An Analysis of Girard's Paradox

Thierry Coquand

CMU and INRIA

Introduction

The purpose of this article is to present applications of the Burali-Forti paradox to some formal systems. The first such application is due to J.Y. Girard, who showed [13] that the original system of Martin-Löf [17] was inconsistent, considering an extension of Church's type system with second-order types.

There are two parts to this article. The first part is a study of extensions of Church's higher-order logic. All these extensions are presented in an uniform way. A logical system will always have two components: its functional part, that is almost always a type λ-calculus, and its purely logical part, that is a natural deduction system built on top of this functional system. All the systems that we study have this particularity that logical types are types of the functional system. That is the essence of the impredicativity, and we show that, if this impredicativity is harmless for a lot of calculus, it has to be handled with precaution.

One extension introduces second order types. We explain why this calculus is inconsistent, and apply this result to the type system of Martin-Löf [17] (all the ideas are in Girard's thesis[13], but the basic construction is sligthly different). Another natural extension is the introduction of ML polymorphism[21]. We show is still inconsistent (this was known in essence to Russell [28]). We note that the same paradox applies when one tries to extend Church's system with the notion of a category of all categories. However, it is possible to generalize Church's system with a (weak) notion of polymorphism in a consistent way, and we present the corresponding type system.

The second part is almost a reformulation of the first in a natural deduction framework, where we consider explicitely the structure of proofs. A derivation corresponds then roughly to a λ-term. If normalisation entails consistency, it is easy to see that, conversely, the existence of a paradox entails the existence of non-normalisable λ-terms. This λ-term represents here the explicit writing of the Burali-Forti paradox in natural deduction's style. We give two applications: the inconsistency of the extension of the construction calculus[9] with four levels, and the inconsistency of the extension of the construction calculus with a strong notion of sums *à la Martin-Löf*. We then explain why these results appear as a first step in the analysis of the Curry-Howard analogy between propositions and types. We developp a general argument why a paradox entails in general the undecidability of type-checking, by using an idea due to A. Meyer and M. Reinholt.

An important point is that some constructions of λ-terms have been checked on a computer (a complete formal hand-checking seems quite impossible). We hope that such mechanisation of metamathematical results will help us for a better understanding of the (purely syntactical) phenomena that appear both in mathematical foundations and in the design of type systems for programming languages.

1 Intuitionistic Church's Higher-Order Logic

We assume known the basic notions of λ-calculus, such as β-reduction. The terms have the inductive structure:

1. constants: $Prop$, $Type$,$\Rightarrow$,$\rightarrow$
2. identifiers
3. abstraction $\lambda x.M$, where M is a term, and x an identifier,
4. application $(M\ N)$ where M and N are terms

The notion of reduction is the usual notion of β-reduction[2]. This relation is noted "red". We write "conv" for the smallest congruence on terms (for abstraction and application) which contains the reduction. When adding other operators to the pure λ-calculus, we shall always indicate the new rules of reduction and conversion for these new operators.

1.1 Simple type calculus

Types are generated by the following inductive rules

$$\frac{}{Prop : Type}$$

$$\frac{A : Type \quad B : Type}{A \to B : Type}$$

Definition. A type environment for the calculus of Church is a list $(x_1{:}A_1,...,x_n{:}A_n)$ of pairs of identifiers and types, the identifiers being pairwise distincts.

We shall denote by $\gamma, x : A$ the concatenation of the environment γ and the pair (x, A) (if x is distinct from all x_i), so that $(x_1 : A_1, ..., x_n : A_n)$ is also written $x_1 : A_1, ..., x_n : A_n$.

1.2 Terms

Definition. The typing relation is the smallest relation $\vdash$ between type environment and pairs of identifiers and types such that

$$\frac{x_1 : A_1, ..., x_n : A_n \text{ environment} \quad 1 \leq i \leq n}{x_1 : A_n, ..., x_n : A_n \vdash x_i : A_i}$$

$$\frac{\gamma \text{ environment}}{\gamma \vdash \Rightarrow : Prop \to (Prop \to Prop)}$$

$$\frac{\gamma \vdash t : A \to Prop}{\gamma \vdash \forall(t) : Prop}$$

$$\frac{\gamma \vdash t : A \to B \quad \gamma \vdash u : A}{\gamma \vdash (t\ u) : B}$$

$$\frac{\gamma, x : A \vdash t : B}{\gamma \vdash \lambda x.t : A \to B}$$

If $\gamma \vdash t : A$, one says that the term t is of type A in the environment γ. For example, $\forall(\lambda x.x)$ is of type $Prop$ in the empty environment, and $\lambda x.x$ is of type $Prop \to Prop$ (note that in the present presentation, as in Martin-Löf's[19], we do not have unicity of type).

1.3 The logic

In order to have a logical calculus, one must first define a notion of provable formulae. The comparaison with the rule of HOL [14] may be usefule here.

Definition. A γ-formula is a term of type $Prop$ in the environment γ, the class of provable γ-formulae is defined by the following inductive rules

$$\frac{\gamma \vdash t \text{ is provable} \quad t \text{ conv } u}{\gamma \vdash u \text{ is provable}}$$

$$\frac{\gamma \vdash t : Prop \quad \gamma \vdash u : Prop}{\gamma \vdash t \Rightarrow (u \Rightarrow t) \;\text{ is provable}}$$

$$\frac{\gamma \vdash t : Prop \quad \gamma \vdash u : Prop \quad \gamma \vdash v : Prop}{\gamma \vdash (t \Rightarrow (u \Rightarrow v)) \Rightarrow ((t \Rightarrow u) \Rightarrow (t \Rightarrow v)) \;\text{ is provable}}$$

$$\frac{\gamma, x : A \vdash \forall(\lambda x.\varphi) \text{ is provable} \quad \gamma \vdash t : A}{\gamma \vdash [t/x]\varphi \text{ is provable}}$$

$$\frac{\gamma, x : A \vdash \psi \Rightarrow \varphi \text{ is provable} \quad x \text{ does not occur in } \psi}{\gamma \vdash \psi \Rightarrow \forall(\lambda x.\varphi) \;\text{ is provable}}$$

The only rule of inference is the modus-ponens:

$$\frac{\gamma \vdash \varphi\psi \text{ is provable} \quad \gamma \vdash \varphi \text{ is provable}}{\gamma \vdash \psi \text{ is provable}}$$

For instance, one can show that $\varphi \Rightarrow \varphi$ is a provable formula in the environment $\varphi : Prop$. Hence, $\forall(\lambda\varphi.(\varphi \Rightarrow \varphi))$ is a provable formula in the empty environment.

It is known that all other logical connectives are definable in this calculus[11]. We shall in the sequel use the common logical notations for denoting their translation in this system. For instance, if φ and ψ are terms of type $Prop$, then $\psi \wedge \varphi$ denotes $\forall(\lambda\delta.(\varphi \Rightarrow \psi \Rightarrow \delta) \Rightarrow \delta)$, and $(\exists x : A)\varphi$ denotes $\forall(\lambda\delta.\forall(\lambda x.\varphi \Rightarrow \delta) \Rightarrow \delta)$.

We have to note two important points in this presentation. Firstly, the system has no special rules of equality between terms dependant on their mutual type (the only equality here is the "syntactic" notion of β-conversion), i.e. the system is an intensional one, by opposition, for instance, to the presentation of Lambek[15]. Secondly, the logic of this calculus is the intuitionistic logic, by opposition to the original calculus of Church[8], presented in a classical framework.

