# Syntactic and Semantic properties of useful $\lambda$-calculi: Church-Rosser, reducibility, realisability

Vincent Rahli
supervisors: Professor Fairouz Kamareddine and Doctor J. B. Wells

ULTRA group, MACS, Heriot Watt University

August 6, 2008

# A bit of History
## formalisation of Mathematics (and functions)

- First formalisation of the concept of function by Frege (1879, premises of the formalisation of Mathematics)
- Discovery of some **paradoxes** in Mathematics (around 1900).
- Functions can be applied to any function: reflexiveness.
- Formalisation of the concept of **type** by Russell to restrict application of functions (1908).
- Formalisation of Mathematics: design of logical systems

- Design of the **$\lambda$-calculus** by Church as part of a formal system for logic and functions (1932).
- The full system was inconsistent:
  - Church uses the type free $\lambda$-calculus to investigate functions (successful model for computation);
  - Church adds simple types ($\mathrm{int}$, $\mathrm{int} \rightarrow \mathrm{int}$) to $\lambda$-calculus in a system with logical axioms to deal with logic and function.
- Functions are studied as **computational rules** rather than as sets of pairs.

# A bit of History

## Improvement of the type systems

- Definition of typed programming languages such as ML: a statically typed functional programming language based on a polymorphic type system.
- Discovery that types in a type system can be associated to formulae in a logical system and that the proofs of formulae can be associated to typable terms: Curry-Howard isomorphism.
    - Realisability semantics: connection between recursive functions and intuitionism.
    - Reducibility: semantic method based on realisability to prove properties of calculi.

# Outline

- The $\lambda$-calculus, its variants and their properties.
- Semantics of typed $\lambda$-calculi using realisability.
- Application of the extension of a typed $\lambda$-calculus.
- Plan of the thesis.

Let Var be a countably infinite set of variables and $x, y, z, f \in \text{Var}$.

$$M, N \in \Lambda ::= x \mid (\lambda x.M) \mid (MN)$$

Evaluation:
the **$\beta$-reduction** is the compatible closure[†] of the following rule:

$$(\lambda x.M_1)M_2 \rightarrow_\beta M_1[x := M_2]$$

Extensionality:
The **$\eta$-reduction** is the compatible closure of the rule:

$$\lambda x.Mx \rightarrow_\eta M, \qquad \text{if } x \notin \mathrm{fv}(M)$$

[†] if $M_1 \rightarrow_\beta M_2$ then $\lambda x.M_1 \rightarrow_\beta \lambda x.M_2$ and $M_1 N \rightarrow_\beta M_2 N$ and $NM_1 \rightarrow_\beta NM_2$

For example: the Church-Rosser property.

for all $M$, if $M$ reduces to $M_1$ and $M_2$ then there exists $M_3$ such that $M_1$ and $M_2$ both reduce to $M_3$.

$$\begin{tikzcd}
 & 4+2+3 \arrow[dl] \arrow[dr] & \\
6+3 \arrow[dr] & & 4+5 \arrow[dl] \\
 & 9 &
\end{tikzcd}$$

Usual steps of a proof of the Church-Rosser property of a $\lambda$-calculus:

- Introduction of a new relation (developments).
- Proof of the confluence of this new relation.
- Equivalence between:
  - the transitive closure of the new relation
  - the reflexive and transitive closure of the reduction relation of the considered calculus.

The proofs of the Church-Rosser property can be divided as follows:

- First division:
  - Encoding the development using a reduction relation: Tait and Martin-Löf (1972), Takahashi (1989).
  - Encoding the development using a set of terms: Barendregt et al. (1972), Ghilezan and Kunčak (2001), Koletsos and Stavrinos (2007).
- Second division:
  - Using a semantic method: Koletsos and Stavrinos.
  - Using a syntactic method: Barendregt et al., Tait and Martin-Löf, Takahashi.

