Script **generated by TTT**

Title: Lammich: FDS (29.06.2018)

Date: Fri Jun 29 08:35:57 CEST 2018

Duration: 86:22 min

Pages: 60

Chapter 9

Priority Queues

Implementation type

datatype
$'a\ lheap = Leaf \mid Node\ nat\ ('a\ tree)\ 'a\ ('a\ tree)$

Archive of Formal Proofs

`https://www.isa-afp.org/entries/Priority_Queue_Braun.shtml`

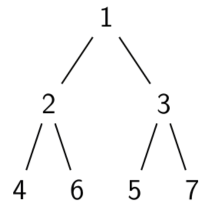
What is a Braun tree?

$braun :: {}'a\ tree \Rightarrow bool$

$braun\ \langle\rangle = True$

$braun\ \langle l,\ x,\ r\rangle =$

$(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun\ l \wedge braun\ r)$



Idea of invariant maintenance

$braun\ \langle\rangle = True$

$braun\ \langle l,\ x,\ r\rangle =$

$(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun\ l \wedge braun\ r)$

Priority queue implementation

Implementation type: $'a\ tree$

del_min

$del\_min :: 'a\ tree \Rightarrow 'a\ tree$

del_min

$del\_min :: 'a\ tree \Rightarrow 'a\ tree$

$del\_min\ \langle\rangle = \langle\rangle$
$del\_min\ \langle\langle\rangle,\ x,\ r\rangle = \langle\rangle$
$del\_min\ \langle l,\ x,\ r\rangle =$
$(\text{let } (y,\ l') = del\_left\ l \text{ in } sift\_down\ r\ y\ l')$

1. Delete leftmost element y
2. Sift y from the root down

sift_down

$sift\_down :: 'a\ tree \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a\ tree$

$sift\_down\ \langle\rangle\ a\ \langle\rangle = \langle\langle\rangle,\ a,\ \langle\rangle\rangle$
$sift\_down\ \langle\langle\rangle,\ x,\ \langle\rangle\rangle\ a\ \langle\rangle =$
$(\text{if } a \le x \text{ then } \langle\langle\langle\rangle,\ x,\ \langle\rangle\rangle,\ a,\ \langle\rangle\rangle$
$\ \text{else } \langle\langle\langle\rangle,\ a,\ \langle\rangle\rangle,\ x,\ \langle\rangle\rangle)$
$sift\_down\ (\langle l_1,\ x_1,\ r_1\rangle =: t_1)\ a\ (\langle l_2,\ x_2,\ r_2\rangle =: t_2) =$
$\text{if } a \le x_1 \wedge a \le x_2 \text{ then } \langle t_1,\ a,\ t_2\rangle$
$\text{else if } x_1 \le x_2 \text{ then } \langle sift\_down\ l_1\ a\ r_1,\ x_1,\ t_2\rangle$
$\quad \text{else } \langle t_1,\ x_2,\ sift\_down\ l_2\ a\ r_2\rangle$

Maintains *braun*

Sorting with priority queue

$pq\ [] = empty$
$pq\ (x\#xs) = insert\ x\ (pq\ xs)$

$mins\ q =$
$(\text{if } is\_empty\ q \text{ then } []$
$\ \text{else } get\_min\ h\ \#\ mins\ (del\_min\ h))$

Sorting with priority queue

$pq\ [] = empty$
$pq\ (x\#xs) = insert\ x\ (pq\ xs)$

$mins\ q =$
(if $is\_empty\ q$ then $[]$
else $get\_min\ h\ \#\ mins\ (del\_min\ h)$)

$sort\_pq = mins \circ pq$

HOL/Data_Structures/ Binomial_Heap.thy

Numerical method

Idea: only use trees t_i of size 2^i

Example

To store (in binary) 11001 elements: $[t_0,0,0,t_3,t_4]$

Numerical method

Idea: only use trees t_i of size 2^i

Example

To store (in binary) 11001 elements: $[t_0,0,0,t_3,t_4]$

Merge $\approx$ addition with carry

Numerical method

Idea: only use trees t_i of size 2^i

Example

To store (in binary) 11001 elements: $[t_0,0,0,t_3,t_4]$

Merge $\approx$ addition with carry

Needs function to combine two trees of size 2^i into one tree of size 2^{i+1}

Binomial tree

datatype *'a tree* =
  Node (rank: nat) (root: 'a) ('a tree list)

Binomial tree

datatype *'a tree* =
  Node (rank: nat) (root: 'a) ('a tree list)

Invariant: Node of rank r has children $[t_{r-1},\dots t_0]$ of ranks $[r-1,\dots,0]$

Binomial tree

datatype $'a\ tree =$
$\quad Node\ (rank\!: nat)\ (root\!: 'a)\ ('a\ tree\ list)$

Invariant: Node of rank r has children $[t_{r-1},\dots t_0]$
of ranks $[r{-}1,\dots,0]$

$invar\_btree\ (Node\ r\ x\ ts) =$
$((\forall\ t{\in}set\ ts.\ invar\_btree\ t) \wedge map\ rank\ ts = rev\ [0..{<}r])$

Binomial tree

datatype $'a\ tree =$
$\quad Node\ (rank\!: nat)\ (root\!: 'a)\ ('a\ tree\ list)$

Invariant: Node of rank r has children $[t_{r-1},\dots t_0]$
of ranks $[r{-}1,\dots,0]$

$invar\_btree\ (Node\ r\ x\ ts) =$
$((\forall\ t{\in}set\ ts.\ invar\_btree\ t) \wedge map\ rank\ ts = rev\ [0..{<}r])$

