Introduction & overview

Jeremy Yallop
jeremy.yallop@cl.cam.ac.uk
About the class
Intraseminar structure

**Structure**

- **High marks**
- **Low marks**
- **PL**
- **Running**
- **Designing**
- **Analysing**

**3:05pm**
- Student presentation
- Discussion

**4:15pm**
- Student presentation
- Discussion

**4:30pm**
- Student presentation
- Discussion

**4:55pm**
- Mini-lecture: next week’s topic
Interseminar structure

Structure

High marks

Low marks

PL

Running

Designing

Analysing

Seminar 1 (9 Oct)

garbage collection
essay

Seminar 2 (16 Oct)

delimited continuations

Seminar 3 (23 Oct)

depen typ

Released

Due

Released

Due

Released
What you’ll do each week

- Introductory mini-lecture
- Background reading
- Read papers
- Optional: wider reading
- Write & submit essay
- Take part in discussion
## Presentation slot assignments

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Speaker 1</th>
<th>Speaker 2</th>
<th>Speaker 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 Oct</td>
<td>Garbage collection</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>23 Oct</td>
<td>Delimited continuations</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>30 Oct</td>
<td>Dependent types</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6 Nov</td>
<td>Module systems</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>13 Nov</td>
<td>Abstract interpretation</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>20 Nov</td>
<td>Partial evaluation</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>27 Nov</td>
<td>Program synthesis</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

How well did allocation work? Everyone received his/her 1st or 2nd choice.

Opportunity: One remaining vacancy for module systems. Volunteers welcome!
How to get high marks in this class
Essay marks are awarded for **understanding**, for **insight and analysis**, and for **writing quality**.

Essays should be around 1500 words.

1. Contextualise *widely*
2. Analyse *deeply*
3. Appraise *thoughtfully*
4. Elucidate *carefully*
5. Describe *originally*
6. Synthesise *insightfully*
7. Expound *illustratively*
8. Write *stylishly*
1. Introduction

...
Presentation marks are awarded for *clarity*,
for *effective communication*,
and for *selection and organisation of topics*

1. *engage* with the audience
2. *empathize* with the audience
3. *bring people along*
4. explain the *problem*
5. bring out the *key idea*
6. have one *key example*
Read *a book*

Look at *some slides*

Watch *a presentation*
How to get low marks in this class
How to get a low mark in an essay

1. be exclusively *critical*
2. *quote* extensively
3. assert *without evidence*
4. stay *vague and noncommittal*
1. read your slides

- my first point
- my second point

2. overrun

3. stuff your slides

4. disregard structure
Programming languages: themes
Q: what is a program?
Q: what *undecidable question* are we approximating?
Overview

Structure

High marks

Low marks

PL

Running

Designing

Analysing

Oct 9

Oct 16

Oct 23

Oct 30

Nov 6

Nov 13

Nov 20

Nov 27

running programs

Garbage collection

Delimited continuations

Dependent types

Module systems

Abstract interpretation

Partial evaluation

Program synthesis

designing languages
Running programs
**Garbage collection**

**Question:** How can we efficiently automatically reclaim storage that is no longer needed by a program?

**A program is** a process that mutates memory by allocating, freeing, reading and writing blocks of memory.

**What’s undecidable?** **Liveness:** it is not possible to determine whether each value can be used by the program in future.
**Delimited continuations**

**Question:** How can we extend programming languages with operators that allow powerful manipulation of control flow?

*A program is* a calculation that may interact with its context

**More questions:**
- How can we give types to delimited control operators?
- How can we elaborate programs with delimited control?
- What is the connection with algebraic effects?
Designing programming languages
Question: How can we build a powerful, usable, and efficient programming language out of type theory?

A program is a blend of logic and computation.

What's undecidable? Type equivalence is undecidable in general.

More questions:
How should we handle equality?
How might we write programs in a dependently-typed language?
How might we compile programs effectively?
Question: How can we construct a language that allows us to assemble large systems from well-specified components?

A program is a large modular system assembled from separately-defined components.

More questions:
How can we support abstraction and flexible composition?
What might a core language of modules look like?
How might we add support for recursion, higher-order modules, and first-class modules?
What problems might arise in sophisticated module systems?

```plaintext
module type SET =
sig
  type t
  type elem
  val empty : t
  val add : elem → t → t
  val mem : elem → t → bool
end

module MakeSet (Elem: ORDERED) :
  SET with type elem = Elem.t
```
Analysing programs
Abstract interpretation

**Question:** How can we analyse a program to obtain information about it?

**A program is** an object that can be given a variety of semantics of varying levels of precision

**What’s undecidable?** Most questions. Instead, deal with sound overapproximations.
**Question:** How might we perform as much computation as possible in advance?

**A program is** an open term that can be simplified using reductions.

**What's undecidable?** Whether a program is optimally partially evaluated is undecidable.

More questions:
- How can we transform a program to improve its partial evaluation?
- Is partial evaluation useful in practice?
- How can we incorporate equations other than $\beta$?
**Question:** How can we generate programs from specifications?

**A program is** an object in a very large search space.

**What’s undecidable?** Whether a program meets a specification is undecidable in general.