Balancing weight-balanced trees

Article *in* Journal of Functional Programming · May 2011

DOI: 10.1017/S0956796811000104 · Source: DBLP

Yoichi Hirai
The University of Tokyo
8 PUBLICATIONS **173** CITATIONS

JFP **21** (3): 287–307, 2011. © Cambridge University Press 2011
doi:10.1017/S0956796811000104

Balancing weight-balanced trees

YOICHI HIRAI
The University of Tokyo, JSPS Research Fellow
(*e-mail:* `yh@lyon.is.s.u-tokyo.ac.jp`)

KAZUHIKO YAMAMOTO
IIJ Innovation Institute Inc.
(*e-mail:* `kazu@iij.ad.jp`)

Abstract

A weight-balanced tree (WBT) is a binary search tree, whose balance is based on the sizes of the subtrees in each node. Although purely functional implementations on a variant WBT algorithm are widely used in functional programming languages, many existing implementations do not maintain balance after deletion in some cases. The difficulty lies in choosing a valid pair of rotation parameters: one for standard balance and the other for choosing single or double rotation. This paper identifies the exact valid range of the rotation parameters for insertion and deletion in the original WBT algorithm where one and only one integer solution exists. Soundness of the range is proved using a proof assistant Coq. Completeness is proved using effective algorithms generating counterexample trees. For two specific parameter pairs, we also proved in Coq that set operations also maintain balance. Since the difference between the original WBT and the variant WBT is small, it is easy to change the existing buggy implementations based on the variant WBT to the certified original WBT with a rational solution.

1 Introduction

Weight-balanced trees (WBTs) (Nievergelt & Reingold, 1972) are binary search trees, which can be used to implement finite sets and finite maps (associative arrays). Although other balanced binary search trees, such as AVL trees (Adel'son-Vel'skii & Landis, 1962) and red–black trees (Guibas & Sedgewick, 1978), use the height of subtrees for balancing, the balance of WBTs is based on the sizes (number of elements) of the subtrees below each node. Its purely functional implementations are widely used in functional programming languages. In fact, fundamental modules `Data.Set` and `Data.Map` in Haskell (Marlow, 2010) and the `wttree.scm` library in MIT/GNU Scheme and `slib` are based on a variant of the WBT algorithm (Adams, 1993).

In order to ensure performance, the algorithm keeps the height of a tree logarithmic to its size by balancing the sizes of the subtrees in each node. In 2010, a bug report[1] confirmed that the `Data.Map` library broke the tree balance after

[1] `http://hackage.haskell.org/trac/ghc/ticket/4242`

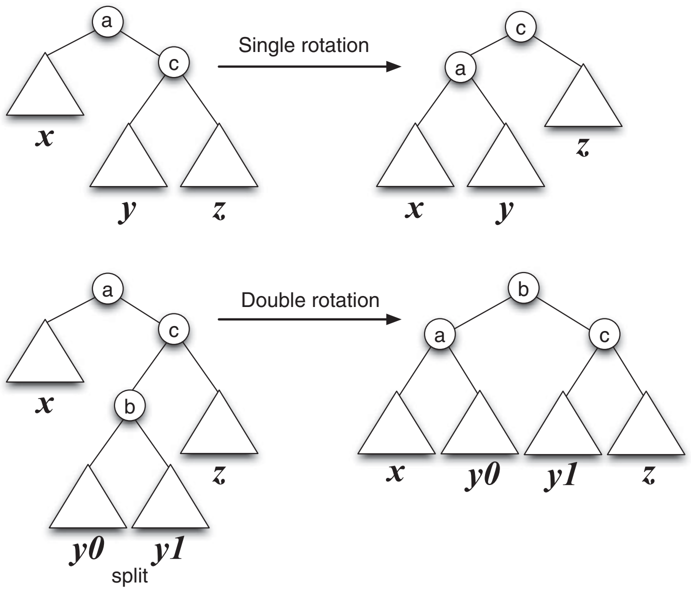

Fig. 1. A single left rotation and a double left rotation. a, b, and c are elements. x, y, $y0$, $y1$, and z are the size of each tree. If y is too large, a double rotation is chosen. Otherwise, a single rotation is used.

deletion.[2] We investigated the existing literature but failed to find a rigorous proof that both insertion and deletion preserve the balance of WBTs. Instead, we found that proving balance preservation requires checking several inequalities in 14 cases of program behaviors for five different parameter zones. We used a proof assistant Coq (Bertot & Casteran, 2004) in order to cope with this intensive case analysis.

To keep the balance of WBTs, there are two important parameters Δ and Γ: Δ decides whether any rotation is made at all and Γ chooses a single rotation or a double rotation (Figure 1). These parameters must ensure that a newly created tree is balanced after any insertion or deletion in a given balanced WBT. The original paper of WBT suggests $\langle \Delta, \Gamma \rangle = \langle 1 + \sqrt{2}, \sqrt{2} \rangle$, though the irrational parameters are expensive to implement (Roura, 2001). We found valid rational parameters, which can be implemented at low cost.

Contributions of the paper are as follows:

- Exact identification of the range of valid parameters of WBT where insertion and deletion preserve the balance constraints of the original paper
  - — $\langle \Delta, \Gamma \rangle = \langle 3, 2 \rangle$ is the only integer solution in the range,
  - — Valid parameter pairs lie in a bounded range,
  - — The boundary is complicated, consisting of nine different lines and a curve (Figure 5),

[2] The Data.Map of the `container` library 0.3.0.0 or earlier has this bug.

  - The original proposal $\langle \Delta, \Gamma \rangle = \langle 1 + \sqrt{2}, \sqrt{2} \rangle$ in Nievergelt & Reingold (1972) maintains the strictest balance condition available within the range,
  - The smaller Δ is the better performer;
- (soundness) A proof in Coq that insertion and deletion preserve balance of trees when the parameters lie within the range;
- (completeness) An implemented and tested procedure for producing counterexamples for any parameter pair outside the range.

This paper is organized as follows. We define terminology in Section 2. In Section 3, we describe the basics of WBT algorithms and explain the difference between the original WBT algorithm and a variant WBT algorithm. We point out the drawback of the variant WBT algorithm in Section 4 and explain that our goal is to find the valid range of the original WBT in Section 5. The valid range is identified by preliminary tests in Section 6. For soundness of the range, we describe a Coq certified proof in Section 7. For completeness, we show the method of producing counterexamples outside the valid range in Section 8. Section 9 explains that our suggested parameter retains almost the same performance as the existing parameters. We show related work and conclusions in Sections 10 and 11, respectively.

2 Terminology

The original paper (Nievergelt & Reingold, 1972) uses the name "binary search trees of bounded balance." However, following Knuth's book (Knuth, 1998), we call the same family of algorithms "weight-balanced trees." We use "the original WBT" for the WBT algorithm in the original paper (Nievergelt & Reingold, 1972) and "the variant WBT" for the WBT algorithm in Adams's technical report (Adams, 1992) .

To check the balance of a WBT, *weights* of its left subtree and right subtree are used. In the original WBT, the weight of a tree is the number of contained elements plus one. In the variant WBT, the weight of a tree is simply the number of contained elements. Both WBT algorithms use two parameters Δ and Γ mentioned in Section 1. For the original WBT, we use $\langle \Delta, \Gamma \rangle$. For the variant WBT, we use (Δ, Γ).

