SkipList specification

Ann Weine

March 31, 2017

1 Data structure

A skiplist is an ordered data structure. It means that each element is more or equal than the previous one. To put the strictly more condition is currenly under the discussion. It is needed for the skipList with the possibility of having same elements more than once.
This trick could be done as usual skip list + addition to the leaf with the list of leafs of same value.
type SkipList with two **constructors**:

- Mk - the one for the element of skipList
- MkRoot - the one for the root of skipList

The contrustructors have the following **fields**:

- Mk - value(the value to be stored in datastructure, the one that could be compared with another value of the such type), levels - the number of links to other skipList (it was added for simplicity and in future will be deleted), list of links to other skipLists, hash of the value
- MkRoot - empty constructor

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

- For the values - any type that could be compared and built an ordered relations
- for hashes - $uint8$ buffer with the size of $hash - function - output - size$
- for length - $Fstar.Uint32.t$

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.

2 Methods over the structure

2.1 Existance check

Due to the properties of check list, search of the element is done with small complexity.
A result of the search will return the element skipList in case of the element is found and None otherwise.

2.2 Insert

Due to the fact the in our current version we're currently using pure style, addition of the element is supposed to be consuming due to the fact that we need to re-build the tree each time. This approach was chosen to simplify the proofs and could be adopted further.
In such a manner, addition of the elements consists on two steps:

- Going through the whole structure to create a list with references and finding the place for the new element.
- Updating the whole tree.

2.3 Delete

Due to the fact that the structure is append-only, there is no need in delete implementation.

2.4 Hashing

To be able to prove non-repudiation of the structure, each leaf will consists on the information about it and hash of the leaf. For future secrecy this information can further be updated to the signature of the list element.

2.5 Back-track of the skiplist

Due to the fact that skiplist could be used at the linked list, we suppose to be efficient to implement also the possibility of backtracking of the elements. In other words the structure will have the link to the previous element.