Chapter 1
A computational interpretation of forcing in Type Theory

Thierry Coquand and Guilhem Jaber

Abstract In a previous work, we showed the uniform continuity of definable functionals in intuitionistic type theory as an application of forcing with dependent types. The argument was constructive, and so contains implicitly an algorithm which computes a witness that a given functional is uniformly continuous. We present here such an algorithm, which provides a possible computational interpretation of forcing.

Introduction

In a previous work [6], we considered intuitionistic type theory with a type of natural numbers N and a type of Booleans N_2. The type $C = N \to N_2$ represents *Cantor space*, the space of functions from natural numbers to Booleans, and it has a natural topology, with basic compact open subsets defined by a finite set of conditions of the form $f\ n_i = b_i$ about a function $f : C$. We have shown [6] that any definable functional $F : C \to N_2$ is *uniformly continuous*. This means that we can find a partition of Cantor space in a finite number of conditions $p_1, \ldots, p_n$ with corresponding Boolean values $b_1, \ldots, b_n$ such that $F\ f = b_i$ whenever f satisfies the condition p_i. The argument in [6] is constructive, and thus can be seen *implicitly* an algorithm which computes an uniform modulus of continuity. We explicitate here a possible algorithm, which given such a functional F, produces a covering $p_1, \ldots, p_n$ and a list $b_1, \ldots, b_n$. To simplify the presentation, we limit ourselves to a type system which is an extension of Gödel system T [7] with a type of Booleans. This computation can be readily expressed in a functional programming language, here Haskell, using the notion of monads [14].

We briefly outline the paper. We first recall the syntax for terms and conditions. We then give a simple operational semantics corresponding to forcing. We prove the termination of this evaluation, and give an algorithm to

compute the modulus of continuity of a given functional. These computation combines in a non trivial way realizability and Beth models, and we end by commenting on this point, following Goodman [9]. A first appendix presents a representation in the programming language Haskell and a second appendix explains how we can give computational sense to universal quantification over Cantor space by iterating this forcing construction.

1.1 Terms, types and conditions

1.1.1 Terms

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

$$t, u ::= x \mid \lambda x.t \mid t\ t \mid \mathsf{natrec}(t,t) \mid \mathsf{boolrec}(t,t) \mid S(t)$$

We consider terms up to α-conversion. Besides β-reduction, natrec and boolrec have the reduction rules

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

and

$$\mathsf{boolrec}(a_0,a_1)\ 0 \to a_0 \qquad \mathsf{boolrec}(a_0,a_1)\ 1 \to a_1$$

This forms an extension of β-reduction which still has the Church-Rosser property [11], sometimes called β,ι-reduction [1].

If k is a natural number, we write $\overline{k}$ the term $S^k(0)$.

1.1.2 Typing rules

The basic types are N, for natural numbers, and N_k, for finite types with k elements. If $A,\ B$ are types then so is $A \to B$. The typing judgements are of the form $\Gamma \vdash t{:}A$, where Γ is a context $x_1{:}A_1, \ldots, x_n{:}A_n$ (with $x_i \neq x_j$ for $i \neq j$).

The typing rules are as follows.

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

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

$$\frac{}{\Gamma \vdash 0 : N_2} \qquad \frac{}{\Gamma \vdash 1 : N_2} \qquad \frac{\Gamma \vdash a_0 : B \quad \Gamma \vdash a_1 : B}{\Gamma \vdash \mathsf{boolrec}(a_0, a_1) : N_2 \to B}$$

1.1.3 Conditions

The conditions $p, q, \ldots$ represent finite amount of information about the infinite object we want to describe. 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

