

Amortized Complexity Verified

Tobias Nipkow

April 19, 2020

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

1	Amortized Complexity (Unary Operations)	3
1.1	Binary Counter	4
1.2	Dynamic tables: insert only	5
1.3	Stack with multipop	8
1.4	Queue	8
1.5	Dynamic tables: insert and delete	10
2	Amortized Complexity Framework	11
3	Simple Examples	13
3.1	Binary Counter	13
3.2	Stack with multipop	15
3.3	Dynamic tables: insert only	15
3.4	Dynamic tables: insert and delete	19
3.5	Queue	20
4	Skew Heap Analysis	23
5	Splay Tree	29
5.1	Basics	29
5.2	Splay Tree Analysis	32
5.3	Splay Tree Analysis (Optimal)	42
6	Splay Heap	54

7	Pairing Heaps	60
7.1	Binary Tree Representation	60
7.2	Okasaki's Pairing Heap	66
7.3	Transfer of Tree Analysis to List Representation	74
7.4	Okasaki's Pairing Heap (Modified)	77

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 Although it is superseded by the model in *Amortized_Framework* that allows arbitrarily many parameters, it is still of interest because of its simplicity.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

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 p_{incr} :: bool list \Rightarrow real (Φ_{incr}) where
 $\Phi_{incr} bs = length(filter id bs)$

lemma a_{incr}: $t_{incr} bs + \Phi_{incr}(incr bs) - \Phi_{incr} bs = 2$
apply(induction bs rule: incr.induct)
apply (simp_all add: p_{incr}-def)
done

interpretation incr: *Amortized*
where *init* = [] **and** *next* = %₋. incr **and** *inv* = $\lambda_{-} . True$
and *t* = $\lambda_{-} . t_{incr}$ **and** $\Phi = \Phi_{incr}$ **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: p_{incr}-def)
next
case 4 show ?case by(simp add: p_{incr}-def)
next
case 5 show ?case by(simp add: a_{incr})
qed

thm *incr.ub*

1.2 Dynamic tables: insert only

fun $t_{ins} :: nat \times nat \Rightarrow real$ **where**
 $t_{ins} (n,l) = (if\ n < l\ then\ 1\ else\ n+1)$

interpretation *ins*: *Amortized*

where $init = (0::nat, 0::nat)$

and $nxt = \lambda_ (n,l). (n+1, if\ n < l\ then\ l\ else\ if\ l=0\ then\ 1\ else\ 2*l)$

and $inv = \lambda(n,l). if\ l=0\ then\ n=0\ else\ n \leq l \wedge l < 2*n$

and $t = \lambda_ t_{ins}$ **and** $\Phi = \lambda(n,l). 2*n - l$ **and** $U = \lambda_ .. 3$

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

locale *table_insert* =

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*)

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 ¬ *n*<*l*

hence [*simp*]: *n*=*l* **using** 2 (<*l*≠0) **by** *simp*

have 1: (*b*/*a*) * *ceiling*(*c* * *l*) ≤ *real l* + 1

proof–

have (*b*/*a*) * *ceiling*(*c* * *l*) = *ceiling*(*c* * *l*)/(*a**(*c* - 1))

by(*simp add: b_def*)

also have *ceiling*(*c* * *l*) ≤ *c***l* + 1 **by** *simp*

also have ... ≤ *c**(*real l*+1) **by** (*simp add: algebra_simps*)

also have ... / (*a**(*c* - 1)) = (*c*/(*a**(*c* - 1))) * (*real l* + 1) **by**

simp

also have *c*/(*a**(*c* - 1)) ≤ 1 **using** *ac2* **by** (*simp add: field_simps*)

finally show ?*thesis* **by** (*simp add: divide_right_mono*)

qed

have 2: *real l* + 1 ≤ *ceiling*(*c* * *real l*)

proof–

```

    have real  $l + 1 = of\_int(int(l)) + 1$  by simp
    also have  $\dots \leq ceiling(c * real l)$  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 (simp add: not_le zero_less_mult_iff)
  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

```

```

    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 ⇒ 'a list ⇒ 'a list where
next_stk (Push x) xs = x # xs |
next_stk (Pop n) xs = drop n xs

```

```

fun t_stk :: 'a op_stk ⇒ 'a list ⇒ 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** $U = \lambda f \dots case\ f\ of\ Push\ _ \Rightarrow 2 \mid Pop\ _ \Rightarrow 0$

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 op_q = Enq 'a | Deq

```

```

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

```



```

fun  $next_q :: 'a\ op_q \Rightarrow 'a\ queue \Rightarrow 'a\ queue$  where
 $next_q\ (Enq\ x)\ (xs,ys) = (x\#\ xs,ys) \mid$ 
 $next_q\ Deq\ (xs,ys) = (if\ ys = []\ then\ ([],\ tl(rev\ xs))\ else\ (xs,tl\ ys))$ 

```

```

fun  $t_q :: 'a\ op_q \Rightarrow 'a\ queue \Rightarrow real$  where
 $t_q\ (Enq\ x)\ (xs,ys) = 1 \mid$ 
 $t_q\ Deq\ (xs,ys) = (if\ ys = []\ then\ length\ xs\ else\ 0)$ 

```

interpretation *queue: Amortized*

```

where  $init = ([],[])$  and  $next = next_q$  and  $inv = \lambda\_.\ True$ 
and  $t = t_q$  and  $\Phi = \lambda(xs,ys).\ length\ xs$  and  $U = \lambda f\ _.\ case\ f\ of\ Enq\ _ \Rightarrow$ 
 $2 \mid Deq \Rightarrow 0$ 

```

proof (*standard, goal_cases*)

case 1 **show** ?case **by** auto

next

case (2 s) **thus** ?case **by**(cases s) auto

next

case (3 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  $next\_q2 :: 'a\ op_q \Rightarrow 'a\ queue \Rightarrow 'a\ queue$  where
 $next\_q2\ (Enq\ a)\ (xs,ys) = balance\ (a\#\ xs,ys) \mid$ 
 $next\_q2\ Deq\ (xs,ys) = balance\ (xs,\ tl\ ys)$ 

```

```

fun  $t\_q2 :: 'a\ op_q \Rightarrow 'a\ queue \Rightarrow real$  where
 $t\_q2\ (Enq\ _)\ (xs,ys) = 1 + (if\ size\ xs + 1 \leq\ size\ ys\ then\ 0\ else\ size\ xs +$ 
 $1 + size\ ys) \mid$ 
 $t\_q2\ Deq\ (xs,ys) = (if\ size\ xs \leq\ size\ ys - 1\ then\ 0\ else\ size\ xs + (size\ ys$ 
 $- 1))$ 

```

interpretation *queue2: Amortized*

```

where  $init = ([], [])$  and  $nxt = nxt\_q2$ 
and  $inv = \lambda(xs, ys). size\ xs \leq size\ ys$ 
and  $t = t\_q2$  and  $\Phi = \lambda(xs, ys). 2 * size\ xs$ 
and  $U = \lambda f \_ . case\ f\ of\ Enq\ \_ \Rightarrow 3 \mid Deq \Rightarrow 0$ 
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  $optb = Ins \mid Del$ 

```

```

fun  $nxt_{tb} :: optb \Rightarrow nat * nat \Rightarrow nat * nat$  where
 $nxt_{tb}\ Ins\ (n, l) = (n + 1, if\ n < l\ then\ l\ else\ if\ l = 0\ then\ 1\ else\ 2 * l) \mid$ 
 $nxt_{tb}\ Del\ (n, l) = (n - 1, if\ n = 1\ then\ 0\ else\ if\ 4 * (n - 1) < l\ then\ l\ div\ 2$ 
   $else\ l)$ 

```

```

fun  $t_{tb} :: optb \Rightarrow nat * nat \Rightarrow real$  where
 $t_{tb}\ Ins\ (n, l) = (if\ n < l\ then\ 1\ else\ n + 1) \mid$ 
 $t_{tb}\ Del\ (n, l) = (if\ n = 1\ then\ 1\ else\ if\ 4 * (n - 1) < l\ then\ n\ else\ 1)$ 

```

interpretation tb : Amortized

```

where  $init = (0, 0)$  and  $nxt = nxt_{tb}$ 
and  $inv = \lambda(n, l). if\ l = 0\ then\ n = 0\ else\ n \leq l \wedge l \leq 4 * n$ 
and  $t = t_{tb}$  and  $\Phi = (\lambda(n, l). if\ 2 * n < l\ then\ l / 2 - n\ else\ 2 * n - l)$ 
and  $U = \lambda f \_ . case\ f\ of\ Ins \Rightarrow 3 \mid Del \Rightarrow 2$ 
proof (standard, goal_cases)
  case 1 show ?case by auto
next
  case (2 s f) thus ?case by(cases s, cases f) (auto split: if_splits)
next
  case (3 s) thus ?case by(cases s)(simp split: if_splits)
next

```

```

    case 4 show ?case by(simp)
next
  case (5 s f) thus ?case apply(cases s) apply(cases f)
    by (auto simp: field_simps)
qed

