Fundamenta Informaticae 100 (2010) 1–10
DOI 10.3233/FI-2010-0000
IOS Press

A Note on Forcing and Type Theory

Thierry Coquand*
Computer Science Department
Gothenburg University
S 412 96 Göteborg, Sweden
`coquand@chalmers.se`

Guilhem Jaber
Computer Science Department
Gothenburg University
S 412 96 Göteborg, Sweden

Abstract. The goal of this note is to show the uniform continuity of definable functional in intuitionistic type theory as an application of forcing with dependent type theory.

Keywords: Type theory, dependent types, forcing

1. Introduction

The goal of this note is to show the uniform continuity of definable functional in intuitionistic type theory as an application of forcing with dependent type theory. The discovery of uniform continuity of definable functional originates in Brouwer's work [5], who proved as a corollary of his bar theorem that, in intuitionistic mathematics, any function everywhere defined on the unit interval is uniformly continuous. The technique of using forcing to prove uniform continuity of functional is presented in Beeson's book *Foundations of Constructive Mathematics*. This proof can be seen as a possible formal counterpart of Brouwer's arguments. However, Beeson's book contains no treatment of Martin-Löf type

*Address for correspondence: Computer Science Department, Gothenburg University, S 412 96 Göteborg, Sweden

theory and this is formulated as an open problem: "Forcing has yet to be worked out directly for Martin-Löf's system" [3]. We present in this note a possible way to combine forcing and intuitionistic type theory.

The first step is to specify the version of type theory we are working with. This version, so called *intensional* type theory, is quite close to the one presented by P. Martin-Löf [11]. The notion of *computability*, introduced by Gödel [7] for a simpler type system, can be defined also for the present theory. We extend it with a "Cohen real", a generic function from natural numbers to Boolean, and explain how to define a suitable notion of computability for this extension. We show then that any well-typed term is computable. The uniform continuity of definable functional is then a direct corollary.

2. (Standard) Type Theory

2.1. Terms

The terms of Type Theory are untyped λ-calculus extended with constants, and with the following syntax.

$$t, u, A, F ::= x \mid \lambda x.t \mid t\; t \mid c \mid f$$

We consider terms up to α-conversion. There are two kinds of constants: *constructors* $c, c', \ldots$ and *defined constants* $f, g, \ldots$. We consider only the (recursively) defined constant natrec and natrec_k with the reduction rules

$$\mathsf{natrec}\; a\; g\; 0 \to a \qquad \mathsf{natrec}\; a\; g\; (\mathsf{S}\; n) \to g\; a\; (\mathsf{natrec}\; a\; g\; n)$$

and, for each $j < k$

$$\mathsf{natrec}_k\; a_0\; \ldots\; a_{k-1}\; j \to a_j$$

This forms an extension of β-reduction which still has the Church-Rosser property [11], sometimes called β, ι-reduction [2]. We write $t_1 = t_2$ to mean that t_1 and t_2 have a common reduct for β, ι reduction.

The constructors are $U, N, N_k,\; j$ (arity 0), for $k, j = 0, 1, \ldots$, S (arity 1) and Π (arity 2). If n is a natural number, we may write $\overline{n}$ instead of $\mathsf{S}^n\; 0$. We write $(\Pi x{:}A)B$ instead of $\Pi\; A\; (\lambda x.B)$, and $A \to B$ instead of $\Pi\; A\; (\lambda x.B)$ if x is not free in B. A *context* is a sequence $x_1 : A_1, \ldots, x_n : A_n$; we write $()$ for the the empty context.

2.2. Typing rules

They are three forms of judgements

$$\Gamma \vdash A \qquad \Gamma \vdash t : A \qquad \Gamma \vdash$$

The last judgement $\Gamma \vdash$ expresses that Γ is a well-typed context. We may write $J\; [x : A]$ for $x : A \vdash J$.

