Functional Data Structures

with Isabelle/HOL

Tobias Nipkow

Fakultät für Informatik
Technische Universität München

2017-2-3

Part II

Functional Data Structures

Chapter 1

Binary Trees

Library/Tree.thy

Binary trees

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

Abbreviations:

$$\begin{aligned} \langle\rangle &\equiv Leaf \\ \langle l,\ a,\ r\rangle &\equiv Node\ l\ a\ r \end{aligned}$$

In the sequel: tree = binary tree

Tree traversal

$inorder :: {}'a\ tree \Rightarrow {}'a\ list$

$inorder\ \langle\rangle = []$
$inorder\ \langle l,\ x,\ r\rangle = inorder\ l\ @\ [x]\ @\ inorder\ r$

$preorder :: {}'a\ tree \Rightarrow {}'a\ list$

$preorder\ \langle\rangle = []$
$preorder\ \langle l,\ x,\ r\rangle = x\ \#\ preorder\ l\ @\ preorder\ r$

$postorder :: {}'a\ tree \Rightarrow {}'a\ list$

$postorder\ \langle\rangle = []$
$postorder\ \langle l,\ x,\ r\rangle = postorder\ l\ @\ postorder\ r\ @\ [x]$

Size

$size :: {}'a\ tree \Rightarrow nat$

$|\langle\rangle| = 0$
$|\langle l,\ \_,\ r\rangle| = |l| + |r| + 1$

$size1 :: {}'a\ tree \Rightarrow nat$
$|t|_1 = |t| + 1$

$\Longrightarrow$
$|\langle\rangle|_1 = 1$
$|\langle l,\ x,\ r\rangle|_1 = |l|_1 + |r|_1$

Lemma The number of leaves in t is $|t|_1$.

Warning: $|.|$ and $|.|_1$ only on slides

Height

$height :: {}'a\ tree \Rightarrow nat$

$h(\langle\rangle) = 0$
$h(\langle l,\ \_,\ r\rangle) = max\ (h(l))\ (h(r)) + 1$

Warning: $h(.)$ only on slides

Lemma $h(t) \leq |t|$

Lemma $|t|_1 \leq 2^{h(t)}$

3 Interlude: Arithmetic in Isabelle

- Numeric Types
- Chains of (In)Equations
- Proof Automation

Numeric types: nat, int, $real$

Need conversion functions (inclusions):

$$\begin{array}{rcl} int & :: & nat \Rightarrow int \\ real & :: & nat \Rightarrow real \\ real\_of\_int & :: & int \Rightarrow real \end{array}$$

If you need type $real$,
import theory $Complex\_Main$ instead of $Main$

Numeric types: nat, int, $real$

Isabelle inserts conversion functions automatically
(with theory $Complex\_Main$)
If there are multiple correct completions,
Isabelle chooses an arbitrary one

Examples

$(i{::}int) + (n{::}nat) \quad \rightsquigarrow \quad i + int\ n$

$((n{::}nat) + n) :: real \quad \rightsquigarrow \quad real(n{+}n),\ real\ n + real\ n$

Numeric types: nat, int, $real$

Coercion in the other direction:

$$\begin{array}{rcl} nat & :: & int \Rightarrow nat \\ floor & :: & real \Rightarrow int \\ ceiling & :: & real \Rightarrow int \end{array}$$

Overloaded arithmetic operations

- Numbers are overloaded: $0,\ 1,\ 2,\ \ldots :: {}'a$
- Basic arithmetic functions are overloaded:
  $op\ +,\ op\ -,\ op\ * :: {}'a \Rightarrow {}'a \Rightarrow {}'a$
  $- :: {}'a \Rightarrow {}'a$
- Division on nat and int:
  $op\ div,\ op\ mod :: {}'a \Rightarrow {}'a \Rightarrow {}'a$
- Division on $real$: $op\ / :: {}'a \Rightarrow {}'a \Rightarrow {}'a$
- Exponentiation with nat: $op\ \hat{} :: {}'a \Rightarrow nat \Rightarrow {}'a$
- Exponentiation with $real$: $op\ powr :: {}'a \Rightarrow {}'a \Rightarrow {}'a$
- Absolute value: $abs :: {}'a \Rightarrow {}'a$

3 Interlude: Arithmetic in Isabelle

Numeric Types

Chains of (In)Equations

Proof Automation

Chains of equations

Textbook proof

$$
\begin{aligned}
t_1 &= t_2 \quad \langle\text{justification}\rangle \\
&= t_3 \quad \langle\text{justification}\rangle \\
&\vdots \\
&= t_n \quad \langle\text{justification}\rangle
\end{aligned}
$$

In Isabelle:

have "$t_1 = t_2$" ⟨proof⟩
also have "$\ldots = t_3$" ⟨proof⟩
⋮
also have "$\ldots = t_n$" ⟨proof⟩
finally have "$t_1 = t_n$" .