Actually, these two points are not relevant here, and the discussion we give about Burali-Forti's paradox extends as well in the framework of topos theory (as shown by the traduction of Lambek-Scott[15]), and Church's theory with the axiom of extensionality. If we want classical logic, it is sufficient to add

$$\frac{\gamma \vdash \varphi : Prop}{\gamma \vdash ((\varphi \Rightarrow \bot) \Rightarrow \bot) \Rightarrow \varphi \;\text{ is provable}}$$

that is, the usual double-negation law, where $\perp$ is the term $\forall(\lambda\varphi.\varphi)$ (which stands for the absurd proposition of the intuitionistic logic). Another possibility is to add Pierce's axiom:

$$\frac{\gamma \vdash \varphi : Prop \quad \psi : Prop}{\gamma \vdash (((\varphi \Rightarrow \psi) \Rightarrow \varphi) \varphi) \Rightarrow \varphi \;\text{ is provable}}.$$

All the derivations of Burali-Forti paradox are still valid in this extensional and/or classical framework.

It is easy to prove, using the truth-table (or valuation) method that this calculus is consistent:

Definition. The interpretation $\mathcal{I}(Prop)$ is $\{0,1\}$, and $\mathcal{I}(A \to B)$ is the set of all function from $\mathcal{I}(A)$ to $\mathcal{I}(B)$.

Definition. A *valuation* v of an environment $x_1 : A_1, ..., x_n : A_n$ is a function defined on $\{x_1, ..., x_n\}$ such that $v(x_i) \in \mathcal{I}(A_i)$ for all i such that $1 \leq i \leq n$. The value $\tilde{v}(t)$ of a term t of type A in the environment $x_1 : A_1, ..., x_n : A_n$ is the element of $\mathcal{I}(A)$ defined by induction on the construction of t:

1. $\tilde{v}(x_i)$ is $v(x_i)$;
2. $\tilde{v}(\lambda x.u)$, where $\lambda x.u$ is of type A, is the function defined from A which associate for $a \in \mathcal{I}(A)$ the value $\tilde{w}(u)$, where w extends v by putting $w(x) = a$;
3. $(t\ u)$ is $\tilde{v}(t)(\tilde{v}(u))$;
4. $\tilde{v}(\Rightarrow)$ and $\tilde{v}(\forall)$ are the usual semantic meaning of $\Rightarrow$ and $\forall$.

Theorem. The previous calculus is consistent, in the sense that there exist formulae which are not provable.

Indeed, the valuation of a provable formula is 1 (by a direct induction on the definition of the notion of truth), and the valuation of $\forall(\lambda\varphi.\varphi)$ is 0. Note that this simple finitary argument cannot be done if we add to the system the infinity axiom (in which case we have to consider not even enumerable sets, but also sets of the cardinality of the continuum).

2 Relations between sets and type theory

2.1 Translation of set-theoretic concepts

Since we are going to derive variants of Burali-Forti paradox in various type system, the first question to address is: how to represent a binary relation in type theory? The answer seems easy: simply, it will be a pair consisting of a type A and a term of type $A \to (A \to Prop)$. There is however some subtlety in this definition. For example, the naive definition of an embedding from a relation (A, R) to a relation (B, S), which says that there exists a term f of type $A \to B$ such that $(\forall x:A)(\forall y:A)(R(x,y) \Rightarrow S(f(x), f(y)))$ and $(\exists b:B)(\forall x:A)S(f(x), b)$ does not work properly when one tries to develop relation theory in a type system. The reason is that in general, the relation R is not "defined" over all the type A, and that the condition $(\exists b:B)(\forall x:A)S(f(x), b)$ is too strong. One needs to consider the *field* of a relation, as follows:

Definition. The *field* of a binary relation R over a type A is the predicate over A $\lambda x.((\exists y : A)R(x,y) \vee (\exists y:A)R(y,x))$. We shall write $D(R)$ for the field of the relation R.

Definition. We shall say that a relation R defined over a type A is *embedded* in a relation S defined over a type B, if, and only if, there exists $f : A\to B$ such that $(\forall x : A)(\forall y : A)D(R)(x)\Rightarrow D(R)(y)\Rightarrow R(x,y)\Rightarrow S(f(x),f(y))$ and $(\exists b : B)(\forall x : A)D(R)(x)\Rightarrow S(f(x),b)$.

The fact that we have been forced to consider not only a term of type A, but also a predicate over A, is actually a particular case of a more general phenomenon, which appears in the translation of type theory into topos theory [15]. An object of the free topos will be not simply a type, but a pair consisting of a type A with a binary relation $E : A\to A\to Prop$ over it, which is transitive and symetric (and must be thought of as a partial equality relation over A). In that case, the field $D(E)$ becomes simply $\lambda x.E(x,x)$ (and it can be thought of as an existence predicate over the type A).

One equality over the type $Prop$ is the logical equivalence. If E is the equality over A and F the equality over B, we can define an equality over $A\to B$ as $\lambda f.\lambda g.(\forall x : A)E(x,x)\Rightarrow F(f(x),g(x))$.

To summarize, we have to relativize mathematical statements with respect to "existence" predicates in type theory. A good example is a statement about a class of predicates. In general, such a statement is only true for extensional classes, and so one must relativize the assertion with the extensionality predicate on the type of predicates of predicates. Here is an instance of this situation: if $C : (A\to Prop)\to Prop$ and C is closed under arbitrary union, i.e. one has, $(\forall D : (A\to Prop)\to Prop)(inclus(D,C)\Rightarrow C(union(D)))$ where $union(D)$ is the predicate $\lambda x.(\exists P : A\to Prop)D(P)\wedge P(x)$ and $inclus$ the relation $\lambda P.\lambda Q.(\forall x : A)(P\ x)\Rightarrow(Q\ x)$, then we want to say that the class C contains the empty predicate $P_0 = \lambda x.\ \bot$. Indeed, if we take $\lambda x.\ \bot$ of type $(A\to Prop)\to Prop$ for the class D (the empty class), then we have $inclus(D,C)$, hence $C(union(D))$. Furthermore, $union(D)$ is extensionally equal to P_0. If we want $C(P_0)$, we need something more, namely that the class C is extensional, i.e.
$(\forall P : A\to Prop)(\forall Q : A\to Prop)((\forall x : A)P(x)\Leftrightarrow Q(x))\Rightarrow(C(P)\Leftrightarrow C(Q))$, for having $C(P_0)$.

The extensionality predicate is a natural example of an "existence" predicate over a type.

An example of "equality" over a type is the intentional equality on a type A. This follows an idea of Leibniz: two elements x and y of type A are *intentionally equal* if the proposition $(\forall P : A\to Prop)P(x)\Rightarrow P(y)$ is provable, i.e. x satisfies every property that satisfies y. It is possible to prove that this relation is reflexive, symmetric and transitive [11].

Once we understand how to code set-theoretical concepts in type theory[15], it is straigthforward to develop relation concepts, such as order, well-founded relation, the embedding relation between two orders. It seems actually that this formulation is in practice more suitable than the usual one in set theory.

2.2 Inadequacy of Church's calculus

The previous calculus, which is roughly the one of the Principia (without so called "typical" ambiguity), is actually not sufficient in practice for the development of real proofs in it (but in a theoretical point of view, it is quite sufficient!). The problem with this system is its lack of uniformity. The following example will explain this.

