

COMP3610  
Principles of Programming Languages  
Lecture 26: Summary and Exam Revision

---

Ben Lippmeier  
Australian National University  
Semester 2  
2009

Part 1: Lambda calculus and transformation.

---

- **Lambda calculus**  
beta-reduction, Church encoding, variable capture, confluence, Church-Rosser theorem, evaluation order, divergence.
- **Reduction strategies**  
specification of, call-by-value, call-by-name, normal forms.
- **Typed lambda calculus**  
typing rules, proofs of well-typedness
- **Programs are proofs**  
Curry-Howard isomorphism, proof witnesses.

Drop In / Exam Preparation Tute

---

- When?
- Monday before the exam?
- Come with questions.
- **You should already be studying by then!!!**

Part 1: Lambda calculus and transformation.

---

- **Compilation by transformation**  
derived forms, desugaring, optimisation.
- **Type inference**  
Type constraints, unification.
- **System-F and the lambda cube**  
Type abstractions and applications, dependency.
- **Syntactic approach to type soundness.**  
Progress and Preservation.

## What should you be able to do?

---

- Reduce lambda expressions to normal form, using a specified evaluation strategy.
- Identify the normal forms produced by a given evaluation strategy.
- Explain the difference between call-by-value and call-by-name evaluation, and give their evaluation rules.
- Draw type derivations for statements in a given (polymorphic) language, and use them to identify type errors.
- Give a Haskell proof witness for a given statement in propositional logic.

## Part 2: Lexing and Parsing.

---

- **Lexing**  
languages and automata, regular expressions, NFA vs DFA.
- **Top-down parsing.**  
grammars and automata, ambiguity, starters and followers, director symbols, eliminating left recursion, left factoring.
- **Bison and Flex**
- **Bottom up parsing**  
push down automata, shift-reduce conflicts, reduce-reduce conflicts, push down automata.

## What should you be able to do?

---

- Desugar a Haskell program to a System-F style language.
- Write down type constraints and perform type inference for a simple, monomorphic functional language.
- Know what rule induction is, and how a proof of soundness for a typing system fits together.

## What should you be able to do?

---

- Identify the ambiguity in a given grammar and give an example that demonstrates it.
- Eliminate left recursion from a grammar, and left factor a grammar.
- Determine the starter and follower symbols in a grammar.
- **Build an SLR parse table for a grammar**  
... identify shift-reduce and reduce-reduce conflicts  
... and know how they arise.

## Part 3: Semantics.

---

- **Fixpoint theory**  
complete partial orders, monotonic functions, Kleene's first recursion theorem
- **Operational semantics**  
Specification of languages, proofs of semantic equivalence closures and environments, blocks and procedures
- **Denotational semantics**  
....., use of fixpoints to represent looping, continuations and control.

## Part 4: Tying it together.

---

- **Strictness Analysis**
- **Effect Typing**
- These topics were intended to draw together most of the material presented in the course.
- They won't be examined directly, but if you can follow what is going on then you've absorbed the most of the material.

## What should you be able to do?

---

- Determine the least fixpoint of a recursive function using Kleene's first recursion theorem. (and know what that is)
- Prove the semantic equivalence of two expressions (or commands) with both operational and denotational semantics.
- Be able to describe in words why a given operational equivalence is valid or invalid.
- ... or how we could restrict the terms involved to make it valid.

## Exam Outline

---

- Q1 [15 marks] Lambda Calculus
- Q2 [25 marks] Types
- Q3 [15 marks] Top down parsing
- Q4 [15 marks] Bottom-up parsing
- Q5 [10 marks] Fixpoint theory
- Q6 [20 marks] Semantics
- Total 100 marks. 3 hours, any materials (except electronics)

## Exam Revision Advice

---

### Just reading lecture notes is a terrible way to study!!!

*(It's like trying to learn karate by watching it on TV)*

- All material presented in the course is fair game.  
... but I'm not expecting you to have absorbed everything...
- Not all questions in the exam are based on tutorial questions.
- Imagine yourself setting an exam for this course.  
Make up some questions. Then try to solve them!
- Imagine trying to explain the concepts to someone else...

It is easier to write an incorrect program  
than to understand a correct one.

-- Alan Perlis

## Course Feedback Forms (esp Open Ended)

---

- Constructive Criticism!
- I don't expect to be able to do everything well all the time,  
so if you think something could be improved please say so.
- I use this feedback to try and improve my teaching style, and  
to improve the course for subsequent years.
- What parts of the course did you enjoy / dislike / really get  
into / not really understand / wanted more of?
- Does it the material seem relevant to what you want to do in  
the future? Do you see a point to it all?