Weight-Balanced Trees

Tobias Nipkow and Stefan Dirix

March 17, 2025

Abstract

This theory provides a verified implementation of weight-balanced trees following the work of Hirai and Yamamoto [4] who proved that all parameters in a certain range are valid, i.e. guarantee that insertion and deletion preserve weight-balance. Instead of a general theorem we provide parameterized proofs of preservation of the invariant that work for many (all?) valid parameters.

1 Introduction

Weight-balanced trees (*WB* trees) are a class of binary search trees of logarithmic height. They were invented by Nievergelt and Reingold [5, 6] who called them *trees of bounded balance*. They are parametrized by a constant. Parameters are called *valid* if they guarantee that insertion and deletion preserve the WB invariant. Blum and Mehlhorn [3] later discovered that there is a flaw in Nievergelt and Reingold's analysis of valid parameters and gave a detailed correctness proof for a modified range of parameters. Adams [1, 2] considered a slightly modified version of WB trees and analyzed which parameters are valid. The Haskell libraries `Data.Set` and `Data.Map` are based on Adams' papers but it was found that the implementation did not preserve the invariant. This motivated Hirai and Yamamoto [4] to verify the valid parameter range for the original definition of WB tree formally in Coq. They also showed that Adams' analysis is flawed by giving a counterexample to Adams' claimed range of valid parameters. Straka [8] analyzes valid parameters for Adam's variant. Yet another variant of WB trees was considered by Roura [7].

2 Weight-Balanced Trees Have Logarithmic Height, and More

theory *Weight_Balanced_Trees_log*
imports
Complex_Main

$HOL{-}Library.Tree$
begin

lemmas $neq0\_if$ = $less\_imp\_neq$ $dual\_order.strict\_implies\_not\_eq$

2.1 Logarithmic Height

The locale below is parameterized wrt to Δ. The original definition of weight-balanced trees [5, 6] uses α. The constants α and Δ are interdefinable. Below we start from Δ but derive α-versions of theorems as well.

locale $WBT0$ =
fixes $\Delta :: real$
begin

fun $balanced1 :: 'a\ tree \Rightarrow 'a\ tree \Rightarrow bool$ **where**
$balanced1\ t1\ t2 = (size1\ t1 \leq \Delta * size1\ t2)$

fun $wbt :: 'a\ tree \Rightarrow bool$ **where**
$wbt\ Leaf = True\ |$
$wbt\ (Node\ l\ \_\ r) = (balanced1\ l\ r \wedge balanced1\ r\ l \wedge wbt\ l \wedge wbt\ r)$

end

locale $WBT1 = WBT0\ +$
assumes $Delta$: $\Delta \geq 1$
begin

definition $\alpha :: real$ **where**
$\alpha = 1/(\Delta+1)$

lemma $Delta\_def$: $\Delta = 1/\alpha - 1$
$\langle proof \rangle$

lemma shows $alpha\_pos$: $0 < \alpha$ **and** $alpha\_ub$: $\alpha \leq 1/2$
$\langle proof \rangle$

lemma wbt_Node_alpha: $wbt\ (Node\ l\ x\ r) =$
$((let\ q = size1\ l\ /\ (size1\ l + size1\ r)$
$in\ \alpha \leq q \wedge q \leq 1 - \alpha) \wedge$
$wbt\ l \wedge wbt\ r)$
$\langle proof \rangle$

lemma $height\_size1\_Delta$:
$wbt\ t \Longrightarrow (1 + 1/\Delta)\ \hat{}\,(height\ t) \leq size1\ t$
$\langle proof \rangle$

lemma $height\_size1\_alpha$:
$wbt\ t \Longrightarrow (1/(1-\alpha))\ \hat{}\,(height\ t) \leq size1\ t$

⟨*proof*⟩

lemma *height_size1_log_Delta*: **assumes** *wbt t*
shows $height\ t \le log\ 2\ (size1\ t)\ /\ log\ 2\ (1 + 1/\Delta)$
⟨*proof*⟩

lemma *height_size1_log_alpha*: **assumes** *wbt t*
shows $height\ t \le log\ 2\ (size1\ t)\ /\ log\ 2\ (1/(1-\alpha))$
⟨*proof*⟩

end

2.2 Every $1 \le \Delta < 2$ Yields Exactly the Complete Trees

declare *WBT0.wbt.simps* [*simp*] *WBT0.balanced1.simps* [*simp*]

lemma *wbt1_if_complete*: **assumes** $1 \le \Delta$ **shows** $complete\ t \Longrightarrow WBT0.wbt\ \Delta\ t$
⟨*proof*⟩

lemma *complete_if_wbt2*: **assumes** $\Delta < 2$ **shows** $WBT0.wbt\ \Delta\ t \Longrightarrow complete\ t$
⟨*proof*⟩

end

3 Weight Balanced Tree Implementation of Sets

This theory follows Hirai and Yamamoto but we do not prove their general theorem. Instead we provide a short parameterized theory that, when interpreted with valid parameters, will prove perservation of the invariant for these parameters.

