

Root-Balanced Tree

Tobias Nipkow

June 15, 2026

Abstract

Andersson [1, 2] introduced *general balanced trees*, search trees based on the design principle of partial rebuilding: perform update operations naively until the tree becomes too unbalanced, at which point a whole subtree is rebalanced. This article defines and analyzes a functional version of general balanced trees, which we call *root-balanced trees*. Using a lightweight model of execution time, amortized logarithmic complexity is verified in the theorem prover Isabelle.

This is the Isabelle formalization of the material described in the APLAS 2017 article *Verified Root-Balanced Trees* by the same author [3] which also presents experimental results that show competitiveness of root-balanced with AVL and red-black trees.

1 Time Monad

theory *Time-Monad*

imports

Main

HOL-Library.Monad-Syntax

begin

datatype $'a\ tm = TM\ (val: 'a)\ nat$

fun $val :: 'a\ tm \Rightarrow 'a$ **where**

$val\ (TM\ v\ n) = v$

fun $time :: 'a\ tm \Rightarrow nat$ **where**

$time\ (TM\ v\ n) = n$

definition $bind_tm :: 'a\ tm \Rightarrow ('a \Rightarrow 'b\ tm) \Rightarrow 'b\ tm$ **where**

$bind_tm\ s\ f = (case\ s\ of\ TM\ u\ m \Rightarrow case\ f\ u\ of\ TM\ v\ n \Rightarrow TM\ v\ (m+n))$

adhoc-overloading $Monad-Syntax.bind \Leftarrow bind_tm$

definition $tick\ v = TM\ v\ 1$

definition $return\ v = TM\ v\ 0$

abbreviation $eqtick :: 'a\ tm \Rightarrow 'a\ tm \Rightarrow bool$ (**infix** $\langle =1 \rangle 50$) **where**
 $eqtick\ l\ r \equiv (l = (r \gg= tick))$

translations $CONST\ eqtick\ l\ r\ <= (l = (bind-tm\ r\ CONST\ tick))$

lemmas $tm-simps = bind-tm-def\ return-def\ tick-def$

lemma $time-return[simp]: time\ (return\ x) = 0$
 $\langle proof \rangle$

lemma $surj-TM: v = val\ tm \Longrightarrow t = time\ tm \Longrightarrow tm = TM\ v\ t$
 $\langle proof \rangle$

The following lemmas push *Time-Monad.val* into a monadic term:

lemma $val-return[simp]: val\ (return\ x) = x$
 $\langle proof \rangle$

lemma $val-bind-tm[simp]: val\ (bind-tm\ m\ f) = (let\ x = val\ m\ in\ val(f\ x))$
 $\langle proof \rangle$

lemma $val-tick[simp]: val\ (tick\ x) = x$
 $\langle proof \rangle$

lemma $val-let: val\ (let\ x = t\ in\ f(x)) = (let\ x = t\ in\ val(f\ x))$
 $\langle proof \rangle$

lemma $let-id: (let\ x = t\ in\ x) = t$
 $\langle proof \rangle$

lemmas $val-simps =$
 $val-return$
 $val-bind-tm$
 $val-tick$
 $val-let$
 $let-id$
 $if-distrib[of\ val]$
 $prod.case-distrib[of\ val]$

lemmas $val-cong = arg-cong[\mathbf{where}\ f=val]$

The following congruence rule enables termination proofs for recursive functions using this monad.

lemma $bind-tm-cong[fundef-cong]:$
assumes $m1 = m2$
assumes $f1\ (val\ m1) = f2\ (val\ m2)$
shows $m1 \gg= f1 = m2 \gg= f2$
 $\langle proof \rangle$

hide-const *TM*

end

2 Root Balanced Tree

theory *Root-Balanced-Tree*

imports

Amortized-Complexity.Amortized-Framework0

HOL-Library.Tree-Multiset

HOL-Data-Structures.Tree-Set

HOL-Data-Structures.Balance

Time-Monad

begin

2.1 Time Prelude

Redefinition of some auxiliary functions, but now with *tm* monad:

2.1.1 *size-tree*

fun *size-tree-tm* :: 'a tree \Rightarrow nat *tm* **where**

size-tree-tm $\langle \rangle$ = 1 return 0 |

size-tree-tm $\langle l, x, r \rangle$ = 1

do { *m* \leftarrow *size-tree-tm* *l*;

n \leftarrow *size-tree-tm* *r*;

return (*m*+*n*+1)}

definition *size-tree* :: 'a tree \Rightarrow nat **where**

size-tree *t* = val(*size-tree-tm* *t*)

lemma *size-tree-Leaf*[*simp,code*]: *size-tree* $\langle \rangle$ = 0

\langle proof \rangle

lemma *size-tree-Node*[*simp,code*]:

size-tree $\langle l, x, r \rangle$ =

(let *m* = *size-tree* *l*;

n = *size-tree* *r*

in *m*+*n*+1)

\langle proof \rangle

lemma *size-tree*: *size-tree* *t* = *size* *t*

\langle proof \rangle

definition *T-size-tree* :: 'a tree \Rightarrow nat **where**

T-size-tree *t* = *time*(*size-tree-tm* *t*)

lemma *T-size-tree-Leaf*: *T-size-tree* $\langle \rangle$ = 1

<proof>

lemma *T-size-tree-Node*:

T-size-tree $\langle l, x, r \rangle = T\text{-size-tree } l + T\text{-size-tree } r + 1$
<proof>

lemma *T-size-tree*: *T-size-tree* $t = 2 * \text{size } t + 1$

<proof>

2.1.2 *inorder*

fun *inorder2-tm* :: 'a tree \Rightarrow 'a list \Rightarrow 'a list tm **where**

inorder2-tm $\langle \rangle xs = 1$ return *xs* |

inorder2-tm $\langle l, x, r \rangle xs = 1$

do { *rs* \leftarrow *inorder2-tm* *r* *xs*; *inorder2-tm* *l* (*x* # *rs*) }

definition *inorder2* :: 'a tree \Rightarrow 'a list \Rightarrow 'a list **where**

inorder2 *t* *xs* = *val*(*inorder2-tm* *t* *xs*)

lemma *inorder2-Leaf*[*simp,code*]: *inorder2* $\langle \rangle xs = xs$

<proof>

lemma *inorder2-Node*[*simp,code*]:

inorder2 $\langle l, x, r \rangle xs = (\text{let } rs = \text{inorder2 } r \text{ } xs \text{ in } \text{inorder2 } l \text{ } (x \# rs))$

<proof>

lemma *inorder2*: *inorder2* *t* *xs* = *Tree.inorder2* *t* *xs*

<proof>

definition *T-inorder2* :: 'a tree \Rightarrow 'a list \Rightarrow nat **where**

T-inorder2 *t* *xs* = *time*(*inorder2-tm* *t* *xs*)

lemma *T-inorder2-Leaf*: *T-inorder2* $\langle \rangle xs = 1$

<proof>

lemma *T-inorder2-Node*:

T-inorder2 $\langle l, x, r \rangle xs = T\text{-inorder2 } r \text{ } xs + T\text{-inorder2 } l \text{ } (x \# \text{inorder2 } r \text{ } xs) + 1$

