
    Amortized_Framework
    Priority_Queue_ops_merge
    Lemmas_log
begin
    Amortized analysis of a modified version of the pairing heaps defined by
    Okasaki [6].

fun lift_hp :: 'b ⇒ ('a hp ⇒ 'b) ⇒ 'a heap ⇒ 'b where
    lift_hp c f None = c |
    lift_hp c f (Some h) = f h

fun size_hps :: 'a hp list ⇒ nat where
    size_hps (Hp x hsl # hsr) = size_hps hsl + size_hps hsr + 1 |
    size_hps [] = 0

definition size_hp :: 'a hp ⇒ nat where
    [simp]: size_hp h = size_hps (hps h) + 1

fun Φ_hps :: 'a hp list ⇒ real where
    Φ_hps [] = 0 |
    Φ_hps (Hp x hsl # hsr) = Φ_hps hsl + Φ_hps hsr + log 2 (size_hps hsl
    + size_hps hsr + 1)

definition Φ_hp :: 'a hp ⇒ real where
    [simp]: Φ_hp h = Φ_hps (hps h) + log 2 (size_hps (hps h) + 1)

abbreviation Φ :: 'a heap ⇒ real where
    Φ ≡ lift_hp 0 Φ_hp

abbreviation size_heap :: 'a heap ⇒ nat where
    size_heap ≡ lift_hp 0 size_hp

lemma Φ_hps_ge0: Φ_hps hs ≥ 0
by (induction hs rule: size_hps.induct) auto

declare algebra_simps[simp]

lemma size_hps_Cons[simp]: size_hps(h # hs) = size_hp h + size_hps
    hs
by(cases h) simp