We use the Haskell syntax to describe algorithms.

3 WBT

In this section, we briefly explain the basics of WBT.

3.1 Data structure

The data structure used in the WBT algorithms can be defined as follows[3]:

```
type Size = Int
data Set a = Tip | Bin Size a (Set a) (Set a)
```

[3] Our code for algorithm implementation, tests, benchmarks, and proofs are available on the author's web page `http://www.mew.org/~kazu/proj/weight-balanced-tree/` and the archive of the journal.

A WBT is either `Tip` containing no element or a `Bin`-constructed tree containing an element and two subtrees. For convenience, we define three basic operations—getting the `Size` information, creating a singleton, and creating an intermediate node.

```
size :: Set a -> Size
size Tip           = 0
size (Bin sz _ _ _) = sz

singleton :: a -> Set a
singleton k = Bin 1 k Tip Tip

bin :: a -> Set a -> Set a -> Set a
bin k l r = Bin (size l + size r + 1) k l r
```

As shown above, the size of `Tip` is 0 and the size of a `Bin`-constructed tree is the sum of the sizes of its two subtrees, plus 1.

3.2 Balance of WBT

In order to avoid deep, thin trees shaped like a list, WBT algorithms impose a balance condition on trees. The balance condition is parameterized with the `isBalanced` predicate taking two trees. First, `Tip` is balanced. Second, a `Bin`-constructed tree is balanced if

1. the weights of the left and right subtrees satisfy `isBalanced` predicate, and
2. its left and right subtrees are balanced.

For testing purposes, the balance condition can be implemented as follows:

```
balanced :: Set a -> Bool
balanced Tip           = True
balanced (Bin _ _ l r) = isBalanced l r && isBalanced r l
                      && balanced l     && balanced r
```

The `balanced` function uses the `isBalanced` predicate in order to check the first condition of balancing. Each WBT algorithm has its own `isBalanced` predicate, which is shown later. The `balanced` function then recursively calls itself on both subtrees in order to check the second condition of balancing.

3.3 Creating WBT

An element is inserted into a WBT according to the order of the element. We assume a total order for the elements. For example, an element larger than the element in the top node is inserted into the right subtree. In Haskell, we use the `compare` function provided by `Ord` class to compare two elements.

```
insert :: Ord a => a -> Set a -> Set a
insert kx Tip = singleton kx
insert kx (Bin sz ky l r) = case compare kx ky of
    LT -> balanceR ky (insert kx l) r
    GT -> balanceL ky l (insert kx r)
    EQ -> Bin sz kx l r
```

After an element is inserted, the balance between the left and right subtrees might be broken. This is checked by `isBalanced`, which uses Δ. If the two subtrees are still balanced, a new node is simply created; otherwise, a rotation is performed. The exact condition depends on whether the original or the variant WBT is used. If an element is inserted into the right subtree and the balance is broken, a left rotation is performed.

```
balanceL :: a -> Set a -> Set a -> Set a
balanceL k l r
  | isBalanced l r = bin k l r
  | otherwise      = rotateL k l r
```

For the rest of this paper, we only consider left rotations. Symmetric arguments can be applied to right rotations. Note that a left rotation is also performed when an element is deleted from the left subtree. There are two kinds of left rotations. One is called a single rotation and the other is called a double rotation (Figure 1). The `isSingle` predicate, which uses Γ, decides which rotation is used.

```
rotateL :: a -> Set a -> Set a -> Set a
rotateL k l r@(Bin _ _ rl rr)
  | isSingle rl rr = singleL k l r
  | otherwise      = doubleL k l r
rotateL _ _ _      = error "rotateL"

singleL :: a -> Set a -> Set a -> Set a
singleL k1 t1 (Bin _ k2 t2 t3) = bin k2 (bin k1 t1 t2) t3
singleL _ _ _                  = error "singleL"

doubleL :: a -> Set a -> Set a -> Set a
doubleL k1 t1 (Bin _ k2 (Bin _ k3 t2 t3) t4)
  = bin k3 (bin k1 t1 t2) (bin k2 t3 t4)
doubleL _ _ _ = error "doubleL"
```

Both single and double rotations move a part of the right subtree into the left subtree. A single rotation moves the left subtree of the right subtree. A double rotation moves a smaller part: the left subtree of the tree moved by a single rotation. A single rotation can break the balance of the whole tree if the subtree being moved is too large. To prevent that the `isSingle` function chooses a double rotation when the subtree is much larger than its right sibling. Each WBT algorithm has its own `isSingle` function, which is shown later. The error cases in these above three functions does not occur since the `insert` operation adds one element onto the right subtree.

3.4 Original balancing method

In the original WBT, the weight of a tree is its size plus one. After insertion into the right subtree, if the weight of the left subtree multiplied by Δ is larger than or equal to the weight of the right subtree, balance is maintained. Otherwise, the balance is broken.

```
isBalanced :: Set a -> Set a -> Bool
isBalanced a b = delta * (size a + 1) >= (size b + 1)
```

When the balance is broken, the algorithm chooses a single rotation or a double rotation by comparing the weights of two subtrees of the right subtree (node c in Figure 1). A single rotation is chosen if the weight of the left subtree of node c is less than the weight of the right subtree of node c multiplied by Γ. Otherwise, a double rotation is chosen.

```
isSingle :: Set a -> Set a -> Bool
isSingle a b = (size a + 1) < gamma * (size b + 1)
```

3.5 Variant balancing method

In the variant WBT, the weight of a tree is simply its size. Since the weight of an empty tree is zero, a small tree must be treated specially in `isBalanced` in order to have at least one balanced tree of size two. This is implemented as `x + y <= 1` in the following code.

```
isBalanced :: Set a -> Set a -> Bool
isBalanced a b = x + y <= 1 || delta * x >= y
  where x = size a
        y = size b

isSingle :: Set a -> Set a -> Bool
isSingle a b = size a < gamma * size b
```

4 Balancing bugs in the variant WBT

As we said in Section 1, the variant WBT is widely used, but unfortunately a bug was reported for the `Data.Map` library of Haskell, which used $(5, 2)$. If one element is removed from a specific balanced tree, the balance of the resulting tree is broken in some cases. We defined 86 cases for smoke testing (simple testing in normal cases) and 32 property tests with QuickCheck (Claessen & Hughes, 2000). Our tests found bugs for all integer parameter pairs except $(4, 2)$ and $(3, 2)$. `Data.Set` uses $(4, 2)$, so we cannot find the bug in the current `Data.Set` implementation. Note that such testing cannot prove the absence of counterexamples.

The technical report by Adams (1992) recommends (4 or larger, 1) and his later paper (Adams, 1993) uses $(5, 1)$. His Standard ML implementation makes use of $(3, 1)$ and his Scheme implementation `wttree.scm` both in MIT/GNU Scheme and `slib` uses $(5, 1)$. They are all buggy.[4]

Figure 2 shows a counterexample to the variant WBT with $(5, 1)$. The original tree on the left side of the figure is balanced. If we delete the element a, a double rotation is performed, and then, the tree on the right side of the figure is created. At the node of the element e, the weight of the left and right subtree is 0 and 2, respectively. This does not satisfy the balance constraint.

