Tamás Kispéter

Monadic Concurrency in OCaml

Part II in Computer Science

Churchill College

March 4, 2014

Proforma

Name:	**Tamás Kispéter**
College:	**Churchill College**
Project Title:	**Monadic Concurrency in OCaml**
Examination:	**Part II in Computer Science, July 2014**
Word Count:	**1587**[1] **(well less than the 12000 limit)**
Project Originator:	Tamás Kispéter
Supervisor:	Jeremy Yallop

Original Aims of the Project

To write an OCaml framework for lightweight threading. This framework should be defined from basic semantics and have these semantics represented in a theorem prover setting for verification. The verification should include proofs of basic monadic laws. This theorem prover representation should be extracted to OCaml where the extracted code should be as faithful to the representation as possible. The extracted code should be able to run OCaml code concurrently.

Work Completed

All that has been completed appears in this dissertation.

Special Difficulties

Learning how to incorporate encapulated postscript into a LaTeX document on both CUS and Thor.

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

Declaration

I, Tamás Kispéter of Churchill College, being a candidate for Part II of the Computer Science Tripos, 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 March 4, 2014

Contents

List of Figures

Acknowledgements

Chapter 1

Introduction

The goal of this project is to build a concurrency framework for OCaml. This framework will be designed with correctness in mind: developing the well defined semantics, modelled in a proof assistant and finally extracted to actual code. This project aims to be a verifiable reference implementation.

1.1 Overview of concurrency

Concurrency is the concept of more than one thread of execution can make progress in the same time period. A particular form of concurrency is parallelism, when threads physically run simultaneously.

Concurrent computation has became the norm for most applications with the rise of faster systems often with multiple cores. This can be exploited on multiple levels ranging from hardware supported instruction and thread level parallelism to software based heavy and lightweight models.

This project aims to model lightweight, cooperative concurrency. No threads are exposed to the underlying operating system or hardware. This means that blocking operations on the process level will still block all internal threads. The threads themselves expose the points of possible interleaving.

The issue of concurrency occurs in most general programming languages. However, functional languages often both less developed in this area and also fit well with concurrency without races for mutable data structures. Functional languages that have both actual industrial applications and large set of features are of particular interest. These languages include OCaml and Haskell. This project focuses on OCaml.

1.2 Overview of OCaml

OCaml is a high level programming language. It combines functional, object-oriented and imperative paradigms. used in large scale industrial and academic projects where speed and correctness are of utmost importance. It uses one of the most powerful type systems and inference available to make efficient and correct software engineering possible.

1.3 Current implementations of a concurrency framework in OCaml

There are two very successful monadic concurrency frameworks, LWT[2] and Async[24]. They both provide the primitives for concurrent development in OCaml, however neither is supported by a clear semantic description. This is because their main focus was ease of use and speed.

LWT, the lightweight cooperative threading library[25] was designed as an open source framework entirely written in OCaml in a monadic style. It was successfully used in several large projects including the Unison file synchroniser and the Ocsigen Web server. This library includes many primitives to provide a feature rich framework, including primitives for thread creation, composition and cancel, thread local storage and support for various synchronisation techniques.

```
1 let x = 3;;
```

Async is an open source concurrency library for OCaml developed by Jane Street. Unlike LWT the basic semantics are designed with promise monads in mind. *Some more goes here*

1.4 Motivation

With these implementations in mind the motivation of the project is to investigate the lack of certified implementation of a concurrency framework. Verified concurrent systems have been researched for languages like C[21], C++ and Java[18], but not yet for OCaml.

1.5 Semantics of concurrency

There have been a lot of work on how exactly to formulate the semantics of concurrent and distributed systems. Some of the most common models for lightweight concurrency[10] are captured[11] and delimited[14] continuations[22], trampolined style[12], continuation monads[9], promise monads[17] and event based programming(used in the OCamlNet[23] project). This work focuses on the continuation monad style.

A monad[13] in functional programming is a construct to structure computations that are somehow "sequenced" together. A monad requires two operations, >>= and ret, which are related by the monadic associativity rules.

The continuation monad consists of a monadic type $\alpha\,\mathbf{con}$, where α is a type parameter describing the type of operation enclosed and three key operations

- >>=, also known as bind. The operation sequences two operations one after the other. More formally it has a type $\alpha\,\mathbf{con} \to (\alpha \to \beta\,\mathbf{con}) \to \beta\,\mathbf{con}$ and
- ret, also known as return. It has type $\alpha \to \alpha\,\mathbf{con}$ and simply evaluates its parameter and boxes up the result.

The third operation is fork, which is the way to spawn new threads (two in this particular case). The approach taken in this work is to have fork take two arguments, evaluate the two paths concurrently. The concurrency is achieved by reducing each path step by step based on some scheduler. When either path reduced to a value it returns a boxed up pair of results.

These results can be both values, or a value and a boxed up computation (that is, the partially reduced other path).

