My exchange program in UTokyo

Aliénor Goubault--Larrecq

Curriculum

- Student in the **Computer Science** in ENS de Lyon
- In 4th year of the ENS de Lyon's program : already have a M2

Curriculum

- Student in the **Computer Science** in ENS de Lyon
- In 4th year of the ENS de Lyon's program : already have a M2
- In exchange at the Graduate School of Mathematical Sciences at UTokyo as a Special Auditor
- Interested in Graph, Automata, Proof theories and Logic
- Hence here in **Applied Mathematics**

What I do as an Auditing Student

- Free to attend any courses if the ENS de Lyon agrees
- Attend courses from the Graduate School of **Mathematical Sciences**
- Attend courses from the Graduate School of **Information Science and Technology**
- Attend seminars with students of Professor Ryu Hasegawa
  → Mainly about type theory, but a lot of use of categories

Type theory

→ Used in Logic and Computer Science

→ Several type theories, in particular

Alonzo Church's **typed λ-calculus** cf. later

Per Martin-Löf's **intuitionistic type theory** (constructive mathematics)

Necessity of type theory

Russel's paradox : $R = \{w \mid w \notin w\}$

$$R \in R \text{ iff } R \notin R$$

The class of all sets cannot be a set itself.

Necessity of type theory

Russel's paradox : $R = \{w \mid w \notin w\}$

$$R \in R \text{ iff } R \notin R$$

The class of all sets cannot be a set itself.

One solution : hierarchy of types and assigning each concrete mathematical entity to a type

Necessity of type theory

Russel's paradox : $R = \{w \mid w \notin w\}$

$R \in R$ iff $R \notin R$

The class of all sets cannot be a set itself.

One solution : hierarchy of types and assigning each concrete mathematical entity to a type

Entities of a given type are built exclusively of subtypes of that type :
→ preventing an entity from being defined using itself.

Simply Typed Lambda Calculus

Types:

- Atomic types $T_1, T_2, \dots, T_n$;
- U and V are types then $U \to V$ is a type (**function** type).

Terms:

- Variables $x_1^T, x_2^T, \dots$ for each type T;
- If v is a term of type V and x_n^U is a variable of type U then $\lambda x_n^U.\ v$ is a term of type $U \to V$;
- If t and u are terms of types respectively $U \to V$ and U, then tu is a term of type V.

Application of Lambda-calculus

Booleans

$$\mathrm{Bool}_{\mathrm{U}} = \mathrm{U} \to \mathrm{U} \to \mathrm{U}$$

Application of Lambda-calculus

Booleans

$$\mathrm{Bool}_U = U \to U \to U$$

Church's booleans

$$\mathrm{true} = \lambda x\,.\,\lambda y\,.\,x \quad \text{of type } \mathrm{Bool}_U$$
$$\mathrm{false} = \lambda x\,.\,\lambda y\,.\,y \quad \text{of type } \mathrm{Bool}_U$$

Application of Lambda-calculus

Booleans

$$\mathrm{Bool}_U = U \to U \to U$$

Church's booleans

$\mathrm{true} = \lambda x . \lambda y . x$ of type Bool_U

$\mathrm{false} = \lambda x . \lambda y . y$ of type Bool_U

$\mathrm{ifelse} = \lambda p . \lambda a . \lambda b . p\ a\ b$

of type $\mathrm{Bool}_U \to U \to U \to U$

Application of Lambda-calculus

Booleans

$$\text{Bool}_U = U \to U \to U$$

Church's booleans

$\text{true} = \lambda x \,.\, \lambda y \,.\, x$ of type Bool_U

$\text{false} = \lambda x \,.\, \lambda y \,.\, y$ of type Bool_U

$\text{ifelse} = \lambda p \,.\, \lambda a \,.\, \lambda b \,.\, p\ a\ b$

of type $\text{Bool}_U \to U \to U \to U$

$$\begin{aligned}\text{ifelse true } 39\ 58 &= \text{true } 39\ 58 \\ &\to (\lambda x \,.\, \lambda y \,.\, x)\ 39\ 58 \\ &\to (\lambda y \,.\, 39)\ 58 \\ &\to 39\end{aligned}$$

Curry Howard Principle

Analogy between formal logic and computational calculi

Logic side	Programming side
Formula	Type
Proof	Term

Curry Howard Principle

Analogy between formal logic and computational calculi

Logic side	Programming side
Formula	Type
Proof	Term
$\dfrac{\begin{matrix}[A]\\ \vdots \\ B\end{matrix}}{A \Rightarrow B}\Rightarrow\mathcal{I}$	$\lambda x_i^A.\ v$
$\dfrac{\begin{matrix}\vdots \\ A\end{matrix} \qquad \begin{matrix}\vdots \\ A \Rightarrow B\end{matrix}}{B}\Rightarrow\mathcal{E}$	$t\ u$

Curry Howard's isomorphism between natural deduction and Simply Typed Lambda Calculus

Other variants

Logic side	Programming side
Intuitionistic Natural Deduction	Simply typed Lambda Calculus
Hilbert’s style Natural Deduction	Typed Combinatory Logic
System F	Polymorphic Lambda Calculus
Classical Natural Deduction	Typed Lambda-Mu Calculus

Difference Japan-France

- System of student seminar
- No internship as in France
- Place of Master program in the curriculum : often linked with License program in France, but with PhD program in Japan

Difference Japan-France

- System of student seminar
- No internship as in France
- Place of Master program in the curriculum : often linked with License program in France, but with PhD program in Japan

Since at the crossroad of Mathematics and Informatics

→ Don't know whether these differences are due to differences Japan/France or to differences Mathematicians/ Computer scientists