First Year Report: Realisability methods of proof and semantics with application to expansion

Supervisors : Professor Fairouz Kamareddine

Doctor Joe B. Wells

Student : Vincent Rahli

July 26, 2007

Contents

1	Introduction	3
2	Definitions and notations	4
3	Untyped λ -Calculus 3.1 Presentation of the λ -Calculus	5 5 7
4	Typed λ-calculus 4.1 A brief history on type systems 4.2 Properties of Type Systems 4.3 Intersection Types 4.4 Example of an intersection type system	9 11 12 13
5	Expansion	15
6	Realisability	16
7	Work done during the first year and Ph.D. Goals 7.1 Semantics of expansion using realisability	19 19 21
8	A reflection on what to come	23
9	Conclusion	24

1 Introduction

After the discovery of some controversial results in analysis during the nineteenth century [42], a number of mathematicians and logicians have taken the formalization of Mathematics seriously. A new area of research was then developed whose aim is to give solid foundations for mathematics and to rid the subject of any shaky foundations.

A proof in Mathematics is usually a certain reasoning based on presupposed statements leading to a result to which one is accorded a certain truth value. This leads to the definition of logical systems which enable to derive denotations of mathematical statements from a set of postulates and to define a process of reasoning using logical rules.

Mathematicians and logicians such as Frege, Russell, Curry, Church, etc. participated in the construction of such systems. In [60], Russel presents some contradictions that exist in Mathematics from which he hopes the subject to recover. According to Russell, the main problem is how functions are defined and to which range of objects they can be applied. In order to avoid contradictions, Russell [60] defined a type theory where types are used to restrict the domain of the application of functions.

A remarkable improvement in the formalization of Mathematics is the design of the λ -calculus [12], a powerful system which enables the study of functions, but unfortunately which does not reach its initial purpose. Since then, important properties of this calculus have been widely studied in the literature. These properties include the confluence property (also associated with the consistency of the calculus [59]) or the normalization property (also associated with processes of evaluation of the calculus). In order to prove these properties, different methods have been developed and then improved and generalized. For example, two well known methods to prove the confluence of the λ -calculus are the method of parallel reductions and the method of finiteness of developments, both presented in [4].

Another remarkable improvement has been the introduction of type systems in which proofs are introduced as part of the defined theory. This led to the discovery that types in type system can be associated to formulae in logical system and that proofs of formulae can be associated to typable terms. This association is known as the Curry-Howard or the Curry-De Bruijn-Howard isomorphism. Brouwer, Heyting and Kolmogorov already suggested this isomorphism in their BHK-semantics [2, 63, 64] which interprets formulae of intuitionistic logic by proofs, which are constructive methods based on functions. Kleene [49] also proposed an interpretation, called realisability, which stresses the connection between recursive functions and intuitionism.

Many applications have been found to realisability. Initially, realisability has been designed as a method to interpret formulae, i.e. in the semantics domain. This semantics enables to stress the constructivity of systems. Based on realisability, Tait [62] developed a method called reducibility to prove properties of the λ -calculus (such as normalization properties). Since then, this method has been improved by Girard [29, 30], Koletsos [52] and Gallier [22, 23, 24, 25] amongst others.

So far we have discussed the λ -calculus, type systems and the proof methods known as realisability/reducibility. A system of λ -calculus with types allows different expressive power depending on the interactions that exist between λ -terms and types. Eight representative λ -calculi with types have been generalized in the well known λ -cube of Barendregt [5]. This λ -cube enables to express different kinds of abstraction, allowing to express among other things, polymorphism. The most popular way to express polymorphism is using the quantifier \forall as is the case in the system F of Girard. Coppo and Dezani introduced another way to express

polymorphism in [15] using intersection types. These intersection types are lists of usages. Because of the ramified structure of these types, Coppo, Dezani and Venneri introduced in [17] an operation called expansion in order to restore the principal typing property in such systems (in fact, to be able to calculate any typing from a principal one). Since then, this operation has been extensively improved [11, 10].

In Section 2, we give some common notations used in this report. In Section 3, we introduce the syntax of the λ -calculus formalized by Church in 1932 [12] and some principal properties of the λ -calculus. These properties have been widely studied in the literature. As a matter of fact, each such property has been established (for the whole λ -calculus or some subsets thereof) using a number of different proofs. Each time the calculus is updated, these properties have to be re-established from scratch and the existing proofs, may or may not apply to the updated calculus. Hence, new methods of proofs may have to be found or existing proof methods will have to be seriously updated in order to apply. Hence, a challenge in the domain is to find some generalizations of these proofs that can be applied to a wide range of calculi. A method which may help in this way is the reducibility method based on realisability which is introduced in Section 6. Given a type system, this method consists of an interpretation of types based on a property which we want to be held by the typable terms. So, we present type systems in Section 4, focusing particularly on those in which we have been interested during this year: intersection type systems introduced in [15]. In Section 5 we present the expansion process which has been introduced within the framework of principal typing for intersection type systems [17]. Thereafter, in Section 7 we present the work done during this year and in Section 8 we present our research plans for the forthcoming period of the thesis. Finally, we conclude in Section 9.

2 Definitions and notations

We assume familiarity with the notations of term rewriting, lambda calculus and type theory as can be found in [4, 5, 54, 3].

Usually, we let n, m, i, j, k range over the set of natural numbers Nat.

When we define a metavariable, say a, to range over a set, we implicitly define, the matavariable a_i , $\forall i \geq 0$, to range over the same set.

Let S and T, be any sets and let a and b range over S and T. We generally define a pair of elements of S and T as $(a,b) \in S \times T$, where $a \in S$ is the first projection and $b \in T$ is the second projection.

Let R be a binary relation on $S \times T$. R can be seen as the set of pairs $(a, b) \in S \times T$ such that a and b are in relation by R. We write aRb or $(a, b) \in R$ if a and b are in relation by R.

Given two relations $R_1 \subseteq \mathcal{S} \times \mathcal{T}$ and $R_2 \subseteq \mathcal{T} \times \mathcal{U}$, we define $R_1 \circ R_2 = \{(a, c) \in \mathcal{S} \times \mathcal{U} \mid \exists b \in \mathcal{T} \text{ such that } (a, b) \in R_1 \wedge (b, c) \in R_2\}.$

If R is a binary relation on $S \times S$, we define:

- $R^{-1} = \{(b, a) \mid (a, b) \in R\}.$
- $R^0 = \{(a, a) \mid a \in \mathcal{S}\}.$
- $R^{i+1} = R^i \circ R \text{ for } i > 0.$
- $R^+ = \bigcup_{i>0} R^i$ (the transitive closure of R).
- $R^* = R^+ \cup R^0$ (the reflexive and transitive closure of R).

• $=_R = (R \cup R^{-1})^*$ (the symmetric, reflexive and transitive closure of R).

We use the usual notation $2^{\mathcal{S}}$ to denote the set $\{\mathcal{T} \mid \mathcal{T} \subseteq \mathcal{S}\}$, the set of subsets of \mathcal{S} or usually called the set of parts of \mathcal{S} .

3 Untyped λ -Calculus

3.1 Presentation of the λ -Calculus

For an introduction to the λ -calculus, see [4, 59] for examples.

Following the work done by Frege, Russel, Curry, etc., Church, in [12], introduced a system for "the foundation of formal logic". The set of terms of this system was defined as a superset of the so-called λI -calculus. In addition, Church introduced two sets of postulates. The first one called "rules of procedure" which allows one, among other things, to deal with conversion of λ -terms. The second set contains the "formal postulates" which are logical axioms. However, this system and some of its subsystems turned out to be inconsistent as shown by Kleene and Rosser in [50]. Nevertheless, the subsystem dealing only with functions (in fact, the generalization of which is the λ -calculus) appears to be a "successful model for computable functions" [4]. As stressed in the introduction of Barendregt's book, this theory presents functions as rules, and not as sets of pairs, in order to deal with their computational aspects. This λ -calculus appears to be a generalization of the definition of functions given, for example, by Russel ("propositional functions") as explained in [42]. A modern and common presentation of the λ -calculus is as follows:

$$M \in \Lambda ::= x \mid (\lambda x.M) \mid (M_1 M_2)$$

where x,y,z range over an enumerable infinite set of λ -terms variables $\mathcal V$ (or just variables when no ambiguity occurs) and M,N,P,Q,R range over Λ . The syntactic forms belonging to the set Λ are called λ -terms, or just terms. We use the usual convention about parenthesis and we omit them when no confusion arises. If a λ -term appears in (is a part of) another λ -term, we say that the former one is a subterm of the latter one. We write $M_1 \subset M_2$ when M_1 is a subterm of M_2 , i.e. if $M_1 \in \mathsf{SubTerm}(M_2)$ where $\mathsf{SubTerm}(M_2)$ is defined by induction on the structure of M_2 :

$$\left\{ \begin{array}{l} \mathsf{SubTerm}(x) = \{x\} \\ \mathsf{SubTerm}(\lambda x. M_3) = \{M_2\} \cup \mathsf{SubTerm}(M_3) \\ \mathsf{SubTerm}(M_3 M_4) = \{M_2\} \cup \mathsf{SubTerm}(M_3) \cup \mathsf{SubTerm}(M_4) \end{array} \right.$$

A λ -term of the form $\lambda x.M$ stands for a λ -abstraction, or just abstraction, and denotes the function M in which the variable x represents its argument. In $\lambda x.M$, the variable x, which has been abstracted, is said to be bound. When a variable is not bound in a λ -term it is said to be free. The set of free variable of a term can be defined, by induction on M, as: $FV(x) = \{x\}$, $FV(\lambda x.M) = FV(M) \setminus \{x\}$ and $FV(MN) = FV(M) \cup FV(N)$. A term of the form MN stands for the application of M to N. Hence, in the λ -calculus, each λ -term stands for a function with one argument. We abbreviate $(\dots((M_1M_2)M_3)\dots M_n)$, where $n \geq 2$, by $M_1M_2\dots M_n$. The process which allows us to compute the application of a λ -abstraction to a λ -term is called β -reduction and is usually presented as the compatible closure (we define this closure after the definition of context) of the following rule:

$$(\beta) \qquad (\lambda x.M)N \to_{\beta} M[x := N]$$

where M[x := N] is the result of the substitution of all occurrences of the variable x in M by N. This process of computation was already enunciated as above in

[12] (one of the "rules of procedure") and called "conversion". In the same article Church states two other rules of conversion, now known as β -expansion and α -conversion. The first one enables to convert M[x:=N] into $(\lambda x.M)N$ and the second one enables to convert $\lambda x.M$ into $\lambda y.M[x:=y]$ when the variable y does not occur in M. We consider the λ -terms equivalent modulo α -conversion, i.e. =, the equivalence relation between λ -terms is the reflexive, transitive, symmetric and compatible closure of the rule:

(
$$\alpha$$
) $\lambda x.M \to_{\beta} \lambda y.M[x := y]$ if y does not occur in M

As pointed out in [4], "lambda terms denote processes" and so, from the observation that the two processes $\lambda x.Mx$ and M (where $x \notin FV(M)$) yield the same result MN when applied to a process N, arises the reduction rule:

$$(\eta)$$
 $\lambda x.Mx \to_{\eta} M$, where $x \notin FV(M)$

As for \rightarrow_{β} , we define \rightarrow_{η} as the compatible closure (see below) of the rule (η) .

