Carnap's remarks on Impredicative Definitions and the Genericity Theorem. *

Thomas Fruchart
DMI
Ecole Normale Supérieure
45 rue d'Ulm
75005 Paris, France
fruchart@dmi.ens.fr

Giuseppe Longo
LIENS(CNRS) and DMI
Ecole Normale Supérieure
45 rue d'Ulm
75005 Paris, France
longo@dmi.ens.fr

December 5, 1996

Abstract

In a short, but relevant paper [Car31], Rudolf Carnap summarizes the logicist foundation of mathematics, largely following Frege and Russell's view. Carnap moves away though from Russell's approach on a crucial aspect: a detailed justification of impredicative definitions (a formal version of Russell's "vicious circle"), that he accepts. In this note we revisit Carnap's justification of impredicativity, within the frame of impredicative Type Theory. More precisely, we recall the treatment of impredicativity given in Girard's System F and justify it by reference to a recent result, the Genericity Theorem in [LMS93], which may help to set on mathematical grounds Carnap's informal remark. We then discuss the logical complexity of (the proof of) that theorem. Finally, the role of the Genericity Theorem in understanding the surprising "uniformities" of the consistency proof of Arithmetic, via System F, is hinted.

The problem

A definition is said to be impredicative, if it defines a concept in terms of a totality to which the concept belongs. Impredicative definitions, in general, use universal or existensial quantifiers over a variable that could be instanciated by the object being defined. For example, a set S is impredicatively defined, if, for some predicate $\mathcal{P}$, possibly depending on X and y, one has $S = \{y/\forall X.\mathcal{P}(X, y)\}$, where the universally quantified variable X ranges over *all* sets, including the one which is being defined. Similarly, a predicate or a proposition is impredicatively given, if it contains a (quantified) variable which ranges on the collection of predicates or propositions.

*to appear in *Logic in Florence, 1995* (A. Cantini et al. eds), Kluwer, 1997

The "vicious circle" should be clear, as well as the problem it poses, in view also of its frequent use in mathematics (in Analysis in particular). We will discuss it in the frame of todays' Type Theory, where "Types are Propositions" (that is, where Type Theory is seen as a constructive system of Logic, whose "formulae" are given by the types.)

Some History

In Russell's view, a sound foundation of mathematics should not permit vicious circles nor other forms of reflexivity or self-reference, as these could lead to paradoxes. In particular, he claimed that "no whole may contain parts that are definable only in terms of the whole".

The notion of impredicative definition, as considered here, was set on precise grounds also by Poincaré and Weyl ([Poi13], [Wey18]). The first observed as well that, thought doubtful, these definitions did not need to generate paradoxes. Both authors referred to this form of "circularity" in the framework of Set Theory. Weyl, in particular, was essentially interested in Analysis, in his 1918 book, and his contribution turns out today to be a anticipation of a long lasting and informative debate on whether predicative tools in Analysis (and other area of mathematics) suffice for the expressiveness of these topics (see the work of Kreisel, Wang, Feferman, Shutte, Simpson...; see, for example, [Kreis60], [Fefe64], [Fefe75], [Fefe88], [Lon87], [Lon93] for results, discussions and more references; Per Martin-Löf developped a predicative system of Intuitionistic Type Theory, [ML84]).

Carnap's discussion on this matter begins by a criticism of Ramsey's solution to the "vicious circle". Ramsey's solution to the problem of circularity is based on the usual platonist approach to mathematics: the totality of sets or properties already exists in itself, thus the possibility of defining one of them by reference to this totality is an empirical fact that nothing has to do with logic (see [Car31]). Carnap departs from the metaphysics (the absolutism, as he says) of platonic realms and proposes a constructive understanding of predicative definitions, inspired by the practice of mathematics.

Prototype Proofs

Consider first a universally quantified proposition of everyday mathematics, such as $\forall x \mathcal{P}(x)$, where x ranges on some intended collection of individuals (the reals, the complex numbers ...). This is a first order quantification. How do mathematicians prove a universal proposition of this kind? Assume, for instance, that in $\mathcal{P}(x)$, x is a real number variable. We seek to prove $\forall x \mathcal{P}(x)$, i.e. that for *any* real x, $\mathcal{P}(x)$ holds for this specific real. Any working mathematician would prove $\forall x \mathcal{P}(x)$ by saying: $\ll$ let x be an arbitrary or "generic" real number, then observe ...$\gg$ and would write *one* single proof, valid for *any* particular real, as there is no way to inspect all reals, one by one. In other words, his/her core remark would be that the proof does not depend on the specific (and arbitrary) real chosen, but only on *the assumption* that x is a real number. In type-theoretic terms, a sound proof would only depend on the *type* of x, not on its value. Thus, if we want to prove $\mathcal{P}$ for a specific real, π say, we only need to replace everywhere in our proof, the "generic real" by the specific value, π. This is sound, as we used in the proof only properties that are verified by any real.

Herbrand called that kind of "uniform" proofs **prototype**:

> $\ll$... when we say that a theorem is true for all x, we mean that for each x individually it is possible to iterate its proof, which may just be considered a *prototype* of each individual proof.$\gg$ (pp. 288-9, note 5 in [Gold87]).