theory *Weight_Balanced_Trees*
imports
 HOL−Data_Structures.Isin2
begin

lemma *neq_Leaf2_iff*: $t \ne Leaf \longleftrightarrow (\exists\, l\ a\ n\ r.\ t = Node\ l\ (a,n)\ r)$
⟨*proof*⟩

type-synonym *'a wbt* = (*'a* ∗ *nat*) *tree*

fun *size_wbt* :: *'a wbt* ⇒ *nat* **where**
size_wbt Leaf = 0 |
size_wbt (*Node* _ (_, *n*) _) = *n*

Smart constructor:

fun *N* :: *'a wbt* ⇒ *'a* ⇒ *'a wbt* ⇒ *'a wbt* **where**
N l a r = *Node l* (*a*, *size_wbt l* + *size_wbt r* + 1) *r*

Basic Rotations:

fun $rot1L :: 'a\ wbt \Rightarrow 'a \Rightarrow 'a\ wbt \Rightarrow 'a \Rightarrow 'a\ wbt \Rightarrow 'a\ wbt$ **where**
$rot1L\ A\ a\ B\ b\ C = N\ (N\ A\ a\ B)\ b\ C$

fun $rot1R :: 'a\ wbt \Rightarrow 'a \Rightarrow 'a\ wbt \Rightarrow 'a \Rightarrow 'a\ wbt \Rightarrow 'a\ wbt$ **where**
$rot1R\ A\ a\ B\ b\ C = N\ A\ a\ (N\ B\ b\ C)$

fun $rot2 :: 'a\ wbt \Rightarrow 'a \Rightarrow 'a\ wbt \Rightarrow 'a \Rightarrow 'a\ wbt \Rightarrow 'a\ wbt$ **where**
$rot2\ A\ a\ (Node\ B1\ (b,\_)\ B2)\ c\ C = N\ (N\ A\ a\ B1)\ b\ (N\ B2\ c\ C)$

3.1 WB trees

Parameters:

Δ determines when a tree needs to be rebalanced

Γ determines whether it needs to be single or double rotation.

We represent rational numbers as pairs: $\Delta = \Delta1/\Delta2$ and $\Gamma = \Gamma1/\Gamma2$.

Hirai and Yamamoto [4] proved that under the following constraints insertion and deletion preserve the WB invariant, i.e. Δ and Γ are *valid*:

definition *valid_params* :: $nat \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow bool$ **where**
valid_params $\Delta1\ \Delta2\ \Gamma1\ \Gamma2$ = (
$\Delta1 * 2 < \Delta2 * 9$ — right: $\Delta < 4.5\ \wedge$
$\Gamma1 * \Delta2 + \Gamma2 * \Delta2 \le \Gamma2 * \Delta1$ — left: $\Gamma + 1 \le \Delta\ \wedge$
$\Gamma1 * \Delta1 \ge \Gamma2 * (\Delta1 + \Delta2)$ — lower: $\Gamma \ge (\Delta + 1) / \Delta\ \wedge$
— upper:
$(5{*}\Delta2 \le 2{*}\Delta1 \wedge 1{*}\Delta1 < 3{*}\Delta2 \longrightarrow \Gamma1{*}2 \le \Gamma2{*}3)$
— $\Gamma \le 3/2$ if $2.5 \le \Delta < 3\ \wedge$
$(3{*}\Delta2 \le 1{*}\Delta1 \wedge 2{*}\Delta1 < 7{*}\Delta2 \longrightarrow \Gamma1{*}2 \le \Gamma2{*}4)$
— $\Gamma \le 4/2$ if $3 \le \Delta < 3.5\ \wedge$
$(7{*}\Delta2 \le 2{*}\Delta1 \wedge 1{*}\Delta1 < 4{*}\Delta2 \longrightarrow \Gamma1{*}3 \le \Gamma2{*}4)$
— $\Gamma \le 4/3$ when $3.5 \le \Delta < 4\ \wedge$
$(4{*}\Delta2 \le 1{*}\Delta1 \wedge 2{*}\Delta1 < 9{*}\Delta2 \longrightarrow \Gamma1{*}3 \le \Gamma2{*}5)$
— $\Gamma \le 5/3$ when $4 \le \Delta < 4.5$
)