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:  $\text{inv } s \Longrightarrow \Phi s \geq 0$ 
assumes U:  $\llbracket \forall s \in \text{set } ss. \text{inv } s; \text{length } ss = \text{arity } f \rrbracket \Longrightarrow \text{cost } f \text{ } ss + \Phi(\text{exec } f \text{ } ss) - \text{sum\_list } (\text{map } \Phi \text{ } ss) \leq U f \text{ } ss$ 
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  $\Longrightarrow$  inv(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* - Φ (*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*: $\llbracket \forall s \in \text{set } ss. \text{inv } s; \text{length } ss = \text{arity } f \rrbracket \Longrightarrow \text{acost } f \text{ } ss \leq U \text{ } f \text{ } ss$
by(*simp add: acost_def U*)

lemma *u_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 u_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 Φ 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 Φ U*
for *arity* :: 'op \Rightarrow nat **and** *exec* **and** *inv* :: 's \Rightarrow bool **and** *cost Φ U* +

```

fixes exec' :: 'op ⇒ 't list ⇒ 't
fixes inv' :: 't ⇒ bool
fixes cost' :: 'op ⇒ 't list ⇒ nat
fixes U' :: 'op ⇒ 't list ⇒ real
fixes hom :: 't ⇒ 's
assumes exec': [∀ s ∈ set ts. inv' s; length ts = arity f ]
  ⇒ hom(exec' f ts) = exec f (map hom ts)
assumes inv_exec': [∀ s ∈ set ss. inv' s; length ss = arity f ]
  ⇒ inv'(exec' f ss)
assumes inv_hom: inv' t ⇒ inv (hom t)
assumes cost': [∀ s ∈ set ts. inv' s; length ts = arity f ]
  ⇒ cost' f ts = cost f (map hom ts)
assumes U': [∀ s ∈ set ts. inv' s; length ts = arity f ]
  ⇒ U' f ts = U f (map hom ts)
begin

sublocale A': Amortized arity exec' inv' cost' Φ 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 $op = Empty \mid Incr$

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

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

fun $t_{incr} :: bool\ list \Rightarrow nat$ **where**
 $t_{incr}\ [] = 1 \mid$
 $t_{incr}\ (False\#\ bs) = 1 \mid$
 $t_{incr}\ (True\#\ bs) = t_{incr}\ bs + 1$

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

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

fun $cost :: op \Rightarrow bool\ list\ list \Rightarrow nat$ **where**
 $cost\ Empty\ _ = 1 \mid$
 $cost\ Incr\ [bs] = t_{incr}\ bs$

interpretation $Amortized$

where $exec = exec$ **and** $arity = arity$ **and** $inv = \lambda_. True$
and $cost = cost$ **and** $\Phi = \Phi$ **and** $U = \lambda f _ . case\ f\ of\ Empty \Rightarrow 1 \mid Incr \Rightarrow$
 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 \Rightarrow nat **where**

arity *Empty* = 0 |

arity (*Push* _) = 1 |

arity (*Pop* _) = 1

fun *exec* :: 'a op \Rightarrow 'a list list \Rightarrow 'a list **where**

exec *Empty* [] = [] |

exec (*Push* x) [xs] = x # xs |

exec (*Pop* n) [xs] = drop n xs

fun *cost* :: 'a op \Rightarrow 'a list list \Rightarrow 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* = λ f_. case f of *Empty* \Rightarrow 1 | *Push* _ \Rightarrow 2 | *Pop* _ \Rightarrow 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 ⇒ tab list ⇒ 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 ⇒ tab list ⇒ 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 = λ(n,l). if l=0 then n=0 else n ≤ l ∧ l < 2*n
and cost = cost and Φ = λ(n,l). 2*n - l
and U = λf _ . case f of Empty ⇒ 1 | Ins ⇒ 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 ≥ 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)

```

```

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:  $\text{real } l + 1 \leq \text{ceiling}(c * \text{real } l)$ 
    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 (simp add: not_le zero_less_mult_iff)
  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≠0
    show ?thesis
    proof cases
      assume n<l
      thus ?thesis using 3 ac by (simp add: algebra_simps b_def)
    next
      assume ¬ n<l
      hence [simp]: n=l using 3 by simp
      have cost Ins [(n,l)] + Φ (ins (n,l)) - Φ(n,l) = l + a + 1 + (-
b*ceiling(c*l)) + b*l
        using ⟨l≠0⟩
        by (simp add: algebra_simps less_trans[of -1::real 0])
      also have - b * ceiling(c*l) ≤ - 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 × nat

datatype op = Empty | Ins | Del

fun arity :: op ⇒ nat where

```

```

arity Empty = 0 |
arity Ins = 1 |
arity Del = 1

```

```

fun exec :: op ⇒ tab list ⇒ 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≤1 then 0 else if 4*(n - 1)<l then l div 2 else
l)

```

```

fun cost :: op ⇒ tab list ⇒ 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  $\Rightarrow$  'a queue list  $\Rightarrow$  '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  $\Rightarrow$  'a queue list  $\Rightarrow$  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 =  $\lambda f\_.$  case f of Empty  $\Rightarrow$  0 | Enq _  $\Rightarrow$  2 | 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 @ 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, tl ys)
```

fun cost :: 'a op \Rightarrow 'a queue list \Rightarrow nat **where**

```
cost Empty _ = 0 |
```

$cost (Enq\ x) [(xs,ys)] = 1 + (if\ size\ xs + 1 \leq size\ ys\ then\ 0\ else\ size\ xs + 1 + size\ ys) \mid$
 $cost\ Deq\ [(xs,ys)] = (if\ size\ xs \leq size\ ys - 1\ then\ 0\ else\ size\ xs + (size\ ys - 1))$

interpretation *Amortized*
where $arity = arity$ **and** $exec = exec$
and $inv = \lambda(xs,ys). size\ xs \leq size\ ys$
and $cost = cost$ **and** $\Phi = \lambda(xs,ys). 2 * size\ xs$
and $U = \lambda f _ . case\ f\ of\ Empty \Rightarrow 0 \mid Enq\ _ \Rightarrow 3 \mid Deq \Rightarrow 0$
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 $arity :: 'a\ op \Rightarrow nat$ **where**
 $arity\ Empty = 0 \mid$
 $arity\ (Insert\ _) = 1 \mid$
 $arity\ Del_min = 1 \mid$
 $arity\ Merge = 2$

end

4 Skew Heap Analysis

theory *Skew_Heap_Analysis*
imports
Complex_Main
Skew_Heap.Skew_Heap
Amortized_Framework
Priority_Queue_ops_merge

begin

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

right-heavy:

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

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

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

Function Δ 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 $Gexp: 2^{\wedge} lrh\ h \leq size\ h + 1$
by (*induction h*) (*auto simp: rh_def*)

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

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

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

function $t_merge :: 'a::linorder\ heap \Rightarrow 'a\ heap \Rightarrow nat$ **where**
 $t_merge\ Leaf\ h = 1$ |
 $t_merge\ h\ Leaf = 1$ |
 $t_merge\ (Node\ l1\ a1\ r1)\ (Node\ l2\ a2\ r2) =$
 $(if\ a1 \leq a2\ then\ t_merge\ (Node\ l2\ a2\ r2)\ r1\ else\ t_merge\ (Node\ l1\ a1\ r1)\ r2) + 1$
by *pat_completeness auto*

termination

by (*relation measure* ($\lambda(x, y). size\ x + size\ y$)) *auto*

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

lemma Φ_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 ≤ 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 $\dots = t_merge\ ?t2\ r1 + 1 + \Phi\ ?m + rh\ ?m\ l1 - \Phi\ r1 - rh$
 $l1\ r1 - \Phi\ ?t2$

by *simp*

also have $\dots \leq lrh\ ?m + rlh\ ?t2 + rlh\ r1 + rh\ ?m\ l1 + 2 - rh\ l1\ r1$

using $\exists.IH(1)[OF\ True]$ **by** *linarith*

also have $\dots = lrh\ ?m + rlh\ ?t2 + rlh\ r1 + rh\ ?m\ l1 + 1 + (1 - rh$
 $l1\ r1)$

using $rh1[of\ l1\ r1]$ **by** (*simp*)

also have $\dots = lrh\ ?m + rlh\ ?t2 + rlh\ ?t1 + rh\ ?m\ l1 + 1$

by (*simp*)

also have $\dots = lrh (merge\ ?t1\ ?t2) + rlh\ ?t1 + rlh\ ?t2 + 1$

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*:

$$t_merge\ t1\ t2 + \Phi(merge\ t1\ t2) - \Phi\ t1 - \Phi\ t2 \leq \\ 3 * \log\ 2\ (size1\ t1 + size1\ t2) + 1\ (\text{is } ?l \leq -)$$

proof –

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

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

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

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

definition *t_insert* :: $'a::linorder \Rightarrow 'a\ heap \Rightarrow int$ **where**

$t_insert\ a\ h = t_merge\ (Node\ Leaf\ a\ Leaf)\ h + 1$

lemma *a_insert*: $t_insert\ a\ h + \Phi(Skew_Heap.insert\ a\ h) - \Phi\ h \leq 3 * \log\ 2\ (size1\ h + 2) + 2$

using *a_merge*[*of Node Leaf a Leaf h*]

by (*simp add: numeral_eq_Suc t_insert_def Skew_Heap.insert_def rh_def*)

definition *t_del_min* :: $('a::linorder)\ heap \Rightarrow int$ **where**

$t_del_min\ h = (case\ h\ of\ Leaf \Rightarrow 1 \mid Node\ t1\ a\ t2 \Rightarrow t_merge\ t1\ t2 + 1)$

lemma *a_del_min*: $t_del_min\ h + \Phi(del_min\ h) - \Phi\ h \leq 3 * \log\ 2\ (size1\ h + 2) + 2$

proof (*cases h*)

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))) ≤
    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 t_del_min_def rh_def)
qed

```

4.0.1 Instantiation of Amortized Framework

```

lemma t_merge_nneg: t_merge h1 h2 ≥ 0
by(induction h1 h2 rule: t_merge.induct) auto

```

```

fun exec :: 'a::linorder op ⇒ 'a heap list ⇒ 'a heap where
exec Empty [] = Leaf |
exec (Insert a) [h] = Skew_Heap.insert a h |
exec Del_min [h] = del_min h |
exec Merge [h1,h2] = merge h1 h2

```

```

fun cost :: 'a::linorder op ⇒ 'a heap list ⇒ nat where
cost Empty [] = 1 |
cost (Insert a) [h] = t_merge (Node Leaf a Leaf) h |
cost Del_min [h] = (case h of Leaf ⇒ 1 | Node t1 a t2 ⇒ t_merge t1 t2) |
cost Merge [h1,h2] = t_merge h1 h2

```

```

fun U where
U Empty [] = 1 |
U (Insert _) [h] = 3 * log 2 (size1 h + 2) + 1 |
U Del_min [h] = 3 * log 2 (size1 h + 2) + 3 |
U Merge [h1,h2] = 3 * log 2 (size1 h1 + size1 h2) + 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 h) show ?case using Φ_nneg[of h] 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 h where [simp]: ss = [h] using 3(2) by (auto)
    thus ?thesis using a_merge[of Node Leaf a Leaf h]
      by (simp add: numeral_eq_Suc insert_def rh_def t_merge_nneg)
  next
  case [simp]: Del_min
  obtain h where [simp]: ss = [h] using 3(2) by (auto)
  thus ?thesis
  proof (cases h)
    case Leaf with Del_min show ?thesis by simp
  next
    case (Node t1 _ t2)
    have [arith]:  $\log 2 (2 + (\text{real } (\text{size } t1) + \text{real } (\text{size } t2))) \leq$ 
       $\log 2 (4 + (\text{real } (\text{size } t1) + \text{real } (\text{size } t2)))$  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 h1 h2 where ss = [h1,h2] using 3(2) by (auto simp: numeral_eq_Suc)
  thus ?thesis using a_merge[of h1 h2] 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 numeral_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

begin

declare *size1_size*[*simp*]