Impredicatively given Sets and Propositions

Impredicativity explicitly shows up, though, when quantification is given over sets or predicates, not just individuals, as in the first order case: in this case the definiens may use the definiendum, which is a set or a predicate. That is, in order to deal formally with a "totality" of concepts or predicates or sets, one needs a second order language, namely a formalized language that may represent, internally, quantifications over collections of predicates or sets. Note that Analysis may be viewed as second order Arithmetic, since real numbers may be defined as sets of integers.

As already mentioned, the set S above is impredicatively defined, since, for some predicate $\mathcal{P}$, one has $S = \{y/\forall X.\mathcal{P}(X, y)\}$, where y is an individual-variable and X is an set-variable (that is, X is intended to range over *sets* of individuals). Thus, S is a particular set and, if one wants to check wether a given a is in S, one needs to consider the property $\forall X\mathcal{P}(X, a)$, with X ranging over all sets. In particular, one has to handle the "circular" case $\mathcal{P}(S, a)$. One may then wonder if a proof of $\forall X\mathcal{P}(X, a)$ is still possible by the use of a "prototype proof" in the sense given above for first order sentences. In particular, one may wonder what it may mean exactly for an instance of variable X to be "generic", in the second order case.

Carnap's answer to the first question is a positive one. He gives a conceptual justification of this possibility:

> $\ll$ If we had to examine every single property, an unbreakable circle would indeed result. [...] But the verification of a universal logical or mathematical sentence does not consist in running through a series of individual cases. [...] The belief that we must run through all individual cases rests on a confusion of "numerical" generality. [...] We do not establish specific generalities by running through individual cases but by logically deriving certain properties from certain others. $\gg$ [Car31]

In our example, assume that S is a set of integers. In order to check whether n is in S, one should logically derive $\mathcal{P}(X, n)$ from the only assumption that X has the property of "being a set of integers" (i.e. from an assumption on its type and nothing else); thus, we do not run throughout all sets of integers, including the one we are defining, but we just make a logical derivation based on the properties of the "type of the sets of integers".

Carnap continues:

> $\ll$ That proofs that the defined property obtains (or does not obtain) in individual cases can be given shows that the definition is meaningful. If we reject the belief that it is necessary to run through individual cases and rather make it clear to ourselves that the complete verification of a statement means nothing more than its logical validity for an arbitrary property, we will come to the conclusion that impredicative definitions are logically admissible. $\gg$ [Car31]

Carnap claims that the fact that a proof may be (correctly and logically) given $\ll$shows that the definition is meaningful$\gg$. In modern terms, this seems very close to the "realizability interpretation" in Intuitionistic Logic: according to the intuitionistic approach, that we follow in this paper, the meaning of a statement is given by the "set of its realizers" (that is, its possible or candidate proofs - if any, as this set may be empty), [Troe73]. This constructive understanding of logical systems is also related the so called BHK explanation of the intuistionistic meaning of the logical connectives (BHK stands for Brouwer-Heyting-Kolmogorov, [Troe73]). In both semantics, one constructively gives meaning to a statement, to a defined mathematical concept, by discussing its provability, as truth *is* provability. Carnap seems to claim that the possibility of an analysis of provability justifies "logical admissibility". We share this opinion.

However, some technical questions need to be clarified. How can we be sure that the proof given is actually independent from the specific X used throughout the proof? How can we prove that from the given proof of a specific case, we can actually reconstruct *uniformly* a proof of the general case, that is of $\forall X.\mathcal{P}(X, a)$? Or that X is truely generic and that the proof is a prototype, in the sense of Herbrand?

Mathematicians solve these questions, in the practice, by handwaving, experience and a common insight into proofs. This is perfectly sound in the first order case, where the stratification of individuals and predicates (or sets) poses no problem. It is a more delicate point in second order systems, when the issue of impredicativity actually raises.

Constructive logical systems, Type Theory in particular, by looking at proofs explicitly, as terms or computations, gives a precise answer to these questions, as a simple corollary of a (difficult) result. In our views, this may contribute to set on more solid grounds impredicatively given properties.

System Fc and Impredicativity

System F is known as Impredicative Type Theory, or Polymorphic Lambda Calculus; it was introduced by Girard, [Gir71] (see [GLT89] for a recent presentation). It consists of *types* and *terms* (well typed terms). One has to stress that the type system of F allows *explicit quantification on type variables* (second order quantification); its proof-theoretic strenght is such as to prove, by the normalization theorem, the consistency of (II order) Arithmetic. Let us first recall the syntax and rules for this calculus.

A **type** is either a type variable, an arrow type or a polymorphic type (one may add atomic types, when dealing with specific extension of the "pure theory" presented here; however, most key predicates for Logic and Computer Science are codable in the pure system, see [GLT89]). Types then are constructed using the following schemes:

- *variables:* X, Y ... are types.
- *arrow types:* $\tau \rightarrow \sigma$ is a type, if σ and τ are types.
- *universal types:* $\forall X.\tau$ is a type, if τ is a type.

