Advances in Programming Languages

Lecture 5: Coursework Assigment Topics

Ian Stark

School of Informatics
The University of Edinburgh

Tuesday 4 October 2016
Semester 1 Week 3

THE UNIVERSITY *of* EDINBURGH

Outline

Course grades !

Final grades are based on an exam (80%) and a written coursework assignment (20%).

The exam is in April/May, and has a standard "choose two questions out of three" format. Past papers are available through the course web page.

The examinable material for this course is the content of the lectures and their accompanying homework exercises. The following will not be assessed in the examination:

- Any guest lectures;
- The written coursework;
- Further references in the course blog.

The written coursework requires investigation of a topic in programming languages and writing a 10-page report with example code.

Aims of homework exercises !

The aims of the homework exercises set in lectures include:

- To prepare for forthcoming lectures
- To review lecture material
- To give some context for understanding the lectures
- To provide other sources and views on the lecture topic
- To help you learn by exploring the subject

Crucially, these effects arise from doing the exercises. They are not assessed, nor would this necessarily be useful: they are intended to be largely self-validating (i.e. you can tell when you have succeeded).

Although your coursework reports will be assessed, most of this also applies there: the purpose is for you to find out and learn new things.

Homework

1. Read This

Information about the APL written coursework appears today on the course website. The assignment is to investigate one of five programming-language topics and write a 10-page report on it.

During next Tuesday's lecture I shall give a brief overview of each topic, and explain more about what's involved in the assignment. Before then, download and read the assignment handout on the course website.

2. Do This

Work through those "Exercises for the Reader" on encoding products and sums in System F.

Extension

Find out how { Java, Scala, C#, Haskell, ... } handles type { variance, bounds, quantification }.

Encoding Products in System F

Product Type and Pairing

$$\mathsf{Prod}\ X\ Y = \forall Z.((X \to Y \to Z) \to Z)$$

$$\mathsf{pair} = \Lambda X.\Lambda Y.(\lambda x{:}X.\lambda y{:}Y.(\Lambda Z.\lambda f{:}(X \to Y \to Z).(f\ x\ y)))$$

$$\mathsf{pair} : \forall X.\forall Y.(X \to Y \to \mathsf{Prod}\ X\ Y)$$

$$\mathsf{pair}\ A\ B\ M\ N \stackrel{\beta}{\Longrightarrow} \Lambda Z.\lambda f{:}(A \to B \to Z).f\ M\ N$$

First Projection

$$\mathsf{fst} = \Lambda X.\Lambda Y.\lambda p{:}(\mathsf{Prod}\ X\ Y).p\ X\ (\lambda x{:}X.\lambda y{:}Y.x)$$

$$\mathsf{fst} : \forall X.\forall Y.\mathsf{Prod}\ X\ Y \to X$$

$$\mathsf{fst}\ A\ B\ (\mathsf{pair}\ A\ B\ M\ N) \stackrel{\beta}{\Longrightarrow} M$$

Syntactic Sugar

$$A \times B = \mathsf{Prod}\ A\ B$$

$$(M, N)_{A,B} = \mathsf{pair}\ A\ B\ M\ N : A \times B$$

$$\mathsf{fst}_{A,B} = \mathsf{fst}\ A\ B : A \times B \to A$$

(Earlier versions of this slide gave fst incorrectly)

Exercises for the Reader

Based on the preceding encoding for products in System F:

- Write out a definition for second projection "snd";
- Show that it has the right type, and reduces with

$$\mathsf{snd}\ A\ B\ (\mathsf{pair}\ A\ B\ M\ N) \stackrel{\beta}{\Longrightarrow} N$$

- Define terms inl and inr and case for the following definition of sum types:

$$\mathsf{Sum}\ X\ Y = \forall Z.((X \to Z) \to (Y \to Z) \to Z)$$

- What is the type corresponding to $(\forall X.X \to X)$?
  What about $(\forall X.X)$?

