Formal Verification of Incremental Merkle Tree Algorithm

Daejun Park and Yi Zhang

Runtime Verification, Inc.

Abstract. We formalize the incremental Merkle tree algorithm [1,5], and prove its correctness w.r.t. the original one [3].

1 Introduction

The deposit contract of Eth 2.0 [1], written in Vyper, is a smart contract that records a set of deposits for the Beacon chain [2]. The essence of the contract is the implementation of a Merkle tree that stores the set of deposits, where the tree is updated whenever a new deposit is received. The Merkle tree employed in the contract is expected to be very large. Indeed, a Merkle tree of height 32, which can store up to 2^{32} deposits, is implemented in the current version[1] of the contract. Since the size of the Merkle tree is huge, it is not practical to reconstruct the whole tree every time a new deposit is received. To reduce both time and space requirement (thus saving the gas cost), the contract implements the incremental Merkle tree algorithm [5]. The incremental algorithm enjoys $O(h)$ time and space complexity to reconstruct (more precisely, compute the root of) a Merkle tree of height h, while a naive algorithm would require $O(2^h)$ time or space complexity. Specifically, the algorithm maintains two arrays of length h, and each construction of a new tree requires to compute only a chain starting from the new leaf (i.e., the new deposit) to the root, where the computation of the chain requires only the two arrays, thus achieving the linear time and space complexity in the height of a tree. The efficient incremental algorithm, however, leads to the deposit contract implementation being unintuitive, and makes it non-trivial to ensure its correctness. Considering the utmost importance of the deposit contract, formal verification is demanded (indeed, the only known way) to ultimately guarantee the correctness of the contract.

2 Formalization of Incremental Merkle Tree Algorithm

Notations Let T be a perfect binary tree [4] (i.e., every node has exactly two child nodes) of height h, and $T(l, i)$ denote its node at level l and index i, where the level of leafs is 0, and the index of the left-most node is 1. For example,

[1] `https://github.com/ethereum/deposit_contract/blob/ed9c81dd6788142d22106df93d5654578063eb32/deposit_contract/contracts/validator_registration.v.py`

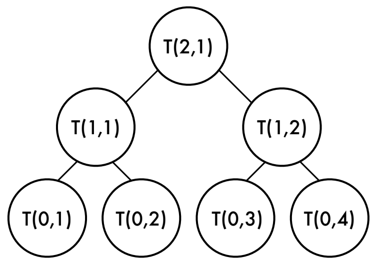

Fig. 1. A Merkle tree of height 2. We write $T(l, i)$ to denote the node of a tree T at the level l and the index i, where the level of leafs is 0, and the index of the left-most node is 1.

if $h = 2$, then $T(2, 1)$ denotes the root whose children are $T(1, 1)$ and $T(1, 2)$, and the leafs are denoted by $T(0, 1)$, $T(0, 2)$, $T(0, 3)$, and $T(0, 4)$, as shown in Figure 1. We write $[\![T(l, i)]\!]$ to denote the value of the node $T(l, i)$, but we omit $[\![\cdot]\!]$ when the meaning is clear in the context.

Let us define two functions, $\upharpoonleft$ and $\Lsh$, as follows:

$$\upharpoonleft x = \lceil x/2 \rceil \tag{1}$$

$$\Lsh x = \lfloor x/2 \rfloor \tag{2}$$

Moreover, let us define $\upharpoonleft^k x = \upharpoonleft (\upharpoonleft^{k-1} x)$ for $k \geq 2$, $\upharpoonleft^1 x = \upharpoonleft x$, and $\upharpoonleft^0 x = x$. Let $\{T(k, \upharpoonleft^k x)\}_{k=0}^h$ be a chain $\{T(0, \upharpoonleft^0 x), T(1, \upharpoonleft^1 x), T(2, \upharpoonleft^2 x), \cdots, T(h, \upharpoonleft^h x)\}$. We write $\{T(k, \upharpoonleft^k x)\}_k$ if h is clear in the context. Let us define $\Lsh^k$ and $\{T(k, \Lsh^k x)\}_k$ similarly. For the presentation purpose, let $T(l, 0)$ denote a dummy node which has the parent $T(l+1, 0)$ and the children $T(l-1, 0)$ and $T(l-1, 1)$. Note that, however, these dummy nodes are only conceptual, allowing the aforementioned chains to be well-defined, but *not* part of the tree at all.