lemma link2: link (Hp x lx) h = (case h of (Hp y ly) ⇒
    (if x < y then Hp x (Hp y ly # lx) else Hp y (Hp x lx # ly)))

```

**by**(*simp split: hp.split*)

**lemma** *size\_hps\_link*:  $size\_hps(hps (link\ h1\ h2)) = size\_hp\ h1 + size\_hp\ h2 - 1$

**by** (*induction rule: link.induct*) *simp\_all*

**lemma** *pass1\_size*[*simp*]:  $size\_hps (pass_1\ hs) = size\_hps\ hs$

**by** (*induct hs rule: pass1.induct*) (*simp\_all add: size\_hps\_link*)

**lemma** *pass2\_None*[*simp*]:  $pass_2\ hs = None \longleftrightarrow hs = []$

**by**(*cases hs*) *auto*

**lemma**  $\Delta\Phi\_insert$ :

$\Phi (Pairing\_Heap\_List2.insert\ x\ h) - \Phi\ h \leq \log\ 2 (size\_heap\ h + 1)$

**by**(*induct h*)(*auto simp: link2.split: hp.split*)

**lemma**  $\Delta\Phi\_link$ :  $\Phi\_hp (link\ h1\ h2) - \Phi\_hp\ h1 - \Phi\_hp\ h2 \leq 2 * \log\ 2 (size\_hp\ h1 + size\_hp\ h2)$

**by** (*induction h1 h2 rule: link.induct*) (*simp add: add\_increasing*)

**fun** *sum\_ub* :: 'a hp list  $\Rightarrow$  real **where**

*sum\_ub* [] = 0

| *sum\_ub* [Hp \_ \_] = 0

| *sum\_ub* [Hp \_ lx, Hp \_ ly] =  $2 * \log\ 2 (2 + size\_hps\ lx + size\_hps\ ly)$

| *sum\_ub* (Hp \_ lx # Hp \_ ly # ry) =  $2 * \log\ 2 (2 + size\_hps\ lx + size\_hps\ ly + size\_hps\ ry)$

–  $2 * \log\ 2 (size\_hps\ ry) - 2 + sum\_ub\ ry$

**lemma**  $\Delta\Phi\_pass1\_sum\_ub$ :  $\Phi\_hps (pass_1\ h) - \Phi\_hps\ h \leq sum\_ub\ h$

**proof** (*induction h rule: sum\_ub.induct*)

**case** ( $\exists\ lx\ x\ ly\ y$ )

**have**  $0: \bigwedge x\ y::real. 0 \leq x \implies x \leq y \implies x \leq 2 * y$  **by** *linarith*

**show** ?*case* **by** (*simp add: add\_increasing 0*)

**next**

**case** ( $4\ x\ hsx\ y\ hsy\ z\ hsize\_hp$ )

**let** ?*ry* =  $z \# hsize\_hp$

**let** ?*rx* =  $Hp\ y\ hsy \# ?ry$

**let** ?*h* =  $Hp\ x\ hsx \# ?rx$

**have**  $\Phi\_hps(pass_1\ ?h) - \Phi\_hps\ ?h$

$\leq \log\ 2 (1 + size\_hps\ hsx + size\_hps\ hsy) - \log\ 2 (1 + size\_hps\ hsy$

$+ size\_hps\ ?ry) + sum\_ub\ ?ry$

**using** *4.IH* **by** *simp*

**also have**  $\log\ 2 (1 + size\_hps\ hsx + size\_hps\ hsy) - \log\ 2 (1 + size\_hps$

$hsy + size\_hps\ ?ry$   
 $\leq 2 * \log 2 (size\_hps\ ?h) - 2 * \log 2 (size\_hps\ ?ry) - 2$   
**proof** –  
**have**  $\log 2 (1 + size\_hps\ hsx + size\_hps\ hsy) + \log 2 (size\_hps\ ?ry)$   
 $- 2 * \log 2 (size\_hps\ ?h)$   
 $= \log 2 ((1 + size\_hps\ hsx + size\_hps\ hsy) / (size\_hps\ ?h)) + \log 2$   
 $(size\_hps\ ?ry / size\_hps\ ?h)$   
**by** (*simp add: log\_divide*)  
**also have**  $\dots \leq -2$   
**proof** –  
**have**  $2 + \dots$   
 $\leq 2 * \log 2 ((1 + size\_hps\ hsx + size\_hps\ hsy) / size\_hps\ ?h +$   
 $size\_hps\ ?ry / size\_hps\ ?h)$   
**using** *ld\_sum\_inequality* [*of*  $(1 + size\_hps\ hsx + size\_hps\ hsy) /$   
 $size\_hps\ ?h (size\_hps\ ?ry / size\_hps\ ?h)$ ] **by** *simp*  
**also have**  $\dots \leq 0$  **by** (*simp add: field\_simps log\_divide add\_pos\_nonneg*)  
**finally show** *?thesis* **by** *linarith*  
**qed**  
**finally have**  $\log 2 (1 + size\_hps\ hsx + size\_hps\ hsy) + \log 2 (size\_hps$   
 $?ry) + 2$   
 $\leq 2 * \log 2 (size\_hps\ ?h)$  **by** *simp*  
**moreover have**  $\log 2 (size\_hps\ ?ry) \leq \log 2 (size\_hps\ ?rx)$  **by** *simp*  
**ultimately have**  $\log 2 (1 + size\_hps\ hsx + size\_hps\ hsy) - \dots$   
 $\leq 2 * \log 2 (size\_hps\ ?h) - 2 * \log 2 (size\_hps\ ?ry) - 2$  **by** *linarith*  
**thus** *?thesis* **by** *simp*  
**qed**  
**finally show** *?case* **by** (*simp*)  
**qed** *simp\_all*

**lemma**  $\Delta\Phi\_pass1$ : **assumes**  $hs \neq []$   
**shows**  $\Phi\_hps (pass1\ hs) - \Phi\_hps\ hs \leq 2 * \log 2 (size\_hps\ hs) - length$   
 $hs + 2$   
**proof** –  
**have**  $sum\_ub\ hs \leq 2 * \log 2 (size\_hps\ hs) - length\ hs + 2$   
**using** *assms* **by** (*induct hs rule: sum\_ub.induct*) (*simp\_all*)  
**thus** *?thesis* **using**  $\Delta\Phi\_pass1\_sum\_ub[of\ hs]$  **by** *linarith*  
**qed**

**lemma**  $size\_hps\_pass2$ :  $pass2\ hs = Some\ h \implies size\_hps\ hs = size\_hps(hs$   
 $h) + 1$   
**apply** (*induction hs arbitrary: h rule:  $\Phi\_hps.induct$* )  
**apply** (*auto simp: link2 split: option.split hp.split*)  
**done**