"…" is literally three dots

Chains of equations and inequations

Instead of $=$ you may also use $\leq$ and $<$.

Example

have "$t_1 < t_2$" ⟨proof⟩
also have "... $= t_3$" ⟨proof⟩
  ⋮
also have "... $\leq t_n$" ⟨proof⟩
finally have "$t_1 < t_n$" .

How to interpret "..."

have "$t_1 \leq t_2$" ⟨proof⟩
also have "$\ldots = t_3$" ⟨proof⟩

Here "..." is internally replaced by t_2

In general, if $this$ is the formula $p\ t_1\ t_2$ where p is some constant, then "..." stands for t_2.

3 Interlude: Arithmetic in Isabelle

Linear formulas

Only:

variables

numbers

number $*$ variable

$+, -$

$=, \leq, <$

$\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$

Examples

Linear: $3 * x + 5 * y \leq z \longrightarrow x < z$

Nonlinear: $x \leq x * x$

Extended linear formulas

Also allowed:

min, max

$even$, odd

$t\ div\ n$, $t\ mod\ n$ where n is a number

conversion functions

nat, $floor$, $ceiling$, abs

Automatic proof of arithmetic formulas

by $arith$

Proof method $arith$ tries to solve arithmetic formulas.

- Succeeds or fails
- Decision procedure for extended linear formulas; for types nat and int, the extended linear formulas may also contain $\forall$ and $\exists$
- Nonlinear subformulas are viewed as (new) variables; for example, $x \leq x * x$ is viewed as $x \leq y$

Automatic proof of arithmetic formulas

by (*simp add: algebra_simps*)

- The lemmas list *algebra_simps* helps to simplify arithmetic formulas
- It applies associativity, commutativity and distributivity of $+$ and $*$.
- This may prove the formula, may make it simpler, or may make it unreadable.
- It is a decision procedure for equations over rings (e.g. *int*)

Automatic proof of arithmetic formulas

by ($simp\ add\!: field\_simps$)

- The lemmas list $field\_simps$ extends $algebra\_simps$ by rules for $/$
- Can only cancel common terms in a quotient, e.g. $x * y \;/\; (x * z)$, if $x \neq 0$ can be proved.

End of interlude, back to trees ...

Tree.thy

$$|t|_1 \leq 2^{h(t)}$$

Minimal height

$min\_height :: {}'a\ tree \Rightarrow nat$

$mh(\langle\rangle) = 0$
$mh(\langle l,\ \_,\ r\rangle) = min\ (mh(l))\ (mh(r)) + 1$

Warning: $mh(.)$ only on slides

Lemma $mh(t) \leq h(t)$

Lemma $2^{mh(t)} \leq |t|_1$

Internal path length

$ipl :: {'a}\ tree \Rightarrow nat$

$ipl\ \langle\rangle = 0$
$ipl\ \langle l,\ \_,\ r\rangle = ipl\ l + |l| + ipl\ r + |r|$

Why relevant?

Upper bound?

Complete tree

$complete :: 'a\ tree \Rightarrow bool$

$complete\ \langle\rangle = True$

$complete\ \langle l,\ \_,\ r\rangle =$

$(complete\ l \wedge complete\ r \wedge h(l) = h(r))$

Lemma $complete\ t = (mh(t) = h(t))$

Lemma $complete\ t \Longrightarrow |t|_1 = 2^{h(t)}$

Lemma $|t|_1 = 2^{h(t)} \Longrightarrow complete\ t$

Lemma $|t|_1 = 2^{mh(t)} \Longrightarrow complete\ t$

Corollary $\neg\ complete\ t \Longrightarrow |t|_1 < 2^{h(t)}$

Corollary $\neg\ complete\ t \Longrightarrow 2^{mh(t)} < |t|_1$

Complete tree: ipl

Lemma A complete tree of height h has internal path length $(h-2)*2^h+2$.

