Continuous Semantics for Strong Normalization

Ulrich Berger
u.berger@swan.ac.uk

University of Wales Swansea

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

The problem of proving strong normalization for typed λ-calculi and higher type rewrite systems has been studied extensively in the literature [14, 9, 15, 10, 17, 7, 5, 8, 1, 16, 6, 11]. In this paper we present a new method for proving strong normalization of higher type rewrite systems based on a strict domain-theoretic semantics. The idea is similar to Plotkin's adequacy proof for PCF [12]: One gives a suitable interpretation of terms in a domain-theoretic model and uses a continuity argument to show that any term not denoting $\bot$ normalizes. The main difference between Plotkin's and our result is that while Plotkin considers terms with a general fixed point operator, for which, of course, only *weak* normalization can be proven, we consider recursion schemes defined by pattern matching and we prove *strong* normalization.

Another new aspect of our method is that it allows for a modular normalization proof: First one proves strong normalization for the underlying typed λ-calculus with *stratified constants* only, i.e. constants with conversion rules that do not involve recursion. This can be done by an extension of Tait's computability method [14]. Then one uses a continuity argument to lift strong normalization to recursively defined constants that have a total value w.r.t. to a strict domain-theoretic semantics.

We will apply our results to a λ-calculus formulation of Gödel's system T extended by two versions of bar recursion in finite types: Spector's original version [13], and a version due to Berardi, Bezem and Coquand [2]. For this system strong normalization was hitherto unknown.

In this paper we consider a simply typed system over the booleans and the integers, closed under the formation of list and function types. The motivation for this (somewhat ad hoc) choice is that this allows for a convenient formulation of bar recursion. Our results could be easily extended to type systems closed under arbitrary strictly positive definitions. Also extensions to second-order polymorphic types seem to be possible.

2 Extended Gödel's system T

The set of *types*, $\rho, \sigma, \ldots$ is generated from the base types boole and nat by the formation of *list types*, ρ^*, and *function types*, $\rho \to \sigma$. As usual we write $\boldsymbol{\rho} \to \sigma$ for $\rho_1 \to \ldots \to \rho_n \to \sigma$ where $\to$ associates to the right. Types which are not function types, i.e. boole, nat and list types ρ^*, are called *inductive* (because their elements will be generated inductively). We let the letter ι range over inductive types.

The term language is determined by a set $\mathcal{C}$ of *typed constants* c^ρ. *Typed terms* are constructed from *typed variables*, x^ρ, and constants, c^ρ, by abstraction, $(\lambda x^\rho M^\sigma)^{\rho\to\sigma}$, application, $(M^{\rho\to\sigma} N^\rho)^\sigma$, and *constructor term* formation, 0^{nat}, $\mathsf{S}(M^{\mathsf{nat}})^{\mathsf{nat}}$, $[]^{\rho^*}$, $\mathsf{cons}(M^\rho, N^{\rho^*})^{\rho^*}$. Type information will often be omitted provided this doesn't cause ambiguities. Instead of M^ρ we will sometimes write $M\colon\rho$. We let the symbol co range over the constructors 0, S, $[]$ and cons. In a constructor term $\mathsf{co}(\boldsymbol{M})^\iota$ the terms $\boldsymbol{M}$ are called *arguments*.

β-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 θ, which are mappings from 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 $\rho_1 \to \ldots \to \rho_n \to \sigma$ is determined by *constant-conversion rules* of the form

$$c\, L_1^{\rho_1} \ldots L_n^{\rho_n} \;\mapsto\; R^\sigma$$

We require that for any constant c the number n above is fixed, i.e. if $c\boldsymbol{L} \mapsto R$ and $c\boldsymbol{L}' \mapsto R'$ are rules, then the vectors $\boldsymbol{L}$ and $\boldsymbol{L}'$ must have the same length. In the situation above we say that c *takes* n *arguments*.