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)

```

```

fun t_splay :: 'a::linorder  $\Rightarrow$  'a tree  $\Rightarrow$  nat where
t_splay x Leaf = 1 |
t_splay x (Node AB b CD) =
  (case cmp x b of
    EQ  $\Rightarrow$  1 |
    LT  $\Rightarrow$  (case AB of
      Leaf  $\Rightarrow$  1 |
      Node A a B  $\Rightarrow$ 
        (case cmp x a of EQ  $\Rightarrow$  1 |
          LT  $\Rightarrow$  if A = Leaf then 1 else t_splay x A + 1 |
          GT  $\Rightarrow$  if B = Leaf then 1 else t_splay x B + 1)) |
    GT  $\Rightarrow$  (case CD of
      Leaf  $\Rightarrow$  1 |
      Node C c D  $\Rightarrow$ 
        (case cmp x c of EQ  $\Rightarrow$  1 |
          LT  $\Rightarrow$  if C = Leaf then 1 else t_splay x C + 1 |
          GT  $\Rightarrow$  if D = Leaf then 1 else t_splay x D + 1)))

```

lemma t_splay_simps[simp]:

```

t_splay a (Node l a r) = 1
a < b  $\implies$  t_splay a (Node Leaf b r) = 1
a < b  $\implies$  t_splay a (Node (Node ll a lr) b r) = 1
a < b  $\implies$  a < c  $\implies$  t_splay a (Node (Node ll c lr) b r) =
  (if ll = Leaf then 1 else t_splay a ll + 1)
a < b  $\implies$  c < a  $\implies$  t_splay a (Node (Node ll c lr) b r) =
  (if lr = Leaf then 1 else t_splay a lr + 1)
b < a  $\implies$  t_splay a (Node l b Leaf) = 1
b < a  $\implies$  t_splay a (Node l b (Node rl a rr)) = 1
b < a  $\implies$  a < c  $\implies$  t_splay a (Node l b (Node rl c rr)) =
  (if rl = Leaf then 1 else t_splay a rl + 1)
b < a  $\implies$  c < a  $\implies$  t_splay a (Node l b (Node rl c rr)) =
  (if rr = Leaf then 1 else t_splay a rr + 1)

```

by auto

declare t_splay_simps(2)[simp del]

```

fun t_splay_max :: 'a::linorder tree  $\Rightarrow$  nat where
t_splay_max Leaf = 1 |
t_splay_max (Node l b Leaf) = 1 |

```

t_splay_max (Node l b (Node rl c rr)) = (if $rr=Leaf$ then 1 else t_splay_max $rr + 1$)

definition t_delete :: 'a::linorder \Rightarrow 'a tree \Rightarrow nat **where**
 t_delete a t = (if $t = Leaf$ then 0 else t_splay a t + (case $splay$ a t of
 Node l x r \Rightarrow
 if $x=a$ then case l of Leaf \Rightarrow 0 | _ $\Rightarrow t_splay_max$ l
 else 0))

lemma $ex_in_set_tree$: $t \neq Leaf \implies bst\ t \implies$
 $\exists a' \in set_tree\ t. splay\ a'\ t = splay\ a\ t \wedge t_splay\ a'\ t = t_splay\ a\ t$

proof(induction a t rule: $splay.induct$)

case (6 a b c ll)

hence $splay\ a\ ll \neq Leaf$ **by** $simp$

then **obtain** lll u llr **where** [$simp$]: $splay\ a\ ll = Node\ lll\ u\ llr$

by ($metis\ tree.exhaust$)

have $b < c$ $bst\ ll$ **using** 6.prem1 **by** $auto$

from 6.IH[OF $\langle ll \neq Leaf \rangle$ $\langle bst\ ll \rangle$]

obtain e **where** $e \in set_tree\ ll$ $splay\ e\ ll = splay\ a\ ll$ $t_splay\ e\ ll = t_splay\ a\ ll$

by $blast$

moreover hence $e < b$ **using** 6.prem2 **by** $auto$

ultimately show ?case **using** $\langle a < c \rangle$ $\langle a < b \rangle$ $\langle b < c \rangle$ $\langle bst\ ll \rangle$ **by** $force$

next

case (8 b a c lr)

hence $splay\ a\ lr \neq Leaf$ **by** $simp$

then **obtain** lrl u lrr **where** [$simp$]: $splay\ a\ lr = Node\ lrl\ u\ lrr$

by ($metis\ tree.exhaust$)

have $b < c$ $bst\ lr$ **using** 8.prem1 **by** $auto$

from 8.IH[OF $\langle lr \neq Leaf \rangle$ $\langle bst\ lr \rangle$]

obtain e **where** $e \in set_tree\ lr$ $splay\ e\ lr = splay\ a\ lr$ $t_splay\ e\ lr = t_splay\ a\ lr$

by $blast$

moreover hence $b < e$ & $e < c$ **using** 8.prem2 **by** $simp$

ultimately show ?case **using** $\langle a < c \rangle$ $\langle b < a \rangle$ $\langle b < c \rangle$ $\langle bst\ lr \rangle$ **by** $force$

next

case (11 c a b rl)

hence $splay\ a\ rl \neq Leaf$ **by** $simp$

then **obtain** rll u rlr **where** [$simp$]: $splay\ a\ rl = Node\ rll\ u\ rlr$

by ($metis\ tree.exhaust$)

have $c < b$ $bst\ rl$ **using** 11.prem1 **by** $auto$

from 11.IH[OF $\langle rl \neq Leaf \rangle$ $\langle bst\ rl \rangle$]

obtain e **where** $e \in set_tree\ rl$ $splay\ e\ rl = splay\ a\ rl$ $t_splay\ e\ rl = t_splay\ a\ rl$

```

    by blast
  moreover hence  $c < e$  &  $e < b$  using 11.prem1 by simp
  ultimately show ?case using  $\langle c < a \rangle \langle a < b \rangle \langle c < b \rangle \langle bst \ rr \rangle$  by force
next
case (14 c a b rr)
hence  $splay \ a \ rr \neq Leaf$  by simp
then obtain rrl u rrr where [simp]:  $splay \ a \ rr = Node \ rrl \ u \ rrr$ 
  by (metis tree.exhaust)
have  $c < b$  bst rr using 14.prem1 by auto
from 14.IH[OF  $\langle rr \neq Leaf \rangle \langle bst \ rr \rangle$ ]
obtain e where  $e \in set\_tree \ rr$   $splay \ e \ rr = splay \ a \ rr$   $t\_splay \ e \ rr = t\_splay \ a \ rr$ 
  by blast
moreover hence  $b < e$  using 14.prem2 by simp
ultimately show ?case using  $\langle c < a \rangle \langle b < a \rangle \langle c < b \rangle \langle bst \ rr \rangle$  by force
qed (auto simp: le_less)

```

```

datatype 'a op = Empty | Splay 'a | Insert 'a | Delete 'a

```

```

fun arity :: 'a::linorder op  $\Rightarrow$  nat where
arity Empty = 0 |
arity (Splay a) = 1 |
arity (Insert a) = 1 |
arity (Delete a) = 1

```

```

fun exec :: 'a::linorder op  $\Rightarrow$  'a tree list  $\Rightarrow$  'a tree where
exec Empty [] = Leaf |
exec (Splay a) [t] = splay a t |
exec (Insert a) [t] = Splay_Tree.insert a t |
exec (Delete a) [t] = Splay_Tree.delete a t

```

```

fun cost :: 'a::linorder op  $\Rightarrow$  'a tree list  $\Rightarrow$  nat where
cost Empty [] = 1 |
cost (Splay a) [t] = t_splay a t |
cost (Insert a) [t] = t_splay a t |
cost (Delete a) [t] = t_delete a t

```

```

end

```

5.2 Splay Tree Analysis

```

theory Splay_Tree_Analysis
imports

```


begin

5.2.1 Analysis of splay

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

lemma $a_splay_simps[simp]: a_splay \ a \ (Node \ l \ a \ r) = 1$
 $a < b \implies a_splay \ a \ (Node \ (Node \ ll \ a \ lr) \ b \ r) = \varphi \ (Node \ lr \ b \ r) - \varphi \ (Node \ ll \ a \ lr) + 1$
 $b < a \implies a_splay \ a \ (Node \ l \ b \ (Node \ rl \ a \ rr)) = \varphi \ (Node \ rl \ b \ l) - \varphi \ (Node \ rl \ a \ rr) + 1$
by(*auto simp add: a_splay_def algebra_simps*)

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 $splay2 \ (<) \ t = splay2 \ (not \ o \ (<)) \ (mirror \ t)$. This would simplify the code and the proofs.

lemma *zig_zig*: **fixes** $lx \ rx \ lb \ b \ rb \ a \ ra \ u \ lb1 \ lb2$
defines $[simp]: X == Node \ lx \ (x) \ rx$ **defines** $[simp]: B == Node \ lb \ b \ rb$
defines $[simp]: t == Node \ B \ a \ ra$ **defines** $[simp]: A' == Node \ rb \ a \ ra$
defines $[simp]: t' == Node \ lb1 \ u \ (Node \ lb2 \ b \ A')$
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** $[simp]: B' = Node \ lb2 \ b \ A'$
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 $\dots \leq 2 * \varphi \ lb + \varphi \ B' + \varphi \ A' - \varphi \ B - 3 * \varphi \ X + 1$
using *IH prems(2)* **by**(*auto simp: algebra_simps*)
also have $\dots \leq \varphi \ lb + \varphi \ B' + \varphi \ A' - 3 * \varphi \ X + 1$ **by**(*simp*)
also have $\dots \leq \varphi \ B' + 2 * \varphi \ t - 3 * \varphi \ X$
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
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 C)
  let ?A = Node (Node A x B) b C
  have x ∉ set_tree C using 3.prems(1) by auto
  with 3 show ?case using
    log_le_cancel_iff[of 2 size1(Node B b C) size1 ?A]
    log_le_cancel_iff[of 2 size1(Node A x B) size1 ?A]
    by (auto simp: algebra_simps simp del:log_le_cancel_iff)
next
  case (9 a x A B C)
  let ?A = ⟨A, a, ⟨B, x, C⟩⟩
  have x ∉ set_tree A using 9.prems(1) by auto
  with 9 show ?case using
    log_le_cancel_iff[of 2 size1(Node A a B) size1 ?A]
    log_le_cancel_iff[of 2 size1(Node B x C) size1 ?A]
    by (auto simp: algebra_simps simp del:log_le_cancel_iff)
next
  case (6 x b c AB C D)
  hence 0: x ∉ set_tree C ∧ x ∉ set_tree D using 6.prems(1) by auto
  hence 1: x ∈ set_tree AB using 6.prems ⟨x < b⟩ by (auto)