In this notation, for a non-leaf, non-root node of index i, its left child index is $2i - 1$, its right child index is $2i$, and its parent index is $\upharpoonleft i$. Also, note that $\{T(k, \upharpoonleft^k m)\}_k$ is the chain starting from the m-th leaf going all the way up to the root.

First, we show that two chains $\{T(k, \upharpoonleft^k x)\}_k$ and $\{T(k, \Lsh^k (x-1))\}_k$ are parallel with a "distance" of 1.

Lemma 1. *For all $x \geq 1$, and $k \geq 0$, we have:*

$$(\upharpoonleft^k x) - 1 = \Lsh^k (x - 1) \tag{3}$$

Proof. Let us prove by induction on k. When $k = 0$, we have $(\upharpoonleft^0 x) - 1 = x - 1 = \Lsh^0 (x-1)$. When $k = 1$, we have two cases:

– When x is odd, that is, $x = 2y + 1$ for some $y \geq 0$:

$$(\upharpoonleft x) - 1 = (\upharpoonleft (2y+1)) - 1 = \left\lceil \frac{2y+1}{2} \right\rceil - 1 = y = \left\lfloor \frac{2y}{2} \right\rfloor = \Lsh 2y = \Lsh (x-1)$$

– When x is even, that is, $x = 2y$ for some $y \geq 1$:

$$(\uparrow x) - 1 = (\uparrow 2y) - 1 = \left\lceil \frac{2y}{2} \right\rceil - 1 = y - 1 = \left\lfloor \frac{2y-1}{2} \right\rfloor = \Lsh (2y-1) = \Lsh (x-1)$$

Thus, we have:

$$(\uparrow x) - 1 = \Lsh (x - 1) \tag{4}$$

Now, assume that (3) holds for some $k = l \geq 1$. Then,

$$\begin{aligned}
\uparrow^{l+1} x &= \uparrow (\uparrow^l x) && \text{(By the definition of } \uparrow^k\text{)} \\
&= \uparrow ((\Lsh^l (x-1)) + 1) && \text{(By the assumption)} \\
&= (\Lsh (\Lsh^l (x-1))) + 1 && \text{(By the equation 4)} \\
&= \Lsh^{l+1} (x-1) + 1 && \text{(By the definition of } \Lsh^k\text{)}
\end{aligned}$$

which concludes.

Now let us define the Merkle tree.

Definition 1. *A perfect binary tree T of height h is a* Merkle tree *[3], if the leaf node contains data, and the non-leaf node's value is the hash of its children's, i.e.,*

$$\forall 0 < l \leq h.\ \forall 0 < i \leq 2^{h-l}.\ T(l, i) = \mathsf{hash}(T(l-1, 2i-1), T(l-1, 2i)) \tag{5}$$

Let T_m be a partial Merkle tree up-to m *whose first m leafs contain data and the other leafs are zero, i.e.,*

$$T_m(0, i) = 0 \quad \textit{for all } m < i \leq 2^h \tag{6}$$

Let Z be the zero Merkle tree *whose leafs are all zero, i.e., $Z(0, i) = 0$ for all $0 < i \leq 2^h$. That is, $Z = T_0$. Since all nodes at the same level have the same value in Z, we write $Z(l)$ to denote the value at the level l, i.e., $Z(l) = Z(l, i)$ for any $0 < i \leq 2^{h-l}$.*

Now we formulate the relationship between the partial Merkle trees.

Lemma 2. *Let T_m be a partial Merkle tree up-to $m > 0$ of height h, and let T_{m-1} be another partial Merkle tree up-to $m - 1$ of the same height. Suppose their leafs agree up to $m-1$, that is, $T_{m-1}(0, i) = T_m(0, i)$ for all $1 \leq i \leq m-1$. Then, for all $0 \leq l \leq h$, and $1 \leq i \leq 2^{h-l}$,*

$$T_{m-1}(l, i) = T_m(l, i) \quad \textit{when } i \neq \uparrow^l m \tag{7}$$

Proof. Let us prove by induction on l. When $l = 0$, we immediately have $T_{m-1}(0, i) = T_m(0, i)$ for any $i \neq m$ by the premise and the equation (6). Now, assume that (7) holds for some $l = k$. Then by the equation 5, we have $T_{m-1}(k+1, i) = T_m(k+1, i)$ for any $i \neq \uparrow (\uparrow^k m) = \uparrow^{k+1} m$, which concludes.