Therefore the signature of fork is slightly more complicated

$$\mathbf{fork} : \alpha\,\mathbf{con} \to \beta\,\mathbf{con} \to ((\alpha * \beta) + (\alpha * \beta\,\mathbf{con}) + (\alpha\,\mathbf{con} * \beta))\,\mathbf{con}$$

Where + refers to sum types.

1.6 Semantics to logic

These semantics can be modelled in logic, in particular logics used by proof assistants. Mechanisation of semantics in proof assistant has been accepted as a way of formal verification of properties about said semantics [6, 8, 7, 15]. Most often used proof assistants include Coq [4], HOL and Isabelle. There has been much effort to make this mechanisation as easy as possible with tools like Ott[19],

which enable working semanticists to describe their language in a simple, ascii-art like style while versatile in destination languages, including the above mentioned proof assistants, LaTeX and some direct OCaml representations.

This project was developed with the Coq proof assistant in mind.

Coq is formal proof assistant with a mathematical higher-level language called *Gallina*, based around the Calculus of Inductive Constructions, that can be used to define functions and predicates, state, formally prove and machine check mathematical theorems and extract certified programs to high level languages like Haskell and OCaml.

1.7 Logic to actual code

This logical representation is more amenable to proofs than to actual execution and the evaluation with respect to other implementations, which is why much research has gone into extraction of these representations as certified programs in functional programming languages including OCaml [16] and Haskell.

Inductive sets often map simply to data structures of type constructors and constructive proofs to computable functions.

Chapter 2

Preparation

2.1 Design of monadic semantics

2.2 Design of concurrent semantics

2.3 Tools

2.3.1 Ott

2.3.2 Coq

2.3.3 OCaml

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

Simple diagrams can be written directly in LaTeX. For example, see figure 3.1 on page 9 and see figure 3.2 on page 9.

3.4 Adding more complicated graphics

The use of LaTeX format can be tedious and it is often better to use encapsulated postscript to represent complicated graphics. Figure 3.3 and 3.5 on page 10 are examples. The second figure was drawn using `xfig` and exported in `.eps` format. This is my recommended way of drawing all diagrams.

[1] A footnote

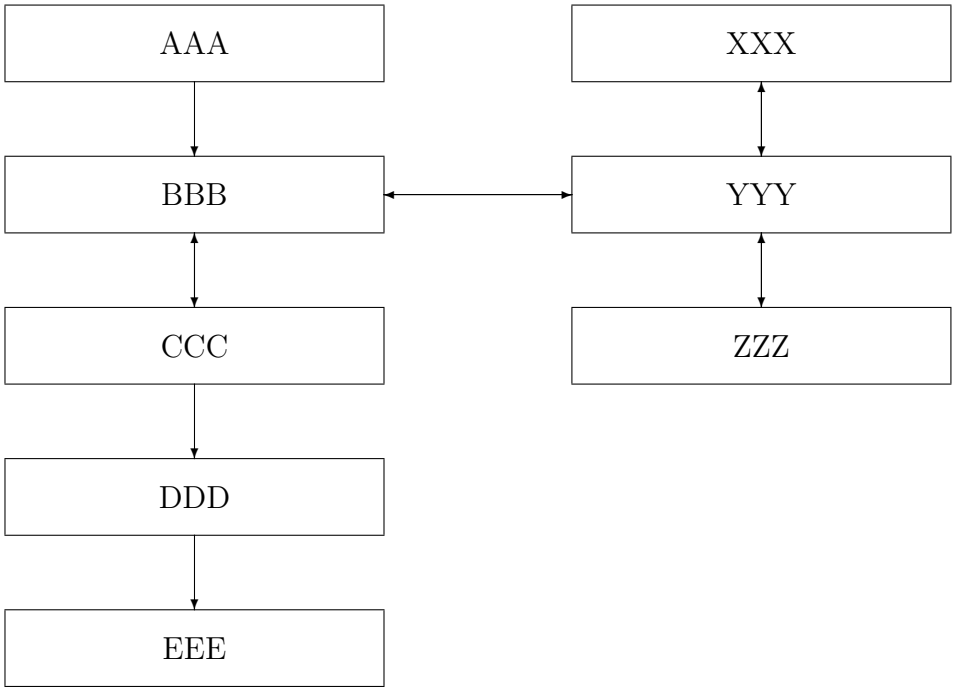

Figure 3.1: A picture composed of boxes and vectors.

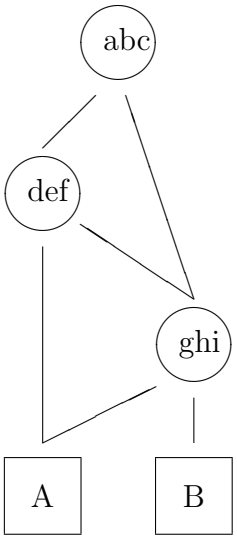

Figure 3.2: A diagram composed of circles, lines and boxes.

Figure 3.3: Example figure using encapsulated postscript

Figure 3.4: Example figure where a picture can be pasted in

Figure 3.5: Example diagram drawn using `xfig`

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.