```

obtain $A a B$ **where** $sp: splay\ x\ AB = Node\ A\ a\ B$
using $splay_not_Leaf[OF\ \langle AB \neq Leaf \rangle]$ **by** $blast$
let $?X = Node\ l\ x\ r$ **let** $?ABC = Node\ AB\ b\ C$ **let** $?ABCD = Node\ ?ABC\ c\ D$
let $?AB = Node\ A\ a\ B$
let $?CD = Node\ C\ c\ D$ **let** $?BCD = Node\ B\ b\ ?CD$
have $a_splay\ x\ ?ABCD = a_splay\ x\ AB + \varphi\ ?BCD + \varphi\ ?CD - \varphi\ ?ABC - \varphi\ ?AB + 1$
using $6.prem\ 1\ sp$
by($auto\ simp: a_splay_def\ size_if_splay\ algebra_simps\ split: tree.split$)
also have $\dots \leq 3 * \varphi\ AB + \varphi\ ?BCD + \varphi\ ?CD - \varphi\ ?ABC - \varphi\ ?AB - 3 * \varphi\ ?X + 2$
using $6\ 0$ **by**($auto\ simp: algebra_simps$)
also have $\dots = 2 * \varphi\ AB + \varphi\ ?BCD + \varphi\ ?CD - \varphi\ ?ABC - 3 * \varphi\ ?X + 2$
using sp **by**($simp\ add: size_if_splay$)
also have $\dots \leq \varphi\ AB + \varphi\ ?BCD + \varphi\ ?CD - 3 * \varphi\ ?X + 2$ **by**($simp$)
also have $\dots \leq \varphi\ ?BCD + 2 * \varphi\ ?ABCD - 3 * \varphi\ ?X + 1$
using $sp\ ld_ld_1_less[of\ size1\ AB\ size1\ ?CD]$
by($simp\ add: size_if_splay$)
also have $\dots \leq 3 * \varphi\ ?ABCD - 3 * \varphi\ ?X + 1$
using sp **by**($simp\ add: size_if_splay$)
finally show $?case$ **by** $simp$
next
case ($8\ a\ x\ c\ BC\ A\ D$)
hence $0: x \notin set_tree\ A \wedge x \notin set_tree\ D$
using $8.prem\ 1\ \langle x < c \rangle$ **by**($auto$)
hence $1: x \in set_tree\ BC$ **using** $8.prem\ \langle a < x \rangle\ \langle x < c \rangle$ **by** ($auto$)
obtain $B\ b\ C$ **where** $sp: splay\ x\ BC = Node\ B\ b\ C$
using $splay_not_Leaf[OF\ \langle BC \neq Leaf \rangle]$ **by** $blast$
let $?X = Node\ l\ x\ r$ **let** $?ABC = Node\ A\ a\ BC$ **let** $?ABCD = Node\ ?ABC\ c\ D$
let $?BC = Node\ B\ b\ C$
let $?AB = Node\ A\ a\ B$ **let** $?CD = Node\ C\ c\ D$
have $a_splay\ x\ ?ABCD = a_splay\ x\ BC + \varphi\ ?AB + \varphi\ ?CD - \varphi\ ?ABC - \varphi\ ?BC + 1$
using $8.prem\ 1\ sp$
by($auto\ simp: a_splay_def\ size_if_splay\ algebra_simps\ split: tree.split$)
also have $\dots \leq 3 * \varphi\ BC + \varphi\ ?AB + \varphi\ ?CD - \varphi\ ?ABC - \varphi\ ?BC - 3 * \varphi\ ?X + 2$
using $8\ 0$ **by**($auto\ simp: algebra_simps$)
also have $\dots = 2 * \varphi\ BC + \varphi\ ?AB + \varphi\ ?CD - \varphi\ ?ABC - 3 * \varphi\ ?X + 2$
using sp **by**($simp\ add: size_if_splay$)

also have $\dots \leq \varphi BC + \varphi ?AB + \varphi ?CD - 3 * \varphi ?X + 2$ **by**(*simp*)
also have $\dots \leq \varphi BC + 2 * \varphi ?ABCD - 3 * \varphi ?X + 1$
using *sp ld.ld.1_less*[of *size1 ?AB size1 ?CD*]
by(*simp add: size_if_splay*)
also have $\dots \leq 3 * \varphi ?ABCD - 3 * \varphi ?X + 1$ **by**(*simp*)
finally show *?case* **by** *simp*
next
case (11 *a x c BC A D*)
hence 0: $x \notin \text{set_tree } D \wedge x \notin \text{set_tree } A$
using 11.prem1 (1) $\langle a < x \rangle$ **by** (*auto*)
hence 1: $x \in \text{set_tree } BC$ **using** 11.prem1 $\langle a < x \rangle \langle x < c \rangle$ **by** (*auto*)
obtain *B b C* **where** *sp: splay x BC = Node B b C*
using *splay_not_Leaf*[OF $\langle BC \neq \text{Leaf} \rangle$] **by** *blast*
let *?X = Node l x r* **let** *?BCD = Node BC c D* **let** *?ABCD = Node A*
a ?BCD
let *?BC = Node B b C*
let *?CD = Node C c D* **let** *?AB = Node A a B*
have $a_splay\ x\ ?ABCD = a_splay\ x\ BC + \varphi ?CD + \varphi ?AB - \varphi ?BCD$
 $- \varphi ?BC + 1$
using 11.prem1 *sp*
by(*auto simp: a_splay_def size_if_splay algebra_simps split: tree.split*)
also have $\dots \leq 3 * \varphi BC + \varphi ?CD + \varphi ?AB - \varphi ?BCD - \varphi ?BC -$
 $3 * \varphi ?X + 2$
using 11 0 **by**(*auto simp: algebra_simps*)
also have $\dots = 2 * \varphi BC + \varphi ?CD + \varphi ?AB - \varphi ?BCD - 3 * \varphi ?X$
 $+ 2$
using *sp* **by**(*simp add: size_if_splay*)
also have $\dots \leq \varphi BC + \varphi ?CD + \varphi ?AB - 3 * \varphi ?X + 2$ **by**(*simp*)
also have $\dots \leq \varphi BC + 2 * \varphi ?ABCD - 3 * \varphi ?X + 1$
using *sp ld.ld.1_less*[of *size1 ?CD size1 ?AB*]
by(*simp add: size_if_splay algebra_simps*)
also have $\dots \leq 3 * \varphi ?ABCD - 3 * \varphi ?X + 1$ **by**(*simp*)
finally show *?case* **by** *simp*
next
case (14 *a x b CD A B*)
hence 0: $x \notin \text{set_tree } B \wedge x \notin \text{set_tree } A$
using 14.prem1 (1) $\langle b < x \rangle$ **by**(*auto*)
hence 1: $x \in \text{set_tree } CD$ **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 $\langle CD \neq \text{Leaf} \rangle$] **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 a : set_tree t*
shows $a_splay\ a\ t \leq 3 * (\varphi\ t - 1) + 1$
proof –
from *assms(2)* **obtain** *la ra* **where** *N: Node la a ra : subtrees t*
by (*metis set_treeE*)
have $a_splay\ a\ t \leq 3 * (\varphi\ t - \varphi(Node\ la\ a\ ra)) + 1$ **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** $a_splay\ a\ t \leq 3 * \varphi\ t + 1$
proof *cases*
assume *t = Leaf* **thus** *?thesis* **by**(*simp add: a_splay_def*)
next
assume *t ≠ Leaf*
from *ex_in_set_tree[OF this assms]* **obtain** *a'* **where**
a': a' ∈ set_tree t splay a' t = splay a t t_splay a' t = t_splay a t
by *blast*
have [*arith*]: $\log\ 2\ 2 > 0$ **by** *simp*
show *?thesis* **using** *a_splay_ub2[OF assms a'(1)]* **by**(*simp add: a_splay_def a' log_divide*)
qed

5.2.2 Analysis of insert

lemma *amor_insert*: **assumes** *bst t*
shows $t_splay\ a\ t + \Phi(Splay_Tree.insert\ a\ t) - \Phi\ t \leq 4 * \log\ 2\ (size1\ t) + 2$ (**is** $?l \leq ?r$)
proof *cases*
assume *t = Leaf* **thus** *?thesis* **by**(*simp*)
next
assume *t ≠ 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 = Φ l + Φ r* **let** *?Ps = Φ t*
let *?slr = real(size1 l) + real(size1 r)* **let** *?LR = log 2 (1 + ?slr)*
have *opt*: $?t + \Phi(splay\ a\ t) - ?Ps \leq 3 * \log\ 2\ (real\ (size1\ t)) + 1$
using *a_splay_ub3[OF ‹bst t›, simplified a_splay_def, of a]* **by** (*simp*)
from *less_linear[of e a]*
show *?thesis*
proof (*elim disjE*)

```

assume  $e=a$ 
have  $nneg: \log 2 (1 + \text{real}(\text{size } t)) \geq 0$  by simp
thus  $?thesis$  using  $\langle t \neq \text{Leaf} \rangle \text{opt } \langle e=a \rangle$ 
  apply(simp add: algebra_simps) using nneg by arith
next
let  $?L = \log 2 (\text{real}(\text{size1 } l) + 1)$ 
assume  $e < a$  hence  $e \neq a$  by simp
hence  $?l = (?t + ?Plr - ?Ps) + ?L + ?LR$ 
  using  $\langle t \neq \text{Leaf} \rangle \langle e < a \rangle$  by(simp)
also have  $?t + ?Plr - ?Ps \leq 2 * \log 2 ?slr + 1$ 
  using opt size_splay[of a 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 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  $?l = (?t + ?Plr - ?Ps) + ?R + ?LR$ 
  using  $\langle t \neq \text{Leaf} \rangle \langle a < e \rangle$  by(simp)
also have  $?t + ?Plr - ?Ps \leq 2 * \log 2 ?slr + 1$ 
  using opt size_splay[of a 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 a t, symmetric] by simp
qed
qed