The typing rules are as follows. The rules for forming contexts are

$$\frac{}{() \vdash} \qquad \frac{\Gamma \vdash A}{\Gamma, x : A \vdash}$$

The rules for forming types are

$$\frac{\Gamma \vdash}{\Gamma \vdash U} \qquad \frac{\Gamma \vdash A : U}{\Gamma \vdash A} \qquad \frac{\Gamma, x : A \vdash B}{\Gamma \vdash (\Pi x{:}A)B}$$

The rules for forming elements are

$$\frac{(x : A) \in \Gamma \quad \Gamma \vdash}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : (\Pi x{:}A)B} \qquad \frac{\Gamma \vdash v : (\Pi x{:}A)B \quad \Gamma \vdash u : A}{\Gamma \vdash v\ u : B(u)}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash B \quad A = B}{\Gamma \vdash t : B}$$

We use the notation $B(u)$ to denote B where all free occurrences of x have been replaced by u.

We express that the universe U contains N and N_k, the type of natural numbers $< k$, and is closed under the product operation.

$$\frac{\Gamma \vdash}{\Gamma \vdash N : U} \qquad \frac{\Gamma \vdash}{\Gamma \vdash N_k : U} \qquad \frac{\Gamma \vdash A : U \quad \Gamma, x : A \vdash B : U}{\Gamma \vdash (\Pi x{:}A)B : U}$$

The special constants are natrec, natrec_k with the typing rules

$$\frac{\Gamma \vdash}{\Gamma \vdash 0 : N} \qquad \frac{\Gamma \vdash}{\Gamma \vdash S : N \to N} \qquad \frac{\Gamma \vdash a : B(0) \qquad \Gamma \vdash g : (\Pi x{:}N)\ B(x) \to B(S\ x)}{\Gamma \vdash \mathsf{natrec}\ a\ g : (\Pi x{:}N)B}$$

Thinking of $B(x)$ as a proposition $\mathsf{natrec}\ a\ g$ is a proof of the universal proposition $(\Pi x : N)B(x)$ which we get by applying the principle of *mathematical induction*. In the case $B(x)$ does not depend explicitly on x we get the schema of primitive recursion (at higher types), schema introduced by Hilbert [8] and used later by Gödel [7].

The typing rules for natrec_k are

$$\frac{\Gamma \vdash}{\Gamma \vdash 0 : N_k} \quad \dots \quad \frac{\Gamma \vdash}{\Gamma \vdash k-1 : N_k} \qquad \frac{\Gamma \vdash a_0 : B(0) \ \ \dots \quad \Gamma \vdash a_{k-1} : B(k-1)}{\Gamma \vdash \mathsf{natrec}_k\ a_0\ \dots\ a_{k-1} : (\Pi x{:}N_k)B}$$

The type N_0 represents the empty type and natrec_0 represents a dependent version of the "ex falso quodlibet" rule. The type N_1 represents a true proposition. We can define the equality on N as $eq_N = \mathsf{natrec}\ f\ g$ where f is the equality to zero, defined as the term $\mathsf{natrec}\ N_1\ (\lambda x \lambda y.N_0)$, and g is the term $\lambda x.\lambda y.\mathsf{natrec}\ N_0\ (\lambda n.\lambda z.y\ n)$. We can then check the following β, ι conversions

$$eq_N\ 0\ 0 = N_1 \quad eq_N\ 0\ (S\ y) = N_0 \quad eq_N\ (S\ x)\ 0 = N_0 \quad eq_N\ (S\ x)\ (S\ y) = eq_N\ x\ y$$

Using this, it is direct to translate Heyting arithmetic in this version of type theory. Similarly, we can define for each k an equality eq_{N_k} on each type N_k.

2.3. Possible extensions

We can as well introduce the type Ord, the type of *ordinal numbers* [10]

$$0 : Ord,\ S\ x : Ord\ [x : N],\ L\ u : Ord\ [u : N \to Ord]$$

with the corresponding elimination rule which expresses both the principle of transfinite induction over the second number class ordinals and definition of objects by transfinite recursion. It is quite remarkable that such abstract concepts can be described in this computational framework.