Consider, for example, the constants $\mathsf{if}\colon \mathsf{boole} \to \rho \to \rho \to \rho$, $<\colon \mathsf{nat} \to \mathsf{nat} \to \mathsf{boole}$, $\mathsf{lh}\colon \rho^* \to \mathsf{nat}$, $\mathsf{get}\colon \rho^* \to \mathsf{nat} \to \rho$, and ${+\!\!+}\colon \rho^* \to \rho^* \to \rho^*$,

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

$$\begin{aligned}
\mathsf{get}\,\mathsf{cons}(x,s)\,\mathsf{S}(n) &\mapsto \mathsf{get}\,s\,n\\
[] +\!\!+\, t &\mapsto t\\
\mathsf{cons}(x,s) +\!\!+\, t &\mapsto \mathsf{cons}(x, s +\!\!+\, t)
\end{aligned}$$

where 0^ρ is some closed term of type ρ. These are examples *primitive recursion in higher types*. Gödel's system T summarizes this definition pattern by constants for primitive recursion, $\mathsf{R}_{\mathsf{nat},\rho}\colon \rho \to (\mathsf{nat} \to \rho \to \rho) \to \mathsf{nat} \to \rho$, with the conversion rules

$$\begin{aligned}
\mathsf{R}_{\mathsf{nat},\rho} xy0 &\mapsto x\\
\mathsf{R}_{\mathsf{nat},\rho} xy\mathsf{S}(z) &\mapsto yz(\mathsf{R}_{\mathsf{nat},\rho} xyz)
\end{aligned}$$

Similar rules can be introduced for recursion constants for the other inductive types. In sections 5 we will also consider constants with rules that cannot be derived from primitive recursion.

By a *conversion* we mean a β-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 θ. We write $M \to_1 N$ 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*, $\mathsf{SN}(M)$, if M is in the accessible part of the relation $\to_1$, i.e. there is no infinite reduction sequence $M \to_1 M_1 \to_1 M_2 \to_1 \ldots$. Equivalently, the predicate SN can be inductively defined by the rule

$$\frac{\forall K\,(M \to_1 K \quad \to \quad \mathsf{SN}(K))}{\mathsf{SN}(M)}$$

We call a system of constant-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 types 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 ρ what it means for a term M^ρ to be *strongly computable*, $\mathsf{SC}_\rho(M)$. The definition is by recursion on (the built up of) ρ. For an inductive type ι the predicate SC_ι is defined inductively. We only give the rules for a list type ρ^*. For boole and nat the rules are similar.

$$\mathsf{SC}_{\rho^*}([]) \qquad \frac{\mathsf{SC}_\rho(M) \qquad \mathsf{SC}_{\rho^*}(N)}{\mathsf{SC}_{\rho^*}(\mathsf{cons}(M,N))}$$

$$\frac{\forall K\,(M{\to_1}K \;\to\; \mathsf{SC}_{\rho^*}(K))}{\mathsf{SC}_{\rho^*}(M)} \qquad (M \text{ not a constructor term})$$

$\mathsf{SC}_{\rho\to\sigma}$ is defined explicitely from SC_ρ and SC_σ.

$$\mathsf{SC}_{\rho\to\sigma}(M) \equiv \forall N\,(\mathsf{SC}_\rho(N) \to \mathsf{SC}_\sigma(MN))$$

Lemma 1. *(a) If* $\mathsf{SC}_\rho(M)$ *and* $M \to_1 M'$*, then* $\mathsf{SC}_\rho(M')$*.*
(b) A constructor term is strongly computable iff all its arguments are.

Proof. (a) Easy induction on ρ. If ρ is an inductive type the assertion is proved by a side induction on the definition of SC_ρ. For function types we use the (main) induction hypothesis.

(b) Obvious.

Lemma 2. *(a)* $\mathsf{SC}_\rho(M) \to \mathsf{SN}(M)$*.*
(b) $\mathsf{SC}_\rho(x)$ *for every variable* x *of type* ρ*.*

Proof. Induction on ρ. In order to get the proof through we need to strengthen part (b) to

(b') $\mathsf{SN}(A) \to \mathsf{SC}_\rho(A)$ 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 ρ is an inductive type, then the implication follows easily by a side induction on the definition of $\mathsf{SC}_\rho(M)$, possibly using the main induction hypothesis. *Case* $\rho \to \sigma$. Assume $\mathsf{SC}_{\rho\to\sigma}(M)$. By i.h. (b') we have $\mathsf{SC}(x^\rho)$. Hence $\mathsf{SC}_\sigma(Mx)$. By i.h. (a), $\mathsf{SN}(Mx)$. Hence $\mathsf{SN}(M)$.

