

# CONTINUOUS SEMANTICS FOR TERMINATION PROOFS

ULRICH BERGER

DRAFT, July 7, 2004

**Abstract.** We prove a general strong normalization theorem for higher type rewrite systems based on Tait's strong computability predicates and a strictly continuous domain-theoretic semantics. The theorem applies to extensions of Gödel's system  $T$ , but also to various forms of bar recursion for which strong normalization was hitherto unknown.

## §1. Introduction.

**§2. Extended Gödel's system  $T$ .** We introduce extensions of Gödel's system  $T$  by strictly positive inductive types in way which is convenient for our purposes.

A *type system* consists of a set  $\mathcal{T}$  of *types*, a family  $\mathcal{E}$  of *type equations*, and a function  $\text{lev}$  from  $\mathcal{T}$  to some wellordering such that the following three conditions are satisfied. (1)  $\mathcal{T}$  must be a set of (finite) expressions which is closed under *function types*, i.e. if  $\rho, \sigma \in \mathcal{T}$ , then  $\rho \rightarrow \sigma \in \mathcal{T}$ . Types that are not of the form  $\rho \rightarrow \sigma$  are called *inductive types*. (2)  $\mathcal{E}$  assigns to every inductive type a *defining equation*

$$\iota = \text{co}_1(\vec{\rho}_1) \mid \dots \mid \text{co}_n(\vec{\rho}_n)$$

where each  $\vec{\rho}_i$  is a finite list of types, and each  $\text{co}_i$  is a symbol called *constructor* for  $\iota$  with *argument types*  $\vec{\rho}_i$ . Each constructor must occur only once in  $\mathcal{E}$ . (3) The function  $\text{lev}$  must be such that  $\text{lev}(\rho \rightarrow \sigma) > \text{lev}(\rho)$ ,  $\text{lev}(\rho \rightarrow \sigma) \geq \text{lev}(\sigma)$ , and for every inductive type  $\iota$  the *stratification condition* holds, i.e.  $\text{lev}(\iota) \geq \text{lev}(\rho)$  for every argument type  $\rho$  of a constructor of  $\iota$ .

---

Supported by EPSRC

The set of *inhabited types* is defined inductively as follows: A type  $\rho \rightarrow \sigma$  is inhabited if  $\sigma$  is inhabited. An inductive type  $\iota$  is inhabited if it has at least one constructor  $\mathbf{co}$  such that all argument types of  $\mathbf{co}$  are inhabited. A type system is called inhabited if all its types are.

The intuitive meaning of an inductive type is the same as that of a type introduced in e.g. Haskell by the keyword `data`. So, the bar means a disjoint sum and  $\mathbf{co}_i \vec{\rho}_i$  means a cartesian product. The stratification condition ensures that inductive types can be understood as simultaneously defined sets of wellfounded trees. Inhabited types are those containing at least one element. A precise definition of the semantics of types as the least solution to a domain equation will be given later.

As an example let us consider the set of types  $\mathcal{T}$  generated from the symbols `boole`, `nat`, `ftree`, `ctree` (for the booleans, the natural numbers and finitely and countably branching trees) by closure under sums,  $\rho + \sigma$ , products,  $\rho \times \sigma$ , finite lists,  $\rho^*$ , and function spaces,  $\rho \rightarrow \sigma$ . The defining equation for inductive types are (slightly incorrectly we overload the constructors `0` and `S`):

$$\begin{aligned} \mathbf{boole} &= \mathbf{T} \mid \mathbf{F} \\ \mathbf{nat} &= \mathbf{0} \mid \mathbf{S}(\mathbf{nat}) \\ \mathbf{ftree} &= \mathbf{branch}(\mathbf{ftree}^*) \\ \mathbf{ctree} &= \mathbf{0} \mid \mathbf{S}(\mathbf{ctree}) \mid \mathbf{lim}(\mathbf{nat} \rightarrow \mathbf{ctree}) \\ \rho + \sigma &= \mathbf{inleft}(\rho) \mid \mathbf{inright}(\sigma) \\ \rho \times \sigma &= \mathbf{\pi}(\rho, \sigma) \\ \rho^* &= \mathbf{\square} \mid \mathbf{cons}(\rho, \rho^*) \end{aligned}$$

The function `lev` is defined by  $\mathbf{lev}(\mathbf{boole}) = \mathbf{lev}(\mathbf{nat}) = \mathbf{lev}(\mathbf{ftree}) = 0$ ,  $\mathbf{lev}(\mathbf{ctree}) = 1$ ,  $\mathbf{lev}(\rho + \sigma) = \mathbf{lev}(\rho \times \sigma) = \max(\mathbf{lev}(\rho), \mathbf{lev}(\sigma))$ ,  $\mathbf{lev}(\rho^*) = \mathbf{lev}(\rho)$ ,  $\mathbf{lev}(\rho \rightarrow \sigma) = \max(\mathbf{lev}(\rho) + 1, \mathbf{lev}(\sigma))$ . Clearly the stratification condition is satisfied and the system is inhabited.

In the rest of this paper we will always assume that we are give an inhabited type system.

As usual we write  $\vec{\rho} \rightarrow \sigma$  for  $\rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow \sigma$  where  $\rightarrow$  associates to the right. Each type  $\rho$  can be uniquely written in the form  $\rho = \vec{\rho} \rightarrow \iota$  where  $\mathbf{lev}(\rho_i) < \mathbf{lev}(\rho)$  and  $\iota$  is an inductive type with  $\mathbf{lev}(\iota) \leq \mathbf{lev}(\rho)$ . Using the fact that types are finite expressions we can also define the *height* of a type recursively by  $\mathbf{ht}(\iota) = 0$  for inductive types

$\iota$  and  $\text{ht}(\rho \rightarrow \sigma) = \max(\text{ht}(\rho), \text{ht}(\sigma)) + 1$ . Finally we define the *size* of a type by  $\text{size}(\rho) := (\text{lev}(\rho), \text{ht}(\rho))$  and we consider the pairs  $(\text{lev}(\rho), \text{ht}(\rho))$  to be ordered lexicographically. Note that  $\text{size}(\rho) < \text{size}(\rho \rightarrow \sigma)$ ,  $\text{size}(\sigma) < \text{size}(\rho \rightarrow \sigma)$ , and if  $\vec{\sigma} \rightarrow \iota'$  is an argument type of a constructor of  $\iota$ , then  $\text{size}(\sigma_i) < \text{size}(\iota)$  and  $\text{size}(\iota') \leq \text{size}(\iota)$ .

The term language (over a given type system) is determined by a set  $\mathcal{C}$  of *typed constants*  $c^\rho$ . *Typed terms* are constructed from *typed variables*,  $x^\rho$ , and constants,  $c^\rho$ , by abstraction,  $(\lambda x^\rho M^\sigma)^{\rho \rightarrow \sigma}$ , application,  $(M^{\rho \rightarrow \sigma} N^\rho)^\sigma$ , and *constructor term* formation,  $\text{co}(\vec{M}^{\vec{\rho}})^\iota$ , for each  $\text{co}(\vec{\rho})$  occurring in the defining equation of the inductive type  $\iota$ . Type information will often be omitted provided this doesn't cause ambiguities. Instead of  $M^\rho$  we will sometimes write  $M : \rho$ . In a constructor term  $\text{co}(\vec{M})^\iota$  the terms  $\vec{M}$  are called *arguments*.

LEMMA 2.1. *For every type  $\rho$  (of an inhabited type system) there exists a closed term of type  $\rho$  without constants.*

PROOF. Induction along the definition of being inhabited. ⊢

We select for each type  $\rho$  a term without constants and denote it  $0^\rho$ .