In a search tree, finding the node labelled x takes as many steps as the path from the root to x is long. Thus the average time to find an element that is in the tree is $ipl\ t\ /\ |t|$.

Lemma Let t be a complete search tree of height h. The average time to find a random element that is in the tree is asymptotically $h-2$ (as h approaches ∞):

$$ipl\ t\ /\ |t| \sim h-2$$

Complete tree: ipl

A problem: $(h - 2) * 2^h + 2$ is only correct if interpreted over type int, not nat.

Correct version:
Lemma $complete\ t \Longrightarrow$
$int\ (ipl\ t) = (int\ (h(t)) - 2) * 2^{h(t)} + 2$

We do not cover the Isabelle formalization of limits.

Balanced tree

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

$balanced\ t = (h(t) - mh(t) \leq 1)$

Balanced trees have optimal height:
Lemma *If* $balanced\ t \wedge |t| \leq |t'|$ *then* $h(t) \leq h(t')$.

Warning

- The terms *complete* and *balanced*
  are not defined uniquely in the literature.

- For example,
  Knuth calls *complete* what we call *balanced*.

Chapter 2

Search Trees

6. Unbalanced BST

7. AVL Trees

8. Red-Black Trees

Most of the material focuses on
BSTs = binary search trees

BSTs represent sets

Any tree represents a set:

$set\_tree :: {}'a\ tree \Rightarrow {}'a\ set$

$set\_tree\ \langle\rangle = \{\}$
$set\_tree\ \langle l,\ x,\ r\rangle = set\_tree\ l \cup \{x\} \cup set\_tree\ r$

A BST represents a set that can be searched in time $O(h(t))$

Function set_tree is called an *abstraction function*
because it maps the implementation
to the abstract mathematical object

bst

$bst :: 'a\ tree \Rightarrow bool$

$bst\ \langle\rangle = True$
$bst\ \langle l,\ a,\ r\rangle =$
$(bst\ l \wedge bst\ r \wedge$
$(\forall x \in set\_tree\ l.\ x < a) \wedge$
$(\forall x \in set\_tree\ r.\ a < x))$

Type $'a$ must be in class $linorder$ ($'a :: linorder$) where $linorder$ are *linear orders* (also called *total orders*).

Note: nat, int and $real$ are in class $linorder$

Interface

An implementation of sets of elements of type $'a$ must provide

- An implementation type $'s$
- $empty :: {'s}$
- $insert :: {'a} \Rightarrow {'s} \Rightarrow {'s}$
- $delete :: {'a} \Rightarrow {'s} \Rightarrow {'s}$
- $isin :: {'s} \Rightarrow {'a} \Rightarrow bool$

Alternative interface

Instead of a set, a search tree can also implement a map from $'a$ to $'b$:

- An implementation type $'m$
- $empty :: 'm$
- $update :: 'a \Rightarrow 'b \Rightarrow 'm \Rightarrow 'm$
- $delete :: 'a \Rightarrow 'm \Rightarrow 'm$
- $lookup :: 'm \Rightarrow 'a \Rightarrow 'b\ option$

Sets are a special case of maps

Comparison of elements

We assume that the element type $'a$ is a linear order

Instead of using $<$ and $\leq$ directly:

datatype $cmp\_val = LT \mid EQ \mid GT$

$cmp\ x\ y =$
(if $x < y$ then LT else if $x = y$ then EQ else GT)

6 Unbalanced BST

7 AVL Trees

8 Red-Black Trees

Implementation

Implementation type: $'a\ tree$

$empty = Leaf$

$$
\begin{aligned}
&insert\ x\ \langle\rangle = \langle\langle\rangle,\ x,\ \langle\rangle\rangle \\
&insert\ x\ \langle l,\ a,\ r\rangle = (\textsf{case}\ cmp\ x\ a\ \textsf{of} \\
&\qquad\qquad LT \Rightarrow \langle insert\ x\ l,\ a,\ r\rangle \\
&\qquad\quad\ \mid EQ \Rightarrow \langle l,\ a,\ r\rangle \\
&\qquad\quad\ \mid GT \Rightarrow \langle l,\ a,\ insert\ x\ r\rangle)
\end{aligned}
$$