[4] `wttree.scm` in `slib` 3b3 or earlier has this bug but our fix was merged in December 2010. `wttree.scm` in MIT/GNU Scheme 9.0.1 or earlier has this bug but our fix was merged in January 2011.

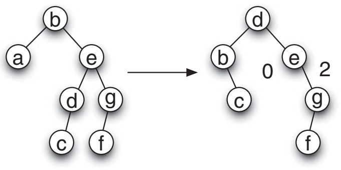

Fig. 2. Counterexample to the variant WBT with (5,1).

5 Goal

We chose the original WBT as our target algorithm because it has two benefits:

- Since weights are nonzero natural numbers, we do not have to treat small trees as special. This makes mathematical analysis easier because there are fewer cases to consider;
- The mathematical analysis in Nievergelt & Reingold (1972) is credible. It considers the balance preservation by both insertions and deletions.

Note that it is easy to convert an existing program based on the variant WBT to the original WBT because the only difference is in the `isBalanced` and `isSingle` functions. We will discuss the original WBT in the rest of this paper. We chose to seek rational valid parameters in addition to $\langle 1+\sqrt{2}, \sqrt{2}\rangle$ suggested by the original paper. To implement the originally suggested balance condition with integer arithmetic, we have to compare the squares of the weights. Here is a straightforward implementation:

```
isBalanced :: Set a -> Set a -> Bool
isBalanced a b = 2 * y * y <= z * z
  where x = size a + 1
        y = size b + 1
        z = x + y

isSingle :: Set a -> Set a -> Bool
isSingle a b = z * z < 2 * w * w
  where z = size a + 1
        w = size b + 1
```

Since integers in typical computer languages are fixed length, this calculation is prone to overflow. In order to avoid this problem, we have to deploy more complicated implementation. Rational parameters are preferable.

6 Identifying valid range

Before delving into rigorous mathematical analysis, we used our test suite described in Section 4 and the Omega solver (Pugh, 1991) to identify the range of valid parameters. Since only $(4,2)$ and $(3,2)$ are possible integer solutions for the variant WBT, we guessed the valid range for the original WBT is around them. We tested the original WBT using our test suite, where Δ is an integer between 1 and 10 and Γ is an integer between 1 and 10. These tests showed that only $\langle 3,2\rangle$ is a possible integer solution for the original WBT.

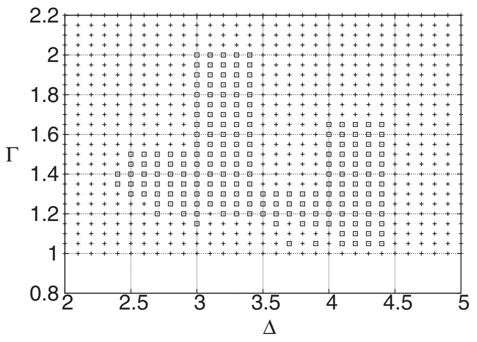

Fig. 3. Results of tests plotted along $\langle \Delta, \Gamma \rangle$. The dotted square symbols ⊡ indicate that no insertion nor deletion broke the balance. The plus symbols $+$ indicate discovery of concrete counterexamples.

6.1 Tests

To obtain a more precise parameter range around $\langle 3, 2 \rangle$, we tested not only with integer parameters, but also with rational parameters. Figure 3 shows the results with the range, where $\Delta = 2, 2.1, \ldots, 5$ and $\Gamma = 1, 1.05, \ldots, 2.2$. The shape of the valid range seemed more complex than we had expected.

6.2 Automated arithmetic solver

We also ran the automated arithmetic solver Omega (Pugh, 1991) with the same set of parameter pairs. For each parameter pair, we gave a logical formula in Presburger arithmetic to the solver. The logical formulas expressed the possibility of balance breaking by insertion or deletion in a WBT tree.

We produced the inputs for Omega by defining the balance preservation condition in Coq and then converting the condition into an assertion that only contains linear inequalities. We then replaced the conjunction in Coq (`/\`) with the conjunction in Omega (`&&`), added parentheses, and applied some other cosmetic changes. The Omega solver (Pugh, 1991) showed three kinds of behavior. In some cases, the Omega solver gave a concrete counterexample within a second. In other cases, within a second, the solver confirmed any large enough balanced WBT tree is balanced after any insertion or deletion. Sometimes, the solver gave up or did not respond for a couple of minutes, so we terminated it. Figure 4 illustrates the result.

Comparing Figures 3 and 4, we saw that Omega found new counterexamples for parameters, where QuickCheck was not able to. We guessed that the first experiment could not find some existing counterexamples outside the lower boundaries.

6.3 Finding boundaries

We conjectured that the following four boundaries determine the valid parameter range:

Right	$\Delta < 4.5$
Left	$\Gamma \leqslant \Delta - 1$

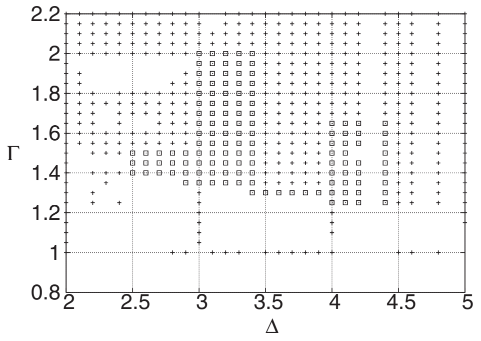

Fig. 4. Results using the automated arithmetic solver Omega plotted along $\langle \Delta, \Gamma \rangle$. The square symbols ⊡ indicate the parameter pairs are valid. The plus symbols + indicate discovery of concrete counterexamples. Blanks mean time-out.

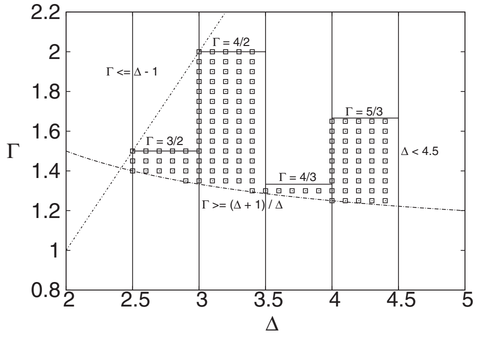

Fig. 5. The boundaries of the valid parameter range for the original WBT are shown with inequalities. The points ⊡ are some rational valid parameters.

$$\text{Lower} \quad \Gamma \geqslant (\Delta + 1)/\Delta$$
$$\text{Upper} \quad \Gamma \leqslant \begin{cases} 3/2 & (2.5 \leqslant \Delta < 3) \\ 4/2 & (3 \leqslant \Delta < 3.5) \\ 4/3 & (3.5 \leqslant \Delta < 4) \\ 5/3 & (4 \leqslant \Delta < 4.5). \end{cases}$$

Plotting these lines results in Figure 5.

The only integer solution is $\langle 3, 2 \rangle$. The suggested value of the original paper $\langle 1 + \sqrt{2}, \sqrt{2} \rangle$ is the point of intersection of the left boundary and lower boundary and is most balanced since $1 + \sqrt{2}$ is the smallest value of Δ.