$\beta$ -conversion is defined as usual by

$$(\lambda x M)N \mapsto M[N/x]$$

where by  $M[N/x]$  we mean the usual substitution of every free occurrence of  $x$  in  $M$  by  $N$  renaming bound variables in  $M$  if necessary. More general we will consider substitutions  $\theta$ , which are mappings from (all) variables to terms of the same type, and define  $M\theta$  as the simultaneous replacement in  $M$  of  $x$  by  $\theta(x)$  renaming bound variables in  $M$  if necessary.

The operational meaning of a constant  $c \in \mathcal{C}$  of type  $\rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow \sigma$  is determined by *constant-conversion rules* of the form

$$c L_1^{\rho_1} \dots L_n^{\rho_n} \mapsto R^\sigma$$

Consider, for example, the constants  $\text{if} : \text{boole} \rightarrow \rho \rightarrow \rho \rightarrow \rho$ ,  $< : \text{nat} \rightarrow \text{nat} \rightarrow \text{boole}$ ,  $\text{lh} : \rho^* \rightarrow \text{nat}$ ,  $\text{get} : \rho^* \rightarrow \text{nat} \rightarrow \rho$ , and  $++ : \rho^* \rightarrow \rho^* \rightarrow \rho^*$ ,

with the rules

$$\begin{aligned}
& \text{if } T \ x \ y \mapsto x \\
& \text{if } F \ x \ y \mapsto y \\
& n < 0 \mapsto F \\
& 0 < S(m) \mapsto T \\
& S(n) < S(m) \mapsto n < m \\
& \text{lh } [] \mapsto 0 \\
& \text{lh } \text{cons}(x, s) \mapsto S(\text{lh}(s)) \\
& \text{get } [] \ n \mapsto 0^\rho \\
& \text{get } \text{cons}(x, s) \ 0 \mapsto x \\
& \text{get } \text{cons}(x, s) \ S(n) \mapsto \text{get } s \ n \\
& [] \ ++ \ t \mapsto t \\
& \text{cons}(x, s) \ ++ \ t \mapsto \text{cons}(x, s \ ++ \ t)
\end{aligned}$$

The conversion rules above are all examples of definition by *primitive recursion in higher types*. Gödel's system  $T$  in its original form summarizes this definition pattern by constants for primitive recursion

$$R_{\text{nat}, \rho} : \rho \rightarrow (\text{nat} \rightarrow \rho \rightarrow \rho) \rightarrow \text{nat} \rightarrow \rho$$

with the rules

$$\begin{aligned}
R_{\text{nat}, \rho} xy 0 & \mapsto x \\
R_{\text{nat}, \rho} xy S(z) & \mapsto yz(R_{\text{nat}, \rho} xyz)
\end{aligned}$$

Similar conversion rules for recursion constants  $R_{\iota, \rho}$  can be introduced for any inductive type  $\iota$ .

In sections 5 and 6 we will also consider constants with rules that cannot be derived from primitive recursion.

By a *conversion* we mean a  $\beta$ -conversion or an instance of a constant-conversion rule, i.e.  $L\theta \mapsto R\theta$  for some constant-conversion rule  $L \mapsto R$  and substitution  $\theta$ . We write  $M \rightarrow_1 N$  if  $N$  is obtained from  $M$  by replacing one subterm occurrence of the left hand side of a conversion by its right hand side. We call a term  $M$  *strongly normalizing*,  $\text{SN}(M)$ , if  $M$  is in the accessible part of the relation  $\rightarrow_1$ , i.e. there is no infinite

reduction sequence  $M \rightarrow_1 M_1 \rightarrow_1 M_2 \rightarrow_1 \dots$ . We prefer the (classically equivalent) definition of **SN** as a predicate inductively defined by the rule

$$\frac{\forall P (M \rightarrow_1 P \rightarrow \text{SN}(P))}{\text{SN}(M)}$$

We call a system of conversion rules  $\mathcal{R}$  strongly normalizing if every term is strongly normalizing with respect to  $\mathcal{R}$ .

It is well-known that Gödel's system  $T$ , i.e. the system of conversion rules for primitive recursion in finite type is strongly normalizing. In the next section we will reexamine the proof of this fact using Tait's strong computability predicates and generalize it so as to accommodate further constants and conversions.

### §3. Proving strong normalization using strong computability.

We define for every type  $\rho$  what it means for a term  $M^\rho$  to be *strongly computable*,  $\text{SC}_\rho(M)$ . The definition is by recursion on  $\text{size}(\rho)$ . For all inductive types  $\iota$  of the same size (or level)  $\text{SC}_\iota$  is given by a simultaneous inductive definition, whereas  $\text{SC}_{\rho \rightarrow \sigma}$  is defined explicitly from  $\text{SC}_\rho$  and  $\text{SC}_\sigma$ . For an inductive type  $\iota$  the predicate  $\text{SC}_\iota$  is defined by two groups of rules, constructor rules for each constructor  $\text{co}$  and a reduction rule.

$$\begin{array}{l} \text{Cons} \quad \frac{\forall \vec{N} (\text{SC}_{\vec{\sigma}_i}(\vec{N}) \rightarrow \text{SC}_{\iota_i}(M_i \vec{N})) \quad (i = 1, \dots, n)}{\text{SC}_\iota(\text{co}(M_1^{\vec{\sigma}_1 \rightarrow \iota_1}, \dots, M_n^{\vec{\sigma}_n \rightarrow \iota_n}))} \\ \\ \text{Red} \quad \frac{\forall P (M \rightarrow_1 P \rightarrow \text{SC}_\iota(P))}{\text{SC}_\iota(M)} \quad (M \text{ not a constructor term}) \end{array}$$

where  $\text{SC}_{\vec{\sigma}}(\vec{N})$  is shorthand for  $\text{SC}_{\sigma_1}(N_1) \wedge \dots \wedge \text{SC}_{\sigma_k}(N_k)$ . For function types we define

$$\text{SC}_{\rho \rightarrow \sigma}(M) \equiv \forall N (\text{SC}_\rho(N) \rightarrow \text{SC}_\sigma(MN))$$

LEMMA 3.1. (a) *If  $\text{SC}_\rho(M)$  and  $M \rightarrow_1 M'$ , then  $\text{SC}_\rho(M')$ .*

(b) *A constructor term is strongly computable iff all its arguments are.*

PROOF. (a) Easy induction on  $\text{lev}(\rho)$ . If  $\rho$  is an inductive type the assertion is proved by a side induction on the definition of  $\text{SC}_\rho$ . For function types we use the (main) induction hypothesis.

(b) Obvious. ⊣

- LEMMA 3.2. (a)  $\text{SC}_\rho(M) \rightarrow \text{SN}(M)$ .  
 (b)  $\text{SC}_\rho(x)$  for every variable  $x$  of type  $\rho$ .

PROOF. Induction on  $\text{size}(\rho)$ . In order to get the proof through we need to strengthen part (b) to

(b')  $\text{SN}(A) \rightarrow \text{SC}_\rho(A)$  for every term  $A$  with ‘variable head’,

where terms with variable head are variables and terms of the form  $AM$  where  $A$  is a term with variable head.

(a) If  $\rho$  is an inductive type, then the implication follows easily by induction on the definition of  $\text{SC}_\rho(M)$ . *Case*  $\rho \rightarrow \sigma$ . Assume  $\text{SC}_{\rho \rightarrow \sigma}(M)$ . By i.h. (b') we have  $\text{SC}(x^\rho)$ . Hence  $\text{SC}_\sigma(Mx)$ . By i.h. (a),  $\text{SN}(Mx)$ . Hence  $\text{SN}(M)$ .