Lemma
$invar\_btree\ t \Longrightarrow |t| = 2^{rank\ t}$

Combining two trees

How to combine two trees of rank i
into one tree of rank $i{+}1$

$link\ (Node\ r\ x_1\ ts_1 =: t_1)\ (Node\ r'\ x_2\ ts_2 =: t_2) =$
(if $x_1 \le x_2$ then $Node\ (r+1)\ x_1\ (t_2\ \#\ ts_1)$
else $Node\ (r+1)\ x_2\ (t_1\ \#\ ts_2)$)

Binomial heap

Use sparse representation for binary numbers:
$[t_0,0,0,t_3,t_4]$ represented as $[\ (0,t_0),\ (3,t_3),(4,t_4)\ ]$

Combining two trees

How to combine two trees of rank i
into one tree of rank $i{+}1$

$link\ (Node\ r\ x_1\ ts_1 =: t_1)\ (Node\ r'\ x_2\ ts_2 =: t_2) =$
$(\text{if } x_1 \leq x_2 \text{ then } Node\ (r + 1)\ x_1\ (t_2\ \#\ ts_1)$
$\ \text{else } Node\ (r + 1)\ x_2\ (t_1\ \#\ ts_2))$

Binomial heap

Use sparse representation for binary numbers:
$[t_0,0,0,t_3,t_4]$ represented as $[\ (0,t_0),\ (3,t_3),(4,t_4)\ ]$

type_synonym $'a\ heap = {}'a\ tree\ list$

Inserting a tree

$ins\_tree\ t\ [] = [t]$
$ins\_tree\ t_1\ (t_2\ \#\ ts) =$
$(\text{if } rank\ t_1 < rank\ t_2 \text{ then } t_1\ \#\ t_2\ \#\ ts$
$\ \text{else } ins\_tree\ (link\ t_1\ t_2)\ ts)$

merge

$merge\ ts_1\ [] = ts_1$
$merge\ []\ ts_2 = ts_2$
$merge\ (t_1\ \#\ ts_1)\ (t_2\ \#\ ts_2) =$
$(\text{if } rank\ t_1 < rank\ t_2 \text{ then } t_1\ \#\ merge\ ts_1\ (t_2\ \#\ ts_2)$
$\ \text{else if } rank\ t_2 < rank\ t_1 \text{ then } t_2\ \#\ merge\ (t_1\ \#\ ts_1)\ ts_2$
$\quad \text{else } ins\_tree\ (link\ t_1\ t_2)\ (merge\ ts_1\ ts_2))$

$merge$

$merge\ ts_1\ [] = ts_1$
$merge\ []\ ts_2 = ts_2$
$merge\ (t_1\ \#\ ts_1)\ (t_2\ \#\ ts_2) =$
$(\text{if } rank\ t_1 < rank\ t_2 \text{ then } t_1\ \#\ merge\ ts_1\ (t_2\ \#\ ts_2)$
$\ \text{else if } rank\ t_2 < rank\ t_1 \text{ then } t_2\ \#\ merge\ (t_1\ \#\ ts_1)\ ts_2$
$\quad \text{else } ins\_tree\ (link\ t_1\ t_2)\ (merge\ ts_1\ ts_2))$

Intuition: Addition of binary numbers

Get/delete minimum element

All trees are min-heaps.

Smallest element may be any root node:

$ts \neq [] \Longrightarrow get\_min\ ts = Min\ (set\ (map\ root\ ts))$

Get/delete minimum element

All trees are min-heaps.

Smallest element may be any root node:

$ts \neq [] \Longrightarrow get\_min\ ts = Min\ (set\ (map\ root\ ts))$

Similar:
$get\_min\_rest :: 'a\ tree\ list \Rightarrow 'a\ tree \times 'a\ tree\ list$
Returns tree with minimal root, and remaining trees

Get/delete minimum element

All trees are min-heaps.

Smallest element may be any root node:

$ts \neq [] \Longrightarrow get\_min\ ts = Min\ (set\ (map\ root\ ts))$

Similar:
$get\_min\_rest :: 'a\ tree\ list \Rightarrow 'a\ tree \times 'a\ tree\ list$
Returns tree with minimal root, and remaining trees

$del\_min\ ts =$
$(\text{case } get\_min\_rest\ ts \text{ of}$
$\quad (Node\ r\ x\ ts_1,\ ts_2) \Rightarrow merge\ (rev\ ts_1)\ ts_2)$

Why rev?

Get/delete minimum element

All trees are min-heaps.

Smallest element may be any root node:

$ts \neq [] \Longrightarrow get\_min\ ts = Min\ (set\ (map\ root\ ts))$

Similar:

$get\_min\_rest :: 'a\ tree\ list \Rightarrow 'a\ tree \times 'a\ tree\ list$

Returns tree with minimal root, and remaining trees

$del\_min\ ts =$
$(\text{case}\ get\_min\_rest\ ts\ \text{of}$
$\quad (Node\ r\ x\ ts_1,\ ts_2) \Rightarrow merge\ (rev\ ts_1)\ ts_2)$

Why rev? Rank decreasing in ts_1 but increasing in ts_2

Complexity

Recall: $|t| = 2^{rank\ t}$

Similarly for heap: $2^{length\ ts} \leq |ts| + 1$

Complexity

Recall: $|t| = 2^{rank\ t}$

Similarly for heap: $2^{length\ ts} \leq |ts| + 1$

Complexity of operations: linear in length of heap

Complexity

Recall: $|t| = 2^{rank\ t}$

Similarly for heap: $2^{length\ ts} \leq |ts| + 1$

Complexity of operations: linear in length of heap

i.e., logarithmic in number of elements

Complexity of $merge$

$merge\ (t_1 \# ts_1)\ (t_2 \# ts_2) =$
(if $rank\ t_1 < rank\ t_2$ then $t_1 \# merge\ ts_1\ (t_2 \# ts_2)$
else if $rank\ t_2 < rank\ t_1$ then $t_2 \# merge\ (t_1 \# ts_1)\ ts_2$
else $ins\_tree\ (link\ t_1\ t_2)\ (merge\ ts_1\ ts_2))$