7 Soundness: proving validity in Coq

We first ensure the soundness, that is, for parameter pairs inside the valid range, insertion and deletion always keep the balance. We formalized a proof in Coq (Bertot & Casteran, 2004), a proof assistant. The soundness proof involved many cases, and in each case, we had to check balance constraints on many different nodes. Keeping

```
Definition isBalancedSize (x y:Q): Prop :=delta × (x + 1) ≥ (y + 1).
Definition isSingleSize (x y:Q): bool :=(1 +x) '<' (ratio × (1 +y))
Definition balancedQ (l r:Q) :=l ≥ 0 ∧ r ≥ 0 ∧ isBalancedSize l r ∧ isBalancedSize r l.
Definition singly_balanced (x y z w: Q) := balancedQ x (y +z + 1) ∧ balancedQ y z ∧
  balancedQ (x +y +z +1 + 1) w.
Definition doubly_balanced (x y z w: Q) :=
  balancedQ x y ∧ balancedQ z w ∧ balancedQ (x +y + 1) (z +w + 1).
Definition deep_insert (x v y z w: Q) := balancedQ x v → ~balancedQ x (v + 1) →
  y +z + 1 +w + 1 == v + 1 → balancedQ y z → balancedQ (y +z + 1) w →
  if (isSingleSize (y +z + 1) w) then singly_balanced x y z w else doubly_balanced x y z w.
Definition deep_delete (x v y z w: Q) :=
    0 ≤ x → balancedQ (1 +x) v → ~balancedQ x v → y +z + 1 +w + 1 ==v → balancedQ y z →
    balancedQ (y +z + 1) w →
    if (isSingleSize (y +z + 1) w) then singly_balanced x y z w else doubly_balanced x y z w.
Lemma NR_deep_insert: good_params → ∀ (x v y z w: Z), deep_insert (x#1) (v#1) (y#1) (z#1) (w#1).
Lemma NR_deep_delete: good_params → ∀ (x v y z w:Z), deep_delete (x#1) (v#1) (y#1) (z#1) (w#1).
```

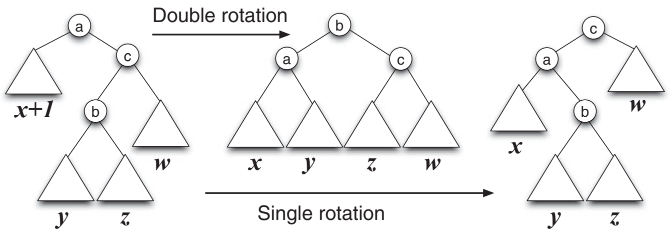

Fig. 6. Two intermediate arithmetic lemmas. The variables x, y, z, and w informally stand for the size of each subtree shown in the picture although the formal definition says nothing about tree structures. *good_params* denotes the restriction on parameters shown in Section 6.3. The operator "<" denotes comparison that returns a Boolean. Note that in the final lemmas, the domain is restricted to integers. We used this restriction for case analysis on particularly small trees. An integer z can be converted into a rational number by writing $(z\#1)$.

track of those numerous cases and constraints by hand is troublesome and error-prone.

In the 15,700 lines[5] of Coq proof script, the first and second parts are abount insertion and deletion, while the third and fourth parts are about set operations. The first part proves arithmetic statements involving Δ and Γ. The second part proves that any insertion or deletion on any balanced tree yields a balanced tree if the parameter pair $\langle \Delta, \Gamma \rangle$ lies within the conjectured range.

The first arithmetic statements are made in terms of rational numbers. We just considered the sizes of five subtrees involved in rotations. Since Δ and Γ are not fixed, the problem did not fit in Presburger arithmetic so that we could not use `omega` tactic. While proving lemmas shown in Figure 6, we intensively used the Psatz interface to the CSDP solver (Borchers, 1999). For a proof goal consisting of

[5] An anonymous referee pointed out that since the script is written in a sparse manner, it is probably possible to prove the same results with about 3,000 lines in a more concise style.

```
Lemma insert_balanced: good_params →
  ∀ (t: FSet) (k: K), balance_rec t → validsize_rec t → balance_rec (insert k t).
Lemma delete_balanced: good_params →
  ∀ (t: FSet) (k: K), balance_rec t → validsize_rec t → balance_rec (delete k t).
```

Fig. 7. Two program theorems shown in the second half. *balance_rec* means that the whole tree is balanced. *validsize_rec* means that every node in the tree has the correct size information on it.

polynomial inequations, the Psatz interface tries to build a proof automatically with the help of an external solver called CSDP. For some special small sizes, we had to manually compute the ceil function when we stated that an integer less than 13.5 must be less than or equal to 13.

In the second part, we treated actual programs operating on actual tree structures and proved lemmas shown in Figure 7. The second part does not rely on rational numbers. Instead, we introduced integer variables called *deltaU* and *deltaD* to denote the rational number *deltaU*/*deltaD*. In this part, we had to use induction on trees. Moreover, we had to give special treatment to some particularly small trees because the arithmetic lemmas required trees to have sufficiently many subtrees.

Set operations. We also verified that the set operations (union, difference, and intersection) preserve the balance condition under parameter pairs $\langle 3, 2\rangle$ and $\langle 5/2, 3/2\rangle$, respectively, in the third and fourth parts. (The reason for choosing the second pair is the benchmark described in Section 9.) That is, if two WBTs are balanced, their union, difference, and intersection are also balanced.[6] For the set operations, we used the efficient hedge-union algorithms used in the current version of `Data.Set` and `Data.Map` library, not the simple divide-and-conquer. The technical paper by Adams (1992) describes the hedge approach as well as the divide-and-conquer approach.

8 Completeness: producing counterexamples

This section shows how to produce counterexamples outside the valid parameter range. First of all, we exclude parameter pairs not satisfying some basic constraints. The basic constraints are as follows:

$$\Delta \geqslant 2, \tag{1}$$

$$1 \leqslant \Gamma \leqslant \Delta. \tag{2}$$

If the first constraint is not satisfied, there are no balanced trees of size two. On the other hand, if the second constraint is broken, only single or double rotations are chosen so that it is impossible to maintain balance.

[6] For intersection, we used an experimental `Function` command of Coq 8.2 because the intersection function has a complicated recursion: when the intersection function calls itself, the new argument is generated by another function with recursion so the ordinary `Fixpoint` command cannot guess the decreasing argument.

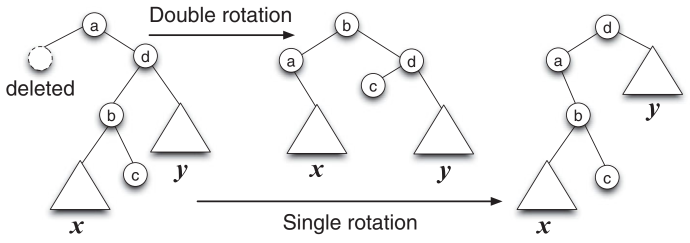

Fig. 8. A counterexample outside the right boundary. x and y denote the size of each subtree. The original tree on the left side is balanced. However, after deletion of the only element in the left subtree, neither a single rotation nor a double rotation maintains the balance at node a.