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

where $n_1, \ldots, n_k$ are distinct natural numbers and $b_1, \ldots, b_k$ Booleans. The *domain* $\mathsf{dom}(p)$ of this condition p is the finite set $n_1, \ldots, n_k$. We write $q \leqslant p$ if the condition q extends the condition p. One can think of a condition p as a compact open subset C_p of Cantor space C, which is the space of functions from natural numbers to the discrete space of Booleans, with the product topology. A condition p represents also some finite amount of information about a generic element of Cantor space. If p and q are compatible conditions, we can consider $pq = qp$, by taking the union of the conditions p and q. We clearly have $C_q \subseteq C_p$ if $q \leqslant p$ and $C_{pq} = C_p \cap C_q$ if p and q are compatible. Any condition p can be considered to be the product of elementary conditions $\mathsf{f}\ n = b$. 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.) In general a non trivial partition $p_i,\ i \in I$ of p is built from one partition $p_i,\ i \in I_0$ of $p(\mathsf{f}\ l = 0)$ and one partition $p_i,\ i \in I_1$ of $p(\mathsf{f}\ l = 1)$ for some l not in the domain of p.

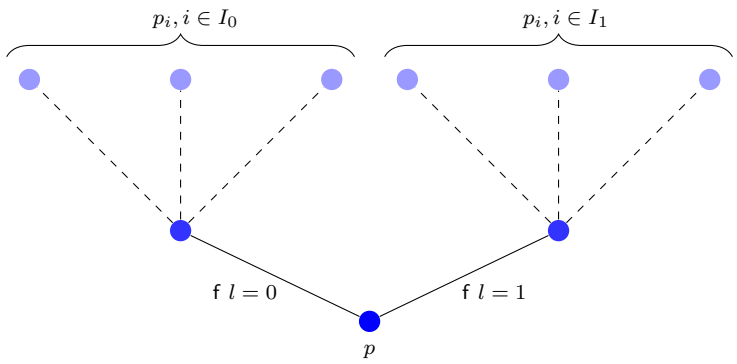

Fig. 1.1 An example of condition

1.1.4 Generic function

We extend the syntax of terms with a new function symbol f. To each condition p we associate the reduction relation $\to_p$ which extends β, ι reduction with the rule $\mathsf{f}\ \overline{n} \to_p b$ 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 $\to_p$.

1.2 Computational interpretation of forcing

1.2.1 Operational semantics

In ordinary type theory, the computation is described by a rewriting relation $t \to t'$ between terms. Here the computation deals with a pair pt of a condition p (which can be thought as a state) and a term t. Furthermore the computation (process) may open during the computation independent computations, and the computation step is a relation $pt \to \alpha$ between pt and a formal sum $\alpha = \Sigma p_i t_i$ where $p_1, \ldots, p_n$ is a partition of p. The definition is the following.

$$\frac{pt \to \Sigma p_i t_i}{p(t\ u) \to \Sigma p_i (t_i\ u)} \qquad \frac{}{p((\lambda x.t)\ u) \to pt[x/u]}$$

$$\frac{}{p(\mathsf{boolrec}(t_0, t_1)\ 0) \to pt_0} \qquad \frac{}{p(\mathsf{boolrec}(t_0, t_1)\ 1) \to pt_1}$$

$$\frac{}{p(\mathsf{natrec}(t_0, t_1)\ 0) \to pt_0} \qquad \frac{}{p(\mathsf{natrec}(t_0, t_1)\ S(t)) \to p(t_1\ t\ (\mathsf{natrec}(t_0, t_1)\ t))}$$

$$\frac{pt \to \Sigma p_i t_i}{p(\mathsf{natrec}(t_0, t_1)\ t) \to \Sigma p_i (\mathsf{natrec}(t_0, t_1)\ t_i)}$$

$$\frac{pt \to \Sigma p_i t_i}{p(\mathsf{boolrec}(t_0, t_1)\ t) \to \Sigma p_i (\mathsf{boolrec}(t_0, t_1)\ t_i)}$$

The remaining crucial rules are that $p(\mathsf{f}\ \overline{k}) \to pb$ if $\mathsf{f}\ k = b$ is in p and otherwise $p(\mathsf{f}\ \overline{k}) \to p_0 0 + p_1 1$ with $p_i = p(\mathsf{f}\ k = i)$. Finally, we have $p(\mathsf{f}\ S^n(t)) \to \Sigma p_i (\mathsf{f}\ S^n(t_i))$ whenever $pt \to \Sigma p_i t_i$.