The following comment, extracted from one of the first presentation of type theory [10], expresses the interest of this system for constructive mathematics: "In the formal theory the abstract entities (natural numbers, ordinals, functions, types, and so on) become represented by certain symbol configurations, called *terms*, and the definitional schema, read from the left to the right, become mechanical *reduction rules* for these symbol configurations. Type theory effectuates the computerization of abstract intuitionistic mathematics that above all Bishop has asked for."

2.4. Computability

Type theory extends the system introduced by Gödel [7]. This type system had only one base type N and one forming type operation $A \to B$. For this system, Gödel introduced a notion of *computability* at all types which is defined by induction on the types. This was used later by Tait [13] to show that all well-typed terms are normalizable. We show here how to extend the notion of computability to type theory.

We define $\delta_{N_k}(t)$ to mean $t = j$ for some $j < k$, which is equivalent to the fact that t reduces to j by β, ι reduction, and $\delta_N(t)$ to mean that $t = \overline{k}$ for some numeral k. If δ is a predicate on terms, and $\nu = (\nu_t)$ is a family of predicates on terms indexed by terms, then $\Pi\ \delta\ \nu$ is the predicate such that $(\Pi\ \delta\ \nu)(v)$ holds iff $\delta(t)$ implies $\nu_t(v\ t)$ for all terms t. Finally we say that two predicates $\delta_1,\ \delta_2$ on terms are *extensionally equal*, written $\delta_1 =_{ext} \delta_2$ iff for all terms t, $\delta_1(t)$ iff $\delta_2(t)$.

Following S. Allen [1], we define the relation $R(A, \delta)$ between terms and predicates on terms inductively by the clauses:

- $R(N, \delta_N)$
- $R(N_2, \delta_{N_2})$
- $R((\Pi x{:}A)B, \Pi\ \delta\ \nu)$ whenever $R(A, \delta)$ and $\delta(t)$ implies $R(B(t), \nu_t)$
- if $A_1 = A_2$ and $\delta_1 =_{ext} \delta_2$ and $R(A_1, \delta_1)$ then $R(A_2, \delta_2)$

Lemma 2.1. If $R(A_1, \delta_1)$ and $R(A_2, \delta_2)$ and $A_1 = A_2$ then $\delta_1 =_{ext} \delta_2$.

Proof:
Using the Church-Rosser property of β, ι reduction. □

We define $\psi(A)$ to mean

$$\exists \delta. R(A, \delta)$$

and $\psi_A(t)$ is defined by

$$\exists \delta. R(A, \delta) \wedge \delta(t)$$

or, equivalently, by Lemma 2.1, provided $\psi(A)$ holds

$$\forall \delta. R(A, \delta) \to \delta(t)$$

Using the predicate ψ we define in a similar way another relation $S(A, \delta)$ by the clauses:

- $S(N, \delta_N)$
- $S(N_2, \delta_{N_2})$
- $S(U, \psi)$
- $S((\Pi x{:}A)B, \Pi\ \delta\ \nu)$ whenever $S(A, \delta)$ and $\delta(t)$ implies $S(B(t), \nu_t)$
- if $A_1 = A_2$ and $\delta_1 =_{ext} \delta_2$ and $S(A_1, \delta_1)$ then $S(A_2, \delta_2)$

Lemma 2.2. If $S(A_1, \delta_1)$ and $S(A_2, \delta_2)$ and $A_1 = A_2$ then $\delta_1 =_{ext} \delta_2$.

Proof:
Using the Church-Rosser property of β, ι reduction. □

We define $\varphi(A)$ to mean

$$\exists \delta. S(A, \delta)$$

and $\varphi_A(t)$ is defined by

$$\exists \delta. S(A, \delta) \wedge \delta(t)$$

or, equivalently, by Lemma 2.2, provided $\varphi(A)$ holds

$$\forall \delta. S(A, \delta) \rightarrow \delta(t)$$

Lemma 2.3. If $\psi(A)$ then $\varphi(A)$ and $\varphi_A = \psi_A$.

