Anik Roy

A probabilistic programming language in Ocaml

Diploma in Computer Science

Christ's College

October 23, 2019

Proforma

Name:	**Anik Roy**
College:	**Christ's College**
Project Title:	**A probabilistic programming language in Ocaml**
Examination:	**Diploma in Computer Science, July 2020**
Word Count:	–[1] **(well less than the 12000 limit)**
Project Originator:	Dr R. Mortier
Supervisor:	Dr R. Mortier

Original Aims of the Project

Work Completed

Special Difficulties

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

Declaration

Contents

List of Figures

Acknowledgements

This document owes much to an earlier version written by Simon Moore [6]. His help, encouragement and advice was greatly appreciated.

Chapter 1

Introduction

1.1 Overview of the files

This document consists of the following files:

- `Makefile` — The Makefile for the dissertation and Project Proposal
- `diss.tex` — The dissertation
- `propbody.tex` — Appendix C – the project proposal
- `proposal.tex` — A LaTeX main file for the proposal
- `figs` – A directory containing diagrams and pictures
- `refs.bib` — The bibliography database

1.2 Building the document

This document was produced using LaTeX 2$_\varepsilon$which is based upon LaTeX[3]. To build the document you first need to generate `diss.aux` which, amongst other things, contains the references used. This if done by executing the command:

```
latex diss
```

Then the bibliography can be generated from `refs.bib` using:

```
bibtex diss
```

Finally, to ensure all the page numbering is correct run `latex` on `diss.tex` until the `.aux` files do not change. This usually takes 2 more runs.

1.2.1 The makefile

To simplify the calls to `latex` and `bibtex`, a makefile has been provided, see Appendix B.1. It provides the following facilities:

- `make`
  Display help information.
- `make prop`
  Run `latex proposal; xdvi proposal.dvi`.
- `make diss.ps`
  Make the file `diss.ps`.
- `make gv`
  View the dissertation using ghostview after performing `make diss.ps`, if necessary.
- `make gs`
  View the dissertation using ghostscript after performing `make diss.ps`, if necessary.
- `make count`
  Display an estimate of the word count.
- `make all`
  Construct `proposal.dvi` and `diss.ps`.
- `make pub`
  Make a `.tar` version of the `demodiss` directory and place it in my `public_html` directory.
- `make clean`
  Delete all files except the source files of the dissertation. All these deleted files can be reconstructed by typing `make all`.
- `make pr`
  Print the dissertation on your default printer.

1.3 Counting words

An approximate word count of the body of the dissertation may be obtained using:

```
wc diss.tex
```

Alternatively, try something like:

```
detex diss.tex | tr -cd '0-9A-Z a-z\n' | wc -w
```

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

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 11 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