Implementation

$$
\begin{aligned}
&isin\ \langle\rangle\ x = False \\
&isin\ \langle l,\ a,\ r\rangle\ x = (\textsf{case}\ cmp\ x\ a\ \textsf{of} \\
&\qquad\qquad\qquad\quad\ \ LT \Rightarrow isin\ l\ x \\
&\qquad\qquad\qquad\quad \mid EQ \Rightarrow True \\
&\qquad\qquad\qquad\quad \mid GT \Rightarrow isin\ r\ x)
\end{aligned}
$$

Implementation

$$
\begin{array}{l}
\mathit{delete}\ x\ \langle\rangle = \langle\rangle \\
\mathit{delete}\ x\ \langle l,\ a,\ r\rangle = \\
(\textsf{case}\ \mathit{cmp}\ x\ a\ \textsf{of} \\
\quad LT \Rightarrow \langle \mathit{delete}\ x\ l,\ a,\ r\rangle \\
\ \mid EQ \Rightarrow \textsf{if}\ r = \langle\rangle\ \textsf{then}\ l \\
\qquad\qquad\ \textsf{else let}\ (a',\ r') = \mathit{del\_min}\ r\ \textsf{in}\ \langle l,\ a',\ r'\rangle \\
\ \mid GT \Rightarrow \langle l,\ a,\ \mathit{delete}\ x\ r\rangle)
\end{array}
$$

$$
\begin{array}{l}
\mathit{del\_min}\ \langle l,\ a,\ r\rangle = \\
(\textsf{if}\ l = \langle\rangle\ \textsf{then}\ (a,\ r) \\
\ \textsf{else let}\ (x,\ l') = \mathit{del\_min}\ l\ \textsf{in}\ (x,\ \langle l',\ a,\ r\rangle))
\end{array}
$$

6 Unbalanced BST

Correctness

Correctness Proof Method Based on Sorted Lists

Why is this implementation correct?

Because	$empty$	$insert$	$delete$	$isin$
simulate	$\{\}$	$\cup \{.\}$	$- \{.\}$	$\in$:

$$set\_tree\ empty = \{\}$$
$$set\_tree\ (insert\ x\ t) = set\_tree\ t \cup \{x\}$$
$$set\_tree\ (delete\ x\ t) = set\_tree\ t - \{x\}$$
$$isin\ t\ x = (x \in set\_tree\ t)$$

Under the assumption $bst\ t$

Also: bst must be invariant

$$bst\ empty$$
$$bst\ t \Longrightarrow bst\ (insert\ x\ t)$$
$$bst\ t \Longrightarrow bst\ (delete\ x\ t)$$

6 Unbalanced BST

Correctness

Correctness Proof Method Based on Sorted Lists

$sorted :: 'a\ list \Rightarrow bool$

$sorted\ [] = True$
$sorted\ [x] = True$
$sorted\ (x \# y \# zs) = (x < y \wedge sorted\ (y \# zs))$

No duplicates!

Structural invariant

The proof method works not just for unbalanced trees.
We assume that there is some structural invariant on the search tree:

$inv : 's \Rightarrow bool$

e.g. some balance criterion.

Correctness of $insert$

$$inv\ t \wedge sorted\ (inorder\ t) \Longrightarrow$$
$$inorder\ (insert\ x\ t) = ins\_list\ x\ (inorder\ t)$$

where

$$ins\_list :: 'a \Rightarrow 'a\ list \Rightarrow 'a\ list$$

inserts an element into a sorted list.

Also covers preservation of bst

Correctness of $delete$

$$inv\ t \wedge sorted\ (inorder\ t) \Longrightarrow$$
$$inorder\ (delete\ x\ t) = del\_list\ x\ (inorder\ t)$$

where

$$del\_list :: 'a \Rightarrow 'a\ list \Rightarrow 'a\ list$$

deletes an element from a sorted list.

Also covers preservation of bst

Correctness of $isin$

$$inv\ t \wedge sorted\ (inorder\ t) \Longrightarrow$$
$$isin\ t\ x = (x \in elems\ (inorder\ t))$$

where

$$elems :: {'a}\ list \Rightarrow {'a}\ set$$

converts a list into a set.