Bibliography

[1] Lem, a tool for lightweight executable mathematics. `http://www.cs.kent.ac.uk/people/staff/sao/lem/`.

[2] Lwt, lightweight threading library. `http://ocsigen.org/lwt/`.

[3] Ocaml. `http://ocaml.org/`.

[4] Ott, a tool for writing definitions of programming languages and calculi. `http://coq.inria.fr/`.

[5] Ott, a tool for writing definitions of programming languages and calculi. `http://www.cl.cam.ac.uk/~so294/ocaml/`, 2008.

[6] Nick Benton and Vasileios Koutavas. A mechanized bisimulation for the nu-calculus. *Higher-Order and Symbolic Computation (to appear, 2013)*, 2008.

[7] Sandrine Blazy, Zaynah Dargaye, and Xavier Leroy. Formal verification of a c compiler front-end. In *FM 2006: Formal Methods*, pages 460–475. Springer, 2006.

[8] Sandrine Blazy and Xavier Leroy. Mechanized semantics for the clight subset of the c language. *Journal of Automated Reasoning*, 43(3):263–288, 2009.

[9] Koen Claessen. Functional pearls: A poor man's concurrency monad, 1999.

[10] Christophe Deleuze. Light weight concurrency in ocaml: Continuations, monads, promises, events.

[11] Daniel P Friedman. Applications of continuations. In *Proceedings of the ACM Conference on Principles of Programming Languages*, 1988.

[12] Steven E Ganz, Daniel P Friedman, and Mitchell Wand. Trampolined style. In *ACM SIGPLAN Notices*, volume 34, pages 18–27. ACM, 1999.

[13] CAR Hoareetal. Tackling the awkward squad: monadic input/output, concurrency, exceptions, and foreign-language calls in haskell. *Engineering theories of software construction*, 180:47, 2001.

[14] Oleg Kiselyov. Delimited control in ocaml, abstractly and concretely: System description. In *Functional and Logic Programming*, pages 304–320. Springer, 2010.

[15] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[16] Pierre Letouzey. Extraction in coq: An overview. In *Logic and Theory of Algorithms*, pages 359–369. Springer, 2008.

[17] Barbara Liskov and Liuba Shrira. *Promises: linguistic support for efficient asynchronous procedure calls in distributed systems*, volume 23. ACM, 1988.

[18] Andreas Lochbihler. *A Machine-Checked, Type-Safe Model of Java Concurrency: Language, Virtual Machine, Memory Model, and Verified Compiler*. KIT Scientific Publishing, 2012.

[19] Francesco Zappa Nardelli. Ott, a tool for writing definitions of programming languages and calculi. `http://www.cl.cam.ac.uk/~pes20/ott/`.

[20] Scott Owens. A sound semantics for ocaml light. In *Programming Languages and Systems*, pages 1–15. Springer, 2008.

[21] Jaroslav Ŝevčik, Viktor Vafeiadis, Francesco Zappa Nardelli, Suresh Jagannathan, and Peter Sewell. Relaxed-memory concurrency and verified compilation. In *ACM SIGPLAN Notices*, volume 46, pages 43–54. ACM, 2011.

[22] Chung-chieh Shan. Shift to control. In *Proceedings of the 5th workshop on Scheme and Functional Programming*, pages 99–107, 2004.

[23] Gerd Stolpmann. Ocamlnet. `http://projects.camlcity.org/projects/ocamlnet.html`.

[24] Jane Street. Async, open source concurrency library.

[25] Jérôme Vouillon. Lwt: a cooperative thread library. In *Proceedings of the 2008 ACM SIGPLAN workshop on ML*, pages 3–12. ACM, 2008.