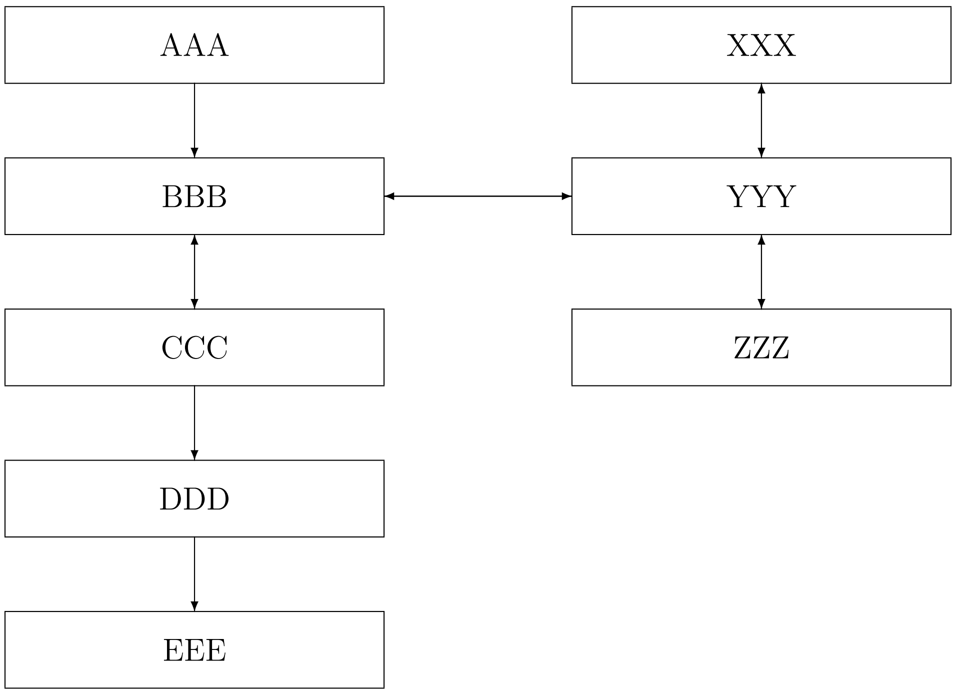

Figure 3.1: A picture composed of boxes and vectors.

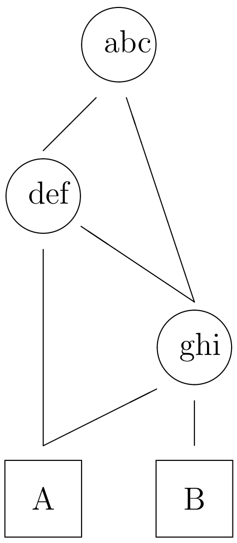

Figure 3.2: A diagram composed of circles, lines and boxes.

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] Noah Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. *arXiv preprint arXiv:1206.3255*, 2012.

[2] Oleg Kiselyov and Chung-Chieh Shan. Embedded probabilistic programming. In *IFIP Working Conference on Domain-Specific Languages*, pages 360–384. Springer, 2009.

[3] L. Lamport. *LaTeX — a document preparation system — user's guide and reference manual.* Addison-Wesley, 1986.

[4] Claus Möbus. Structure and interpretation of webppl. 2018.

[5] Dave Moore and Maria I. Gorinova. Effect handling for composable program transformations in edward2. *CoRR*, abs/1811.06150, 2018.

[6] S.W. Moore. How to prepare a dissertation in latex, 1995.

[7] Adam Ścibior, Zoubin Ghahramani, and Andrew D Gordon. Practical probabilistic programming with monads. In *ACM SIGPLAN Notices*, volume 50, pages 165–176. ACM, 2015.

[8] Shen SJ Wang and Matt P Wand. Using infer. net for statistical analyses. *The American Statistician*, 65(2):115–126, 2011.

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}

\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

\begin{document}

\bibliographystyle{plain}


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Title
```

```
\pagestyle{empty}

\hfill{\LARGE \bf Anik Roy}

\vspace*{60mm}
\begin{center}
\Huge
{\bf A probabilistic programming language in Ocaml} \\
\vspace*{5mm}
Diploma in Computer Science \\
\vspace*{5mm}
Christ's 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 Anik Roy                        \\
College:            & \bf Christ's College                      \\
Project Title:      & \bf A probabilistic programming language in Ocaml \\
Examination:        & \bf Diploma in Computer Science, July 2020        \\
Word Count:         & \bf --\footnotemark[1]
(well less than the 12000 limit) \\
Project Originator: & Dr R.~Mortier                    \\
Supervisor:         & Dr R.~Mortier                    \\
\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}


\section*{Work Completed}


\section*{Special Difficulties}


\newpage
\section*{Declaration}
```

```
I, Anik Roy of Christ's 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.

\bigskip
\leftline{Signed [signature]}

\medskip
\leftline{Date [date]}

\cleardoublepage

\tableofcontents

\listoffigures

\newpage
\section*{Acknowledgements}

This document owes much to an earlier version written by Simon Moore
\cite{Moore95}.  His help, encouragement and advice was greatly
appreciated.