<proof>

lemma *T-inorder2*: *T-inorder2* *t* *xs* = $2 * \text{size } t + 1$

<proof>

2.1.3 *split-min*

fun *split-min-tm* :: 'a tree \Rightarrow ('a * 'a tree) tm **where**

split-min-tm *Leaf* = 1 return *undefined* |

split-min-tm (*Node* *l* *x* *r*) = 1

(if *l* = *Leaf* then return (*x*, *r*)

else do { (*y*, *l'*) \leftarrow *split-min-tm* *l*; return (*y*, *Node* *l'* *x* *r*) }

definition *split-min* :: 'a tree \Rightarrow ('a * 'a tree) **where**
split-min t = val (*split-min-tm* t)

lemma *split-min-Node*[simp,code]:
split-min (Node l x r) =
 (if l = Leaf then (x,r)
 else let (y,l') = *split-min* l in (y, Node l' x r))
 <proof>

definition *T-split-min* :: 'a tree \Rightarrow nat **where**
T-split-min t = time (*split-min-tm* t)

lemma *T-split-min-Node*[simp]:
T-split-min (Node l x r) = (if l = Leaf then 1 else *T-split-min* l + 1)
 <proof>

lemma *split-minD*:
split-min t = (x,t') \Longrightarrow t \neq Leaf \Longrightarrow x # inorder t' = inorder t
 <proof>

2.1.4 Balancing

fun *bal-tm* :: nat \Rightarrow 'a list \Rightarrow ('a tree * 'a list) **tm where**
bal-tm n xs = 1
 (if n=0 then return (Leaf,xs) else
 (let m = n div 2
 in do { (l, ys) \leftarrow *bal-tm* m xs;
 (r, zs) \leftarrow *bal-tm* (n-1-m) (tl ys);
 return (Node l (hd ys) r, zs)}))

declare *bal-tm.simps*[simp del]

lemma *bal-tm-simps*:
bal-tm 0 xs = 1 return(Leaf, xs)
 n > 0 \Longrightarrow
bal-tm n xs = 1
 (let m = n div 2
 in do { (l, ys) \leftarrow *bal-tm* m xs;
 (r, zs) \leftarrow *bal-tm* (n-1-m) (tl ys);
 return (Node l (hd ys) r, zs)})
 <proof>

definition *bal* :: nat \Rightarrow 'a list \Rightarrow ('a tree * 'a list) **where**
bal n xs = val (*bal-tm* n xs)

lemma *bal-def2*[code]:
bal n xs =
 (if n=0 then (Leaf,xs) else

$(let\ m = n\ div\ 2;$
 $\quad (l,\ ys) = bal\ m\ xs;$
 $\quad (r,\ zs) = bal\ (n-1-m)\ (tl\ ys)$
 $\quad in\ (Node\ l\ (hd\ ys)\ r,\ zs))$
 $\langle proof \rangle$

lemma *bal-simps*:
 $bal\ 0\ xs = (Leaf,\ xs)$
 $n > 0 \implies$
 $bal\ n\ xs =$
 $(let\ m = n\ div\ 2;$
 $\quad (l,\ ys) = bal\ m\ xs;$
 $\quad (r,\ zs) = bal\ (n-1-m)\ (tl\ ys)$
 $\quad in\ (Node\ l\ (hd\ ys)\ r,\ zs))$
 $\langle proof \rangle$

lemma *bal-eq*: $bal\ n\ xs = Balance.bal\ n\ xs$
 $\langle proof \rangle$

definition *T-bal* :: $nat \Rightarrow 'a\ list \Rightarrow nat$ **where**
 $T-bal\ n\ xs = time\ (bal-tm\ n\ xs)$

lemma *T-bal*: $T-bal\ n\ xs = 2*n+1$
 $\langle proof \rangle$

definition *bal-list-tm* :: $nat \Rightarrow 'a\ list \Rightarrow 'a\ tree\ tm$ **where**
 $bal-list-tm\ n\ xs = do\ \{ (t,-) \leftarrow bal-tm\ n\ xs; return\ t \}$

definition *bal-list* :: $nat \Rightarrow 'a\ list \Rightarrow 'a\ tree$ **where**
 $bal-list\ n\ xs = val\ (bal-list-tm\ n\ xs)$

lemma *bal-list-def2*[code]: $bal-list\ n\ xs = (let\ (t,ys) = bal\ n\ xs\ in\ t)$
 $\langle proof \rangle$

lemma *bal-list*: $bal-list\ n\ xs = Balance.bal-list\ n\ xs$
 $\langle proof \rangle$

definition *bal-tree-tm* :: $nat \Rightarrow 'a\ tree \Rightarrow 'a\ tree\ tm$ **where**
 $bal-tree-tm\ n\ t =1\ do\ \{ xs \leftarrow inorder2-tm\ t\ []; bal-list-tm\ n\ xs \}$

definition *bal-tree* :: $nat \Rightarrow 'a\ tree \Rightarrow 'a\ tree$ **where**
 $bal-tree\ n\ t = val\ (bal-tree-tm\ n\ t)$

lemma *bal-tree-def2*[code]:
 $bal-tree\ n\ t = (let\ xs = inorder2\ t\ []\ in\ bal-list\ n\ xs)$
 $\langle proof \rangle$

lemma *bal-tree*: $bal-tree\ n\ t = Balance.bal-tree\ n\ t$

<proof>

definition *T-bal-tree* :: *nat* \Rightarrow *'a tree* \Rightarrow *nat* **where**
T-bal-tree *n xs* = *time (bal-tree-tm n xs)*

lemma *T-bal-tree*: *n = size xs* \implies *T-bal-tree n xs* = $4*n+3$
<proof>

2.2 Naive implementation (insert only)

fun *node* :: *bool* \Rightarrow *'a tree* \Rightarrow *'a* \Rightarrow *'a tree* \Rightarrow *'a tree* **where**
node twist s x t = (*if twist then Node t x s else Node s x t*)

datatype *'a up* = *Same* | *Bal 'a tree* | *Unbal 'a tree*

locale *RBTi1* =

fixes *bal-i* :: *nat* \Rightarrow *nat* \Rightarrow *bool*

assumes *bal-i-balance*:

bal-i (size t) (height (balance-tree (t::'a::linorder tree)))

assumes *mono-bal-i*: $\llbracket \text{bal-i } n \ h; \ n \leq \ n'; \ h' \leq \ h \rrbracket \implies \text{bal-i } n' \ h'$

begin

2.2.1 Functions

definition *up* :: *'a* \Rightarrow *'a tree* \Rightarrow *bool* \Rightarrow *'a up* \Rightarrow *'a up* **where**

up x sib twist u = (*case u of Same* \Rightarrow *Same* |

Bal t \Rightarrow *Bal(node twist t x sib)* |

Unbal t \Rightarrow *let t' = node twist t x sib; h' = height t'; n' = size t'*

in if bal-i n' h' then Unbal t'

else Bal(balance-tree t'))

declare *up-def*[*simp*]

fun *ins* :: *nat* \Rightarrow *nat* \Rightarrow *'a::linorder* \Rightarrow *'a tree* \Rightarrow *'a up* **where**

ins n d x Leaf =

(*if bal-i (n+1) (d+1) then Bal (Node Leaf x Leaf) else Unbal (Node Leaf x Leaf)*)

|

ins n d x (Node l y r) =

(*case cmp x y of*

LT \Rightarrow *up y r False (ins n (d+1) x l)* |

EQ \Rightarrow *Same* |

GT \Rightarrow *up y l True (ins n (d+1) x r)*)

fun *insert* :: *'a::linorder* \Rightarrow *'a tree* \Rightarrow *'a tree* **where**

insert x t =

(*case ins (size t) 0 x t of*

Same \Rightarrow *t* |

Bal t' \Rightarrow *t'*)