We do not make use of these constraints and do not prove that they guarantee preservation of the invariant. Instead, we provide generic proofs of invariant preservation that work for many (all?) interpretations of locale *WBT* (below) with valid parameters. Further down we demonstrate this by interpreting *WBT* with a selection of valid parameters. [For some parameters, some *smt* proofs fail because *smt* on *nat*s fails although on non-negative *int*s it succeeds, i.e. the goal should be provable. This is a shortcoming of *smt* that is under investigation.]

Locale *WBT* comes with some minimal assumptions ($\Gamma1 > \Gamma2$ and $\Delta1 > \Delta2$) which follow from *valid_params* and from which we conclude some simple lemmas.

locale *WBT* =
fixes Δ1 Δ2 :: *nat* **and** Γ1 Γ2 :: *nat*
assumes *Delta_gr*1: Δ1 > Δ2 **and** *Gamma_gr*1: Γ1 > Γ2
begin

3.1.1 Balance Indicators

fun *balanced*1 :: *'a wbt* ⇒ *'a wbt* ⇒ *bool* **where**
*balanced*1 *t*1 *t*2 = (Δ1 ∗ (*size_wbt t*1 + 1) ≥ Δ2 ∗ (*size_wbt t*2 + 1))

The global weight-balanced tree invariant:

fun *wbt* :: *'a wbt* ⇒ *bool* **where**
wbt Leaf = *True*|
wbt (*Node l* (_, *n*) *r*) =
(*n* = *size l* + *size r* + 1 ∧ *balanced*1 *l r* ∧ *balanced*1 *r l* ∧ *wbt l* ∧ *wbt r*)

lemma *size_wbt_eq_size*[*simp*]: *wbt t* ⟹ *size_wbt t* = *size t*
⟨*proof*⟩

fun *single* :: *'a wbt* ⇒ *'a wbt* ⇒ *bool* **where**
*single t*1 *t*2 = (Γ1 ∗ (*size_wbt t*2 + 1) > Γ2 ∗ (*size_wbt t*1 + 1))

3.1.2 Code

fun *rotateL* :: *'a wbt* ⇒ *'a* ⇒ *'a wbt* ⇒ *'a wbt* **where**
rotateL A a (*Node B* (*b*, _) *C*) =
(*if single B C then rot*1*L A a B b C else rot*2 *A a B b C*)

fun *balanceL* :: *'a wbt* ⇒ *'a* ⇒ *'a wbt* ⇒ *'a wbt* **where**
balanceL l a r = (*if balanced*1 *l r then N l a r else rotateL l a r*)

fun *rotateR* :: *'a wbt* ⇒ *'a* ⇒ *'a wbt* ⇒ *'a wbt* **where**
rotateR (*Node A* (*a*, _) *B*) *b C* =
(*if single B A then rot*1*R A a B b C else rot*2 *A a B b C*)

fun *balanceR* :: *'a wbt* ⇒ *'a* ⇒ *'a wbt* ⇒ *'a wbt* **where**
balanceR l a r = (*if balanced*1 *r l then N l a r else rotateR l a r*)

fun *insert* :: *'a::linorder* ⇒ *'a wbt* ⇒ *'a wbt* **where**
insert x Leaf = *Node Leaf* (*x*, 1) *Leaf* |
insert x (*Node l* (*a*, *n*) *r*) =
(*case cmp x a of*
LT ⇒ *balanceR* (*insert x l*) *a r* |
GT ⇒ *balanceL l a* (*insert x r*) |
EQ ⇒ *Node l* (*a*, *n*) *r*)

fun *split_min* :: *'a wbt* ⇒ *'a* ∗ *'a wbt* **where**
split_min (*Node l* (*a*, _) *r*) =
(*if l* = *Leaf then* (*a*,*r*) *else let* (*x*,*l'*) = *split_min l in* (*x*, *balanceL l' a r*))

fun *del_max* :: *'a wbt* ⇒ *'a* ∗ *'a wbt* **where**
del_max (*Node l* (*a*, _) *r*) =
  (*if r* = *Leaf then* (*a*,*l*) *else let* (*x*,*r'*) = *del_max r in* (*x*, *balanceR l a r'*))

