Practical Verification of QuadTrees

Jonathan Brouwer (jtbrouwer@student.tudelft.nl) Delft University of Technology

1. Introduction

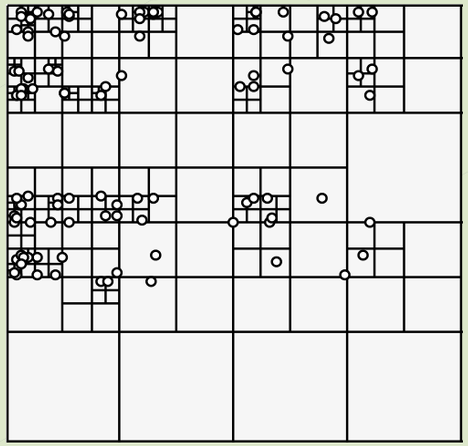
QuadTree is a Haskell library. This paper aims to rewrite this in Agda, so it can be formally verified using Curry Howard. It can then be compiled back to Haskell using Agda2hs.

```
takesGtFive : (n : Nat)
            -> IsTrue (n > 5) -> ?
```

Can agda2hs be used to produce a verified implementation of the QuadTree library?

2. QuadTrees

QuadTrees are used for storing two-dimensional information in a functional style. They consist of a size and a root quadrant. Each quadrant is either a Leaf, or a Node consisting of 4 sub-quadrants.



3. Implementation

How was the QuadTree library re-implemented?

- Translating the code
- Agda does not allow non-termination
- Agda does not have escape latches
- Agda2hs modifications needed

4. Verification

How was the QuadTree library verified?

- Find properties to prove, using techniques from the “Ready, set, verify!” paper
- Depending on the type of property, verify them in a certain way
- To reduce the time needed:
  - Postulate theorems about libraries
  - Use automatic proof search
  - First prove invariants and preconditions, then post-conditions.

5. Conclusions

The library was successfully verified!
Verified 2 invariants, 3 preconditions and 4 post-conditions. Still, quite a lot of effort! Whether it is worth it depends on the situation.

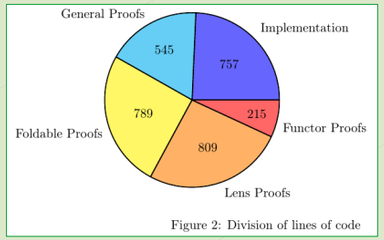


Figure 2: Division of lines of code

Invariants

Invariants are proven by adding the proof as an implicit constructor argument. To verify that a quadrant is compressed (no identical leafs) and has a certain depth, we can use:

```
data VQuadrant (t : Set) {dep : Nat} : Set where
  CVQuadrant : (qd : Quadrant t)
             -> {.(IsTrue (depth qd <= dep && isCompressed qd))}
             -> VQuadrant t {dep}
```

Preconditions

Preconditions are proven by adding the proofs as implicit arguments to the function. To verify that the location given to getLocation is in the QuadTree, one can use:

```
getLocation : (loc : Nat × Nat) -> {dep : Nat}
    -> (qt : QuadTree t)
    -> {.( IsTrue (isInsideQuadTree loc qt) )} -> t
```

Alternatively, we can pass in a datatype with an invariant. This getLeaf function only takes a leaf as input

```
getLeaf : (VQuadrant t {0}) -> t
```

Postconditions

Postconditions are proven as separate functions.
For example, this is a proof that this function returns a number greater than 5.

```
gt5 : Bool -> Nat
gt5 _ = 42
```

```
gt5-is-gt5 : (b : Bool)
  -> IsTrue (gt5 b > 5)
```