2.2.2 Functional Correctness and Invariants

lemma *height-balance*: $\llbracket \neg \text{bal-}i \text{ (size } t) \text{ } h \rrbracket$
 $\implies \text{height (balance-tree (t::'a::linorder tree))} < h$
 $\langle \text{proof} \rangle$

lemma *mono-bal- i'* :
 $\llbracket \text{ASSUMPTION}(\text{bal-}i \text{ } n \text{ } h); n \leq n'; h' \leq h \rrbracket \implies \text{bal-}i \text{ } n' \text{ } h'$
 $\langle \text{proof} \rangle$

lemma *inorder-ins*: $\text{sorted}(\text{inorder } t) \implies$
 $(\text{ins } n \text{ } d \text{ } x \text{ } t = \text{Same} \longrightarrow \text{ins-list } x \text{ (inorder } t) = \text{inorder } t) \wedge$
 $(\text{ins } n \text{ } d \text{ } x \text{ } t = \text{Bal } t' \longrightarrow \text{ins-list } x \text{ (inorder } t) = \text{inorder } t') \wedge$
 $(\text{ins } n \text{ } d \text{ } x \text{ } t = \text{Unbal } t' \longrightarrow \text{ins-list } x \text{ (inorder } t) = \text{inorder } t')$
 $\langle \text{proof} \rangle$

lemma *ins-size*:
shows $\text{ins } n \text{ } d \text{ } x \text{ } t = \text{Bal } t' \implies \text{size } t' = \text{size } t + 1$
and $\text{ins } n \text{ } d \text{ } x \text{ } t = \text{Unbal } t' \implies \text{size } t' = \text{size } t + 1$
 $\langle \text{proof} \rangle$

lemma *ins-height*:
shows $\text{ins } n \text{ } d \text{ } x \text{ } t = \text{Bal } t' \implies \text{height } t' \leq \text{height } t + 1$
and $\text{ins } n \text{ } d \text{ } x \text{ } t = \text{Unbal } t' \implies \text{height } t \leq \text{height } t' \wedge \text{height } t' \leq \text{height } t + 1$
 $\langle \text{proof} \rangle$

lemma *bal-i0*: $\text{bal-}i \text{ } 0 \text{ } 0$
 $\langle \text{proof} \rangle$

lemma *bal-i1*: $\text{bal-}i \text{ } 1 \text{ } 1$
 $\langle \text{proof} \rangle$

lemma *bal-i-ins-Unbal*:
assumes $\text{ins } n \text{ } d \text{ } x \text{ } t = \text{Unbal } t'$ **shows** $\text{bal-}i \text{ (size } t') \text{ (height } t')$
 $\langle \text{proof} \rangle$

lemma *unbal-ins-Unbal*:
 $\text{ins } n \text{ } d \text{ } x \text{ } t = \text{Unbal } t' \implies \neg \text{bal-}i \text{ (} n+1 \text{) (height } t' + d)$
 $\langle \text{proof} \rangle$

lemma *height-Unbal-l*: **assumes** $\text{ins } n \text{ (} d+1 \text{) } x \text{ } l = \text{Unbal } l'$
 $\text{bal-}i \text{ } n \text{ (height } \langle l, y, r \rangle + d)$
shows $\text{height } r < \text{height } l'$ (**is** ?P)
 $\langle \text{proof} \rangle$

lemma *height-Unbal-r*: **assumes** $\text{ins } n \text{ (} d+1 \text{) } x \text{ } r = \text{Unbal } r'$
 $\text{bal-}i \text{ } n \text{ (height } \langle l, y, r \rangle + d)$
shows $\text{height } l < \text{height } r'$ (**is** ?P)
 $\langle \text{proof} \rangle$

lemma *ins-bal-i-Bal*:

```

[[ ins n d x t = Bal t'; bal-i n (height t + d) ]]
  => bal-i (n+1) (height t' + d)
<proof>

```

lemma *ins0-neq-Unbal*: **assumes** $n \geq \text{size } t$ **shows** $\text{ins } n \ 0 \ a \ t \neq \text{Unbal } t'$
 <proof>

lemma *inorder-insert*: $\text{sorted}(\text{inorder } t)$
 $\implies \text{inorder } (\text{insert } x \ t) = \text{ins-list } x \ (\text{inorder } t)$
 <proof>

lemma *bal-i-insert*: **assumes** $\text{bal-i } (\text{size } t) \ (\text{height } t)$
shows $\text{bal-i } (\text{size}(\text{insert } x \ t)) \ (\text{height}(\text{insert } x \ t))$
 <proof>

end

This is just a test that (a simplified version of) the intended interpretation works (so far):

interpretation *Test*: $\text{RBTi1 } \lambda n \ h. \ h \leq \log 2 \ (\text{real}(n + 1)) + 1$
 <proof>

2.3 Efficient Implementation (insert only)

fun *imbal* :: 'a tree \Rightarrow nat **where**
imbal Leaf = 0 |
imbal (Node l - r) = nat(abs(int(size l) - int(size r))) - 1

declare *imbal.simps* [simp del]

lemma *imbal0-if-wbalanced*: $\text{wbalanced } t \implies \text{imbal } t = 0$
 <proof>

The degree of imbalance allowed: how far from the perfect balance may the tree degenerate.

axiomatization *c* :: real **where**
c1: $c > 1$

definition *bal-log* :: 'a tree \Rightarrow bool **where**
bal-log t = (height t \leq ceiling($c * \log 2 \ (\text{size1 } t)$))

fun *hchild* :: 'a tree \Rightarrow 'a tree **where**
hchild (Node l - r) = (if height l \leq height r then r else l)

lemma *size1-imbal*:
assumes $\neg \text{bal-log } t$ **and** $\text{bal-log } (\text{hchild } t)$ **and** $t \neq \text{Leaf}$
shows $\text{imbal } t > (2 \ \text{powr } (1 - 1/c) - 1) * \text{size1 } (t) - 1$
 <proof>

The following key lemma shows that *imbal* is a suitable potential because

it can pay for the linear-time cost of restructuring a tree that is not balanced but whose higher son is.

lemma *size1-imal2*:

assumes $\neg \text{bal-log } t$ **and** $\text{bal-log } (\text{hchild } t)$ **and** $t \neq \text{Leaf}$

shows $\text{size1 } (t) < (2 \text{ powr } (1/c) / (2 - 2 \text{ powr } (1/c))) * (\text{imal } t + 1)$

<proof>

datatype $'a \text{ up2} = \text{Same2} \mid \text{Bal2 } 'a \text{ tree} \mid \text{Unbal2 } 'a \text{ tree } \text{nat } \text{nat}$

type-synonym $'a \text{ rbt1} = 'a \text{ tree} * \text{nat}$

An implementation where size and height are computed incrementally:

locale *RBTi2* = *RBTi1* +

fixes $e :: \text{real}$

assumes $e0: e > 0$

assumes *imal-size*:

$\llbracket \neg \text{bal-i } (\text{size } t) (\text{height } t);$
 $\text{bal-i } (\text{size}(\text{hchild } t)) (\text{height}(\text{hchild } t));$
 $t \neq \text{Leaf} \rrbracket$