Lemma 2 induces an incremental Merkle tree insertion algorithm [5].

Corollary 1. *T_m can be constructed from T_{m-1} by computing only $\{T_m(k, \uparrow^k m)\}_k$, the chain from the new leaf, $T_m(0, m)$, to the root.*

Proof. By Lemma 2.

Let us formulate more properties of a partial Merkle tree.

Lemma 3. *Let T_m be a partial Merkle tree up-to m of height h, and Z be the zero Merkle tree of the same height. Then, for all $0 \leq l \leq h$, and $1 \leq i \leq 2^{h-l}$,*

$$T_m(l, i) = Z(l) \quad \textit{when } i > \uparrow^l m \tag{8}$$

Proof. Let us prove by induction on l. When $l = 0$, we immediately have $T_m(0, i) = Z(0) = 0$ for any $m < i \leq 2^h$ by the equation (6). Now, assume that (8) holds for some $0 \leq l = k < h$. First, for any $i \geq (\uparrow^{k+1} m) + 1$, we have:

$$2i - 1 \geq (2 \uparrow^{k+1} m) + 1 = 2 \left\lceil \frac{\uparrow^k m}{2} \right\rceil + 1 \geq 2 \frac{\uparrow^k m}{2} + 1 = (\uparrow^k m) + 1 \tag{9}$$

Then, for any $\uparrow^{k+1} m < i \leq 2^{h-(k+1)}$, we have:

$$\begin{aligned} T_m(k+1, i) &= \mathsf{hash}(T_m(k, 2i-1), T_m(k, 2i)) && \text{(By the equation 5)} \\ &= \mathsf{hash}(Z(k), Z(k)) && \text{(By the equations 8 and 9)} \\ &= Z(k+1) && \text{(By the definition of } Z\text{)} \end{aligned}$$

which concludes.

Lemma 4. *A chain $\{T_m(k, \uparrow^k m)\}_k$ can be computed by using only two other chains, $\{T_{m-1}(k, \Lsh^k (m-1)))\}_k$ and $\{Z(k)\}_k$.*

Proof. We will construct the chain from the leaf, $T_m(0, m)$, which is given. Suppose we have constructed the chain up to $T_m(q, \uparrow^q m)$ for some $q > 0$ by using only two other sub-chains, $\{T_{m-1}(k, \Lsh^k (m-1))\}_{k=0}^{q-1}$ and $\{Z(k)\}_{k=0}^{q-1}$. Then, to construct $T_m(q+1, \uparrow^{q+1} m)$, we need the sibling of $T_m(q, \uparrow^q m)$, where we have two cases:

- Case $(\uparrow^q m)$ is odd. Then, we need the right-sibling $T_m(q, (\uparrow^q m) + 1)$, which is $Z(q)$ by Lemma 3.
- Case $(\uparrow^q m)$ is even. Then, we need the left-sibling $T_m(q, (\uparrow^q m) - 1)$, which is $T_m(q, \Lsh^q (m-1))$ by Lemma 1, which is in turn $T_{m-1}(q, \Lsh^q (m-1))$ by Lemma 2.

By the mathematical induction on k, we conclude.

Lemma 5. *Let $h =$ `TREE_HEIGHT`. For any integer $0 \leq m < 2^h$, the two chains $\{T_m(k, \Lsh^k m)\}_k$ and $\{T_{m+1}(k, \Lsh^k (m+1))\}_k$ always converge, that is, there exists unique $0 \leq l \leq h$ such that:*

$$(\Lsh^k m) + 1 = \Lsh^k (m+1) \textit{ is even for all } 0 \leq k < l \tag{10}$$

$$(\Lsh^k m) + 1 = \Lsh^k (m+1) \textit{ is odd for } k = l \tag{11}$$

$$\Lsh^k m = \Lsh^k (m+1) \textit{ for all } l < k \leq h \tag{12}$$

$$T_m(k, \Lsh^k m) = T_{m+1}(k, \Lsh^k (m+1)) \textit{ for all } l < k \leq h \tag{13}$$