Appendix A

Latex source

A.1 diss.tex

```
% The master copy of this demo dissertation is held on my filespace
% on the cl file serve (/homes/mr/teaching/demodissert/)

% Last updated by MR on 2 August 2001

\documentclass[12pt,twoside,notitlepage]{report}

\usepackage{a4}
\usepackage{verbatim}

\usepackage{url}
\input{epsf}                            % to allow postscript inclusions
% On thor and CUS read top of file:
%     /opt/TeX/lib/texmf/tex/dvips/epsf.sty
% On CL machines read:
%     /usr/lib/tex/macros/dvips/epsf.tex



\raggedbottom                           % try to avoid widows and orphans
\sloppy
\clubpenalty1000%
\widowpenalty1000%

\addtolength{\oddsidemargin}{6mm}       % adjust margins
\addtolength{\evensidemargin}{-8mm}

\renewcommand{\baselinestretch}{1.1}    % adjust line spacing to make
                                        % more readable

\usepackage{listings}
\usepackage{color}

\definecolor{mygreen}{rgb}{0,0.6,0}
\definecolor{mygray}{rgb}{0.5,0.5,0.5}
\definecolor{mymauve}{rgb}{0.58,0,0.82}
```

```
\lstset{ %
  backgroundcolor=\color{white},   % choose the background color; you must add \usepackage{color} or \usepackage{xcolor}
  basicstyle=\footnotesize,        % the size of the fonts that are used for the code
  breakatwhitespace=false,         % sets if automatic breaks should only happen at whitespace
  breaklines=true,                 % sets automatic line breaking
  captionpos=b,                    % sets the caption-position to bottom
  commentstyle=\color{mygreen},    % comment style
  deletekeywords={...},            % if you want to delete keywords from the given language
  escapeinside={\%*}{*)},          % if you want to add LaTeX within your code
  extendedchars=true,              % lets you use non-ASCII characters; for 8-bits encodings only, does not work with UTF-8
  frame=single,                    % adds a frame around the code
  keepspaces=true,                 % keeps spaces in text, useful for keeping indentation of code (possibly needs columns=flex
  keywordstyle=\color{blue},       % keyword style
  language=[Objective]Caml,        % the language of the code
  morekeywords={*,...},            % if you want to add more keywords to the set
  numbers=left,                    % where to put the line-numbers; possible values are (none, left, right)
  numbersep=5pt,                   % how far the line-numbers are from the code
  numberstyle=\tiny\color{mygray}, % the style that is used for the line-numbers
  rulecolor=\color{black},         % if not set, the frame-color may be changed on line-breaks within not-black text (e.g. com
  showspaces=false,                % show spaces everywhere adding particular underscores; it overrides 'showstringspaces'
  showstringspaces=false,          % underline spaces within strings only
  showtabs=false,                  % show tabs within strings adding particular underscores
  stepnumber=2,                    % the step between two line-numbers. If it's 1, each line will be numbered
  stringstyle=\color{mymauve},     % string literal style
  tabsize=2,                       % sets default tabsize to 2 spaces
  title=\lstname                   % show the filename of files included with \lstinputlisting; also try caption instead of ti
}

\begin{document}

\bibliographystyle{plain}


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Title


\pagestyle{empty}

\hfill{\LARGE \bf Tam\'as Kisp\'eter}

\vspace*{60mm}
\begin{center}
\Huge
{\bf Monadic Concurrency in OCaml} \\
\vspace*{5mm}
Part II in Computer Science \\
\vspace*{5mm}
Churchill College \\
\vspace*{5mm}
\today  % today's date
\end{center}

\cleardoublepage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Proforma, table of contents and list of figures

\setcounter{page}{1}
```

```
\pagenumbering{roman}
\pagestyle{plain}

\chapter*{Proforma}

{\large
\begin{tabular}{ll}
Name:               & \bf Tam\'as Kisp\'eter                    \\
College:            & \bf Churchill College                     \\
Project Title:      & \bf Monadic Concurrency in OCaml \\
Examination:        & \bf Part II in Computer Science, July 2014        \\
Word Count:         & \bf 1587\footnotemark[1]
(well less than the 12000 limit) \\
Project Originator: & Tam\'as Kisp\'eter                    \\
Supervisor:         & Jeremy Yallop                   \\
\end{tabular}
}
\footnotetext[1]{This word count was computed
by {\tt detex diss.tex | tr -cd '0-9A-Za-z $\tt\backslash$n' | wc -w}
}
\stepcounter{footnote}