(b') Let A be a strongly normalizing term with variable head. If A has an inductive type, then we show $\mathsf{SC}(A)$ by a side induction on $\mathsf{SN}(A)$. Since A is not a constructor term, it suffices to show $\mathsf{SC}(B)$ for all one step reducts B of A. Clearly B has variable head, hence $\mathsf{SC}(B)$ by side induction hypothesis. If A has type $\rho \to \sigma$, we assume $\mathsf{SC}\rho(M)$ and have to show $\mathsf{SC}_\sigma(AM)$. By induction hypothesis (a) we have $\mathsf{SN}(M)$. Hence $\mathsf{SN}(AM)$ (one easily proves $\mathsf{SN}(A)\wedge\mathsf{SN}(M) \to \mathsf{SN}(AM)$ for terms $A^{\rho\to\sigma}$ 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 $\mathsf{SC}(AM)$, by induction hypothesis (b').

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 θ. 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 $M^{\rho\to\sigma}$ is neutral, then for any term N^ρ the term is 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. *A neutral term is strongly computable iff all its one step reducts are.*

Proof. Because of Lemma 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 type of M. For inductive types ι the assertion is proved by a side induction on neutral terms of type ι. Take, for example, we may assume $\iota = \rho^*$ If M^ι is not a constructor term, then the assertion holds by definition of SC_ι. Now let $M^\iota = \mathsf{cons}(M_1, M_2)$. Since we assume that all one step reducts of M are strongly computable, it follows, by Lemma 1 (b), that all one steps reducts of the arguments M_i are strongly computable. Hence, by main- respectively side-induction hypothesis, M_1 and M_2 are strongly computable. Let finally M be of type $\rho \to \sigma$. We show $\mathsf{SC}_\sigma(MN)$ for all strongly computable terms N by a side induction on $\mathsf{SN}(N)$. 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.

Lemma 4. *If $M[N/x]$ is strongly computable for all strongly computable terms N, then $\lambda x\, M$ is strongly computable.*

Proof. Let $M^{\rho\to\sigma}$ fulfill the assumption of the lemma, and assume $\mathsf{SC}_\rho(N)$. We have to show $\mathsf{SC}_\sigma((\lambda x\, M)N)$. Since the latter term is neutral it suffices to show that all its one step reducts are strongly computable. By Lemma 2 (a) and Lemma 1 (a) we may argue by induction on $\mathsf{SN}(M, N)$. Assume $(\lambda x\, M)N \to_1 K$. If the conversion has happened within M or N, then we may use the induction hypothesis. If not, then we must have $K = M[N/x]$ which is strongly computable by assumption.

Proposition 1. *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 θ such that $\theta(x)$ is strongly computable for all variables x in the domain of θ. For variables and constants this 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 4. The proposition now follows with the empty substitution and Lemma 2 (a).

Proposition 2. *Gödel's system T is strongly normalizing.*

Proof. By proposition 1 it suffices to show that all constants, i.e. the recursors are strongly normalizing. We have to show that $\mathsf{R}_{\sigma^*,\rho}MNL$ is strongly computable for all strongly computable terms M, N, L of appropriate types. Using Lemma 2 (a) and Lemma 1 (a) we argue by induction on $\mathsf{SN}(M, N, L)$. We also use a side induction on L. Since $\mathsf{R}_{\sigma^*,\rho}MNL$ is a neutral term it suffices, by Lemma 3, to show $\mathsf{SC}_\rho(K)$ for all K such that $\mathsf{R}_{\sigma^*,\rho}MNL \to_1 K$. If the conversion took place within one of the terms in M, N, L, then we use the main induction hypothesis and Lemma 1 (a). Otherwise the visible recursor was involved

in the conversion. If $L = []$ and $K = M$, then we are done since, by assumption, M is strongly computable. If $L = \mathsf{cons}(H, T)$ and $K = NHT(\mathsf{R}_{\sigma^*,\rho}MNT)$, then H and T are strongly computable, in particular $\mathsf{SC}_\rho(\mathsf{R}_{\sigma^*,\rho}MNT)$ by the side induction hypothesis. Again it follows that K is strongly computable.

4 Stratified terms

For the rest of this paper we will restrict constant conversion rules to the form

$$cP_1 \dots P_n \mapsto R$$

where $\mathsf{FV}(cP_1 \dots P_n) \subseteq \mathsf{FV}(R)$ and the P_i are *constructor patterns*, i.e. terms built from variables by constructor application only. All examples of constant conversion rules we have seen so far are of this form.

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 just use them as a technical tool in our termination proof based on strict semantics (section 5).