Complexity of $merge$

$merge\ (t_1 \# ts_1)\ (t_2 \# ts_2) =$
(if $rank\ t_1 < rank\ t_2$ then $t_1 \# merge\ ts_1\ (t_2 \# ts_2)$
else if $rank\ t_2 < rank\ t_1$ then $t_2 \# merge\ (t_1 \# ts_1)\ ts_2$
else $ins\_tree\ (link\ t_1\ t_2)\ (merge\ ts_1\ ts_2))$

Complexity of ins_tree: $t\_ins\_tree\ t\ ts \leq length\ ts + 1$

A call $merge\ t_1\ t_2$ (where $length\ t_1 = length\ t_2 = n$) can lead to calls of ins_tree on lists of length 1, ..., n.

$\sum \in O(n^2)$

Complexity of $merge$

$merge\ (t_1 \# ts_1)\ (t_2 \# ts_2) =$
(if $rank\ t_1 < rank\ t_2$ then $t_1 \# merge\ ts_1\ (t_2 \# ts_2)$
else if $rank\ t_2 < rank\ t_1$ then $t_2 \# merge\ (t_1 \# ts_1)\ ts_2$
else $ins\_tree\ (link\ t_1\ t_2)\ (merge\ ts_1\ ts_2))$

Relate time and length of input/output:

$t\_ins\_tree\ t\ ts + length\ (ins\_tree\ t\ ts) = 2 + length\ ts$

$length\ (merge\ ts_1\ ts_2) + t\_merge\ ts_1\ ts_2$
$\leq 2 * (length\ ts_1 + length\ ts_2) + 1$

Yields desired linear bound!

Sources

The inventor of the binomial heap:

Jean Vuillemin.
A Data Structure for Manipulating Priority Queues.
CACM, 1978.

Complexity of $merge$

$merge\ (t_1\ \#\ ts_1)\ (t_2\ \#\ ts_2) =$
$(\text{if } rank\ t_1 < rank\ t_2 \text{ then } t_1\ \#\ merge\ ts_1\ (t_2\ \#\ ts_2)$
$\ \text{else if } rank\ t_2 < rank\ t_1 \text{ then } t_2\ \#\ merge\ (t_1\ \#\ ts_1)\ ts_2$
$\quad\quad \text{else } ins\_tree\ (link\ t_1\ t_2)\ (merge\ ts_1\ ts_2))$

Relate time and length of input/output:

$t\_ins\_tree\ t\ ts + length\ (ins\_tree\ t\ ts) = 2 + length\ ts$

$length\ (merge\ ts_1\ ts_2) + t\_merge\ ts_1\ ts_2$
$\leq 2 * (length\ ts_1 + length\ ts_2) + 1$

Yields desired linear bound!

Skew Binomial Heap

Similar to binomial heap, but involving also *skew binary numbers*:

Skew Binomial Heap

Similar to binomial heap, but involving also *skew binary numbers*:

$d_1 \ldots d_n$ represents $\sum_{i=1}^{n} d_i * (2^{i+1} - 1)$
where $d_i \in \{0, 1, 2\}$

Complexity

Skew binomial heap:

insert in time $O(1)$
del_min and *merge* still $O(\log n)$

Complexity

Skew binomial heap:

insert in time $O(1)$
del_min and *merge* still $O(\log n)$

Fibonacci heap (imperative!):

insert and *merge* in time $O(1)$
del_min still $O(\log n)$

Complexity

Skew binomial heap:

insert in time $O(1)$
del_min and *merge* still $O(\log n)$

Fibonacci heap (imperative!):

insert and *merge* in time $O(1)$
del_min still $O(\log n)$

Every operation in time $O(1)$?

Puzzle

Design a functional queue
with (worst case) constant time *enq* and *deq* functions

Puzzle

Design a functional queue
with (worst case) constant time *enq* and *deq* functions

Complexity

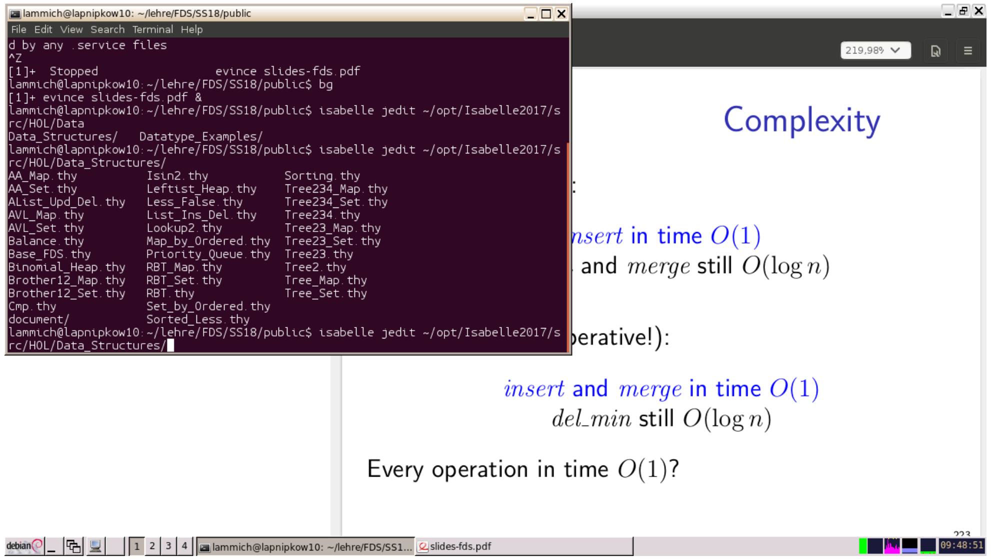
Skew binomial heap:

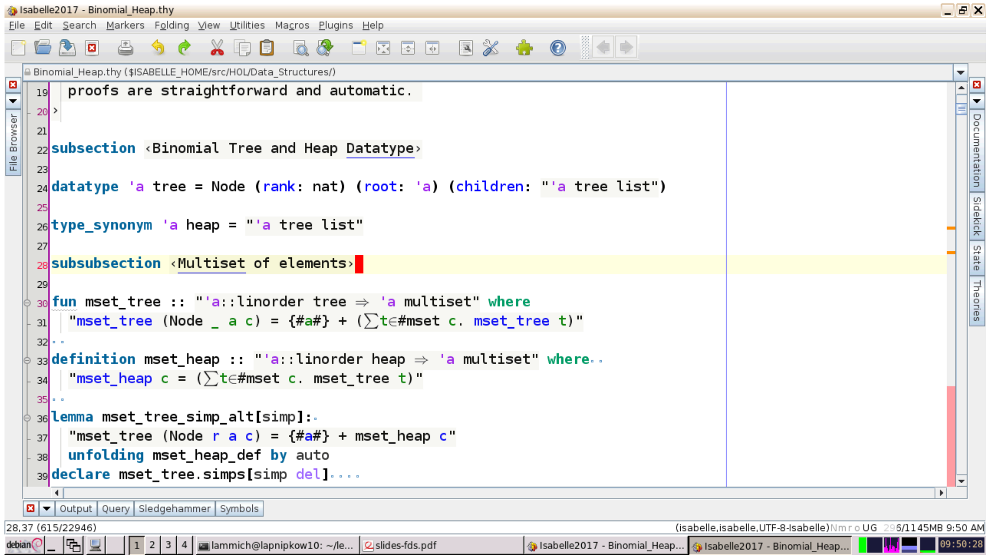
insert in time $O(1)$
del_min and *merge* still $O(\log n)$

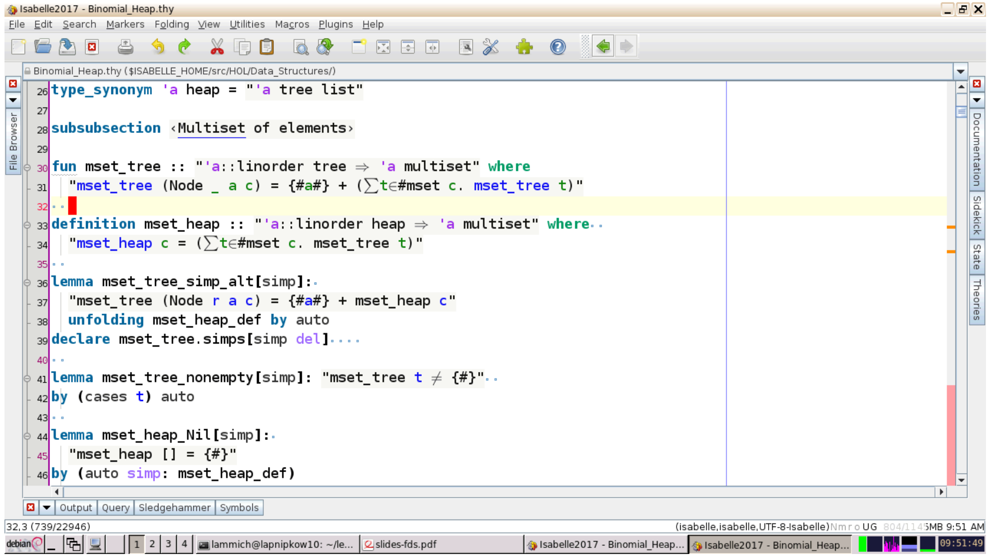
Fibonacci heap (imperative!):

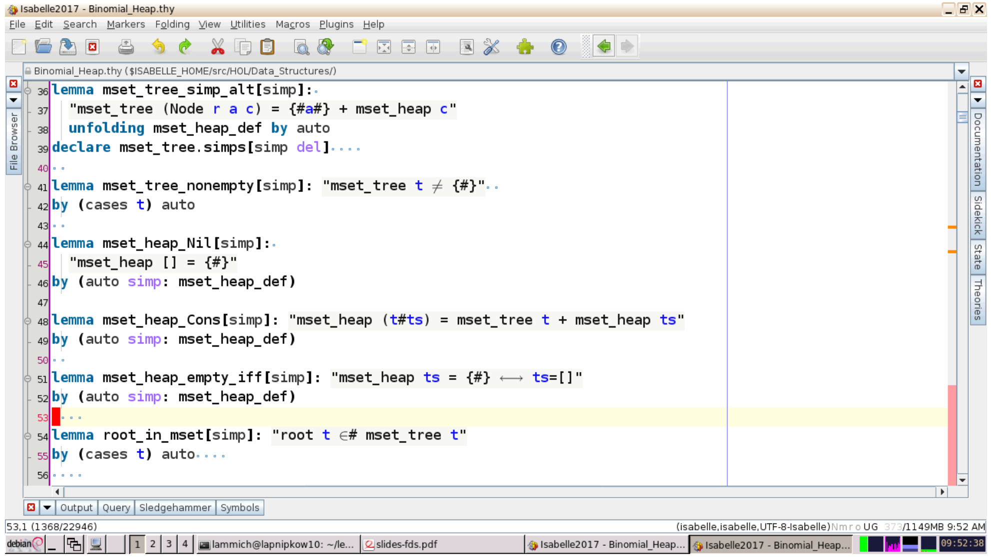
insert and *merge* in time $O(1)$
del_min still $O(\log n)$