8.1 Outside the right boundary

Assume that the parameter pair is outside the right boundary:

$$\Delta \geqslant 4.5.$$

Consider the trees in Figure 8. The size of each tree is defined as follows:

$$x = \lfloor\Delta\rfloor,\ y = \lfloor 2\Delta\rfloor - \lfloor\Delta\rfloor - 4.$$

This original tree on the left side of the figure is balanced. To see that, we look at each node. Let us consider the balance at node a first. Since the size of the right subtree $x + y + 3$ is larger than that of the left subtree, we only have to confirm that the right subtree is not too large. More specifically, Δ times the weight of the left subtree must be greater than or equal to the weight of the right subtree. Since the weight of a tree is the size plus one, we are seeking this inequality:

$$\Delta \times (1 + 1) \geqslant (3 + x + y) + 1.$$

This can be confirmed as follows:

$$2\Delta - (x + y + 4) = 2\Delta - \lfloor 2\Delta\rfloor \geqslant 0.$$

Node d also must satisfy a similar constraint as follows:

$$f(\Delta) = \Delta(y + 1) - (x + 3) = \Delta\lfloor 2\Delta\rfloor - (\Delta + 1)(\lfloor\Delta\rfloor + 3) \geqslant 0.$$

It is possible to mathematically analyze that the left-hand side has a positive value for $\Delta \geqslant 4.5$ but the analysis is boring. Instead, we plot the graph of $f(\Delta)$ in Figure 9, where $f(\Delta)$ denotes the expression above.

It is obvious that node b is balanced. Thus, the entire original tree is balanced. If we delete the left subtree of node a, the balance is broken:

$$\Delta - (x + y + 4) = \Delta - \lfloor 2\Delta\rfloor < 0 \quad \text{(by Equation (1))}.$$

So, either a single or double rotation takes place. After a double rotation, the balance at node a is broken because $\Delta - (\lfloor\Delta\rfloor + 1) < 0$. Otherwise, after a single rotation, the balance at node a is also broken since $\Delta - (\lfloor\Delta\rfloor + 3) < 0$.

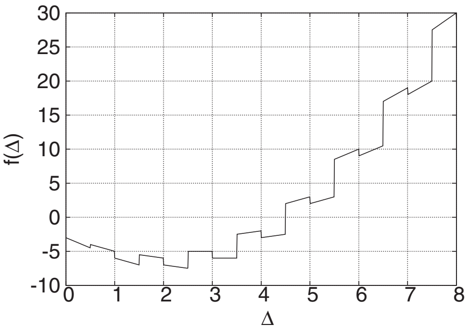

Fig. 9. The value of $f(\Delta)$ is greater than 0 if $\Delta \geqslant 4.5$.

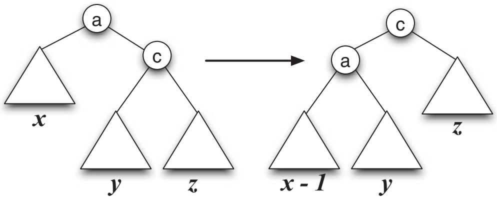

Fig. 10. A counterexample of the left boundary. The original tree in the left side is balanced. However, after deletion of one element in the left subtree, a single rotation breaks the balance at node c if y is large enough.

8.2 Outside the left boundary

Assume that the parameter pair is outside the left boundary:

$$\Gamma > \Delta - 1 \quad \text{equivalently} \quad \Delta - \Gamma - 1 < 0. \tag{3}$$

Consider the trees in Figure 10. Each of x, y, and z is the size of a subtree of the original tree. x and z are defined using y:

$$z = \left\lfloor \frac{y+1}{\Gamma} \right\rfloor, \quad x = \left\lceil \frac{y+z+2}{\Delta} \right\rceil - 1.$$

This original tree is balanced because of the following. At node c, the right subtree of size z is not too much larger than the left subtree of size y for sufficiently large y:

$$\begin{aligned}\Delta(y+1) - (z+1) &= \Delta(y+1) - \left(\left\lfloor \frac{y+1}{\Gamma} \right\rfloor + 1\right)\\ &\geqslant \Delta(y+1) - \left(\frac{y+1}{\Gamma} + 1\right)\\ &= \left(\Delta - \frac{1}{\Gamma}\right) y + C\\ &\geqslant 0\end{aligned}$$

where C does not contain x, y, or z. The last inequality holds for large y because the coefficient for y is $\Delta - 1/\Gamma$, which is positive since $\Delta > 1 \geqslant 1/\Gamma$.

The left subtree of c is not too much larger than the right subtree, either:

$$\begin{aligned}\Delta(z+1)-(y+1) &= \Delta\left(\left\lfloor\frac{y+1}{\Gamma}\right\rfloor+1\right)-(y+1)\\ &> \Delta\frac{y+1}{\Gamma}-(y+1)\\ &= \left(\frac{\Delta}{\Gamma}-1\right)(y+1)\\ &\geqslant 0 \quad \text{(by Equation (2)).}\end{aligned}$$

At node a, the right subtree is not smaller: $x \leqslant y+z+1$. At the same time, the right subtree is not too large:

$$\Delta(x+1)-(y+z+2) = \Delta\left(\left\lceil\frac{y+z+2}{\Delta}\right\rceil\right)-(y+z+2) \geqslant \Delta\frac{y+z+2}{\Delta}-(y+z+2)=0.$$

If we delete one element from the left subtree of node a, the balance is broken as follows:

$$\Delta x-(y+z+2) = \Delta\left(\left\lceil\frac{y+z+2}{\Delta}\right\rceil-1\right)-(y+z+2) < \Delta\frac{y+z+2}{\Delta}-(y+z+2)=0.$$

At this time, a single rotation is chosen because

$$\Gamma(z+1)-(y+1) = \Gamma\left(\left\lfloor\frac{y+1}{\Gamma}\right\rfloor+1\right)-(y+1) > \Gamma\frac{y+1}{\Gamma}-(y+1)=0.$$

After the single rotation, the size of the left subtree of node c is $x+y$ and that of the right subtree is z. The balance is broken if the following expression has a negative value:

$$\begin{aligned}\Delta(z+1)-(x+y+1) &= \Delta\left(\left\lfloor\frac{y+1}{\Gamma}\right\rfloor+1\right)-\left(\left\lceil\frac{y+\lfloor\frac{y+1}{\Gamma}\rfloor+2}{\Delta}\right\rceil+y-1\right)\\ &< \Delta\left(\frac{y+1}{\Gamma}+1\right)-\left(\frac{y+\lfloor\frac{y+1}{\Gamma}\rfloor+2}{\Delta}+y-1\right)\\ &< \Delta\left(\frac{y+1}{\Gamma}+1\right)-\left(\frac{y+\frac{y+1}{\Gamma}+1}{\Delta}+y-1\right)\\ &= \frac{(\Delta+1)(\Delta-\Gamma-1)}{\Delta\Gamma}y+C\end{aligned}$$

where C does not contain x, y, or z. The coefficient for y is negative by Equation (3). This implies, for sufficiently large y, the rotated tree becomes unbalanced after a delete operation.