Let $r \in \{\beta, \eta\}$. In rule (r), the λ -term on the left of the \to_r is called an r-redex or, just a redex when no ambiguity arises, and the λ -term on the right of the \to_r is called its contractum [36]. We define \mathcal{R}^r to be the set of λ -terms which are r-redexes. We define $\to_{\beta\eta}$ to be $\to_{\beta} \cup \to_{\eta}$ and $\mathcal{R}^{\beta\eta} = \mathcal{R}^{\beta} \cup \mathcal{R}^{\eta}$. Let $r \in \{\beta, \eta, \beta\eta\}$. We prefer the notation $=_r$ instead of $=_{\to_r}$ for the symmetric, reflexive and transitive closure of \to_r .

In order to define the compatible closure, we define \mathcal{C} a set of λ -context¹ (or just context when no ambiguity arises). A λ -context is a λ -term in which a subterm has been replaced by a "hole", denoted by the symbol \square , which is an empty space which may be filled by any λ -term. We define this set as follows:

$$C \in \mathcal{C} ::= \Box \mid (\lambda x.C) \mid (CM \mid MC)$$

The function -[-] which enables us to fill the "hole" is defined by induction on its first argument as follows:

$$\begin{cases}
\Box[M] = M \\
\lambda x. C[M] = \lambda x. C[M] \\
(CM_1)[M_2] = C[M_2]M_1 \\
(M_1C)[M_2] = M_1C[M_2]
\end{cases}$$

Then, the compatible closure of (β) may be defined as the follows:

$$M_1 \to_{\beta} M_2 \Rightarrow \forall C \in \mathcal{C}, C[M_1] \to_{\beta} C[M_2]$$

We may also redefine the subterm relation as follows:

$$M_1 \subset M_2 \iff \exists C \in \mathcal{C}, M_2 = C[M_1]$$

A redex might occurs more than once in a λ -term. For example, the redex, $(\lambda x.x)y$ occurs twice in the λ -term $((\lambda x.x)y)((\lambda x.x)y)$. The λ -contexts clearly help to distinguish the two different occurrences. For our particular example, let $C_1 = ((\lambda x.x)y)\Box$ and $C_2 = \Box((\lambda x.x)y)$. Obviously, $M = C_1[R] = C_2[R]$. When M = C[R], we say that C is the associated context to the particular occurrence of the redex R in the λ -term M. For our particular example, C_1 (resp. C_2) is the associated context to the second (resp. first) occurrence of the redex $(\lambda x.x)y$. We can easily

¹Our definition differs from the one in [4] in the sense that a context contains one and only one "hole". This is the definition of a one-hole context of [8].

prove that contexts associated with different occurrences of a redex are themselves different. We denote the occurrences of a redex using the λ -context associated to these occurrences (as in [8]). Let $r \in \{\beta, \eta, \beta\eta\}$. We define $\mathcal{R}_M^r = \{C \mid C \in \mathcal{C} \land \exists R \in \mathcal{R}^r, C[R] = M\}$ which denotes the set of occurrences of redexes in M. If $M_1 \to_{\beta} M_2$ results by contracting the occurrence of the r-redex R in $M_1 = C[R]$ then by definition $C \in \mathcal{R}_{M_1}^r$ and we write $M_1 \xrightarrow{C} M_2$ where $M_2 = C[R']$ and R' is the contractum of R.

As pointed by Rosser in [59], some important questions need to be answered about the λ -calculus taken as part as a logical system:

• Some forms of consistency has been proved since the conception of λ -calculus such as the Theorem 1 of [14] of which a stronger form is now well known as the Church-Rosser theorem. A binary relation R on Λ satisfies the Church-Rosser property if:

$$\forall M, M_1, M_2, MR^*M_1 \land MR^*M_2 \Rightarrow \exists M_3, M_1R^*M_3 \land M_2R^*M_3.$$

Let $\mathsf{CR} = \{ M \mid M \to_{\beta}^* M_1 \wedge M \to_{\beta}^* M_2 \Rightarrow \exists M_3, M_1 \to_{\beta}^* M_3 \wedge M_2 \to_{\beta}^* M_3 \}$. It has been shown that each λ -term belongs to the set CR , i.e. \to_{β} satisfies the Church-Rosser property. We will come back on this property in section 3.2.

- Church enunciates a result of completeness know as the Church's thesis: "Effectively calculable functions from positive integers to positive integers are just those definable in the λ-calculus". Since, the intuitive notion of "Effectively calculable function" has been clarified: a "general recursive function" according to Godel and Kleene, "calculable on a Turing machine" according by Turing.
- Finally, Scott was the first to find a model of the λ-calculus. He built a model based on the category of complete lattices with continuous functions. Since, his model has been simplified and others have been built. We may cite for example the syntactical model (called λ-model) developed by Hindley and Longo in [39]. The filter model of [7] and the "simple semantics" of [38] turn out to be λ-models.

We have to notice that the results obtained in [14] are stated for a subsystem of the λ -calculus called λI -calculus. This calculus is defined as the λ -calculus but requires the variable x to belong to the set FV(M) when constructing the λ -abstraction $\lambda x.M$. A term of the λI -calculus is called a λI -term or just a term when there is no ambiguity.

3.2 Properties of the λ -calculus

In this section we recall some of the most important properties of the λ -calculus that we will be concerned with in this thesis.

Normalization

The Corollary 2 of [14] (corollary of the theorem 1) enunciates that if a λI -term possesses a normal form, then its normal form is unique.

A β -normal form, or just a normal form, is a λ -term M such that for every λ -term $M' \neq M$, M does not reduce to M' by \rightarrow_{β}^* . The set of β -normal form is defined as following:

$$\mathsf{NF} = \{ M \mid \forall M' \neq M, M \not\to_\beta^* M' \}.$$

²a Turing machine is the abstract idea of a computer.

We say that a λ -term M_1 possesses a β -normal form M_2 if $M_2 \in NF$ and $M_1 \to_{\beta}^* M_2$. This set as been shown equivalent to the following one, which presents the shape of a β -normal form:

$$\{\lambda x_1 \dots \lambda x_m . x M_1 \dots M_n \mid n, m \ge 0 \land \forall i \in \{1, \dots, n\}, M_i \in \mathsf{NF}\}$$

We call WN the set of λ -terms which possesses a β -normal form:

$$\mathsf{WN} = \{ M \mid \exists M' \in \mathsf{NF}, M \to_\beta^* M' \}.$$

A λ -term which belongs to WN is called weakly normalizable. A λ -term M belongs to the set of terms SN, if all the reductions started from M are finite:

$$\mathsf{SN} = \{ M \mid \exists n \geq 0, \forall M', M \to_\beta^m M' \Rightarrow m \leq n \}$$

A λ -term which belongs to SN is called strongly normalizable.

Normalization properties have been useful to prove consistency of the λ -calculus and some other properties, but they are also interesting on their own. β -reduction is the computation process of the λ -calculus. Hence, the β -normal form of a λ -term denotes the final result of the process, the result of the evaluation of the λ -term. Since, $\mathsf{SN} \neq \Lambda$ (the simplest example is $\Delta\Delta$ where $\Delta = \lambda x.xx$) it is not possible to find a function which associates to a λ -term its β -normal form.

Church-Rosser

In the literature, there are different names to refer to the Church-Rosser property. For example, we can find the name Church-Rosser in [59, 26, 27, 4] to refer to the property which we presented in the section 3.1. In [52, 53] a λ -term is said to be possessing the Church-Rosser property if it belongs to the set CR. We can also find the name confluence in, for example, [26, 27]. Sometimes the following property on R (a binary relations on Λ) is called Church-Rosser [14, 18]:

$$\forall M_1, M_2, M_1 =_R M_2 \Rightarrow \exists M_3, (M_1, M_3) \in R^* \land (M_2, M_3) \in R^*.$$

It has been shown that a binary relations on Λ possesses the latter Church-Rosser property if and only if it possesses the former one.

In order to illustrate the consistency of the λ -calculus, take the two syntactically distinct β -normal forms x and $y(\lambda z.z)$. Suppose $x =_{\beta} y(\lambda z.z)$, then by the second definition of the Church-Rosser property, $\exists M$ such that $x \to_{\beta}^* M$ and $y(\lambda z.z) \to_{\beta}^* M$. Since x and $y(\lambda z.z)$ are normal forms $x = y(\lambda z.z)$. So the hypothesis is wrong, $x \neq_{\beta} y(\lambda z.z)$. Hence, for example, two λ -terms can be shown not belonging to the relation $=_{\beta}$ is they possess two syntactically distinct normal forms.

Standardization

In order to prove normalization theorems, some methods turned out to be very useful. For example, the standardization method, which consists in defining a standard order in which redexes have to be reduced. For example, a well known standard order is the systematic choice of the head β -redex, followed by the systematic choice of an internal β -redex: if the λ -term M is of the form $\lambda x_1 \dots \lambda x_m . (\lambda x. M_0) M_1 \dots M_n$ such that $m \geq 0$ and $n \geq 1$ then the β -redex $(\lambda x. M_0) M_1$ is called the head β -redex of M or just head redex when there is no ambiguity. We write $M_1 \to_h M_2$ if $M_1 = C[\lambda x_1 \dots \lambda x_m.(\lambda x. M_0) M_1 \dots M_n]$ and $M_2 = C[\lambda x_1 \dots \lambda x_m.M_0[x := M_1] M_2 \dots M_n]$, where $m \geq 0$ and $n \geq 1$. We define $\to_i = \to_\beta \setminus \to_h$. If $M_1 \to_i M_2$, the reduction is said to be internal. In [4], it is proved that if $M_1 \to_\beta^* M_2$ then $\exists M_3$ such that $M_1 \to_h^* M_3 \to_i^* M_2$.

Another well known order is the systematic choice of the left-most β -redex. If the λ -term M possesses a β -redex (i.e. $\mathcal{R}_M^\beta \neq \emptyset$) we can easily see that it is either a λ -abstraction or an application and the left-most β -redex of M (Left-Most-Redex(M) $\in \mathcal{R}_M^\beta$) may be defined as follows:

```
 \begin{cases} & \mathsf{Left\text{-}Most\text{-}Redex}(\lambda x.M_1) = \lambda x.\mathsf{Left\text{-}Most\text{-}Redex}(M_1) \\ & \mathsf{Left\text{-}Most\text{-}Redex}(M_1M_2) = \square & \text{if } M_1M_2 \in \mathcal{R}^\beta \\ & \mathsf{Left\text{-}Most\text{-}Redex}(M_1M_2) = \mathsf{Left\text{-}Most\text{-}Redex}(M_1) & \text{if } M_1M_2 \notin \mathcal{R}^\beta \end{cases}
```

If $n \geq 0$, $M_1 \stackrel{C_1}{\to}_{\beta} M_2 \dots \stackrel{C_{n-1}}{\to}_{\beta} M_n$ and $\forall i \in \{1, \dots, n-1\}$, Left-Most-Redex $(M_i) = C_i$ then we write $M_1 \to_{\beta_l}^* M_n$ which is called a standard reduction. It has been shown in [18] that if $M_1 \to_{\beta}^* M_2$ then $M_1 \to_{\beta_l}^* M_2$. In [4], Barendregt uses the previous standardization result to prove this second one.