We can then define the computation of the normal form (for ground types):

$$\frac{}{p0 \Rightarrow p0} \qquad \frac{}{p1 \Rightarrow p1} \qquad \frac{pt \Rightarrow \Sigma p_i \overline{k_i}}{pS(t) \Rightarrow \Sigma p_i \overline{1 + k_1}} \qquad \frac{pt \to \Sigma p_i t_i \qquad p_i t_i \Rightarrow \alpha_i}{pt \Rightarrow \Sigma \alpha_i}$$

Lemma 1. *If $pt \to \Sigma p_i t_i$ or $pt \Rightarrow \Sigma p_i t_i$ then (p_i) is a partition of p and $t \to^*_{p_i} t_i$.*

If $\alpha = \Sigma p_i t_i$ is a formal sum, with (p_i) partition of p, and $q \leqslant p$ we can define $q\alpha = \Sigma(qp_i)t_i$ where we limit the sum to the p_i compatible with q.

Lemma 2. *If $pt \to \alpha$ and $q \leqslant p$ then $qt \to q\alpha$. If $pt \Rightarrow \alpha$ and $q \leqslant p$ then $qt \Rightarrow q\alpha$.*

1.2.2 Computability predicate

We define $p \Vdash \varphi_N(t)$ inductively

- $p \Vdash \varphi_N(0)$
- $p \Vdash \varphi_N(S(t))$ if $p \Vdash \varphi_N(t)$
- $p \Vdash \varphi_N(t)$ if $pt \to \Sigma p_i t_i$ with $p_i \Vdash \varphi_N(t_i)$ for all i

This is equivalent to the fact that we have a relation $t \Rightarrow \Sigma p_i \overline{k_i}$. Similarly $p \Vdash \varphi_{N_2}(t)$ is defined by the clauses

- $p \Vdash \varphi_{N_2}(0)$
- $p \Vdash \varphi_{N_2}(1)$
- $p \Vdash \varphi_{N_2}(t)$ if $pt \to \Sigma p_i t_i$ with $p_i \Vdash \varphi_{N_2}(t_i)$ for all i

and this is equivalent to the fact that $pt \Rightarrow \Sigma p_i v_i$ with $v_i = 0$ or $v_i = 1$ for all i. Finally, $p \Vdash \varphi_{A\to B}(t)$ means that $q \leqslant p$ and $q \Vdash \varphi_A(u)$ implies $q \Vdash \varphi_B(t\ u)$.

$p \Vdash \varphi_A(t)$ can be read as "p forces that t is computable at type A". In the case $A = N$ or $A = N_2$ this means that we have $pt \Rightarrow \alpha$ for some α, i.e. that the computation of pt terminates.

Lemma 3. *If $p \Vdash \varphi_A(t)$ and $q \leqslant p$ then $q \Vdash \varphi_A(t)$.*

Proof. This is direct if A is a function type and follows from Lemma 2 in the case $A = N$ or $A = N_2$.

Lemma 4. *If $pt \to \Sigma p_i t_i$ and $p_i \Vdash \varphi_A(t_i)$ for all i then $p \Vdash \varphi_A(t)$.*

Proof. This is clear if $A = N$ or $A = N_2$. If $A = A_1 \to A_2$ and $pt \to \Sigma p_i t_i$ and $p_i \Vdash \varphi_A(t_i)$ for all i and if $q \leqslant p$ then we have $qt \to \Sigma(qp_i)t_i$ by Lemma 2. If $q \Vdash \varphi_{A_1}(u)$ we have $q(t\ u) \to \Sigma(qp_i)(t_i\ u)$ and $qp_i \Vdash \varphi_{A_2}(t_i\ u)$. By induction we have $q \Vdash \varphi_{A_2}(t\ u)$ as desired.