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

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

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

\chapter{Introduction}

\section{Overview of the files}

This document consists of the following files:

\begin{itemize}
\item {\tt Makefile} --- The Makefile for the dissertation and Project Proposal
\item {\tt diss.tex} --- The dissertation
\item {\tt propbody.tex} --- Appendix~C  -- the project proposal
\item {\tt proposal.tex}  --- A \LaTeX\ main file for the proposal
\item{\tt figs} -- A directory containing diagrams and pictures
\item{\tt refs.bib} --- The bibliography database
\end{itemize}

\section{Building the document}

This document was produced using \LaTeXe which is based upon
\LaTeX\cite{Lamport86}.  To build the document you first need to
generate {\tt diss.aux} which, amongst other things, contains the
references used.  This if done by executing the command:

{\tt latex diss}
```

```
\noindent
Then the bibliography can be generated from {\tt refs.bib} using:

{\tt bibtex diss}

\noindent
Finally, to ensure all the page numbering is correct run {\tt latex}
on {\tt diss.tex} until the {\tt .aux} files do not change.  This
usually takes 2 more runs.

\subsection{The makefile}

To simplify the calls to {\tt latex} and {\tt bibtex},
a makefile has been provided, see Appendix~\ref{makefile}.
It provides the following facilities:

\begin{itemize}

\item{\tt make} \\
 Display help information.

\item{\tt make prop} \\
 Run {\tt latex proposal; xdvi proposal.dvi}.

\item{\tt make diss.ps} \\
 Make the file {\tt diss.ps}.

\item{\tt make gv} \\
 View the dissertation using ghostview after performing
{\tt make diss.ps}, if necessary.

\item{\tt make gs} \\
 View the dissertation using ghostscript after performing
{\tt make diss.ps}, if necessary.

\item{\tt make count} \\
Display an estimate of the word count.

\item{\tt make all} \\
Construct {\tt proposal.dvi} and {\tt diss.ps}.

\item{\tt make pub} \\ Make a {\tt .tar} version of the {\tt demodiss}
directory and place it in my {\tt public\_html} directory.

\item{\tt make clean} \\ Delete all files except the source files of
the dissertation. All these deleted files can be reconstructed by
typing {\tt make all}.

\item{\tt make pr} \\
Print the dissertation on your default printer.

\end{itemize}


\section{Counting words}

An approximate word count of the body of the dissertation may be
obtained using:
```

```
{\tt wc diss.tex}

\noindent
Alternatively, try something like:

\verb/detex diss.tex | tr -cd '0-9A-Z a-z\n' | wc -w/




\cleardoublepage



\chapter{Preparation}

This chapter is empty!


\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{propbody}

\end{document}
```

A.2 proposal.tex

```
% This is a LaTeX driving document to produce a standalone copy
% of the project proposal held in propbody.tex.  Notice that
% propbody can be used in this context as well as being incorporated
% in the dissertation (see diss.tex).

\documentclass[12pt,a4paper]{article}
\usepackage[parfill]{parskip}
\usepackage{fullpage}
\usepackage{enumitem}
\usepackage{hyperref}
\bibliographystyle{plain}


\begin{document}

\include{propbody}



\clearpage
```

```
\bibliography{refs}


\end{document}
```

A.3 propbody.tex

```
% !TeX root = ./proposal.tex

\vfil


\begin{center}
      {\Large Computer Science Tripos -- Part II -- Project Proposal} \\
      \vspace{0.4in}
      {\huge \bf A probabilistic programming language in OCaml } \\
      \vspace{0.4in}
      {\large A. Roy, Christ's College} \\
      \vspace{0.1in}
      {\large \today} \\

\end{center}
\vspace{0.4in}

\vfil

\textbf{Project Originator:} Dr. L. Wang
\vspace{0.1in}

\textbf{Project Supervisor:} Dr R. Mortier
\vspace{0.1in}

\textbf{Director of Studies:} Dr R. Mortier
\vspace{0.1in}

\textbf{Project Overseers:} Dr~J.~A.~Crowcroft  \& Dr~T.~Sauerwald

\vfil

% Main document

\section*{Introduction}

A probabilistic programming language (PPL) is a framework in which one can create statistical models and have inference ru

PPLs work well when working with \textit{generative} models, meaning the model describes how some data is generated. This 

The power of a probabilistic programming language comes in being able to describe models using the programming language - 

\section*{Starting Point}

There do exist PPLs for OCaml, such as IBAL \cite{kiselyov2009embedded}, as well as PPLs for other languages, such as WebP

I will be using an existing OCaml numerical computation library (Owl). This library does not contain methods for probabili

I have experience with the core SML language, which will aid in learning basic OCaml due to similarities in the languages,
\section*{Substance and Structure of the Project}