fun *combine* :: *'a wbt* ⇒ *'a wbt* ⇒ *'a wbt* **where**
combine Leaf Leaf = *Leaf*|
combine Leaf r = *r*|
combine l Leaf = *l*|
combine l r =
  (*if size l* > *size r then*
    let (*lMax*, *l'*) = *del_max l in balanceL l' lMax r*
  else
    let (*rMin*, *r'*) = *split_min r in balanceR l rMin r'*)

fun *delete* :: *'a*::*linorder* ⇒ *'a wbt* ⇒ *'a wbt* **where**
delete _ *Leaf* = *Leaf* |
delete x (*Node l* (*a*, _) *r*) =
 (*case cmp x a of*
   LT ⇒ *balanceL* (*delete x l*) *a r* |
   GT ⇒ *balanceR l a* (*delete x r*) |
   EQ ⇒ *combine l r*)

3.2 Functional Correctness Proofs

A WB tree must be of a certain structure if balanced1 and single are False.

lemma *not_Leaf_if_not_balanced1*:
 assumes ¬ *balanced1 l r*
 shows $r \neq Leaf$
⟨*proof*⟩

lemma *not_Leaf_if_not_single*:
 assumes ¬ *single l r*
 shows $l \neq Leaf$
⟨*proof*⟩

3.2.1 Inorder Properties

lemma *inorder_rot2*:
 $B \neq Leaf \Longrightarrow inorder(rot2\ A\ a\ B\ b\ C) = inorder\ A\ @\ a\ \#\ inorder\ B\ @\ b\ \#\ inorder\ C$
⟨*proof*⟩

lemma *inorder_rotateL*:
 $r \neq Leaf \Longrightarrow inorder(rotateL\ l\ a\ r) = inorder\ l\ @\ a\ \#\ inorder\ r$
⟨*proof*⟩

lemma *inorder_rotateR*:
 $l \neq Leaf \Longrightarrow inorder(rotateR\ l\ a\ r) = inorder\ l\ @\ a\ \#\ inorder\ r$
⟨*proof*⟩

lemma *inorder_insert*:
$sorted(inorder\ t) \Longrightarrow inorder(insert\ x\ t) = ins\_list\ x\ (inorder\ t)$
⟨*proof*⟩

lemma *split_minD*:
$split\_min\ t = (x,t') \Longrightarrow t \neq Leaf \Longrightarrow x\ \#\ inorder\ t' = inorder\ t$
⟨*proof*⟩

lemma *del_maxD*:
$del\_max\ t = (x,t') \Longrightarrow t \neq Leaf \Longrightarrow inorder\ t'\ @\ [x] = inorder\ t$
⟨*proof*⟩

lemma *inorder_combine*:
$inorder(combine\ l\ r) = inorder\ l\ @\ inorder\ r$
⟨*proof*⟩

lemma *inorder_delete*:
$sorted(inorder\ t) \Longrightarrow inorder(delete\ x\ t) = del\_list\ x\ (inorder\ t)$
⟨*proof*⟩

3.3 Size Lemmas

3.3.1 Insertion

lemma *size_rot2L*[*simp*]:
$B \neq Leaf \Longrightarrow size(rot2\ A\ a\ B\ b\ C) = size\ A + size\ B + size\ C + 2$
⟨*proof*⟩

lemma *size_rotateR*[*simp*]:
$l \neq Leaf \Longrightarrow size(rotateR\ l\ a\ r) = size\ l + size\ r + 1$
⟨*proof*⟩

lemma *size_rotateL*[*simp*]:
$r \neq Leaf \Longrightarrow size(rotateL\ l\ a\ r) = size\ l + size\ r + 1$
⟨*proof*⟩

lemma *size_length*: $size\ t = length\ (inorder\ t)$
⟨*proof*⟩

lemma *size_insert*: $size\ (insert\ x\ t) = (if\ isin\ t\ x\ then\ size\ t\ else\ Suc\ (size\ t))$
⟨*proof*⟩

3.3.2 Deletion

lemma *size_delete_if_isin*: $isin\ t\ x \Longrightarrow size\ t = Suc\ (size(delete\ x\ t))$
⟨*proof*⟩

lemma *delete_id_if_wbt_notin*: $wbt\ t \Longrightarrow \neg\ isin\ t\ x \Longrightarrow delete\ x\ t = t$
⟨*proof*⟩

lemma *size_split_min*: $t \neq Leaf \Longrightarrow size\ t = Suc\ (size\ (snd\ (split\_min\ t)))$
⟨*proof*⟩

lemma *size_del_max*: $t \neq Leaf \Longrightarrow size\ t = Suc(size(snd(del\_max\ t)))$
⟨*proof*⟩