Lemma 5. *If $p \Vdash \varphi_A(t_0)$ and $p \Vdash \varphi_{N\to A\to A}(t_1)$ then $p \Vdash \varphi_{N\to A}(\mathsf{natrec}(t_0, t_1))$. Similarly, if $p \Vdash \varphi_A(t_0)$ and $p \Vdash \varphi_A(t_1)$ then $p \Vdash \varphi_{N_2\to A}(\mathsf{boolrec}(t_0, t_1))$.*

Proof. This follows from Lemma 4.

Lemma 6. *The generic function is computable, i.e. $p \Vdash \varphi_{N\to N_2}(\mathsf{f})$ for all p.*

Proof. We assume $p \Vdash \varphi_N(t)$ and we prove $p \Vdash \varphi_{N_2}(\mathsf{f}\ t)$. We have $pt \Rightarrow \Sigma p_i \overline{k_i}$ and, using Lemma 4, we are reduced to prove that $p \Vdash \varphi_{N_2}(\mathsf{f}\ \overline{k})$, which is direct, by case if k is in the domain of p or not.

Theorem 1. *If* $x_1{:}A_1, \ldots, x_n{:}A_n \vdash t{:}A$ *and* $p \Vdash \varphi_{A_1}(t_1), \ldots, p \Vdash \varphi_{A_n}(t_n)$ *then we have* $p \Vdash \varphi_A(t[x_1/t_1, \ldots, x_n/t_n])$. *In particular, if* $\vdash t{:}A$ *then* $p \Vdash \varphi_A(t)$ *for all* p.

Proof. By induction on the proof of $x_1{:}A_1, \ldots, x_n{:}A_n \vdash t{:}A$ using Lemmas 5 and 6.

If we have $\vdash F : C \to N_2$ it is possible to use this result and compute a modulus of uniform continuity for F as follows. Using Theorem 1 and Lemma 6, we have $\Vdash \varphi_{N_2}(F\ h)$. Hence we have $F\ h \Rightarrow \Sigma p_i v_i$ with $v_i = 0$ or $v_i = 1$ for all i, and p_i is a partition of Cantor space. By Lemma 1, we have $F\ h \to^*_{p_i} v_i$. We can see the modulus of continuity of F as the greatest k such that a condition of the form $\mathsf{f}\ k = b$ appears in one of the p_i.

1.2.3 Baire space

Our argument can be adapted to the case of Baire space $N \to N$ instead of Cantor space. The generic function f is now of type $N \to N$ and an elementary condition is of the form $\mathsf{f}\ n = m$, where n and m are natural numbers. The partitions are not finite objects anymore but well-founded trees. The inductive definition of partition is the following: the condition p itself is a (trivial) partition of p, and if n is not in the domain of p, and for each m we have a partition P_m of $p(\mathsf{f}\ n = m)$, then the union of all P_m is a partition of p. Similarly the formal sums $\Sigma p_i t_i$ are now indexed by well-founded trees: we have the formal sum pt over p, and if if n is not in the domain of p, and for each m we have a formal sum σ_m over $p(\mathsf{f}\ n = m)$, then the formal sum $\Sigma_m \sigma_m$ is a formal sum over p. The operational semantics have the same rules, except that $p(\mathsf{f}\ \overline{k}) \to p\overline{l}$ if $\mathsf{f}\ k = l$ is in p and $p(\mathsf{f}\ \overline{k}) \to \Sigma p_n \overline{n}$ with $p_n = p(\mathsf{f}\ k = n)$ otherwise. Whenever $\vdash F : (N \to N) \to N$ it is possible in this way to associate to F a well-founded tree (a *bar* on Baire space) with natural numbers at each leaves, by computing $F\ \mathsf{f}$. This gives a strong form of the continuity of definable functionals on Baire space[1].

