About the exam

- The lecture after this one will be revision.
- I will present some hints, exercises, and model solutions.
- Also relevant: the exercises on my slides.

Past exams

- It is worth looking at the exams of the last two years (Prof. Pym): enter

  `http://www.bath.ac.uk/library/exampapers/search.html`

  and search for comp0071.
- Relevant questions are Q1 and Q2 in the 2002 exam, and all questions in the 2003 exam.
- I did not do Bunched Logic, but Hoare logic instead.
- There will be some exercises where you are asked to find natural deduction proofs and sequent proofs for given formulæ or sequents.

The λ-calculus & the Propositions-as-Types paradigm

Overview

- We shall introduce the famous λ**-calculus**, which was originally invented as a notation for computable functions and later became the basis of functional programming.
- We shall study a surprising connection between λ-calculus and logic.
- This connection is called the **Propositions-as-types** paradigm.

The λ-operator

We write

$$\lambda x.M$$

for the function that takes an x and returns the value described by the term M.

Examples:

- $\lambda x.x * 2.$ is the function that doubles its argument.
- $F = \lambda x.\lambda y.x * y$ is the function that, given x, returns a function that, given y, returns $x * y$. E.g., $F(2)$ is the doubling function above.

λ-terms

The syntax of the **untyped λ-calculus** is given as follows:

$$\begin{array}{rl} \text{Variables:} & x, y, \ldots \\ \text{Constants:} & c, d ::= + \mid * \mid 0 \mid 1 \mid 2 \ldots \\ \lambda\text{-terms:} & M, N ::= \lambda x.M \mid M\,N \mid x \mid c \end{array}$$

- $M\,N$ stands for the application of the "function" M to the argument N (i.e. for what is often written as $M(N)$).
- The infix notation $x * y$ we used is just a nice-looking notation for the λ-term $((*\,x)\,y)$.

History of the λ-calculus

- Introduced by A. Church in the 1930's as a notation for computable functions, in the context of studying the foundations of mathematics.
- In the 1950's, became the basis of real-life programming languages based on it, e.g. **LISP**. (LISP actually has the keyword "lambda".)
- Since the 1960's, key tool in programming-language theory.

Pairing

The λ-calculus is often extended with

- **pairing** terms of the form (M, N), and
- **projection** terms of the forms $\pi_1(M)$, $\pi_2(M)$.

E.g., the term

$$\lambda x.\lambda y.(\pi_1(x), y)$$

takes an argument x (which is a pair), and then an argument y, and returns the pair whose second component is y, and whose first component is the first component of x.

Types

λ-terms are often given **types**. Types are given by the following grammar:

$A, B ::=$	$A \to B$	**(function types)**
	$A \times B$	**(product types**, i.e., types of pairs)
	$int, nat, bool, \ldots$	**(atomic types)**.

E.g., a possible type of $\lambda f.\lambda g.\lambda x.(fx) - (gx)$ is

$$(nat \to int) \to ((nat \to int) \to (nat \to int)).$$

Contexts

To give types to variables, we introduce **contexts**.

Definition. A **context** Γ is a finite sequence

$$x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$$

where x_i is a variable and A_i is a type for every i, and the x_i are mutually different.

Typing judgments

- We shall introduce judgments of the form

$$\Gamma \vdash M : A,$$

where Γ is a context, M is a λ-term, and A is a type.

- The intended meaning is

"In context Γ, M has type A."

- In the special case where Γ is empty, we say that M **inhabits** the type A.

$$\vdash M : A$$

Typing rule for variables

The typing rule for variables is

$$\frac{}{\Gamma \vdash x : A} \text{ if } x : A \text{ is in } \Gamma \qquad Ax$$

Note the similarity with the Ax rule in natural deduction.

Typing rules involving $\rightarrow$

$$\frac{\Gamma \vdash M : A \rightarrow B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \rightarrow e$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \rightarrow B} \rightarrow i$$

(The "$: A$" after "λx" is only needed to preserve the knowledge that x has type A.)

Note the similarity with the ND rules for $\rightarrow$.

Example

- Q: Is there a λ-term M that inhabits the type

$$A \to ((A \to B) \to B)?$$

- A: Yes: $M = \lambda x : A.\lambda f : A \to B.fx$. To see this, consider the typing derivation below.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x :A, f :A \to B \vdash f :A \to B}\,Ax \qquad \dfrac{}{x :A, f :A \to B \vdash x :A}\,Ax}{x :A, f :A \to B \vdash fx :B}\to e}{x :A \vdash \lambda f : A \to B.fx :(A \to B) \to B}\to i}{\vdash \lambda x : A.\lambda f : A \to B.fx :A \to ((A \to B) \to B)}\to i.$$

Example

- Q: Is there a λ-term N that inhabits the type

$$(((A \to B) \to B) \to B) \to (A \to B)?$$

- A: Yes:

$$N \quad = \quad \lambda h : ((A \to B) \to B) \to B.\lambda x : A.h(Mx),$$

where M is the term from the previous slide.