Data_Structures/AVL_Set.thy

Data_Structures/RBT_Set.thy

Relationship to 2-3-4 trees

Red-black trees

datatype $color = Red \mid Black$

datatype
$'a\ rbt = Leaf \mid Node\ color\ ('a\ tree)\ 'a\ ('a\ tree)$

Abbreviations:

$$\begin{array}{rcl} \langle\rangle & \equiv & Leaf \\ \langle c,\ l,\ a,\ r\rangle & \equiv & Node\ c\ l\ a\ r \\ R\ l\ a\ r & \equiv & Node\ Red\ l\ a\ r \\ B\ l\ a\ r & \equiv & Node\ Black\ l\ a\ r \end{array}$$

Color

$$color :: {}'a\ rbt \Rightarrow color$$

$$color\ \langle\rangle = Black$$
$$color\ \langle c,\ \_,\ \_,\ \_\rangle = c$$

$$paint :: color \Rightarrow {}'a\ rbt \Rightarrow {}'a\ rbt$$

$$paint\ c\ \langle\rangle = \langle\rangle$$
$$paint\ c\ \langle\_,\ l,\ a,\ r\rangle = \langle c,\ l,\ a,\ r\rangle$$

Invariants

$rbt :: {}'a\ rbt \Rightarrow bool$

$rbt\ t = (invc\ t \wedge invh\ t \wedge color\ t = Black)$

$invc :: {}'a\ rbt \Rightarrow bool$

$invc\ \langle\rangle = True$

$invc\ \langle c,\ l,\ \_,\ r\rangle =$

$(invc\ l \wedge invc\ r \wedge$

$\ (c = Red \longrightarrow color\ l = Black \wedge color\ r = Black))$

Invariants

$$invh :: 'a\ rbt \Rightarrow bool$$

$$invh\ \langle\rangle = True$$
$$invh\ \langle\_,\ l,\ \_,\ r\rangle = (invh\ l \wedge invh\ r \wedge bh(l) = bh(r))$$

$$bheight :: 'a\ rbt \Rightarrow nat$$

$$bh(\langle\rangle) = 0$$
$$bh(\langle c,\ l,\ \_,\ \_\rangle) =$$
$$(\text{if } c = Black \text{ then } bh(l) + 1 \text{ else } bh(l))$$

Exercise

Is $invh$ what we want?

Define a function $Bpl :: 'a\ rbt \Rightarrow nat\ set$
such that $Bpl\ t$ ("black path lengths") is the set of all n such that there is a path from the root of t to a leaf that contains exactly n black nodes.

Prove $invh\ t \Longrightarrow Bpl\ t = \{bh(t)\}$

Logarithmic height

Lemma

$rbt\ t \Longrightarrow h(t) \leq 2 * \log_2 |t|_1$

Insertion

$$insert :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$$

$$insert\ x\ t = paint\ Black\ (ins\ x\ t)$$

$$ins :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$$

$$\begin{aligned}
&ins\ x\ \langle\rangle = R\ \langle\rangle\ x\ \langle\rangle \\
&ins\ x\ (B\ l\ a\ r) = (\textsf{case}\ cmp\ x\ a\ \textsf{of} \\
&\qquad\qquad LT \Rightarrow baliL\ (ins\ x\ l)\ a\ r \\
&\qquad\quad\ \mid EQ \Rightarrow B\ l\ a\ r \\
&\qquad\quad\ \mid GT \Rightarrow baliR\ l\ a\ (ins\ x\ r))
\end{aligned}$$

$$\begin{aligned}
&ins\ x\ (R\ l\ a\ r) = (\textsf{case}\ cmp\ x\ a\ \textsf{of} \\
&\qquad\qquad LT \Rightarrow R\ (ins\ x\ l)\ a\ r \\
&\qquad\quad\ \mid EQ \Rightarrow R\ l\ a\ r \\
&\qquad\quad\ \mid GT \Rightarrow R\ l\ a\ (ins\ x\ r))
\end{aligned}$$

Adjusting colors

$ins\ x\ (B\ l\ a\ r) = \ldots\ bal\ (ins\ x\ l)\ a\ r\ \ldots\ bal\ l\ a\ (ins\ x\ r)\ \ldots$

$baliL,\ baliR :: 'a\ rbt \Rightarrow 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$
- Treat invariant violation Red-Red in l/r