A **term** is either a variable, an abstraction, an application, a type abstraction, or a type application. Thus, terms are constructed using the following schemes:

- *variables:* x^τ of type τ, if τ is a type.
- *abstraction:* $\lambda x^\tau.M$ of type $\tau \rightarrow \sigma$, if σ is the type of M (λ *binds* term-variables in terms, or x^τ is not *free* in $\lambda x^\tau.M$).
- *application:* MN of type σ, if M is of type $\tau \rightarrow \sigma$ and N of type τ.
- *type abstraction:* $\Lambda X.M$ of type $\forall X.\tau$, if M is of type τ and X is not free in the type of any free term-variable of M (Λ binds type-variables in terms).
- *type application:* $M\sigma$ of type $\tau[\sigma/X]$, if M is of type $\forall X.\tau$ (where $\tau[\sigma/X]$ is the result of the replacement of all free occurrences of X by σ in τ.)

Reduction rules

$$(\lambda x^\tau.M)N \xrightarrow{\beta_1} M[N/x] \qquad \lambda x^\tau.(Mx) \xrightarrow{\eta_1} M \text{ if } x \text{ is not free in } M$$
$$(\Lambda X.M)\tau \xrightarrow{\beta_2} M[\tau/X] \qquad \Lambda X.(MX) \xrightarrow{\eta_2} M \text{ if } X \text{ is not free in } M$$

We will write $\xrightarrow{F}$ for the reflexive and transitive closure of the union of these reductions, and $=_F$ for the symetric closure of $\xrightarrow{F}$.

Second order quantification is explicitly given by the rule of *type abstraction*, which binds second order variables, both in types and terms. Thus, $\forall X.\tau$ is defined, as a type, by a quantification over the collection of all types, which includes $\forall X.\tau$, the "definiendum". Moreover, by virtue of the Curry-Howard isomorphism, polymorphic types can be viewed as second order logical propositions. Hence, a type of the form $\forall X.\tau$ "means" that the property τ is possessed by *all types* (in particular, by $\forall X.\tau$ itself). Finally, the terms of system F compute functions, either on *terms* (see the β_1 axiom) or on *types* (see the β_2 axiom). In particular, a polymorphic term takes types as inputs (including, possibly, its own type) and gives terms as outputs, by β_2. Here is then a the typical form of impredicativity, namely a type-theoretic formalization of the "vicious circle". It even shows up at two levels, for types and for terms. However, exactly because of this constructive frame, where proofs are seen as computations (are coded by terms), it will be possible to look closely at the nature of "prototype proofs". Indeed, we want to argue that these impredicative constructions are *safe* along the lines of Carnap's argument, by showing that system F contains a precise notion of prototype proof, in the sense of Herbrand, and of generic types, with strong "coherence" properties.

Axiom C and its meaning

The key point of our analysis relies on the observation that all types in system F are "generic", in the sense to be specified, and that, from a computational point of view, they act like variables. In short, in the constructive frame of second order Type Theory, outputs do not depend on inputs, when the input is a type: in this case, only the *type* of the output may depend on the input, not its "value" (see below). This is blatantly false for first order terms, in Type Theory or everywhere in mathematics: functions and computations do depend on inputs as first order individuals, as soon as a sufficiently expressive system is given.

This observation will be established in two steps. First, in this section, by a simple remark on the compatibility of an axiomatic extension of system F; later on, by the Genericity Theorem.

The first remark is inspired by a result in [Gir71]: in system F, there is no definable term that discriminates between types. That is, there is no term J_σ such that J_σ applied to type ρ is 1 if $\sigma = \rho$, and is 0 if $\sigma \neq \rho$. In other words, there is no term whose output values are all in the same type (the type of integers, or any other type with at least two elements) and depend on the input type. This idea was taken up in [LMS93] by extending system F with the following axiom, some sort of a "generalized dual" of Girard's result [1]:

[1] Independently of Girard's remark, in [CMMS91] a similar extension was proposed, for the purposes of subtyping, a notion motivated by programming.

Axiom C: *If* $M : \forall X.\sigma$ *and* $X \notin FV(\sigma)$ *then for all* τ, τ', $M\tau = M\tau' : \sigma$.

Axiom C intuitively means that an input type (τ), which is not used to establish the type (as σ does not depend on X) of the corresponding output value ($M\tau$), bears no information as input. So if M has the type $\forall X.\sigma$ and X is not free in the type σ (i.e. σ is not a function of X), then it does not matter whether one applies M to τ or τ' and one may consider both results to be equal. Equivalently, since there are no type discriminators by Girard's remark, Axiom C forces terms of universally quantified type, whose outputs live in the same type, to be constant. Informally, this is sound, because we are in a constructive frame and types have the intended meaning of a possibly infinite domain of interpretation. Thus a term, as effective computation, cannot compare nor discriminate on the grounds of possibly infinite information.

We write Fc for the extension of F by Axiom C. Axiom C is not derivable in system F. Indeed, let $x : \forall X.\sigma$ and $X \notin FV(\sigma)$ then for any τ and τ', $x\tau$ is a normal form, different from $x\tau'$. However, Axiom C is formally *compatible* with F, as there are models of Fc. As a matter of fact, all known and non-trivial models (e.g. not term-models nor models of Type:Type) realize Axiom C. In short, all "parametric" models of F, in the sense of Reynolds, [MR92], the coherent domains, [Gir86], and the PER models are all models of Fc [2]. It is also possible to extend system F by a reduction relation which is strongly normalizing, Church-Rosser and induces exactly the Fc-equality (which is thus decidable, [Bel97]).