Proof. The equation 12 follows from the equation 11, since for an odd integer x, $\Lsh (x-1) = \Lsh x$. Also, the equation 13 follows from Lemma 2, since $\uparrow^k (m+1) = (\Lsh^k m) + 1 \neq \Lsh^k m = \Lsh^k (m+1)$ by Lemma 1 and the equation 12. Thus, we only need to prove the unique existence of l satisfying (10) and (11). The existence of l is obvious since $1 \leq m+1 \leq 2^h$, and one can find the smallest l satisfying (10) and (11). Now, suppose there exist two different $l_1 < l_2$ satisfying (10) and (11). Then, $\Lsh^{l_1} (m+1)$ is odd since l_1 satisfies (11), while $\Lsh^{l_1} (m+1)$ is even since l_2 satisfies (10), which is contradiction, thus l is unique, and we conclude.

3 Verification of Incremental Merkle Tree Algorithm

Now we verify the correctness of the incremental Merkle tree algorithm given in Figure 2.

Lemma 6 (`init`). *Once `init` is executed, `zerohashes` denotes Z, that is,*

$$\texttt{zerohashes}[k] = Z(k) \tag{14}$$

for $0 \leq k <$ `TREE_HEIGHT`.

Proof. By the implementation of `init` and the definition of Z in Definition 1.

Lemma 7 (`deposit`). *Suppose that, before executing `deposit`, we have:*

$$\texttt{deposit\_count} = m < 2^{\texttt{TREE\_HEIGHT}} - 1 \tag{15}$$

$$\texttt{branch}[k] = T_m(k, \Lsh^k m) \quad \textit{if } \Lsh^k m \textit{ is odd} \tag{16}$$

Then, after executing $\texttt{deposit}(v)$, we have:

$$\texttt{deposit\_count}' = m + 1 \leq 2^{\texttt{TREE\_HEIGHT}} - 1 \tag{17}$$

$$\texttt{branch}'[k] = T_{m+1}(k, \Lsh^k (m+1)) \quad \textit{if } \Lsh^k (m+1) \textit{ is odd} \tag{18}$$

for any $0 \leq k <$ `TREE_HEIGHT`, where:

$$T_{m+1}(0, m+1) = v \tag{19}$$

```
# globals
zerohashes: int[TREE_HEIGHT] = {0} # zero array
branch:     int[TREE_HEIGHT] = {0} # zero array
deposit_count: int = 0  # max: 2^TREE_HEIGHT - 1

fun init() -> unit:
    i: int = 0
    while i < TREE_HEIGHT - 1:
        zerohashes[i+1] = hash(zerohashes[i], zerohashes[i])
        i += 1

fun deposit(value: int) -> unit:
    assert deposit_count < 2^TREE_HEIGHT - 1
    deposit_count += 1

    size: int = deposit_count
    i: int = 0
    while i < TREE_HEIGHT:
        if size % 2 == 1:
            break
        size /= 2
        i += 1

    j: int = 0
    while j < i:
        value = hash(branch[j], value)
        j += 1
    branch[i] = value

fun get_deposit_root() -> int:
    root: int = 0
    size: int = deposit_count
    h: int = 0
    while h < TREE_HEIGHT:
        if size % 2 == 1: # size is odd
            root = hash(branch[h], root)
        else:             # size is even
            root = hash(root, zerohashes[h])
        size /= 2
        h += 1
    return root
```

Fig. 2. Pseudocode implementation of the incremental Merkle tree algorithm employed in the deposit contract [1].

Proof. Let $h =$ `TREE_HEIGHT`. The equation 17 is obvious by the implementation of `deposit`. Let us prove the equation 18. Let l be the unique integer described in Lemma 5. We claim that `deposit` updates only `branch`$[l]$ to be $T_{m+1}(l, \Lsh^l (m+1))$. Then, for all $0 \leq k < l$, $\Lsh^k (m+1)$ is not odd. For $k = l$, we conclude by the aforementioned claim. For $l < k \leq h$, we conclude by the equation 13 and the fact that `branch`$[k]$ is not modified (by the aforementioned claim).