Proposition 3. *Every stratified term is strongly normalizing.*

Proof. We proceed similarly as in the proof of proposition 1. By induction on the stratification of M we show that $M\theta$ is strongly computable for every substitution θ such that $\theta(x)$ is strongly computable for all variables $x \in \mathsf{FV}(M)$. Only the case that M is a constant is interesting. All other cases are as in proposition 1, that is, we use the induction hypothesis. Let c be a constant that takes n arguments. We have to show that $cM_1 \dots M_n$ is strongly computable for all strongly computable M_i. We do a side induction on the strong normalizability of the M_i (using Lemma 2 (a)). Since $cM_1 \dots M_n$ is neutral it suffices to show 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 θ 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 follows from the strong computability of the M_i, by repeated application of Lemma 1 (b), that $\theta(x)$ is strongly computable for each $x \in \mathsf{FV}(M)$. Hence $R\theta$ is strongly computable, by the main induction hypothesis.

5 Strong normalization based on strict semantics

We will now develop a general semantic method for proving strong normalization of higher type rewrite systems. To begin with we discuss the rewrite systems we

will apply this method to: Spector's bar recursion [13] and a version of bar recursion due to Berardi, Bezem and Coquand [2].

Let us write ρ^ω for $\mathsf{nat} \to \rho$, $\mathsf{if}\, B\, \mathsf{then}\, M\, \mathsf{else}\, N$ for $\mathsf{if}\, BMN$, $|M|$ for $\mathsf{lh}\, M$, $M{*}N$ for $M +\!\!+ \langle N\rangle$ where $\langle N\rangle := \mathsf{cons}(N, [])$, and $\widehat{M}$ for $\mathsf{get}\, M$. *Spector's bar recursion in finite types* is given (for each pair of types ρ, σ) by a constant

$$\Phi : (\rho^\omega \to \mathsf{nat}) \to (\rho^* \to \sigma) \to (\rho^* \to (\rho \to \sigma) \to \sigma) \to \rho^* \to \sigma$$

with the following defining equation

$$\Phi ygh s = \mathsf{if}\, y\widehat{s} < |s|\, \mathsf{then}\, gs\, \mathsf{else}\, hs(\lambda x.\Phi ygh(s{*}x))$$

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 Ψ with an extra boolean argument in order to force evaluation of the test $y\widehat{s} < |s|$ before the subterm $\Phi ygh(s{*}x)$ may be reduced further (Vogel's trick).

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

We denote the rewrite system above by *BR*.

Berardi, Bezem and Coquand's variant of bar recursion, which is also discussed in [4], is given by

$$\begin{aligned} &\Phi\colon (\rho^\omega \to \mathsf{nat}) \to (\mathsf{nat} \to (\rho \to \mathsf{nat}) \to \rho) \to \rho^* \to \mathsf{nat} \\ &\Phi ygs = y(\lambda k.\mathsf{if}\ k < |s|\ \mathsf{then}\ s_k\ \mathsf{else}\ gk(\lambda x.\Phi yg(s{*}x))) \end{aligned}$$

where $s_k := \mathsf{get}\, s\, k$. Applying Vogel's trick again we obtain the rewrite system

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

which we call *MBR* (modified bar recursion).

The rewrite systems BR and MBR (and all rewrite systems discussed earlier) are instances of a class of rewrite systems which are distinguished by the fact that they induce a semantic interpretation of constants in a canonical way: A *functional rewrite system* is a system of constant-conversion rules

$$cP_1 \dots P_n \mapsto R$$

(P_i constructor patterns with $\mathsf{FV}(cP_1 \dots P_n) \subseteq \mathsf{FV}(R)$) 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.

Our semantic method will work for arbitrary functional rewrite systems. The idea is to deduce the termination of a term M directly from the totality of the

constants in M with respect to a strict interpretation of terms as total elements in the domain theoretic model $\hat{\mathrm{C}}$ of partial continuous functionals. The model $\hat{\mathrm{C}}$ assigns to every type ρ a Scott domain $\hat{\mathrm{C}}(\rho)$ such that $\hat{\mathrm{C}}(\rho \to \sigma) \equiv [\hat{\mathrm{C}}(\rho) \to \hat{\mathrm{C}}(\sigma)]$, the domain of continuous functions form $\hat{\mathrm{C}}(\rho)$ to $\hat{\mathrm{C}}(\sigma)$ where '$\equiv$' means 'isomorphic', and $\hat{\mathrm{C}}(\rho^*)$ is defined by the 'recursive domain equation'