The soundness of Fc gives the first hint towards the "generic" nature of types as inputs, in system F. That is, we may consistently consider each type exactly as a variable, at least under the special circumstances that it is an input for a term M of type $\forall X.\sigma$, where $X \notin FV(\sigma)$

The Genericity Theorem, Prototype Proofs, Generic Types.

In [LMS93], Axiom C was introduced in order to prove the Genericity Theorem below (note that there is no restriction on σ).

Theorem (Genericity). *Let* M *and* N *have type* $\forall X.\sigma$. *Then:* (Exists τ, $M\tau =_{Fc} N\tau$) $\Longrightarrow M =_{Fc} N$.

This theorem shows that two polymorphic terms that are equal on *one* input type are equal on *any* input type. In other words, the behaviour of polymorphic terms is so "uniform" that one can reduce Fc equality on *every* possible types to Fc equality on one *single* type τ (no matter which one!). That is, if ($M\tau =_{Fc} N\tau$) then $MX =_{Fc} NX$.

[2] The categorical significance of the PER models, i.e. the meaning of "quantification as product" as well as the meaning of Axiom C are both given by the validity in the Effective Topos of the Uniformity Principle, see [Lon87], [LM91], [Lon95].

For the purposes of our forthcoming application, observe that the Genericity Theorem in [LMS93] is actually shown by proving the following Main Lemma:

Let M and N have type σ. Then:
(Exists $\tau, [\tau/X]M =_{Fc} [\tau/X]N) \Longrightarrow M =_{Fc} N$.

With reference to very different matters, as an anology, one may remember the "regular behaviour" of analytic functions of complex variable: when known on the border of a regular shape, they are known everywhere inside. Thus, if two of these functions coincide on the border, they coincide everywhere [Rud80].

This regularity or uniformity could be surprising, and perhaps it is, but this is actually what makes second order impredicative Type Theory become safe (and allows the proof of the Normalization Theorem, see the last section).

Recall now that, if M is a polymorphic term, it can take as input *any* type, and in particular types that are more complex that its own type. One can then wonder what happens in these "circular" cases. The answer inspired by the Genericity theorem is: "It happens the same as on a simpler input type, because the computational behaviour of M "in extenso" is determined by its behaviour on a single type". In this sense, the term or computation does not depend on the input type.

As shown in the next section, the proof of Genericity is (difficult but) "elementary". More precisely, it is possible to code it into PRA (Primitive Recursive Arithmetic), provided that the Church-Rosser property for system F is assumed (but this is elementary and easy). Indeed, PRA is an elementary arithmetic theory, where one can handle basic mathematical computations and deductions. Thus, in spite of the circularity generated by polymorphism, a strong "regularity" property of terms in system Fc is established by the Genericity Theorem and the proof of this regularity is elementary, i.e. logicaly complex reasonings are not necessary to deal with this key property of the impredicativity of Fc.

According to the realizability or BHK interpretations, the constructive meaning of $\forall X.\sigma$, the crucial, impredicatively given type, is the following. A proof-term $M{:}\forall X.\sigma$ is a computation or function that takes *any type* ρ to a proof $M\rho{:}[\rho/X]\sigma$. Thus, from a term $M{:}\forall X.\sigma$ one can reconstruct the terms or proofs $M\rho$ for *each* specific instance $[\rho/X]\sigma$ of σ. However, as Carnap stresses, a proof of $\forall X.\sigma$ is not constructed by running through all specific cases or input types ρ, but by giving a prototype proof, in the sense of Herbrand, w.r.t. a "generic" instance ρ. The most obvious prototype proof and generic case is given by $MX{:}\,\sigma$. However, this does not save us from the circularity of impredicativity, as variables in Type Theory have a double "status": they are atomic entities (types in this case) but they also formally represent the mathematical use of "variable" as arbitrary elements of the intended domain of variation, since they may be instantiated by any element of that domain. Then, in particular, the variable X which may occur in σ can be instantiated by σ or even $\forall X.\sigma$.

Our thesis though, in view of the Genericity Theorem, is that an arbitrary specific instance type, possibly simpler than σ, may suffice to determine a fully general proof. The idea then is to start from a specific instance $[\rho/X]\sigma$ and discuss the prototype nature of its proof, if any. In a sense we want to describe the backwards process, w.r.t. the one described above, as we want to go from a proof of $[\rho/X]\sigma$ to one of $\forall X.\sigma$.

Assume then that from a proof N of an instance $[\rho/X]\sigma$ one tries to reconstruct a proof of the universal proposition $\forall X.\sigma$. In general, this may not be possible. It is possible, though, when the *structure* of a specific proof N of $[\rho/X]\sigma$, that is of $N{:}\,[\rho/X]\sigma$, is "parametric" in ρ or it may be described uniformely as a substitution of a type variable by ρ. In that case, we call N a prototype proof, in the sense of Herbrand. This is formalized by the following definition:

> **Definition.** *Given a type σ, we say that a type ρ is* **generic** *and a proof $N{:}\,[\rho/X]\sigma$ is a* **prototype** *if there exists $M{:}\,\sigma$, such that X is not free in the type of a free term variable of M and*
> $$[\rho/X]M =_{Fc} N{:}\,[\rho/X]\sigma.$$

Notice that if ρ and a proof $N{:}\,[\rho/X]\sigma$ are, respectively, generic and prototype, by $M{:}\,\sigma$, then $\Lambda X.M{:}\,\forall X.\sigma$. That is, the construction (existence) of M, from the prototype and generic proof and type N and ρ, allows to give a proof of the universal statement. As the converse is trivial, given a type, there exist a generic and prototype type and proof if and only if the corresponding universal statement is provable. Clearly, not any proof nor type of a specific instanciated type need be prototype and generic: for example, $[\rho/X]X$ has no prototype proof with ρ generic, otherwise the universal statement $\forall X.X$, the absurdum or empty type, would be provable. We claim that, given N,σ and ρ, it is decidable whether N is a prototype proof with ρ generic (ongoing work).