$baliL\ (R\ (R\ t_1\ a_1\ t_2)\ a_2\ t_3)\ a_3\ t_4 =$
$R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
$baliL\ (R\ t_1\ a_1\ (R\ t_2\ a_2\ t_3))\ a_3\ t_4 =$
$R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$

- Principle: replace Red-Red by Red-Black
- Last equation: $baliL\ l\ a\ r = B\ l\ a\ r$
- Symmetric: $baliR$

Correctness via sorted lists

Lemma

$inorder\ (baliL\ l\ a\ r) = inorder\ l\ @\ a\ \#\ inorder\ r$

$inorder\ (baliR\ l\ a\ r) = inorder\ l\ @\ a\ \#\ inorder\ r$

Lemma

$sorted\ (inorder\ t) \Longrightarrow$

$inorder\ (ins\ x\ t) = ins\_list\ x\ (inorder\ t)$

Corollary

$sorted\ (inorder\ t) \Longrightarrow$

$inorder\ (insert\ x\ t) = ins\_list\ x\ (inorder\ t)$

Proofs easy!

Preservation of invariant

Theorem

$rbt\ t \Longrightarrow rbt\ (insert\ x\ t)$

Chapter 3

Priority Queues

9. Priority Queues

10. Leftist Heap

11. Skew Heap

12. Priority Queues Based on Braun Trees

9. Priority Queues

10. Leftist Heap

11. Skew Heap

12. Priority Queues Based on Braun Trees

Priority queue informally

Collection of elements with priorities

Operations:

- empty
- emptiness test
- insert
- get element with minimal priority
- delete element with minimal priority

We focus on the priorities:
element = priority

Priority queues are multisets

The same element can be contained multiple times
in a priority queue

$$\Longrightarrow$$

The abstract view of a priority queue is a multiset

Multisets in Isabelle

Import `"Library/Multiset"`

Interface of implementation

The type of elements (= priorities) $'a$ is a linear order

An implementation of a priority queue of elements of type $'a$ must provide

- An implementation type $'q$
- $empty :: 'q$
- $is\_empty :: 'q \Rightarrow bool$
- $insert :: 'a \Rightarrow 'q \Rightarrow 'q$
- $get\_min :: 'q \Rightarrow 'a$
- $del\_min :: 'q \Rightarrow 'q$

More operations

- $merge :: {}'q \Rightarrow {}'q \Rightarrow {}'q$
  Often provided
- decrease key/priority
  Not easy in functional setting

Correctness of implementation

A priority queue represents a multiset of priorities.
Correctness proof requires:

Abstraction function: $mset :: 'q \Rightarrow 'a\ multiset$
Invariant: $invar :: 'q \Rightarrow bool$

Correctness of implementation

Must prove $invar\ q \Longrightarrow$

$mset\ empty = \{\#\}$

$is\_empty\ q = (mset\ q = \{\#\})$

$mset\ (insert\ x\ q) = mset\ q + \{\#x\#\}$

$mset\ (del\_min\ q) = mset\ q - \{\#get\_min\ q\#\}$

$q \neq empty \Longrightarrow$
$get\_min\ q \in set\ q \wedge (\forall\, x \in set\ q.\ get\_min\ q \leq x)$
where $set\ q = set\_mset\ (mset\ q)$

$invar\ empty$
$invar\ (insert\ x\ q)$
$invar\ (del\_min\ q)$

Terminology

A tree is a heap if for every subtree
the root is $\geq$ all elements in the subtrees.

The term “heap” is frequently used synonymously with “priority queue”.

Priority queue via heap

- $empty = \langle\rangle$
- $is\_empty\ h = (h = \langle\rangle)$
- $get\_min\ \langle\_,\ a,\ \_\rangle = a$
- Assume we have $merge$
- $insert\ a\ t = merge\ \langle\langle\rangle,\ a,\ \langle\rangle\rangle\ t$
- $del\_min\ \langle l,\ a,\ r\rangle = merge\ l\ r$

Priority queue via heap

A naive merge:

$$
\begin{array}{l}
merge\ t_1\ t_2 = (\text{case } (t_1, t_2) \text{ of} \\
\quad (\langle\rangle,\ \_) \Rightarrow t_2 \mid \\
\quad (\_,\ \langle\rangle) \Rightarrow t_1 \mid \\
\quad (\langle l_1, a_1, r_1\rangle,\ \langle l_2, a_2, r_2\rangle) \Rightarrow \\
\qquad \text{if } a_1 \le a_2 \text{ then } \langle merge\ l_1\ r_1,\ a_1,\ t_2\rangle \\
\qquad \text{else } \langle t_1,\ a_2,\ merge\ l_2\ r_2\rangle
\end{array}
$$

Challenge: how to maintaining some kind of balance

Data_Structures/Leftist_Heap.thy

Leftist tree informally

The rank of a tree is the depth of the rightmost leaf.