$$\hat{\mathrm{C}}(\rho^*) \equiv 1 + \hat{\mathrm{C}}(\rho) \times \hat{\mathrm{C}}(\rho^*)$$

where '+' means the domain theoretic disjoint sum (adding a new bottom element). We denote the canonical injection of the one point space 1 into $\hat{\mathrm{C}}(\rho^*)$ by [] and the other canonical injection by cons. The definitions of $\hat{\mathrm{C}}(\mathsf{boole})$ and $\hat{\mathrm{C}}(\mathsf{nat})$ are similar.

The *total elements* in $\hat{\mathrm{C}}(\rho)$ are defined by recursion on ρ in the obvious way: A continuous function $f \in \hat{\mathrm{C}}(\rho \to \sigma)$ is total if $f(a)$ is total for all total arguments a. The set of total elements of $\hat{\mathrm{C}}(\rho^*)$ is given by an inductive definition: [] is total, and if $a \in \hat{\mathrm{C}}(\rho)$ and $b \in \hat{\mathrm{C}}(\sigma)$ are total, then $\mathsf{cons}(a, b)$ is total. Similar definitions apply to the other inductive types. Note that the total elements of $\hat{\mathrm{C}}(\rho^*)$ may be viewed as finite lists of total elements of $\hat{\mathrm{C}}(\rho)$ and the total elements of $\hat{\mathrm{C}}(\mathsf{boole})$ and $\hat{\mathrm{C}}(\mathsf{nat})$ are copies of the usual boolean values and the natural numbers respectively.

Let CEnv denote the domain of all *constant environments*, that is, families α assigning to each constant $c^\rho \in \mathcal{C}$ some $\alpha(c) \in \hat{\mathrm{C}}(\rho)$. Similarly, VEnv denotes the domain of all *variable environments*, i.e. families η assigning to each variable x^ρ some $\eta(x) \in \hat{\mathrm{C}}(\rho)$. For every term M^ρ we define the *strict semantics*, $[M]\colon \mathsf{CEnv} \to \mathsf{VEnv} \to \hat{\mathrm{C}}(\rho)$, by

$$\begin{aligned}
[x]^\alpha\eta &= \eta(x)\\
[c]^\alpha\eta &= \alpha(c)\\
[\lambda x\, M]^\alpha\eta(a) &= [M]^\alpha\eta^a_x\\
[MN]^\alpha\eta &= \begin{cases} [M]^\alpha\eta(a) & \text{if } a := [N]^\alpha\eta \neq \bot\\ \bot & \text{otherwise}\end{cases}\\
[\mathsf{co}(M_1, \ldots, M_k)]^\alpha\eta &= \begin{cases} \mathsf{co}(a_1, \ldots, a_k) & \text{if } a_i := [M_i]^\alpha\eta \neq \bot \text{ for all } i\\ \bot & \text{otherwise}\end{cases}
\end{aligned}$$

Lemma 5. *(a) If $\alpha(c)$ is total for all constants c in M and $\eta(x)$ is total for all $x \in \mathsf{FV}(M)$, then $[M]^\alpha\eta$ is total.*
(b) If $\alpha(c) = \bot$ for some constant c occurring in M, then $[M]^\alpha\eta = \bot$.
(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 ζ is a 'constant substitution', i.e. $\zeta(c^\rho)$ is a term of type ρ for each constant c^ρ, and $([\zeta]^\alpha\eta)(c) := [\zeta(c)]^\alpha\eta$.

Proof. Easy inductions on M.

Next we define the semantics of constants induced by their conversion rules.

We let $\bot$ denote the 'undefined' variable environment, i.e. $\bot(x) = \bot_\rho$ for all variables x^ρ.

For a vector $\boldsymbol{P}\colon\boldsymbol{\rho}$ of constructor patterns containing each variable at at most one place and $\boldsymbol{a} \in \hat{\mathrm{C}}(\boldsymbol{\rho})$ we define the $\boldsymbol{P}$-*predecessor* of $\boldsymbol{a}$, $\mathsf{pred}_{\boldsymbol{P}}(\boldsymbol{a}) \in \mathsf{VEnv}$, by recursion on the number of constructors occurring in $\boldsymbol{P}$.