Proof:
It is direct that $R(A, \delta)$ implies $S(A, \delta)$, hence the result. □

If Γ is a context $x_1{:}A_1, \ldots, x_n{:}A_n$ and $t_1, \ldots, t_n$ a vector of terms we define $\varphi_\Gamma(t_1, \ldots, t_n)$ to mean

$$\varphi(A_1),\ \varphi_{A_1}(t_1),\ \ \ldots,\ \varphi(A_n(t_1, \ldots, t_{n-1})),\ \varphi_{A_n(t_1,\ldots,t_{n-1})}(t_n)$$

Theorem 2.1. If we have $\varphi_\Gamma(t_1, \ldots, t_n)$ then $\varphi(A(t_1, \ldots, t_n))$ whenever $\Gamma \vdash A$ and if we have $\varphi(A(t_1, \ldots, t_n))$ then we have $\varphi_{A(t_1,\ldots,t_n)}(t(t_1, \ldots, t_n))$ whenever $\Gamma \vdash t : A$. In particular, if $\vdash A$ then $\varphi(A)$ and if $\varphi(A)$ and $\vdash t : A$ then $\varphi_A(t)$.

Proof:
The proof is by induction on the derivation of $\Gamma \vdash A$ and $\Gamma \vdash t : A$. □

Corollary 2.1. If $\vdash g : N \rightarrow N_2$ then for any natural number n we have a Boolean b such that $g\,\overline{n} \rightarrow^* b$.

3. Forcing Extension

3.1. Conditions

The conditions $p, q, \ldots$ represent finite amount of information about the infinite object we want to describe[1]. Since we want to force the addition of a Cohen real, the conditions are finite sub-graphs of function from natural numbers to Booleans. Thus the conditions can be represented as a finite list of equations

$$\mathsf{f}\ n_1 = b_1 \quad \ldots \quad \mathsf{f}\ n_k = b_k$$

where $n_1, \ldots, n_k$ are distinct natural numbers and $b_1, \ldots, b_k$ Booleans. The *domain* of this condition is the finite set $n_1, \ldots, n_k$. We write $q \leq p$ if the condition q extends the condition p. If n is not in the domain of p we write $p,\ \mathsf{f}\ n = b$ the extension of the condition p with the equation $\mathsf{f}\ n = b$. If n is not in the domain of p then the two conditions $p,\ \mathsf{f}\ n = 0$ and $p,\ \mathsf{f}\ n = 1$ form an *elementary partition* of p. By iterating this construction, we obtain the general notion of *partition* $p_1, \ldots, p_l$ of a condition p (this includes as well the trivial partition p of p.)

One can think of a condition as a compact open of Cantor space, which is the space of functions from natural numbers to the discrete space of Booleans, with the product topology. A partition $p_1, \ldots, p_l$ of p represents a partition of p in smaller compact opens.

3.2. Terms

We extend the syntax of terms with a new function symbol f. To each condition p we associate the reduction relation $\rightarrow_p$ which extends β, ι reduction with the rule $\mathsf{f}\ \overline{n} \rightarrow_p b$ whenever $\mathsf{f}\ n = b$ is in p. This extension still satisfies the Church-Rosser property, by the usual Martin-Löf/Tait argument (as presented for instance in [11]). We define then $t =_p u$ to mean that t and u have a common reduct for $\rightarrow_p$.

We define next

$$p \Vdash t = u$$

to mean that there is a partition $p_1, \ldots, p_l$ of p such that $t =_{p_i} u$ for all i. For instance, if t is $\mathsf{natrec}_2\ (\mathsf{f}\ 0)\ u\ u$ we have $\Vdash t = u$ because the empty condition admits a partition in two conditions $\mathsf{f}\ 0 = 0$ and $\mathsf{f}\ 0 = 1$ and that we have $t \rightarrow_p u$ for each of these two conditions. We write $\Vdash t = u$ instead of $p \Vdash t = u$ if p is the empty condition.