In a leftist tree, the rank of every left child is $\geq$ the rank of its right sibling

Implementation type

datatype

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

Abbreviations $\langle\rangle$ and $\langle h,\ l,\ a,\ r\rangle$ as usual

Abstraction function:

$mset\_tree :: 'a\ lheap \Rightarrow 'a\ multiset$

$mset\_tree\ \langle\rangle = \{\#\}$

$mset\_tree\ \langle\_,\ l,\ a,\ r\rangle =$

$\{\#a\#\} + mset\_tree\ l + mset\_tree\ r$

Heap

$$heap :: 'a\ lheap \Rightarrow bool$$

$$heap\ \langle\rangle = True$$

$$heap\ \langle\_,\ l,\ a,\ r\rangle = (heap\ l \wedge heap\ r \wedge$$
$$(\forall\, x \in\#\ mset\_tree\ l + mset\_tree\ r.\ a \leq x))$$

Leftist tree

$rank :: {}'a\ lheap \Rightarrow nat$

$rank\ \langle\rangle = 0$
$rank\ \langle\_,\ \_,\ \_,\ r\rangle = rank\ r + 1$

Node $\langle n,\ l,\ a,\ r\rangle$: $n =$ rank of node

$ltree :: {}'a\ lheap \Rightarrow bool$

$ltree\ \langle\rangle = True$
$ltree\ \langle n,\ l,\ \_,\ r\rangle =$
$(n = rank\ r + 1 \wedge rank\ r \leq rank\ l \wedge ltree\ l \wedge ltree\ r)$

Leftist heap invariant

$$invar\ h = (heap\ h \wedge ltree\ h)$$

Why leftist tree?

Lemma $\mathit{ltree}\ t \Longrightarrow 2^{\mathit{rank}\ t} \leq |t|_1$

Lemma Execution time of $\mathit{merge}\ t_1\ t_2$ is bounded by $\mathit{rank}\ t_1 + \mathit{rank}\ t_2$

merge

Principle: descend on the right

$merge\ \langle\rangle\ t_2 = t_2$
$merge\ t_1\ \langle\rangle = t_1$
$merge\ \langle n_1,\ l_1,\ a_1,\ r_1\rangle\ \langle n_2,\ l_2,\ a_2,\ r_2\rangle =$
$(\text{if}\ a_1 \le a_2\ \text{then}\ node\ l_1\ a_1\ (merge\ r_1\ \langle n_2,\ l_2,\ a_2,\ r_2\rangle)$
$\ \text{else}\ node\ l_2\ a_2\ (merge\ r_2\ \langle n_1,\ l_1,\ a_1,\ r_1\rangle))$

$node :: 'a\ lheap \Rightarrow 'a \Rightarrow 'a\ lheap \Rightarrow 'a\ lheap$

$node\ l\ a\ r =$
$(\text{let}\ rl = rk\ l;\ rr = rk\ r$
$\ \text{in if}\ rr \le rl\ \text{then}\ \langle rr + 1,\ l,\ a,\ r\rangle\ \text{else}\ \langle rl + 1,\ r,\ a,\ l\rangle)$
$\text{where}\ rk\ \langle n,\ \_,\ \_,\ \_\rangle = n$

$merge$

$$merge\ \langle n_1,\ l_1,\ a_1,\ r_1\rangle\ \langle n_2,\ l_2,\ a_2,\ r_2\rangle =$$
$$(\text{if } a_1 \le a_2 \text{ then } node\ l_1\ a_1\ (merge\ r_1\ \langle n_2,\ l_2,\ a_2,\ r_2\rangle)$$
$$\text{else } node\ l_2\ a_2\ (merge\ r_2\ \langle n_1,\ l_1,\ a_1,\ r_1\rangle))$$

Function $merge$ terminates because
decreases with every recursive call.