I will be building a PPL in OCaml, essentially writing a domain specific language. There are 2 main components to th

\subsection*{Modelling}
The modelling API is used to represent a statistical model. For example, in mathematical notation, a random variable
\begin{verbatim}
      Variable<double> x = Variable.GaussianFromMeanAndVariance(0, 1)
\end{verbatim}
in the Infer.Net language. In OCaml, there will be many different options for representing distributions, and a choi

I will also need to make sure the design of the modelling language is suitable for the implementation of the inferen

\subsection*{Inference Engine}
There are many different options for a possible inference engine. A decision also need to be made about whether to u

In both these cases, I will need to decide how to convert from a program into a data structure that allows inference

\section*{Evaluation}

The PPL developed here will be compared to existing PPLs for OCaml (in particular, IBAL), comparing performance for p

I will also want to quantify exactly what kind of problems need to be supported by my PPL and make sure these kind o

\subsection*{Success Criteria}

The project will succeed if a usable probabilistic programming language is created. Usable is defined by the followi

\begin{itemize}
      \item \textbf{Language Features}: I will aim to support some subset of language features, such as 'if' stateme
      \item \textbf{Available distributions}: I will aim to make sure my PPL has at minimum the bernoulli and normal
      \item \textbf{Correctness of inference}: I will use the PPL developed on sample problems mentioned before to e
      \item \textbf{Performance}: This is a quantitative measure, comparing programs written in my PPL to equivalent
\end{itemize}

\section*{Extensions}

There are several extensions which could be considered, time permitting:

\begin{enumerate}
      \item There could be more options for the inference engine, i.e. implementing more than one inference algorith
      \item Optimisations could be considered to ensure the performance of inference was better than other comparabl
      \item I could add more distributions, as well as the ability to create custom distributions
      \item Include the ability to visualise results using the plotting module in owl.
      \item Include the ability to visualise the model in which inference is being performed (e.g. the factor graph)
\end{enumerate}


\section*{Schedule}
Planned starting date is {28/10/19}, the Monday after handing in project proposal. Work is broken up into roughly 2 

\subsection*{Michaelmas Term}
\begin{itemize}
      \item {\bf Weeks 3-4 (28/10/19 -- 10/11/19)} \\ Set up IDE and local environment - installing Owl and practici
      \item \textbf{Weeks 5-6 (11/10/19 -- 24/11/19)} \\ Design a basic modelling API and write module/function sign
      \item \textbf{Weeks 7-8 (25/10/19 -- 08/12/19)} \\ Begin to implement the modelling API, and allow running a m
\end{itemize}
\subsection*{Christmas Holidays}
\begin{itemize}
```

```
      \item \textbf{Weeks 1-2 (09/12/19 -- 22/12/19)}\\ Begin to implement a basic inference algorithm (such as MCMC) allo
      \item \textbf{Weeks 3-4 (23/12/19 -- 05/01/20)} [\textit{Christmas Break}]
      \item \textbf{Weeks 5-6 (06/01/20 -- 19/01/19)}\\ Aim to finish the main bulk of implementation by the end of this w
\end{itemize}
\subsection*{Lent Term}
\begin{itemize}
      \item \textbf{Weeks 1-2 (20/01/19 -- 02/02/20)}\\ Finish progress report and implementation as well as any extension
            \\ \textit{Milestone: Progress report deadline (31/01/19)}
      \item \textbf{Weeks 3-4 (03/02/20 -- 16/02/20)}\\ Prepare for the presentation, begin planning the dissertation, par
            \\ \textit{Milestone: Progress report presentations (06/02/20)}
      \item \textbf{Weeks 5-6 (17/02/20 -- 01/02/20)}\\ Finish writing up the bulk of the implementation section.
      \item \textbf{Weeks 7-8 (02/03/20 -- 15/03/20)}\\ Complete first draft of dissertation and complete any unfinished t
\end{itemize}
\subsection*{Easter Holidays}
\begin{itemize}
      \item \textbf{Weeks 1-6 (16/03/20 -- 26/04/20)}\\ Improve dissertation based on supervisor feedback
\end{itemize}
\subsection*{Easter Term}
\begin{itemize}
      \item \textbf{Weeks 1-2 (27/04/20 -- 07/04/20)}\\ Finalise disseration after proof reading and hand in.
            \\ \textit{Milestone: Electronic Submission deadline (08/04/20)}
\end{itemize}


