Razvan Kusztos

Verified functional datastructures in Agda

Diploma in Computer Science

Girton College

January 16, 2017

Proforma

Name:	**Razvan Kusztos**
College:	**Girton College**
Project Title:	**Verified functional datasturcures and algo**
Examination:	**Part II Project**
Word Count:	**0**[1] **(well less than the 12000 limit)**
Project Originator:	Dr Timothy Griffin
Supervisor:	Dr Timothy Griffin

[1]This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

Declaration

I, [Name] of [College], being a candidate for Part II of the Computer Science Tripos [or the Diploma in Computer Science], hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed [signature]

Date [date]

Contents

List of Figures

Acknowledgements

This document owes much to an earlier version written by Simon Moore [**?**]. His help, encouragement and advice was greatly appreciated.

Chapter 1

Introduction

1.1 Functional Datastructures

There has always been an imballance between the use of functional programming versus imperative programming both in industry, as well as in terms of available resource. Functional programming has often been ruled out in the past because of it running slow, or simply because it was stigmatised as belonging into academia, regardless of its properties [**?**]. However, this paradigm is being introduced now at the forefront of business development. This is because of the persistency^0 of data-structures, useful for the ever-present multicore environment. The reliablity aspect, formal verification and keeping runtime errors to a minimum have also been relevant. The issue of the runtime speed has been addressed gracefully by Okasaki [**?**], which, introduced a reusable concept that can help designers build efficient data structures: the implicit recursive slowdown.

1.2 Dependent Typing

An important breakthrough in writing verifiably correct code is the introduction of dependent types.[**?**] In this setting the distinction between types and values becomes blurry, allowing us to define types that depend on values.

A good practical motivation is performing the sum of two vectors. The usual programming paradigm would be (in pseudocode):

```
def sum (l1, l2):
  if (l1.length != l2.length)
    raise ListsNotEqualException;
  ...
```

This can cause a runtime error and arguably disrupts the logical flow of the program. In a program that supports dependent types, we can construct lists that are both parametrized by a variable (as in the usual polymorphic programming), but also ’indexed’.

The usual definiton of lists would be (in Agda - but easilty and functional language)

```
  data List (A : Set) : Set where
    nil : List A
    _∷_ : A → List A → List A
```

Compare this with the dependent definition:

```
  data Vec (A : Set) : ℕ → Set where
    nil : Vec A 0
    _∷_ : ∀ {n : ℕ} → A → Vec A n → Vec A (n + 1)
```

This allows us to write functions that require a ’proof’ that the two vectors are equal.

```
  sum : ∀ {n : ℕ} → Vec ℕ n → Vec ℕ n → Vec ℕ n
  sum xs ys = ?
```

If this is not obvious from the context, the program will not type check. The developer is forced to only write correct programs.

Further details about agda syntax will be provided in section (Introduction to Agda)

1.3 Nested Types

Another way of maintaing invariants throughout the program is the trick or ’irregular’ or ’nested’ datatypes. They allow forcing strong structural invariants on the datastructure and have gained interest as they have interesting practical properties.

[**?**] They introduce some difficulties in what concerns inductive proofs, fact which will be covered and discussed in section (TODO implementation/section$_nest_example).$

The motivation of this concepts for this dissertation is that it allows maintaining the property of a full tree with labels at its leafs only.

```
data BinTree (A : Set) : Set where
  empty : BinTree A
  single : A → BinTree A
  deep :  BinTree (Node A) → BinTree A
```

Where Node is simply:

```
data Node (A : Set) : Set where
  node : A → A → Node A
```

It can be seen from the declaration that the structure will be forced to be a sequence of deep constructors, followed by either a single (Node^n A) or an empty constructor. The number of elements is forced therefore to be a power of two (2^n) making it equivalent with the leafs of a full binary tree.

This dissertation is concerned with 2-3 trees, the basis for the FingerTrees, which will be studied in detail in the Implementation section. They are an example to show arising problems when proving properties of nested and dependent typed structures, what limits are imposed and how some of them can be overcome.

1.4 Introduction to Agda

Agda is a dependently typed programming language, developed in the spirit of Haskell, kept as simple as possible [**?**]. All the previous examples are written in Agda, and their syntax and the newly introduced syntax will be described as we go on. Along other programming languages like Coq [**?**] or Isabelle [**?**], Agda is used as an interactive (or automatic) theorem prover. What makes it different from the two previously mentioned system is the ability to write the code and the proofs in the same environment. Its relative simplicity also motivated its use in this project.

1.5 Overview of the files

1.6 Building the document

1.6.1 The makefile

To simplify the calls to `latex` and `bibtex`, a makefile has been provided, see Appendix It provides the following facilities:

- `make`
  Display help information.

Chapter 2

Preparation

This chapter is empty!

Chapter 3

Implementation

3.1 Verbatim text

Verbatim text can be included using `\begin{verbatim}` and `\end{verbatim}`. I normally use a slightly smaller font and often squeeze the lines a little closer together, as in:

```
GET "libhdr"

GLOBAL { count:200; all  }

LET try(ld, row, rd) BE TEST row=all
                        THEN count := count + 1
                        ELSE { LET poss = all & ~(ld | row | rd)
                               UNTIL poss=0 DO
                               { LET p = poss & -poss
                                 poss := poss - p
                                 try(ld+p << 1, row+p, rd+p >> 1)
                               }
                             }
LET start() = VALOF
{ all := 1
  FOR i = 1 TO 12 DO
  { count := 0
    try(0, 0, 0)
    writef("Number of solutions to %i2-queens is %i5*n", i, count)
    all := 2*all + 1
  }
  RESULTIS 0
}
```

3.2 Tables

Here is a simple example[1] of a table.

Left Justified	Centred	Right Justified
First	A	XXX
Second	AA	XX
Last	AAA	X

There is another example table in the proforma.

3.3 Simple diagrams

[1] A footnote

Chapter 4

Evaluation

4.1 Printing and binding

If you have access to a laser printer that can print on two sides, you can use it to print two copies of your dissertation and then get them bound by the Computer Laboratory Bookshop. Otherwise, print your dissertation single sided and get the Bookshop to copy and bind it double sided.

Better printing quality can sometimes be obtained by giving the Bookshop an MSDOS 1.44 Mbyte 3.5” floppy disc containing the Postscript form of your dissertation. If the file is too large a compressed version with `zip` but not `gnuzip` nor `compress` is acceptable. However they prefer the uncompressed form if possible. From my experience I do not recommend this method.

4.1.1 Things to note

- Ensure that there are the correct number of blank pages inserted so that each double sided page has a front and a back. So, for example, the title page must be followed by an absolutely blank page (not even a page number).
- Submitted postscript introduces more potential problems. Therefore you must either allow two iterations of the binding process (once in a digital form, falling back to a second, paper, submission if necessary) or submit both paper and electronic versions.

- There may be unexpected problems with fonts.

4.2 Further information

See the Computer Lab’s world wide web pages at URL:
`http://www.cl.cam.ac.uk/TeXdoc/TeXdocs.html`

Chapter 5

Conclusion

I hope that this rough guide to writing a dissertation is LaTeX has been helpful and saved you time.