We focused on a second order notion of prototype proof and generic type. Of course, the definition can be easily extended to first order statements, the more usual ground of "prototype" proofs in mathematics: if r is an arbitrary real and the proof of $P(r) = [r/x]P$ does not depend on r (or r is generic and the proof is a prototype similarly as in the definition above), any mathematician would say that we actually proved $\forall x.P(x)$. However, as already mentioned, the first order case is not problematic at all: individuals are distinct from propositions and there is no apparent vicious circle. This is not so in the impredicative case, which motivates the doubts of many in the use of impredicative second order quantifications (and variables). Thus, the very simple notions of generic types and prototype proof turn out to be a more delicate issue in impredicative systems.

However, exactly because of the relevant property of System F given by the Genericity Theorem, we are now able to assure that prototype proofs are sound, also when the generic type may be as complex as the universal assertion to be proved. The soundness is given, as in Category Theory, by a "*coherence* result", which states the independence of the reconstruction of the proof of the universal statement from specific prototype proof and generic types.

> **Corollary (Coherence).** *Given a type σ, let ρ and a proof $N : [\rho/X]\sigma$ be generic and prototype, respectively. Then, if $[\rho/X]M =_{Fc} N =_{Fc} [\rho/X]M' : [\rho/X]\sigma$, one has $M =_{Fc} M'$ and, thus, $\Lambda X.M =_{Fc} \Lambda X.M' : \forall X.\sigma$.*

This immediate corollary to the Genericity Theorem says that no matter how we extract a proof of a universal statement from a prototype one of a specific instance, in any case we obtain just one proof (modulo "$=_{Fc}$"). Thus, also the type ρ does not matter, or it is truely generic, since from the **unique** proof $\Lambda X.M$ of $\forall X.\sigma$ we can obtain, uniformely and effectively, proofs for each instance $[\rho'/X]\sigma$, just by application $(\Lambda X.M)\rho'$.

The independence of the proof of the universal statement from the specific "structure" of a proof of a specific instance, as well as from the generic type used, garanties that, exactly in the "shaky" second order case, the mathematical soundness of those statements. The system is "coherent" both in the categorical-technical sense, and in the sense of the possibility of disregarding the complexity of the instantiating type, since all types are generic and act like variables.

As already mentioned, this garanty is given "exactly" in the critical second order case, as the (obvious variant of the) Genericity Theorem is clearly (and fortunately) false in the first order case.

The proof of Genericity Theorem in PRA.

So far we tried to justify the "non-elementary" tools of impredicativity by an application of the Genericity Theorem. We next show that the proof of the Genericity Theorem *is* elementary. Our goal is to focus here on the main points that allow to encode the Genericity Theorem into Primitive Recursive Arithmetic (PRA) without many detail, since the complete coding would be very long (and not very interesting). Hence, we first recall the basics of PRA, then we explain how to code types, terms, and properties of system F into PRA, and finally how to code the Genericity *proof* into PRA.

Primitive Recursive Arithmetic

The language of this theory contains one function letter for each primitive recursive function, and one predicate letter for each primitive recursive predicate. The axioms of PRA are the (usual) equality axioms, the definitional axioms— that define the behaviour of primitive recursive functions letters (resp predicates letters) as expected— and the *induction* axiom:

$$\mathcal{P}(0) \wedge \forall x(\mathcal{P}(x) \rightarrow \mathcal{P}(Sx)) \rightarrow \mathcal{P}(y)$$

where $\mathcal{P}$ is quantifier free, and Sx stands for the successor of x.

In PRA, one may express the usual mathematical reasonings. Some difficult theorems can be proved in this theory (Gödel first incompleteness theorem, for instance). The core of its expressive power is the induction axiom.

This axiom defines primitive recursive induction, that is induction on ω. More complex forms of induction (like induction on ω^2, used to prove the totality of the Ackerman function) are not possible in PRA. In what follows, we sketch the proof that induction on ω suffices for coding and proving the Genericity Theorem in PRA. For more details see [Fru96].

Coding F into PRA

Types and terms of F will be coded by *integers*, type assignment by a *function letter*, reductions by *predicate letters*, and finally equalities by *formulas* of PRA. This coding is rather long, so we just mention here the key points.