\section*{Resources Required}

\paragraph*{Hardware}
I intend to use my personal laptop for the main development and subsequent write up (HP Pavilion 15, 8GB RAM, i5-8265U CPU

\paragraph*{Software}
The required software includes the ocaml compiler, with a build system (dune) and a package manager (opam). I will also us

\paragraph*{Backups}
For backups, I will use GitHub to host my git repository remotely, pushing frequently. I will also backup weekly to a USB
```

Appendix B

Makefile

B.1 Makefile

B.2 refs.bib

```
@BOOK{Lamport86,
TITLE = "{LaTeX} --- a document preparation system --- user’s guide
and reference manual",
AUTHOR = "Lamport, L.",
PUBLISHER = "Addison-Wesley",
YEAR = "1986"}

@REPORT{Moore95,
TITLE = "How to prepare a dissertation in LaTeX",
AUTHOR = "Moore, S.W.",
YEAR = "1995"}

@inproceedings{kiselyov2009embedded,
  title={Embedded probabilistic programming},
  author={Kiselyov, Oleg and Shan, Chung-Chieh},
  booktitle={IFIP Working Conference on Domain-Specific Languages},
  pages={360--384},
  year={2009},
  organization={Springer}
}

@article{goodman2012church,
  title={Church: a language for generative models},
  author={Goodman, Noah and Mansinghka, Vikash and Roy, Daniel M and Bonawitz, Keith and Tenenbaum, Joshua B},
  journal={arXiv preprint arXiv:1206.3255},
  year={2012}
}

@article{mobus2018structure,
  title={Structure and Interpretation of WebPPL},
  author={M{\"o}bus, Claus},
  year={2018}
}
```

```
@article{wang2011using,
  title={Using infer. net for statistical analyses},
  author={Wang, Shen SJ and Wand, Matt P},
  journal={The American Statistician},
  volume={65},
  number={2},
  pages={115--126},
  year={2011},
  publisher={Taylor \& Francis}
}

@inproceedings{Stucki:2013:OPP:2489837.2489848,
 author = {Stucki, Sandro and Amin, Nada and Jonnalagedda, Manohar and Rompf, Tiark},
 title = {What Are the Odds?: Probabilistic Programming in Scala},
 booktitle = {Proceedings of the 4th Workshop on Scala},
 series = {SCALA '13},
 year = {2013},
 isbn = {978-1-4503-2064-1},
 location = {Montpellier, France},
 pages = {11:1--11:9},
 articleno = {11},
 numpages = {9},
 url = {http://doi.acm.org/10.1145/2489837.2489848},
 doi = {10.1145/2489837.2489848},
 acmid = {2489848},
 publisher = {ACM},
 address = {New York, NY, USA},
 keywords = {EDSL, Scala, probabilistic inference, probabilistic programming, probability monad},
}

@inproceedings{scibior2015practical,
  title={Practical probabilistic programming with monads},
  author={{\'S}cibior, Adam and Ghahramani, Zoubin and Gordon, Andrew D},
  booktitle={ACM SIGPLAN Notices},
  volume={50},
  number={12},
  pages={165--176},
  year={2015},
  organization={ACM}
}