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

We say $\boldsymbol{a}$ *matches* $\boldsymbol{P}$ if in the definition of $\mathsf{pred}_{\boldsymbol{P}}(\boldsymbol{\alpha})$ the last clause has never been used.

Let $\mathcal{R}$ be a functional rewrite system. For a given vector $\boldsymbol{a} \in \hat{\mathrm{C}}$ there can be at most one rule $c\boldsymbol{P} \mapsto R \in \mathcal{R}$ such that $\boldsymbol{a}$ matches $\boldsymbol{P}$, because the rules in $\mathcal{R}$ are mutually disjoint. Therefore the following operator $\Gamma_{\mathcal{R}}\colon \mathsf{CEnv} \to \mathsf{CEnv}$ is welldefined and continuous.

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

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

We now state the main the result of this paper.

Theorem 1. *Let $\mathcal{R}$ be a functional rewrite system. If $\alpha_{\mathcal{R}}(c)$ 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 $\mathcal{R}$.

Lemma 6. *Let $c\boldsymbol{P} \mapsto R \in \mathcal{R}$ be a rule, θ a substitution and η a variable environment. Set $\boldsymbol{a} := [\boldsymbol{P}\theta]^{\alpha_{\mathcal{R}}}\eta$. If $\bot \notin \boldsymbol{a}$, then $\boldsymbol{a}$ matches $\boldsymbol{P}$ and $\alpha_{\mathcal{R}}(c)(\boldsymbol{a}) = [R]^{\alpha_{\mathcal{R}}}\mathsf{pred}_{\boldsymbol{P}}(\boldsymbol{a})$.*

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

Lemma 7. *If $M \to_1 N$, then $[M]^{\alpha_{\mathcal{R}}}\eta \sqsubseteq [N]^{\alpha_{\mathcal{R}}}\eta$.*

Proof. Induction on M, where w.l.o.g. we assume $[M]^{\alpha_{\mathcal{R}}}\eta \neq \bot$.

Case $(\lambda xM)N \to_1 M[N/x]$. Setting $a := [N]^{\alpha_{\mathcal{R}}}\eta$ we have $[(\lambda xM)N]^{\alpha_{\mathcal{R}}}\eta \sqsubseteq ([\lambda xM]^{\alpha_{\mathcal{R}}}\eta)(a) = [M]^{\alpha_{\mathcal{R}}}\eta^a_x = [M[N/x]]^{\alpha_{\mathcal{R}}}\eta$, by Lemma 5 (c).

Case $c\boldsymbol{P}\theta \to_1 R\theta$ for some rule $c\boldsymbol{P} \mapsto R \in \mathcal{R}$. $[c\boldsymbol{P}\theta]^{\alpha_{\mathcal{R}}}\eta \sqsubseteq \alpha_{\mathcal{R}}(c)(\boldsymbol{a}) = [R]^{\alpha_{\mathcal{R}}}\mathsf{pred}_{\boldsymbol{P}}(\boldsymbol{a}) = [R\theta]^{\alpha_{\mathcal{R}}}\eta$. The last two equations hold by Lemmas 6 and 5 (c).

All other cases (i.e. conversion of a proper subterm) follow immediately from the induction hypothesis and the fact that constructors and application are interpreted strictly.

We now introduce a stratified variant $\mathcal{R}_\omega$ of $\mathcal{R}$. Let $\mathcal{C}$ be the set of constants of $\mathcal{R}$. For each $c \in \mathcal{C}$ and every natural number n 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

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

Clearly $\mathcal{R}_\omega$ is again a functional rewrite system. Furthermore all constants c_n are stratified (induction on n). 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 8. *If $A \preceq M$ and A contains no constant of the form c_0, then to every $\mathcal{C}$-term M' such that $M \to_1 M'$ there is a $\mathcal{C}_\omega$-term A' such that $A \to_1 A'$ and $A' \preceq M'$.*

Proof. Easy induction on M.