(b') Let  $A$  be a strongly normalizing term with variable head. If  $A$  has an inductive type, then we show  $\text{SC}(A)$  by a side induction on  $\text{SN}(A)$ . By rule Red it suffices to show  $\text{SC}(B)$  for all one step reducts  $B$  of  $A$ . Clearly  $B$  has variable head, hence  $\text{SC}(B)$  by side induction hypothesis. If  $A$  has type  $\rho \rightarrow \sigma$ , we assume  $\text{SC}_\rho(M)$  and have to show  $\text{SC}_\sigma(AM)$ . By induction hypothesis (a) we have  $\text{SN}(M)$ . Hence  $\text{SN}(AM)$  (one easily proves  $\text{SN}(A) \wedge \text{SN}(M) \rightarrow \text{SN}(AM)$  for terms  $A^{\rho \rightarrow \sigma}$  with variable head, since a reduction of  $AM$  can only take place in  $A$  or in  $M$  and any reduct of  $A$  has variable head). Hence  $\text{SC}(AM)$ , by induction hypothesis (b').  $\dashv$

We call a term *reactive* if it is an abstraction, or of the form  $(cL_1 \dots L_k)\theta$  for some conversion rule  $cL_1 \dots L_n \mapsto R$  with  $n > k$  and some substitution  $\theta$ . The property of a term  $M$  to be *neutral* is defined by recursion on  $M$ . If  $M$  is not a constructor term, then  $M$  is neutral if  $M$  is not reactive. If  $M$  is a constructor term, then  $M$  is neutral iff all its arguments are neutral. Clearly, if  $M^{\rho \rightarrow \sigma}$  is neutral, then for any term  $N^\rho$  the term  $MN$  is again neutral, and any one step reduction of  $MN$  must happen by converting either  $M$  or  $N$ . However, neutral terms are *not* closed under one step reduction.

LEMMA 3.3. *A neutral term is strongly computable iff all its one step reducts are.*

PROOF. Because of lemma 3.1 (a) it suffices to show that a neutral term  $M$  is strongly computable provided all of its one step reducts are. The proof is by induction on the size of the type of  $M$ . For inductive types  $\iota$  the assertion is proved by a side induction on neutral terms of type  $\iota$ . If  $M^\iota$  is not a constructor term, then the assertion holds by definition

of  $\text{SC}_\iota$ . Now let  $M^\iota = \text{co}(\vec{M})$ . We assume that all one step reducts of  $M$  are strongly computable. It follows, by lemma 3.1 (b), that all one steps reducts of the arguments  $M_i$  are strongly computable. Let  $M_i$  be of type  $\vec{\sigma} \rightarrow \iota$  and assume  $\text{SC}_{\vec{\sigma}}(\vec{N})$ . We have to show  $\text{SC}(M_i\vec{N})$ . If the vector  $\vec{N}$  is empty, then  $M_i$  is a neutral subterm of  $M$  of type  $\iota$ . Hence the side induction hypothesis applies. If  $\vec{N}$  is nonempty, we argue by a second side induction on  $\text{SN}(\vec{N})$  (using lemma 3.2 (a)). Since in this case the term  $M_i\vec{N}$  is not a constructor term, it suffices to show that all its one step reducts are strongly computable. Since  $M_i$  is neutral, either  $M_i$  or one of the  $N_k$  must be reduced. In the first case we are done since all one step reducts of  $M_i$  are strongly computable. In the second case we apply the second side induction hypothesis. Let finally  $M$  be of type  $\rho \rightarrow \sigma$ . We show  $\text{SC}_\sigma(MN)$  for all strongly computable terms  $N$  by a side induction on  $\text{SN}(N)$ . Since the term  $MN$  is neutral it suffices, by the main induction hypothesis, to show the strong computability of all its one step reducts. If  $M$  is reduced, we are done by assumption on  $M$ , if  $N$  is reduced, we use the side induction hypothesis.  $\dashv$

LEMMA 3.4. *If  $M[N/x]$  is strongly computable for all strongly computable terms  $N$ , then  $\lambda x M$  is strongly computable.*

PROOF. Let  $M^{\rho \rightarrow \sigma}$  fulfill the assumption of part (a) and assume  $\text{SC}_\rho(N)$ . We have to show  $\text{SC}_\sigma((\lambda x M)N)$ . Since the latter term is neutral it suffices to show that all its one step reducts are strongly computable. By lemma 3.2 (a) and lemma 3.1 (a) we may argue by induction on  $\text{SN}(M, N)$ . Assume  $(\lambda x M)N \rightarrow_1 P$ . If the conversion has happened within  $M$  or  $N$ , then we may use the induction hypothesis. If not, then we must have  $P = M[N/x]$  which is strongly computable by assumption.  $\dashv$

PROPOSITION 3.5. *A term containing only strongly computable constants is strongly normalizable.*

PROOF. By induction on terms  $M$  containing only strongly computable constants we show that  $M\theta$  is strongly computable for every substitution  $\theta$  such that  $\theta(x)$  is strongly computable for all variables  $x$  in the domain of  $\theta$ . The assertion then follows with the empty substitution and lemma 3.2 (a).

For variables and constants the assertion holds by assumption. For constructor terms and applications we use the induction hypothesis and the definition of strong computability. Abstractions are taken care of by the induction hypothesis and lemma 3.4.

⊣

PROPOSITION 3.6. *Gödel's system  $T$  is strongly normalizing.*

PROOF. By proposition 3.5 it suffices to show that all constants, i.e. the recursors are strongly normalizing. We only carry out in detail the proof for recursion on lists and only sketch the general case since later we will prove a more general result (that does not depend on this proposition). We have to show that  $R_{\sigma^*, \rho} MNK$  is strongly computable for all strongly computable terms  $M, N, K$  of appropriate types. Using lemma 3.2 (a) and lemma 3.1 (a) we argue by induction on  $\text{SN}(M, N, K)$ . We also use a side induction on  $K$ . Since  $R_{\sigma^*, \rho} MNK$  is a neutral term it suffices, by lemma 3.3, to show  $\text{SC}_\rho(P)$  for all  $P$  such that  $R_{\sigma^*, \rho} MNK \rightarrow_1 P$ . If the conversion took place within one of the terms in  $M, N, K$ , then we use the main induction hypothesis and lemma 3.1 (a). Otherwise the visible recursor was involved in the conversion. If  $K = []$  and  $P = M$ , then we are done since, by assumption,  $M$  is strongly computable. If  $K = \text{cons}(H, T)$  and  $P = NHT(R_{\sigma^*, \rho} MNT)$ , then  $H$  and  $T$  are strongly computable and hence we know  $\text{SC}_\rho(R_{\sigma^*, \rho} MNT)$  by the side induction hypothesis. Again it follows that  $P$  is strongly computable.

In the general case, when constructors with recursive arguments of a type  $\vec{\sigma} \rightarrow \iota$  with nonempty  $\vec{\sigma}$  may occur, one must replace the term induction on  $K$  by an induction on the value of  $K$  in, e.g. in the model  $\hat{\mathcal{C}}$  of partial continuous functionals (w.r.t. to an arbitrary but fixed variable environment). More precisely, in  $\hat{\mathcal{C}}$  every term of an inductive type denotes some wellfounded labeled tree, and one does induction on the (wellfounded) subtree relation. ⊣

**§4. Stratified terms.** For the rest of this paper we will only consider conversion rules of the form

$$cP_1 \dots P_n \mapsto R$$

where  $\text{FV}(cP_1 \dots P_n) \subseteq \text{FV}(R)$  and the  $P_i$  are *constructor patterns*, i.e. terms built from variables and constructors only. More precisely, a term is a constructor pattern iff it is a variable or of the form  $\text{co}(\vec{P})$  where all the arguments  $P_i$  are constructor patterns.