@article{DBLP:journals/corr/abs-1811-06150,
  author    = {Dave Moore and
               Maria I. Gorinova},
  title     = {Effect Handling for Composable Program Transformations in Edward2},
  journal   = {CoRR},
  volume    = {abs/1811.06150},
  year      = {2018},
  url       = {http://arxiv.org/abs/1811.06150},
  archivePrefix = {arXiv},
  eprint    = {1811.06150},
  timestamp = {Sun, 25 Nov 2018 18:57:12 +0100},
  biburl    = {https://dblp.org/rec/bib/journals/corr/abs-1811-06150},
  bibsource = {dblp computer science bibliography, https://dblp.org}
}
```

Appendix C

Project Proposal

Computer Science Tripos – Part II – Project Proposal

A probabilistic programming language in OCaml

A. Roy, Christ's College

October 23, 2019

Project Originator: Dr. L. Wang

Project Supervisor: Dr R. Mortier

Director of Studies: Dr R. Mortier

Project Overseers: Dr J. A. Crowcroft & Dr T. Sauerwald

Introduction

A probabilistic programming language (PPL) is a framework in which one can create statistical models and have inference run on them automatically. A PPL can take the form of its own language (i.e. a DSL), or be embedded within an existing language (such as OCaml). The ability to write probabilistic programs within OCaml would allow us to leverage the benefits of OCaml, such as expressiveness, a strong type system, and memory safety. The use of a numerical computation library, Owl, will allow us to perform inference in a performant way.

PPLs work well when working with *generative* models, meaning the model describes how some data is generated. This means the model can be run 'forward' to generate outputs based on the model. The more interesting application, however, is to run it 'backwards' in order to infer a distribution for the model.

The power of a probabilistic programming language comes in being able to describe models using the programming language - in my case, probabilistic models would be described in OCaml code, with basic distributions being able to be combined using functions and operators as in OCaml code.

Starting Point

There do exist PPLs for OCaml, such as IBAL [2], as well as PPLs for other languages, such as WebPPL - JS[4], Church - LISP[1] or Infer.Net - F#[8] to name a few. My PPL can draw on some of the ideas introduced by these languages, particularly in implementing efficient inference engines.

I will be using an existing OCaml numerical computation library (Owl). This library does not contain methods for probabilistic programming in general, although it does contain modules which will help in the implementation of an inference engine such as efficient random number generation and lazy evaluation.

I have experience with the core SML language, which will aid in learning basic OCaml due to similarities in the languages, however I will still have to learn the modules system. 1B Foundations of Data Science also gives me an understanding of basic statistics and bayesian inference. I do not have experience with domain specfic languages in OCaml, although the 1B compilers course did implement a compiler in OCaml.

Substance and Structure of the Project

I will be building a PPL in OCaml, essentially writing a domain specific language. There are 2 main components to the system, namely the modelling API (language design) and the inference engine.

Modelling

The modelling API is used to represent a statistical model. For example, in mathematical notation, a random variable representing a coin flip may be represented as $X \sim N(0, 1)$, but in a PPL we need to represent this as code. An example would be