We first define the code of types and terms of system F by induction respectively on types and terms. We will write $^\dagger M^\dagger$ for the code of M, and $< a_0, \ldots, a_{n-1} >$ for the integer $2^{a_0+1} \times \cdots \times p_n^{a_{n-1}+1}$, where p_n stands for the n^{st} prime number:

$$\begin{array}{rcl@{\qquad}rcl}
\multicolumn{3}{c}{Types} & \multicolumn{3}{c}{Terms} \\
 & & & ^\dagger t_1\ t_2{}^\dagger & = & < 3, {}^\dagger t_1{}^\dagger, {}^\dagger t_2{}^\dagger > \\
^\dagger\forall X_i.T^\dagger & = & < 5, {}^\dagger X_i{}^\dagger, {}^\dagger T^\dagger > & ^\dagger\lambda x_i^\tau.t^\dagger & = & < 7, {}^\dagger x_i^\tau{}^\dagger, {}^\dagger t^\dagger > \\
^\dagger T_1 \to T_2{}^\dagger & = & < 9, {}^\dagger T_1{}^\dagger, {}^\dagger T_2{}^\dagger > & ^\dagger t.T^\dagger & = & < 11, {}^\dagger t^\dagger, {}^\dagger T^\dagger > \\
^\dagger X_i{}^\dagger & = & 13 + 4i & ^\dagger\Lambda X_i.t^\dagger & = & < 15, {}^\dagger X_i{}^\dagger, {}^\dagger t^\dagger > \\
 & & & ^\dagger x_i^\tau{}^\dagger & = & 19 + 4 < i, {}^\dagger\tau^\dagger >
\end{array}$$

Both coding and decoding are primitive recursive. Hence this allows to define functions and predicates by induction on the code of types (respectively terms). Since codes are integers, those inductions are usual "course of value" inductions, i.e, ordinary primitive recursive inductions.

For instance, one may define the (primitive recursive) predicates $Type(n)$ and $Term(n)$ which mean respectively "n is the code of a type of F", and "n is the code of a (well typed) term of F", by course of value induction on ω.

The main function, that is also defined in this way, associates the code of its type to each code of a well typed term. This function is called *typing*, and its definition codes formally the following property:

$$\begin{array}{rcl}
typing({}^\dagger M^\dagger) & = & {}^\dagger T_2{}^\dagger \text{ if } M \equiv t_1 t_2,\ t_2 : T_1, \text{ and } t_1 : T_1 \to T_2 \\
 & & {}^\dagger T_1 \to T_2{}^\dagger \text{ if } M \equiv \lambda x^{T_1}.t, \text{ and } t : T_2 \\
 & & {}^\dagger T_1[T_2/X]^\dagger \text{ if } M \equiv t.T_2, \text{ and } t : \forall X.T_1 \\
 & & {}^\dagger\forall X_i.T^\dagger \text{ if } M \equiv \Lambda X_i.t, \text{ and } t : T, \text{ and } X_i \text{ is not free in} \\
 & & \quad \text{the type of a free term variable of } t
\end{array}$$

$^\dagger\tau^\dagger$ if $M \equiv x^\tau$

0 else

$FcEqual$ is the main predicate used in the proof of the Genericity Theorem in PRA. It is *not* a predicate letter of PRA, but it is defined directly by the use of the primitive recursive predicate letter $FcEqualproof$ such that:

> $FcEqualproof(a, b, c)$ means "c is the code of an Fc equality proof between both terms coded respectively by a and b".

The definition of $FcEqualproof$ is complex, and requires in particular the coding of F reductions (β_1, β_2, η_1, and η_2), which would be too long to present here in detail. Thus, we just mention the *possibility* of coding Fc equality, $M =_{Fc} N$ with the predicates $FcEqual$:

$$FcEqual(^\dagger M^\dagger, ^\dagger N^\dagger) \equiv \exists c\ FcEqualproof(^\dagger M^\dagger, ^\dagger N^\dagger, c)$$

By this coding, one can express the Genericity Theorem by the following formula of PRA:

$$Type(\sigma) \wedge (typing(^\dagger M^\dagger) = {}^\dagger\forall X_0.\sigma^\dagger) \wedge (typing(^\dagger N^\dagger) = {}^\dagger\forall X_0.\sigma^\dagger) \wedge$$

$$\big(\exists\tau(Type(\tau) \wedge FcEqual(^\dagger M\tau^\dagger, ^\dagger N\tau^\dagger)\big)$$

$$\Downarrow$$

$$FcEqual(^\dagger M^\dagger, ^\dagger N^\dagger)$$

Now we know how to code the Genericity Theorem by a formula of PRA, we have to find how to code its *proof.*

Structure of the proof

The proof of the Genericity Theorem is rather long, and divided in several lemmas. As it would be extraordinarily tiresome to code the proof lemma after lemma, and line after line, we just point out the main arguments that are used in the proof, and we explain how they can be coded by deductions in PRA.

In what follows, we will not consider the basic logical reasonings (that can be coded easily), but, first the main direct arguments used for proving a "type generalization" lemma, and then the inductive arguments.

Direct arguments (no induction)

As a matter of fact, the Genericity Theorem deals with *type substitution*: actually, a (trivially) equivalent wording for it (we just apply a β_2 reduction) is the following (formulated as the Main Lemma in [LMS93]):

$$M, N:\ \sigma \wedge\ [\tau/X]M =_{Fc} [\tau/X]N \Longrightarrow M =_{Fc} N$$

Thus, the use of type-substitution properties in the Genericity proof is absolutely not surprising. The most important properties of type substitution are called "type generalization", and "term generalization". We focus here on the first one that deals with type substitution in types themselves (Lemma 5.1 in [LMS93]):

If σ_1 and σ_2 are two types such that $\sigma_1[\tau/X] \equiv \sigma_2[\rho/Y]$, then exists σ_0 (type generalizer of σ_1 and σ_2) such that, for suitable μ_1 and μ_2:

$$\sigma_0[\mu_1/Z] \equiv \sigma_1, \;\; \sigma_0[\mu_2/Z] \equiv \sigma_2$$

The proof of this lemma is based on the notion of *occurrence*.