Lemma 3.1. If $p_1, \ldots, p_l$ is a partition of p and $p_i \Vdash t = u$ for all i then $p \Vdash t = u$.

If $\vdash g : N \rightarrow N_2$ we say that g satisfies the condition p iff we have $g\ \overline{k} = b$ whenever $\mathsf{f}\ k = b$ is in p.

Lemma 3.2. If $p \Vdash t = u$ and $\vdash g : N \rightarrow N_2$ and g satisfies the condition p then $t(\mathsf{f}/g) = u(\mathsf{f}/g)$.

Proof:
We have a partition $p_1, \ldots, p_l$ of p such that $t =_{p_i} u$ for all i. On the other hand, g satisfies exactly one condition p_{i_0}, and $t =_{p_{i_0}} u$ implies $t(\mathsf{f}/g) = u(\mathsf{f}/g)$. $\square$

[1] It may be appropriate to recall some motivations for the notion of forcing, according to R. Platek: "Cohen's original discovery of forcing was motivated by an attempt to prove analysis consistent. The idea was that statements which seemed to involve infinities could be reduced to pieces of finite information" [12]. This is reminiscent of the use of sheaf models and generic elements in constructive mathematics to explain computationally non effective principles.

3.3. Typing rules

The typing rules of the forcing extension of type theory are similar to the one of type theory. The only changes are the equality rule and the typing rule for the constant f. We have the new judgements

$$\Gamma \Vdash_p A \qquad \Gamma \Vdash_p t : A \qquad \Gamma \Vdash_p$$

and we write $\Gamma \Vdash J$ for $\Gamma \Vdash_p J$ if p is the empty condition.

The typing rule for the generic constant f is

$$\frac{\Gamma \Vdash_p}{\Gamma \Vdash_p \mathsf{f} : N \rightarrow N_2}$$

and the rule for equality is

$$\frac{\Gamma \Vdash_p t : A \quad \Gamma \Vdash_p B \quad p \Vdash A = B}{\Gamma \Vdash_p t : B}$$

Otherwise, the other rules are a copy of the rules of type theory, by indexing them with a condition. For instance the rule for elements are

$$\frac{(x : A) \in \Gamma \quad \Gamma \Vdash_p}{\Gamma \Vdash_p x : A} \qquad \frac{\Gamma, x : A \Vdash_p t : B}{\Gamma \Vdash_p \lambda x.t : (\Pi x{:}A)B} \qquad \frac{\Gamma \Vdash_p v : (\Pi x{:}A)B \quad \Gamma \Vdash_p u : A}{\Gamma \Vdash_p v\ u : B(u)}$$

We have directly the fact that these rules define an *extension* of standard type theory.

Proposition 3.1. If $\Gamma \vdash J$ then $\Gamma \Vdash_p J$ for any condition p. Furthermore, if $\Gamma \Vdash_p J$ and $q \leq p$ then $\Gamma \Vdash_q J$. If $p_1, \ldots, p_l$ is a partition of p and $\Gamma \Vdash_{p_i} J$ for all i then $\Gamma \Vdash_p J$.

For instance if $\vdash t : N \rightarrow N$ then we also have $\Vdash t : N \rightarrow N$ without changing t. This is a difference with forcing in *set theory*, where the structure of function spaces is modified by forcing extension. On the other hand, the forcing extension is a conservative extension in the following sense.

Proposition 3.2. If $\Gamma \Vdash_p J$ and $\vdash g : N \rightarrow N_2$ and g satisfies the condition p then $\Gamma(\mathsf{f}/g) \vdash J(\mathsf{f}/g)$. In particular, if Γ, J do not mention f and $\Gamma \Vdash J$ then $\Gamma \vdash J$, and if Γ, A do not mention f and $\Gamma \Vdash_p t : A$ then there exists t' such that $\Gamma \vdash t' : A$. (This term t' is obtained by replacing f by any function satisfying the condition p.)