Suppose we want to define the notion of inclusion between two predicates. In Church's calculus, one has to define relative inclusion to a type: if $A : Type$ then the corresponding notion of inclusion is $\lambda P.\lambda Q.(\forall x : A)P(x)\Rightarrow Q(x)$. But clearly, many theorems about inclusion are completely "polymorphic" in the type A. For instance, the statement that the inclusion relation is transitive, as a relation over $A\to Prop$. The same can be said about the notion of transitivity.

It is thus natural to try to extend Church's calculus by using more general types for capturing this uniformity. This is precisely what the second-order calculus[13, 26] does. So, the motivation for extending simple type calculus, though in a completely distinct context, namely the computer science context for Reynolds [26], was basically the same as the present one.

In the next section, we shall try to mix this notion of second-order types with the logic of Church's calculus, obtaining a formal system which is fundamentally the system U of Girard[13].

3 Extension with second-order types

First, we extend our class of types. We introduce a set $\mathcal{V}$ of type variables and allow quantification on these variables, so that

$$\frac{v \in \mathsf{Prop}}{v : Type}$$

$$\frac{v \in \mathsf{Prop} \quad A : Type}{(\Pi v)A : Type}$$

The symbol Π is a binder, and the same convention as the one for λ holds here. For example, $(\Pi v)(v \to v)$ is the concrete representation of $\Pi\ 1 \to 1$. The intuitive meaning of $(\Pi v)A$ is the product of all types $[B/v]A$, where B varies through the set of all types. There is clearly a kind of circularity here, as $[B/v]A$ may be the type $(\Pi v)B$ as well.

Next, we extend our class of terms.

$$\frac{\gamma \vdash t : A \quad v \text{ does not appear in } \gamma}{\gamma \vdash \lambda v.t : (\Pi v)A}$$

$$\frac{\gamma \vdash t : (\forall v)A \quad B : Type}{\gamma \vdash (t\ B) : [B/v]A}$$

$$\frac{\gamma \vdash t : (\Pi v)Prop}{\gamma \vdash \forall(t) : Prop}$$

Note that we allow non-homogeneous application and abstraction, and we extend our notion of reduction and conversion for these new notions. J.Y. Girard[13] has shown that this calculus has still the normalisation property. A formula of Church's calculus with second order types is a term of type $Prop$ under the previous typing rules.

We also need to extend our class of provable formulae.

$$\frac{\gamma \vdash t \text{ is provable} \quad t \text{ conv } u}{\gamma \vdash u \text{ is provable}}$$

$$\frac{\gamma, v : A \vdash \psi \Rightarrow \varphi \text{ is provable} \quad v \text{ does not occur in } \psi}{\gamma \vdash \psi \Rightarrow \forall(\lambda v.\varphi) \text{ is provable}}$$

$$\frac{\gamma \vdash \forall(t) \text{ is provable} \quad A : Type}{\gamma \vdash (t\ A) \text{ is provable}}$$

This seems to take care of our previous objections to Church's calculus. Now, we can state and prove "generic" statement. For example, we can define the predicate *reflexive* on relation as the polymorphic term $\lambda v.\lambda R.\forall(\lambda x.R(x,x))$ of type $(\Pi v)((v \to (v \to Prop)) \to Prop)$.

What about consistency? We have to note that the result of Girard about normalisation property of the second-order λ-calculus [13] shows that there can be no paradox like Russel's. Indeed, this paradox lays upon the existence of a non-normalisable term of type $Prop$. But, as discovered by J.Y. Girard [13], we have inconsistency in the following sense:

Theorem (Girard's paradox). All formulae of the calculus of Church with second order types are provable.

This shows that this calculus is not valid as a logical calculus (though we have the normalisation property for terms). The next section presents a proof of this theorem.

4 Girard's Paradox

4.1 An abstract presentation of Burali-Forti's paradox

Definition. Let (A, R) and (B, S) be two relations. Then a morphism from (A, R) to (B, S) is a term $f : A \to B$ such that $(\forall x : A)(\forall y : A)R(x, y) \Rightarrow S(f(x), f(y))$ is provable.

The main concept, whose existence entails a paradox in type system, is the following one:

Definition. A *universal system of notations for relations* is a type $A : Type$ together with a term $i : (\Pi v)((v \to v \to Prop) \to A)$, such that if $i(B, R)$ and $i(C, S)$ are intentionally equal, then there exists a morphism from (B, R) to (C, S).

We allow us to use the uncurried notation $i(A, R)$ for $(i\ A\ R)$.

Once we have such a system, say $A_0 : Type$ and $i_0 : (\Pi v)((v \to v \to Prop) \to A_0)$, all we have to do is to follow the Burali-Forti paradox. We first define the embedding relation

$$\mathrm{EMB} : (\Pi A)(A \to A \to Prop) \to (\Pi B)(B \to B \to Prop) \to Prop$$

as above, and the predicate of well-foundeness

$$\mathrm{WF} : (\Pi A)(A \to A \to Prop) \to Prop$$

in the usual way. We then define a predicate wf_0 over A_0:

$$wf_0 = \lambda x.(\exists B)(\exists S : B \to B \to Prop)(x = i_0(B, S) \wedge \mathrm{WF}(B, S)),$$

where $=$ denotes the intensional equality over A. We define in the same way

$$emb : A_0 \to A_0 \to Prop,$$

where $emb(x, y)$ is
$(\exists A)(\exists R : A \to A \to Prop)(\exists B)(\exists S : B \to B \to Prop)(x = i_0(A, R) \wedge y = i_0(B, S) \wedge \mathrm{EMB}(A, R, B, S))$.

Note that we need A_0 to be universal system for proving that emb is a transitive relation. We can then define

$$emb_0 = \lambda x.\lambda y.emb(x, y) \wedge wf_0(x) \wedge wf_0(y).$$

The main properties are that we have $\mathrm{WF}(A_0, emb_0)$, and

$$(\forall A)(\forall R : A \to A \to Prop)\mathrm{WF}(A, R) \Rightarrow \mathrm{EMB}(A, R, A_0, emb_0).$$

But it is straigthforward to show that

$$(\forall A)(\forall R : A \to A \to Prop)\mathrm{WF}(A, R) \Rightarrow \mathrm{EMB}(A, R, A, R) \Rightarrow \perp$$

hence the contradiction, since we have then

$$\mathrm{EMB}(A_0, emb_0, A_0, emb_0)$$

and

$$\mathrm{EMB}(A_0, emb_0, A_0, emb_0) \Rightarrow \perp .$$

This derivation can be thought of as the abstract scheme of Burali-Forti paradox. A type system is inconsistent as soon as it is possible to construct a universal system of notation in it.

4.2 Application to Girard's paradox

We shall now apply this general scheme to the special case of the second-order Church calculus. We have to construct a universal system of notation for orders. One possiblity is

$$A_0 = ((\Pi B : Type)((B \to B \to Prop) \to Prop)) \to Prop.$$

We shall take i_0 to be the term $\lambda B.\lambda R.\lambda x.x(B, R)$ (this seems to be a little simpler than the one of Girard's thesis[13]). Our choice is motivated by the usual way of embedding a type A in its associated type of class $(A \to Prop) \to Prop$. This is a general method of construction of class in the Principia [28]. We can give two other instances of this method. We can view pairs over the types A and B as elements of type $(A \to B \to Prop) \to Prop$. In the same way, if R is an equivalence relation on a type A, then an equivalence class for R is built as an element of type $(A \to Prop) \to Prop$. Here, this method solves our problem of the definition of a universal system of notation for relations.