$\implies e * (\text{imal } t + 1) \geq \text{size1 } (t::'a::\text{linorder tree})$

begin

2.3.1 Functions

definition *up2* :: $'a \Rightarrow 'a \text{ tree} \Rightarrow \text{bool} \Rightarrow 'a \text{ up2} \Rightarrow 'a \text{ up2}$ **where**

up2 $x \text{ sib twist } u = (\text{case } u \text{ of } \text{Same2} \Rightarrow \text{Same2} \mid$

$\text{Bal2 } t \Rightarrow \text{Bal2}(\text{node twist } t \text{ } x \text{ sib}) \mid$

$\text{Unbal2 } t \text{ } n1 \text{ } h1 \Rightarrow$

$\text{let } n2 = \text{size sib}; h2 = \text{height sib};$

$t' = \text{node twist } t \text{ } x \text{ sib};$

$n' = n1 + n2 + 1; h' = \max h1 \text{ } h2 + 1$

$\text{in if } \text{bal-i } n' \text{ } h' \text{ then } \text{Unbal2 } t' \text{ } n' \text{ } h'$

$\text{else } \text{Bal2}(\text{bal-tree } n' \text{ } t')$)

declare *up2-def[simp]*

up2 traverses *sib* twice; unnecessarily, as it turns out:

definition *up3-tm* :: $'a \Rightarrow 'a \text{ tree} \Rightarrow \text{bool} \Rightarrow 'a \text{ up2} \Rightarrow 'a \text{ up2 } \text{tm}$ **where**

up3-tm $x \text{ sib twist } u = 1 (\text{case } u \text{ of}$

$\text{Same2} \Rightarrow \text{return } \text{Same2} \mid$

$\text{Bal2 } t \Rightarrow \text{return } (\text{Bal2}(\text{node twist } t \text{ } x \text{ sib})) \mid$

$\text{Unbal2 } t \text{ } n1 \text{ } h1 \Rightarrow$

$\text{do } \{ n2 \leftarrow \text{size-tree-tm } \text{sib};$

$\text{let } t' = \text{node twist } t \text{ } x \text{ sib};$

$n' = n1 + n2 + 1;$

$h' = h1 + 1$

$\text{in if } \text{bal-i } n' \text{ } h' \text{ then return } (\text{Unbal2 } t' \text{ } n' \text{ } h')$

$\text{else do } \{ t'' \leftarrow \text{bal-tree-tm } n' \text{ } t';$

$\text{return } (\text{Bal2 } t'') \}$)

definition $up3 :: 'a \Rightarrow 'a \text{ tree} \Rightarrow \text{bool} \Rightarrow 'a \text{ up2} \Rightarrow 'a \text{ up2}$ **where**
 $up3 \ a \ sib \ twist \ u = \text{val} \ (up3\text{-tm} \ a \ sib \ twist \ u)$

lemma $up3\text{-def2}[simp,code]$:

$up3 \ x \ sib \ twist \ u = (\text{case } u \ \text{of}$
 $\text{Same2} \Rightarrow \text{Same2} \ |$
 $\text{Bal2} \ t \Rightarrow \text{Bal2} \ (\text{node} \ \text{twist} \ t \ x \ sib) \ |$
 $\text{Unbal2} \ t \ n1 \ h1 \Rightarrow$
 $\text{let } n2 = \text{size-tree} \ sib; \ t' = \text{node} \ \text{twist} \ t \ x \ sib; \ n' = n1 + n2 + 1; \ h' = h1 + 1$
 $\text{in if } \text{bal-i} \ n' \ h' \ \text{then } \text{Unbal2} \ t' \ n' \ h'$
 $\text{else let } t'' = \text{bal-tree} \ n' \ t' \ \text{in } \text{Bal2} \ t'')$
 $\langle \text{proof} \rangle$

definition $T\text{-up3} :: 'a \Rightarrow 'a \text{ tree} \Rightarrow \text{bool} \Rightarrow 'a \text{ up2} \Rightarrow \text{nat}$ **where**
 $T\text{-up3} \ x \ sib \ twist \ u = \text{time} \ (up3\text{-tm} \ x \ sib \ twist \ u)$

lemma $T\text{-up3}\text{-def2}[simp]$: $T\text{-up3} \ x \ sib \ twist \ u =$

$(\text{case } u \ \text{of } \text{Same2} \Rightarrow 1 \ |$
 $\text{Bal2} \ t \Rightarrow 1 \ |$
 $\text{Unbal2} \ t \ n1 \ h1 \Rightarrow$
 $\text{let } n2 = \text{size} \ sib; \ t' = \text{node} \ \text{twist} \ t \ x \ sib; \ h' = h1 + 1; \ n' = n1 + n2 + 1$
 $\text{in } 2 * \text{size} \ sib + 1 + (\text{if } \text{bal-i} \ n' \ h' \ \text{then } 1 \ \text{else } T\text{-bal-tree} \ n' \ t' + 1))$
 $\langle \text{proof} \rangle$

fun $ins2 :: \text{nat} \Rightarrow \text{nat} \Rightarrow 'a::\text{linorder} \Rightarrow 'a \text{ tree} \Rightarrow 'a \text{ up2}$ **where**

$ins2 \ n \ d \ x \ \text{Leaf} =$
 $(\text{if } \text{bal-i} \ (n+1) \ (d+1) \ \text{then } \text{Bal2} \ (\text{Node} \ \text{Leaf} \ x \ \text{Leaf}) \ \text{else } \text{Unbal2} \ (\text{Node} \ \text{Leaf} \ x$
 $\text{Leaf}) \ 1 \ 1) \ |$
 $ins2 \ n \ d \ x \ (\text{Node} \ l \ y \ r) =$
 $(\text{case } \text{cmp} \ x \ y \ \text{of}$
 $\text{LT} \Rightarrow \text{up2} \ y \ r \ \text{False} \ (ins2 \ n \ (d+1) \ x \ l) \ |$
 $\text{EQ} \Rightarrow \text{Same2} \ |$
 $\text{GT} \Rightarrow \text{up2} \ y \ l \ \text{True} \ (ins2 \ n \ (d+1) \ x \ r))$

Definition of timed final insertion function:

fun $ins3\text{-tm} :: \text{nat} \Rightarrow \text{nat} \Rightarrow 'a::\text{linorder} \Rightarrow 'a \text{ tree} \Rightarrow 'a \text{ up2} \ \text{tm}$ **where**

$ins3\text{-tm} \ n \ d \ x \ \text{Leaf} = 1$
 $(\text{if } \text{bal-i} \ (n+1) \ (d+1) \ \text{then } \text{return} \ (\text{Bal2} \ (\text{Node} \ \text{Leaf} \ x \ \text{Leaf}))$
 $\text{else } \text{return} \ (\text{Unbal2} \ (\text{Node} \ \text{Leaf} \ x \ \text{Leaf}) \ 1 \ 1)) \ |$
 $ins3\text{-tm} \ n \ d \ x \ (\text{Node} \ l \ y \ r) = 1$
 $(\text{case } \text{cmp} \ x \ y \ \text{of}$
 $\text{LT} \Rightarrow \text{do} \ \{l' \leftarrow \text{ins3}\text{-tm} \ n \ (d+1) \ x \ l; \ \text{up3}\text{-tm} \ y \ r \ \text{False} \ l'\} \ |$
 $\text{EQ} \Rightarrow \text{return} \ \text{Same2} \ |$
 $\text{GT} \Rightarrow \text{do} \ \{r' \leftarrow \text{ins3}\text{-tm} \ n \ (d+1) \ x \ r; \ \text{up3}\text{-tm} \ y \ l \ \text{True} \ r'\})$