The set of *stratified terms* is defined inductively as follows: Every variable is stratified; a constant  $c$  is stratified if for every rule  $cP_1 \dots P_n \mapsto R$

the term  $R$  is stratified; a composite term is stratified if all its immediate subterms are.

Clearly a term is stratified iff it contains stratified constants only.

Note that stratification is a severe restriction. For example any constant with a recursive conversion rule, i.e. the constant reappears on the right hand side of the rule, is not stratified. We do not claim that stratified terms are of particular interest as such. We will use them as a technical tool in our termination proof based on strict semantics (section 6).

**PROPOSITION 4.1.** *Every stratified term is strongly normalizing.*

**PROOF.** We proceed similar as in the proof of proposition 3.5. By induction on the stratification of  $M$  we show that  $M\theta$  is strongly computable for every substitution  $\theta$  such that  $\theta(x)$  is strongly computable for all variables  $x \in \text{FV}(M)$ .

Only the case that  $M$  is a constant is interesting. All other cases are as in proposition 3.5, that is, we use the induction hypothesis.

Let  $c$  be a constant. Without loss of generality we assume that all rules for  $c$  are of the form  $cP_1 \dots P_n \mapsto R$  with the number  $n$  of arguments being the same for all rules. Let  $M_1, \dots, M_n$  be strongly computable. We have to show that  $cM_1 \dots M_n$  is strongly computable. We do a side induction on the strong normalizability of the  $M_i$ . Since  $cM_1 \dots M_n$  is neutral it suffices that all one step reducts of this term are strongly computable. If one of the  $M_i$  is reduced, we apply the side induction hypothesis. Otherwise there is a rule  $cP_1 \dots P_n \mapsto R$  and a substitution  $\theta$  with  $(cP_1 \dots P_n)\theta = cM_1 \dots M_n$  and the reduct is  $R\theta$ . Since the  $P_i$  are constructor patterns, it clearly follows from the strong computability of the  $M_i$  that  $\theta(x)$  is strongly computable for each variable  $x \in \text{FV}(M)$ . Hence  $R\theta$  is strongly computable, by the main induction hypothesis.  $\dashv$

**§5. Bar recursion.** We now prove that the extension of system  $T$  by a suitable formulation of Spector's bar recursion is strongly normalizing using domain theoretic semantics with totality. We work with the example type system discussed in section 2.

We will write  $\rho^\omega$  for  $\text{nat} \rightarrow \rho$ , if  $B$  then  $M$  else  $N$  for  $\text{if } BMN$ ,  $|M|$  for  $\text{lh } M$ ,  $M*N$  for  $M ++ \langle N \rangle$  where  $\langle N \rangle := \text{cons}(N, [])$ , and  $\widehat{M}$  for  $\text{get } M$ . Spector's bar recursion in finite types is given (for each pair of types  $\rho, \sigma$ ) by a constant

$$\Phi : (\rho^\omega \rightarrow \text{nat}) \rightarrow (\rho^* \rightarrow \sigma) \rightarrow (\rho^* \rightarrow (\rho \rightarrow \sigma) \rightarrow \sigma) \rightarrow \rho^* \rightarrow \sigma$$

with the following defining equation

$$\Phi yghs = \text{if } y\hat{s} < |s| \text{ then } gs \text{ else } hs(\lambda x. \Phi ygh(s*x))$$

Turning this into a conversion rule would clearly not be strongly normalizing. Therefore we replace the right hand side by a call of an auxiliary constant  $\Psi$  with an extra boolean argument in order to force evaluation of the test  $y\hat{s} < |s|$  before the subterm  $\Phi ygh(s*x)$  may be reduced further (Vogel's trick).

$$\begin{aligned} \Phi yghs &\mapsto \Psi yghs(y\hat{s} < |s|) \\ \Psi yghs\top &\mapsto gs \\ \Psi yghs\text{F} &\mapsto hs(\lambda x. \Phi ygh(s*x)) \end{aligned}$$

In the proof that  $\Phi$  and  $\Psi$  are strongly computable we will make use of the interpretation of terms as total elements in the model domain theoretic model  $\hat{\mathcal{C}}$  of partial continuous functionals (for our type system). The model  $\hat{\mathcal{C}}$  assigns to every type  $\rho$  a Scott domain  $\hat{\mathcal{C}}(\rho)$  such that  $\hat{\mathcal{C}}(\rho \rightarrow \sigma) \equiv [\hat{\mathcal{C}}(\rho) \rightarrow \hat{\mathcal{C}}(\sigma)]$ , the domain of continuous functions from  $\hat{\mathcal{C}}(\rho)$  to  $\hat{\mathcal{C}}(\sigma)$  where ' $\equiv$ ' means 'isomorphic', and for every inductive type  $\iota$  with defining equation  $\iota = \text{co}_1(\vec{\rho}_1) \mid \dots \mid \text{co}_n(\vec{\rho}_n)$

$$\hat{\mathcal{C}}(\iota) \equiv \text{co}_1(\hat{\mathcal{C}}(\vec{\rho}_1)) + \dots + \text{co}_n(\hat{\mathcal{C}}(\vec{\rho}_n))$$

where '+' means the domain theoretic disjoint sum (adding a new bottom element) with injections  $\text{co}_i$  and  $\hat{\mathcal{C}}(\vec{\rho}_i)$  is the domain theoretic product of the domains  $\hat{\mathcal{C}}(\rho_{ij})$ . Since the operations, continuous function space, disjoint sum, and product are continuous co-variant functors on the category of domains with embedding/projection pairs the family of domains  $(\hat{\mathcal{C}}_\rho)_{\rho \in \mathcal{T}}$  can be obtained as the least fixed point of a continuous functor. The stratifying level function is not needed here. We do need stratification, however, for defining the *total elements*. The definition is parallel to the definition of strong computability. The only differences are that term application is replaced by function application, and, of course there is no rule concerning reduction. We define what the total elements in  $\hat{\mathcal{C}}(\rho)$  are by recursion on  $\text{size}(\rho)$ . For types of the same size totality is defined by a simultaneous inductive definition. A continuous function  $f \in \hat{\mathcal{C}}(\rho \rightarrow \sigma)$  is total if  $f(a)$  is total for all total arguments  $a$ . An element of  $\hat{\mathcal{C}}(\iota)$  where  $\iota$  is an inductive type with defining equation as given above is total if it is of the form  $\text{co}_i(\vec{a})$  where all  $a_j$  are total.