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

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

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

Lemma 9. *$\alpha_{\mathcal{R}}(c) = \bigsqcup_{n\in\omega} \alpha_{\mathcal{R}_\omega}(c_n)$ and $\alpha_{\mathcal{R}_\omega}(c_n) \sqsubseteq \alpha_{\mathcal{R}_\omega}(c_{n+1})$ for every constant $c \in \mathcal{C}$.*

Proof. Set $\alpha_n(c) := \alpha_{\mathcal{R}_\omega}(c_n)$. Since $\alpha_{\mathcal{R}} = \bigsqcup_{n\in\omega} \Gamma^n_{\mathcal{R}}\bot$ and $\Gamma^n_{\mathcal{R}}\bot \sqsubseteq \Gamma^{n+1}_{\mathcal{R}}\bot$, it suffices to show $\alpha_n = \Gamma^n_{\mathcal{R}}\bot$ for all n. We prove this by induction on n. For $n = 0$ both sides are $\bot$.

$$\begin{aligned}
\alpha_{n+1}(c)(\boldsymbol{a}) &= \alpha_{\mathcal{R}_\omega}(c_{n+1})(\boldsymbol{a}) \\
&= \bigsqcup\{[R_{[n]}]^{\alpha_{\mathcal{R}_\omega}}\,\mathsf{pred}_{\boldsymbol{P}}(\boldsymbol{a}) \mid c\boldsymbol{P} \mapsto R \in \mathcal{R},\ \boldsymbol{a} \text{ matches } \boldsymbol{P}\} \\
&= \bigsqcup\{[R]^{\alpha_n}\,\mathsf{pred}_{\boldsymbol{P}}(\boldsymbol{a}) \mid c\boldsymbol{P} \mapsto R \in \mathcal{R},\ \boldsymbol{a} \text{ matches } \boldsymbol{P}\} \text{ (Lemma 5 (d))} \\
&= \Gamma_{\mathcal{R}}(\alpha_n)(c)(\boldsymbol{a}) \\
&= (\Gamma^{n+1}_{\mathcal{R}}\bot)(c)(\boldsymbol{a}) \text{ (induction hypothesis)}
\end{aligned}$$

Lemma 10. *$[M]^{\alpha_{\mathcal{R}}}\eta = \bigsqcup_{n\in\omega} [M_{[n]}]^{\alpha_{\mathcal{R}_\omega}}\eta$ for every $\mathcal{C}$-term M and every variable environment η.*

Proof. Set, as in the previous proof, $\alpha_n(c) := \alpha_{\mathcal{R}_\omega}(c_n)$. By Lemma 9 we have $\alpha_{\mathcal{R}} = \bigsqcup_{n\in\omega} \alpha_n$ where $\alpha_n \sqsubseteq \alpha_{n+1}$. Hence, because $[M]$ is a continuous function,

$$[M]^{\alpha_{\mathcal{R}}}\eta = \bigsqcup_{n\in\omega} [M]^{\alpha_n}\eta \overset{5\,(\mathrm{d})}{=} \bigsqcup_{n\in\omega} [M_{[n]}]^{\alpha_{\mathcal{R}_\omega}}\eta$$

Now we are ready to prove Theorem 1. Let M be a ($\mathcal{C}$-)term such that $\alpha_{\mathcal{R}}(c)$ is total for every constant in M. Let η be any total environment. Then $[M]^{\alpha_{\mathcal{R}}}\eta$ is total, by Lemma 5 (a), and therefore different from $\bot$. By Lemma 10 it follows that there is some n such that $[M_{[n]}]^{\alpha_{\mathcal{R}_\omega}}\eta \neq \bot$. Clearly $M_{[n]} \preceq M$. Therefore it suffices to show that whenever $A \preceq N$ and $[A]^{\alpha_{\mathcal{R}_\omega}}\eta \neq \bot$, then N is strongly normalizing. We prove this by induction on the strong normalizability of the $\mathcal{C}_\omega$-term A, using proposition 3. We need to show that all one step reducts of N

are strongly normalizing. So, assume $N \to_1 N'$. Since $[A]^{\alpha\mathcal{R}_\omega}\eta \neq \bot$ we know, by Lemma 5 (b), that A does not contain a constant of the form c_0. It follows with Lemma 8 that $A \to_1 A'$ with $A' \preceq N'$ for some $\mathcal{C}_\omega$-term A'. By Lemma 7 $[A']^{\alpha\mathcal{R}_\omega}\eta \neq \bot$, hence we can apply the induction hypothesis to A' and N'.

Let us now apply Theorem 1 to prove strong normalization for bar recursion.