Proof:
Direct by Lemma 3.2. □

The partition property of Proposition 3.1 is reminiscent of Beth models [4]. Notice however that the rule for implication

$$\frac{\Gamma, x : A \Vdash_p t : B}{\Gamma \Vdash_p \lambda x.t : A \rightarrow B}$$

is different from the implication rule for Beth and Kripke models.

3.4. Computability

We define $p \Vdash \delta_N(t)$ to mean that there is a partition $p_1, \dots, p_l$ of p and numerals $n_1, \dots, n_l$ such that $p_i \Vdash t = \overline{n_i}$ for all i. We get an equivalent definition if we replace $p_i \Vdash t = \overline{n_i}$ by $t =_{p_i} \overline{n_i}$.

Similarly, we define $p \Vdash \delta_{N_k}(t)$ to mean that there is a partition $p_1, \dots, p_l$ of p and elements $j_1, \dots, j_l < k$ such that $p_i \Vdash t = j_i$ for all i.

We define $p \Vdash \delta_1 =_{ext} \delta_2$ to mean that for all $q \leq p$ and all terms t we have $q \Vdash \delta_1(t)$ iff $q \Vdash \delta_2(t)$. (Recall that $q \leq p$ means that q extends p.) We define a relation $p \Vdash R(A, \delta)$ inductively, where δ is a relation $p \Vdash \delta(t)$ between terms and conditions.

- $p \Vdash R(N, \delta_N)$
- $p \Vdash R(N_k, \delta_{N_k})$
- $p \Vdash R((\Pi x{:}A)B, \Pi\ \delta\ \nu)$ whenever $p \Vdash R(A, \delta)$ and $q \leq p$, $q \Vdash \delta(t)$ imply $q \Vdash R(B(t), \nu_t)$
- if $p_1, \dots, p_n$ is a partition of p and $p_i \Vdash A = A_i$ and $p_i \Vdash \delta =_{ext} \delta_i$ and $p_i \Vdash R(A_i, \delta_i)$ for all i then $p \Vdash R(A, \delta)$

Lemma 3.3. If we have $p \Vdash R(A, \delta)$ and $r \leq q \leq p$ then $q \Vdash \delta(u)$ implies $r \Vdash \delta(u)$. Also if $q \leq p$ and $q_1, \dots, q_m$ is a covering of q and $q_i \Vdash \delta(t_i)$ and $q_i \Vdash t = t_i$ for all i then $q \Vdash \delta(t)$.

Lemma 3.4. If we have $p \Vdash R(A_1, \delta_1)$ and $p \Vdash R(A_2, \delta_2)$ and $p \Vdash A_1 = A_2$ then $p \Vdash \delta_1 =_{ext} \delta_2$.

We define $p \Vdash \delta_U(A)$ to mean that there exists a predicate δ such that $p \Vdash R(A, \delta)$. We can then define the relation $p \Vdash S(A, \delta)$ inductively:

- $p \Vdash S(N, \delta_N)$
- $p \Vdash S(N_k, \delta_{N_k})$
- $p \Vdash S(U, \delta_U)$
- $p \Vdash S((\Pi x{:}A)B, \Pi\ \delta\ \nu)$ whenever $p \Vdash S(A, \delta)$ and $q \leq p$, $q \Vdash \delta(t)$ imply $q \Vdash S(B(t), \nu_t)$
- if $p_1, \dots, p_n$ is a partition of p and $p_i \Vdash A = A_i$ and $p_i \Vdash \delta =_{ext} \delta_i$ and $p_i \Vdash S(A_i, \delta_i)$ for all i then $p \Vdash S(A, \delta)$

We define $p \Vdash \varphi(A)$ to mean that there exists δ such that $p \Vdash S(A, \delta)$ and $q \Vdash \varphi_A(t)$ for $q \leq p$ is then defined to mean $q \Vdash \delta(t)$.

If p and q are compatible conditions we define $p \wedge q = p \cup q$. We have the following "gluing" property.

Lemma 3.5. If $p_1, \dots, p_l$ is a partition of p and we have a family of relations $\delta_1, \dots, \delta_l$ then there exists a relation δ such that $p_i \Vdash \delta =_{ext} \delta_i$ for $i = 1, \dots, l$.