Hence, a standard reduction defines a reduction strategy and trivially, if a λ -term M_1 possesses a β -normal form M_2 then there exists a standard reduction from M_1 to M_2 .

Developments

Another well known method uses the result of finiteness of "reduction of residuals" or finiteness of "developments". This result has been proved for various calculi and various reductions, by different methods [18, 6, 51, 35, 36, 54]. This result turned to be useful in proofs of the Church-Rosser property [18, 6]. Given an initial set of redexes of a λ -term M, a development of M is a reduction whose each step reduces only a residual of a redex of the initial set. Krivine, in [54], introduced a more or less formal method to define residuals of a redex. This method has been improved in [53] and we use it in [47]. In order to not enter in technical details which can be found in the previous references, we simply present an example of residual. Let $M = \lambda y.((\lambda x.M)N)((\lambda z.z)P)$ and $R = (\lambda x.M)N$. Since $R \in \mathcal{R}^{\beta}$, $C = \lambda y.\square((\lambda z.z)P) \in \mathcal{R}^{\beta}_{M}$. Hence, $M \xrightarrow{C}_{\beta} C[M[x := N]] = \lambda y.M[x := N]((\lambda z.z)P) = M'$. Since $(\lambda z.P)Q \in \mathcal{R}^{\beta}$, then $C' = \lambda y.((\lambda x.M)N)\square \in \mathcal{R}^{\beta}_{M}$. Now, $C'' = \lambda y.M[x := N]\square \in \mathcal{R}^{\beta}_{M'}$ is called a residual of C'.

4 Typed λ -calculus

"Illative systems were introduced for overcoming the weakness of pure λ -calculus as a support for studying properties of functions and logical deductions" [17].

4.1 A brief history on type systems

Even if types were already present in Mathematics before [60], Russel was the first one who explicitly developed a theory of types in [60]. His aim was to go over some contradictions present in some previous systems. These contradictions share a common characteristic (called "reflexiveness") which can be avoided by the so-called "vicious-circle principle" defined to be the condition "no totality can contain members defined in terms of itself". Russell defines a type to be "the collection of arguments for which a function has values" (domain of a function) and uses them to restrict the application of a term to a term. The paradox discovered by Russel may be stated as follows: let $\mathcal{X} = \{a \mid a \notin a\}$ then by definition $\mathcal{X} \in \mathcal{X}$ if and only if $\mathcal{X} \notin \mathcal{X}$. As done in section 5G of [18], $\mathcal{X} \in \mathcal{X}$ may be represented by the λ -term $P = (\lambda x. \neg (xx))(\lambda x. \neg (xx))$ where \neg is a new constant symbol. Note that since $P \to_{\beta} \neg P$, there exists an only one reduction from P which is infinite: $P \to_{\beta} \neg P \to_{\beta} \neg (\neg P) \to_{\beta} \dots$

Initially, Church intended the λ -calculus to be a system for logic and functions. As noticed in section 3.1, the system defined by Church in [12] turns out to be inconsistent. To solve the problem, Church used type free λ -calculus to investigate functions [13], and in order to deal with logic and functions he added simple types to λ -calculus giving us the "simple theory of types". The simple theory of types is based on a decorated (terms decorated with types) subsystem of λ -calculus and a set of "formal axioms" (logical axioms), where types are used to avoid paradoxes. At approximately the same time, the theory of functionality of Curry and Feys [18] has been a notable development in the nowadays called the simply typed λ -calculus (which can be expressed in the Curry or the Church style [5]). The simply typed λ -calculus is a type system based on functional types (simple types are built from a set of type variables, a set of type constants and a functional type). If we use \rightarrow to denote the functional type and if σ and τ are two simple types, $\sigma \rightarrow \tau$ stands for the type of a function which associate an object of type τ to an object of type σ .

Type theory, since its invention, has been aimed at avoiding the paradoxes. Initial type systems were too restricted to allow strong expressiveness. The literature is rife with attempts to enriching type systems so that more expressiveness can be achieved. The literature is also full of such type systems which later turned out to be inconsistent. The most representative type systems varying in power have been depicted in the so called λ -cube of Barendregt [5]. The simply typed lambda-calculus is the simplest type system in the λ -cube. This λ -cube enables to express different kinds of abstraction, introducing different systems with different powers:

- \bullet Terms depending on types (polymorphism): for example system F of Girard,
- Types depending on types (type operators): for example system F_{ω} of Girard,
- Types depending on terms (dependent types): for example system λP of de Bruijn.

Polymorphism in system F is introduced by means of the quantifier \forall . Since [15] another way is known to express polymorphism in a type system: intersection types (see section 4.3).

A modern type system is usually defined by a set of terms, a set a types and a derivability relation. Usually the derivability relation is defined on the set of terms, the set of types, a set of environments of typing and a set of typing rules. A typing rule is a scheme composed by a premise and a conclusion which enables to derive an instantiation of the conclusion from an instantiation of the premise. A rule without premise is called an axiom. Let S a type system defined by $\mathcal X$ a set of terms, $\mathcal Y$ be a set of types, $\mathcal Z$ a set of environments of typing, and \vdash a derivability relation built from $\mathcal X$, $\mathcal Y$, $\mathcal Z$ and a set of typing rules. Let Rule be any of these typing rules. Then, $\Gamma \vdash M : \sigma$, states that the type $\sigma \in \mathcal Y$ can be assigned to the term $M \in \mathcal X$ in the environment $\Gamma \in \mathcal Z$ using the derivability relation \vdash . The pair (Γ, σ) is called a typing of M. As defined in [5], $M \in \mathcal X$ is called legal w.r.t. the type system (or just legal when no ambiguity arises) if $\exists \Gamma \in \mathcal Z$ and $\exists \sigma \in \mathcal Y$ such that $\Gamma \vdash M : \sigma$. Similarly, $\sigma \in \mathcal Y$ is called legal w.r.t. the type system (or just legal when no ambiguity arises) if $\exists \Gamma \in \mathcal Z$ and $\exists M \in \mathcal X$ such that $\Gamma \vdash M : \sigma$.

A derivation may be seen as a tree in which each node is a type judgment. A link between two nodes denotes the instantiation of a typing rule: the node from which there is a path to a leaf not including the other one is the premise of the instantiation of the typing rule, the other one is the conclusion. The root of the tree is the conclusion of the derivation.

4.2 Properties of Type Systems

Subject Reduction and Subject Expansion

Properties which we may want to be satisfied by a type system include the properties that typings are preserved by evaluation and/or by a defined equivalence relation. The properties which insure that the previous properties are held by the type system are the subject reduction and subject expansion properties. The subject reduction property states that typings are preserved by reduction. The subject expansion property states that typings are preserved by expansion. For example, given a type system based on the λ -calculus and β -reduction, the subject reduction property is defined as follows:

If
$$\Gamma \vdash M : \sigma$$
 and $M \to_{\beta}^* M'$ then $\Gamma \vdash M' : \sigma$

Principal Typing

Another property which we may want to be satisfied is the principal typing property. A type system has the principal typing property if for each typable term, there is a particular typing, from which it is possible to obtain all the other typings by some operations.

It is easy to see that in most type systems, a term may be assigned many types. Existence of a principal type scheme come from the fact that the terms represent some "abstract notions", it "shows an internal coherence between all functional characters" of a term [34, 17]. If we consider as interpretation of types, the set of terms which can be typed by this type (a realisability semantics) w.r.t. a type system, we can ask ourselves what it means that a term belongs to the interpretation of two different types and it realizes two different types. Using a simple notion of substitution, Hindley proved in [34], among other things, that each object in combinatory logic which possesses a type in the defined type system (the one defined in [18]), has a principal type scheme.

Since, the definition of principal type scheme has been improved. The definition has been extended to principal typing, taking care of the context in which a term is typed. In [68], Wells gives a new definition, independent from the type system in which the term is typed, of the principal typing of a term. His definition of a principal typing is as follows:

- Let $S \triangleright M : t$ if $t = (\Gamma, \sigma)$ and the typing judgment $\Gamma \vdash M : \sigma$ is derivable in the type system S.
- Let $t \leq_S t'$ if and only if $\mathsf{Terms}_S(t) \subseteq \mathsf{Terms}_S(t')$ where $\mathsf{Terms}_S(t) = \{M \mid S \rhd M : t\}$.
- "A typing t is principal in system S for term M iff $S \triangleright M : t$ and $S \triangleright M : t'$ implies $t \leq_S t'$ "

Strong normalization

A type system is said to be strongly normalizable if each term typable in the type system is strongly normalizable. In type systems where no reductions take place in the types this question boils down to the study of the normalization of the term M in a typing judgment $\Gamma \vdash M : \sigma$ (derivable in a such type system). However, in the syntax of the λ -cube, there is no distinction between types and terms, they are all called "pseudo-expressions". So, the study of the strong normalization of a type system in the λ -cube is then the study of the normalization of the expressions X_1 and X_2 in a typing judgment $\Gamma \vdash X_1 : X_2$ (derivable in a such type system).

Hence, a type system of the λ -cube is said to be strongly normalizable if whenever the typing judgment $\Gamma \vdash X_1 : X_2$ is derivable in the type system, then both X_1 and X_2 are strongly normalizable. Barendregt proves the strong normalization of each type system of the λ -cube in [5].

4.3 Intersection Types

Intersection types have been introduced in [15] for studying the property that a λ -term possesses a normal form. In that article it is proved, among other results, that if a λ -term is typable then it possesses a β -normal form (it belongs to WN). The word "intersection" in "intersection type" comes from the fact that, if types are interpreted by sets (a set-theoretical semantics), usually, the intersection types are interpreted by intersection of sets. The authors extended their type system in [16] in order to type more terms than those typable by simple types (see section 4.1) (each λ -term belonging to SN turns out to be typable) and to overcome some problems of those type systems such as the non preservation of types by conversion (this means that a term and its reducts can be typed by different types).