Lemma. (A_0, i_0) is a universal system of notations.

Suppose indeed that $i_0(B, R)$ and $i_0(C, S)$ are intentionally equal, i.e. that we have a proof of

$$(\forall P : A \to Prop)P(i_0(B, R)) \Rightarrow P(i_0(C, S)).$$

We have then to show that there exists a morphism from (B, R) to (C, S). Let MOR be the relation such that $\mathrm{MOR}(A, R, B, S)$ is

$$(\exists f : A \to B)(\forall x : A)(\forall y : A)R(x, y) \Rightarrow S(f(x), f(y)).$$

We can instantiate the given proof of equality between $i(B, R)$ and $i(C, S)$ on the predicate

$$Q \;=\; \lambda x.x(F),$$

where F is the term

$$\lambda D.\lambda T.\mathrm{MOR}(B, R, D, T).$$

Then, $Q(i_0(D,T))$ is convertible to $\mathrm{MOR}(B,R,D,T)$, and as MOR is a reflexive relation, we obtain in this way a proof of $\mathrm{MOR}(B,R,C,S)$. This proves that (A_0, i_0) is a universal notation system for relations.

Another possiblity[13], is to define A_0 as

$$(\Pi v)((\Pi w)((w \to w \to Prop) \to v) \to v).$$

This is obtained by copying at the level of types the translation of the existential quantifier in intuitionistic higher-order logic. We define then i_0 as the λ-term $\lambda B.\lambda R.\lambda v.\lambda x.x(B,R)$.

If we now look back to our rules, we see that the paradox is actually not so surprising: the operation $(\forall v)A$, where v varies through types, should correspond to a set-theoretic product over all sets. But such an operation is not allowed in set-theory.

5 Church's calculus with ML polymorphism

One can think that second-order types are a too strong mean to allow "generic" statement. we shall study now a weaker notion of polymorphism, namely the ML's notion of polymorphic type. As a solution of the "uniformity" problem, Milner [21] proposed the notion of polymorphic constant. I assume known this notion[21].

The example of the definition of the concept of "inclusion" will be sufficient for understanding what's going on. In ML's type calculus, we are able to define a generic constant INCLUS by

$$\lambda P.\lambda Q.(\forall x)P(x) \Rightarrow Q(x),$$

of type

$$(* \to Prop) \to (* \to Prop) \to Prop,$$

where $*$ denotes a type variable. In the same way, we are able to define a generic constant TRANSITIVE as

$$\lambda R.(\forall x)(\forall y)(\forall z)R(x,y) \Rightarrow R(y,z) \Rightarrow R(x,z)$$

of type

$$(* \to * \to Prop) \to Prop.$$

Then the term TRANSITIVE(INCLUS) is a well-typed term which expresses, in a generic way, that the relation of *inclusion* is a *transitive* one. The fundamental remark of Milner[21] was that the unification algorithm of Herbrand-Robinson is well-suited for this kind of type-checking. This system seems attractive for a practical use of type theory.

But if this weak notion is convenient for the programmation of functionals, it appears to be too strong for logic.

Theorem. Church's calculus with ML's polymorphism is inconsistent, i.e. $\perp$ is provable in this calculus.

The reasoning is very close to the one in the case of second-order calculus. Indeed, we can define a generic constant WF, such that $\mathrm{WF}(R)$ says that the relation R is well-founded. We can also define a generic constant EMB such that $\mathrm{EMB}(R,S)$ says that R, S are well-founded relation and that R can be embeded strictly in S (as the previous case of second-order type). Then it is possible to show that WF(EMB) is a provable formula, that

$$(\forall R)\mathrm{WF}(R) \Rightarrow \mathrm{EMB}(R, \mathrm{EMB}),$$

and that

$$(\forall R)\mathrm{WF}(R) \Rightarrow \mathrm{EMB}(R, R) \Rightarrow \perp .$$

We have then a contradiction, since

$$\mathrm{EMB}(\mathrm{EMB}, \mathrm{EMB})$$

and

$$\mathrm{EMB}(\mathrm{EMB}, \mathrm{EMB}) \Rightarrow \perp$$

are both provable.

It is interesting to note that the notion of *stratified* formula of Quine [24] is very close to this notion of generic statement. Indeed, the test for checking that a given formula is stratified [24] is a particular case of the ML type-checking algorithm. See also the derivation of Rosser[25] of the Burali-Forti paradox in an earlier version of the system of Quine.

6 Consistent extensions of Church's calculus

6.1 Predicative polymorphism

It is worth to note that the language HOL of M. Gordon[14], which is a generalisation of Church's calculus by using type variable, does not allow the full polymorphism of ML, so that the previous contradiction does not apply to this calculus.

We shall try to give a consistent formal system which is consistent and allows a (weak) form of polymorphism, which could be thought of as a formalisation of the type system used in HOL. We introduce first another constant $Type_1$ such that $Type$ is of type $Type_1$, and we relativize the rules of second-order typing in a predicative way, where the environments are now a list of pairs $x_1 : A_1, ..., x_n : A_n$ where $A_j : Type_1$ for $1 \leq j \leq n$:

$$\frac{A : Type}{A : Type_1}$$

$$\frac{\gamma \vdash v : Type \quad A : Type_1}{\gamma \vdash (\forall v).A : Type_1}$$

$$\frac{\gamma \vdash t : A \quad v \text{ does not appear in } \gamma}{\gamma \vdash \lambda v.t : (\forall v).A}$$

$$\frac{\gamma \vdash t : (\forall v)A \quad \gamma \vdash B : Type}{\gamma \vdash (t\ B) : [B/v]A}$$

$$\frac{\gamma \vdash t : (\forall v)Prop}{\gamma \vdash \Pi(t) : Prop}$$

The important point is that we no longer consider that the types $(\forall v)A$ are homogeneous to the "usual" types, but belong to $Type_1$.

This system seems to be the right formalisation of the "polymorphism" used in LCF, and to be the one we have to use for the mechanisation of higher-order logic, as it presents enough uniformity to develop in a generic way usual mathematic arguments, but is still consistent. For instance, we can state in a "generic" way the definition of the equivalence relation related to a preordering, and the proof that this is an equivalence relation. we can instantiate this "abstract" situation on a "concrete" given preordering, by example the inclusion, and specialize both the definition and the proof, obtaining thus the concept of extentional equality, and the proof that this is an equivalence relation.

In practice, it is necessary to synthetize types arguments, with a mechanism analogous, but less general, than the type-checker algorithm of ML. This system appears as a formalisation of the type system of the Principia, with the notion of "typical" ambiguity. This synthesis facility is present in LCF and the system HOL of M. Gordon [14]. We have seen that the polymorphism of ML cannot be used in a logical calculus.

