Verified functional data structures and algorithms in Agda

-Progress Report-

Razvan Kusztos, rek43@cam.ac.uk

Supervisor: Dr. Timothy Griffin
Director of studies: Chris Hadley
Overseers: Dr. Peter Sewell,
Dr. Markus Kuhn

The project is evolving smoothly, according to the plan. A few changes were necessary due to finding new resources, some limitations of Agda, and the desire to generalise, even though most of the work consists of various modifications and proofs over the Finger Tree[1], the concepts are valid for other datastructures.
Finger Trees are an example of a nested datatype[2], which is the reason why it has caught my interest to start with. Inductive definitions can be written, although the process is cumbersome. However, flattened views can be implemented, giving a neater approach.
In this period, I have implemented 4 versions of the finger tree, with associated algorithms. The first one is a slight alteration of a previous version found on github[3], making it runnable. The second one is the natural improvement that introduces measurement caches. The third one is a dependent version of the second one. Finally, the fourth one is a rewriting of an Isabelle implementation[4]. This was included because the Isabelle implementation redefines finger trees as a non-nested datatype, but keeps level invariants through repeated calculations. I thought it would be interesting, as a possible extension, to see the runtime of some operations on this version.
Running inductive definitions on flattened views becomes complicated because of Agda's termination check. The compiler cannot figure out whether the tail of this view is structurally smaller than the whole. In fact, my idea was that implementing a dependent version of the finger tree will sort this problem. Although it provides a first step, it requires some additional machinery.
Unfortunately, the 'with' construct in Agda, inspired by the 'View from the left' [5] doesn't interact well with the termination checker and rejects many correct implementations, examples of which are provided with respect to simpler datatypes. The standard library provides a Well-Founded induction framework, that allows proofs about whether the 'less than' relation is well founded, constructing a wrapped datatype which is structurally recursive. I believe that by reimplementing this framework to work with an indexed relation (for the dependent case), I can finally overcome the dependency hell[6]. Alternatively, the problem could be overcome by using the delay monad[7] to express future computations – although complicating the project.
Lastly, I have started to write the Introduction and Preparation chapters. I believe that since this is quite a niche topic, a carefully written introduction with plenty of motivating examples is necessary.
The evaluation phase plans on running benchmarks to compare the dependent and non-dependent implementations of the FingerTree, comparing various implementations of the Random Access Sequence, and contain proofs of properties that the variants kept by the nested type (full 2-3 tree) as well as the dependent typing (measurement) enable. A qualitative description of what programming with dependent types is like, limitations of Agda, and complications that arise will also be included.

1 *Finger-Trees : A Simple General Purpose Data Structure, Ralf Hinze and Ross Paterson, 2006*
2 *Nested Datatypes, Richard Bird and Lambert Meertens*
3 *https://github.com/fkettelhoit/agda-fingertrees*
4 *https://www21.in.tum.de/~lammich/autoref/Refine/FingerTree.html*
5 *The view from the left, Conor McBride and James McKinna*
6 *Programming FingerTrees in Coq, Matthiew Sozeau*
7 *Normalization by Evaluation in the Delay Monad, Andreas Abel and James Chapman*