We use \cap as the intersection type constructor. A term t which possesses the type $\sigma \cap \tau$ possesses the types σ and τ at the same time. I.e., an intersection type can be seen as a list of types. Thus, an intersection type turns out to be a polymorphic type. For example, a program of type $(\sigma \to \sigma) \cap (\tau \to \tau)$ is either a program calculating a term of type σ from a term of type σ or a program calculating a term of type τ from a term of type τ . The polymorphism of an intersection type is said to be "parametric" [9], since a program which possesses an intersection type works "uniformly" on the range of the types given by the intersection. This is in contrast to "ad-hoc" polymorphism which is the polymorphism of the union types (one program is given for each type of the union). The quantifier " \forall " is well known to express polymorphism as in system F designed by Girard [29, 30]. Advantages of intersection type systems over type systems with the \forall quantifier are well explained in [11] and include:

- Urzyczyn, in [65], (theorem 3.1) found a term which is not typable in the system F_{ω} : $M = (\lambda x.z(x(\lambda f.\lambda u.fu))(x(\lambda v.\lambda g.gv)))(\lambda y.yyy)$ but which turns to be typable in the rank-3³ restriction of intersection types.
- Wells proved in [67] that type inference in system F is undecidable. However, in [48], Kfoury and Wells defined an intersection type system for which every finite-rank restriction has a decidable type inference.
- Wells proved in [68] that system F does not have principal typings for all terms and proved together with Kfoury in [48] that every finite-rank restriction of their defined intersection type system has principal typings.

Since then, intersection type systems have been improved. In [7], the authors give a proof of the soundness (which means that the interpretation of a type contains all the interpretation of the terms typable by this type) and the completeness (which means that the interpretation of a type contains all and only the interpretation of the terms typable by this type) of an intersection type system. This result is stated for all the lambda models [39], including the so-called filter model. Hindley proved a similar result in [37], but using, among others, a semantics called "simple semantics", which is a realisability semantics as defined by Krivine.

Most of the recent intersection type systems involve a subtyping rule [1, 19, 20]. This rule plays a significant part in the power of these type systems. This rule uses a defined preorder relation on types. As it is explained in, for example, [1],

³the notion of rank is, for example, explained in [11]

a preorder relation is built in a way that the pair constituted by the set of types and the preorder relation turns to be a meet semi-lattice: the meet of two types being the intersection of these types. Moreover, regarding the interpretation of types as sets, the interpretation of the intersection type as intersection of sets and the interpretation of the preorder relation as inclusion of sets, the definition of the preorder relation turns out to be quite natural.

Some intersection type systems involve a constant type Ω as a 0-ary version of the intersection types. This type expresses a universality in the sense that this type does not contain any information. Regarding the interpretation of types as sets, this type is usually interpreted by the universe of discourse.

Some generalized results have been proved recently on intersection type systems. In [19, 20], the authors give a characterization of complete intersection type systems w.r.t some set-theoretical semantics similar to those established for the simple types used by Hindley (see section 6). In [1], the authors give a characterization for a intersection type system to possess the invariance of interpretation by β -conversion.

4.4 Example of an intersection type system

During the first year of the Ph.D. studies on which this report is based, we used different intersection type systems for different purposes. One such system was used in [47] to prove some properties of the λ -calculus. It is also one of the two type systems introduced in [27], to prove, among other things, that $\Lambda = \mathsf{CR}$. This type system is called $\lambda \cap^{\Omega}$. It is similar to the one defined in [7] but not equivalent: for example, $\Delta \Delta$ such that $\Delta = \lambda x.xx$ can have the type $\Omega \to \Omega$ in [7], but not in [27], using the non-structural rule $\Omega \leq \Omega \to \Omega$ (non-structural in the sense of [58]). Then, we briefly define the type system D using the definition of $\lambda \cap^{\Omega}$. This system D is, for example, used in [54, 53].

Let \mathcal{A} be an enumerable infinite set of type variables and let $\Omega \notin \mathcal{A}$ be a constant type. The sets of types Type is defined as follows:

$$\sigma \in \mathsf{Type} ::= \alpha \mid \sigma_1 \to \sigma_2 \mid \sigma_1 \cap \sigma_2 \mid \Omega$$

We let α range over \mathcal{A} and $\sigma, \tau, \rho, \ldots$ range over Type. \cap is not associative, commutative or idempotent (see after the definition of $\lambda \cap^{\Omega}$, how to get these properties).

We now define some concepts necessary to define (intersection) type systems and some notations:

- A basis [5] is an element of the following set: $\mathcal{B} = \{\Gamma = \{x : \sigma \mid x \in \mathcal{V}, \sigma \in \mathsf{Type}\} \mid \forall x : \sigma, y : \tau \in \Gamma, \text{ if } \sigma \neq \tau \text{ then } x \neq y\}.$ We let Γ range over \mathcal{B} .
- The domain of a basis is $DOM(\Gamma) = \{x \mid x : \sigma \in \Gamma\}.$
- When $x \notin DOM(\Gamma)$, we write $\Gamma, x : \sigma$ for $\Gamma \cup \{x : \sigma\}$.

We are now able to define the type system called $\lambda \cap^{\Omega}$ in [27]:

- Let ∇ be the set of axioms of figure 1, $\nabla = \{(ref), (tr), (in_L), (in_R), (\rightarrow -\cap), (mon'), (mon), (\rightarrow -\eta), (\Omega), (\Omega' lazy)\}.$
- The relation \leq^{∇} is defined on types Type and the set of axioms ∇ .
- The equivalence relation is defined by: $\sigma \sim^{\nabla} \tau \iff \sigma \leq^{\nabla} \tau \land \tau \leq^{\nabla} \sigma$.
- We define $\lambda \cap^{\Omega}$ to be the type system $(\Lambda, \mathsf{Type}, \vdash)$ such that \vdash is type derivability relation on \mathcal{B} , Λ and Type generated using the typing rules of Figure 2.

Commutativity, associativity and idempotence are given by the preorder relation and specially by the axioms (in_L) , (in_R) , (mon'), (tr) and (ref):

Figure 1: Ordering axioms on types

```
\frac{\overline{\Gamma, x : \sigma \vdash x : \sigma}}{\Gamma \vdash M : \sigma} \stackrel{(ax)}{\longrightarrow} \frac{\overline{\Gamma \vdash M : \Omega}}{\Gamma \vdash M : \tau} \stackrel{(\Omega)}{\longrightarrow} 

\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M : \tau} \stackrel{(\to_E)}{\longrightarrow} \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x . M : \sigma \rightarrow \tau} \stackrel{(\to_I)}{\longrightarrow} 

\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \cap \tau} \stackrel{(\cap_I)}{\longrightarrow} \frac{\Gamma \vdash M : \sigma \quad \sigma \leq^{\nabla} \tau}{\Gamma \vdash M : \tau} \stackrel{(\leq^{\nabla})}{\longrightarrow}
```

Figure 2: Typing rules

- Commutativity: by (in_R) , $\sigma \cap \tau \leq^{\nabla} \tau$ and by (in_L) , $\sigma \cap \tau \leq^{\nabla} \sigma$ so by (mon'), $\sigma \cap \tau \leq^{\nabla} \tau \cap \sigma$. By (in_L) , $\tau \cap \sigma \leq^{\nabla} \tau$ and by (in_R) , $\tau \cap \sigma \leq^{\nabla} \sigma$ so by (mon'), $\tau \cap \sigma \leq^{\nabla} \sigma \cap \tau$. Hence, $\sigma \cap \tau \sim^{\nabla} \tau \cap \sigma$.
- Associativity: by (in_R) , $(\sigma \cap \tau) \cap \rho \leq^{\nabla} \rho$, by (in_L) , $(\sigma \cap \tau) \cap \rho \leq^{\nabla} \sigma \cap \tau$, by (in_R) , $\sigma \cap \tau \leq^{\nabla} \tau$, by (in_L) , $\sigma \cap \tau \leq^{\nabla} \sigma$, so by (tr), $(\sigma \cap \tau) \cap \rho \leq^{\nabla} \sigma$ and $(\sigma \cap \tau) \cap \rho \leq^{\nabla} \tau$. By (mon'), $(\sigma \cap \tau) \cap \rho \leq^{\nabla} \tau \cap \rho$ and again by (mon'), $(\sigma \cap \tau) \cap \rho \leq^{\nabla} \sigma \cap (\tau \cap \rho)$. By (in_L) , $\sigma \cap (\tau \cap \rho) \leq^{\nabla} \sigma$, by (in_R) , $\sigma \cap (\tau \cap \rho) \leq^{\nabla} \tau \cap \rho$, by (in_L) , $\tau \cap \rho \leq^{\nabla} \tau$, by (in_R) , $\tau \cap \rho \leq^{\nabla} \rho$, so by (tr), $\sigma \cap (\tau \cap \rho) \leq^{\nabla} \tau$ and $\sigma \cap (\tau \cap \rho) \leq^{\nabla} \rho$. By (mon'), $\sigma \cap (\tau \cap \rho) \leq^{\nabla} \sigma \cap \tau$ and again by (mon'), $\sigma \cap (\tau \cap \rho) \leq^{\nabla} (\sigma \cap \tau) \cap \rho$. Hence, $(\sigma \cap \tau) \cap \rho \sim^{\nabla} \sigma \cap (\tau \cap \rho)$.
- Idempotence: by (in_L) , $\sigma \cap \sigma \leq^{\nabla} \sigma$ and by (ref) and (mon'), $\sigma \leq^{\nabla} \sigma \cap \sigma$, hence, $\sigma \sim^{\nabla} \sigma \cap \sigma$.

We refer in section 6 to an intersection type system called system D. This system is defined as for $\lambda \cap^{\Omega}$ without the type Ω (on Type^D) and without the typing rules (Ω) and (\leq^{∇}) but adding two typing rules (\cap_{E_1}) and (\cap_{E_2}) allowing the elimination of intersection:

$$\begin{split} \sigma \in \mathsf{Type}^D &::= \alpha \mid \sigma_1 \cap \sigma_2 \mid \sigma_1 \to \sigma_2 \\ \frac{\Gamma \vdash M : \sigma \cap \tau}{\Gamma \vdash M : \sigma} & (\cap_{E_1}) & \frac{\Gamma \vdash M : \sigma \cap \tau}{\Gamma \vdash M : \tau} & (\cap_{E_2}) \end{split}$$

Associativity, commutativity and idempotence of the intersection type are generally useless since by rules (\cap_I) , (\cap_{E_1}) and (\cap_{E_2}) , in system D we have: $\Gamma \vdash M : (\sigma \cap \tau) \cap \rho$ if and only if $\Gamma \vdash M : \sigma \cap (\tau \cap \rho)$; $\Gamma \vdash M : \sigma \cap \tau$ if and only if $\Gamma \vdash M : \tau \cap \sigma$; and $\Gamma \vdash M : \sigma \cap \sigma$ if and only if $\Gamma \vdash M : \sigma$.

Note that the introduction of rules (\cap_{E_1}) and (\cap_{E_2}) is not necessary to define system D. It suffices to keep the rule (\leq^{∇}) and to restrict ∇ to $\{(in_L), (in_R)\}$ or even $\{(in_L), (in_R), (tr), (ref)\}$ as proved in [47]. Hence, using ∇ to range over a set of axioms enables to easily define a set of intersection type systems with various powers.

```
(\alpha := \sigma, E)\alpha
                                                                                      \sigma
                                                         (e := E, E')e
                                                                                      E
E \odot
                           E
                                                         (\alpha := \sigma, E)\alpha'
                                                                                      E if \alpha \neq \alpha'
                           X
\odot X
                                                         (e := E, E')e'
                                                                                      E' if e \neq e'
                          eEX
(eE)X
                                                                                      (SE)X
                                                         S(EX)
(eE)X
                          e(EX)
                                                                                      SX_1 \cap SX_2
                                                         S(X_1 \cap X_2)
(E_1 \cap E_2)X
                           (E_1X)\cap (E_2X)
                                                         S\Omega
\Omega X
                                                                                      (\alpha := S\sigma, SE)
                                                         S(\alpha := \sigma, E)
                                                         S(e := E, E')
                                                                                      (e := SE, SE')
```

Figure 3: Expansion Application

5 Expansion

In the intersection type system framework, substitution is not anymore a sufficient operation to obtain all the types of a term from a principal one (see section 4.2) since the ramified structure of the types involving intersection types complicates the process of obtaining any type of a term from a principal one.