6.2 Introduction of a sum operator

We can add a sum operator over types, i.e. we introduce the pairing operation (M_1, M_2) and the two projections π_1 and π_2 (with the new conversion rule that $\pi_j(M_1, M_2)$ is convertible to M_j for $j = 1$ or 2). We introduce also the new binder $(\Sigma x : A)B$ and the rules

$$\frac{\gamma, v : Type \vdash B : Type}{\gamma \vdash (\Sigma v : Type)B : Type_1}$$

$$\frac{\gamma \vdash A : Type \quad \gamma \vdash t : [A/v]B}{\gamma \vdash (A, t) : (\Sigma v : Type)B}$$

$$\frac{\gamma \vdash t : (\Sigma v : Type)B}{\gamma \vdash \pi_1(t) : Type}$$

$$\frac{\gamma \vdash t : (\Sigma v : Type)B}{\gamma \vdash \pi_2(t) : [\pi_1(t)/v]B}$$

We define then the notion of provable formula as for the calculus with second-order types. Note that system is almost the same as the one of MacQueen[16], but here we have furthermore a logical calculus, i.e. a notion of proposition and of provable formula.

The truth-table method used before for Church's calculus show that this calculus is consistent. It is actually possible to extend it more by the addition of a hierarchy of cumulative universes of the predicative system of Martin-Löf[19]. This type system seems the right formalism for describing modules in ML [16].

6.3 Addition of "small" products and sums

It can be useful to extend the previous calculus by the addition of small product $(\Pi x : A)B$ and sums $(\Sigma x : A)B$, with $A : Type$, and the following inductive rules:

$$\frac{\gamma \vdash A : Type \quad \gamma, x : A \vdash B : Type}{\gamma \vdash (\Pi x : A)B : Type}$$

$$\frac{\gamma \vdash A : Type \quad \gamma, x : A \vdash B : Type}{\gamma \vdash (\Sigma x : A)B : Type}$$

$$\frac{F : (\Pi x : A)B \quad \gamma \vdash t : A}{\gamma \vdash (F\ t) : [t/x]B}$$

$$\frac{\gamma \vdash A : Type \quad \gamma, x : A \vdash F : B}{\gamma \vdash \lambda x.F : (\Pi x : A)B}$$

$$\frac{\gamma \vdash A : Type \quad \gamma, x : A \vdash B : Type \quad \gamma \vdash t : A \quad \gamma \vdash u : [t/x]B}{\gamma \vdash (t, u) : (\Sigma x : A)B}$$

$$\frac{\gamma \vdash t : (\Sigma x : A)B}{\gamma \vdash \pi_1(t) : A}$$

$$\frac{\gamma \vdash t : (\Sigma x : A)B}{\gamma \vdash \pi_2(t) : [\pi_1(t)/x]B}$$

$$\frac{\gamma \vdash A : Type \quad \gamma \vdash t : B \quad A \text{ conv } B}{\gamma \vdash t : A}$$

The notion of provable formula is determined by the same rules as for Church's calculus, and a truth-table argument shows that this calculus is consistent.

Actually, the present generalisation of Church's calculus can be done independently of the introduction of products and sums over types. But the combination of the two notions (respectivally "small" and "large") seems powerful enough for the natural expression of a lot of mathematical concepts.

In the system with both "small" and "large" products, we can axiomatize categories as follows. In the context where:

$$A : Type,$$

$$Hom : (\Pi x : A)(\Pi y : A)Type,$$

$$E : (\Pi x : A)(\Pi y : A)Hom(x,y) \rightarrow Hom(x,y) \rightarrow Prop,$$

$$id : (\Pi x : A)Hom(x,x),$$

and

$$o : (\Pi x : A)(\Pi y : A)(\Pi z : A)Hom(x,y) \rightarrow Hom(y,z) \rightarrow Hom(x,z),$$

we define $\mathrm{CAT}(A, Hom, E, id, o)$ as saying that A is the type of objects of a category determined by the congruence $E(x,y)$ on $Hom(x,y)$ if $x : A$ and $y : A$, the identity $id(x)$, and the composition o[11].

The formal typed system presented here seems to be the natural one for an axiomatisation of categories. Note that in this formal system, there is an object which represents the category of "small" categories (built in a straigthforward way as a sum), but this object is of type $Type_1$, and so it is not itself a small category. It is then natural to ask if it is possible to add some special constant to this calculus, so that we capture formally the idea of the category of all categories.

7 CAT is not a CAT

It appears actually that it is possible to apply the ideas used in the typed-checked paradox to derive a contradiction from the fact that there is a category of all categories. We present first our argument in an informal way, showing that our argument is independant of the underlying formalisation (here, higher-order logic with Church's type system).

Definition. Let C be a category, the order $R(C)$ is defined on $obj(C)$ by $R(C)(a,b)$ if, and only if, $C(a,b)$ is non empty and $C(b,a)$ is empty; then we shall say that C is *well-founded* if, and only if, $R(C)$ is a well-founded ordering.

Definition. Let C and D be two categories, then a functor F between C and D is said to be *dominated* if, and only if, there is an object y of D such that, for every object x of C, we have $R(C)(F(x),y)$.

Proposition. The "category" which has for objects all well-founded categories and for morphisms the identities and the dominated functors does not exist.

The reason is that we can reproduce the typed-checked example if such a category exists., as this category, if it exits, must be both well-founded and such that there exists a dominated functor into itself.

All this argument was developed in an informal way, but it is possible to apply it to a precise formal system which extends Church's system by the addition of a category of all categories.

We add to the calculus of the previous section a special type $Cat : Type$, with the constants i, p_j, for $1 \leq j \leq 5$, the rules

$$\frac{\gamma \vdash \mathrm{CAT}(A, Hom, E, id, o)}{\gamma \vdash i(A, Hom, E, id, o) : Cat}$$

$$\frac{\gamma \vdash x : Cat}{\gamma \vdash \mathrm{CAT}(p_1(x), p_2(x), p_3(x), p_4(x), p_5(x))}$$

and the rules of conversion that say that $p_j(x_1, x_2, x_3, x_4, x_5)$ is convertible to x_j for $1 \leq j \leq 5$.

The special type Cat can be seen as the type of objects of the category of all category. Indeed, we can define terms E,Hom,id, and o, such that (Cat, E, Hom, id, o) represents the category of all category with as morphisms the functor between categories. We have then $i(Cat, E, Hom, id, o) : Cat$. The previous reasoning shows that this typed system is not consistent.

For the formalisation of the notion of a category of all categories, it appears thus to be necessary to weaken the logic (but this logic must be strong enough for the expression of usual categorical theorem). An approach with only equational Horn clauses as formulae seems however still possible.

8 The calculus with "Type:Type"

The next application is the inconsistency of the first calculus of Martin-Löf[17]. This can be seen as the generalisation of the calculus of Church (but, in an intuitionistic framework), where we add that there is a type $Type$ of all types.

Definition. The class of term of the calculus with $Type$:$Type$ is the class defined by the following inductive rules

1. *Type* is a term
2. an identifier is a term
3. an integer (de Bruijn index) is a term
4. if M and N are terms, then $(M\ N)$ is a term
5. if M is a term, then $\lambda(M)$ is a term
6. if M and N are terms, then $\Pi(M, N)$ is a term

We extend our notion of conversion between terms so that it is the least relation which contains the β-reduction and is a congruence for the abstraction, the application and the product. This system constructs some typing sequents, i.e. sequents of the form $\Gamma \vdash M : P$, where Γ is a list of typings $x : A$, where x is an identifier and M, P are terms. This relation can be read as "M is a valid term of type P in the type assignment Γ".