Now, let us prove the aforementioned claim. Since `branch` is updated only at line 28, we only need to prove $\texttt{i} = l$ and $\texttt{value} = T_{m+1}(l, \Lsh^l (m+1))$ at that point. We claim the following loop invariants of the two inner loops in `deposit`.

– First loop invariant at line 18.

$$\texttt{i} = i < \texttt{TREE\_HEIGHT} \tag{20}$$
$$\texttt{size} = \Lsh^i (m+1) \tag{21}$$
$$\Lsh^k (m+1) \text{ is even for any } 0 \leq k < i \tag{22}$$

Note that i cannot reach `TREE_HEIGHT`, since $(m+1) < 2^{\texttt{TREE\_HEIGHT}}$. It is trivial to show the above loop invariant. By the loop invariant, we have the following after the loop at line 23:

$$\texttt{i} = i < \texttt{TREE\_HEIGHT} \tag{23}$$
$$\texttt{size} = \Lsh^i (m+1) \text{ is odd} \tag{24}$$
$$\Lsh^k (m+1) \text{ is even for any } 0 \leq k < i \tag{25}$$

Moreover, by Lemma 5, we have $i = l$.

– Second loop invariant at line 25.

$$\texttt{i} = i \tag{26}$$
$$\texttt{j} = j \leq i \tag{27}$$
$$\texttt{value} = T_{m+1}(j, \Lsh^j (m+1)) \tag{28}$$

By the loop invariant, we have the following after the loop at line 28:

$$\texttt{i} = i \tag{29}$$
$$\texttt{j} = j = i \tag{30}$$
$$\texttt{value} = T_{m+1}(i, \Lsh^i (m+1)) \tag{31}$$

which concludes, since $i = l$.

Now we only need to prove the second loop invariant. First, at the beginning of the first iteration, we have $j = 0$ and $\texttt{value} = v = T_{m+1}(0, m+1)$ by (19), which satisfies the loop invariant. Now, assume that the invariant holds at the beginning of the j^{th} iteration. Here we need to consider only the case of $j < i$, since the loop immediately terminates when $j = i$. Then, $\mathtt{j}' = j + 1 \le i$, and:

$$
\begin{aligned}
T_{m+1}(j+1, \Lsh^{j+1} (m+1)) &= \mathsf{hash}(T_{m+1}(j, \Lsh^{j} m), T_{m+1}(j, \Lsh^{j} (m+1))) \\
&\qquad\qquad\qquad\qquad \text{(by Equation 10)} \\
&= \mathsf{hash}(T_m(j, \Lsh^{j} m), \mathtt{value}) \\
&\qquad\qquad \text{(by Lemmas 1 \& 2 and Equation 28)} \\
&= \mathsf{hash}(\mathtt{branch}[j], \mathtt{value}) \quad \text{(by Equations 16 \& 10)} \\
&= \mathtt{value}'
\end{aligned}
$$

Thus, the loop invariant holds at the beginning of the $(j+1)^{\text{th}}$ iteration as well, and we conclude.

Lemma 8 (Contract Invariant). *Let $m =$ `deposit_count`. Then, once `init` is executed, the following contract invariant holds. For all $0 \le k <$ `TREE_HEIGHT`,*

1. *`zerohashes`$[k] = Z(k)$*
2. *`branch`$[k] = T_m(k, \Lsh^k m)$ if $\Lsh^k m$ is odd*
3. *`deposit_count` $\le 2^{\mathtt{TREE\_HEIGHT}} - 1$*

Proof. Let us prove each invariant item.

1. By Lemma 6, and the fact that `zerohashes` is updated by only `init`.
2. By Lemma 7, and the fact that `branch` is updated by only `deposit`.
3. By the assertion of `deposit` (at line 13 of Figure 2), and the fact that `deposit_count` is updated by only `deposit`.