The expansion operation has been introduced in [17] for calculating any type which can be assigned to a λ -term from a principal one, in a defined intersection type systems.

As explained in [11], the mechanism of expansion "simulates on a typing the effect of inserting uses of the intersection-introduction typing rule into a derivation of that typing". Since the previous mechanism of expansion was quite complicated, expansion variables have been introduced in [48] to simplify this mechanism. Then, this work has been generalized and extended [11].

In [10], Carlier defines expansion in such a way that (we adapt his definitions to the intersection type system $\lambda \cap^{\Omega}$ introduced in section 4.4):

- The author introduces an enumerable infinite set of expansion variables, say E-var, over which e ranges.
- Then, he introduces the introduction rule for expansion variables which is similar to the following one:

$$\frac{\Gamma \vdash M : \sigma}{e\Gamma \vdash M : e\sigma} \ e_I$$

where $e\Gamma = \{x : e\sigma \mid x : \sigma \in \Gamma\}$. The use of this rule marks a typing judgment in a derivation, in order to be able, using a substitution replacing the expansion variable by an expansion, to operate an expansion on its types simulating the replacement of this rule by some typing rules in the derivation.

• The set of expansions is defined as follows:

$$E \in \mathsf{Expansion} ::= E_1 E_2 \mid E_1 \cap E_2 \mid \Omega \mid \boxdot \mid S$$

Where \boxdot is a constant called "null expansion" acting as a unit when applied to a type or a term (see figure 3) and where S ranges over S, the set of substitutions defined as following:

$$S \in \mathcal{S} ::= \alpha := \sigma, E \mid e := E_1, E_2$$

where σ ranges over Type^E , an extension of the set of types defined in section 4.4 in order to include the application of an expansion to a type:

$$\sigma \in \mathsf{Type}^E ::= \alpha \mid \sigma_1 \to \sigma_2 \mid \sigma_1 \cap \sigma_2 \mid \Omega \mid E\sigma$$

• Then, he defines a set of equality to apply expansions and substitutions. Let $X \in \mathsf{Entity} ::= \sigma \mid M$. The equality rules are defined in figure 3.

A key point is that the introduction of the expansion rule is admissible:

$$\frac{\Gamma \vdash M : \sigma}{E\Gamma \vdash M : E\sigma} E_I$$

A typing rule is admissible w.r.t a type system, or just admissible when no ambiguity arises, when the conclusion can be obtained from the premise, using the set of typing rules associated to the type system.

Such a conclusion is obtained by the application of a substitution on a type judgment. $S(\Gamma \vdash M : \sigma)$ stands for $S\Gamma \vdash M : S\sigma$, where $S\Gamma = \{(x : S\sigma) \mid (x : \sigma) \in \Gamma\}$.

The presentation of expansion above is a restricted version of that given in [10], adapted to the syntax presented in section 4.4.

We have to notice that expansion is more than the part that we are presenting here. But this part seems to be a good starting point to understand what expansion is and how it can be used.

6 Realisability

The aim of intuitionism, whom Brouwer was the instigator, is to reject all the non-constructive principles of mathematics. In order to illustrate the viewpoint of a constructivist, we give a common example which can be found in [64, 33]: given A a mathematical statement which has not been proved or refuted, let p be a natural number equal to 1 if A is verified and 0 otherwise. But, while A has not been decided, there does not exist a method for calculating p. This kind of definition is rejected by a constructivist. We can extract from this example the rejection of the Principle of Excluded Middle (called PEM and stating that, given the mathematical statement A, the statement " $A \lor \neg A$ " is a valid axiom of mathematics). "Constructively, accepting PEM as a general principle means that we have a universal method for obtaining for any proposition A, either a proof of A or a proof of $\neg A$ " [64].

Thus, the central viewpoint of intuitionism and of constructive mathematics is to consider proofs as objects. According to Brouwer, intuitionistic truth means provability (see [2]), i.e. the existence of a proof. Heyting and Kolmogorov formalized this semantics which has become known as the BHK⁴ semantics and which is recognized as the semantics for intuitionistic logic. A proposition is interpreted by the set of its proofs and then true propositions are those inhabited by proofs.

This semantics is also known as Heyting's semantics in contrast to Tarski's semantics, which is a denotational semantics (see [31]). Two entities having the same semantics if they have the same denotation, i.e. the same result w.r.t. some operations.

The formalization of the BHK semantics by Heyting and Kolmogorov treating proofs as objects led to the association of propositions to types in type systems (propositions-as-types). The BHK semantics proposes to interpret the implication $A \Rightarrow B$, where A and B are mathematical statements, by the set of functions which associate a proof of B to a proof of A. Regarding the set of proofs of A and the set of proofs of B as types, this suggests to regard the implication $A \Rightarrow B$ as a functional type of such functions w.r.t. a type system.

In that way, the Curry-Howard isomorphism states a correspondence between systems of formal logic and type systems. Since proofs are associated to functions (constructive objects), for a long time, the Curry-Howard isomorphism was

⁴for Brouwer, Heyting and Kolmogorov

only stated for intuitionistic logic. But recent work showed that some programming methods ("call/cc" which means "call with current continuation" and which is a kind of exception) correspond to the proofs by contradiction (see the work by Krivine).

Realisability semantics was first introduced by Kleene in [49] for studying intuitionistic arithmetic. As explained by Hilbert and Bernays, as well as by Kleene, an existential statement, such that $\exists x.t$, is an incomplete translation of the pair consisting of a term (witness) which may be substituted for x and a proof of t in witch x is substituted by the witness. Kleene's realisability method explains how to interpret incomplete statements by complete ones. An existential statement is said to be incomplete since it refers to an information which is not given in the statement. Kleene presents realisability as "a kind of intuitionistic truth notion".

In [49], Kleene constructs realizers of formulas of arithmetic in such a way that if a formula is an arithmetic formula then there exists a realizer which realizes the formula. A realizer is defined as a natural number. As pointed in [2], realizers of Kleene are not proofs but make the computational content, of arithmetic formula, explicit.

For a survey on realisability and references, read [2, 66, 63, 64]. In [2], Artemov claims that "Kleene itself denied any connection of his realisability with BHK interpretation". But in [64], the authors claim that if "realizers" are regarded as "proofs" and "partial recursive function" (allowing the interpretation of implication) are regarded as "construction" then the realisability of Kleene is a variant of the BHK interpretation. Whatever the issue, the BHK-interpretation as well as the realisability semantics try to give an intuitionistic interpretation of mathematics statements.

Since then, other realisability semantics have been defined. For example, in [56], Lipton defines a realisability semantics where realizers are elements of a "partial combinatory algebra" based on the combinators of Curry. This semantics has been proved sound for the intuitionistic Zermelo-Fraenkel set theory. In [54], Krivine proves a soundness result of a realisability semantics for the system AF_2 (second order functional arithmetic). As explained in [66] realisability has been applied to many fields connected to the λ -calculus.

Realisability in proof semantics

Realisability is used, among other things, in model theory. We can cite for example the work of Krivine [54, 55], the work of Hindley [38, 37] or the work of Kamareddine and Nour [21, 44]. In [44] (or in [38]), a type is interpreted by a set of λ -terms depending on the form of the type. These interpretations are defined in order to interpret types by saturated sets. A set is said to be saturated if it is closed by an expansion relation. Given the β -reduction (see section 3.1, rule β) of the λ -calculus, if a λ -term M belongs to a saturated set of λ -terms, each λ -term which β -reduces to M has to belong to this set. We can define the set of saturated sets of λ -terms as follows:

$$\mathsf{Sat} = \{ \mathcal{S} \mid M \in \mathcal{S} \land N \to_{\beta} M \Rightarrow N \in \mathcal{S} \}$$

So, the interpretation of a type is proved to be an element of Sat. The property of an interpretation to be a saturated set is natural regarding the reduction relation. In the context of λ -calculus, a λ -term M which reduces to the λ -term N realizes the same specification than N. Moreover, this property turns out to be necessary for proving some other properties such as the soundness of the semantics.

Recall the Propositions As Types/Proofs As Terms (PAT) principle of the Curry-Howard isomorphism. The interpretation of a type (which according to PAT can be

regarded as a proposition) turns out to be a saturated set of λ -terms (which according to PAT can be regarded as a set of proofs) realizing the type. In other words, each λ -terms belonging to the interpretation of a type, realizes the specification denoted by the type.

Given a type system and a realisability semantics, an important property is the completeness of the semantics w.r.t. the type system: a term belongs to the interpretation of a type if and only if it is typable by this type. This property enables to give a meaning to the type judgment derived by the type system. Moreover, this property enables to characterize the realizers of a type. As presented in section 4.3, the if direction is called the soundness (or correctness or adequacy) property.

Realisability in proof of term properties

In addition to the uses mentioned above, realisability has been very useful in the characterization of properties of the λ -calculus and type systems. The method, based on realisability, used to prove properties of the λ -calculus is known as reducibility. There are many kinds of reducibility methods, but most of them consist of the choice of an interpretation of types, such that for each type, its interpretation is a subset of a set of λ -terms satisfying a property, such that SN or CR. Then, it remains to find some conditions such that if a λ -term is typable then it belongs to the interpretation of its type (soundness). Hence, each typable λ -term possesses the property.

A lot of work has been done in this way. In particular, we mention the following:

- Tait [62]: Tait developed the reducibility method in [62] to prove the strong normalization of the terms typable in the system T of Gödel, which may be regarded as an extension of the theory of functionality of Curry and Feys [18] or an extension of the simply typed λ -calculus. Terms of this system are the two combinators of Curry, a combinator for recursion and the two symbols 0 and S (for successor) which enable to represent natural numbers (S0 stands for the natural number 1, S(S0) stands for the natural number 2, etc.) and types are simple types. Types are interpreted by sets of convertible terms (a notion related to the strong normalization): to each type and each term of this type, one can associate a predicate stating the normalization of the terms provable true in another system.
- Girard [29, 30]: As explained by Gallier in [23], in order to prove the strong normalization of the terms typable in type systems like F or F_{ω} , Girard [29, 30] defines what he calls "candidates of reducibility", which are sets of (typed) terms satisfying certain closure conditions such as each of these candidates is a subset of SN. In this process, the interpretation of each type is a candidate of reducibility and a term typable by a certain type is shown to belong to the interpretation of this type.
- Koletsos [52]: Koletsos proves that the set of λ -terms typable in the simply typed λ -calculus is a subset of CR using a reducibility method. The difference between this and the work of Girard is that the interpretation of types is based on the set CR and not on the set SN.
- Krivine [54]: Krivine proves the strong normalization of the λ -terms typable in the system D (see section 4.4) using a reducibility method.
- Barendregt [5]: Barendregt proves the strong normalization of each term of each typing judgment derivable in his λ -cube. This proof may be divided into two main proofs.