\section*{Original Aims of the Project}

To write an OCaml framework for lightweight threading. This framework should be defined from basic semantics and have th


\section*{Work Completed}

All that has been completed appears in this dissertation.

\section*{Special Difficulties}

Learning how to incorporate encapulated postscript into a \LaTeX\
document on both CUS and Thor.

\newpage
\section*{Declaration}

I, Tam\'as Kisp\'eter of Churchill College, being a candidate for Part II of the Computer
Science Tripos, 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.

\bigskip
\leftline{Signed [signature]}

\medskip
\leftline{Date \today}

\cleardoublepage

\tableofcontents

\listoffigures
```

```
\newpage
\section*{Acknowledgements}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% now for the chapters

\cleardoublepage        % just to make sure before the page numbering
                        % is changed

\setcounter{page}{1}
\pagenumbering{arabic}
\pagestyle{headings}

\chapter{Introduction}

The goal of this project is to build a concurrency framework for OCaml. This framework will be designed with correctness in mi

\section{Overview of concurrency}
Concurrency is the concept of more than one thread of execution can make progress in the same time period. A particular form o

Concurrent computation has became the norm for most applications with the rise of faster systems often with multiple cores. Th

This project aims to model lightweight, cooperative concurrency. No threads are exposed to the underlying operating system or

The issue of concurrency occurs in most general programming languages. However, functional languages often both less developed

\section{Overview of OCaml}
OCaml is a high level programming language. It combines functional, object-oriented and imperative paradigms.  used in large s


\section{Current implementations of a concurrency framework in OCaml}
There are two very successful monadic concurrency frameworks, LWT\cite{LWT} and Async\cite{Async}. They both provide the primi

LWT, the lightweight cooperative threading library\cite{vouillon2008lwt} was designed as an open source framework entirely wri

\begin{lstlisting}
let x = 3;;
\end{lstlisting}

Async is an open source concurrency library for OCaml developed by Jane Street. Unlike LWT the basic semantics are designed wi
\textit{Some more goes here}

\section{Motivation}
With these implementations in mind the motivation of the project is to investigate the lack of certified implementation of a c

\section{Semantics of concurrency}

There have been a lot of work on how exactly to formulate the semantics of concurrent and distributed systems. Some of the mos

A monad\cite{hoareetal2001tackling} in functional programming is a construct to structure computations that are somehow "sequ


The continuation monad consists of a monadic type $ \alpha \, \textbf{con} $, where $ \alpha $ is a type parameter describing
\begin{itemize}
\item{$>>=$, also known as bind. The operation sequences two operations one after the other. More formally it has a type $ \al
\item{ret, also known as return. It has type $ \alpha \rightarrow \alpha \, \textbf{con} $ and simply evaluates its parameter
\end{itemize}

The third operation is fork, which is the way to spawn new threads (two in this particular case). The approach taken in this w
```

```
These results can be both values, or a value and a boxed up computation (that is, the partially reduced other path).

Therefore the signature of fork is slightly more complicated
\[ \textbf{fork} \, : \, \alpha \, \textbf{con} \rightarrow \beta \, \textbf{con} \rightarrow ((\alpha * \beta ) \, + \,
Where $ + $ refers to sum types.

\section{Semantics to logic}
These semantics can be modelled in logic, in particular logics used by proof assistants. Mechanisation of semantics in p

This project was developed with the Coq proof assistant in mind.

Coq is formal proof assistant with a mathematical higher-level language called \textit{Gallina}, based around the Calcul


\section{Logic to actual code}
This logical representation is more amenable to proofs than to actual execution and the evaluation with respect to other

Inductive sets often map simply to data structures of type constructors and constructive proofs to computable functions.

\chapter{Preparation}

\section{Design of monadic semantics}

\section{Design of concurrent semantics}

\section{Tools}
\subsection{Ott}
\subsection{Coq}
\subsection{OCaml}

\cleardoublepage