Functional correctness proofs

including preservation of $invar$

Straightforward

Logarithmic complexity

Complexity measures t_merge, t_insert t_del_min: count calls of $merge$.

Lemma $t\_merge\ l\ r \leq rank\ l + rank\ r + 1$
Lemma $ltree\ l \wedge ltree\ r \Longrightarrow$
$t\_merge\ l\ r \leq \log_2 |l|_1 + \log_2 |r|_1 + 1$

Lemma
$ltree\ t \Longrightarrow t\_insert\ x\ t \leq \log_2 |t|_1 + 2$

Lemma
$ltree\ t \Longrightarrow t\_del\_min\ t \leq 2 * \log_2 |t|_1 + 1$

Can we avoid the rank info in each node?

Archive of Formal Proofs

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

Note: $merge$ is called $meld$

Implementation type

Ordinary binary trees

Invariant: $heap$

$meld$

Principle: swap subtrees when descending

$meld\ h_1\ h_2 =$
(case h_1 of
$\quad \langle\rangle \Rightarrow h_2$
$\mid \langle l_1,\ a_1,\ r_1\rangle \Rightarrow$
$\quad\quad$ case h_2 of
$\quad\quad\quad \langle\rangle \Rightarrow h_1$
$\quad\quad \mid \langle l_2,\ a_2,\ r_2\rangle \Rightarrow$
$\quad\quad\quad\quad$ if $a_1 \le a_2$ then $\langle meld\ h_2\ r_1,\ a_1,\ l_1\rangle$
$\quad\quad\quad\quad$ else $\langle meld\ h_1\ r_2,\ a_2,\ l_2\rangle$)

Function $meld$ terminates because ...

Functional correctness proofs

including preservation of $heap$

Straightforward

Logarithmic complexity

Amortized only:

Theorem A sequence of n $insert$, del_min and $meld$ operations runs in time $O(n * \log n)$.

$\Longrightarrow$

Average cost of each operation is $O(\log n)$
(even in the worst case)

Archive of Formal Proofs

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

What is a Braun tree?

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

$braun\ \langle\rangle = True$

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

$(|r| \le |l| \wedge |l| \le Suc\ |r| \wedge braun\ l \wedge braun\ r)$

Lemma $braun\ t \Longrightarrow 2^{h(t)} \le 2 * |t| + 1$

Priority queue implementation

Implementation type: ordinary binary trees

Invariants: $heap$ and $braun$

No $merge$ — $insert$ and del_min defined explicitly

$insert$

$insert :: 'a \Rightarrow 'a\ tree \Rightarrow 'a\ tree$

$insert\ a\ \langle\rangle = \langle\langle\rangle,\ a,\ \langle\rangle\rangle$

$insert\ a\ \langle l,\ x,\ r\rangle =$

$(\textsf{if}\ a < x\ \textsf{then}\ \langle insert\ x\ r,\ a,\ l\rangle\ \textsf{else}\ \langle insert\ a\ r,\ x,\ l\rangle)$

Correctness and preservation of invariant straightforward.

del_min

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

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

$sift\_down$

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

$$sift\_down\ \langle\rangle\ a\ \langle\rangle = \langle\langle\rangle,\ a,\ \langle\rangle\rangle$$

$$sift\_down\ \langle\langle\rangle,\ x,\ \langle\rangle\rangle\ a\ \langle\rangle =$$
$$(\text{if } a \le x \text{ then } \langle\langle\langle\rangle,\ x,\ \langle\rangle\rangle,\ a,\ \langle\rangle\rangle$$
$$\text{ else } \langle\langle\langle\rangle,\ a,\ \langle\rangle\rangle,\ x,\ \langle\rangle\rangle)$$

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

Functional correctness proofs for deletion

including preservation of *heap* and *braun*

Many lemmas, mostly straightforward

Logarithmic complexity

Running time of $insert$, del_left and $sift\_down$ (and therefore del_min) bounded by height

Remember: $braun\ t \Longrightarrow 2^{h(t)} \leq 2 * |t| + 1$

$\Longrightarrow$

Above running times logarithmic in size

Sorting with priority queue

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

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

$sort\_pq = mins \circ pq$

Complexity of $sort$: $O(n \log n)$
if all priority queue functions have complexity $O(\log n)$

Sorting with priority queue

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

Not optimal. Linear time possible.