3.4 Auxiliary Definitions

fun *balanced1_arith* :: *nat* ⇒ *nat* ⇒ *bool* **where**
$balanced1\_arith\ a\ b = (\Delta 1 * (a + 1) \geq \Delta 2 * (b + 1))$

fun *balanced2_arith* :: *nat* ⇒ *nat* ⇒ *bool* **where**
$balanced2\_arith\ a\ b = (balanced1\_arith\ a\ b \wedge balanced1\_arith\ b\ a)$

fun *singly_balanced_arith* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
$singly\_balanced\_arith\ x\ y\ w = (balanced2\_arith\ x\ y \wedge balanced2\_arith\ (x{+}y{+}1)\ w)$

fun *doubly_balanced_arith* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
$doubly\_balanced\_arith\ x\ y\ z\ w =$
$(balanced2\_arith\ x\ y \wedge balanced2\_arith\ z\ w \wedge balanced2\_arith\ (x{+}y{+}1)\ (z{+}w{+}1))$

end

3.5 Preservation of WB tree Invariant for Concrete Parameters

A number of sample interpretations with valid parameters:

interpretation *WBT* **where**
$\Delta 1 = 25$ **and** $\Delta 2 = 10$ **and** $\Gamma 1 = 14$ **and** $\Gamma 2 = 10$

⟨*proof*⟩

lemma *wbt_insert*:
$wbt\ t \Longrightarrow wbt\ (insert\ x\ t)$

⟨*proof*⟩

declare [[*smt_nat_as_int*]]

Show that invariant is preserved by deletion in the left/right subtree:

lemma *wbt_balanceL*:
assumes $wbt\ (Node\ l\ (a,\ n)\ r)\ wbt\ l'\ size\ l = size\ l' + 1$
shows $wbt\ (balanceL\ l'\ a'\ r)$
⟨*proof*⟩

lemma *wbt_balanceR*:
assumes $wbt\ (Node\ l\ (a,\ n)\ r)\ wbt\ r'\ size\ r = size\ r' + 1$
shows $wbt\ (balanceR\ l\ a'\ r')$
⟨*proof*⟩

lemma *wbt_split_min*: $t \neq Leaf \Longrightarrow wbt\ t \Longrightarrow wbt\ (snd\ (split\_min\ t))$
⟨*proof*⟩

lemma *wbt_del_max*: $t \neq Leaf \Longrightarrow wbt\ t \Longrightarrow wbt\ (snd\ (del\_max\ t))$
⟨*proof*⟩

lemma *wbt_delete*: $wbt\ t \Longrightarrow wbt\ (delete\ x\ t)$
⟨*proof*⟩

3.6 The final correctness proof

interpretation *S*: *Set_by_Ordered*
where *empty* = *Leaf* **and** *isin* = *isin* **and** *insert* = *insert* **and** *delete* = *delete*
and *inorder* = *inorder* **and** *inv* = *wbt*
⟨*proof*⟩

end

References

[1] S. Adams. Implementing sets efficiently in a functional language. Technical Report CSTR 92-10, Department of Electronics and Computer Science, University of Southampton, 1992.

[2] S. Adams. Efficient sets - A balancing act. *Journal of Functional Programming*, 3(4):553–561, 1993.

[3] N. Blum and K. Mehlhorn. On the average number of rebalancing operations in weight-balanced trees. *Theoretical Compututer Science*, 11:303–320, 1980.

[4] Y. Hirai and K. Yamamoto. Balancing weight-balanced trees. *Journal of Functional Programming*, 21(3):287–307, 2011.

[5] J. Nievergelt and E. M. Reingold. Binary search trees of bounded balance. In *Proc. 4th ACM Symposium on Theory of Computing*, STOC '72, pages 137–142. ACM, 1972.

[6] J. Nievergelt and E. M. Reingold. Binary search trees of bounded balance. *SIAM Journal on Computing*, 2(1):33–43, 1973.

[7] S. Roura. A new method for balancing binary search trees. In F. Orejas, P. Spirakis, and J. van Leeuwen, editors, *Automata, Languages and Programming, ICALP 2001*, volume 2076 of *LNCS*, pages 469–480. Springer, 2001.

[8] M. Straka. Adams' trees revisited. Correct and efficient implementation. In *Trends in Functional Programming*, volume 7193 of *LNCS*, pages 130–145. Springer, 2011.