\chapter{Implementation}

\section{Verbatim text}

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

{\renewcommand{\baselinestretch}{0.8}\small\begin{verbatim}
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
}
\end{verbatim}
}

\section{Tables}

\begin{samepage}
Here is a simple example\footnote{A footnote} of a table.

\begin{center}
\begin{tabular}{l|c|r}
Left      & Centred & Right \\
Justified &         & Justified \\[3mm]
%\hline\\%[-2mm]
First     & A       & XXX \\
Second    & AA      & XX  \\
Last      & AAA     & X   \\
\end{tabular}
\end{center}

\noindent
There is another example table in the proforma.
\end{samepage}

\section{Simple diagrams}

Simple diagrams can be written directly in \LaTeX.  For example, see
figure~\ref{latexpic1} on page~\pageref{latexpic1} and see
figure~\ref{latexpic2} on page~\pageref{latexpic2}.

\begin{figure}
\setlength{\unitlength}{1mm}
\begin{center}
\begin{picture}(125,100)
\put(0,80){\framebox(50,10){AAA}}
\put(0,60){\framebox(50,10){BBB}}
\put(0,40){\framebox(50,10){CCC}}
\put(0,20){\framebox(50,10){DDD}}
\put(0,00){\framebox(50,10){EEE}}

\put(75,80){\framebox(50,10){XXX}}
\put(75,60){\framebox(50,10){YYY}}
\put(75,40){\framebox(50,10){ZZZ}}

\put(25,80){\vector(0,-1){10}}
\put(25,60){\vector(0,-1){10}}
\put(25,50){\vector(0,1){10}}
\put(25,40){\vector(0,-1){10}}
\put(25,20){\vector(0,-1){10}}

\put(100,80){\vector(0,-1){10}}
\put(100,70){\vector(0,1){10}}
\put(100,60){\vector(0,-1){10}}
\put(100,50){\vector(0,1){10}}
```

```
\put(50,65){\vector(1,0){25}}
\put(75,65){\vector(-1,0){25}}
\end{picture}
\end{center}
\caption{\label{latexpic1}A picture composed of boxes and vectors.}
\end{figure}

\begin{figure}
\setlength{\unitlength}{1mm}
\begin{center}

\begin{picture}(100,70)
\put(47,65){\circle{10}}
\put(45,64){abc}

\put(37,45){\circle{10}}
\put(37,51){\line(1,1){7}}
\put(35,44){def}

\put(57,25){\circle{10}}
\put(57,31){\line(-1,3){9}}
\put(57,31){\line(-3,2){15}}
\put(55,24){ghi}

\put(32,0){\framebox(10,10){A}}
\put(52,0){\framebox(10,10){B}}
\put(37,12){\line(0,1){26}}
\put(37,12){\line(2,1){15}}
\put(57,12){\line(0,2){6}}
\end{picture}

\end{center}
\caption{\label{latexpic2}A diagram composed of circles, lines and boxes.}
\end{figure}



\section{Adding more complicated graphics}

The use of \LaTeX\ format can be tedious and it is often better to use
encapsulated postscript to represent complicated graphics.
Figure~\ref{epsfig} and ~\ref{xfig} on page \pageref{xfig} are
examples. The second figure was drawn using {\tt xfig} and exported in
{\tt.eps} format. This is my recommended way of drawing all diagrams.


\begin{figure}[tbh]
%\centerline{\epsfbox{figs/cuarms.eps}}
\caption{\label{epsfig}Example figure using encapsulated postscript}
\end{figure}

\begin{figure}[tbh]
\vspace{4in}
\caption{\label{pastedfig}Example figure where a picture can be pasted in}
\end{figure}


\begin{figure}[tbh]
%\centerline{\epsfbox{figs/diagram.eps}}
```

```
\caption{\label{xfig}Example diagram drawn using {\tt xfig}}
\end{figure}



\cleardoublepage
\chapter{Evaluation}

\section{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 {\tt zip} but not {\tt gnuzip} nor {\tt
compress} is acceptable. However they prefer the uncompressed form if
possible. From my experience I do not recommend this method.

\subsection{Things to note}

\begin{itemize}
\item 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).

\item 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.

\item There may be unexpected problems with fonts.

\end{itemize}

\section{Further information}

See the Computer Lab's world wide web pages at URL:

{\tt http://www.cl.cam.ac.uk/TeXdoc/TeXdocs.html}


\cleardoublepage
\chapter{Conclusion}

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



\cleardoublepage
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the bibliography

\addcontentsline{toc}{chapter}{Bibliography}
\bibliography{refs}
\cleardoublepage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the appendices
\appendix

\chapter{Latex source}

\section{diss.tex}
{\scriptsize\verbatiminput{diss.tex}}

\section{proposal.tex}
{\scriptsize\verbatiminput{proposal.tex}}

\section{propbody.tex}
{\scriptsize\verbatiminput{propbody.tex}}



\cleardoublepage

\chapter{Makefile}

\section{\label{makefile}Makefile}
{\scriptsize\verbatiminput{makefile.txt}}

\section{refs.bib}
{\scriptsize\verbatiminput{refs.bib}}


\cleardoublepage

\chapter{Project Proposal}

\input{partIIproposal}

\end{document}
```