Our contribution:

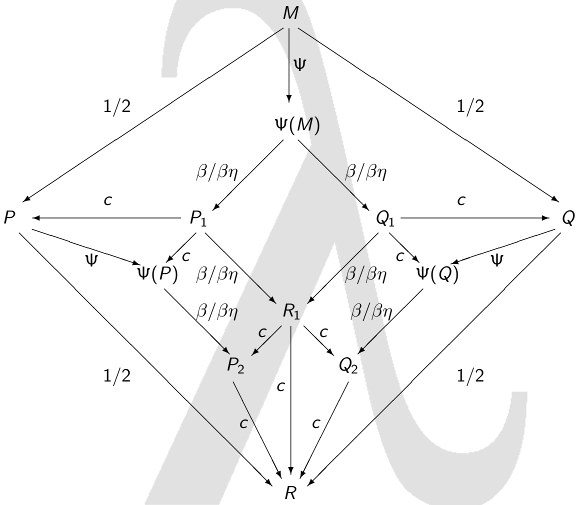
- We extended Koletsos and Stavrinos's semantic proof to the $\beta\eta$-case: [KRW08].
- We simplified and extended Koletsos and Stavrinos's proof as well as that of Ghilezan and Kunčak to obtain a syntactic proof: [KR08].

Our second method is a syntactic proof based on the encoding of developments using sets of terms rather than a reduction relation.

Advantages of our second method:

- We do not deal with types as Koletsos and Stavrinos (or Ghilezan and Kunčak) and our proof is simpler than similar syntactic proofs such as the one of Barendregt et al.
- Our proof of the confluence of developments is parametric (we can easily prove the finiteness of developments).
- Our proof can be seen as a bridge between semantic proofs and syntactic proofs

Our proof of this property is as follows:



Our proof of this property is as follows:

confluence of the simple calculus

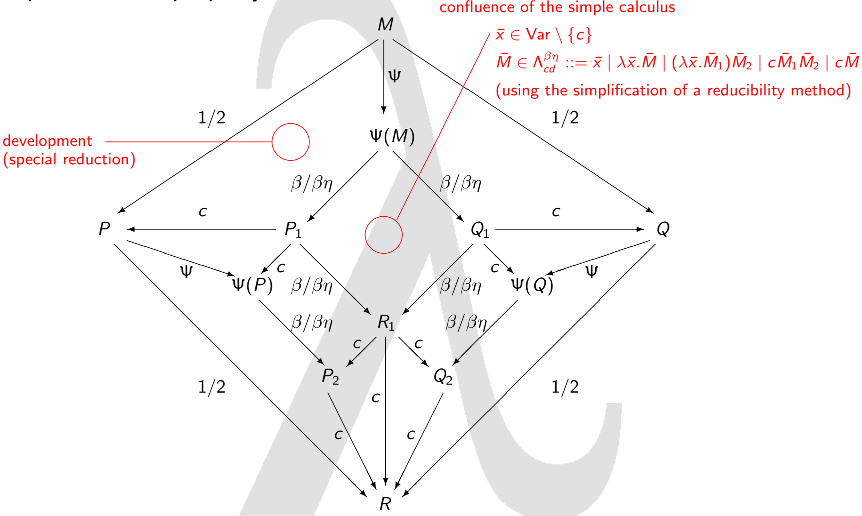
$\bar{x} \in \mathsf{Var} \setminus \{c\}$

$\bar{M} \in \Lambda_{cd}^{\beta\eta} ::= \bar{x} \mid \lambda\bar{x}.\bar{M} \mid (\lambda\bar{x}.\bar{M}_1)\bar{M}_2 \mid c\bar{M}_1\bar{M}_2 \mid c\bar{M}$

(using the simplification of a reducibility method)

$$
\begin{tikzcd}
 & & & M \arrow[ddlll, "1/2"'] \arrow[d, "\Psi"] \arrow[ddrrr, "1/2"] & & & \\
 & & & \Psi(M) \arrow[dl, "\beta/\beta\eta"'] \arrow[dr, "\beta/\beta\eta"] & & & \\
P \arrow[dr, "\Psi"] \arrow[dddrrr, "1/2"'] & & P_1 \arrow[ll, "c"'] \arrow[dl, "c"] \arrow[dr, "\beta/\beta\eta"] & & Q_1 \arrow[rr, "c"] \arrow[dl, "\beta/\beta\eta"'] \arrow[dr, "c"] & & Q \arrow[dl, "\Psi"'] \arrow[dddlll, "1/2"] \\
 & \Psi(P) \arrow[dr, "\beta/\beta\eta"] & & R_1 \arrow[dl, "c"'] \arrow[dr, "c"] \arrow[dd, "c"] & & \Psi(Q) \arrow[dl, "\beta/\beta\eta"'] & \\
 & & P_2 \arrow[dr, "c"] & & Q_2 \arrow[dl, "c"'] & & \\
 & & & R & & &
\end{tikzcd}
$$