```

5.2.3 Analysis of delete

definition $a_splay_max :: 'a::\text{linorder tree} \Rightarrow \text{real}$ **where**
 $a_splay_max t = t_splay_max t + \Phi(\text{splay_max } t) - \Phi t$

```

lemma a_splay_max_ub:  $\llbracket \text{bst } t; t \neq \text{Leaf} \rrbracket \implies a\_splay\_max\ t \leq 3 * (\varphi\ t - 1) + 1$ 
proof(induction t rule: splay_max.induct)
  case 1 thus ?case by (simp)
next
  case (2 l b)
  thus ?case using one_le_log_cancel_iff[of 2 size1 l + 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 = 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  $\dots \leq 3 * (\varphi\ rr - 1) + \varphi\ ?B' + \varphi\ ?C' - \varphi\ rr - \varphi\ ?C + 2$ 
      using 3 (rr ≠ Leaf) by auto
    also have  $\dots = 2 * \varphi\ rr + \varphi\ ?B' + \varphi\ ?C' - \varphi\ ?C - 1$  by simp
    also have  $\dots \leq \varphi\ rr + \varphi\ ?B' + \varphi\ ?C' - 1$  by simp
    also have  $\dots \leq 2 * \varphi\ ?B + \varphi\ ?C' - 2$ 
      using ld_ld_1_less[of size1 ?B' size1 rr] by(simp add:)
    also have  $\dots \leq 3 * \varphi\ ?B - 2$  using 1 by simp
    finally show ?case by simp
  qed
qed

lemma a_splay_max_ub3: assumes bst t shows  $a\_splay\_max\ t \leq 3 * \varphi\ t + 1$ 
proof cases

```

```

  assume  $t = \text{Leaf}$  thus ?thesis by (simp add: a_splay_max_def)
next
  assume  $t \neq \text{Leaf}$ 
  have [arith]:  $\log 2 2 > 0$  by simp
  show ?thesis using a_splay_max_ub[OF assms  $\langle t \neq \text{Leaf} \rangle$ ] by (simp add:
a_splay_max_def)
qed

lemma amor_delete: assumes bst t
shows  $t\_delete\ a\ t + \Phi(\text{Splay\_Tree.delete}\ a\ t) - \Phi\ t \leq 6 * \log 2\ (\text{size1}\ t) + 2$ 
proof (cases t)
  case Leaf thus ?thesis by (simp add: Splay_Tree.delete_def t_delete_def)
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 =  $\text{real}(\text{t\_splay}\ a\ t)$ 
  let ?Plr =  $\Phi\ l + \Phi\ r$  let ?Ps =  $\Phi\ t$ 
  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  $\geq 0$  by simp
  have opt:  $?t + \Phi(\text{splay}\ a\ t) - ?Ps \leq 3 * \log 2\ (\text{real}(\text{size1}\ t)) + 1$ 
  using a_splay_ub3[OF  $\langle \text{bst}\ t \rangle$ , 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 t_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:  $\log 2\ (\text{real}(\text{size}\ r) + 2) \geq 0$  by (simp)
      show ?thesis
      using Leaf opt apply (simp add: Splay_Tree.delete_def t_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]:  $\text{splay\_max}\ (\text{Node}\ ll\ y\ lr) = \text{Node}\ l'\ y'\ r'$ 

```



```

using splay_max_Leaf_iff tree.exhaust by blast
have bst l using bst_splay[OF ‹bst t›, of a] by simp
have  $\Phi r' \geq 0$  apply (induction r') by (auto)
have optm:  $\text{real}(t\_splay\_max\ l) + \Phi (splay\_max\ l) - \Phi l \leq 3 * \varphi\ l +$ 
1
using a_splay_max_ub3[OF ‹bst l›, simplified a_splay_max_def] by
(simp add: field_simps Node)
have 1:  $\log 2 (2 + (\text{real}(size\ l') + \text{real}(size\ r))) \leq \log 2 (2 + (\text{real}(size\ l) + \text{real}(size\ r)))$ 
using size_splay_max[of l] Node by simp
have 2:  $\log 2 (2 + (\text{real}(size\ l') + \text{real}(size\ r))) \geq 0$  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:  $\log 2 (\text{real}(size\ ll) + (\text{real}(size\ lr) + 2)) \leq ?slr$ 
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 t_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) + 2 |
U (Delete _) [t] = 6 * log 2 (size1 t) + 2

```

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 ≤ ?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 3 by auto
  thus ?thesis using Splay a_splay_ub3[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 3 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 3 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  $\alpha :: \text{real}$  and  $\beta :: \text{real}$ 
assumes  $a1[\text{arith}]$ :  $\alpha > 1$ 
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( $\text{rule powr\_le\_cancel\_iff}[THEN \text{iff}D1, OF a1]$ )
apply( $\text{simp add: powr\_add mult.commute[of } \beta] \text{ powr\_powr[symmetric] } A1$ )
done

```

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

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

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

lemma $A.\text{simps}[\text{simp}]$: $A \ a \ (\text{Node } l \ a \ r) = 1$
 $a < b \implies A \ a \ (\text{Node } (\text{Node } ll \ a \ lr) \ b \ r) = \varphi \ lr \ r - \varphi \ lr \ ll + 1$
 $b < a \implies A \ a \ (\text{Node } l \ b \ (\text{Node } rl \ a \ rr)) = \varphi \ rl \ l - \varphi \ rr \ rl + 1$
by($\text{auto simp add: } A.\text{def } \varphi.\text{def algebra_simps}$)

lemma $A.\text{ub}$: $\llbracket \text{bst } t; \text{Node } la \ a \ ra : \text{subtrees } t \rrbracket$
 $\implies A \ a \ t \leq \log \alpha ((\text{size1 } t)/(\text{size1 } la + \text{size1 } ra)) + 1$
proof($\text{induction } a \ t \ \text{rule: splay.induct}$)
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 (3 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:  $\varphi\_def \log\_divide \text{algebra\_simps}$ )
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:  $\varphi\_def \log\_divide \text{algebra\_simps}$ )
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.prem1 ( $x < b$ ) by (auto)
  then obtain lu u ru where sp:  $\text{splay } x \text{ lb} = \text{Node } lu \ u \ ru$ 
  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  $\varphi\_def \text{size\_if\_splay algebra\_simps log\_divide}$ )
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** $8.\text{prems}(1)$ **by** (*auto*)
hence 1: $x \in \text{set_tree } rb$ **using** $8.\text{prems } \langle b < x \rangle \langle x < a \rangle$ **by** (*auto*)
then obtain $lu\ u\ ru$ **where** $sp: \text{splay } x\ rb = \text{Node } lu\ u\ ru$
using $8.\text{prems}(1,2)$ **by**(*cases splay x rb*) *auto*
let $?lu = \text{real } (size1\ lu)$ **let** $?ru = \text{real } (size1\ ru)$
let $?lb = \text{real}(size1\ lb)$ **let** $?r = \text{real}(size1\ ra)$
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** $?L \leq ?R$)
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 β] powr_powr [symmetric]*)
qed
thus $?case$ **using** $8\ 0\ 1\ sp$
by(*auto simp add: A_def size_if_splay φ _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** $11.\text{prems}(1)$ **by** (*auto*)
hence 1: $x \in \text{set_tree } lb$ **using** $11.\text{prems } \langle a < x \rangle \langle x < b \rangle$ **by** (*auto*)
then obtain $lu\ u\ ru$ **where** $sp: \text{splay } x\ lb = \text{Node } lu\ u\ ru$
using $11.\text{prems}(1,2)$ **by**(*cases splay x lb*) *auto*
let $?lu = \text{real } (size1\ lu)$ **let** $?ru = \text{real } (size1\ ru)$
let $?l = \text{real}(size1\ la)$ **let** $?rb = \text{real}(size1\ rb)$
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** $?L \leq ?R$)
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** $14.\text{prems}(1)$ **by**(*auto*)
hence 1: $x \in \text{set_tree } rb$ **using** $14.\text{prems } \langle a < x \rangle \langle b < x \rangle$ **by** (*auto*)
then obtain $lu\ u\ ru$ **where** $sp: \text{splay } x\ rb = \text{Node } lu\ u\ ru$
using $14.\text{prems}(1,2)$ **by**(*cases splay x rb*) *auto*
let $?la = \text{real}(size1\ la)$ **let** $?lb = \text{real}(size1\ lb)$
let $?lu = \text{real } (size1\ lu)$ **let** $?ru = \text{real } (size1\ ru)$
have $1 + \log \alpha (?lu + ?ru) + \beta * \log \alpha (?la + ?lb) + \beta * \log \alpha (?lu +$

$?la + ?lb) \leq$
 $\beta * \log \alpha (?lu + ?ru) + \beta * \log \alpha (?lu + ?lb + ?ru) + \log \alpha (?lu$
 $+ ?lb + ?ru + ?la) \text{ (is } ?L \leq ?R)$
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 $\dots \leq \log \alpha ((\text{size1 } t)/2) + 1$ **by**(*simp add: field_simps*)
finally show *?thesis* **by** *simp*
qed

lemma *A_ub3*: **assumes** *bst t* **shows** $A a t \leq \log \alpha (\text{size1 } t) + 1$
proof *cases*
assume $t = \text{Leaf}$ **thus** *?thesis* **by**(*simp add: A_def*)
next
assume $t \neq \text{Leaf}$
from *ex.in_set_tree*[*OF this assms*] **obtain** *a'* **where**
 $a' : a' \in \text{set_tree } t \text{ splay } a' t = \text{splay } a t \text{ t.splay } a' t = \text{t.splay } a t$
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**
 $Am t = \text{t.splay_max } t + \Phi(\text{splay_max } t) - \Phi t$

lemma *Am_simp3'*: $\llbracket c < b; \text{bst } rr; rr \neq \text{Leaf} \rrbracket \Longrightarrow$
 $Am (\text{Node } l c (\text{Node } rl b rr)) =$
 $(\text{case } \text{splay_max } rr \text{ of } \text{Node } rrl _ rrr \Rightarrow$
 $Am rr + \varphi rrl (\text{Node } l c rl) + \varphi l rl - \varphi rl rr - \varphi rrl rrr + 1)$
by(*auto simp: Am_def φ _def size_if_splay_max algebra_simps neq_Leaf_iff*
split: tree.split)