- The first is the implication of the strong normalization of each term of each typing judgment derivable in the most powerful system of the λ -cube (the Calculus of Construction) by the strong normalization of each term of each typing judgment derivable in one type system of the λ -cube (F_{ω}) .
- The second is the proof of the strong normalization of each term of each typing judgment derivable in F_{ω} using amongst other things a reducibility method.
- Gallier [22, 24, 25]: Gallier introduces some properties that a set of λ -terms with a given property needs to satisfy, to either contain the set of λ -terms typable in a given type system, or be equal to the set of λ -terms typable in a given type system. This is, for example, done for systems D and F_{ω} . Among other things, Gallier generalizes, in [23], the work of Girard.
- Ghilezan and Likavec [27]: This work can be seen as an extension of [25]. They state some properties that a set of λ-terms with a given property has to satisfy, to either contain the set of λ-terms typable in a given type system, or be equal to the set of λ-terms of the λ-calculus. But, as we showed in [47], it turns out that these results are both false. In [47] we propose a solution to repair the first statement, but the reparation of the second one seems to be a challenge. Instead of being an extension of the work of Gallier in [25], the corrected version of [27] turns out to be a different version of a part of the work done in [25].

As an example, Koletsos and Stravinos in [53] use a reducibility method to prove $\Lambda = \mathsf{CR}$ using the type system D. In that work, types are interpreted as follows: let \mathcal{I} be a function from \mathcal{A} to 2^{Λ} , then an interpretation of a type is a function form Type^D (see section 4.4) to 2^{Λ} :

- $\llbracket \alpha \rrbracket_{\mathcal{I}} = \mathcal{I}(\alpha)$,
- $\llbracket \sigma \to \tau \rrbracket_{\mathcal{I}} = \{ M \mid \forall N \in \llbracket \sigma \rrbracket_{\mathcal{I}}, MN \in \llbracket \tau \rrbracket_{\mathcal{I}} \} \cap \mathsf{CR},$
- $\bullet \ \llbracket \sigma \cap \tau \rrbracket_{\mathcal{I}} = \llbracket \sigma \rrbracket_{\mathcal{I}} \cap \llbracket \tau \rrbracket_{\mathcal{I}}.$

Then \mathcal{I} is taken to be a function such that $\forall \alpha \in \mathcal{A}, \mathsf{CR}_0 \subseteq \mathcal{I}(\alpha) \subseteq \mathsf{CR}$ where $\mathsf{CR}_0 = \{xM_1 \dots M_n \mid \forall i \in \{1, \dots, n\}, M_i \in \mathsf{CR}\}.$

The definition of interpretation of Ghilezan and Likavec in [27] is similar to that of [53]. However, they use, in addition, an interpretation of λ -terms, which, finally, turns out to be useless as we show in [47].

7 Work done during the first year and Ph.D. Goals

This section is a summary of three articles [45, 46, 47] written by Fairouz Kamareddine, Karim Nour, Joe B. Wells and Vincent Rahli. These articles are attached to this report since there is no relevance to integrate them in this report since they would make this report too long considering the guidelines.

7.1 Semantics of expansion using realisability

Realisability is a powerful and intuitive tool of the semantics domain. After having given the syntax of expansion (see section 5), we now want to know what the meaning of expansion is. In particular, we need to give the meaning of an expanded type and of the mechanism of expansion. Moreover, there is still no semantics for type systems with expansion.

The author of this report has been involved in the first steps towards such a semantics:

- In [45] a semantics was given for an intersection type system with only one expansion variable and without any expansion mechanism. In that article (which is attached at the end of this report), a realisability semantics is given to two different type systems (given in definition 20 of [45]). It is well known how to interpret functional types, intersection types and type variables. But it is not so clear how to interpret expansion types (in [45], this only means an expansion variable applied to a type). Then, rather than dividing the space of meaning (the domain of interpretation), it is considered as a hierarchy of the space of meaning, and an expansion type is interpreted in a higher level than that of the expanded type (see definition 16 of [45]). Moreover, two other restrictions are taken into consideration:
 - The first is, as observed previously, that this work does not deal with the expansion mechanism at all.
 - The second is the restriction of a calculus without K-redexes, in order to avoid the complication of the Ω -type.

The domain of interpretation is then the λI -calculus with additional information, added to variables, called degrees given in definition 1 of [45] and called $\lambda I^{\mathbb{N}}$ -calculus. These degrees enable the change of the space of meaning. Then, a degree is associated to each term in this calculus and the increase of the degrees of the variables of a term increases the degree of the term (see definition 7 and lemma 8 of [45]).

The soundness property is then shown for both systems in lemma 38, whereas completeness is held only for the second one (theorem 53). This is due to a the lack of an elimination rule for the intersection type as shown in remark 42. Moreover, because of the choice of interpretation of expansion types, completeness cannot be proved for the second type system if we make distinction between two different expansion variables. Indeed, the expansion variable of an expansion type is not taken into account when the interpretation of the type is given. The defined semantics does not differentiate an expansion variable from another one. As we can see in definition 16, $[e_1\sigma]_{\mathcal{I}} = [e_2\sigma]_{\mathcal{I}}$ even if $e_1 \neq e_2$, where \mathcal{I} is a function from \mathcal{A} to $2^{\mathcal{M}^0}$ (which interprets type variables, \mathcal{M}^0 is the set of terms of the $\lambda I^{\mathbb{N}}$ -calculus with degree 0) and $-^+$ is the function which increases the degree of a term (extended to a set of terms).

• To solve this problem, a second article [46] has been written in order to give a semantics of an intersection type system using an infinity of expansion variables. This intersection type system uses a more complex calculus called $\lambda^{\mathbb{L}_{\mathbb{N}}}$ -calculus (see definition 1) and the semantics differentiates two different expansion variables (see definition 49). In this calculus, degrees are replaced by lists of degrees. These lists of degrees enable the change of the space of meaning, as the degrees did in [45]. Now, the change is not denoted by the increasing of the degrees but by the addition of a degree in the lists. The difference between two degrees enables to differentiate an expansion variable from another one: $\llbracket e_1 \sigma \rrbracket_{\mathcal{I}}^{-1} \rrbracket_{\mathcal{I}} = \llbracket \sigma \rrbracket_{\mathcal{I}}^{-1} \rrbracket_{\mathcal{I}}^{-1} = \llbracket \sigma \rrbracket_{$

One of the goals of this Ph.D., is to find a semantics of a more powerful intersection type system using an infinity of expansion variables and taking care of the expansion mechanism, for example as defined in section 5. The initial steps carried out in [45, 46] have been a good preparation to this goal and will play a crucial role in the rest of the Ph.D.

7.2 Realisability in proof of term properties

As presented in section 6, reducibility is a powerful method used to prove properties of the λ -calculus. In [47], we discover that the results given in [27] are false, we correct these results and we extend those given in [53]. Again, [47] is attached at the end of this article and in the rest of this section, we give a brief summary of what exactly we do there. This section needs to be read together with [47].

How we discover that the reducibility method of [27] fails

One of the principal result of [27, 28] is $\mathsf{CR} = \Lambda$. The proof of this result as given in [27] is based on the type system called $\lambda \cap^{\Omega}$, presented in section 4.4 of this report. Let $\mathcal{P}, \mathcal{X} \subseteq \Lambda$. Basically, this proof of [27] follows the following steps:

• The definition of an interpretation of the types of Type:

```
\begin{split} &- \left[\!\left[\alpha\right]\!\right] = \mathcal{P}, \\ &- \left[\!\left[\sigma \to \tau\right]\!\right] = \left\{M \in \mathcal{P} \mid \forall N \in \left[\!\left[\sigma\right]\!\right], MN \in \left[\!\left[\tau\right]\!\right]\right\}, \\ &- \left[\!\left[\sigma \cap \tau\right]\!\right] = \left[\!\left[\sigma\right]\!\right] \cap \left[\!\left[\tau\right]\!\right], \\ &- \left[\!\left[\Omega\right]\!\right] = \Lambda \end{split}
```

- The definition of closure properties:
 - $VAR(\mathcal{P}, \mathcal{X})$: $\forall x \in \mathcal{V}, \forall n \geq 0, \forall M_1, \dots, M_n \in \mathcal{P}, xM_1 \dots M_n \in \mathcal{X}$.
 - $\mathsf{SAT}(\mathcal{P}, \mathcal{X})$: $\forall M, N \in \Lambda, \forall n \geq 0, \forall M_1, \dots, M_n \in \mathcal{P},$ $M[x := N]M_1 \dots M_n \in \mathcal{X} \Rightarrow (\lambda x.M)NM_1 \dots M_n \in \mathcal{X}.$
 - $\mathsf{CLO}(\mathcal{P}, \mathcal{X}): \forall M \in \mathcal{X}, \lambda x. M \in \mathcal{P}.$
 - $\mathsf{INV}(\mathcal{P}) \colon \forall M \in \Lambda, M \in \mathcal{P} \iff \lambda x. M \in \mathcal{P}.$
- A proof of: $\forall \sigma \in \mathsf{Type}, \sigma \not\sim^{\nabla} \Omega \Rightarrow \llbracket \sigma \rrbracket \subseteq \mathcal{P}.$
- A proof of a soundness result: $(\forall \sigma \in \mathsf{Type}, \mathsf{VAR}(\mathcal{P}, \llbracket \sigma \rrbracket) \land \mathsf{SAT}(\mathcal{P}, \llbracket \sigma \rrbracket) \land \mathsf{CLO}(\mathcal{P}, \llbracket \sigma \rrbracket)) \Rightarrow (\forall \sigma \in \mathsf{Type}, \Gamma \vdash M : \sigma \Rightarrow M \in \llbracket \sigma \rrbracket).$ Hence, $(\mathsf{VAR}(\mathcal{P}, \llbracket \sigma \rrbracket) \land \mathsf{SAT}(\mathcal{P}, \llbracket \sigma \rrbracket) \land \mathsf{CLO}(\mathcal{P}, \llbracket \sigma \rrbracket)) \Rightarrow (\forall \sigma \in \mathsf{Type}, \Gamma \vdash M : \sigma \land \sigma \not\sim^{\nabla} \Omega \Rightarrow M \in \mathcal{P}).$
- A generalization of hypotheses: $(\mathsf{VAR}(\mathcal{P},\mathcal{P}) \land \mathsf{SAT}(\mathcal{P},\mathcal{P}) \land \mathsf{CLO}(\mathcal{P},\mathcal{P})) \Rightarrow (\forall \sigma \in \mathsf{Type}, \Gamma \vdash M : \sigma \land \sigma \not\sim^{\nabla} \Omega \land (\forall \tau \in \mathsf{Type}, \tau \not\sim^{\nabla} \Omega \Rightarrow \sigma \not\sim^{\nabla} \Omega \to \tau) \Rightarrow M \in \mathcal{P}).$
- A proof of the proof method: $VAR(\mathcal{P}, \mathcal{P}) \wedge SAT(\mathcal{P}, \mathcal{P}) \wedge INV(\mathcal{P}) \Rightarrow \mathcal{P} = \Lambda$.
- A proof of one of the principal results: VAR(CR, CR), SAT(CR, CR) and INV(CR) are true. Hence, $\Lambda = CR$.