definition $ins3 :: \text{nat} \Rightarrow \text{nat} \Rightarrow 'a::\text{linorder} \Rightarrow 'a \text{ tree} \Rightarrow 'a \text{ up2}$ **where**
 $ins3 \ n \ d \ x \ t = \text{val} \ (ins3\text{-tm} \ n \ d \ x \ t)$

lemma *ins3-Leaf*[simp,code]:

ins3 n d x Leaf =
(if *bal-i (n+1) (d+1)* then *Bal2 (Node Leaf x Leaf)* else *Unbal2 (Node Leaf x Leaf) 1 1*)
<proof>

lemma *ins3-Node*[simp,code]:

ins3 n d x (Node l y r) =
(case *cmp x y* of
 LT ⇒ let *l' = ins3 n (d+1) x l in up3 y r False l' |*
 EQ ⇒ *Same2 |*
 GT ⇒ let *r' = ins3 n (d+1) x r in up3 y l True r')*
<proof>

definition *T-ins3* :: *nat* ⇒ *nat* ⇒ 'a::linorder ⇒ 'a tree ⇒ *nat* **where**
T-ins3 n d x t = time(ins3-tm n d x t)

lemma *T-ins3-Leaf*[simp]: *T-ins3 n d x Leaf = 1*
<proof>

lemma *T-ins3-Node*[simp]: *T-ins3 n d x (Node l y r) =*
(case *cmp x y* of
 LT ⇒ *T-ins3 n (d+1) x l + T-up3 y r False (ins3 n (d+1) x l) |*
 EQ ⇒ *0 |*
 GT ⇒ *T-ins3 n (d+1) x r + T-up3 y l True (ins3 n (d+1) x r) + 1*)
<proof>

fun *insert2* :: 'a::linorder ⇒ 'a rbt1 ⇒ 'a rbt1 **where**

insert2 x (t,n) =
(case *ins2 n 0 x t* of
 Same2 ⇒ (t,n) |
 Bal2 t' ⇒ (t',n+1))

fun *insert3-tm* :: 'a::linorder ⇒ 'a rbt1 ⇒ 'a rbt1 tm **where**

insert3-tm x (t,n) = 1
(do { *u* ← *ins3-tm n 0 x t*;
 case *u* of
 Same2 ⇒ return (t,n) |
 Bal2 t' ⇒ return (t',n+1) |
 Unbal2 - - - ⇒ return undefined })

definition *insert3* :: 'a::linorder ⇒ 'a rbt1 ⇒ 'a rbt1 **where**
insert3 a t = val (insert3-tm a t)

lemma *insert3-def2*[simp]: *insert3 x (t,n) =*
(let *t' = ins3 n 0 x t in*

case t' of
Same2 $\Rightarrow (t, n) \mid$
Bal2 t' $\Rightarrow (t', n+1)$)
 <proof>

definition *T-insert3* :: 'a::linorder \Rightarrow 'a rbt1 \Rightarrow nat **where**
T-insert3 a t = time (insert3-tm a t)

lemma *T-insert3-def2*: *T-insert3* x (t, n) = *T-ins3* n 0 x t + 1
 <proof>

2.3.2 Equivalence Proofs

lemma *ins-ins2*:
shows *ins2* n d x t = *Same2* \Longrightarrow *ins* n d x t = *Same*
and *ins2* n d x t = *Bal2* t' \Longrightarrow *ins* n d x t = *Bal* t'
and *ins2* n d x t = *Unbal2* t' n' h'
 \Longrightarrow *ins* n d x t = *Unbal* t' \wedge n' = size t' \wedge h' = height t'
 <proof>

lemma *ins2-ins*:
shows *ins* n d x t = *Same* \Longrightarrow *ins2* n d x t = *Same2*
and *ins* n d x t = *Bal* t' \Longrightarrow *ins2* n d x t = *Bal2* t'
and *ins* n d x t = *Unbal* t'
 \Longrightarrow *ins2* n d x t = *Unbal2* t' (size t') (height t')
 <proof>

corollary *ins2-iff-ins*:
shows *ins2* n d x t = *Same2* \longleftrightarrow *ins* n d x t = *Same*
and *ins2* n d x t = *Bal2* t' \longleftrightarrow *ins* n d x t = *Bal* t'
and *ins2* n d x t = *Unbal2* t' n' h' \longleftrightarrow
ins n d x t = *Unbal* t' \wedge n' = size t' \wedge h' = height t'
 <proof>

lemma *ins3-ins2*:
bal-i n (height t + d) \Longrightarrow *ins3* n d x t = *ins2* n d x t
 <proof>

lemma *insert2-insert*:
insert2 x (t, size t) = (t', n') \longleftrightarrow t' = *insert* x t \wedge n' = size t'
 <proof>

lemma *insert3-insert2*:
bal-i n (height t) \Longrightarrow *insert3* x (t, n) = *insert2* x (t, n)
 <proof>

2.3.3 Amortized Complexity

fun Φ :: 'a tree \Rightarrow real **where**
 Φ Leaf = 0 |

$$\Phi (\text{Node } l \ x \ r) = 6 * e * \text{imbal} (\text{Node } l \ x \ r) + \Phi \ l + \Phi \ r$$

lemma Φ -nn: $\Phi \ t \geq 0$
 <proof>

lemma Φ -sum-mset: $\Phi \ t = (\sum s \in \# \text{ subtrees-mset } t. 6 * e * \text{imbal } s)$
 <proof>

lemma Φ -wbalanced: **assumes** *wbalanced* *t* **shows** $\Phi \ t = 0$
 <proof>

lemma *imbal-ins-Bal*: *ins* *n* *d* *x* *t* = *Bal* *t'* \implies
 $\text{real}(\text{imbal} (\text{node } tw \ t' \ y \ s)) - \text{imbal} (\text{node } tw \ t \ y \ s) \leq 1$
 <proof>

lemma *imbal-ins-Unbal*: *ins* *n* *d* *x* *t* = *Unbal* *t'* \implies
 $\text{real}(\text{imbal} (\text{node } tw \ t' \ y \ s)) - \text{imbal} (\text{node } tw \ t \ y \ s) \leq 1$
 <proof>

lemma *T-ins3-Same*:
 $\text{ins3 } n \ d \ x \ t = \text{Same2} \implies T\text{-ins3 } n \ d \ x \ t \leq 2 * \text{height } t + 1$
 <proof>

lemma *T-ins3-Unbal*:
 $\llbracket \text{ins3 } n \ d \ x \ t = \text{Unbal2 } t' \ n' \ h'; \text{ bal-i } n \ (\text{height } t + d) \rrbracket \implies$
 $T\text{-ins3 } n \ d \ x \ t \leq 2 * \text{size } t + 1 + \text{height } t$
 <proof>

lemma *Phi-diff-Unbal*:
 $\llbracket \text{ins3 } n \ d \ x \ t = \text{Unbal2 } t' \ n' \ h'; \text{ bal-i } n \ (\text{height } t + d) \rrbracket \implies$
 $\Phi \ t' - \Phi \ t \leq 6 * e * \text{height } t$
 <proof>

lemma *amor-Unbal*:
 $\llbracket \text{ins3 } n \ d \ x \ t = \text{Unbal2 } t' \ n' \ h'; \text{ bal-i } n \ (\text{height } t + d) \rrbracket \implies$
 $T\text{-ins3 } n \ d \ x \ t + \Phi \ t' - \Phi \ t \leq 2 * \text{size1 } t + (6 * e + 1) * \text{height } t$
 <proof>

lemma *T-ins3-Bal*:
 $\llbracket \text{ins3 } n \ d \ x \ t = \text{Bal2 } t'; \text{ bal-i } n \ (\text{height } t + d) \rrbracket$
 $\implies T\text{-ins3 } n \ d \ x \ t + \Phi \ t' - \Phi \ t \leq (6 * e + 2) * (\text{height } t + 1)$
 <proof>

lemma *T-insert3-amor*: **assumes** $n = \text{size } t \text{ bal-i } (\text{size } t) \ (\text{height } t)$
 $\text{insert3 } a \ (t, n) = (t', n')$
shows $T\text{-insert3 } a \ (t, n) + \Phi \ t' - \Phi \ t \leq (6 * e + 2) * (\text{height } t + 1) + 1$
 <proof>

end

The insert-only version is shown to have the desired logarithmic amortized complexity. First it is shown to be linear in the height of the tree.