8.3 Outside the lower boundary

Assume that the parameter pair is outside the lower boundary:

$$\Gamma < \frac{\Delta+1}{\Delta} \quad \text{equivalently,} \quad \Gamma\Delta-\Delta-1<0. \tag{4}$$

Consider the tree in Figure 11. We define the sizes x, y, and w using z and an auxiliary variable named r:

$$w=\lfloor\Delta(z+1)\rfloor, \quad y=w-1, \quad r=y+z+w+2, \quad x=\left\lceil\frac{r+1}{\Delta}\right\rceil-1.$$

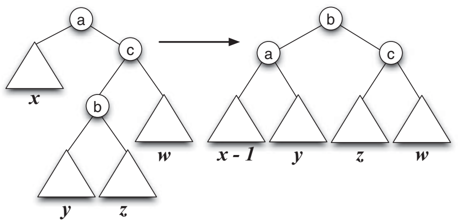

Fig. 11. A counterexample for parameter pairs outside the lower boundary. x, y, z, and w denote the size of each subtree. The original tree on the left side is balanced. However, after deletion of one element in the left subtree, a double rotation breaks the balance at node c if z is large enough.

For large values of x, the original tree on the left side of the figure is balanced. To see that, let us look at each node. On node a, the right subtree is not too large:

$$\Delta(x+1) = \Delta\left\lceil\frac{r+1}{\Delta}\right\rceil \geqslant \Delta\frac{r+1}{\Delta} = r+1.$$

The left subtree of node a is not too large if r is large enough:

$$\Delta(r+1) > r+2 > \lceil r+1 \rceil > \left\lceil\frac{r+1}{\Delta}\right\rceil = x+1.$$

On node c, the right subtree is not too large:

$$\Delta(y+z+1+1) = \Delta\left(\lfloor\Delta(z+1)\rfloor + z + 1\right) \geqslant \lfloor\Delta(z+1)\rfloor + 1 = w+1.$$

On the other hand, the left subtree of node c is not too large for large values of z:

$$\Delta(w+1) = \Delta\left(\lfloor\Delta(z+1)\rfloor + 1\right) \geqslant \lfloor\Delta(z+1)\rfloor + z + 1 = y+z+2$$

where the inequality in the middle holds when z is large enough. This is because both sides are almost linear on z, where the coefficient on the left side Δ^2 is larger than the coefficient on the right side $\Delta + 1$. The inequality between the coefficients $\Delta^2 > \Delta + 1$ holds because $\Delta \geqslant 2$.

On node b, the right subtree is not too large:

$$\Delta(y+1) = \Delta w = \Delta\lfloor\Delta(z+1)\rfloor > \lfloor\Delta(z+1)\rfloor \geqslant z+1$$

where the inequalities come from $\Delta \geqslant 2$ and the fact that z is an integer. On the other hand, the left subtree of node b is not too large:

$$\Delta(z+1) \geqslant \lfloor\Delta(z+1)\rfloor = w = y+1.$$

Although the original tree is balanced as we have seen, if we delete an element from the left subtree of node a, the balance is broken:

$$\Delta x - (r+1) = \Delta\left(\left\lceil\frac{r+1}{\Delta}\right\rceil - 1\right) - (r+1) < \Delta\left(\frac{r+1}{\Delta}\right) - (r+1) = 0.$$

This implies either a single rotation or a double rotation takes place. Actually, a double rotation takes place if z is large enough because the following expression has a nonpositive value:

$$\begin{aligned}\Gamma(w+1) - (y+z+2) &= \Gamma(\lfloor\Delta(z+1)\rfloor + 1) - (\lfloor\Delta(z+1)\rfloor + z + 1)\\ &< \Gamma(\Delta(z+1)+1) - (\Delta(z+1)+z)\\ &= (\Gamma\Delta - \Delta - 1)z + C.\end{aligned}$$

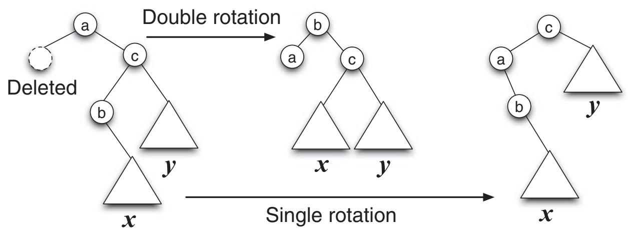

Fig. 12. A counterexample for parameter pairs outside the upper boundary. x and y denote the sizes of the subtrees. The original tree on the left side is balanced. However, deletion of the single element in the left subtree breaks the balance. A double rotation maintains the balance but a single rotation breaks the balance at node a. When the parameter pair is outside the upper boundary, a single rotation is chosen.

where C does not contain x, y nor z. The coefficient of z is negative according to the inequation (4). So, we can choose a large enough z that ensures a double rotation. If a double rotation is chosen, the balance is broken at node c:

$$\Delta(z+1) - (w+1) = \Delta(z+1) - (\lfloor \Delta(z+1) \rfloor + 1) < \Delta(z+1) - \Delta(z+1) = 0.$$

8.4 Outside the upper boundaries

Some specific small trees determine the upper boundaries. Consider the trees in Figure 12. The sizes of the subtrees in the figure are defined as follows:

$$x = \lfloor \Delta \rfloor - 1, \quad y = \lfloor \Delta - 1/2 \rfloor - 1.$$

Since we already have the other boundaries, we only have to consider $2.5 \leqslant \Delta < 4.5$. Thus, in this last case, we only have to deal with four different small trees. It is easy to check that these four trees are balanced and that its balance is broken if the left subtree of node a is removed. If a double rotation is chosen, the resulting tree is balanced. If a single rotation is chosen, the balance at node a is broken:

$$\Delta - (x+2) = \Delta - \lfloor \Delta \rfloor - 1 < 0.$$

In order to obtain a counterexample, it is enough to ensure a single rotation. For this, satisfying the following inequality is enough:

$$\Gamma > \frac{x+2}{y+1} = \overline{\Gamma}.$$

In the table below, we summarize the above result for the four different trees.

	x	y	$\overline{\Gamma} = (x+2)/(y+1)$
$2.5 \leqslant \Delta < 3$	1	1	3/2
$3 \leqslant \Delta < 3.5$	2	1	4/2
$3.5 \leqslant \Delta < 4$	2	2	4/3
$4 \leqslant \Delta < 4.5$	3	2	5/3

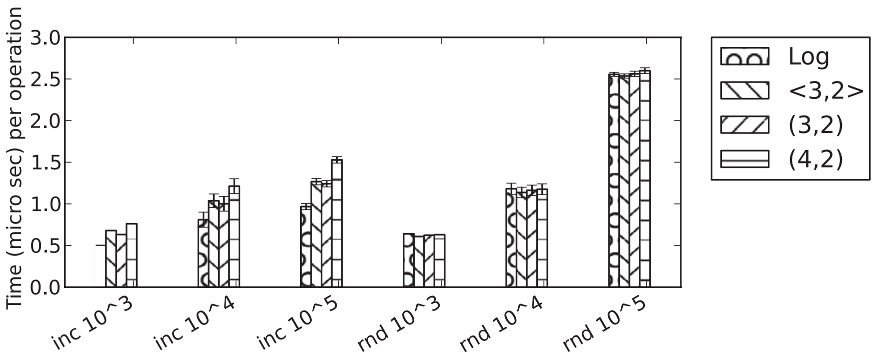

Fig. 13. Performance of the insert operation using different WBT algorithms.