8.1 Assignments

$$\text{the empty environment is valid}$$

$$\frac{\Gamma \text{ is valid} \quad \Gamma \vdash M : Type \quad x \text{ does not occur in } \Gamma}{\Gamma, x : M \text{ is valid}}$$

8.2 Type Inference Rule

$$\frac{\Gamma \text{ is valid}}{\Gamma \vdash Type : Type}$$

$$\frac{\Gamma \text{ is valid} \quad x \text{ occurs in } \Gamma \text{ with the type } M}{\Gamma \vdash x : M}$$

$$\frac{\Gamma, x : M \vdash N : P}{\Gamma \vdash \lambda x.N : (\Pi x : M)P}$$

$$\frac{\Gamma, x : M \vdash N : Type}{\Gamma \vdash (\Pi x : M)N : Type}$$

$$\frac{\Gamma \vdash M : (\Pi x : A)R \quad \Gamma \vdash N : A}{\Gamma \vdash (MN) : [N/x]R}$$

$$\frac{\Gamma \vdash M : P \quad \Gamma \vdash Q : Type \quad P \text{ conv } Q}{\Gamma \vdash M : Q}$$

Some comments are necessary, to make the connection between this type system and the previous extension of Church's calculus. Note we do not have $\rightarrow$, $\Rightarrow$ and Π any more, but all these constants are replaced by the same binding $(\Pi x : M)N$, which may be thought of as the product over the type M of the family N. Indeed, if N does not depend on x, we get intuitively the type $M \rightarrow N$. For getting the $\Rightarrow$ and Π of the calculus of Church, the idea is to think of a proposition as the type of its proof. Then we see that $(\Pi x : M)N$ is effectively the type of proofs of the proposition $M \Rightarrow N$ if M and N are propositions, and that $(\Pi x : M)N$ is the type of proofs of the proposition $\Pi((\lambda x.N))$ if M is a type and N is a proof.

The original motivation of this system was the identification of the types and the propositions. Then $Prop$ is identified with $Type$ and if we write down what becomes the calculus of Church with this identification, we get this calculus. Each proposition must be thought of as the type of its proof, but conversely each type also becomes a proposition: namely, the proposition that this type is non empty. We have no longer any need of inductive rules for the notion of provable formula. We define simply a provable formula (or type) as a term M such that $M : Type$ and M, as a type, is non empty. This is a very elegant aspect of this calculus as a formal system.

There is something wrong however, since the inconsistency of Church's system with second order type can be done here. Indeed, as a logical calculus, the present system contains the intuitionistic Church calculus, but also second order quantification over types (as propositions and types are now identified). So, we get that the "proposition" $(\Pi p : Type)p$ is provable, i.e., with the definition of truth, we get a closed term of type $(\Pi p : Type)p$. Such a term cannot be in head normal form, hence it is not normalisable.

So we must give up the identification of propositions and types. A possible solution is the calculus of constructions[9], where we keep only the identification of a proposition with the type of its proof, but we no longer identify every type with a proposition.

There have been proposed some programming or specification languages which contains the idea of a type $Type$ of all types, together with the fact that this type is also of type $Type$[3, 7]. It is indeed possible, by using ideas from Scott and Martin-Löf[29] to build models of the theory with $Type : Type$ (in this notion of "model", the logical side of the calculus does not appear). Though the termination property is not the primary concern of the computer programmer, it seems very important to study the "computationnal" relevance of Girard's Paradox (what seems to be lost forever is the possibility of doing proofs about programs in such systems).

9 The calculus of constructions with four levels

"All attemps to strengthen this system (the system of constructions), in particular to temper with the fourth level, should be very cautious: the Tarpeian Rock is close to the Capitol." (J.Y. Girard)

The general idea now is to write all the previous results with explicit proofs. We are going to apply our results to the study of some extension of the calculus of construction[10], which is a general formalism for a mechanical study of higher-order proofs (it seemed thus natural to check on a computer Girard's paradox in that formalism).

We present rules of typing of the extension of the construction calculus where we allow "context" variables and general polymorphism on them. This corresponds to the extension of Church's calculus with second-order types.

We first extend our class of terms, which are now representing as well proofs and terms.

Definition. The class of term of the calculus of construction is the class defined by the following inductive rules:

1. $Type$ and $Prop$ are terms
2. an identifier is a term

3. an integer (de Bruijn index) is a term
4. if M and N are terms, then $(M\ N)$ is a term
5. if M is a term, then $\lambda(M)$ is a term
6. if M and N are terms, then $\Pi(M, N)$ is a term

Note that we introduce a binary operator for products. We shall not need then any special constant such as $\Rightarrow$ neither special rules of inference. The motivation is that we shall simply express the proposition-as-types principle: we identify a proposition with the type of its proofs, so that the previous quantifier $(\forall x : A)\varphi$ becomes the product $\Pi(A, \varphi)$. Then, $A \Rightarrow B$ is definable as $\Pi(A, B)$. It seems so that for building a type system with an associated logic, all we need is to have the λ-operation and the product formation. All the (semantic) rule about provable formulae of Church's calculus appear as derived rules of a very simple typing mechanism.

We shall generalize the previous rules of the construction calculus[9],[11], by the introduction of *Type*, the type of so-called "contexts"[9], and we shall try to extend this calculus with four levels.

9.1 Assignments

$$\text{the empty assignment is valid}$$

$$\frac{\Gamma \text{ is valid} \quad \Gamma \vdash M : Prop \quad x \text{ does not appear in } \Gamma}{\Gamma, x : M \text{ is valid}}$$

$$\frac{\Gamma \text{ is valid} \quad \Gamma \vdash M : Type \quad x \text{ does not appear in } \Gamma}{\Gamma, x : M \text{ is valid}}$$

$$\frac{\Gamma \text{ is valid} \quad x \text{ does not appear in } \Gamma}{\Gamma, x : Type \text{ is valid}} \qquad (*)$$

9.2 Type Inference Rules

$$\frac{\Gamma \text{ is valid}}{\Gamma \vdash Prop : Type}$$

$$\frac{\Gamma \text{ is valid} \quad x \text{ occurs in } \Gamma \text{ with the type M}}{\Gamma \vdash x : M}$$

$$\frac{\Gamma, x : M \vdash N : P}{\Gamma \vdash \lambda x.N : (\Pi x : M)P}$$

$$\frac{\Gamma, x : M \vdash N : Prop}{\Gamma \vdash (\Pi x : M)N : Prop}$$

$$\frac{\Gamma, x : M \vdash N : Type}{\Gamma \vdash (\Pi x : M)N : Type}$$

$$\frac{\Gamma \vdash M : (\Pi x : A)R \quad \Gamma \vdash N : A}{\Gamma \vdash (MN) : [N/1]R}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash R : Prop \quad A \text{ conv } R}{\Gamma \vdash M : R}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash R : Type \quad A \text{ conv } R}{\Gamma \vdash M : R}$$

If we want the usual calculus of constructions[9], we simply delete the rule of type variable introduction. We can see in this way that this calculus is very close to the one presented in an unpublished paper of de Bruijn[6].

Note the fact that *Type* and *Prop* play a very similar role. We can, as in Automath-like languages, define the degree of a term [5] with $degree(Type) = 0$. We have then

Definition. We shall say that M is a *Type* if, and only if, $degree(M) = 1$, and that M is a *Prop* if, and only if, $\Gamma \vdash M : N$ with N conv *Prop*.