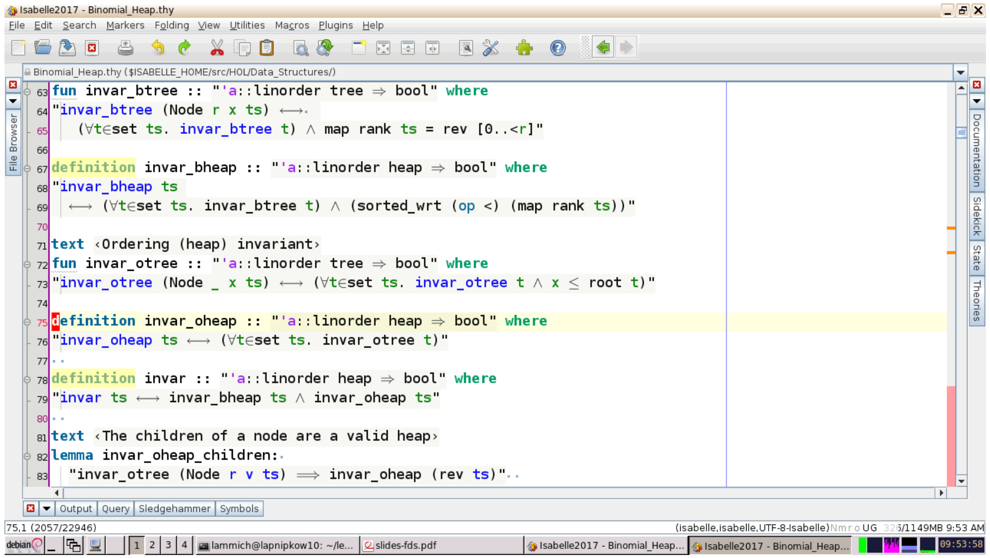
Every operation in time $O(1)$?