```

lemma Am_ub:  $\llbracket \text{bst } t; t \neq \text{Leaf} \rrbracket \implies \text{Am } t \leq \log \alpha ((\text{size1 } t)/2) + 1$ 
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  $rr = \text{Leaf}$ 
    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: splay_max  $rr = \text{Node } l' u r'$ 
    using splay_max.Leaf_iff tree.exhaust by blast
    hence 1:  $\text{size } rr = \text{size } l' + \text{size } r' + 1$ 
    using size_splay_max[of rr] by(simp)
    let ?l = real (size1 l) let ?rl = real (size1 rl)
    let ?l' = real (size1 l') let ?r' = 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 ≤ ?R)
    proof(rule powr_le_cancel_iff[THEN iffD1, OF a1])
      show  $\alpha \text{ powr } ?L \leq \alpha \text{ powr } ?R$  using A2[of ?r' ?l' ?rl ?l]
      by(simp add: powr_add add.commute add.left_commute mult.commute[of
     $\beta$ ] powr_powr[symmetric])
    qed
    thus ?case using  $\exists$  sp 1 ( $rr \neq \text{Leaf}$ )
    by(auto simp add: Am_simp3'  $\varphi$ _def log_divide algebra_simps)
  qed
qed

```

lemma *Am_ub3*: **assumes** *bst t* **shows** $\text{Am } t \leq \log \alpha (\text{size1 } t) + 1$

```

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 assms (t ≠ Leaf)] 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: real_root_mult 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: real_root_mult 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*: $\llbracket (l'::\text{real}) \geq 1; r' \geq 1; lr \geq 1; r \geq 1 \rrbracket \implies$

$4 * (l' + r') * (lr + r) \leq (l' + lr + r' + r) \wedge 2$

by (*sos (((A < 0 * R < 1) + (R < 1 * (R < 1 * [r + ~1 * l' + lr + ~1 * r'] ^ 2))))*)

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

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

proof –

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

have 2: $0 \leq (l' + lr + r' + r) \wedge 2$ **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 $4 * (l' + r') \wedge 2 * (lr + r) * (lr + r' + r)$

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

proof–

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

have 2: $0 \leq (l' + lr + r' + r) \wedge 3$ **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** $(l'::real) \geq 1 \ r' \geq 1 \ ll \geq 1 \ r \geq 1$
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** $(l'::real) \geq 1 \ r' \geq 1 \ ll \geq 1 \ r \geq 1$
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** $(l'::real) \geq 1 \ r' \geq 1 \ ll \geq 1 \ r \geq 1$
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*: $x > (0::real) \implies c * x^2 * y * z \leq u * v \implies c * x^3 * y * z \leq x * u * v$
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  $\leq (3/2) * \log 2 (size1 t) + 1$ 
proof –
  have S34.A a t  $\leq \log (root\ 3\ 4) (size1 t) + 1$  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.Φ 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 ≤ log 2 ?slr / 2 by (simp)
    also have ?LR/2 ≤ log 2 ?slr / 2 + 1/2
  proof -
    have ?LR/2 ≤ log 2 (2 * ?slr) / 2 by simp
    also have ... ≤ 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 + real(size r))
  assume a<e hence e ≠ a by simp
  hence ?l = (?t + ?Plr - ?Ps) + ?R / 2 + ?LR / 2
    using ⟨t ≠ Leaf⟩ ⟨a<e⟩ 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 ?R/2 ≤ log 2 ?slr / 2 by (simp)
  also have ?LR/2 ≤ log 2 ?slr / 2 + 1/2
  proof -
    have ?LR/2 ≤ log 2 (2 * ?slr) / 2 by simp
    also have ... ≤ 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 bst t using 3 by auto
show ?thesis
proof (cases t)
  case Leaf thus ?thesis
    by (simp add: Splay_Tree.delete_def t_delete_def S34.φ_def log4_log2)
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 = 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)
let ?lslr = log 2 (real (size ls) + (real (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 t_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 t_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)) ≤ ?lsr
    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 t_delete_def field_simps)
    using 4 ‹S34.Φ r' ≥ 0› by arith
  qed
  qed
  qed
  qed

```

```

end
theory Priority_Queue_ops
imports Main
begin

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

fun arity :: 'a op ⇒ 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 φ 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 + Φ (del_min t) - Φ t \leq 2 * φ 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  $\dots \leq 2 + 2 * \varphi ll + \Phi ll - \Phi ?l' + \Phi (del\_min ?t) - \Phi ?t$ 
  using  $3 ll$  by linarith
also have  $\dots = 2 + 2 * \varphi ll + \varphi ?t' + \varphi ?s' - \varphi ?t - \varphi ?s$  by(simp)
also have  $\dots \leq 2 + \varphi ll + \varphi ?s'$  using  $0 1$  by linarith
also have  $\dots < 2 * \varphi ?t + 1$  using  $2 ld\_ld\_1\_less[of size1 ll size1 ?s']$ 
  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  $t == Node s a (Node u b r)$  and  $t' == Node (Node s a u) b r1'$ 
assumes  $size r1' \leq size r$ 
   $t\_part p r + \Phi r1' + \Phi r2' - \Phi r \leq 2 * \varphi r + 1$ 
shows  $t\_part p r + 1 + \Phi t' + \Phi r2' - \Phi t \leq 2 * \varphi t + 1$ 
proof -
  have  $1: \varphi r \leq \varphi (Node u b r)$  by (simp add: size1_size)
  have  $2: \log 2 (real (size1 s + size1 u + size1 r1')) \leq \varphi t$ 
    using assms(3) by (simp add: t_def size1_size)
  from  $ld\_ld\_1\_less[of size1 s + size1 u size1 r]$ 
  have  $1 + \varphi r + \log 2 (size1 s + size1 u) \leq 2 * \log 2 (size1 s + size1 u + size1 r)$ 
    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  $t \equiv Node s a (Node r b u)$  and  $t1' == Node s a r1'$  and  $t2' \equiv Node u b r2'$ 
assumes  $size r = size r1' + size r2'$ 
   $t\_part p r + \Phi r1' + \Phi r2' - \Phi r \leq 2 * \varphi r + 1$ 
shows  $t\_part p r + 1 + \Phi t1' + \Phi t2' - \Phi t \leq 2 * \varphi t + 1$ 

```



```

proof –
  have 1:  $\varphi r \leq \varphi (\text{Node } u \text{ } b \text{ } r)$  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  $a \leq p$ 
    show ?thesis
    proof (cases r)
      case Leaf thus ?thesis using  $\langle a \leq p \rangle 2.\text{prems}$  by fastforce
    next
      case [simp]: (Node rl b rr)
      let ?t = Node l a r
      show ?thesis
      proof cases
        assume  $b \leq p$ 
        with  $\langle a \leq p \rangle 2.\text{prems}$  obtain rrl
          where 0: partition p rr = (rrl, r')  $l' = \text{Node } (\text{Node } l \ a \ rl) \ b \ rrl$ 
          by (auto split: tree.splits prod.splits)
          have  $\text{size } rrl \leq \text{size } rr$ 
          using size_partition[OF 0(1)] by (simp add: size1_size)
          with 0  $\langle a \leq p \rangle \langle b \leq p \rangle 2.\text{prems}(1) 2.IH(1)$ [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  $\neg b \leq p$ 
        with  $\langle a \leq p \rangle 2.\text{prems}$  obtain rll rlr
          where 0: partition p rl = (rll, rlr)  $l' = \text{Node } l \ a \ rll \ r' = \text{Node } rlr \ b \ rr$ 
          by (auto split: tree.splits prod.splits)
          from 0  $\langle a \leq p \rangle \langle \neg b \leq p \rangle 2.\text{prems}(1) 2.IH(2)$ [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  $\neg a \leq p$ 
  show ?thesis
  proof (cases l)
    case Leaf thus ?thesis using  $\langle \neg a \leq p \rangle$  2.prem1 by fastforce
  next
    case [simp]: (Node ll b lr)
    let ?t = Node l a r
    show ?thesis
    proof cases
      assume  $b \leq p$ 
      with  $\langle \neg a \leq p \rangle$  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  $\langle \neg a \leq p \rangle$   $\langle b \leq p \rangle$  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'=lr
and p=p, of a b]
      show ?thesis by (auto simp: algebra_simps)
    next
      assume  $\neg b \leq p$ 
      with  $\langle \neg a \leq p \rangle$  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  $\leq$  size ll
      using size_partition[OF 0(1)] by (simp add: size1_size)
      with 0  $\langle \neg a \leq p \rangle$   $\langle \neg b \leq p \rangle$  2.prem1 2.IH(4)[OF - Node, of l'
llr]
      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 \Rightarrow 'a tree list \Rightarrow '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 Del_min [t] = 2 * φ 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 + size t) < log 2 (2 + size t) **by** simp

with 1 amor_partition[OF ⟨bst_wrt (\leq) t⟩ 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 Φ :: 'a tree \Rightarrow real **where**