Let  $\mathbf{CEnv}$  the domain of all *constant environments*, that is, families  $\alpha$  assigning to each constant  $c^\rho \in \mathcal{C}$  some  $\alpha(c) \in \hat{\mathcal{C}}(\rho)$ . Similarly,  $\mathbf{VEnv}$  is the domain of all *variable environments*, i.e. families  $\eta$  assigning to each variable  $x^\rho \in \mathcal{C}$  some  $\eta(x) \in \hat{\mathcal{C}}(\rho)$ . For each term  $M^\rho$  the *denotational semantics*

$$\llbracket M \rrbracket : \mathbf{CEnv} \rightarrow \mathbf{VEnv} \rightarrow \hat{\mathcal{C}}(\rho)$$

is defined by recursion on  $M$  as follows.

$$\begin{aligned} \llbracket x \rrbracket^{\alpha\eta} &= \eta(x) \\ \llbracket c \rrbracket^{\alpha\eta} &= \alpha(c) \\ \llbracket \lambda x M \rrbracket^{\alpha\eta}(a) &= \llbracket M \rrbracket^{\alpha\eta_x^a} \\ \llbracket MN \rrbracket^{\alpha\eta} &= \llbracket M \rrbracket^{\alpha\eta} \llbracket N \rrbracket^{\alpha\eta} \\ \llbracket \text{co}(M_1, \dots, M_k) \rrbracket^{\alpha\eta} &= \text{co}(\llbracket M_1 \rrbracket^{\alpha\eta}, \dots, \llbracket M_k \rrbracket^{\alpha\eta}) \end{aligned}$$

where  $\eta_x^a(x) := a$  and  $\eta_x^a(y) := \eta(y)$  for variables  $y$  different from  $x$ .

LEMMA 5.1. *If the constant environment  $\alpha$  and the variable environment  $\eta$  are total, then  $\llbracket M \rrbracket^{\alpha\eta}$  is total for every term  $M$ .*

PROOF. Induction on terms (fundamental lemma on logical relations).  $\dashv$

In this section we set  $\llbracket M \rrbracket \eta := \llbracket M \rrbracket^{\alpha_{\mathbf{B}}\eta}$  where  $M$  is a barrecursive term, i.e. constants are either Gödel primitive recursive or the barrecursors  $\Phi$ ,  $\Psi$  above, and  $\alpha_{\mathbf{B}}$  interprets constants as the (simultaneous) least fixed points of the functionals defined by their conversion rules in the obvious way. In the following we will write  $c$  for  $\alpha_{\mathbf{B}}(c)$ .

LEMMA 5.2. *All Gödel primitive recursive constants are total.*

PROOF. Obvious.  $\dashv$

In order to prove the totality of the constants  $\Phi$  and  $\Psi$  we use the following continuity property of total functionals  $y$  of type  $\rho^\omega \rightarrow \mathbf{nat}$ : For any total argument  $\alpha$  there is a number  $n$  such that  $y(\alpha) = y(\beta)$  for all total  $\beta$  coinciding with  $\alpha$  at all arguments  $k < n$ . Now let us define for every total  $y \in \hat{\mathcal{C}}(\rho^\omega \rightarrow \mathbf{nat})$  a binary relation  $\gg_y$  on the total elements of type  $\rho^*$  (these are finite lists of total elements of type  $\rho$ ):

$$s \gg_y t : \equiv y(\hat{s}) \geq |s| \wedge s * a = t \text{ for some total } a$$

LEMMA 5.3. *For every total  $y$  the relation  $\gg_y$  is wellfounded.*

PROOF. If not, we would have a total list  $s$ , of length  $n$  say, and an infinite sequence of total elements  $a_0, a_1, \dots$  such that  $y(\widehat{s}^i) \geq n + i + 1$  where  $\widehat{s}^i := s * a_0 \dots * a_i$  for all  $i$ . Set  $\alpha(k) := s_k$  if  $k < n$  and  $:= a_{k-n}$  if  $k \geq n$ . this defines a total element of  $\widehat{C}(\rho^\omega)$ . By continuity of  $y$  there is some  $k$  such that  $y(\alpha) = y(\widehat{s}^i)$  for all  $i \geq k$  which is a contradiction.  $\dashv$

LEMMA 5.4.  $\Phi$  and  $\Psi$  are total.

PROOF. Easy using previous lemma.  $\dashv$

THEOREM 5.5. *The extension of system  $T$  by the bar recursive constants  $\Phi$  and  $\Psi$  is strongly normalizing.*

PROOF. It suffices to show that  $\Phi$  and  $\Psi$  are strongly computable. We only show the strong computability of  $\Phi$  since  $\text{SC}(\Psi)$  follows then easily. Let  $Y, G, H, M$  be strongly computable. We have to show that  $\Phi YGHM$  is strongly computable. We do this by induction on  $\text{SN}(Y, G, H, M)$ . Let the environment  $\eta_0$  be defined by  $\eta_0(x^\tau) := 0^\tau$  and set  $y := \llbracket Y \rrbracket \eta_0$ . By the previous lemmas the relation  $\gg_y$  is wellfounded, hence we may do a  $\gg_y$ -side induction on  $\llbracket M \rrbracket \eta_0$ . Since the term  $\Phi YGHM$  is neutral it suffices to show that all its one step reducts are strongly computable. If one of  $Y, G, H, M$  is reduced we use the main induction hypothesis. It remains to prove that  $\Psi YGHM(Y\widehat{M} < |M|)$  is strongly computable. Let  $b := \llbracket Y\widehat{M} < |M| \rrbracket \eta_0$ . We show that  $\Psi YGHMB$  is strongly computable for all strongly computable terms  $B$  with  $\llbracket B \rrbracket \eta_0 = b$ . We employ a second side induction on  $\text{SN}(B)$ . Again the term  $\Psi YGHMB$  is neutral, so it suffices to consider its one step reducts  $P$ . Reducing one of  $Y, G, H, M, B$  is dealt with by either the main or the second side induction hypothesis. Two cases remain. If  $B = \top$  and  $P = M_K$  we are done. If however  $B = \text{F}$  and  $P = HM(\lambda x. \Phi YGH(M*x))$ , then  $b$  is false which means that  $y(\widehat{s}) \geq |s|$  where  $s := \llbracket M \rrbracket \eta_0$ . We have to show that  $\lambda x. \Phi YGH(M*x)$  is strongly computable. Assume  $\text{SC}_\rho(N)$ . We have to show that  $\Phi YGH(M*N)$  is strongly computable. Set  $t := \llbracket M*N \rrbracket \eta_0 = s * \llbracket M \rrbracket \eta_0$ . Since  $y(\widehat{s}) \geq |s|$  we have  $s \gg_y t$ . Hence the strong computability of  $\Phi YGH(M*N)$  follows from the first side induction hypothesis.  $\dashv$

*Remark.* Tait [4], Vogel [5], Luckhardt [3] and Bezem [2] proved strong normalization for barrecursive functionals formulated in a combinatorial calculus. Our result seems to be slightly stronger since we work in a  $\lambda$ -calculus framework which allows more reductions. From a logical point of

view our proof is roughly equivalent to the proofs in the work cited, since the partial continuous functionals can be defined primitive recursively (finite neighborhoods, or compact elements of Scott domains) and totality in  $\hat{\mathcal{C}}(\rho)$  has the same logical complexity as, say the definition of strong computability for infinite terms of type  $\rho$ .

Another form of bar recursion was suggested by Berardi, Bezem and Coquand [1]:

$$\Phi : (\rho^\omega \rightarrow \text{nat}) \rightarrow (\text{nat} \rightarrow (\rho \rightarrow \text{nat}) \rightarrow \rho) \rightarrow \rho^* \rightarrow \text{nat}$$

with the defining equation

$$\Phi ygs = y(\lambda k. \text{if } k < |s| \text{ then } s_k \text{ else } gk(\lambda x. \Phi yg(s*x)))$$

where

$$M_K := \mathbf{R}_{\rho^*, \rho^\omega}(\lambda k 0^\rho)(\lambda h \lambda t \lambda f \lambda k \text{ if } k < |t| \text{ then } f k \text{ else } h) M K$$

so  $(\llbracket *N_0 * \dots * N_{k-1} \rrbracket_{\underline{i}} \rightarrow_1^* N_i$  for  $i < k$ . One can attempt to make this strongly normalizing by applying Vogel's trick again:

$$\begin{aligned} \Phi ygs &\mapsto y(\lambda k. \Psi ygs k (k < |s|)) \\ \Psi ygs k \mathbf{T} &\mapsto s_k \\ \Psi ygs k \mathbf{F} &\mapsto gk(\lambda x. \Phi yg(s*x)) \end{aligned}$$

In the next section we will prove a general strong normalization theorem (theorem 6.2) that will us allow to show that system  $T$  plus  $\Phi$  and  $\Psi$  above is strongly normalizing as well.