Proposition 1. If $\Gamma \vdash M : N$, then $degree(M) = degree(N) + 1$, and $degree(M)$ is 1 ,2 or 3.

Proposition 2. If $\Gamma \vdash M : N$, and $degree(M) = 3$ (resp. 2) then we have $\Gamma \vdash N : Prop$ (resp. $\Gamma \vdash N : Type$).

This shows intuitively that this systems has four levels: the proofs, the propositions, the types, and the "supertypes", as *Type* of degree 0.

Nearly all we have said for the system with $Type : Type$ could be repeated in the present context. In particular, we can define the arrow by $M \to N = \Pi(M, N)$ if M, N are both Types, and $M \Rightarrow N = \Pi(M, N)$ if M, N are both Props, in the same type assignment. We have here two notions of "arrows". We must view *Type* as the collection of all sets (and the $\to$ is viewed as exponentiation), and *Prop* as the type of propositions (and the $\Rightarrow$ is viewed as implication).

In the same way, we have two sorts of quantifications on a variable of type *Type*. A quantification at the level of types $(\Pi x : Type)N : Type$, if $N : Type$ (with the hypothesis $x : Type$) and a quantification on the level of propositions, $(\Pi x : Type)N : Prop$, if $N : Prop$ (with the hypothesis $x : Type$). We have seen that the core of the paradox lies in this double quantification (the second one must represent a product over all set, while the first represents simply the usual quantification over sets).

It is worth it to compare this presentation of a typed calculus, which contains higher-order logic, with more standard presentations of higher-logic, such as the one of Takeuti[30].

If we want the usual higher-order type system, all we have to do is to restrict the formation of type assignments by forbidding the introduction of type variables. We obtain thus the calculus of constructions [9, 23].

The main addition in relation to the calculus of Church is that we have now a very concise notation for proofs. We shall be able to study manipulations on proofs, especially the cut-elmination (here simply the β-reduction). It is straigthforward to show that this calculus is more general than Church's calculus with second order type in the following sense: let us define a *provable formula* as a term of type *Prop* such that there exists a term N of type M (in the empty environment). Then all provable formula of Church's calculus with second order type are provable (up to a translation) in the present calculus.

We can now apply the previous results, obtaining thus:

Theorem. There exists a term M such that $\vdash M : (\Pi p : Prop)p$, and no term satisfying this condition is normalisable.

Indeed, no λ-term in head-normal form can be of type $(\Pi p : Prop)p$.

If we look at the process of reduction as the process of "understanding" one proof we can say that the "proof" of Girard's paradox becomes more and more complex when one try to understand it!

10 The calculus of constructions with sums

We shall study now two notions of sums: the weak one (or "package", which is actually definable in the pure calculus of construction) versus the strong one (with the two projections) in the framework of the calculus of constructions. This result may be interesting in the analysis of the representation of "abstract data types" in typed system[22],[16].

For a notion of sum, we need first, as previously, the operation of pairing (M, N) and the two projections π_1 and π_2. Finally, one adds the sum formation $\Sigma(A, B)$ (with the same convention as the one for the product).

The rules of derivation for the construction calculus with the strong sum are

$$\text{the empty assignment is valid}$$

$$\frac{\Gamma \text{ is valid} \quad \Gamma \vdash M : Prop \quad x \text{ does not appear in } \Gamma}{\Gamma, x : M \text{ is valid}}$$

$$\frac{\Gamma \text{ is valid} \quad \Gamma \vdash M : Type \quad x \text{ does not appear in } \Gamma}{\Gamma, x : M \text{ is valid}}$$

$$\frac{\Gamma \text{ is valid}}{\Gamma \vdash Prop : Type}$$

$$\frac{\Gamma \text{ is valid} \quad x \text{ occurs in } \Gamma \text{ with the type M}}{\Gamma \vdash x : M}$$

$$\frac{\Gamma, x : M \vdash N : P}{\Gamma \vdash \lambda x.N : (\Pi x : M)P}$$

$$\frac{\Gamma, x : M \vdash N : Prop}{\Gamma \vdash (\Pi x : M)N : Prop}$$

$$\frac{\Gamma, x : M \vdash N : Prop}{\Gamma \vdash (\Sigma x : M)N : Prop}$$

$$\frac{\Gamma, x : M \vdash N : Type}{\Gamma \vdash (\Pi x : M)N : Type}$$

$$\frac{\Gamma \vdash M : (\Pi x : A)R \quad \Gamma \vdash N : A}{\Gamma \vdash (MN) : [N/1]R}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : [t/x]C}{\Gamma \vdash (t, u) : (\Sigma x : A)C}$$

$$\frac{\Gamma \vdash t : (\Sigma x : A)C}{\Gamma \vdash \pi_1(t) : A} \qquad (*)$$

$$\frac{\Gamma \vdash t : (\Sigma x : A)C}{\Gamma \vdash \pi_2(t) : [\pi_1(t)/x]C} \qquad (*)$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash R : Prop \quad A \text{ conv } R}{\Gamma \vdash M : R}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash R : Type \quad A \text{ conv } R}{\Gamma \vdash M : R}$$

For the weak notion of sum, we need to introduce a new binary operator rep with the fact that $rep((M,N),\lambda(P))$ is convertible to (PMN). We replace then the starred rules by the following one:

$$\frac{\Gamma \vdash t : (\Sigma x : A)C \quad \Gamma \vdash D : Prop \quad \Gamma \vdash u : (\Pi x : A)(C \rightarrow D)}{\Gamma \vdash rep(t,u) : D}.$$

Note that the weak notion of sum is definable in the calculus of constructions. This is the usual translation of the existence quantifier in higher-order intuitionistic logic: we define $(\Sigma x : A)B$ as $(\forall C : Prop)((\forall x : A)B \Rightarrow C) \Rightarrow C$. So the calculus of construction with the weak notion of sum is still consistent, and has still the normalisation property.

For the strong notion of sum however, it is possible to build at the level of propositions a universal system of notation for relation: $A_0 = (\Sigma C : Prop)C \rightarrow C \rightarrow Prop$. Hence, our previous considerations show that it is possible to build a term of type $(\Pi p : Prop)p$, and such a term cannot be normalisable (actually, it is a term without head normal form).

This result appears as a metamathematical justification of the idea of "package": if we allow the user to "see" the program, i.e. if we have the two projections for the sum, then the type system becomes inconsistent. This idea of package has actually purely "programming" motivation[22].

11 Consistent extensions of the construction calculus

We can apply the various consistent extensions of Church's calculus described previously to the construction calculus. Here is the formal system actually implemented in ML. The terms are:

1. $Type(i)$, for i integer, and $Prop$ are terms
2. an identifier is a term
3. an integer (de Bruijn index) is a term
4. if M and N are terms, then $(M\ N)$ is a term
5. if M is a term, then $\lambda(M)$ is a term
6. if M and N are terms, then $\Pi(M,N)$ is a term.