A.2 proposal.tex

A.3 propbody.tex

Appendix B

Makefile

B.1 Makefile

B.2 refs.bib

```
@MISC{OCaml,
 title = {OCaml},
 howpublished = {\url{http://ocaml.org/}}
}

@MISC{LWT,
 title = {LWT, Lightweight Threading library},
 howpublished = {\url{http://ocsigen.org/lwt/}}
}

@MISC{Async,
 title = {Async, open source concurrency library},
 author =  {Jane Street}
 howpublished = {\url{http://janestreet.github.io/}}
}
@MISC{Lem,
 title = {Lem, a tool for lightweight executable mathematics.},
 howpublished = {\url{http://www.cs.kent.ac.uk/people/staff/sao/lem/}}
}
@MISC{Ott,
  title = {Ott, a tool for writing definitions of programming languages and calculi},
  howpublished = {\url{http://www.cl.cam.ac.uk/~pes20/ott/}},
  author = "Francesco Zappa Nardelli"
  }

@MISC{Claessen99functionalpearls, author = {Koen Claessen}, title = {Functional Pearls: A Poor Man's Concurrency Monad},

@article{deleuzelight,
  title={Light Weight Concurrency in OCaml: Continuations, Monads, Promises, Events},
  author={Deleuze, Christophe}
}
```

```
@inproceedings{vouillon2008lwt,
  title={Lwt: a cooperative thread library},
  author={Vouillon, J{\'e}r{\^o}me},
  booktitle={Proceedings of the 2008 ACM SIGPLAN workshop on ML},
  pages={3--12},
  year={2008},
  organization={ACM}
}

@MISC{Coq,
  title = {Ott, a tool for writing definitions of programming languages and calculi},
  howpublished = {\url{http://coq.inria.fr/}}}

@MISC{OCamlLightWeb,
    title = {Ott, a tool for writing definitions of programming languages and calculi},
    howpublished = {\url{http://www.cl.cam.ac.uk/~so294/ocaml/}},
    year={2008}}


@incollection{OCamlLight,
  title={A sound semantics for OCaml light},
  author={Owens, Scott},
  booktitle={Programming Languages and Systems},
  pages={1--15},
  year={2008},
  publisher={Springer}
}

@article{benton2008mechanized,
  title={A mechanized bisimulation for the nu-calculus},
  author={Benton, Nick and Koutavas, Vasileios},
  journal={Higher-Order and Symbolic Computation (to appear, 2013)},
  year={2008}
}

@article{blazy2009mechanized,
  title={Mechanized semantics for the Clight subset of the C language},
  author={Blazy, Sandrine and Leroy, Xavier},
  journal={Journal of Automated Reasoning},
  volume={43},
  number={3},
  pages={263--288},
  year={2009},
  publisher={Springer}
}
@incollection{blazy2006formal,
  title={Formal verification of a C compiler front-end},
  author={Blazy, Sandrine and Dargaye, Zaynah and Leroy, Xavier},
  booktitle={FM 2006: Formal Methods},
  pages={460--475},
  year={2006},
  publisher={Springer}
}
@article{leroy2009formal,
  title={Formal verification of a realistic compiler},
  author={Leroy, Xavier},
  journal={Communications of the ACM},
  volume={52},
  number={7},
```

```
  pages={107--115},
  year={2009},
  publisher={ACM}
}

@incollection{letouzey2008extraction,
  title={Extraction in coq: An overview},
  author={Letouzey, Pierre},
  booktitle={Logic and Theory of Algorithms},
  pages={359--369},
  year={2008},
  publisher={Springer}
}
@incollection{berger1993program,
  title={Program extraction from normalization proofs},
  author={Berger, Ulrich},
  booktitle={Typed Lambda Calculi and Applications},
  pages={91--106},
  year={1993},
  publisher={Springer}
}
@inproceedings{berger1995program,
  title={Program extraction from classical proofs},
  author={Berger, Ulrich and Schwichtenberg, Helmut},
  booktitle={Logic and Computational Complexity},
  pages={77--97},
  year={1995},
  organization={Springer}
}
@incollection{letouzey2003new,
  title={A new extraction for Coq},
  author={Letouzey, Pierre},
  booktitle={Types for proofs and programs},
  pages={200--219},
  year={2003},
  publisher={Springer}
}
@incollection{delahaye2007extracting,
  title={Extracting purely functional contents from logical inductive types},
  author={Delahaye, David and Dubois, Catherine and {\'E}tienne, Jean-Fr{\'e}d{\'e}ric},
  booktitle={Theorem Proving in Higher Order Logics},
  pages={70--85},
  year={2007},
  publisher={Springer}
}

@incollection{fournet2003jocaml,
  title={JoCaml: A language for concurrent distributed and mobile programming},
  author={Fournet, C{\'e}dric and Le Fessant, Fabrice and Maranget, Luc and Schmitt, Alan},
  booktitle={Advanced Functional Programming},
  pages={129--158},
  year={2003},
  publisher={Springer}
}

@inproceedings{shan2004shift,
  title={Shift to control},
  author={Shan, Chung-chieh},
  booktitle={Proceedings of the 5th workshop on Scheme and Functional Programming},
```

```
  pages={99--107},
  year={2004}
}

@inproceedings{friedman1988applications,
  title={Applications of continuations},
  author={Friedman, Daniel P},
  booktitle={Proceedings of the ACM Conference on Principles of Programming Languages},
  year={1988}
}

@incollection{kiselyov2010delimited,
  title={Delimited control in OCaml, abstractly and concretely: System description},
  author={Kiselyov, Oleg},
  booktitle={Functional and Logic Programming},
  pages={304--320},
  year={2010},
  publisher={Springer}
}

@inproceedings{ganz1999trampolined,
  title={Trampolined style},
  author={Ganz, Steven E and Friedman, Daniel P and Wand, Mitchell},
  booktitle={ACM SIGPLAN Notices},
  volume={34},
  number={9},
  pages={18--27},
  year={1999},
  organization={ACM}
}

@article{hoareetal2001tackling,
  title={Tackling the awkward squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell},
  author={Hoareetal, CAR},
  journal={Engineering theories of software construction},
  volume={180},
  pages={47},
  year={2001},
  publisher={IOS Press}
}


@book{liskov1988promises,
  title={Promises: linguistic support for efficient asynchronous procedure calls in distributed systems},
  author={Liskov, Barbara and Shrira, Liuba},
  volume={23},
  number={7},
  year={1988},
  publisher={ACM}
}


@MISC{Ocamlnet,
 title = {OCamlNet},
 howpublished = {\url{http://projects.camlcity.org/projects/ocamlnet.html}},
 author = {Gerd Stolpmann}
}

@inproceedings{sevvcik2011relaxed,
```

```
  title={Relaxed-memory concurrency and verified compilation},
  author={{\^S}ev{\v{c}}ik, Jaroslav and Vafeiadis, Viktor and Zappa Nardelli, Francesco and Jagannathan, Suresh and Sew
  booktitle={ACM SIGPLAN Notices},
  volume={46},
  number={1},
  pages={43--54},
  year={2011},
  organization={ACM}
}

@book{lochbihler2012machine,
  title={A Machine-Checked, Type-Safe Model of Java Concurrency: Language, Virtual Machine, Memory Model, and Verified C
  author={Lochbihler, Andreas},
  year={2012},
  publisher={KIT Scientific Publishing}
}
```

Appendix C

Project Proposal

C.1 Introduction of work to be undertaken

With the rise of ubiquitous multiple core systems it is necessary for a working programmer to use concurrency to the greatest extent. However concurrent code has never been easy to write as human reasoning is often poorly equipped with the tools necessary to think about such systems. That is why it is essential for a programming language to provide safe and sound primitives to tackle this problem.