Outline

Functional Reactive Programming in elm

Functional Reactive Programming (FRP) is a technique for writing programs that interact with their environment over time. FRP is declarative rather than imperative: code describes *what* is desired of a system's behaviour in response to events, rather than *how* it is computed. This makes for a high-level compositional programming style, which a compiler then transforms into efficient interactive code appropriate to the underlying platform.

elm is a declarative and purely-functional language for web applications, compiling succinct descriptions of graphic display and user interaction into low-level HTML, CSS and Javascript.

Original elm, up to v0.16, directly used FRP *Signals* to describe interaction over time. In May 2016 version 0.17 simplified this to the more static *elm architecture*. You can investigate either the old signals or the new architecture, in relation to more general uses of FRP.

Information flow policies in Jeeves

The *Jeeves* language provides a way for programmers to describe and enforce security policies for code. These policies manage *information flow* in a program: identifying what values depend on what others, and in particular where confidential information may travel during execution. In Jeeves these policies are written separately from program code, and the language implementation ensures that policies are enforced when the program is executed.

Jeeves uses *faceted* expressions to track information flow at runtime, which means it can efficiently identify *implicit flows*. It is implemented as an *embedded domain-specific language* for Python and Scala.

Quipper

Programming quantum computation with Quipper

The field of *quantum computation* considers devices that exploit differences between classical and quantum physics. There are several cases where *quantum algorithms* for particular problems are known to give complexity improvements over any classical algorithm. These are often described in terms of *quantum gates* and *quantum circuits*: quantum analogues of the same low-level components in classical computers.

Quipper is a high-level language for describing quantum circuits. It is implemented as an embedded domain-specific language in Haskell, and provides support for generating circuits and simulating their behaviour on a classical computer.

F# query expressions for language-integrated database access

Microsoft's F# language provides several facilities for building and high-level manipulation of computations and metacomputations. In particular *computation expressions* allow libraries to define domain-specific sublanguages for particular kinds of computation.

One example of this is *query expressions*, a way to write code describing access to the tables and records of classic relational databases. These query expressions can combine F# code with familiar operators from database query languages like SQL, which can then be compiled to run efficiently on different kinds of data source.

Dafny: Statically verifying functional correctness

Dafny is an imperative object-oriented programming language that enhances standard compilation and type-checking with static verification of functional correctness. Programmers annotate their Dafny code with *preconditions*, *postconditions*, *loop invariants* and *frame specifications*, and the compiler automatically checks whether the code satisfies these specifications.

Successful compilation of Dafny program ensures that the every run of the resulting executable satisfies the assertions given in its source code.

Outline

Haskell

The GHC compiler for Haskell does not support dynamically linked libraries on Windows [1].

1. GHC Users Guide. `http://www.haskell.org/ghc/docs/latest/html/users_guide/index.html`

Haskell

The GHC compiler for Haskell does not support dynamically linked libraries on Windows [1].

1. GHC Users Guide. `http://www.haskell.org/ghc/docs/latest/html/users_guide/index.html`

(Not true since GHC 7.0)

Haskell

The GHC compiler for Haskell does not support dynamically linked libraries on Windows [1].

1. GHC Users Guide. `http://www.haskell.org/ghc/docs/latest/html/users_guide/index.html`

(Not true since GHC 7.0) [Citation Needed]

Facebook

Facebook signed up 200 million users in its first five years [NYT].

[NYT] “Is Facebook Growing Up Too Fast?” New York Times

Links

The *Links* programming language compiles client code into Javascript (Cooper et al. 2007).

Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop.
Links: Web Programming Without Tiers.
In *Formal Methods for Components and Objects: Proceedings of the 5th International Symposium FMCO 2006.* Lecture Notes in Computer Science 4709, pages 266–296. Springer-Verlag, 2007.
DOI: 10.1007/978-3-540-74792-5_12

Moore

Moore's law says that the number of components in an integrated circuit doubles every year[1].

1. Wikipedia

Events

Using *events* can be an effective technique for concurrent programming†.