This last result is already a well know result, but it does not make the previous ones true, nor does it make the method valid. In [47], we prove that the proof method given in [27] and outlined above, turns out to be false, using a simple counter example based on the set WN. Namely, we show that all of VAR(WN, WN), SAT(WN, WN) and INV(WN) are satisfied, but $\Lambda \neq WN$.

Moreover, each term belonging to the λ -calculus is typable by Ω in $\lambda \cap^{\Omega}$, so each λ -abstraction of each term of the λ -calculus is typable by $\sigma \to \Omega$, whatever σ . By the result generalizing the soundness result, we should obtain that each λ -abstraction of each term of the λ -calculus belongs to \mathcal{P} if $\mathsf{VAR}(\mathcal{P},\mathcal{P})$, $\mathsf{SAT}(\mathcal{P},\mathcal{P})$ and $\mathsf{CLO}(\mathcal{P},\mathcal{P})$ are satisfied. We know that $\mathsf{VAR}(\mathsf{WN},\mathsf{WN})$, $\mathsf{SAT}(\mathsf{WN},\mathsf{WN})$ and

INV(WN) are satisfied, but $\lambda y.\Delta\Delta \notin WN$ (where $\Delta = \lambda x.xx$). In fact, the mistake in the work of [27, 28] comes from this generalization.

By identifying this mistake we could move on to give a partial solution to repair the problem of [27, 28]. So, in [47] we restricted the goals desired by [27, 28] and gave proofs for these new goals. Of course it remains to find out whether the original aims of [27, 28] can be reached. In particular, is it possible to find a set of sufficient conditions such that a set of λ -terms satisfying a property has to satisfy in order to be equal to the set of λ -terms of the λ -calculus? We leave this as a goal to be investigated in the rest of the Ph.D.

The three closure properties defined above are properties needed to prove the soundness result, but we can also remark the similarities with the rules of construction of the λ -calculus. The λ -calculus can be defined as the empty set closed by the following rules:

- VAR(Λ): if $x \in \mathcal{V}$ then $x \in \Lambda$,
- ABS(Λ): if $M \in \Lambda$ and $x \in \mathcal{V}$ then $\lambda x.M \in \Lambda$,
- APP(Λ): if $M, N \in \Lambda$ then $MN \in \Lambda$.

We can remark that $ABS(\mathcal{P})$ is equivalent to $CLO(\mathcal{P}, \mathcal{P})$ and that if \mathcal{P} is closed by $VAR(\mathcal{P}, \mathcal{P})$ then it is closed by $VAR(\mathcal{P})$. So, $\mathcal{P} \subseteq \Lambda$, closed by $VAR(\mathcal{P}, \mathcal{P})$ and $CLO(\mathcal{P}, \mathcal{P})$, is equal to Λ if it is closed by $APP(\mathcal{P})$.

An application term is of one of the two following forms: $(\lambda x.M_0)M_1...M_n$ or $xM_1...M_n$, where $n \geq 1$. So we can replace the closure condition APP(Λ) by the following ones:

- APP-var(Λ): if $x \in \mathcal{V}$, $n \geq 1$ and $M_1, \ldots, M_n \in \Lambda$ then $xM_1 \ldots M_n \in \Lambda$.
- APP-abs(Λ): if $x \in \mathcal{V}$, $n \geq 1$ and $M_0, \ldots, M_n \in \Lambda$ then $(\lambda x. M_0) M_1 \ldots M_n \in \Lambda$.

We can remark that $\mathsf{APP\text{-}var}(\mathcal{P})$ is equivalent to $\mathsf{VAR}(\mathcal{P},\mathcal{P})$. So, $\mathcal{P} \subseteq \Lambda$, closed by $\mathsf{VAR}(\mathcal{P},\mathcal{P})$ and $\mathsf{CLO}(\mathcal{P},\mathcal{P})$, is equal to Λ if it is closed by $\mathsf{APP\text{-}abs}(\mathcal{P})$.

As we noticed in section 6, a similar work has been done by Gallier in [25]. For a comparison between [25] and [27] see the conclusion of [47].

How we extended the reducibility method of [53]

Since the work we have done in relation with [53] is still in progress, we are not able to give some precise references to definitions or lemmas in [47].

The principal result given in [53] is the equality between the sets Λ and CR. The method used to prove this result is a combination of the methods used in [54, 6] (reducibility and developments). As far as we know, the only new result is the proof of the confluence of developments (proof of theorem 3.12 in [53]). In [53], the results and proofs were rather informal for the management of the occurrences of redexes, so we decided to formalize them in another context. We first proved the same result restricted to the λI -calculus using the same method. Then, we extended it in order to deal with η -reduction. To reach this result, we gave a new definition of $\beta \eta$ -development (development dealing with both β - and η -reduction). The difference with the definition given in [6] is that we consider that in $(\lambda x.Mx)N \to_{\eta} MN$ where $x \notin FV(M)$, MN is a residual of the β -redex Mx if Mx is a β -redex, since $(\lambda x.Mx)N \to_{\beta} MN$. Moreover, in $\lambda x.(\lambda y.M)x \to_{\beta} \lambda x.M[y := x]$ where $x \notin FV(\lambda y.M)$ we consider that $\lambda x.M[y := x] = \lambda y.M$ is a residual of the η -redex $\lambda x.(\lambda y.M)x$, since $\lambda x.(\lambda y.M)x \to_{\eta} \lambda y.M$. Our work on extending the results of [53] has a number of advantages:

- We do not leave definitions and notions informal (as is done in [53]). This formalisation work is extensive, and is a necessary step to be able to adapt the method and to generalise it.
- We show that the method is applicable to other reductions rather than simply β -reduction.
- Our work can help to find steps that generalise proofs of properties of the λ -calculus so that the same work does not need to be repeated each time the calculus is extended.

8 A reflection on what to come

Based on the work done during the first year of this Ph.D., and on the goals discussed above, research steps that will be taken during the remaining Ph.D. time consist of the following:

- Giving a semantics of expansion (September 2007 September 2008). We started from a difficult problem which has not yet been approached. Nevertheless, two initial stages have already been solved in [45, 46], so we will now investigate the semantics for a full system with expansion. Although realisability semantics is a pleasant and natural semantics to work with, there exist other semantics which may be investigated. Carlier suggests in [10] that a the denotational semantics (see [61]) should be built for the system E which is a type system integrating expansion. Since expansion involves operations and computational steps transforming types, we may also think about an operational semantics (see [57]). In the framework of realisability semantics, \mathcal{I} being a function from \mathcal{A} to 2^{Λ} and $\llbracket - \rrbracket_{\mathcal{I}}$ an interpretation function from Type^E to 2^{Λ} , if $\sigma, \tau \in \mathsf{Type}^E$ and $\sigma = \tau$ is an equality of figure 3, we certainly would want to have $\llbracket \sigma \rrbracket_{\mathcal{I}} = \llbracket \tau \rrbracket_{\mathcal{I}}$. Even if these methods do not lead us to find a complete semantics for a type system with the full expansion mechanism, we think that we may find some interesting results. Moreover, with expansion being such a crucial operation in computation, its different kinds of semantics need to be given and studied in detail. This thesis aims to give the semantics of expansion.
- Finding a general characterization of properties of the λ-calculus using the reducibility method (July 2007 September 2007 and January 2008 January 2009). The work done by Gallier [22, 23, 24, 25] in that way is already a big step. However, our aim is to concentrate on more particular properties related to particular λ-calculi and type systems and for this, Gallier's results would need to be tamed and/or extended/restricted in order to cope with these varieties of systems. One particular application also involves calculi of expansion. In [10], proofs of properties such as Church-Rosser, Subject Reduction and Strong Normalisation are pretty involved and needed to be seriously revised every time the calculus of expansion was revised. An important goal of our thesis would be how far can the reducibility method generalisation results be applied to involved calculi such as those of expansion. Giving a partial answer would still be very helpful for the generations of language developers who need to constantly revise their calculi to suit the computation needs.
- It may be fruitful to study the generalization of the reducibility method in the λ -cube of Barendregt or in the general framework of intersection type systems dealing with a preorder relation (allowing the introduction of a range of powers depending on the set of axioms defining this relation). The reducibility

method is based on a soundness property and characterization of complete intersection type systems has already been studied in [19, 20]. A generalization of the reducibility method in this framework to more practical type systems and term rewriting systems would be both interesting and useful. Moreover, an investigation of the method used in [26] might lead to a simplification of the method used in [53] and in [47].

- It may be asked why is it that every time a calculus is introduced, proving its properties (especially Confluence, Subject Reduction and Strong Normalisation) involves many revisions and extensions of existing proofs, or even discovering new methods of proofs from scratch. One answer is that these calculi represent computation and so need to be complex and hence proving results about them is also complex. On the other hand, experience shows that sometimes, a calculus can be written to reflect at the same time both the complex computation and the satisfiability of the needed properties. Our goal here is to look at reformulations of known calculi so that their desired properties hold trivially. One particular proposal that will be investigated here is that PTSs⁵ can be generalised in such a way that properties like Strong Normalisation are built in. This is a goal that we will work on (September 2007 July 2009).
- There are other goals that may have to be studied along the way, these include extensions of PTS with:
 - Unified binders [40, 32],
 - Π-application and abbreviations [43, 41] and
 - type inclusion.

9 Conclusion

As is always the case in any Ph.D., the first year is a big learning experience where the student delves into the numerous domains, understands definitions, unpacks problems, attempts to find solutions and realises how big the domain is. This report shows that the author has during his first year, deepened his knowledge, delved into the literature, identified a number of problems in current research and proposed some solutions. The author has also generalised and improved some existing research and identified possible research goals for the next two years. The two immediate research goals are as follows:

• A complete realisability semantics for calculi of expansion. Finding a complete realisability semantics for a type system enables to study the set of legal proofs w.r.t. the type system (see section 4.1). The completeness of a realisability semantics for a type system says something like: the interpretation of a type is inhabited if and only if the type can be assigned, in the type system, to the proofs which inhabit it. For example, given a type system S, and a realisability semantics which interprets the type $\sigma \to \tau$ (say $\llbracket \sigma \to \tau \rrbracket$) by the set $\{M \mid \forall N \in \llbracket \sigma \rrbracket, MN \in \llbracket \tau \rrbracket \}$, if this semantics turns out to be complete, then it gives a characterization of the proofs of such a type in S.

More generally, finding a complete semantics for a type system enables a characterization of legal types, and so verify the intended behavior of the type system. Moreover, by the Curry-Howard isomorphism it enables the study of the logical content of a type system.