8.5 Tests of the counterexamples

To check the correctness of the boundaries, we defined four tests for each boundary that produced counterexample trees following the descriptions given above. The results of the tests are exactly the same as illustrated in Figure 5.

9 Performance

The balance constraints are ultimately for performance. We benchmarked the original WBT with $\langle 3,2\rangle$ to compare against the variant WBT with $(3,2)$ and $(4,2)$, and Logarithmic BST described in Section 10. Their code is based on the Haskell `Data.Map` implementation in the `containers` package version 0.3.0.0.[7] We used Dell OptiPlex 960 with a 2.66 GHz Intel Core 2 Quad CPU with 2 GB memory running Linux 2.6.35. The Haskell compiler was the Glasgow Haskell Compiler version 6.12.3 with the -O2 option. Benchmarking a language with lazy evaluation is not straightforward. We used the `criterion` package version 0.5.0.5 and the `progression` package version 0.4 as reliable benchmark tools. `Data.Map` is defined as strict and we used a strict data type `Int` as key. So, we removed the `toList` overhead used in `criterion` when reducing `Data.Map` to its normal form. We also benchmarked the original WBT with several rational parameters.

Comparison between the original and variant WBTs. We evaluated the performance of the insertion operation, the deletion operation, and the lookup operation. For all operations, we prepared 1k, 10k, and 100k elements both in the increasing order and random order. They are labeled as inc 10^3, inc 10^4, inc 10^5, rnd 10^3, rnd 10^4, and rnd 10^5, respectively, in Figures 13–15. Some error bars are invisibly short.

For the insertion operation, we measured the entire time to construct a WBT tree from all elements. The results are illustrated in Figure 13. For the delete operation, we first constructed a WBT tree from all elements then measured the entire time to delete each element in the insertion order from the full tree. The results are

[7] As of this writing, performance tuning is going on. The `containers` package version 0.3.0.0 does not include such performance tuning.

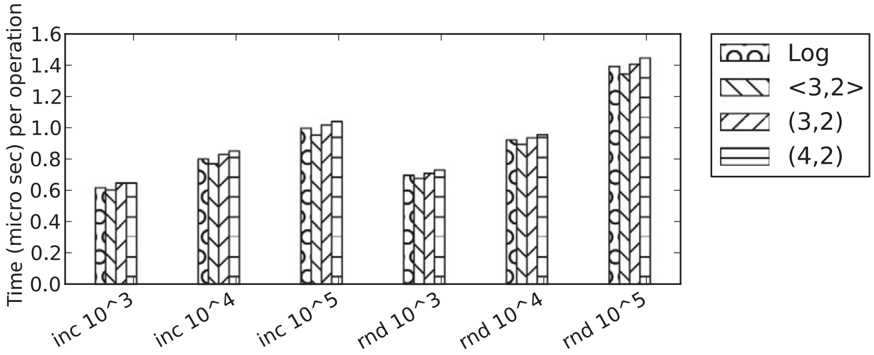

Fig. 14. Performance of the delete operation using different WBT algorithm.

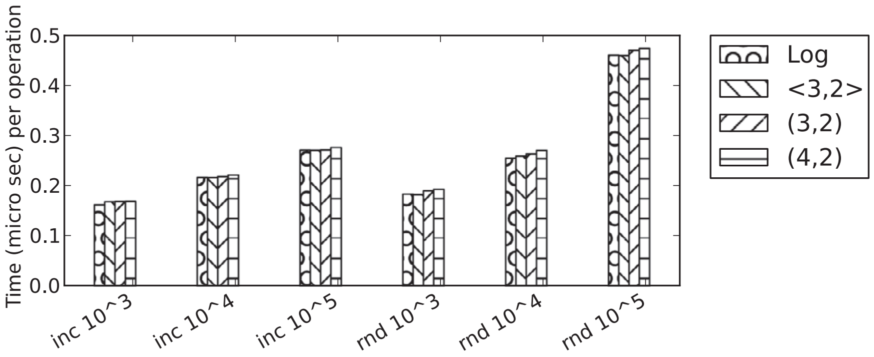

Fig. 15. Performance of the lookup operation using different WBT algorithm.

illustrated in Figure 14. For the lookup operation, we first constructed a WBT tree from all elements then we measured the entire time to look up each element in the tree. The results are illustrated in Figure 15. To show the results of three different sizes in a graph, we divide each entire time by each size. We can say that the original WBT with $\langle 3, 2\rangle$ has at least the same performance as the variant WBT with $(3, 2)$ and $(4, 2)$ and Logarithmic BST.

Comparison among different parameter choices for the original WBT. Likewise, we compared the performance of eight different parameter pairs within the valid range for insertion, deletion, and lookup (Figures 16–18). We found that the smaller Δ, which enforces the stricter balance condition, performs better. For incremental inputs, the largest time difference between the slowest and the fastest reached 43% for insertion. For randomized inputs, the largest difference was 14% for lookup.

10 Related work

Coq verification of balanced tree algorithms. Filliâtre and Letouzey (2004) proved correctness of AVL tree and red-black tree implementations in Coq and extracted OCaml codes from the Coq implementation. At some stages during the

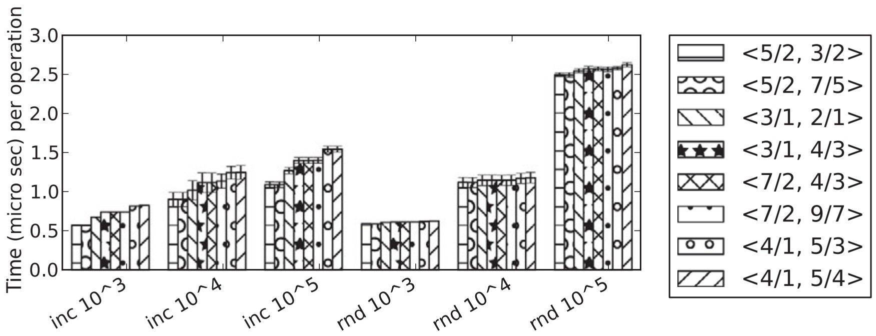

Fig. 16. Performance of the insert operation with different verified parameter pairs.

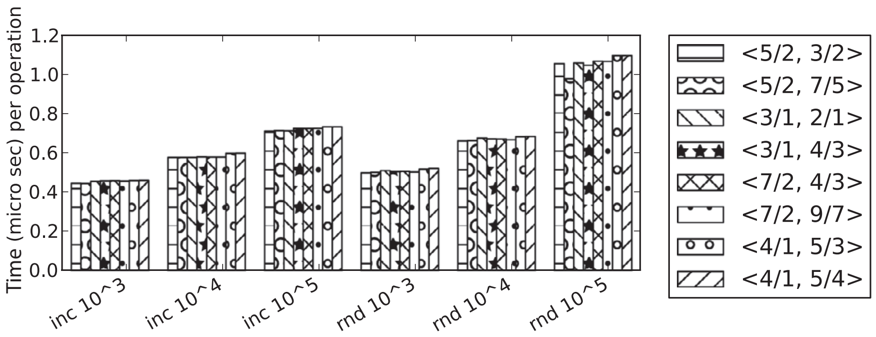

Fig. 17. Performance of the delete operation with different verified parameter pairs.