My project aims to do this in the OCaml[3] language by developing a lightweight cooperating threading framework that holds correctness as a core value. The functional nature allows the use of one of the most recent trends in languages popular in academia, monads, to be used for a correct implementation.

There have been two very successful frameworks, LWT[2] and Async[24] that both provided the primitives for concurrent development in OCaml however neither is supported by a clear semantic description as their main focus was ease of use and speed.

C.2 Description of starting point

My personal starting points are the courses ML under Windows (IA), Semantics of Programming Languages (IB), Logic and Proof (IB) and Concepts in Programming Languages (IB). Furthermore I have done extracurricular reading into semantics and typing and attended the Denotational Semantics (II) course in the past year.

The preparatory research period has to include familiarising myself with OCaml and the chosen specification and proof assistant tools.

C.3 Substance and structure

The project will consist of first creating a formal specification for a simple monad that has three main operations bind, return and choose. The behaviour of these operations will be specified in a current semantics tool like Lem[1] or Ott[19].

As large amount of research has gone into both monadic concurrency and implementations in OCaml, the project will draw inspiration from Claessen[9], Deleuze[10] and Vouillon[25].

Some atomic, blocking operations will also be specified including reading and writing to a console prompt or file to better illustrate the concurrency properties and make testing and evaluation possible.

This theory driven executable specification will be paired by a hand implementation and will be thoroughly checked against each other to ensure that both adhere to the desired semantics.

Both of these implementations will be then compared against the two current frameworks for simplicity and speed on various test cases.

If time allows, an extension will also be carried out on the theorem prover version of the specification to formally verify that the implementation is correct.

C.4 Criteria

For the project to be deemed a success the following items must be successfully completed.

1. A specification for a monadic concurrency framework must be designed in the format of a semantics tool.

2. This specification needs to be exported to a proof assistant and has a runnable OCaml version

3. Test cases must be written that can thoroughly check a concurrency framework

4. A hand implementation needs to be designed, implemented and tested against the specification

5. The implementations must be compared to the frameworks LWT and Async based on speed

6. The dissertation must be planned and written

In case the extension will also become viable then its success criterion is that there is a clear formal verification accompanying the automated theorem prover version of the specification.

C.5 Timetable

The project will be split into two week packages

C.5.1 Week 1 and 2

Preparatory reading and research into tools that can be used for writing the specification and in the extension, the proofs. The tools of choice at the time of proposal are Ott for the specification step and Coq[4] as the proof assistant. Potentially a meeting arranged in the Computer Lab by an expert in using these tools.

Deliverable: Small example specifications to try out the tool chain, including SKI combinator calculus.

C.5.2 Week 3 and 4

Investigating the two current libraries and their design decisions and planning the necessary parts of specification. Identifying the test cases that are thorough and common in concurrent code.

Deliverable: A document describing the major design decisions of the two libraries, the difference in design of the specification and a set of test cases much like the ones used in OCaml Light [20, 5], but with a concurrency focus.

C.5.3 Week 5 and 6

Writing the specification and exporting to automated theorem provers and OCaml.

Deliverable: The specification document in the format of the semantics tool and exported in the formats of the proof assistant and OCaml.

C.5.4 Week 7 and 8

Hand implement a version that adheres to the specification and test it against the runnable semantics.

C.5.5 Week 9 and 10

Evaluating the implementations of the concurrency framework against LWT and Async. Writing up the halfway report.

Deliverable: Evaluation data and charts, the halfway report.

C.5.6 Week 11 and 12

If unexpected complexity occurs these two weeks can be used to compensate, otherwise starting on the verification proof in the proof assistant.

C.5.7 Week 13 and 14

If necessary adding more primitives (I/O, network) to test with, improving performance and finishing the verification proof. If time allows writing guide for future use of the framework.

C.5.8 Week 15 and 16

Combining all previously delivered documents as a starting point for the dissertation and doing any necessary further evaluation and extension. Creating the first, rough draft of the dissertation.

C.5.9 Week 17 and 18

Getting to the final structure but not necessarily final wording of the dissertation, acquiring all necessary graphs and charts, incorporating ongoing feedback from the supervisor.

C.5.10 Week 19 and 20

Finalising the dissertation and incorporating all feedback and polishing.

C.6 Resource Declaration

The project will need the following resources:

- MCS computer access that is provided for all projects
- The OCaml core libraries and compiler

- The LWT and Async libraries
- The Lem tool
- The Ott tool
- The use of my personal laptop, to work more efficiently

As my personal laptop is included a suitable back-up plan is necessary which will consist of the following:

- A backup to my personal Dropbox account
- A Git repository on Github
- Frequent backups (potentially remotely) to the MCS partition

My supervisor and on request my overseers will receive access to both the Dropbox account and Github repository to allow full transparency.