† John Ousterhout. Why Threads are a Bad Idea (for most purposes). In *Proceedings of the USENIX 1996 Technical Conference.* January 1996.

Outline

Dates and submission

Week 2 Friday 30 September	Topics announced
Week 4 Friday 14 October	Preliminary report due **by 3pm**
Week 5 Tuesday 18 October	Feedback on preliminary report
Week 8 Friday 11 November	Final report due **by 3pm**
Week 10 Friday 25 November	Feedback on final report

Each report must be submitted electronically as a PDF document. The recommended method for creating these is pdflatex with the article document class.

In addition, LibreOffice is freely available for Windows, Linux and Mac, installed on Informatics machines, and can write PDF. Mac OS X natively creates PDF. Microsoft Office 2007 and later can save as PDF.

Declaring topic choice

Preliminary report

This document should contain:

- Your student matriculation number;
- The topic you have chosen;
- Three suitable references;
- For each reference, a paragraph summarizing what it says.
- A screenshot by you of the selected system in action.

One reference must be to a published paper; the other two may be also, but could also be white papers, web tutorials, manuals, or similar. In all cases provide enough information for someone else to obtain the document.

To create the screenshot, you will need to have your chosen system downloaded, installed, and running on a suitable machine.

Suggested outline for final report

Heading	Title, date, student number
Abstract	This report describes ...
Introduction	Content summary, overview of report structure
Context	The problem domain
⟨Main topic⟩	What it is, how it works, advantages and limitations
Example	Annotated code, explanation, screenshot
	Salt: The example must in some way concern the election of a head of state (e.g. primaries, candidates, ...)
Resources	For each notable resources used (article, tutorial, manual), give a summary in your own words of what it contains
Related work	Other approaches to the problem
Conclusion	What ⟨topic⟩ does, good and bad points
Bibliography	Full references for all resources used

Total around 10 A4 pages. See the course web pages for further details.

Outline

Academic Integrity !

University of Edinburgh Undergraduate Assessment Regulations

Regulation 29

All work submitted for assessment by students is accepted on the understanding that it is the student's own effort without falsification of any kind.

Academic Integrity !

University of Edinburgh Undergraduate Assessment Regulations

Regulation 30

30.1 Marks or grades can only be given for original work by students at the University. Plagiarism is the act of copying or including in one's own work, without adequate acknowledgement, intentionally or unintentionally, the work of another or one's own previously assessed original work. It is academically fraudulent and an offence against University discipline. ...

Academic Integrity !

University of Edinburgh Undergraduate Assessment Regulations

Regulation 30

30.2 It is academically fraudulent and an offence against University discipline for a student to invent or falsify data, evidence, references, experimental results or other material contributing to any student's assessed work or for a student knowingly to make use of such material. It is also an offence against University discipline for students to collude in the submission of work that is intended for the assessment of individual academic performance or for a student to allow their work to be used by another student for fraudulent purposes.

Academic Integrity !

University of Edinburgh Undergraduate Assessment Regulations

Regulation 29

All work submitted for assessment by students is accepted on the understanding that it is the student's own effort without falsification of any kind.

http://www.inf.ed.ac.uk/teaching/tar

See also:

- University guidance
  http://www.ed.ac.uk/academic-services/students/conduct/academic-misconduct
- Informatics statement
  http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Suitable working practices !

Working practices

- Start with a blank document; all the words must be yours.
- Do not cut and paste from other documents.
  - Except for direct quotations, which must have source declared.
- Do not let others read your text; nor read theirs (except as directed).

Aims of this assignment

- To learn about the chosen topic
- To improve researching and learning skills
- To demonstrate said knowledge and skills

The tangible outcome is a document, composed and written by you, demonstrating what you have learnt.

Homework

Before the next lecture find an online tutorial for each of the assignment topics:

- Functional reactive programming in elm
- Information flow policies in Jeeves
- Programming quantum computation with Quipper
- F# query expressions for language-integrated database access
- Dafny: Verifying functional correctness

Send me your list of five links and I will summarize them for the class.

Use them to help choose your topic.