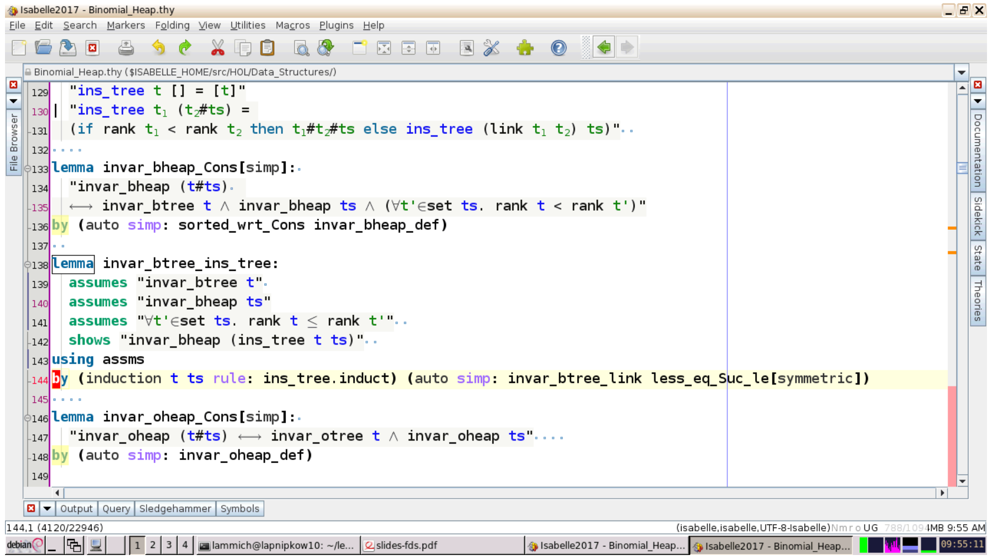


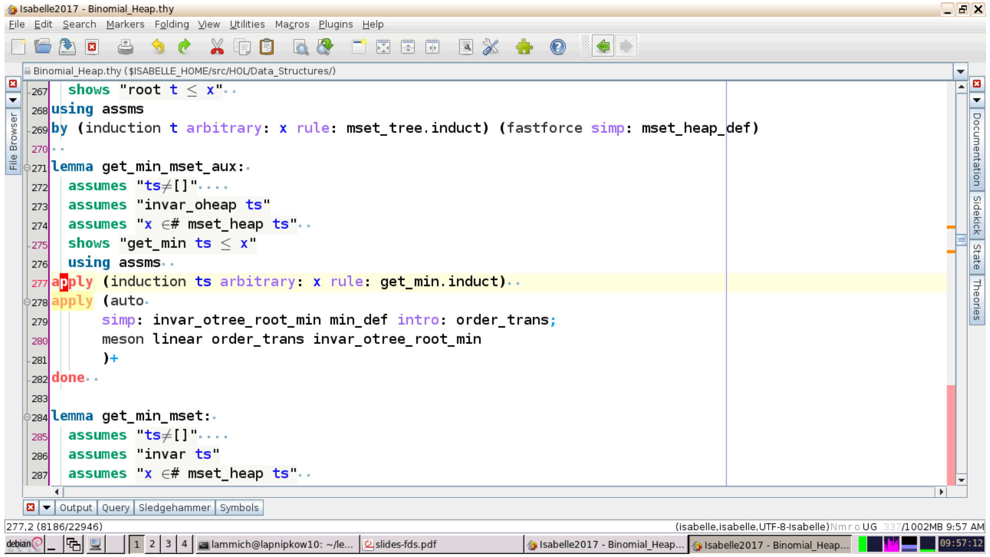


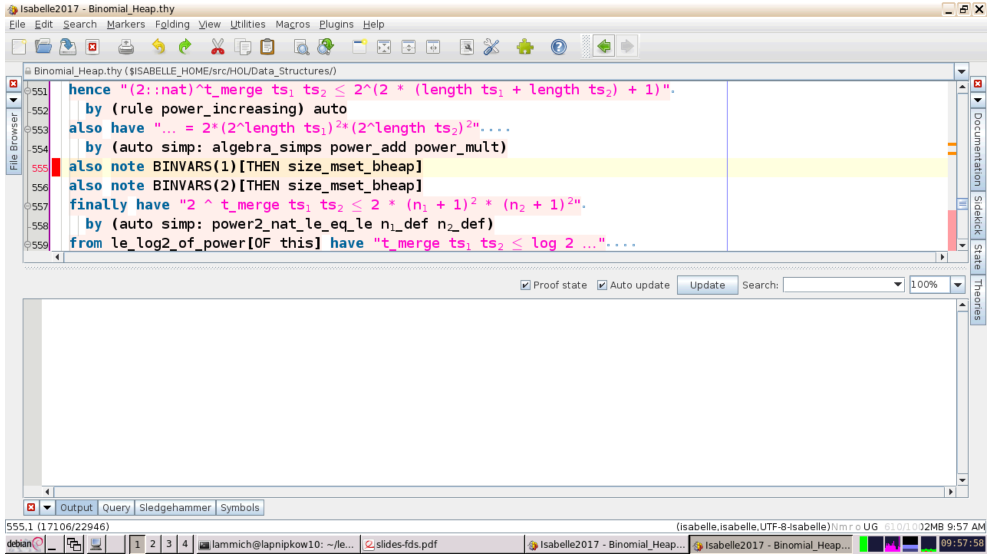

Isabelle2017 - Binomial_Heap.thy
Binomial_Heap.thy ($ISABELLE_HOME/src/HOL/Data_Structures/)
type_synonym 'a heap = "'a tree list"
subsection ‹Multiset of elements›
fun mset_tree :: "'a::linorder tree ⇒ 'a multiset" where
"mset_tree (Node _ a c) = {#a#} + (∑t∈#mset c. mset_tree t)"
definition mset_heap :: "'a::linorder heap ⇒ 'a multiset" where
"mset_heap c = (∑t∈#mset c. mset_tree t)"
lemma mset_tree_simp_alt[simp]:
"mset_tree (Node r a c) = {#a#} + mset_heap c"
unfolding mset_heap_def by auto
declare mset_tree.simps[simp del]
lemma mset_tree_nonempty[simp]: "mset_tree t ≠ {#}"
by (cases t) auto
lemma mset_heap_Nil[simp]:
"mset_heap [] = {#}"
by (auto simp: mset_heap_def)
Output Query Sledgehammer Symbols
32,3 (739/22946)
09:51:49

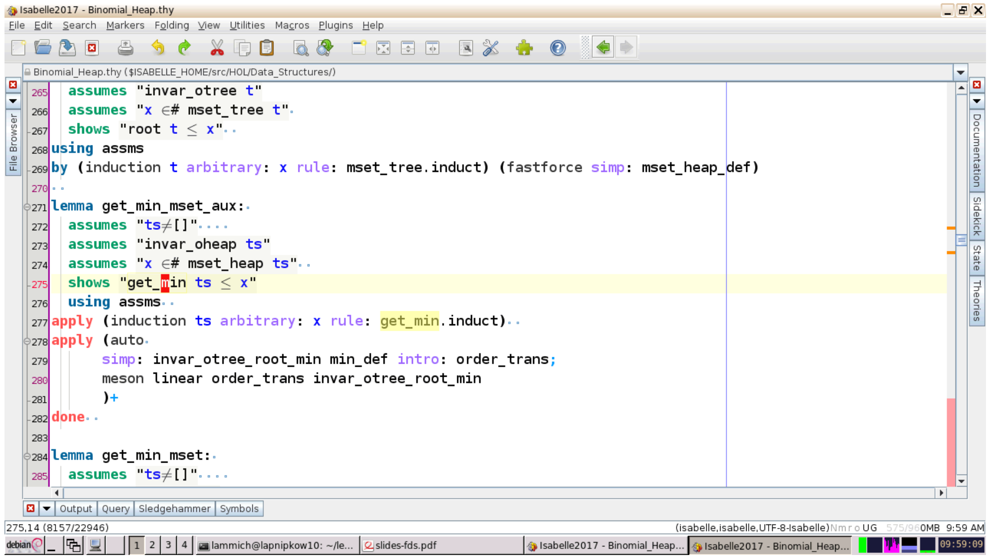

Isabelle2017 - Binomial_Heap.thy
Binomial_Heap.thy ($ISABELLE_HOME/src/HOL/Data_Structures/)
lemma mset_tree_simp_alt[simp]:
"mset_tree (Node r a c) = {#a#} + mset_heap c"
unfolding mset_heap_def by auto
declare mset_tree.simps[simp del]
lemma mset_tree_nonempty[simp]: "mset_tree t ≠ {#}"
by (cases t) auto
lemma mset_heap_Nil[simp]:
"mset_heap [] = {#}"
by (auto simp: mset_heap_def)
lemma mset_heap_Cons[simp]: "mset_heap (t#ts) = mset_tree t + mset_heap ts"
by (auto simp: mset_heap_def)
lemma mset_heap_empty_iff[simp]: "mset_heap ts = {#} ⟷ ts=[]"
by (auto simp: mset_heap_def)
lemma root_in_mset[simp]: "root t ∈# mset_tree t"
by (cases t) auto
Output Query Sledgehammer Symbols
53,1 (1368/22946)
09:52:38