Conclusion

In the reference [9], Goodman compares recursive realizability and Kripke/Beth models as follows. Recursive realizability "emphasizes the active aspect of constructive mathematics... However, Kleene's notion has the weakness that

[1] This result is stated for instance in the reference [3]. In this reference, Bishop argues that an appropriate approach to Brouwer's theory of choice sequence is to express them as part of the metatheory of a system similar to Gödel System T.

it disregards that aspect of constructive mathematics which concern epistemological change... Precisely that aspect of constructive mathematics which Kleene's notion neglects is emphasized by Kripke's semantics for intuitionistic logic... However, Kripke's notion makes it appear that the constructive mathematician is a passive observer of a structure which gradually reveals itself. What is lacking is the emphasis on the mathematician as active which Kleene's notion provides." He then presents a combination of realizability and Kripke semantics. We think that our work illustrates these remarks in a simple and concrete framework. Usual computation rules in type theory, with a rewriting relation on terms, don't involve "epistemological change". In our framework, the condition p represents a state of knowledge. While in usual Kripke/Beth semantics, these states of knowledge are independent of the computations, they are here needed in the computation, and the computation may create new states of knoweldge.

Appendix 1: Representation in Haskell

The operational semantics given in the previous section has a natural representation in the programming language Haskell, using the notion of monad [14]. Written in this way, the program is quite close to an ordinary evaluation program for Gödel system T by head reduction. The monad we use is a composition of the list monad (for nondeterminism) and of the state monad [14].

```
type Name = String

data Exp =
   Zero | One | Succ Exp | App Exp Exp | Natrec Exp Exp
 | Boolrec Exp Exp | Lam Name Exp | Var Name | Gen

-- closed substitution

subst :: Exp -> Name -> Exp -> Exp

subst t x e = case t of
 Var y -> if x == y then e else t
 Lam y t1 -> if x == y then t else Lam y (subst t1 x e)
 App t1 t2 -> App (subst t1 x e) (subst t2 x e)
 Natrec t1 t2 -> Natrec (subst t1 x e) (subst t2 x e)
 Boolrec t1 t2 -> Boolrec (subst t1 x e) (subst t2 x e)
 Succ t1 -> Succ (subst t1 x e)
 _ -> t

type Cond = [(Int,Exp)]     -- uses only Zero or One
```

```
newtype M a = M (Cond -> [(Cond,a)])

app :: M a -> Cond -> [(Cond,a)]
app (M f) p = f p

instance Monad M where
 return x = M (\p -> [(p,x)])
 l >>= k = M (\p -> concat (map (\(p,a) -> app (k a) p)
                                (app l p)))

-- split determines if the condition p contains the value in k,
-- and otherwise forks between the two possibilities

split :: Int -> M Exp

split k = M (\ p ->  case lookup k p of
                     Just b -> [(p,b)]
                     Nothing -> [((k,Zero):p,Zero),
                                 ((k,One):p,One)])

-- gen k e computes e before applying it to split

gen :: Int -> Exp -> M Exp

gen k Zero = split k
gen k (Succ e) = gen (k+1) e
gen k e = do e' <- step e
             gen k e'

-- step implements the reduction

step :: Exp -> M Exp

step (App (Lam x t) u) = return (subst t x u)
step (App (Natrec t0 t1) Zero) = return t0
step (App (Natrec t0 t1) (Succ t)) =
  return (App (App t1 t) (App (Natrec t0 t1) t))
step (App (Boolrec t0 t1) Zero) = return t0
step (App (Boolrec t0 t1) One) = return t1
step (App (Natrec t0 t1) t) =
  do t' <- step t
     return (App (Natrec t0 t1) t')
step (App (Boolrec t0 t1) t) =
  do t' <- step t
```

```
     return (App (Boolrec t0 t1) t')
step (App Gen u) = gen 0 u
step (App t u) = do t' <- step t
                    return (App t' u)
step t = error("step " ++ show t)

-- app (eval t) [] outputs a covering of
-- Cantor space if t is of type N2

eval :: Exp -> M Exp

eval Zero = return Zero
eval One = return One
eval t = do t' <- step t
            eval t'
```