11.1 Assignments

$$\text{the empty assignment is valid}$$

$$\frac{\Gamma \text{ is valid} \quad \Gamma \vdash M : Prop \quad x \text{ does not appear in } \Gamma}{\Gamma, x : M \text{ is valid}}$$

$$\frac{\Gamma \text{ is valid} \quad \Gamma \vdash M : Type(i) \quad x \text{ does not appear in } \Gamma}{\Gamma, x : M \text{ is valid}}$$

11.2 Type Inference Rule

$$\frac{\Gamma \text{ is valid}}{\Gamma \vdash Prop : Type(0)}$$

$$\frac{\Gamma \text{ is valid}}{\Gamma \vdash Type(i) : Type(i+1)}$$

$$\frac{\Gamma \vdash M : Type(i)}{\Gamma \vdash M : Type(i+1)}$$

$$\frac{\Gamma \text{ is valid} \quad x \text{ occurs in } \Gamma \text{ with the type M}}{\Gamma \vdash x : M}$$

$$\frac{\Gamma, x : M \vdash N : P}{\Gamma \vdash \lambda x.N : (\Pi x : M)P}$$

$$\frac{\Gamma, x : M \vdash N : Prop}{\Gamma \vdash (\Pi x : M)N : Prop}$$

$$\frac{\Gamma \vdash M : Type(j) \quad \Gamma, x : M \vdash N : Type(i)}{\Gamma \vdash (\Pi x : M)N : Type(max(i,j))}$$

$$\frac{\Gamma \vdash M : (\Pi x : A)R \quad \Gamma \vdash N : A}{\Gamma \vdash (MN) : [N/1]R}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash R : Prop \quad A \text{ conv } R}{\Gamma \vdash M : R}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash R : Type(i) \quad A \text{ conv } R}{\Gamma \vdash M : R}.$$

The author does not know the proof-theoretic strength of such a system (conjecture: it exceeds the one of Zermelo). Since it is possible to prove in this system the normalisation theorem of Girard's system, it is more powerful than higher-order arithmetic. It could be possible also to add sums and inductive types.

But the important point is less the power of this calculus than the fact that, as a formal system, it is the natural (and "predicative") expression of a reflection principle over the calculus of construction, and Girard's paradox show that the "impredicative" extension, what we call here the system with four levels, is inconsistent. This seems actually to show that the predicativity and non-predicativity are not contradictory concepts: simply, the level of proposition may be non-predicative and the level of type must be predicative.

Conclusion

All these considerations raise the following problem about the Curry-Howard isomorphism between propositions and types: is this really an isomorphism? It seems that there is a problem of "levels". We have the choice a priori for the level of programs, as term of degree 3 or of degree 2. If we choose the degree 3, then we have the general polymorphism for programs. The calculus of construction[9] shows that we can add the dependent product in a consistent way. If we choose the degree 2, then we lose definitely the general polymorphism, but we have a clear set-theoretic semantics for the programs and a clear way for the development of proofs about programs in the construction system. The paper of MacQueen[16] is relevant here.

Girard's paradox seems to have some connections with the result of Reynolds[27] that there is no set-theoretical models of the second-order calculus. Actually, this can be used to show that there is no extensional models with a polymorphic notion of equality[13], but it does not seem that it entails directly Reynolds' theorem, as his definition of what is a set-theoretic model has very weak conditions. It is likely that the derivation of Reynolds can produce a non-normalisable term in the general polymorphic calculus, but not shorter that the typed-checked one. More generally, this raises the following questions: is it possible to derive another kind of paradox in the general polymorphic calculus (for example, Russel's paradox)? If possible, could the different ideas behind these paradoxes be characterized by the behavior of the corresponding λ-terms by reduction?

The derivation of a paradox in the general polymorphic system can be seen as the syntactical counterpart of the fact that there is no set-theoretical model of the second-order typed calculus. This is morever perhaps an explanation of this non-existence: the typed systems showed in this paper are formal systems whose syntax is as complex as what is usually regarded as semantics (i.e. set-theory).

Acknowledgements

J.Y. Girard told me that his results on the system U must show that the calculus of constructions with four levels is inconsistent. The idea of presenting Burali-Forti paradox as a general system of notation for ordinals was suggested by G. Huet.

References

[1] R. Amadio, G. Longo. "A type-free look at types as parameters" Universita' di Pisa, 1985

[2] H. Barendregt. "The lambda -Calculus: Its Syntax and Semantics." North-Holland (1980).

[3] R. Burstall and B.Lampson. "A kernel language for abstract data types and modules." Lecture Notes in Computer Science 173, Springer-Verlag, 1984.

[4] N.G. de Bruijn. "lambda-Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem." Indag. Math. **34,5** (1972), 381–392.

[5] N.G. de Bruijn. "A survey of the project Automath." in Curry Volume, Acc. Press (1980).

[6] N.G. de Bruijn. "Some extensions of Automath: the Aut-4 family." Unpublished paper (1974).

[7] L. Cardelli. "A Polymorphic λ-calculus with Type:Type." Private communication (1986).

[8] A. Church. "A formulation of the simple theory of types." Journal of Symbolic Logic (1940), 56–68.

[9] Th. Coquand. "Une Théorie des Constructions." these de 3eme cycle, Paris VII (1985).

[10] Th. Coquand. "Some extensions of the Theory of Constructions." In preparation (1986).

[11] Th. Coquand, G. Huet. "Constructions: A Higher Order Proof System for Mechanizing Mathematics." EUROCAL85, Linz, Springer-Verlag LNCS 203 (1985).

[12] Th. Coquand, G. Huet. "Concepts Mathématiques et Informatiques formalisés dans le Calcul des Constructions." Papier presenté au Colloque de Logique d'Orsay (1985).

[13] J.Y. Girard. "Interpretation fonctionnelle et elimination des coupures de l'arithmetique d'ordre superieur." These d'Etat, Paris VII (1972).

[14] M. Gordon. "HOL A Machine Oriented Formulation of Higher Order Logic." Cambridge Technical Report, no. 68.

[15] J. Lambek. "From types to sets." Advances in mathematics 35 (1980).

[16] D.B. MacQueen. "Using Dependent Types to Express Modular Structure." ACM (1986).

[17] P. Martin-Löf. "A Theory of Types." unpublished (1971).

[18] P. Martin-Löf. "An intuitionistic theory of types: predicative part." Logic Colloquium, North-Holland (1975).

[19] P. Martin-Löf. "Intuitionistic Type Theory." Bibliopolis, (1980).

[20] A.R. Meyer and J.C. Mitchell. "Second-order Logical Relations." Extended abstract (1985).

[21] R. Milner. "A theory of type polymorphism in programming." JCSS 17(3), 348–375 (1978).

[22] J.C. Mitchell. "Lambda Calculus Models of Typed Programming Languages." Ph. D. thesis, M.I.T. (1984).

[23] C. Mohring "Algorithm Development in the Calculus of Constructions." This volume (1986).

[24] W.O. Quine. "Mathematical logic." Harvard University Press (1940).

[25] J.B. Rosser. "The Burali-Forti paradox." Journal of Symbolic Logic 7, 1–17 (1942).

[26] J. C. Reynolds. "Towards a Theory of Type Structure." Programming Symposium, Paris. Springer Verlag LNCS **19** (1974) 408–425.

[27] J.C. Reynolds. "Polymorphism is not Set-Theoretic." Lecture Notes in Computer Science 173, Springer-Verlag, 1984.

[28] B. Russel and A.N. Whitehead. "Principia Mathematica." Volume 1,2,3 Cambridge University Press (1912).

[29] D. Scott. "Data Types as Lattices." SIAM Journal of Computing **5** (1976) 522–587.

[30] G. Takeuti. "Proof Theory." North-Holland, part II (1975).