locale *RB*T*i2-Amor* = *RB*T*i2*
begin

fun *next* :: 'a ⇒ 'a *rbt1* ⇒ 'a *rbt1* **where**
next *x tn* = *insert3* *x tn*

fun *t_s* :: 'a ⇒ 'a *rbt1* ⇒ *real* **where**
t_s *x tn* = *T-insert3* *x tn*

interpretation *I-RB*T*i2-Amor*: *Amortized*

where *init* = (*Leaf*, 0)

and *next* = *next*

and *inv* = λ(*t*, *n*). *n* = *size* *t* ∧ *bal-i* (*size* *t*) (*height* *t*)

and *T* = *t_s* **and** Φ = λ(*t*, *n*). Φ *t*

and *U* = λ*x* (*t*, -). (6 * *e* + 2) * (*height* *t* + 1) + 1

⟨*proof*⟩

end

Now it is shown that a certain instantiation of *bal-i* that guarantees logarithmic height satisfies the assumptions of locale *RB*T*i2*.

interpretation *I-RB*T*i2*: *RB*T*i2*

where *bal-i* = λ*n h*. *h* ≤ *ceiling*(*c* * *log* 2 (*n* + 1))

and *e* = 2 *powr* (1/*c*) / (2 - 2 *powr* (1/*c*))

⟨*proof*⟩

2.4 Naive implementation (with delete)

axiomatization *cd* :: *real* **where**

cd0: *cd* > 0

definition *bal-d* :: *nat* ⇒ *nat* ⇒ *bool* **where**

bal-d *n dl* = (*dl* < *cd* * (*n* + 1))

lemma *bal-d0*: *bal-d* *n 0*

⟨*proof*⟩

lemma *mono-bal-d*: [*bal-d* *n dl*; *n* ≤ *n'*] ⇒ *bal-d* *n' dl*

⟨*proof*⟩

locale *RB*T*i1* = *RB*T*i2*

begin

2.4.1 Functions

fun *insert-d* :: 'a::*linorder* ⇒ 'a *rbt1* ⇒ 'a *rbt1* **where**

insert-d *x (t, dl)* =

```

(case ins (size t + dl) 0 x t of
  Same => t |
  Bal t' => t', dl)

```

definition *up-d* :: 'a => 'a tree => bool => 'a tree option => 'a tree option **where**
up-d x sib twist u =
 (case u of
 None => None |
 Some t => Some(node twist t x sib))

declare *up-d-def*[simp]

fun *del-tm* :: 'a::linorder => 'a tree => 'a tree option tm **where**
del-tm x Leaf = 1 return None |
del-tm x (Node l y r) = 1
 (case cmp x y of
 LT => do { l' ← *del-tm* x l; return (*up-d* y r False l') } |
 EQ => if r = Leaf then return (Some l)
 else do { (a',r') ← *split-min-tm* r;
 return (Some(Node l a' r')) } |
 GT => do { r' ← *del-tm* x r; return (*up-d* y l True r') }

definition *del* :: 'a::linorder => 'a tree => 'a tree option **where**
del x t = val(*del-tm* x t)

lemma *del-Leaf*[simp]: *del* x Leaf = None
 <proof>

lemma *del-Node*[simp]: *del* x (Node l y r) =
 (case cmp x y of
 LT => let l' = *del* x l in *up-d* y r False l' |
 EQ => if r = Leaf then Some l
 else let (a',r') = *split-min* r in Some(Node l a' r') |
 GT => let r' = *del* x r in *up-d* y l True r')
 <proof>

definition *T-del* :: 'a::linorder => 'a tree => nat **where**
T-del x t = time(*del-tm* x t)

lemma *T-del-Leaf*[simp]: *T-del* x Leaf = 1
 <proof>

lemma *T-del-Node*[simp]: *T-del* x (Node l y r) =
 (case cmp x y of
 LT => *T-del* x l + 1 |
 EQ => if r = Leaf then 1 else *T-split-min* r + 1 |
 GT => *T-del* x r + 1)
 <proof>

```

fun delete :: 'a::linorder  $\Rightarrow$  'a rbt1  $\Rightarrow$  'a rbt1 where
delete x (t,dl) =
  (case del x t of
    None  $\Rightarrow$  (t,dl) |
    Some t'  $\Rightarrow$ 
      if bal-d (size t') (dl+1) then (t',dl+1) else (balance-tree t', 0))

```

```

declare delete.simps [simp del]

```

2.4.2 Functional Correctness

lemma size-insert-d: $insert-d\ x\ (t,dl) = (t',dl') \implies size\ t \leq size\ t'$
 <proof>

lemma inorder-insert-d: $insert-d\ x\ (t,dl) = (t',dl') \implies sorted(inorder\ t) \implies inorder\ t' = ins-list\ x\ (inorder\ t)$
 <proof>

lemma bal-i-insert-d: **assumes** $insert-d\ x\ (t,dl) = (t',dl')$ **bal-i** (size t + dl) (height t)
shows **bal-i** (size t' + dl) (height t')
 <proof>

lemma inorder-del:
 $sorted(inorder\ t) \implies$
 $inorder(case\ del\ x\ t\ of\ None\ \Rightarrow\ t\ |\ Some\ t'\ \Rightarrow\ t') = del-list\ x\ (inorder\ t)$
 <proof>

lemma inorder-delete:
 $\llbracket delete\ x\ (t,dl) = (t',dl');\ sorted(inorder\ t) \rrbracket \implies$
 $inorder\ t' = del-list\ x\ (inorder\ t)$
 <proof>

lemma size-split-min:
 $\llbracket split-min\ t = (a,t');\ t \neq Leaf \rrbracket \implies size\ t' = size\ t - 1$
 <proof>

lemma height-split-min:
 $\llbracket split-min\ t = (a,t');\ t \neq Leaf \rrbracket \implies height\ t' \leq height\ t$
 <proof>

lemma size-del: $del\ x\ t = Some\ t' \implies size\ t' = size\ t - 1$
 <proof>

lemma height-del: $del\ x\ t = Some\ t' \implies height\ t' \leq height\ t$
 <proof>

lemma bal-i-delete:

assumes *bal-i* (size $t + dl$) (height t) *delete* $x (t, dl) = (t', dl')$
shows *bal-i* (size $t' + dl'$) (height t')
 $\langle proof \rangle$

lemma *bal-d-delete*:

$\llbracket \textit{bal-d}$ (size t) dl ; *delete* $x (t, dl) = (t', dl') \rrbracket$
 $\implies \textit{bal-d}$ (size t') dl'
 $\langle proof \rangle$