Our proof of this property is as follows:



Our proof of this property is as follows:

Confluence of developments:

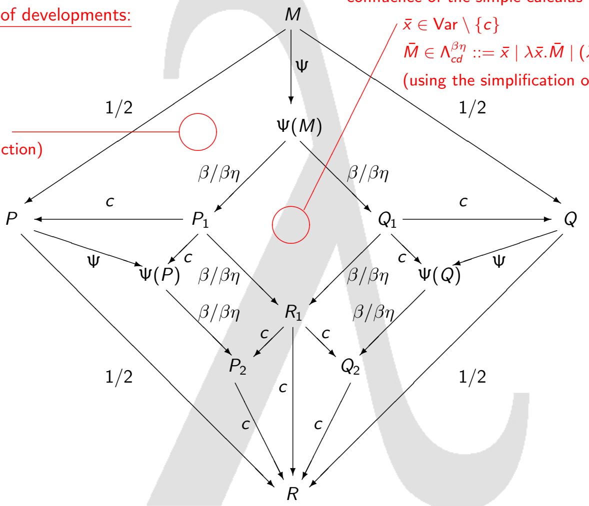
confluence of the simple calculus

$\bar{x} \in \text{Var} \setminus \{c\}$

$\bar{M} \in \Lambda_{cd}^{\beta\eta} ::= \bar{x} \mid \lambda\bar{x}.\bar{M} \mid (\lambda\bar{x}.\bar{M}_1)\bar{M}_2 \mid c\bar{M}_1\bar{M}_2 \mid c\bar{M}$

(using the simplification of a reducibility method)

development
(special reduction)



Our proof of this property is as follows:

Confluence of developments:

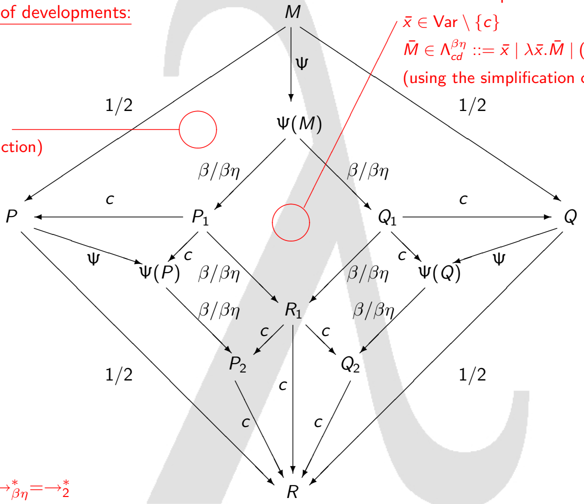
confluence of the simple calculus

$\bar{x} \in \mathrm{Var} \setminus \{c\}$

$\bar{M} \in \Lambda_{cd}^{\beta\eta} ::= \bar{x} \mid \lambda\bar{x}.\bar{M} \mid (\lambda\bar{x}.\bar{M}_1)\bar{M}_2 \mid c\bar{M}_1\bar{M}_2 \mid c\bar{M}$

(using the simplification of a reducibility method)

development
(special reduction)




$\to_{\beta}^{*} = \to_{1}^{*}$ and $\to_{\beta\eta}^{*} = \to_{2}^{*}$

(Simulation of a reduction by a some developments)

# Semantics of typed $\lambda$-calculi using realisability

Different ways to cast some lights on a calculus:

- Denotational semantics
  - Answer to the question: What terms denote?
- Operational semantics
  - Answer to the question: How terms compute?
- **Realisability semantics**
  - Highlight the computational content of a syntactic object.
- ...

Definition of some concepts:

**Principal typing property:** A type system satisfies the principal typing property if for each typable term, there is a typing from which all other typings are obtained via some set of operations

**Intersection type system:** As the $\forall$ quantifier, intersection types allow to express polymorphism but in a finite way. Intersection types are lists of usages ($\mathsf{int} \rightarrow \mathsf{int} \cap \mathsf{real} \rightarrow \mathsf{real}$).