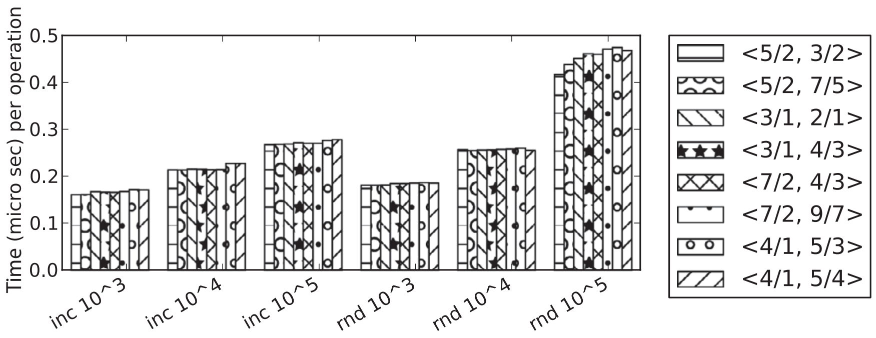

Fig. 18. Performance of the lookup operation with different verified parameter pairs.

implementation, they were not able to prove a balancing condition in Coq. This led to discovery of an implementation bug relating to the balance of the AVL tree implementation in the OCaml standard library at the time. In this paper, we pointed out balancing bugs of the algorithm, not merely in an implementation.

Charguéraud (2010) verified many functional tree algorithms in Okasaki's book (Okasaki, 1998) with a new method of transforming a program into a proposition transformer. However, neither Charguéraud's verification nor the book contains WBT algorithms. If we apply Charguéraud's method to verifying WBT algorithms, it would be much easier to verify an existing WBT implementation. However, the arithmetic argument in the first half of our Coq script would still be useful.

In contrast to both Filliâtre and Letouzey's verification and Charguéraud's verification, we have not verified that our target algorithm correctly implements finite set/map operations. We expect this to be straightforward.

Another difference is that our target algorithm is parameterized and there are many restrictions on the parameters. Moreover, some of the restrictions are combinatorially determined by small trees of size ≈ 10. Many conditions on many cases yield a large amount of case analysis, which makes hand-written proofs more error-prone and machine certified proofs more advantageous in our case.

Other balanced tree algorithms. Logarithmic BST (Roura, 2001) is another variant of WBT. To implement Logarithmic BST, `isBalanced` and `isSingle` use bit operations and other code can be shared with the WBT family.

```
(.<.) :: Size -> Size -> Bool
a .<. b
  | a >= b    = False
  | otherwise = ((a .&. b) `shiftL` 1) < b

isBalanced a b = not (size a .<. (size b `shiftR` 1))

isSingle a b = not (size b .<. size a)
```

The paper (Roura, 2001) says "$\cdots$ (the original WBT with $\langle 1+\sqrt{2}, \sqrt{2}\rangle$) which is anyway an expensive property to check. This seems to be the main reason not to use weighted BSTs as default balancing method." We show the original WBT with our choice of parameters $\langle 3, 2\rangle$ here in order to compare it with the Logarithmic BST version shown above.

```
isBalanced :: Set a -> Set a -> Bool
isBalanced a b = 3 * (size a + 1) >= size b + 1

isSingle :: Set a -> Set a -> Bool
isSingle a b = size a + 1 < 2 * (size b + 1)
```

For mathematical reliability, Logarithmic BST is simpler, but we have shown rigorous analysis of the original WBT is attainable using Coq. For performance, we benchmarked Logarithmic WBT against the original WBT with $\langle 3, 2\rangle$ (Figures 13–15). For large (10^5 elements) trees on randomized inputs, the original WBT performs as well as or slightly better than Logarithmic WBT.

11 Conclusion

We identified the exact range of the valid rotation parameters of the original weight-balanced tree and proved in Coq that it can maintain balance after any insertion and deletion operations. Within the range, the only integer solution is $\langle 3, 2\rangle$, which allows simpler implementation of the original weight-balanced tree. Benchmarks showed that the original weight-balanced tree with $\langle 3, 2\rangle$ works in almost the same performance as the variant at (3,2) and (4,2). We benchmarked other valid rational parameters and found the smaller Δ is the better performer. We proved in Coq that set operations, such as union, intersection, and difference, can maintain balance under $\langle 3, 2\rangle$ and $\langle 5/2, 3/2\rangle$. We also showed how to produce counterexamples outside the boundaries of the valid range.

Acknowledgments

The authors would like to thank Taylor Campbell for his bug report that initiated our research and Eijiro Sumii for discussion and instructive comments on our early draft. The authors are grateful to anonymous referees for a number of presentation improvements and a concise title.

References

Adams, S. (1992) *Implementing sets efficiently in a functional language*, Technical report CSTR 92-10. University of Southampton.

Adams, S. (1993) Efficient sets: A balancing act. *J. Funct. Program.*, **3**(4), 553–562.

Adel'son-Vel'skii, G. M. & Landis, E. M. (1962) An algorithm for the organization of information. *Dokl. Akad. Nauk SSSR*, **146**(2), 263–266.

Bertot, Y. & Casteran, P. (2004) *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer.

Borchers, B. (1999) CSDP, a c library for semidefinite programming. *Optim. Methods Softw.*, **11**(1), 613–623.

Charguéraud, A. (2010) Program verification through characteristic formulae.In *Proceedings of the 15th International Conference on Functional Programming (ICFP)*. ACM.

Claessen, K. & Hughes, J. (2000) QuickCheck: A lightweight tool for random testing of haskell programs. In *Proceedings. of the Fifth International Conference on Functional Programming (ICFP)*. ACM.

Filliâtre, J.-C. & Letouzey, P. (2004) Functors for proofs and programs. In *Programming Languages and Systems*, Schmidt, D. (ed), Lecture Notes in Computer Science, vol. 2986. Springer, pp. 370–384.

Guibas, L. J. & Sedgewick, R. (1978) A dichromatic framework for balanced trees. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science (SFCS '78)*. IEEE, pp. 8–21.

Knuth, D. E. (1998) *The Art of Computer Programming: Sorting and Searching*. 2nd ed., vol. 3. Addison-Wesley.

Marlow, S., *et al.* (2010) *Haskell 2010 Language Report*, Marlow, S. (ed), Available online http://www.haskell.org/ (May 2011).

Nievergelt, J. & Reingold, E. M. (1972) Binary search trees of bounded balance. In *Proceedings of the Fourth Annual Acm Symposium on Theory of Computing*. ACM, pp. 137–142.

Okasaki, C. (1998) *Purely Functional Data Structures*. Cambridge University Pres.

Pugh, W. (1991) The Omega test: A fast and practical integer programming algorithm for dependence analysis. In *Proceedings. of the 1991 ACM/IEEE Conference on Supercomputing*. ACM.

Roura, S. (2001) A new method for balancing binary search trees. In *Automata, Languages and Programming*, Orejas, F., Spirakis, P. & van Leeuwen, J. (eds), Lecture Notes in Computer Science, vol. 2076. Springer, pp. 469–480.