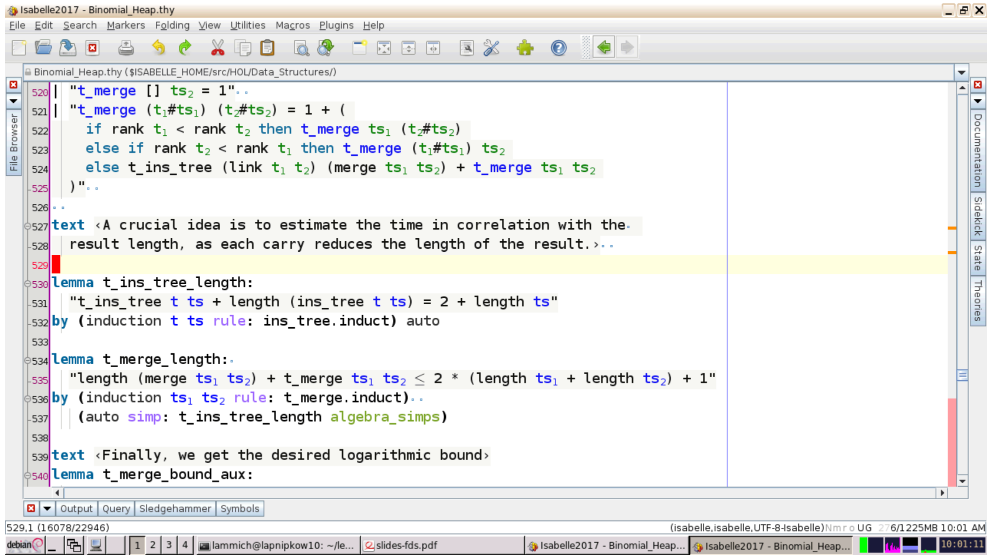

Isabelle2017 - Binomial_Heap.thy
Binomial_Heap.thy ($ISABELLE_HOME/src/HOL/Data_Structures/)
fun invar_btree :: "'a::linorder tree ⇒ bool" where
"invar_btree (Node r x ts) ⟷
(∀t∈set ts. invar_btree t) ∧ map rank ts = rev [0..<r]"
definition invar_bheap :: "'a::linorder heap ⇒ bool" where
"invar_bheap ts
⟷ (∀t∈set ts. invar_btree t) ∧ (sorted_wrt (op <) (map rank ts))"
text ‹Ordering (heap) invariant›
fun invar_otree :: "'a::linorder tree ⇒ bool" where
"invar_otree (Node _ x ts) ⟷ (∀t∈set ts. invar_otree t ∧ x ≤ root t)"
definition invar_oheap :: "'a::linorder heap ⇒ bool" where
"invar_oheap ts ⟷ (∀t∈set ts. invar_otree t)"
definition invar :: "'a::linorder heap ⇒ bool" where
"invar ts ⟷ invar_bheap ts ∧ invar_oheap ts"
text ‹The children of a node are a valid heap›
lemma invar_oheap_children:
"invar_otree (Node r v ts) ⟹ invar_oheap (rev ts)"
Output Query Sledgehammer Symbols
75,1 (2057/22946)
09:53:58