Occurrences are usually defined by induction on the structure of terms. We can here code this notion with integers, by the use of a (primitive recursive) induction on codes of terms, and then code the following notions by induction on the codes of occurrences:

- substitution at an occurrence: $\sigma[u \leftarrow \tau]$ is the result of the substitution of τ at the occurrence u.
- subterm $(\sigma_{|u})$ taken (from σ) at the occurrence (u).

Since all those notions are defined by primitive recursive induction, it is possible to prove the following properties, also by primitive recursive induction (on the occurrences):

$$\sigma[u \leftarrow \sigma_{|u}] = \sigma$$

$$(\sigma[u \leftarrow X])[Y/X] = \sigma[u \leftarrow Y] \text{ for } X \notin Var(\sigma)$$

Now, these properties are essentially sufficient to prove Lemma 5.1. Other arguments that are also used in this proofs are not inductive, (propositional reasoning, case analysis) and easy to formulate by the corresponding deduction in PRA.

Although induction is not explicitly used in the proof of Lemma 5.1 above, the easiest way to formalize it in the frame of PRA is based on induction, as induction is the key tool in PRA and the notion of occurrence is given inductively.

Coding inductions

Different kinds of inductions are used in the proof of Genericity: inductions on types, on terms, on type derivation, on the length of an equalities (or reduction) chain, and on the number of C-equalities in chains of Fc-equalities .

Since types, terms, and chains are coded by integers, one may code inductions on them by inductions on integers. This does not present any problem as the code of a truncated type (resp. term, chain) is strictly less than the code of the original type (resp. term, chain).

As for induction on type derivations, proving $\mathcal{P}(M)$ by such an induction is a "pleasant way" of proving $Term(M) \rightarrow \mathcal{P}(M)$ by induction on *terms*, ($Term$

is the predicate letter that codes the fact that its argument is a *well typed term* of F).

In short, the coding of induction is relatively straightforward, because all inductions used in the Genericity proof are primitive recursive, hence codable into PRA.

Conclusion

The Church-Rosser property (CR) for system F is the only non logical assumption in the proof of the Genericity Theorem [3]: If $M \xrightarrow{F} M_1$ and $M \xrightarrow{F} M_2$, then exists M_0 such that:

$$\begin{array}{ccccc} M_1 & & & & M_2 \\ & \searrow^F & & \swarrow_F & \\ & & M_0 & & \end{array}$$

This property is not more difficult to establish for system F than for the type-free λ-calculus. Hence, the Genericity Theorem is based on a (elementary) property of system F and its proof can be given in PRA. In this sense, one can say that Genericity Theorem is *elementary.*

Second order arithmetic (PA_2)

System F verifies the strong normalisation property: all terms of F have a normal form (i.e: are F-equal to an irreducible term). From a logical point of view, the strong normalisation theorem for system F implies the consistency of *second order arithmetic,* PA_2, and — in virtue of Gödel's second incompleteness theorem — it can not be proved within PA_2. As we will see, the strength and main difficulties of the proof of this theorem are due to the presence of polymorphism, and are related to impredicativity.

The proof of F normalisation is due to Girard, who adapted Tait's notion of *reducibility* [GLT89][Tait67]. At first trial, an intuitive notion of reducibility for F terms could be the following:

> ≪We would like to say that t of type $\forall X.\tau$ is *reducible* iff for all types σ, $t\sigma$ is reducible (of type $\tau[\sigma/X]$). [...] But σ is arbitrary—it may be $\forall X.\tau$—and we need to know the meaning of reducibility of type σ before we can define it! We shall never get anywhere like this.≫ [GLT89, p. 115]

As a matter of fact, the impredicativity of F creates logical difficulties. The key idea that makes the proof possible is based on the notion of *reducibility with parameters.* The parameter is called "reducibility candidate", and it is

[3] CR is used in several places in [LMS93]. As a matter of fact, in the proof of Theorem 6.2, there is a reference to Strong Normalization. However, this reference, which is just a shortpath to the proof, may be easily replaced by a simple application of CR (see [Fru96]).

taken as a temporary definition of reducibility on some types, to break the "circle" discribed in the citation above. The point is that no matter which collection of candidates is chosen, its use as a (type) parameter does not affect the computation (nor the proof).

> ≪A term t of type $\forall X.\tau$ is reducible when, for every type σ and *every reducibility candidate* R of type σ, the term $t\sigma$ is reducible of type $\tau[\sigma/X]$, where reducibility for this type is defined taking R as the definition of reducibility for σ. Of course, if R is the "true" reducibility of type σ, then the definition we shall be using for $\tau[\sigma/X]$ will also be the "true" one. **In other words, everything works as if the rule of universal abstraction** (which forms functions defined for arbitrary types) **were so *uniform* that it operates without any information at all about its arguments.**≫ [GLT89, p. 115]

The *main* definitions (and properties) that are necessary for the proof of F normalisation use implicitely the essential feature of F, that is informally described above: the regularity of polymorphic terms behaviour. Since this "uniformity" allows the (difficult) proof of strong normalisation, it is then at the core of this consistency proof of PA_2.