Theorem 2. *Gödel's system T extended by BR and MBR is strongly normalizing.*

Proof. We only carry out the proof for MBR. For BR the proof is similar and slightly simpler. Our strict semantics interprets the constants Φ and Ψ of MBR as continuous functionals φ and ψ which satisfy for *total arguments* y, g, s, k (in $\hat{\mathrm{C}}$) the equations

$$\begin{aligned}
\varphi ygs &= y(\lambda k.\psi ygsk(k < |s|)) \\
\psi ygsk\mathsf{T} &= s_k \\
\psi ygsk\mathsf{F} &= \begin{cases} gk(\lambda x.\varphi yg(s{*}x)) & \text{if } \varphi yg(s{*}x) \neq \bot \text{ for some } x \\ \bot & \text{otherwise} \end{cases}
\end{aligned}$$

By a continuity argument one shows that for every total y the the binary relation $\gg_y$ on the total elements of type ρ^* defined by

$$s \gg_y t :\equiv y(\lambda k.\mathsf{if}\ k < |s|\ \mathsf{then}\ s_k\ \mathsf{else}\ \bot) = \bot \wedge s{*}a = t \text{ for some total } a$$

is wellfounded. Now the totality of φygs and ψygs for total y, g, s can be proven easily by induction on $\gg_y$. With Theorem 1 strong normalization follows.

Remarks. Tait [14], Vogel [17], Luckhardt [10] and Bezem [5] proved strong normalization for BR formulated in a combinatorial calculus. Our result is slightly stronger since we work in a λ-calculus framework which allows more reductions. Strong normalization for MBR is completely new. Further interesting rewrite rules where Theorem 1 applies to are realizers of the negative- and A-translations of the axiom schemes of countable choice [2] and open induction [3].

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 $\hat{\mathrm{C}}(\rho)$ has the same logical complexity as, say the definition of strong computability for infinite terms of type ρ.

References

1. H. Barendregt. Lambda calculi with types. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Clarendon Press, Oxford, 1992.
2. S. Berardi, M. Bezem, and T. Coquand. On the computational content of the axiom of choice. *Journal of Symbolic Logic*, 63(2):600–622, 1998.

3. U. Berger. A computational interpretation of open induction. In F. Titsworth, editor, *Proceedings of the Nineteenth Annual IEEE Symposium on Logic in Computer Science*, pages 326–334. IEEE Computer Society, 2004.
4. U. Berger and P. Oliva. Modified bar recursion and classical dependent choice. In *Logic Colloquium 2001*. Springer, to appear.
5. M. Bezem. Strong normalization of barrecursive terms without using infinite terms. *Archive for Mathematical Logic*, 25:175–181, 1985.
6. F Blanqui, J-P. Jouannaud, and M. Okada. The calculus of algebraic constructions. In P. Narendran and M. Rusinowitch, editors, *Proceedings of RTA'99*, number 1631 in LNCS, pages 301–316. Springer Verlag, Berlin, Heidelberg, New York, 1999.
7. T. Coquand. *Une théorie des constructions*. PhD thesis, Université Paris VII, 1985.
8. H. Geuvers and M.J. Nederhof. A modular proof of strong normalization for the calculus of constructions. *Journal of Functional Programming*, 1(2):155–189, 1991.
9. J-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 63–92. North–Holland, Amsterdam, 1971.
10. H. Luckhardt. *Extensional Gödel Functional Interpretation – A Consistency Proof of Classical Analysis*, volume 306 of *Lecture Notes in Mathematics*. Springer, 1973.
11. R. Matthes. Monotone inductive and coinductive constructors of rank 2. In L Fribourg, editor, *Computer Science Logic (Proceedings of the Fifteenth CSL Conference)*, number 2142 in LNCS, pages 600–615. Springer Verlag, Berlin, Heidelberg, New York, 2001.
12. G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.
13. C. Spector. Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In F. D. E. Dekker, editor, *Recursive Function Theory: Proc. Symposia in Pure Mathematics*, volume 5, pages 1–27. American Mathematical Society, Providence, Rhode Island, 1962.
14. W.W. Tait. Normal form theorem for barrecursive functions of finite type. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 353–367. North–Holland, Amsterdam, 1971.
15. A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer, 1973.
16. J. van de Pol and H. Schwichtenberg. Strict functionals for termination proofs. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Typed Lambda Calculi and Applications*, volume 902 of *LNCS*, pages 350–364. Springer Verlag, Berlin, Heidelberg, New York, 1995.
17. H. Vogel. Ein starker Normalisationssatz für die barrekursiven Funktionale. *Archive for Mathematical Logic*, 18:81–84, 1985.