**§6. Termination based on strict semantics.** Our leading idea for a general termination theorem based on totality is to deduce the termination of a term  $M$  from the totality of the constants in  $M$ . As we saw, this works for Vogel's variant of bar recursion, but it would clearly fail for Spector's original bar recursion written as the conversion rule

$$\Phi ygs \mapsto y(\lambda k. \text{if } k < |s| \text{ then } s_k \text{ else } gk(\lambda x. \Phi yg(s*x)))$$

Despite the totality of  $\Phi$ 's denotation as the least fixed point of the continuous operator defined by this rule, the corresponding rewrite system is not strongly normalizing. The intuitive reason for this is that the totality hinges on the non-strictness of the **if then else** construct while for strong normalization we also require termination of reduction sequences that ignore this non-strictness (i.e. a branch of the case analysis may be reduced

before reducing the test). On the other hand the operator defined by Vogel's variant

$$\begin{aligned}\Phi yghs &\mapsto \Psi yghs(y\hat{s} < |s|) \\ \Psi yghs\mathbb{T} &\mapsto gs \\ \Psi yghs\mathbb{F} &\mapsto hs(\lambda x.\Phi ygh(s*x))\end{aligned}$$

which has a total least fixed point as well, does not rely on non-strict constructs. This suggests to separate the two versions of bar recursion by a modified denotational semantics  $[M]$  that interprets all constructs in a strict way and derive termination from totality with respect to this strict semantics. The strictness of the modified semantics means that  $[M]^{\alpha\eta} = \perp$  if  $M$  contains a constant  $c$  with  $\alpha(c) = \perp$ . We will achieve this by interpreting the application function and the constructors in a strict way. To this end we define recursively continuous functions  $\delta_\rho: \hat{\mathcal{C}}(\rho) \rightarrow \hat{\mathcal{C}}(\text{boole})$  by  $\delta_{\rho \rightarrow \sigma}(f) := \delta_\sigma(f0^\rho)$ ,  $\delta_\iota(\text{co}(a_1, \dots, a_n)) := \delta(a_1) \wedge \dots \wedge \delta(a_n)$ ,  $\delta_\iota(\perp) := \perp$ . It can be easily shown that  $\delta(a) = \mathbb{T}$  for all total  $a \in \hat{\mathcal{C}}(\rho)$  and  $\delta(\perp) = \perp$  (and these are the only properties we will use of  $\delta$ ). We use  $\delta$  to define strict variants of application and the constructors (at all appropriate types).

$$\begin{aligned}\tilde{\mathfrak{a}}(f, a) &:= \text{if } \delta(a) \text{ then } f(a) \text{ else } \perp \\ \tilde{\mathfrak{c}}\text{co}(a_1, \dots, a_k) &:= \text{if } \delta(a_1) \wedge \dots \wedge \delta(a_k) \text{ then } \text{co}(a_1, \dots, a_k) \text{ else } \perp\end{aligned}$$

Clearly the functions  $\tilde{\mathfrak{a}}(\cdot, \cdot)$  and  $\tilde{\mathfrak{c}}\text{co}(\cdot, \cdot)$  are continuous, strict, and coincide for total arguments with the application function and the constructors, respectively. The strict semantics,  $[M]$ , is now defined by recursion on  $M$  as follows. We now define for each term  $M^\rho$  the *strict denotational semantics*  $[M]: \text{CEnv} \rightarrow \text{VEnv} \rightarrow \hat{\mathcal{C}}(\rho)$  by

$$\begin{aligned}[x]^{\alpha\eta} &= \eta(x) \\ [c]^{\alpha\eta} &= \alpha(c) \\ [\lambda x M]^{\alpha\eta}(a) &= [M]^{\alpha\eta_x^a} \\ [MN]^{\alpha\eta} &= \tilde{\mathfrak{a}}([M]^{\alpha\eta}, [N]^{\alpha\eta}) \\ [\text{co}(M_1, \dots, M_k)]^{\alpha\eta} &= \tilde{\mathfrak{c}}\text{co}([M_1]^{\alpha\eta}, \dots, [M_k]^{\alpha\eta})\end{aligned}$$

LEMMA 6.1. (a) *If  $\alpha(c)$  is total for all constants  $c$  in  $M$  and  $\eta(x)$  is total for all  $x \in \text{FV}(M)$ , then  $[M]^{\alpha\eta}$  is total.*

(b) *If  $\alpha(c) = \perp$  for some constant  $c$  occurring in  $M$ , then  $[M]^{\alpha\eta} = \perp$ .*

- (c)  $[M]^\alpha([\theta]^\alpha\eta) = [M\theta]^\alpha\eta$  where  $([\theta]^\alpha\eta)(x) := [\theta(x)]^\alpha\eta$ .  
(d)  $[M]^{[\zeta]^\alpha\eta} = [M\zeta]^\alpha\eta$  where  $\zeta$  is a ‘constant substitution’, i.e.  $\zeta(c^\rho)$  is a term of type  $\rho$  for each constant  $c^\rho$ , and  $([\zeta]^\alpha\eta)(c) := [\zeta(c)]^\alpha\eta$ .

PROOF. Easy induction on  $M$ . ⊥

We now consider rewrite systems in general that induce a denotational semantics of the constants in a canonical way. A *functional rewrite system* is a system of rules

$$cP_1 \dots P_n \mapsto R$$

which are *left linear*, i.e. a variable occurs at most once in the left hand side of a rule, and *mutually disjoint*, i.e. the left hand sides of two different rules are non-unifiable (and of course the  $P_i$  are constructor patterns and  $\text{FV}(cP_1 \dots P_n) \subseteq \text{FV}(R)$ ). All rewrite systems considered so did satisfy these conditions.

We let  $\perp$  denote the ‘undefined’ variable environment, i.e.  $\perp(x) = \perp_\rho$  for all variables  $x^\rho$ .