Lemma 9 (`get_deposit_root`). *The `get_deposit_root` function computes the chain $\{T_m(k, \uparrow^k (m+1))\}_k$ and returns the root $T_m(h, 1)$, given a Merkle tree T_m of height h, that is, `deposit_count` $= m < 2^h$ and `TREE_HEIGHT` $= h$ when `get_deposit_root` is invoked.*

Proof. We claim the following loop invariant at line 34, which suffices to conclude the main claim.

$$
\begin{aligned}
\mathtt{h} &= k \quad \text{where } 0 \le k \le h \\
\mathtt{size} &= \Lsh^k m \\
\mathtt{root} &= T_m(k, \uparrow^k (m+1))
\end{aligned}
$$

Now let us prove the above loop invariant claim by the mathematical induction on k. The base case ($k = 0$) is trivial, since $\Lsh^0 m = m$, $\uparrow^0 (m+1) = m+1$, and $T_m(0, m+1) = 0$ by Definition 1. Assume that the loop invariant holds for some $k = l$. Let $\mathtt{h}'$, $\mathtt{size}'$, and $\mathtt{root}'$ denote the values at the next iteration $k = l+1$. Obviously, we have $\mathtt{h}' = l + 1$ and $\mathtt{size}' = \Lsh^{l+1} m$. Also, we have $(\Lsh^l m) + 1 = \uparrow^l (m+1)$ by Lemma 1. Now, we have two cases:

– Case $\mathtt{size} = \Lsh^l m$ is odd. Then, $\uparrow^l (m+1)$ is even. Thus,

$$
\begin{aligned}
T_m(l+1, \uparrow^{l+1} (m+1)) &= \mathsf{hash}(T_m(l, \Lsh^l m), T_m(l, \uparrow^l (m+1))) \\
&= \mathsf{hash}(\mathtt{branch}[l], \mathtt{root}) \qquad \text{(by Lemma 8)} \\
&= \mathtt{root}'
\end{aligned}
$$

- Case $\texttt{size} = \Lsh^{l}\, m$ is even. Then, $\uparrow^{l} (m+1)$ is odd. Thus,

$$\begin{aligned}
T_m(l+1, \uparrow^{l+1} (m+1)) &= \mathsf{hash}(T_m(l, \uparrow^{l} (m+1)), T_m(l, (\uparrow^{l} (m+1)) + 1)) \\
&= \mathsf{hash}(\texttt{root}, Z(l)) && \text{(by Lemma 3)} \\
&= \mathsf{hash}(\texttt{root}, \texttt{zerohashes}[l]) && \text{(by Lemma 8)} \\
&= \texttt{root}'
\end{aligned}$$

Thus, we have $\texttt{root}' = T_m(l+1, \uparrow^{l+1} (m+1))$, which concludes.

3.1 Refactoring Suggestion

The `deposit` function can be refactored as follows:

```
fun deposit'(value: int) -> unit:
    assert deposit_count < 2^TREE_HEIGHT - 1
    deposit_count += 1
    size: int = deposit_count
    i: int = 0
    while i < TREE_HEIGHT - 1:
        if size % 2 == 1:
            break
        value = hash(branch[i], value)
        size /= 2
        i += 1
    branch[i] = value
```

The suggested refactoring is based on the following observations:

- Since `deposit` rejects when $\texttt{deposit\_count} \geq 2^{\texttt{TREE\_HEIGHT}} - 1$, the first loop of `deposit` cannot reach the last loop iteration, thus the loop bound can be safely decreased to $\texttt{TREE\_HEIGHT} - 1$.
- The two loops of `deposit` can be combined into a single loop, which is slightly more efficient.

References

1. Ethereum Foundation: Deposit Contract in Eth 2.0. https://github.com/ethereum/deposit_contract/blob/master/deposit_contract/contracts/validator_registration.v.py
2. Ethereum Foundation: Ethereum 2.0 Specifications. https://github.com/ethereum/eth2.0-specs
3. Merkle, R.C.: A digital signature based on a conventional encryption function. In: A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology. pp. 369–378. CRYPTO '87, Springer-Verlag, London, UK, UK (1988), http://dl.acm.org/citation.cfm?id=646752.704751
4. NIST: Perfect Binary Tree. https://xlinux.nist.gov/dads/HTML/perfectBinaryTree.html
5. Vitalik Buterin: Progressive Merkle Tree. https://github.com/ethereum/research/blob/master/beacon_chain_impl/progressive_merkle_tree.py