```

lemma  $\Delta\Phi_{pass2}$ :  $hs \neq [] \implies \Phi (pass_2 hs) - \Phi_{hps} hs \leq \log 2 (size\_hps$ 
 $hs)$ 
proof (induction hs)
  case (Cons h hs)
  thus ?case
  proof -
    obtain  $x hs2$  where [simp]:  $h = Hp\ x\ hs2$  by (metis hp.exhaust)
    show ?thesis
    proof (cases pass_2 hs)
      case [simp]: (Some h2)
        obtain  $y hs3$  where [simp]:  $h2 = Hp\ y\ hs3$  by (metis hp.exhaust)
        from size_hps_pass2[OF Some] Cons show ?thesis
          by(cases hs=[])(auto simp: add_mono)
        qed simp
      qed
    qed simp

```

```

lemma  $\Delta\Phi_{del\_min}$ : assumes  $hps\ h \neq []$ 
  shows  $\Phi (del\_min (Some\ h)) - \Phi (Some\ h)$ 
   $\leq 3 * \log 2 (size\_hps(hps\ h)) - length(hps\ h) + 2$ 
proof -
  let ? $\Delta\Phi_1 = \Phi_{hps}(hps\ h) - \Phi_{hp}\ h$ 
  let ? $\Delta\Phi_2 = \Phi(pass_2(pass_1(hps\ h))) - \Phi_{hps}(hps\ h)$ 
  let ? $\Delta\Phi = \Phi (del\_min (Some\ h)) - \Phi (Some\ h)$ 
  have  $\Phi(pass_2(pass_1(hps\ h))) - \Phi_{hps}(pass_1(hps\ h)) \leq \log 2 (size\_hps(hps$ 
 $h))$ 
    using  $\Delta\Phi_{pass2}$ [of pass_1(hps h)] using size_hps.elims assms by force
    moreover have  $\Phi_{hps}(pass_1(hps\ h)) - \Phi_{hps}(hps\ h) \leq 2 * \dots -$ 
 $length(hps\ h) + 2$ 
    using  $\Delta\Phi_{pass1}$ [OF assms] by blast
    moreover have ? $\Delta\Phi_1 \leq 0$  by (cases h) simp
    moreover have ? $\Delta\Phi = ?\Delta\Phi_1 + ?\Delta\Phi_2$  by (cases h) simp
    ultimately show ?thesis by linarith
  qed

```

```

fun exec :: ' $a$ ' :: linorder op  $\Rightarrow$  ' $a$  heap list  $\Rightarrow$  ' $a$  heap where
exec Empty [] = None |
exec Del_min [h] = del_min h |
exec (Insert x) [h] = Pairing_Heap_List2.insert x h |
exec Merge [h1,h2] = merge h1 h2

```

```

fun  $T_{pass1}$  :: ' $a$  hp list  $\Rightarrow$  nat where

```

```

    Tpass1 [] = 1
  | Tpass1 [_] = 1
  | Tpass1 (_ # _ # hs) = 1 + Tpass1 hs

```

```

fun Tpass2 :: 'a hp list ⇒ nat where
  Tpass2 [] = 1 |
  Tpass2 (_ # hs) = 1 + Tpass2 hs

```

```

fun cost :: 'a :: linorder op ⇒ 'a heap list ⇒ nat where
  cost Empty _ = 1 |
  cost Del_min [None] = 1 |
  cost Del_min [Some(Hp x hs)] = 1 + Tpass2 (pass1 hs) + Tpass1 hs |
  cost (Insert a) _ = 1 |
  cost Merge _ = 1

```