Typing rules involving $\times$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B} \times i$$

$$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1(M) : A} \times e \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2(M) : B} \times e$$

Note the similarity with the ND rules for $\wedge$.

The simply-typed λ-calculus

Definition. The formal system of judgments

$$\Gamma \vdash M : A$$

that are derivable using the rules Ax, $\rightarrow i$, $\rightarrow e$, $\times i$, and $\times e$ is called the **simply-typed λ-calculus** (with atomic types, function types, and product types).

Typing rules of the simply-typed λ-calculus

$$\frac{}{\Gamma \vdash x : A} \text{ if } x : A \text{ is in } \Gamma \qquad Ax$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B} \times i$$

$$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1(M) : A} \times e \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2(M) : B} \times e$$

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N\ A}{\Gamma \vdash MN : B} \to e \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \to B} \to i$$

Propositions as types

- Erasing the λ-terms and reading $\times$ as $\wedge$ yields intuitionistic ND for $\wedge$ and $\rightarrow$. (As we shall see, this can be extended to $\vee$.)
- So, terms of the simply-typed λ-calculus are in perfect correspondence with ND-proofs in intuitionistic propositional logic!
- This is called the **Propositions-as-Types** paradigm (a.k.a. “Curry-Howard isomorphism”), because it shows that propositions (= formulæ) and types are essentially the same.

Significance

- Because λ-terms can be seen as functional programs, we have correspondence between **programs** and **proofs**.
- The scope of the Propositions-as-Types paradigm goes beyond the logics and λ-calculus considered in this lecture.
- It has had a great impact on the design of programming languages, causing a transfer of design principles between logics and programming languages (see e.g. ML).

Inhabited types and validity

Owing to the Propositions-as-Types paradigm, we know:

Proposition. For every type A, the following are equivalent:

1. A, viewed as a formula, is provable in intuitionistic ND (and therefore valid in intuitionistic logic).
2. A is inhabited by a term of the simply-typed λ-calculus.

Example

We know e.g. that the formula

$$((A \to B) \to B) \to A$$

is not generally inhabited by a λ-term. For in the case $B = \bot$, we get

$$((A \to \bot) \to \bot) \to A,$$

which is essentially RAA and not valid for all A in intuitionistic logic.

Towards conversion

Consider how we evaluate λ-terms, e.g.

$$((\lambda x : nat.\lambda y : nat.\pi_1(x, y))\, 2)\, 3$$

(see lecture). The sequence of evaluation steps can be seen as the execution of a program.

Conversion

Definition. A term M **converts** to a term M' if one of the following three cases holds:

$$M = \pi_1(M_1, M_2) \quad M = \pi_2(M_1, M_2) \quad M = (\lambda x : A.N)L$$
$$M' = M_1 \qquad\qquad M' = M_2 \qquad\qquad M' = N[L/x]$$

where $N[L/x]$ is the term that results from N by replacing every free occurrence of x by the term L. The term M is called the **redex**, and M' is called the **contractum**.

Reduction

Definition. A term M **reduces** to a term N if there is a sequence of conversions from M to N, i.e., a sequence

$$M = M_0, M_1, \ldots, M_n = N$$

such that for $i = 0, 1, \ldots, n-1$, M_{i+1} is obtained by replacing a redex by its contractum. We write $M \rightsquigarrow N$.

Reduction and proofs

Conversions of the form

$$\pi_i(M_1, M_2) \leadsto M_i$$

correspond to removing a detour that consists of $(\wedge i)$ followed by $(\wedge e)$:

$$\dfrac{\dfrac{\begin{matrix}\vdots\ \Phi_1 \\ A\end{matrix} \qquad \begin{matrix}\vdots\ \Phi_2 \\ B\end{matrix}}{A_1 \wedge A_2}\ \wedge i}{A_i}\ \wedge e \qquad \leadsto \qquad \begin{matrix}\vdots\ \Phi_i. \\ A_i\end{matrix}$$

Reduction and proofs

Conversions of the form

$$(\lambda x : A.N)L \leadsto N[L/x]$$

correspond to removing a detour that consists of $(\rightarrow i)$ followed by $(\rightarrow e)$:

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \vdots\ \Phi \\ B\end{array}}{A \rightarrow B}\rightarrow i \qquad \begin{array}{c}\vdots\ \Psi \\ A\end{array}}{B}\rightarrow e \qquad \leadsto \qquad \begin{array}{c}\vdots\ \Psi \\ A \\ \vdots\ \Phi. \\ B\end{array}$$

Adding disjunction

The type corresponding to $A \vee B$ is commonly written as $A + B$. The rules for $+$ are

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash in_1(x) : A + B} +i \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash in_2(x) : A + B} +i$$

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, x : A \vdash N_1 : C \quad \Gamma, y : B \vdash N_2 : C}{\Gamma \vdash case\ M\ of\ in_1(x : A) \Rightarrow N_1 \mid in_2(y : B) \Rightarrow N_2 : C} +e.$$

Note the similarity with the ND rules for $\vee$.