Full functional correctness of the naive implementation:

interpretation *Set-by-Ordered*

where *empty* = (*Leaf*, 0) **and** *isin* = $\lambda(t, n). \textit{isin } t$
and *insert* = *insert-d* **and** *delete* = *delete*
and *inorder* = $\lambda(t, n). \textit{inorder } t$ **and** *inv* = $\lambda-. \textit{True}$
 $\langle proof \rangle$

end

interpretation *I-RBTid1*: *RBTid1*

where *bal-i* = $\lambda n h. h \leq \log 2 (\textit{real}(n + 1)) + 1$ $\langle proof \rangle$

2.5 Efficient Implementation (with delete)

type-synonym *'a rbt2* = *'a tree* * *nat* * *nat*

locale *RBTid2* = *RBTi2* + *RBTid1*

begin

2.5.1 Functions

fun *insert2-d* :: *'a::linorder* \Rightarrow *'a rbt2* \Rightarrow *'a rbt2* **where**

insert2-d $x (t, n, dl) =$
 (case *ins2* ($n+dl$) 0 $x t$ of
 Same2 $\Rightarrow (t, n, dl)$ |
 Bal2 $t' \Rightarrow (t', n+1, dl)$)

fun *insert3-d-tm* :: *'a::linorder* \Rightarrow *'a rbt2* \Rightarrow *'a rbt2 tm* **where**

insert3-d-tm $x (t, n, dl) = 1$
 do { $t' \leftarrow \textit{ins3-tm}$ ($n+dl$) 0 $x t$;
 case t' of
 Same2 $\Rightarrow \textit{return}$ (t, n, dl) |
 Bal2 $t' \Rightarrow \textit{return}$ ($t', n+1, dl$) |
 Unbal2 - - $\Rightarrow \textit{return undefined}$ }

definition *insert3-d* :: *'a::linorder* \Rightarrow *'a rbt2* \Rightarrow *'a rbt2* **where**

insert3-d $a t = \textit{val}$ (*insert3-d-tm* $a t$)

lemma *insert3-d-def2*[*simp, code*]: *insert3-d* $x (t, n, dl) =$

$(let\ t' = ins3\ (n+dl)\ 0\ x\ t\ in$
 $\text{case } t' \text{ of}$
 $\text{Same2} \Rightarrow (t, n, dl) \mid$
 $\text{Bal2 } t' \Rightarrow (t', n+1, dl) \mid$
 $\text{Unbal2} \dots \Rightarrow \text{undefined})$
 $\langle \text{proof} \rangle$

definition $T\text{-insert3-d} :: 'a::\text{linorder} \Rightarrow 'a\ \text{rbt2} \Rightarrow \text{nat}$ **where**
 $T\text{-insert3-d } x\ t = \text{time}(\text{insert3-d-tm } x\ t)$

lemma $T\text{-insert3-d-def2}[\text{simp}]$:
 $T\text{-insert3-d } x\ (t, n, dl) = (T\text{-ins3 } (n+dl)\ 0\ x\ t + 1)$
 $\langle \text{proof} \rangle$

fun $\text{delete2-tm} :: 'a::\text{linorder} \Rightarrow 'a\ \text{rbt2} \Rightarrow 'a\ \text{rbt2}\ \text{tm}$ **where**
 $\text{delete2-tm } x\ (t, n, dl) = 1$
 $\text{do } \{ t' \leftarrow \text{del-tm } x\ t;$
 $\text{case } t' \text{ of}$
 $\text{None} \Rightarrow \text{return } (t, n, dl) \mid$
 $\text{Some } t' \Rightarrow$
 $(let\ n' = n-1; dl' = dl + 1$
 $\text{in if } \text{bal-d } n' dl' \text{ then return } (t', n', dl')$
 $\text{else do } \{ t'' \leftarrow \text{bal-tree-tm } n' t';$
 $\text{return } (t'', n', 0)\})\}$

definition $\text{delete2} :: 'a::\text{linorder} \Rightarrow 'a\ \text{rbt2} \Rightarrow 'a\ \text{rbt2}$ **where**
 $\text{delete2 } x\ t = \text{val}(\text{delete2-tm } x\ t)$

lemma delete2-def2 :
 $\text{delete2 } x\ (t, n, dl) =$
 $(let\ t' = \text{del } x\ t\ in$
 $\text{case } t' \text{ of}$
 $\text{None} \Rightarrow (t, n, dl) \mid$
 $\text{Some } t' \Rightarrow (let\ n' = n-1; dl' = dl + 1$
 $\text{in if } \text{bal-d } n' dl' \text{ then } (t', n', dl')$
 $\text{else let } t'' = \text{bal-tree } n' t' \text{ in } (t'', n', 0)))$
 $\langle \text{proof} \rangle$

definition $T\text{-delete2} :: 'a::\text{linorder} \Rightarrow 'a\ \text{rbt2} \Rightarrow \text{nat}$ **where**
 $T\text{-delete2 } x\ t = \text{time}(\text{delete2-tm } x\ t)$

lemma $T\text{-delete2-def2}$:
 $T\text{-delete2 } x\ (t, n, dl) = (T\text{-del } x\ t +$
 $(\text{case } \text{del } x\ t \text{ of}$
 $\text{None} \Rightarrow 1 \mid$
 $\text{Some } t' \Rightarrow (let\ n' = n-1; dl' = dl + 1$
 $\text{in if } \text{bal-d } n' dl' \text{ then } 1 \text{ else } T\text{-bal-tree } n' t' + 1)))$
 $\langle \text{proof} \rangle$

2.5.2 Equivalence proofs

lemma *insert2-insert-d*:

$insert2-d\ x\ (t, size\ t, dl) = (t', n', dl') \longleftrightarrow$
 $(t', dl') = insert-d\ x\ (t, dl) \wedge n' = size\ t'$
 ⟨proof⟩

lemma *insert3-insert2-d*:

$bal-i\ (n+dl)\ (height\ t) \implies insert3-d\ x\ (t, n, dl) = insert2-d\ x\ (t, n, dl)$
 ⟨proof⟩

lemma *delete2-delete*:

$delete2\ x\ (t, size\ t, dl) = (t', n', dl') \longleftrightarrow$
 $(t', dl') = delete\ x\ (t, dl) \wedge n' = size\ t'$
 ⟨proof⟩

2.5.3 Amortized complexity

fun $\Phi_d :: 'a\ rbt2 \Rightarrow real$ **where**

$\Phi_d\ (t, n, dl) = \Phi\ t + 4 * dl / cd$

lemma Φ_d -case: $\Phi_d\ tndl = (case\ tndl\ of\ (t, n, dl) \Rightarrow \Phi\ t + 4 * dl / cd)$
 ⟨proof⟩

lemma *imbal-diff-decr*:

$size\ r' = size\ r - 1 \implies$
 $real(imbal\ (Node\ l\ x'\ r')) - imbal\ (Node\ l\ x\ r) \leq 1$
 ⟨proof⟩

lemma *tinsert-d-amor*:

assumes $n = size\ t\ insert-d\ a\ (t, dl) = (t', dl')\ bal-i\ (size\ t + dl)\ (height\ t)$
shows $T-insert3-d\ a\ (t, n, dl) + \Phi\ t' - \Phi\ t \leq (6 * e + 2) * (height\ t + 1) + 1$
 ⟨proof⟩

lemma *T-split-min-ub*:

$t \neq Leaf \implies T-split-min\ t \leq height\ t + 1$
 ⟨proof⟩

lemma *T-del-ub*:

$T-del\ x\ t \leq height\ t + 1$
 ⟨proof⟩

lemma *imbal-split-min*:

$split-min\ t = (x, t') \implies t \neq Leaf \implies real(imbal\ t') - imbal\ t \leq 1$
 ⟨proof⟩

lemma *imbal-del-Some*:

$del\ x\ t = Some\ t' \implies real(imbal\ t') - imbal\ t \leq 1$
 ⟨proof⟩

lemma *Phi-diff-split-min*:

$split\text{-}min\ t = (x, t') \implies t \neq Leaf \implies \Phi\ t' - \Phi\ t \leq 6 * e * height\ t$
<proof>

lemma *Phi-diff-del-Some*:

$del\ x\ t = Some\ t' \implies \Phi\ t' - \Phi\ t \leq 6 * e * height\ t$
<proof>

lemma *amor-del-Some*:

$del\ x\ t = Some\ t' \implies$
 $T\text{-}del\ x\ t + \Phi\ t' - \Phi\ t \leq (6 * e + 1) * height\ t + 1$
<proof>

lemma *cd1*: $1/cd > 0$

<proof>

lemma *T-delete-amor*: **assumes** $n = size\ t$

shows $T\text{-}delete2\ x\ (t, n, dl) + \Phi_d\ (delete2\ x\ (t, n, dl)) - \Phi_d\ (t, n, dl)$
 $\leq (6 * e + 1) * height\ t + 4/cd + 4$

<proof>

datatype (*plugins del: lifting*) 'b ops = Insert 'b | Delete 'b

fun *nxt* :: 'a ops \Rightarrow 'a rbt2 \Rightarrow 'a rbt2 **where**

nxt (Insert x) t = insert3-d x t |

nxt (Delete x) t = delete2 x t

fun *t_s* :: 'a ops \Rightarrow 'a rbt2 \Rightarrow real **where**

t_s (Insert x) t = T-insert3-d x t |

t_s (Delete x) t = T-delete2 x t

interpretation *RBTid2-Amor*: Amortized

where *init* = (Leaf, 0, 0)

and *nxt* = *nxt*

and *inv* = $\lambda(t, n, dl). n = size\ t \wedge$

$bal\text{-}i\ (size\ t + dl)\ (height\ t) \wedge bal\text{-}d\ (size\ t)\ dl$

and *T* = *t_s* **and** $\Phi = \Phi_d$

and *U* = $\lambda f\ (t, -). case\ f\ of$

$Insert\ - \Rightarrow (6 * e + 2) * (height\ t + 1) + 1 \ |$

$Delete\ - \Rightarrow (6 * e + 1) * height\ t + 4/cd + 4$

<proof>

end

axiomatization *b* :: real **where**

b0: $b > 0$

axiomatization where

cd-le-log: $cd \leq 2 \text{ powr } (b/c) - 1$

This axiom is only used to prove that the height remains logarithmic in the size.

interpretation *I-RBTid2*: *RBTid2*

where *bal-i* = $\lambda n h. h \leq \text{ceiling}(c * \log 2 (n+1))$

and *e* = $2 \text{ powr } (1/c) / (2 - 2 \text{ powr } (1/c))$

<proof>

Finally we show that under the above interpretation of *bal-i* the height is logarithmic:

definition *bal-i* :: *nat* \Rightarrow *nat* \Rightarrow *bool* **where**

bal-i n h = $(h \leq \text{ceiling}(c * \log 2 (n+1)))$

lemma assumes *bal-i* (*size t + dl*) (*height t*) *bal-d* (*size t*) *dl*

shows *height t* $\leq \text{ceiling}(c * \log 2 (\text{size1 } t) + b)$

<proof>

end

3 Tabulating the Balanced Predicate

theory *Root-Balanced-Tree-Tab*

imports

Root-Balanced-Tree

HOL-Decision-Procs.Approximation

HOL-Library.IArray

begin

locale *Min-tab* =

fixes *p* :: *nat* \Rightarrow *nat* \Rightarrow *bool*

fixes *tab* :: *nat list*

assumes *mono-p*: $n \leq n' \implies p n h \implies p n' h$

assumes *p*: $\exists n. p n h$

assumes *tab-LEAST*: $h < \text{length } \text{tab} \implies \text{tab}!h = (\text{LEAST } n. p n h)$

begin

lemma *tab-correct*: $h < \text{length } \text{tab} \implies p n h = (n \geq \text{tab} ! h)$

<proof>

end

definition *bal-tab* :: *nat list* **where**

bal-tab = $[0, 1, 1, 2, 4, 6, 10, 16, 25, 40, 64, 101, 161, 256, 406, 645, 1024, 1625, 2580, 4096, 6501, 10321, 16384, 26007, 41285, 65536, 104031, 165140, 262144, 416127, 660561, 1048576, 1664510, 2642245, 4194304, 6658042, 10568983, 16777216, 26632170, 42275935, 67108864, 106528681, 169103740, 268435456,$

426114725, 676414963, 1073741824, 1704458900, 2705659852, 4294967296 //6847897608]

axiomatization where *c-def*: $c = 3/2$

fun *is-floor* :: *nat* \Rightarrow *nat* \Rightarrow *bool* **where**
is-floor *n* *h* = (let *m* = floor((2::real) powr ((real(*h*)-1)/*c*)) in $n \leq m \wedge m \leq n$)

Note that $n \leq m \wedge m \leq n$ avoids the technical restriction of the *approximation* method which does not support =, even on integers.

lemma *bal-tab-correct*:

$\forall i < \text{length } \text{bal-tab}. \text{is-floor } (\text{bal-tab}!i) \ i$
(*proof*)

lemma *ceiling-least-real*: ceiling(*r*::real) = (LEAST *i*. $r \leq i$)

(*proof*)

lemma *floor-greatest-real*: floor(*r*::real) = (GREATEST *i*. $i \leq r$)

(*proof*)

lemma *LEAST-eq-floor*:

(LEAST *n*. $\text{int } h \leq \lceil c * \log 2 (\text{real } n + 1) \rceil$) = floor((2::real) powr ((real(*h*)-1)/*c*))
(*proof*)

interpretation *Min-tab*

where *p* = *bal-i* **and** *tab* = *bal-tab*

(*proof*)

Now we replace the list by an immutable array:

definition *bal-array* :: *nat* *iarray* **where**

bal-array = *IArray* *bal-tab*

A trick for code generation: how to get rid of the precondition:

lemma *bal-i-code*:

bal-i *n* *h* =
(if $h < \text{IArray.length } \text{bal-array}$ then *IArray.sub* *bal-array* *h* $\leq n$ else *bal-i* *n* *h*)
(*proof*)

end

References

- [1] A. Andersson. Improving partial rebuilding by using simple balance criteria. In F. Dehne, J.-R. Sack, and N. Santoro, editors, *Algorithms and Data Structures (WADS '89)*, volume 382 of *LNCS*, pages 393–402. Springer, 1989.

- [2] A. Andersson. General balanced trees. *J. Algorithms*, 30(1):1–18, 1999.
- [3] T. Nipkow. Verified root-balanced trees. In B.-Y. E. Chang, editor, *Programming Languages and Systems, APLAS 2017*, volume ? of *LNCS*. Springer, 2017. <http://www.in.tum.de/~nipkow/pubs/aplas17.html>.