Appendix 2: Quantification on Cantor space

New conditions

We explain how one can use this operational interpretation of forcing to give a new computational interpretation of an universal quantification $\forall : (C \to N_2) \to N_2$ on Cantor space. There are already computational interpretations [8, 12], using a general recursive program[2]. The interpretation we suggest relies on iterating the previous construction and introducing infinitely generic functions $\mathsf{f}_0, \mathsf{f}_1, \ldots$ It is reminiscent of iterated forcing in set theory, and of the interpretation of choice sequences in intuitionism [13].

The first step is to extend the notion of condition. So far, a condition p represents a compact open subset of Cantor space. We can in the same way consider conditions $r, s, \ldots$ which represent compact open subsets of the product space $C^{\mathbb{N}}$. The elementary conditions are now of the form $\mathsf{f}_l\ k = i$, given an information about the generic function f_l, and a condition r is a finite product of compatible elementary conditions. The set of conditions P is the union of the sets P_n of condition containing only $\mathsf{f}_l\ k = i$ with $l < n$. The conditions we need $p, q, \ldots$ are pairs $p = (r, n)$, with r in P_n. Such a condition represents a compact open subset X of C^n. We define $(s, m) \leqslant (r, n)$ to mean $n \leqslant m$ and $s \leqslant r$. To summarize, each condition $p = (r, n)$ represents a finite amount of information about a finite number of generic functions, and to refine this condition we can either add new informations, or add a new

[2] The termination of this program relies on classical logic and the fact that definable functionals are continuous.

generic function. (Intuitively, the conditions represent compact open subsets of a "variable" space.)

The reduction relations $pt \to \alpha,\ pt \Rightarrow \alpha$ are as before, with $p = (n, r)$, and t a term which may contain $\mathsf{f}_0, \ldots, \mathsf{f}_{n-1}$ and α is now a formal sum $\Sigma p_i t_i$ where $p_i = (n, r_i)$ and (r_i) is a partition of r.

Universal quantification as projection

An element r of P_n represents a compact open subset X of C^n. A formal sum of Booleans $\alpha = \Sigma p_i v_i$ with $p_i = (n, r_i)$ and r_i partition of r represents a continuous function f_α from X to the discrete space N_2.

We define the conjunction operation on formal sums of Booleans $\alpha \wedge \beta$ as

$$(\Sigma p_i v_i) \wedge (\Sigma q_j w_j) = \Sigma p_i q_j (v_i \wedge w_j)$$

in such a way that we have $f_{\alpha\wedge\beta} = f_\alpha \wedge f_\beta$.

If r is a condition in P_{n+1}, we can write $r = r's$ with r' in P_n and s a product of conditions of the form $\mathsf{f}_n\ k = i$. The condition $(n+1, r)$ can thus be thought as representing a product $X \times Y$, with $X \subseteq C^n$ corresponding to the condition (n, r') and Y corresponding to s. If we consider a partition (r_i) of r in P_{n+1}, the formal sum $\alpha = \Sigma p_i v_i$, with $p_i = (n+1, r_i)$ represents a continuous function $f_\alpha : X \times Y \to N_2$. We are going to define the formal sum $\mathsf{p}(\alpha) = \Sigma(n, s_j) w_j$ which represents the function $f_{\mathsf{p}(\alpha)} : X \to N_2$ such that $f_{\mathsf{p}(\alpha)}(x) = 1$ iff $f_\alpha(x, y) = 1$ for all y in Y.