Φ Leaf = 0

| Φ (Node l _ r) = Φ l + Φ r + log 2 (1 + size l + size r)

lemma *link_size[simp]*: size (link h) = size h

by (cases h rule: link.cases) simp_all

lemma *size_pass1*: size (pass₁ h) = size h

by (induct h rule: pass₁.induct) simp_all

lemma *size_pass2*: size (pass₂ h) = size h

by (induct h 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* h \implies Φ (insert x h) - Φ h \leq log 2 (size h + 1)

by (simp split: tree.splits)

lemma $\Delta\Phi$ _merge:

assumes h1 = Node lx x Leaf h2 = Node ly y Leaf

shows Φ (merge h1 h2) - Φ h1 - Φ h2 \leq log 2 (size h1 + size h2) + 1

proof -

let ?hs = Node lx x (Node ly y Leaf)

have Φ (merge h1 h2) = Φ (link ?hs) **using** assms **by** simp

also have $\dots = \Phi \text{ lx} + \Phi \text{ ly} + \log 2 (\text{size lx} + \text{size ly} + 1) + \log 2 (\text{size lx} + \text{size ly} + 2)$
by (*simp add: algebra_simps*)
also have $\dots = \Phi \text{ lx} + \Phi \text{ ly} + \log 2 (\text{size lx} + \text{size ly} + 1) + \log 2 (\text{size h1} + \text{size h2})$
using *assms* **by** *simp*
finally have $\Phi (\text{merge h1 h2}) = \dots$
have $\Phi (\text{merge h1 h2}) - \Phi \text{ h1} - \Phi \text{ h2} =$
 $\log 2 (\text{size lx} + \text{size ly} + 1) + \log 2 (\text{size h1} + \text{size h2})$
 $- \log 2 (\text{size lx} + 1) - \log 2 (\text{size ly} + 1)$
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 lx size ly*] *assms* **by** (*simp add: algebra_simps*)
finally show *?thesis* .
qed

fun *upperbound* :: 'a tree \Rightarrow real **where**
upperbound Leaf = 0
| *upperbound (Node _ _ Leaf)* = 0
| *upperbound (Node lx _ (Node ly _ Leaf))* = $2 * \log 2 (\text{size lx} + \text{size ly} + 2)$

| *upperbound (Node lx _ (Node ly _ ry))* = $2 * \log 2 (\text{size lx} + \text{size ly} + \text{size ry} + 2)$
 $- 2 * \log 2 (\text{size ry}) - 2 + \text{upperbound ry}$

lemma $\Delta \Phi_{\text{pass1_upperbound}}$: $\Phi (\text{pass}_1 \text{ hs}) - \Phi \text{ hs} \leq \text{upperbound hs}$

proof (*induction hs rule: upperbound.induct*)

case ($\exists \text{ lx } x \text{ ly } y$)

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 $\dots \leq \log 2 (\text{size lx} + \text{size ly} + 2)$ **by** *simp*

also have $\dots \leq 2 * \dots$ **by** *simp*

finally show *?case* **by** (*simp add: algebra_simps*)

next

case ($\exists \text{ lx } x \text{ ly } y \text{ lz } z \text{ rz}$)

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 $\Phi (\text{pass}_1 \text{ ?h}) - \Phi \text{ ?h} \leq ?t + \text{upperbound ?ry}$

using *4.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 $?l \leq 2 * \log 2 (size\ lx + size\ ly + size\ ?ry + 1) + \log 2 (size\ ?ry)$
using *ld_sum_inequality* [of *size lx + size ly + 1 size ?ry*] **by** *simp*
also have $\dots \leq 2 * \log 2 (size\ lx + size\ ly + size\ ?ry + 2) + \log 2 (size\ ?ry)$
by *simp*
also have $\dots \leq ?r$ **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** $upperbound\ hs \leq 2 * \log 2 (size\ hs) - len\ hs + 2$
by (*induct hs rule: upperbound.induct*) (*simp_all add: algebra_simps*)
thus *?thesis* **using** $\Delta\Phi_pass1_upperbound$ [of *hs*] *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 $\Delta\Phi_{\text{mergepairs}}$: **assumes** $hs \neq \text{Leaf}$
shows $\Phi (\text{merge_pairs } hs) - \Phi hs \leq 3 * \log 2 (\text{size } hs) - \text{len } hs + 2$
proof –
have $pass_1 hs \neq \text{Leaf}$ **by** (*metis* *assms* *eq_size_0* *size_pass1*)
with *assms* $\Delta\Phi_{\text{pass1}}$ [*of* *hs*] $\Delta\Phi_{\text{pass2}}$ [*of* $pass_1 hs$]
show *?thesis* **by** (*auto* *simp* *add*: *size_pass1* *pass12_merge_pairs*)
qed

lemma $\Delta\Phi_{\text{del_min}}$: **assumes** $lx \neq \text{Leaf}$
shows $\Phi (\text{del_min } (\text{Node } lx x \text{ Leaf})) - \Phi (\text{Node } lx x \text{ Leaf})$
 $\leq 3 * \log 2 (\text{size } lx) - \text{len } lx + 2$
proof –
let $?h = \text{Node } lx x \text{ Leaf}$
let $?\Delta\Phi_1 = \Phi lx - \Phi ?h$
let $?\Delta\Phi_2 = \Phi (\text{pass}_2 (\text{pass}_1 lx)) - \Phi lx$
let $?\Delta\Phi = \Phi (\text{del_min } ?h) - \Phi ?h$
have $lx \neq \text{Leaf} \implies \Phi (\text{pass}_2 (\text{pass}_1 lx)) - \Phi (\text{pass}_1 lx) \leq \log 2 (\text{size } lx)$
using $\Delta\Phi_{\text{pass2}}$ [*of* $pass_1 lx$] **by** (*metis* *eq_size_0* *size_pass1*)
moreover **have** $lx \neq \text{Leaf} \implies \Phi (\text{pass}_1 lx) - \Phi lx \leq 2 * \dots - \text{len } lx + 2$
using $\Delta\Phi_{\text{pass1}}$ **by** *metis*
moreover **have** $?\Delta\Phi \leq ?\Delta\Phi_2$ **by** *simp*
ultimately show *?thesis* **using** *assms* **by** *linarith*
qed

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

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

lemma *is_root_del_min*:
assumes $is_root h$ **shows** $is_root (\text{del_min } h)$
proof (*cases* *h*)
case [*simp*]: ($\text{Node } lx x rx$)
have $rx = \text{Leaf}$ **using** *assms* **by** *simp*
thus *?thesis*
proof (*cases* *lx*)
case ($\text{Node } ly y ry$)

```

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) ≤ len h
by (induct h rule: pass1.induct) simp_all

```

```

fun exec :: 'a :: linorder op ⇒ 'a tree list ⇒ '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 tpass1 :: 'a tree ⇒ nat where
  tpass1 Leaf = 1
| tpass1 (Node _ _ Leaf) = 1
| tpass1 (Node _ _ (Node _ _ ry)) = tpass1 ry + 1

```

```

fun tpass2 :: 'a tree ⇒ nat where
  tpass2 Leaf = 1
| tpass2 (Node _ _ rx) = tpass2 rx + 1

```

```

fun cost :: 'a :: linorder op ⇒ 'a tree list ⇒ nat where
  cost Empty [] = 1
| cost Del_min [Leaf] = 1
| cost Del_min [Node lx _ _] = tpass2 (pass1 lx) + tpass1 lx
| cost (Insert a) _ = 1
| cost Merge _ = 1

```

```

fun U :: 'a :: linorder op ⇒ 'a tree list ⇒ 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  $t_{pass2}(pass_1\ lx) + t_{pass1}\ lx \leq len\ lx + 2$ 
    by (induct lx rule: pass_1.induct) simp_all
  hence  $cost\ f\ ss \leq \dots$  by simp
  moreover have  $\Phi\ (del\_min\ h) - \Phi\ h \leq 3*\log\ 2\ (size\ h + 1) - len$ 
 $lx + 2$ 
  proof (cases lx)
    case [simp]: (Node ly y ry)
    have  $\Phi\ (del\_min\ h) - \Phi\ h \leq 3*\log\ 2\ (size\ lx) - len\ lx + 2$ 
      using  $\Delta\Phi_{del\_min}[of\ lx\ x]$  3 by simp
    also have  $\dots \leq 3*\log\ 2\ (size\ h + 1) - len\ lx + 2$  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  $\Delta\Phi_{\text{merge}}$ ) 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

```

7.2 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*: *pass₁ hs = [] \longleftrightarrow hs = []*

by(*cases hs rule: pass₁.cases*) *auto*

7.2.1 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*: $\llbracket \text{no_Empty } h1; \text{no_Empty } h2 \rrbracket \Longrightarrow \text{no_Empty}$
(*merge* *h1* *h2*)
by (*cases* (*h1,h2*) *rule: merge.cases*) *auto*

lemma *is_root_merge*: $\llbracket \text{is_root } h1; \text{is_root } h2 \rrbracket \Longrightarrow \text{is_root}$ (*merge* *h1* *h2*)
by (*cases* (*h1,h2*) *rule: merge.cases*) *auto*

lemma *no_Emptys_pass1*:
no_Emptys *hs* \Longrightarrow *no_Emptys* (*pass*₁ *hs*)
by(*induction* *hs* *rule: pass*₁.*induct*)(*auto simp: no_Empty_merge*)

lemma *is_root_pass2*: *no_Emptys* *hs* \Longrightarrow *is_root*(*pass*₂ *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*

7.2.2 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**

Φ_hps [] = 0 |

Φ_hps (*heap.Empty* # *hs*) = Φ_hps *hs* |

Φ_hps (*Hp* *x* *hsl* # *hsl*) =

Φ_hps *hsl* + Φ_hps *hsl* + $\log 2$ (*size_hps* *hsl* + *size_hps* *hsl* + 1)

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

Φ *heap.Empty* = 0 |

Φ (*Hp* _ *hs*) = Φ_hps *hs* + $\log 2$ (*size_hps*(*hs*)+1)

lemma Φ_hps_ge0 : Φ_hps *hs* \geq 0

by (*induction* *hs* *rule*: Φ_hps .*induct*) *auto*

lemma *no_Empty_ge0*: *no_Empty* *h* \implies *size_hp* *h* > 0

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

declare *algebra_simps*[*simp*]

lemma Φ_hps1 : Φ_hps [*h*] = Φ *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 $\Delta\Phi_insert$:

Φ (*Pairing_Heap_List1.insert* *x* *h*) - Φ *h* \leq $\log 2$ (*size_hp* *h* + 1)

by(*cases* *h*)(*auto* *simp*: *size_hp_merge*)

lemma $\Delta\Phi_merge$:

Φ (*merge* *h1* *h2*) - Φ *h1* - Φ *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 $\Delta\Phi_{\text{pass1_sum_ub}}$: no_Empty hs \Longrightarrow