 $^{^5 \}mathrm{Pure}$ Type Systems [5], which are a generalization of the $\lambda\text{-}\mathrm{cube}$ of Barendregt

During this year, a step has been done in the way of finding a semantics of expansion. This has opened the door to many possibilities which will be taken into consideration in order to reach the goal of finding a complete semantics for a type system with expansion. In that way, a good comprehension of diverse semantics will be necessary. And finally, such semantics will enable to improve our understanding of expansion.

• Generalisations of methods for proving important properties of type systems. Many type systems have been defined in the history of type theory. It has been observed that the proofs of some properties related to these type systems may share a common structure. The study of some generalizations of such proofs enables to highlight the different aspects of such properties as well as to find general frameworks to study such properties. During the first year, we have studied the reducibility methods for generalising such methods and we have identified that one of these methods is false. We have proposed a partial solution. Moreover, we generalised another method to deal with two different reduction relations. This work sets the ground for our goal of finding general methods for proving important properties of typed λ -calculi.

References

- [1] Fabio Alessi, Franco Barbanera, and Mariangiola Dezani-Ciancaglini. Intersection types and lambda models. *Theor. Comput. Sci.*, 355(2):108–126, 2006.
- [2] Sergei N. Artemov. Explicit provability and constructive semantics. *The Bulletin of Symbolic Logic.*, 7(1), 2001.
- [3] Baader and Nipkow. Term Rewriting and All That. Springer, 1998.
- [4] Henk P. Barendregt. The Lambda Calculus: Its Syntax and Semantics. North Holland, 1981.
- [5] Henk P. Barendregt. Lambda calculi with types. In Handbook of Logic in Computer Science, Volumes 1 (Background: Mathematical Structures) and 2 (Background: Computational Structures), Abramsky & Gabbay & Maibaum (Eds.), Clarendon, volume 2. 1992.
- [6] Henk P. Barendregt, Jan A. Bergstra, Jan Willem Klop, and Henri Volken. Degrees, reductions and representability in the lambda calculus. 1976.
- [7] Henk P. Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completness of type assignment. The Journal of Symbolic Logic., 48(4), 1983.
- [8] Marc Bezem, Jan Willem Klop, Roel de Vrijer, Erik Barendsen, Inge Bethke, Jan Heering, Richard Kennaway, Paul Klint, Vincent van Oostrom, Femke van Raamsdonk, Fer-Jan de Vries, and Hans Zantema. Term Rewriting Systems. Cambridge University Press, 2003.
- [9] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. ACM Computing Surveys, 17(4):471–522, 1985.
- [10] Sébastien Carlier. Expansion Algebra: a Foundational Theory with Applications to Type Systems and Type-Based Program Analysis. PhD thesis, Heriot Watt University, School of Mathematical and Computing Sciences, 2007.

- [11] Sébastien Carlier and Joe B. Wells. Expansion: the crucial mechanism for type inference with intersection types: A survey and explanation. *Electr. Notes Theor. Comput. Sci.*, 136:173–202, 2005.
- [12] Alonzo Church. A set of postulates for the foundation of logic. The Annals of Mathematics, 33(2):346–366, 1932.
- [13] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.
- [14] Alonzo Church and John B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, 1936.
- [15] Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ -terms. Archive for Mathematical Logic, 19(1):139–156, 1978.
- [16] Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ -calculus. Notre Dame Journal of Formal Logic., 21(4), 1979.
- [17] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Principal type schemes and λ -calculus semantic. 1980.
- [18] Haskell B. Curry and Robert Feys. Combinatory logic, vol. 1. 1958.
- [19] Mariangiola Dezani-Ciancaglini, Furio Honsell, and Fabio Alessi. A complete characterization of complete intersection-type theories. CoRR, cs.LO/0011039, 2000.
- [20] Mariangiola Dezani-Ciancaglini, Furio Honsell, and Fabio Alessi. A complete characterization of complete intersection-type preorders. *ACM Trans. Comput. Log.*, 4(1):120–147, 2003.
- [21] Samir Farkh and Karim Nour. Résultats de complétude pour des classes de types du système f2. ITA, 31(6):513–537, 1997.
- [22] Jean H. Gallier. On the correspondence between proofs and λ-terms. 1997. Available at http://www.cis.upenn.edu/~jean/gbooks/logic.html (last visited 2007-05-15).
- [23] Jean H. Gallier. On girard's "candidats de reductibilité". 2002. Available at http://www.cis.upenn.edu/~jean/gbooks/logic.html (last visited 2007-05-15).
- [24] Jean H. Gallier. Proving properties of typed λ-terms using realisability, covers, and sheaves. 2003. Available at http://www.cis.upenn.edu/~jean/gbooks/ logic.html (last visited 2007-05-15).
- [25] Jean H. Gallier. Typing untyped λ-terms, or realisability strikes again!. 2003. Available at http://www.cis.upenn.edu/~jean/gbooks/logic.html (last visited 2007-05-15).
- [26] Silvia Ghilezan and Viktor Kuncak. Confluence of untyped lambda calculus via simple types. Lecture Notes in Computer Science, 2202:38–??, 2001.
- [27] Silvia Ghilezan and Silvia Likavec. Reducibility: A ubiquitous method in lambda calculus with intersection types. *Electr. Notes Theor. Comput. Sci.*, 70(1), 2002.

- [28] Silvia Ghilezan and Silvia Likavec. Extensions of the reducibility method. In 4th Panhellenic Logic Symposium (PLS4), pages 107–112, 2004.
- [29] Jean-Yves Girard. Une extension de l'interpretation de godel a l'analyse, et son application a l'elimination des coupures dans l'analyse et la theorie des types. 1971.
- [30] Jean-Yves Girard. Interpretation Fonctionnelle et Elimination des Coupures de l'Arithmetique d'Ordre Superieur. PhD thesis, Universite de Paris VII, 1972.
- [31] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. 1989. Available at http://www.cs.man.ac.uk/~pt/stable/Proofs+Types. html (last visited 2007-05-15).
- [32] Ferruccio Guidi. Lambda types on the lambda calculus with abbreviations. ArXiv Computer Science e-prints, november 2006.
- [33] Arend Heyting. Intuitionism an Introduction. 1956.
- [34] Roger Hindley. The principal type schemes of an object in combinatory logic. Transaction of the American Mathematical Society., 146:26–60, 1969.
- [35] Roger Hindley. The equivalence of complete reductions. Transaction of the American Mathematical Society., 229:227–248, 1977.
- [36] Roger Hindley. Reductions of residuals are finite. Transaction of the American Mathematical Society., 240:345–361, 1978.
- [37] Roger Hindley. The simple semantics for coppe-dezani-sallé types. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 212–226. Springer, 1982.
- [38] Roger Hindley. The completeness theorem for typing lambda-terms. *Theor. Comput. Sci.*, 22:1–17, 1983.
- [39] Roger Hindley and Giuseppe. Longo. Lambda-calculus models and extensionality. Zeit. Math. Logik, 26:289–310, 1980.
- [40] Fairouz Kamareddine. Typed lambda-calculi with one binder. J. Funct. Program., 15(5):771–796, 2005.
- [41] Fairouz Kamareddine, Roel Bloo, and Rob Nederpelt. On pi-conversion in the lambda-cube and the combination with abbreviations. *Ann. Pure Appl. Logic*, 97(1–3):27–45, 1999.
- [42] Fairouz Kamareddine, Twan Laan, and Rob Nederpelt. A modern Perspective on Type Theory. From its Origins until Today., volume 29. Applied Logic Series, 2004.
- [43] Fairouz Kamareddine and Rob Nederpelt. Canonical typing and pi-conversion in the barendregt cube. *J. Funct. Program.*, 6(2):245–267, 1996.
- [44] Fairouz Kamareddine and Karim Nour. A completeness result for a realisability semantics for an intersection type system. *Annals of Pure and Applied Logic*, 146:5180–198, 2007.
- [45] Fairouz Kamareddine, Karim Nour, Vincent Rahli, and Joe B. Wells. Developing realisability semantics for intersection types and expansion variables. 2006.

- [46] Fairouz Kamareddine, Karim Nour, Vincent Rahli, and Joe B. Wells. Realisability semantics for an intersection typing system with expansion variables. 2007.
- [47] Fairouz Kamareddine and Vincent Rahli. Reducibility proofs in the λ -calculus. 2007.
- [48] Assaf J. Kfoury and Joe B. Wells. Principality and decidability type inference for finite-rank intersection types. pages 161–174, 1999.
- [49] Stephen C. Kleene. On the interpretation of intuitionistic number theory. *The Journal of Symbolic Logic*, 10(4):109–124, 1945.
- [50] Stephen C. Kleene and John B. Rosser. The inconsistency of certain foraml logics. *The Annals of Mathematics*, 36(3):630–636, 1935.
- [51] Jan W. Klop. Combinatory Reductions Systems. PhD thesis, Mathematisch Centrum, Amsterdam, 1980.
- [52] George Koletsos. Church-rosser theorem for typed functional systems. *Journal of Symbolic Logic*, 50(3):782–790, 1985.
- [53] George Koletsos and G. Stravinos. Church-rosser property and intersection types. 2007.
- [54] Jean-Louis Krivine. Lambda-calcul, types et modeles. 1990.
- [55] Jean-Louis Krivine. Typed lambda-calculus in classical zermelo-fraenkel set theory. Archive of Mathematical Logic, 40(3):189–205, 2001.
- [56] James Lipton. Realizability, set theory and term extraction.
- [57] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.
- [58] Jakob Rehof. Strong normalization for non-structural subtyping via saturated sets. *Inf. Process. Lett.*, 58(4):157–162, 1996.
- [59] John B. Rosser. Highlights of the history of the lambda-calculus. In *LFP '82: Proceedings of the 1982 ACM symposium on LISP and functional programming*, pages 216–225, New York, NY, USA, 1982. ACM Press.
- [60] Bertrand Russel. Mathematical logic as based on the theory of types. American Journal of Mathematics, 30(3):222–262, 1908.
- [61] Joseph E. Stoy. Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory. MIT Press, Cambridge, MA, USA, 1981.
- [62] William W. Tait. Intensional interpretations of functionals of finite type i. J. Symb. Log., 32(2):198–212, 1967.
- [63] Anne S. Troelstra. History of constructivism in the 20th century. Available at http://staff.science.uva.nl/~anne/ (last visited 2007-05-15).
- [64] Anne S. Troelstra and D. van Dalen. Constructivism in Mathematics. 1988.
- [65] Pawel Urzyczyn. Type reconstruction in f_{omega}. Mathematical Structures in Computer Science, 7(4):329–358, 1997.

- [66] Jaap van Oosten. Realizability: A historical essay. 2000. Available at citeseer.ist.psu.edu/vanoosten00realizability.html" (last visited 2007-05-15).
- [67] Joe B. Wells. Typability and type checking in system f are equivalent and undecidable. *Ann. Pure Appl. Logic*, 98(1-3):111–156, 1999.
- [68] Joe B. Wells. The essence of principal typings. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 913–925. Springer, 2002.