For a vector  $\vec{P}$ :  $\vec{\rho}$  of constructor patterns containing each variable at at most one place and  $\vec{a} \in \hat{\mathcal{C}}(\vec{\rho})$  we define the  $\vec{P}$ -predecessor of  $\vec{a}$ ,  $\text{pred}_{\vec{P}}(\vec{a}) \in \text{CEnv}$ , by recursion on the number of constructors occurring in  $\vec{P}$ .

$$\begin{aligned} \text{pred}_{\vec{x}}(\vec{a}) &= \perp_{\vec{x}}^{\vec{a}} \\ \text{pred}_{\vec{x}, \text{co}(\vec{Q}), \vec{P}}(\vec{a}, \text{co}(\vec{b}), \vec{c}) &= \text{pred}_{\vec{x}, \vec{Q}, \vec{P}}(\vec{a}, \vec{b}, \vec{c}) \\ \text{pred}_{\vec{x}, \text{co}(\vec{Q}), \vec{P}}(\vec{a}, b, \vec{c}) &= \perp \quad \text{if } b \text{ is not of the form } \text{co}(\vec{b}) \end{aligned}$$

We say  $\vec{a}$  *matches*  $\vec{P}$  if in the definition of  $\text{pred}_{\vec{P}}(\vec{a})$  the last clause has never been used. Let  $\mathcal{F}$  be a functional rewrite system. For a given vector  $\vec{a} \in \hat{\mathcal{C}}$  there can be at most one rule  $c\vec{P} \mapsto R \in \mathcal{F}$  such that  $\vec{a}$  matches  $\vec{P}$ , because the rules in  $\mathcal{F}$  are mutually disjoint. Therefore the following operator  $\Gamma_{\mathcal{F}}: \text{CEnv} \rightarrow \text{CEnv}$  is welldefined and continuous.

$$\Gamma_{\mathcal{F}}(\alpha)(c)(\vec{a}) := \bigsqcup \{ [R]^\alpha \text{pred}_{\vec{P}}(\vec{a}) \mid c\vec{P} \mapsto R \in \mathcal{F}, \vec{a} \text{ matches } \vec{P} \}$$

We define the constant environment  $\alpha_{\mathcal{F}}$  as the least fixed point of the operator  $\Gamma_{\mathcal{F}}$ .

It follows the main the result of this paper.

**THEOREM 6.2.** *Let  $\mathcal{F}$  be a functional rewrite system. If  $\alpha_{\mathcal{F}}(c)$  is total for every constant in  $M$ , then  $M$  is strongly normalizing.*

The proof of this theorem needs some preparation. In the following we fix a functional rewrite system  $\mathcal{F}$ .

LEMMA 6.3. *Let  $c\vec{P} \mapsto R \in \mathcal{F}$  be a rule,  $\theta$  a substitution and  $\eta$  a variable environment. Then  $\vec{a} := [\vec{P}\theta]^{\alpha_{\mathcal{F}}}\eta$  matches  $\vec{P}$  and  $\alpha_{\mathcal{F}}(c)(\vec{a}) = [R]^{\alpha_{\mathcal{F}}}\text{pred}_{\vec{P}}(\vec{a})$ .*

PROOF. That  $[\vec{P}\theta]^{\alpha_{\mathcal{F}}}\eta$  matches  $\vec{P}$  is easily shown by induction on the number of constructors in  $\vec{P}$ . The rest follows immediately from the definition of  $\Gamma_{\mathcal{F}}$ .  $\dashv$

LEMMA 6.4. *If  $M \rightarrow_1 N$ , then  $[M]^{\alpha_{\mathcal{F}}}\eta \sqsubseteq [N]^{\alpha_{\mathcal{F}}}\eta$ .*

PROOF. Induction on  $M$ .

Case  $(\lambda xM)N \rightarrow_1 M[N/x]$ . Set  $a := [N]^{\alpha_{\mathcal{F}}}\eta$ .  $[(\lambda xM)N]^{\alpha_{\mathcal{F}}}\eta = \tilde{a}([\lambda xM]^{\alpha_{\mathcal{F}}}\eta, a) = \text{if } \delta(a) \text{ then } [M]^{\alpha_{\mathcal{F}}}\eta_x^a \text{ else } \perp \sqsubseteq [M]^{\alpha_{\mathcal{F}}}\eta_x^a = [M[N/x]]^{\alpha_{\mathcal{F}}}\eta$ . The last equation holds by lemma 6.1 (c).

Case  $c\vec{P}\theta \rightarrow_1 R\theta$  for some rule  $c\vec{P} \mapsto R \in \mathcal{F}$ .  $[c\vec{P}\theta]^{\alpha_{\mathcal{F}}}\eta = \tilde{a}(\alpha_{\mathcal{F}}(c), \vec{a}) \sqsubseteq \alpha_{\mathcal{F}}(c)(\vec{a}) = [R]^{\alpha_{\mathcal{F}}}\text{pred}_{\vec{P}}(\vec{a}) = [R\theta]^{\alpha_{\mathcal{F}}}\eta$ , where we wrote  $\tilde{a}(\alpha_{\mathcal{F}}(c), \vec{a})$  as an abbreviation for  $\tilde{a}(\dots \tilde{a}(\alpha_{\mathcal{F}}(c), a_1), \dots a_k)$ , and the last two equations hold by lemma 6.3 and lemma 6.1 (c).

All other cases (i.e. conversion of a proper subterm) follow immediately from the induction hypothesis and the fact that the functions  $\tilde{a}(\dots)$  and  $\tilde{c}\tilde{o}(\dots)$  are monotone.  $\dashv$

We now introduce a stratified variant  $\mathcal{F}_{\omega}$  of  $\mathcal{F}$ . Let  $\mathcal{C}$  be the set of constants of  $\mathcal{F}$ . For each  $c \in \mathcal{C}$  and every natural number we introduce a new constant  $c_n$ . Set  $\mathcal{C}_{\omega} := \{c_n \mid c \in \mathcal{C}, n \in \omega\}$ . For any  $\mathcal{C}$ -term  $M$  let  $M_{[n]}$  be the  $\mathcal{C}_{\omega}$ -term obtained from  $M$  by replacing every occurring constant  $c$  by  $c_n$ . We set

$$\mathcal{F}_{\omega} := \{c_{n+1}\vec{P} \mapsto R_{[n]} \mid c\vec{P} \mapsto R \in \mathcal{F}, n \in \omega\}$$

Clearly  $\mathcal{F}_{\omega}$  is again a functional rewrite system. Furthermore all constants  $c_n$  are stratified (induction on  $n$ ). Hence we have, by proposition 4.1

LEMMA 6.5. *Every  $\mathcal{C}_{\omega}$ -term is strongly normalizing.*

We write  $A \preceq M$  if  $M$  is a  $\mathcal{C}$ -term and  $A$  is a  $\mathcal{C}_{\omega}$ -term obtained from  $M$  by replacing every occurrence of a constant  $c$  by some  $c_n$  (different occurrences of the same constant may receive different indices).

LEMMA 6.6. *If  $A \preceq M$  and  $A$  contains no constant of the form  $c_0$ , then to every  $\mathcal{C}$ -term  $M'$  such that  $M \rightarrow_1 M'$  there is a  $\mathcal{C}_\omega$ -term  $A'$  such that  $A \rightarrow_1 A'$  and  $A' \preceq M'$ .*

PROOF. Easy induction on  $M$ . ⊣

We define the  $\mathcal{C}_\omega$ -constant environment  $\alpha_{\mathcal{F}_\omega}$  like the  $\mathcal{C}$ -constant environment  $\alpha_{\mathcal{F}}$ , but with  $\mathcal{F}$  replaced by  $\mathcal{F}_\omega$ . Hence

$$\alpha_{\mathcal{F}_\omega}(c_{n+1})(\vec{a}) = \bigsqcup \{ [R_{[n]}]^{\alpha_{\mathcal{F}_\omega}} \text{pred}_{\vec{P}}(\vec{a}) \mid c\vec{P} \mapsto R \in \mathcal{F}, \vec{a} \text{ matches } \vec{P} \}$$

and  $\alpha_{\mathcal{F}_\omega}(c_0) = \perp$  (there is no rule for  $c_0$ ).

LEMMA 6.7.  $\alpha_{\mathcal{F}}(c) = \bigsqcup_{n \in \omega} \alpha_{\mathcal{F}_\omega}(c_n)$  for every constant  $c \in \mathcal{C}$ .