Proof:
We define $q \Vdash \delta(t)$ to mean $q \wedge p_i \Vdash \delta_i(t)$ for all i such that q and p_i are compatible. $\square$

Lemma 3.6. If $p_1, \ldots, p_l$ is a partition of p and if we have $p_i \Vdash \varphi_A(t)$ for all i then $p \Vdash \varphi_A(t)$.

Proof:
For each i we have a relation δ_i such that $S(A, \delta_i)$. Using Lemma 3.5, we find δ such that $p_i \Vdash \delta =_{ext} \delta_i$ for all i, and so, $p \Vdash S(A, \delta)$ by the last clause of the inductive definition of S. Hence we have $p \Vdash \varphi(A)$. $\square$

The key Lemma expresses the computability of the generic function f.

Lemma 3.7. We have $\Vdash \varphi_{N \to N_2}(\mathsf{f})$.

Proof:
We remark first that we have $q \Vdash \varphi_{N_2}(\mathsf{f}\ \overline{n})$ for all conditions q and all natural numbers n. Indeed, if $\mathsf{f}\ n = b$ is in q then we have $\mathsf{f}\ \overline{n} =_q b$ and and if n is not in the domain of q then q is covered by two conditions q_0, q_1 such that $\mathsf{f}\ \overline{n} =_{q_i} i$.

If $q \leq p$ and $q \Vdash \varphi_N(t)$ then we have a partition $q_1, \ldots, q_l$ and natural numbers $n_1, \ldots, n_l$ such that $q_i \Vdash t = \overline{n_i}$ for all i. We then have $q_i \Vdash \varphi_{N_2}(\mathsf{f}\ t)$ for all i by the remark above and so $q \Vdash \varphi_{N_2}(\mathsf{f}\ t)$ by Lemma 3.6. $\square$

If Γ is a context $x_1{:}A_1, \ldots, x_n{:}A_n$ and $t_1, \ldots, t_n$ a vector of terms we define $p \Vdash \varphi_\Gamma(t_1, \ldots, t_n)$ to mean

$$p \Vdash \varphi(A_1),\ p \Vdash \varphi_{A_1}(t_1),\ \ \ldots,\ p \Vdash \varphi(A_n(t_1, \ldots, t_{n-1})),\ p \Vdash \varphi_{A_n(t_1,\ldots,t_{n-1})}(t_n)$$

Theorem 3.1. If we have $p \Vdash \varphi_\Gamma(t_1, \ldots, t_n)$ then $p \Vdash \varphi(A(t_1, \ldots, t_n))$ whenever $\Gamma \Vdash_p A$ and if we have $p \Vdash \varphi(A(t_1, \ldots, t_n))$ then we have $p \Vdash \varphi_{A(t_1,\ldots,t_n)}(t(t_1, \ldots, t_n))$ whenever $\Gamma \Vdash_p t : A$. In particular, if $\Vdash A$ then $\Vdash \varphi(A)$ and if $\Vdash t : A$ and $\Vdash \varphi(A)$ then $\Vdash \varphi_A(t)$.

We can now state the uniform continuity of the functional definable in standard type theory.

Theorem 3.2. If $\vdash F : (N \to N_2) \to N_2$ then there exists a partition $p_1, \ldots, p_l$ of the empty condition and Booleans $b_1, \ldots, b_l$ such that $F\ \mathsf{f} \to^*_{p_i} b_i$.