```

fun U :: 'a :: linorder op ⇒ 'a heap list ⇒ real where
  U Empty _ = 1 |
  U (Insert a) [h] = log 2 (size_heap h + 1) + 1 |
  U Del_min [h] = 3*log 2 (size_heap h + 1) + 5 |
  U Merge [h1,h2] = 2*log 2 (size_heap h1 + size_heap h2 + 1) + 1

```

**interpretation** pairing: Amortized

**where** arity = arity **and** exec = exec **and** cost = cost **and** inv = λ\_. True

**and** Φ = Φ **and** U = U

**proof** (standard, goal\_cases)

**case** (2 s) **show** ?case **by** (cases s) (auto simp: Φ\_hps\_ge0)

**next**

**case** (3 ss f) **show** ?case

**proof** (cases f)

**case** Empty **with** 3 **show** ?thesis **by**(auto)

**next**

**case** Insert

**thus** ?thesis **using** Insert ΔΦ\_insert 3 **by** auto

**next**

**case** [simp]: Del\_min

**then obtain** ho **where** [simp]: ss = [ho] **using** 3 **by** auto

**show** ?thesis

**proof** (cases ho)

**case** [simp]: (Some h)

**show** ?thesis

**proof** (cases h)

**case** [simp]: (Hp x hs)

**have** T<sub>pass2</sub> (pass1 hs) + T<sub>pass1</sub> hs ≤ 2 + length hs

**by** (induct hs rule: pass1.induct) simp\_all



```

    hence  $cost\ f\ ss \leq 1 + \dots$  by simp
    moreover have  $\Phi\ (del\_min\ ho) - \Phi\ ho \leq 3 * \log\ 2\ (size\_heap\ ho$ 
+ 1) -  $length\ hs + 2$ 
    proof (cases  $hs = []$ )
      case False
        hence  $\Phi\ (del\_min\ ho) - \Phi\ ho \leq 3 * \log\ 2\ (size\_hps\ hs) - length$ 
 $hs + 2$ 
          using  $\Delta\Phi\_del\_min[of\ h]$  by simp
          also have  $\dots \leq 3 * \log\ 2\ (size\_heap\ ho + 1) - length\ hs + 2$ 
            using False  $size\_hps.elims$  by force
          finally show ?thesis .
        qed simp
      ultimately show ?thesis by simp
    qed
  qed simp
next
case [simp]: Merge
then obtain  $ho1\ ho2$  where [simp]:  $ss = [ho1, ho2]$ 
  using 3 by (auto simp: numeral_eq_Suc)
show ?thesis
proof (cases  $ho1 = None \vee ho2 = None$ )
  case True thus ?thesis by auto
next
  case False
    then obtain  $h1\ h2$  where [simp]:  $ho1 = Some\ h1\ ho2 = Some\ h2$ 
    by auto
    have  $\Phi\ (merge\ ho1\ ho2) - \Phi\ ho1 - \Phi\ ho2 \leq 2 * \log\ 2\ (size\_heap$ 
 $ho1 + size\_heap\ ho2)$ 
      using  $\Delta\Phi\_link[of\ h1\ h2]$  by simp
      also have  $\dots \leq 2 * \log\ 2\ (size\_hp\ h1 + size\_hp\ h2 + 1)$  by (simp)
      finally show ?thesis by (simp)
    qed
  qed
qed simp

end

```

## References

- [1] H. Brinkop. Verifikation der amortisierten Laufzeit von Pairing Heaps in Isabelle, 2015. Bachelor's Thesis, Fakultät für Informatik, Technische Universität München.

- [2] M. L. Fredman, R. Sedgewick, D. D. Sleator, and R. E. Tarjan. The pairing heap: A new form of self-adjusting heap. *Algorithmica*, 1(1):111–129, 1986.
- [3] A. Kaldewaij and B. Schoenmakers. The derivation of a tighter bound for top-down skew heaps. *Information Processing Letters*, 37:265–271, 1991.
- [4] T. Nipkow. Amortized complexity verified. In C. Urban and X. Zhang, editors, *Interactive Theorem Proving (ITP 2015)*, volume 9236 of *LNCS*, pages 310–324. Springer, 2015.
- [5] T. Nipkow and H. Brinkop. Amortized complexity verified, 2016. Submitted for publication.
- [6] C. Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1998.
- [7] B. Schoenmakers. A systematic analysis of splaying. *Information Processing Letters*, 45:41–50, 1993.