PROOF. Set  $\alpha_n(c) := \alpha_{\mathcal{F}_\omega}(c_n)$ . Since  $\alpha_{\mathcal{F}} = \bigsqcup_{n \in \omega} \Gamma_{\mathcal{F}}^n \perp$ , it suffices to show  $\alpha_n = \Gamma_{\mathcal{F}}^n \perp$  for all  $n$ . We prove this by induction on  $n$ . For  $n = 0$  both sides are  $\perp$ .

$$\begin{aligned} & \alpha_{n+1}(c)(\vec{a}) \\ &= \alpha_{\mathcal{F}_\omega}(c_{n+1})(\vec{a}) \\ &= \bigsqcup \{ [R_{[n]}]^{\alpha_{\mathcal{F}_\omega}} \text{pred}_{\vec{P}}(\vec{a}) \mid c\vec{P} \mapsto R \in \mathcal{F}, \vec{a} \text{ matches } \vec{P} \} \\ &= \bigsqcup \{ [R]^{\alpha_n} \text{pred}_{\vec{P}}(\vec{a}) \mid c\vec{P} \mapsto R \in \mathcal{F}, \vec{a} \text{ matches } \vec{P} \} \text{ (lemma 6.1 (d))} \\ &= \Gamma_{\mathcal{F}}(\alpha_n)(c)(\alpha) \\ &= (\Gamma_{\mathcal{F}}^{n+1} \perp)(c)(\alpha) \text{ (induction hypothesis)} \end{aligned}$$

⊣

LEMMA 6.8.  $[M]^{\alpha_{\mathcal{F}}} \eta = \bigsqcup_{n \in \omega} [M_{[n]}]^{\alpha_{\mathcal{F}_\omega}} \eta$  for every  $\mathcal{C}$ -term  $M$  and every variable environment  $\eta$ .

PROOF. Set, as in the previous proof,  $\alpha_n(c) := \alpha_{\mathcal{F}_\omega}(c_n)$ . By the lemma 6.7 we have  $\alpha_{\mathcal{F}} = \bigsqcup_{n \in \omega} \alpha_n$ . Hence, because  $[M]$  is a continuous function,

$$[M]^{\alpha_{\mathcal{F}}} \eta = \bigsqcup_{n \in \omega} [M]^{\alpha_n} \eta \stackrel{6.1 \text{ (d)}}{=} \bigsqcup_{n \in \omega} [M_{[n]}]^{\alpha_{\mathcal{F}_\omega}} \eta$$

⊣

Now we are ready to prove theorem 6.2. Let  $M$  be a ( $\mathcal{C}$ -)term such that  $\alpha_{\mathcal{F}}(c)$  is total for every constant in  $M$ . Let  $\eta$  be any total environment. Then  $[M]^{\alpha_{\mathcal{F}}} \eta$  is total, by lemma 6.1 (a), and therefore different from  $\perp$ .

By lemma 6.8 it follows that there is some  $n$  such that  $[M_{[n]}]^{\alpha_{\mathcal{F}_\omega}} \eta \neq \perp$ . Clearly  $M_{[n]} \preceq M$ . Therefore it suffices to show that whenever  $A \preceq N$  and  $[A]^{\alpha_{\mathcal{F}_\omega}} \eta \neq \perp$ , then  $N$  is strongly normalizing. We prove this by induction on the strong normalizability of the  $\mathcal{C}_\omega$ -term  $A$ , using lemma 6.5. We need to show that all one step reducts of  $M$  are strongly normalizing. So, assume  $N \rightarrow_1 N'$ . Since  $[A]^{\alpha_{\mathcal{F}_\omega}} \eta \neq \perp$  we know, by lemma 6.1 (b), that  $A$  does not contain a constant of the form  $c_0$ . It follows with lemma 6.6 that  $A \rightarrow_1 A'$  with  $A' \preceq N'$  for some  $\mathcal{C}_\omega$ -term  $A'$ . By lemma 6.4  $[A']^{\alpha_{\mathcal{F}_\omega}} \eta \neq \perp$ , hence we can apply the induction hypothesis to  $A'$  and  $N'$ .

Let us now apply theorem 6.2 to prove the strong normalization results announced at the end of section 5.

**THEOREM 6.9.** *Gödel's system  $T$  plus the constant  $\Phi$  and  $\Psi$  with the conversion rules*

$$\begin{aligned}\Phi ygs &\mapsto y(\lambda k. \Psi ygs k (k < |s|)) \\ \Psi ygs k \mathbf{T} &\mapsto s_k \\ \Psi ygs k \mathbf{F} &\mapsto gk(\lambda x^\rho. \Phi yg(s*x))\end{aligned}$$

*above is strongly normalizing.*

**PROOF.** Our strict semantics interprets the constants  $\Phi$  and  $\Psi$  as continuous functionals  $\varphi$  and  $\psi$  which satisfy for *total arguments*  $y, g, s, k$  (in  $\hat{\mathcal{C}}$ ) the equations

$$\begin{aligned}\varphi ygs &= \text{if } \delta(\psi ygs 0 (0 < |s|)) \text{ then } y(\lambda k. \psi ygs k (k < |s|)) \text{ else } \perp \\ \psi ygs k \mathbf{T} &= s_k \\ \psi ygs k \mathbf{F} &= \text{if } \delta(\varphi yg(s*0^\rho)) \text{ then } gk(\lambda x. \varphi yg(s*x)) \text{ else } \perp\end{aligned}$$

By bar induction on the wellfounded tree of unsecured sequences of the total continuous functional  $y$  it follows that  $\varphi$  and  $\psi$  are total. This completes the proof.  $\dashv$

It is instructive to see why the constant  $\Phi$  with the conversion rule

$$\Phi ygs = y(\lambda k. \text{if } k < |s| \text{ then } s_k \text{ else } gk(\lambda x. \Phi yg(s*x)))$$

is *not* interpreted as a total functional (it shouldn't because the rule isn't strongly normalizing). The strict semantics interprets the constant  $\Phi$  with the above rule as the least solution  $\varphi$  of an recursive equation of the form

$$\varphi ygs = \text{if } \delta(\varphi yg(s*0^\rho)) \text{ then } \dots \text{ else } \dots$$

which, because of the strictness of  $\delta$ , clearly has  $\varphi = \perp$  as solution.

## REFERENCES

- [1] STEFANO BERARDI, MARC BEZEM, and THIERRY COQUAND, *On the computational content of the axiom of choice*, ***The Journal of Symbolic Logic***, vol. 63 (1998), no. 2, pp. 600–622.
- [2] M. BEZEM, *Strong normalization of barrecursive terms without using infinite terms*, ***Archive for Mathematical Logic***, vol. 25 (1985), pp. 175–181.
- [3] H. LUCKHARDT, *Extensional Gödel functional interpretation – a consistency proof of classical analysis*, Lecture Notes in Mathematics, vol. 306, Springer, 1973.
- [4] W.W. TAIT, *Normal form theorem for barrecursive functions of finite type*, ***Proceedings of the second scandinavian logic symposium*** (J.E. Fenstad, editor), North-Holland, Amsterdam, 1971, pp. 353–367.
- [5] H. VOGEL, *Ein starker normalisationssatz für die barrekursiven funktionale*, ***Archive for Mathematical Logic***, vol. 18 (1985), pp. 81–84.

DEPARTMENT OF COMPUTER SCIENCE  
UNIVERSITY OF WALES SWANSEA  
SINGLETON PARK  
SWANSEA  
SA2 8PP, UNITED KINGDOM  
*E-mail:* u.berger@swan.ac.uk