This definition is by induction on the fact that (r_i) is a partition of r. If (r_i) is the unit partition then we take $\mathsf{p}((n+1, r)v) = (n, r')v$. If it is a partition formed of a partition $(r_i,\ i \in I_0)$ of $r(\mathsf{f}_l\ k = 0)$ and a partition $(r_i,\ i \in I_1)$ of $r(\mathsf{f}_l\ k = 1)$, we can consider by induction

$$\beta_0 = \mathsf{p}(\Sigma_{i\in I_0} p_i v_i) \qquad \beta_1 = \mathsf{p}(\Sigma_{i\in I_1} p_i v_i)$$

If $l = n$, we define $\mathsf{p}(\alpha) = \beta_0 \wedge \beta_1$ and if $l < n$, we define $\mathsf{p}(\alpha) = \beta_0 + \beta_1$.

Computation rules

The only new reduction rule is the following

$$\frac{(n+1, r)(F\ \mathsf{f}_n) \Rightarrow \alpha}{(n, r)(\forall\ F) \to \mathsf{p}(\alpha)}$$

The intuition is that we want to compute $\forall\ F$ and we know that F mentions only the generic functions $\mathsf{f}_0,\ \ldots,\ \mathsf{f}_{n-1}$, satisfying the condition r. We

compute then $F\ \mathsf{f}_n$, where f_n is "fresh" for F, and from the result of this computation we can compute $\forall\ F$ using the function p.

The computability relation $p \Vdash \varphi_A(t)$ is defined as before, for $p = (n, r)$ and t a term which may contain $\mathsf{f}_0, \ldots, \mathsf{f}_{n-1}$.

Lemma 7. *All constant f_l are computable, i.e. $(n, 1) \Vdash \varphi_C(\mathsf{f}_l)$ if $l < n$. The constant $\forall$ is computable, i.e. $\Vdash \varphi_{C\to N_2}(\forall)$.*

Proof. The proof that f_l is computable is the same as the proof of Lemma 6.

If we have $(n, r) \Vdash \varphi_{C\to N_2}(F)$ we show that $(n, r) \Vdash \varphi_{N_2}(\forall\ F)$. For this it is enough to show that $(n + 1, r) \Vdash \varphi_{N_2}(F\ \mathsf{f}_n)$, which follows from $(n, r) \Vdash \varphi_{C\to N_2}(F)$ and $(n + 1, r) \Vdash \varphi_C(\mathsf{f}_n)$.

References

1. H. Barendregt. The impact of the lambda calculus. Bulletin of Symbolic Logic, Volume 3, 1997, 181-215.
2. M.J. Beeson. *Foundations of Constructive Mathematics.* Springer-Verlag, 1985.
3. E. Bishop. Mathematics as a numerical language. In *Intuitionism and Proof Theory*, A. Kino, J. Myhill, R.E. Vesley Eds, North-Holland, 1970.
4. L.E.J. Brouwer. Über Definitionsbereiche von Funktionen. Mathematische Annalen, 97:60-75. English translation in van Heijenoort, (1967, 446-463).
5. P. Cohen. The discovery of forcing. Rocky Mountain J. Math. 32 (2002), 1071-110.
6. Th. Coquand and G. Jaber. A note on forcing in Type Theory. to appear in Fundamenta Informatica, 2010.
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. M. Escardo. Infinite sets that admit fast exhaustive search. LICS 2007, 443-452.
9. N. Goodman Relativised realizability in intuitionistic arithmetic at all finite types. J. Symbolic Logic 43 (1978), 23-44.
10. J.L. Krivine. Structures de réalisabilité, RAM et ultrafiltre sur $\mathbb{N}$. To appear, 2010.
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. A. Simpson. Lazy Functional Algorithms for Exact Real Functionals. in *Mathematical Foundations of Computer Science 1998*, LNCS 1450, 456-464, 1998.
13. A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics, vol. II.* North-Holland, Amsterdam, 1988.
14. Ph. Wadler. The essence of functional programming. Conference Record of the Nineteeth Annual Symposium of Principle of Programming Languages, 1992.
15. J. van Heijenoort (ed.) *From Frege to Hilbert: A Source Book in Mathematical Logic, 1897-1941.* Harvard University Press, 1967.