```
Variable<double> x = Variable.GaussianFromMeanAndVariance(0, 1)
```

in the Infer.Net language. In OCaml, there will be many different options for representing distributions, and a choice will need to be made about whether to create a separate domain specific language (DSL) or whether to embed the language in OCaml as a library.

I will also need to make sure the design of the modelling language is suitable for the implementation of the inference engine. For example, WebPPL[4] uses a continuation passing style transformation, recording continuations when probabilistic functions are called in order to build an execution trace. This then allows the engine to perform inference. A similar approach could be applied here. There are many alternative approaches to build execution traces, such as algebraic effects[5] or monads[7], and one such method will need to be chosen. However I decide to implement this, I will need to ensure that features of OCaml can be used appropriately.

Inference Engine

There are many different options for a possible inference engine. A decision also need to be made about whether to use a trace-based model (as mentioned) or a graph based model (such as Edward, where a computational graph is generated). This decision will need to be made before implementing the inference engine since it will affect the modelling language.

In both these cases, I will need to decide how to convert from a program into a data structure that allows inference to be performed, and then actually carry it out. Ideally, the inference algorithm used is separated from the definition of the models, so that different algorithms can be chosen, or new algorithms added in the future.

Evaluation

The PPL developed here will be compared to existing PPLs for OCaml (in particular, IBAL), comparing performance for programs describing the same models. I will also use the PPL developed on example problems in isolation to ensure it can be used correctly and delivers correct results. An example would be to use it on an established dataset (e.g. the stop-and-search dataset used in 1B Foundations of Data Science) to attempt to fit a model.

I will also want to quantify exactly what kind of problems need to be supported by my PPL and make sure these kind of programs can be run. I will also support a minimum number of standard distributions, e.g. bernoulli, normal, geometric, etc. or enable users to define custom distributions.

Success Criteria

The project will succeed if a usable probabilistic programming language is created. Usable is defined by the following:

- **Language Features**: I will aim to support some subset of language features, such as 'if' statements (to allow models to be conditional), operators and functions. Some of these features may only be available by special added keywords (e.g. a custom 'if' function).
- **Available distributions**: I will aim to make sure my PPL has at minimum the bernoulli and normal distributions available as basic building blocks to build more complex probabilistic programs.
- **Correctness of inference**: I will use the PPL developed on sample problems mentioned before to ensure correct results are produced. This would be determined by comparing to results produced in other PPLs.
- **Performance**: This is a quantitative measure, comparing programs written in my PPL to equivalent programs in other PPLs. I can use the spacetime program to profile my OCaml code. Performing inference should be possible within a reasonable amount of time, even though the project does not have a significant focus on performance.

Extensions

There are several extensions which could be considered, time permitting:

1. There could be more options for the inference engine, i.e. implementing more than one inference algorithm. Different algorithms are suited to different inference tasks, so this would be a worthwhile extension

2. Optimisations could be considered to ensure the performance of inference was better than other comparable languages.

3. I could add more distributions, as well as the ability to create custom distributions

4. Include the ability to visualise results using the plotting module in owl.

5. Include the ability to visualise the model in which inference is being performed (e.g. the factor graph)

Schedule

Planned starting date is 28/10/19, the Monday after handing in project proposal. Work is broken up into roughly 2 week sections.

Michaelmas Term

- **Weeks 3-4 (28/10/19 – 10/11/19)**
  Set up IDE and local environment - installing Owl and practicing using it. Read papers on past PPLs and implementations, both ocaml or otherwise. Set up project repository and directory structure.

- **Weeks 5-6 (11/10/19 – 24/11/19)**
  Design a basic modelling API and write module/function signatures. Decide how to implement this (e.g. DSL vs library). Research inference algorithms and make sure they will fit into the modelling API designed.

- **Weeks 7-8 (25/10/19 – 08/12/19)**
  Begin to implement the modelling API, and allow running a model 'forward', i.e. generating samples.

Christmas Holidays

- **Weeks 1-2 (09/12/19 – 22/12/19)**
  Begin to implement a basic inference algorithm (such as MCMC) allowing programs to be run 'backwards' to infer parameters.

- **Weeks 3-4 (23/12/19 – 05/01/20)** [*Christmas Break*]
- **Weeks 5-6 (06/01/20 – 19/01/19)**
  Aim to finish the main bulk of implementation by the end of this week and consider extensions if finished early. Begin writing up progress report.

Lent Term

- **Weeks 1-2 (20/01/19 – 02/02/20)**
  Finish progress report and implementation as well as any extensions, time permitting. Use the PPL developed on example problems in order to evaluate it, comparing against problems in other languages.
  Milestone: Progress report deadline (31/01/19)
- **Weeks 3-4 (03/02/20 – 16/02/20)**
  Prepare for the presentation, begin planning the dissertation, particularly the structure and the content I need to write for each section. Begin writing, starting with the first sections (e.g. introduction).
  Milestone: Progress report presentations (06/02/20)
- **Weeks 5-6 (17/02/20 – 01/02/20)**
  Finish writing up the bulk of the implementation section.
- **Weeks 7-8 (02/03/20 – 15/03/20)**
  Complete first draft of dissertation and complete any unfinished tasks.

Easter Holidays

- **Weeks 1-6 (16/03/20 – 26/04/20)**
  Improve dissertation based on supervisor feedback

Easter Term

- **Weeks 1-2 (27/04/20 – 07/04/20)**
  Finalise disseration after proof reading and hand in.
  Milestone: Electronic Submission deadline (08/04/20)

Resources Required

Hardware I intend to use my personal laptop for the main development and subsequent write up (HP Pavilion 15, 8GB RAM, i5-8265U CPU, running Ubuntu and Windows dual booted).

Software The required software includes the ocaml compiler, with a build system (dune) and a package manager (opam). I will also use the IDE VSCode with an OCaml extension, as well as git for version control and latex for the write up.

Backups For backups, I will use GitHub to host my git repository remotely, pushing frequently. I will also backup weekly to a USB stick in case of failures. The software I require is available on MCS machines, so I'll be able to continue work in the event of a hardware failure.