Φ_{hps} (pass₁ hs) - Φ_{hps} hs \leq *sum_ub* hs (**is** _ \Longrightarrow ?P hs)

proof (*induction* hs rule: *sum_ub.induct*)

case (3 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: $\bigwedge x y::\text{real}. 0 \leq x \Longrightarrow x \leq y \Longrightarrow x \leq 2*y$ **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 Φ_{hps} (pass₁ ?h) - Φ_{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 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)

using s3 **by** (*simp add: log_divide*)

also have ... \leq -2

proof -

have $2 + \dots$
 $\leq 2 * \log 2 ((1 + \text{size_hps } h_{sx} + \text{size_hps } h_{sy}) / \text{size_hps } ?h + \text{size_hps } ?ry / \text{size_hps } ?h)$
using *ld_sum_inequality* [of $(1 + \text{size_hps } h_{sx} + \text{size_hps } h_{sy}) / \text{size_hps } ?h (\text{size_hps } ?ry / \text{size_hps } ?h)$] **using** *s3* **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 $\log 2 (1 + \text{size_hps } h_{sx} + \text{size_hps } h_{sy}) + \log 2 (\text{size_hps } ?ry) + 2$
 $\leq 2 * \log 2 (\text{size_hps } ?h)$ **by** *simp*
moreover have $\log 2 (\text{size_hps } ?ry) \leq \log 2 (\text{size_hps } ?rx)$ **using** *s3*
by *simp*
ultimately have $\log 2 (1 + \text{size_hps } h_{sx} + \text{size_hps } h_{sy}) - \dots$
 $\leq 2 * \log 2 (\text{size_hps } ?h) - 2 * \log 2 (\text{size_hps } ?ry) - 2$ **by** *linarith*
thus *?thesis* **by** *simp*
qed
finally show *?case* **by** (*simp*)
qed *simp_all*

lemma $\Delta\Phi_pass1$: **assumes** $hs \neq []$ *no_Empty* hs
shows $\Phi_hps (pass_1 hs) - \Phi_hps hs \leq 2 * \log 2 (\text{size_hps } hs) - \text{length } hs + 2$
proof –
have $\text{sum_ub } hs \leq 2 * \log 2 (\text{size_hps } hs) - \text{length } hs + 2$
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*: $hs \neq [] \implies \text{no_Empty } hs \implies$
 $\text{no_Empty}(pass_2 hs) \ \& \ \text{size_hps } hs = \text{size_hps}(hps(pass_2 hs)) + 1$
apply(*induction hs rule: Phi_hps.induct*)
apply (*fastforce simp: merge2 split: heap.split*)
done

lemma $\Delta\Phi_pass2$: $hs \neq [] \implies \text{no_Empty } hs \implies$
 $\Phi (pass_2 hs) - \Phi_hps hs \leq \log 2 (\text{size_hps } hs)$
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  $Cons.prem\ 2$  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  $(simp\ add: add\_increasing\ xt1\ 3)$   $[OF\ mult\_2,\ OF\ add\_increasing]$ 
  next
    case  $(Hp\ y\ hs3)$ 
    with  $Cons\ * size\_hps\_pass2$   $[of\ hs]$  show  $?thesis$  by  $(auto\ simp: add\_mono)$ 
  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 $t_{pass2} :: 'a \text{ heap list} \Rightarrow \text{nat}$ **where**
 $t_{pass2} [] = 1$
| $t_{pass2} (- \# hs) = 1 + t_{pass2} hs$

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

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

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 ($1 \text{ ss } f$) **show** *?thesis*

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*]: ($\text{Hp } _ \text{ } hs$)

show *?thesis*

proof *cases*

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

next

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

using $1(1) \text{ no_Empty_pass1}$ **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  $t_{pass2} (pass_1 hs) + t_{pass1} hs \leq 2 + length\ hs$ 
        by (induct hs rule:  $pass_1.induct$ ) 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  $\Phi (\text{merge } h1 \ h2) - \Phi \ h1 - \Phi \ h2 \leq \log 2 (\text{size\_hp } h1 + \text{size\_hp } h2 + 1) + 1$ 
      using  $\Delta\Phi\_merge[of \ h1 \ h2]$  by simp
      thus ?thesis by (simp)
    qed
  qed
end

```

7.3 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  $pass_1'$  == Pairing_Heap_List1.pass_1
abbreviation  $pass_2'$  == Pairing_Heap_List1.pass_2
abbreviation  $t_{pass_1}'$  == Pairing_Heap_List1_Analysis.t_{pass_1}
abbreviation  $t_{pass_2}'$  == Pairing_Heap_List1_Analysis.t_{pass_2}

```

```

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  $homs\_pass1'$ :  $no\_Emptys \ hs \Longrightarrow \ homs(pass_1' \ hs) = pass_1 \ (homs \ hs)$ 
apply (induction  $hs$  rule: Pairing_Heap_List1.pass_1.induct)
  apply simp
  subgoal for  $h$ 
  apply (case_tac  $h$ )
  apply (auto)
done

```

```

subgoal for  $h1\ h2$ 
apply(case_tac  $h1$ )
  apply simp
apply(case_tac  $h2$ )
  apply (auto)
done
done

```

```

lemma hom_merge':  $\llbracket no\_Emptyys\ lhs; Pairing\_Heap\_List1\_Analysis.is\_root\ h \rrbracket$ 
 $\implies hom\ (merge'\ (Hp\ x\ lhs)\ h) = link\ \langle homs\ lhs,\ x,\ hom\ h \rangle$ 
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)
  apply simp
  subgoal for  $h$ 
    apply(case_tac  $h$ )
    apply (auto)
    done
  subgoal for  $h1\ h2$ 
    apply(case_tac  $h1$ )
    apply simp
    apply(case_tac  $h2$ )
    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*: $is_root_pass2\ no_Emptyys_pass1$)

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

```

7.4 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  $\Rightarrow$  ('a hp  $\Rightarrow$  'b)  $\Rightarrow$  'a heap  $\Rightarrow$  'b where
lift_hp c f None = c |
lift_hp c f (Some h) = f h

```

```

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

```

```

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

```

```

fun  $\Phi$ _hps :: 'a hp list  $\Rightarrow$  real where
 $\Phi$ _hps [] = 0 |
 $\Phi$ _hps (Hp x hsl # hsr) =  $\Phi$ _hps hsl +  $\Phi$ _hps hsr + log 2 (size_hps hsl +
size_hps hsr + 1)

```

```

definition  $\Phi$ _hp :: 'a hp  $\Rightarrow$  real where
[simp]:  $\Phi$ _hp h =  $\Phi$ _hps (hps h) + log 2 (size_hps (hps h) + 1)

```

```

abbreviation  $\Phi$  :: 'a heap  $\Rightarrow$  real where
 $\Phi \equiv$  lift_hp 0  $\Phi$ _hp

```

```

abbreviation size_heap :: 'a heap  $\Rightarrow$  nat where
size_heap  $\equiv$  lift_hp 0 size_hp

```

lemma Φ_hps_ge0 : $\Phi_hps\ hs \geq 0$
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*: $link\ (Hp\ x\ lx)\ h = (case\ h\ of\ (Hp\ y\ ly) \Rightarrow$
(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 $\Phi_{hps}(\text{pass}_1 \text{ ?h}) - \Phi_{hps} \text{ ?h}$
 $\leq \log 2 (1 + \text{size_hps } hsx + \text{size_hps } hsy) - \log 2 (1 + \text{size_hps } hsy + \text{size_hps } ?ry) + \text{sum_ub } ?ry$
using 4.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 $\log 2 (1 + \text{size_hps } hsx + \text{size_hps } hsy) + \log 2 (\text{size_hps } ?ry) - 2 * \log 2 (\text{size_hps } ?h)$
 $= \log 2 ((1 + \text{size_hps } hsx + \text{size_hps } hsy) / (\text{size_hps } ?h)) + \log 2 (\text{size_hps } ?ry / \text{size_hps } ?h)$
by (simp add: log_divide)
also have ... ≤ -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 + size_hps hsx + size_hps hsy) / size_hps ?h (size_hps ?ry / size_hps ?h)] by simp
also have ... ≤ 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 $\log 2 (\text{size_hps } ?ry) \leq \log 2 (\text{size_hps } ?rx)$ 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** $hs \neq []$
shows $\Phi_{hps} (\text{pass}_1 hs) - \Phi_{hps} hs \leq 2 * \log 2 (\text{size_hps } hs) - \text{length } hs + 2$

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*: $pass_2\ hs = Some\ h \implies size_hps\ hs = size_hps(hps\ h)+1$

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 tpass1 :: 'a hp list  $\Rightarrow$  nat where
  tpass1 [] = 1
| tpass1 [-] = 1
| tpass1 (- # - # hs) = 1 + tpass1 hs
```

```
fun tpass2 :: 'a hp list  $\Rightarrow$  nat where
  tpass2 [] = 1
| tpass2 (- # hs) = 1 + tpass2 hs
```

```
fun cost :: 'a :: linorder op  $\Rightarrow$  'a heap list  $\Rightarrow$  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  $\Rightarrow$  'a heap list  $\Rightarrow$  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* = $\lambda_.$ *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 $\Delta\Phi$ _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  $t_{pass2} (pass_1 hs) + t_{pass1} hs \leq 2 + \text{length } hs$ 
            by (induct hs rule: pass_1.induct) simp_all
          hence  $\text{cost } f \text{ ss} \leq 1 + \dots$  by simp
          moreover have  $\Phi (\text{del\_min } ho) - \Phi ho \leq 3 * \log 2 (\text{size\_heap } ho + 1) - \text{length } hs + 2$ 
            proof (cases hs = [])
              case False
                hence  $\Phi (\text{del\_min } ho) - \Phi ho \leq 3 * \log 2 (\text{size\_hps } hs) - \text{length } hs + 2$ 
                  using  $\Delta\Phi_{\text{del\_min}}[\text{of } h]$  by simp
                also have  $\dots \leq 3 * \log 2 (\text{size\_heap } ho + 1) - \text{length } hs + 2$ 
                  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  $\Phi (\text{merge } ho1 ho2) - \Phi ho1 - \Phi ho2 \leq 2 * \log 2 (\text{size\_heap } ho1 + \text{size\_heap } ho2)$ 
              using  $\Delta\Phi_{\text{link}}[\text{of } h1 h2]$  by simp
            also have  $\dots \leq 2 * \log 2 (\text{size\_hp } h1 + \text{size\_hp } h2 + 1)$  by (simp)
            finally show ?thesis by (simp)
          qed
        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. Sedgwick, 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.