In conclusion, one can now consider the Genericity theorem as a (partial) formalisation of this feature. The regularity of the universal abstraction rule, that operates ≪without any information at all about its argument≫ corresponds in our views, to the fact that *any* type is *generic* is the sense described above. One can then wonder about a possible use of the Genericity Theorem for a better understanding of the "mysteries" of the consistency proof of Arithmetic. This issue is part of our current research project.

Aknowledgement

We would like to thank Sergei Soloviev for his detailed comments to this paper. Longo would also like to thank Sergei for the long lasting stimulating collaboration in Type Theory and for the many discussions on key issues in Logic and the foundation of Mathematics.

References

[Bel97] G. Bellè. Syntactical properties of an extension of Girard's System F where types can be taken as "generic" inputs. Preliminary note, DISI - Università di Genova. E-mail: gbelle@disi.unige.it.

[Car31] R. Carnap. The logicist foundation of mathematics, in P. Benacerraf, H. Putnam, *Philosophy of mathematics; selected readings*, Prentice-Hall philosophy series, 1964.

[CMMS91] L. Cardelli, S. Martini, J.C. Mitchell, and A. Scedrov. An extension of system F with subtyping. *Information and Computation* 94, pages 4–56, 1994. First appeared in the proceedings of the Conference on Theoretical Aspects of Computer Software (Sendai, Japan), T. Ito and R. Meyer, eds., Lecture Notes in Computer Science 526, pages 750–770, Springer-Verlag, 1991.

[Fefe64] S. Feferman. Systems of predicative Analysis, *Journal of Symbolic Logic* vol. 29, pages 1–30, 1964.

[Fefe75] S. Feferman. A language and axioms for explicit mathematics, in *Lecture Notes in Mathematics* 450, Springer-Verlag, pages 87–139, 1975.

[Fefe88] S. Feferman. Weyl vindicated: "Das Kontinuum" 70 years later, *Proceedings of the Cesena Conference in Logic and Philosophy of Science, 1988.*

[Fru96] T. Fruchart. *Normalisation et Généricité dans le système F*, rapport de stage de DEA. Available on author's Home page: http://www.ens.fr/~fruchart/publi.html.

[Gir71] J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. Proceedings of the *2nd Scandinavian Logic Symposium*, J.E. Fenstad, ed., pages 63–92, North-Holland, 1971.

[Gir86] J.-Y. Girard. The system F of variable types, fifteen years later, *Theoretical Computer Science*, vol 45, pages 159–192.

[GLT89] J.-Y. Girard, Y. Lafont and P. Taylor. *Proofs and types.* Cambridge Tracts in Theoretical Computer Science 7, Cambridge University Press, 1989.

[Gold87] H. Goldfarb, *Jacques Herbrand: logical writings*, 1987.

[Kreis60] G. Kreisel. La prédicativité, in *Bulletin de la Société Mathématique Française*, vol. 88, pages 371–391, 1960.

[Lon87] G. Longo. Some aspects of impredicativity: notes on Weyl's philosophy of Mathematics and on todays Type Theory, *Logical Colloquium 87*, Studies in Logic (Ebbinghaus et al. eds), North Holland, 1989.

[Lon93] G. Longo. Review of Feferman's paper "Weyl vindicated, Das Continuum 70 years later", *Journal of Symbolic Logic*, vol 58, n. 3, 1993.

[Lon95] G. Longo. Parametric and type-dependent polymorphism. *Fundamenta Informaticae*, 22(1-2):69–92, 1995.

[LM91] G. Longo and E. Moggi. Constructive natural deduction and its ω-set interpretation. *Mathematical Structures in Computer Science* vol. 1, pages 215–253, 1991.

[LMS93] G. Longo, K. Milsted, and S. Soloviev. The Genericity Theorem and the notion of parametricity in the polymorphic λ-calculus. *Theoretical Computer Science* 121, pages 323–349, 1993.

[MR92] Q. Ma and J.C. Reynolds. Types, abstraction, and parametric polymorphism, part 2. Proceedings of the *Conference on Mathematical Foundations of Programming Semantics*, S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, eds., Lecture Notes in Computer Science 598, pages 1–40, Springer-Verlag, 1992.

[ML84] P. Martin-Löf *Intuitionistic Type Theory*, Bibliopolis, Napoli, 1984.

[Poi13] H. Poincaré. *Dernières pensées*, english edition, Dover publ., New York 1963.

[Rud80] F. Rudin. *Real and Complex Analysis*, MacGraw Hill, 1980.

[Tait67] W.W. Tait. Intensional interpretation of functionals of finite type I, *Journal of Symbolic Logic* 32, 1967.

[Troe73] A. Troelstra. *Metamathematical investigation of intuitionistic arithmetic and analysis*, Lecture Notes in Mathematics 344, Springer Verlag, 1973.

[Wey18] H. Weyl. *Das Kontinuum*, italian edition, care of B. Veit, Bibliopolis, Napoli, 1977.