Proof:
Assume $\vdash F : (N \to N_2) \to N_2$. By Proposition 3.1 we have $\Vdash F : (N \to N_2) \to N_2$. By Lemma 3.7 we have $\Vdash \mathsf{f} : N \to N_2$. Hence $\Vdash F\ \mathsf{f} : N_2$. By Theorem 3.1 this implies $\varphi_{N_2}(F\ \mathsf{f})$ and hence $\delta_{N_2}(F\ \mathsf{f})$, so that we have a partition $p_1, \ldots, p_l$ of the empty condition and Booleans $b_1, \ldots, b_l$ such that $F\ \mathsf{f} =_{p_i} b_i$, which by Church-Rosser, implies $F\ \mathsf{f} \to^*_{p_i} b_i$. $\square$

Conclusion

We presented a simple example of forcing extension of intuitionistic type theory, with an application to the proof of uniform continuity of definable functional on Cantor space. For this, we have defined a computability predicate and shown that well-typed terms are computable.

Like in Tait's work [13] it is also possible to use this method to show that well-typed terms and normalizable, and to show then that type-checking for this forcing extension is *decidable*. One remark about our proof of uniform continuity of definable functional is that this proof is carried out in a constructive meta-theory. It is thus possible, and interesting, to extract from it a computation which takes as input a functional $\vdash F : (N \to N_2) \to N_2$, and outputs a partition $p_1, \dots, p_l$ of the empty condition and Booleans $b_1, \dots, b_l$ such that $F\ \mathsf{f} \to^*_{p_i} b_i$.

Using this forcing extension, we can decide if a standard term $\vdash F : (N \to N_2) \to N_2$ is always 1 or not. Indeed, we have a partition $p_1, \dots, p_l$ of the empty condition and Booleans $b_1, \dots, b_l$ such that $F\ \mathsf{f} \to^*_{p_i} b_i$. We can then test if all b_i are equal to 1. By iterating the forcing extension, i.e. by adding infinitely many Cohen reals $\mathsf{f}_0,\ \mathsf{f}_1,\ \mathsf{f}_2, \dots$ we can thus get an extension of type theory with a computable functional

$$\forall : ((N \to N_2) \to N_2) \to N_2$$

Furthermore type-checking for this extension is still decidable. We intend to present all these variations in a following paper.

References

[1] S. Allen. A Non-Type-Theoretic Definition of Martin-Löf's Types. Proceedings of the Second IEE Symposium LICS 1987, 215-224.

[2] H. Barendregt. The impact of the lambda calculus. Bulletin of Symbolic Logic, Volume 3, 1997, 181-215.

[3] M.J. Beeson. *Foundations of Constructive Mathematics.* Springer-Verlag, 1985.

[4] E.W. Beth. *The foundations of mathematics.* North-Holland, Amsterdam, 1965.

[5] L.E.J. Brouwer. Über Definitionsbereiche von Funktionen. Mathematische Annalen, 97:60-75. English translation in van Heijenoort, (1967, 446-463).

[6] P. Cohen. The discovery of forcing. Rocky Mountain J. Math. 32 (2002), 1071-110.

[7] K. Gödel. On a hitherto unexploited extension of the finitary standpoint. in *Collected Works*, Vol. II. Publications 1938-1974, Oxford University Press, 1990.

[8] D. Hilbert. Über das Unendliche. Mathematische Annalen, 95:161-190. Lecture given in Münster, 4 june 1925. English translation in van Heijenoort, (1967, 367-392).

[9] J.L. Krivine. Structures de réalisabilité, RAM et ultrafiltre sur $\mathbb{N}$. To appear, 2010.

[10] P. Martin-Löf. On the strength of intuitionistic reasoning. Unpublished report, talk at the Bucharest conference, 1971.

[11] P. Martin-Löf. An intuitionistic theory of types in *Twenty-Five Years of Type Theory*, G. Sambin and J. Smith Eds., Oxford University Press, 1998 (reprinted version of an unpublished report from 1972).

[12] R. Platek. Generalized Recursion Theory, Stanford and Me. In: Odifreddi, P. (ed.) *Kreiseliana, About and Around Georg Kreisel* (1996).

[13] W.W. Tait. Intensional interpretations of functional of finite types I. Journal of Symbolic Logic 32 (1967), 198-212.

[14] J. van Heijenoort (ed.) *From Frege to Hilbert: A Source Book in Mathematical Logic, 1897-1941.* Harvard University Press, 1967.