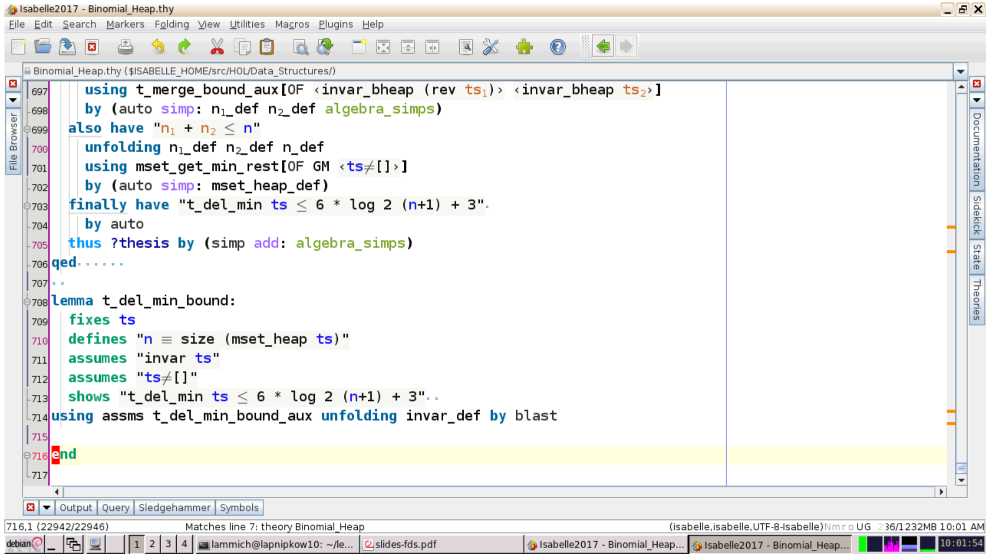

Isabelle2017 - Binomial_Heap.thy
Binomial_Heap.thy ($ISABELLE_HOME/src/HOL/Data_Structures/)
"ins_tree t [] = [t]"
| "ins_tree t₁ (t₂#ts) =
(if rank t₁ < rank t₂ then t₁#t₂#ts else ins_tree (link t₁ t₂) ts)"
lemma invar_bheap_Cons[simp]:
"invar_bheap (t#ts)
⟷ invar_btree t ∧ invar_bheap ts ∧ (∀t'∈set ts. rank t < rank t')"
by (auto simp: sorted_wrt_Cons invar_bheap_def)
lemma invar_btree_ins_tree:
assumes "invar_btree t"
assumes "invar_bheap ts"
assumes "∀t'∈set ts. rank t ≤ rank t'"
shows "invar_bheap (ins_tree t ts)"
using assms
by (induction t ts rule: ins_tree.induct) (auto simp: invar_btree_link less_eq_Suc_le[symmetric])
lemma invar_oheap_Cons[simp]:
"invar_oheap (t#ts) ⟷ invar_otree t ∧ invar_oheap ts"
by (auto simp: invar_oheap_def)
Output Query Sledgehammer Symbols
144,1 (4120/22946)
09:55:11


Isabelle2017 - Binomial_Heap.thy
shows "root t ≤ x"
using assms
by (induction t arbitrary: x rule: mset_tree.induct) (fastforce simp: mset_heap_def)
lemma get_min_mset_aux:
assumes "ts≠[]"
assumes "invar_oheap ts"
assumes "x ∈# mset_heap ts"
shows "get_min ts ≤ x"
using assms
apply (induction ts arbitrary: x rule: get_min.induct)
apply (auto
simp: invar_otree_root_min min_def intro: order_trans;
meson linear order_trans invar_otree_root_min
)+
done
lemma get_min_mset:
assumes "ts≠[]"
assumes "invar ts"
assumes "x ∈# mset_heap ts"


Isabelle2017 - Binomial_Heap.thy
hence "(2::nat)^t_merge ts₁ ts₂ ≤ 2^(2 * (length ts₁ + length ts₂) + 1)"
by (rule power_increasing) auto
also have "... = 2*(2^length ts₁)²*(2^length ts₂)²"
by (auto simp: algebra_simps power_add power_mult)
also note BINVARS(1)[THEN size_mset_bheap]
also note BINVARS(2)[THEN size_mset_bheap]
finally have "2 ^ t_merge ts₁ ts₂ ≤ 2 * (n₁ + 1)² * (n₂ + 1)²"
by (auto simp: power2_nat_le_eq_le n₁_def n₂_def)
from le_log2_of_power[OF this] have "t_merge ts₁ ts₂ ≤ log 2 ..."
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols


Isabelle2017 - Binomial_Heap.thy
assumes "invar_otree t"
assumes "x ∈# mset_tree t"
shows "root t ≤ x"
using assms
by (induction t arbitrary: x rule: mset_tree.induct) (fastforce simp: mset_heap_def)
lemma get_min_mset_aux:
assumes "ts≠[]"
assumes "invar_oheap ts"
assumes "x ∈# mset_heap ts"
shows "get_min ts ≤ x"
using assms
apply (induction ts arbitrary: x rule: get_min.induct)
apply (auto
simp: invar_otree_root_min min_def intro: order_trans;
meson linear order_trans invar_otree_root_min
)+
done
lemma get_min_mset:
assumes "ts≠[]"


Isabelle2017 - Binomial_Heap.thy
| "t_merge [] ts₂ = 1"
| "t_merge (t₁#ts₁) (t₂#ts₂) = 1 + (
if rank t₁ < rank t₂ then t_merge ts₁ (t₂#ts₂)
else if rank t₂ < rank t₁ then t_merge (t₁#ts₁) ts₂
else t_ins_tree (link t₁ t₂) (merge ts₁ ts₂) + t_merge ts₁ ts₂
)"
text ‹A crucial idea is to estimate the time in correlation with the
result length, as each carry reduces the length of the result.›
lemma t_ins_tree_length:
"t_ins_tree t ts + length (ins_tree t ts) = 2 + length ts"
by (induction t ts rule: ins_tree.induct) auto
lemma t_merge_length:
"length (merge ts₁ ts₂) + t_merge ts₁ ts₂ ≤ 2 * (length ts₁ + length ts₂) + 1"
by (induction ts₁ ts₂ rule: t_merge.induct)
(auto simp: t_ins_tree_length algebra_simps)
text ‹Finally, we get the desired logarithmic bound›
lemma t_merge_bound_aux:


Isabelle2017 - Binomial_Heap.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Binomial_Heap.thy ($ISABELLE_HOME/src/HOL/Data_Structures/)
using t_merge_bound_aux[OF ‹invar_bheap (rev ts₁)› ‹invar_bheap ts₂›]
by (auto simp: n₁_def n₂_def algebra_simps)
also have "n₁ + n₂ ≤ n"
unfolding n₁_def n₂_def n_def
using mset_get_min_rest[OF GM ‹ts≠[]›]
by (auto simp: mset_heap_def)
finally have "t_del_min ts ≤ 6 * log 2 (n+1) + 3"
by auto
thus ?thesis by (simp add: algebra_simps)
qed
lemma t_del_min_bound:
fixes ts
defines "n ≡ size (mset_heap ts)"
assumes "invar ts"
assumes "ts≠[]"
shows "t_del_min ts ≤ 6 * log 2 (n+1) + 3"
using assms t_del_min_bound_aux unfolding invar_def by blast
end
Output Query Sledgehammer Symbols
716,1 (22942/22946)
Matches line 7: theory Binomial_Heap
(isabelle,isabelle,UTF-8-Isabelle)
10:01 AM
Documentation Sidekick State Theories
File Browser