Merkle Tree Spec

Ann Weine

March 28, 2017

1 Data structure

type MerkleTreeElement with three **constructors**:

- MerkleTreeLeaf - the one for the leaf
- MerkleTreeHash - the one for the root that references a leaf (it has a hash of the corresponding leaf)
- MerkleTreeRoot - the one for the root.

The contrustructors have following **fields**:

- MerkleTreeRoot - hash, leftLeaf (reference to the left leaf), rightLeaf (reference to the right leaf, if it exists(NB: there are some cases when there is no right leaf, while there is the left root every time)), root (reference to the root, if it exists)
- MerkleTreeHashLeaf - leaf (reference to the corresponding leaf), hash (hash of the leaf), root (reference to the root)
- MerkleTreeLeaf - element, length (size of the element), root (reference to the root that corresponds to the leaf).

Data **types**: The following sub-structures are used

- for hashes - $uint8$ buffer with the size of $hash - function - output - size$
- for length - $uint32$

Sub-functions are used:

- hash function (that is used afterwards as hash(..). It takes a buffer of finite length and it length and returns the buffer of size $hash - function - output - size$. Hash function specification are out of the scope of the specification.
- concatenation function (that is used afterwards as '+'). It takes two buffers of size n each and returns the buffer of size 2n. The inner of the first buffer is placed on the $0..n-1$ addresses of buffer, the inner of the second buffer is placed on the $n..2n-1$ addresses of buffer respectively.

2 Properties

2.1 Correctness

We assume the data structure to be correct iff:

- for all MerkleTreeLeaf l

- $l.root.hash = hash(l)$.
- $l.root.leaf = l$

- for all MerkleTreeRoot l
  - $l.leftLeaf! = None$
  - $l.rightLeft = Some(MerkleTreeElementa)$ or $l.rightLeft = None$
  - $if l.rightLeaf = None$ then $l.hash = hash(l.leftLeaf)$ else $l.hash = hash(l.leftLeaf + l.rightLeaf)$