**Expansion:** Introduced by Coppo, Dezani and Venneri (1980) in order to restore the principal typing property in such systems (extensively improved by Carlier and Wells (2008))

The $\lambda$-term: $M = (\lambda x.x(\lambda y.yz))$

can be assigned the two following typings:

$$\Phi_1 = \langle (z : a) \vdash (((a \to b) \to b) \to c) \to c \rangle \qquad \text{(principal)}$$

$$\Phi_2 = \langle (z : a_1 \sqcap a_2) \vdash (((a_1 \to b_1) \to b_1) \sqcap ((a_2 \to b_2) \to b_2) \to c) \to c \rangle$$

An expansion operation can obtain $\Phi_2$ from $\Phi_1$

In System E (Carlier et al. (2004)), the typing $\Phi_1$ is replaced by:

$$\Phi_3 = \langle (z : ea) \vdash (e((a \to b) \to b) \to c) \to c \rangle$$

$\Phi_2$ can be obtained from $\Phi_3$ by substituting for $e$ the expansion term:

$$E = (a := a_1, b := b_1) \sqcap (a := a_2, b := b_2)$$

Two steps so far:

- [KNRW08c, KNRW08a]: We provided a complete realisability semantics for an intersection type system with one expansion variable and no expansion mechanism.
- [KNRW08b, KNRW08a]: We provided a complete realisability semantics for an intersection type system with an infinite set of expansion variables and no expansion mechanism.

How do we do that:

- Design of a calculus aiming at the capture of the meaning of an expansion variable: encapsulation of a type.
  - $\lambda$-calculus indexed with natural numbers/list of natural numbers
- Design of a suitable type interpretation.
  - An expansion variable make the realisers change level.
- Proof of the soundness and completeness of the semantics w.r.t. a given type system.

# Application of the extension of a typed $\lambda$-calculus
## Type Error Slicing [HW04]

- The aim: accurately identify and report the location of some type errors of a piece of code (for a SML-based programming language), by providing a set of minimal and necessary collection of points in the piece of code (a slice).

- How does it do that?
  - From a piece of code, type error slicing first generates a set of **type constraints**.
  - Then, from this set of type constraints, it runs the **enumeration** of the set of **minimal** errors of the piece of code.
  - Finally, for each minimal error found, it displays the corresponding **slice** (parts of the piece of code corresponding to the minimal error).

The current implementation of Type Error slicing is for a larger language than the theory.

# Application of the extension of a typed $\lambda$-calculus

## Type Error Slicing [HW04]

Our contribution so far:

- Extraction of the different modules of the implementation of Type Error Slicing:
  - Constraint generation
  - Unification
  - Minimisation
  - Enumeration
  - Slicing

  ➤ Largely been achieved.

- Proof of the properties of these modules: Correctness and Termination.

  ➤ Partially achieved.

# Plan of the thesis

During the third year I will focus on the two following subjects:

- **Type error slicing:**
  - Finishing the proofs relative to the current implementation of type error slicing.
  - Extending the Type Error Slicing framework for a rich and sophisticated programming language.
  - Implementing the type error slicer we will develop for the rich language so that we can make our development more practical and can have more impact.
- **Semantics of expansion:** providing a semantics of a typed $\lambda$-calculus with expansion which takes into consideration the expansion mechanism.

Christian Haack and J. B. Wells.
Type error slicing in implicitly typed higher-order languages.
*Science of Computer Programming*, 50(1-3):189–224, 2004.

Fairouz Kamareddine, Karim Nour, Vincent Rahli, and Joe B. Wells.
Challenges and solutions to realisability semantics for intersection types with expansion variables.
Submitted to Fundamenta Informaticae, 2008.

Fairouz Kamareddine, Karim Nour, Vincent Rahli, and Joe B. Wells.
A complete realisability semantics for intersection types and arbitrary expansion variables.
In *ICTAC'08: 5th International Colloquium on Theoretical Aspects of Computing, The Marmara, Istanbul, Turkey, 1-3 September 2008*, volume 5160 of *Lecture Notes in Computer Science*, pages 171–185, 2008.

Fairouz Kamareddine, Karim Nour, Vincent Rahli, and Joe B. Wells.
Developing realisability semantics for intersection types and expansion variables.
Presented to ITRS'08, 4th Workshop on Intersection Types and Related Systems, Turin, Italy, 25 March 2008, 2008.

Fairouz Kamareddine and Vincent Rahli.
Simplified reducibility proofs of church-rosser for $\beta$- and $\beta\eta$-reduction.
Accepted at LSFA'08, Salvador, Bahia, Brasil, 26 August, 2008.

Fairouz Kamareddine, Vincent Rahli, and J. B. Wells.
Reducibility proofs in the $\lambda$-calculus.
Submitted to Fundamenta Informaticae, 2008.