Math. Struct. in Comp. Science (2006), *vol.* 16, *pp.* 751–762. © 2006 Cambridge University Press
doi:10.1017/S0960129506005457 Printed in the United Kingdom

Continuous semantics for strong normalisation

ULRICH BERGER

Department of Computer Science, University of Wales Swansea, Swansea, SA2 8PP, U.K.
Email: u.berger@swan.ac.uk

Received 23 September 2005; revised 21 December 2005

We prove a general strong normalisation 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 barrecursion for which strong normalisation was hitherto unknown.

1. Introduction

The problem of proving strong normalisation for typed λ-calculi and higher type rewrite systems has been studied extensively in the literature: see, for example, Tait (1971), Girard (1972), Troelstra (1973), Luckhardt (1973), Vogel (1976), Coquand (1985), Bezem (1985), Geuvers and Nederhof (1991), Barendregt (1992), van de Pol and Schwichtenberg (1995), McAllester *et al.* (1995), Barbanera *et al.* (1997), Blanqui *et al.* (1999) and Matthes (2001).

In this paper we present a new method for proving strong normalisation of higher type rewrite systems based on a strict domain-theoretic semantics. The idea is similar to Plotkin's adequacy proof for PCF (Plotkin 1977): a suitable interpretation of terms is given in a domain-theoretic model, and then a continuity argument is used to show that any term not denoting $\perp$ strongly normalises. The problem of showing that a term M does not denote $\perp$ can in turn be solved by proving that all constants in M have a total value, using the intuitive argument that led to the introduction of the constant in the first place. The main difference compared with Plotkin's result is that Plotkin considers terms with a general fixed-point operator, for which, of course, only *weak* normalisation can be proved, while we work with recursion schemes based on pattern matching, and we prove *strong* normalisation.

Another new aspect of our method is that it allows for a modular normalisation proof. First we prove strong normalisation for the underlying typed λ-calculus with *stratified constants* only, that is, constants with rewrite rules that do not involve recursion. This can be done by the well-known computability method (due to Tait, Troelstra, Martin-Löf and others). Then we use a continuity argument to lift strong normalisation to recursively defined constants that have a total value.

We will apply our results to a λ-calculus formulation of Gödel's system T extended by two versions of barrecursion in finite types: Spector's original version (Spector 1962), and a version due to Berardi, Bezem and Coquand (Berardi *et al.* 1998). For this system strong normalisation was hitherto unknown.

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press

2. Higher type rewrite systems

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 $\vec{\rho} \to \sigma$ for $\rho_1 \to \ldots \to \rho_n \to \sigma$ where $\to$ associates to the right. Types that are not function types, that is, 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 parametrised by a set $\mathscr{C}$ of *typed constants*. *Terms* are constructed from *typed variables*, x^ρ, and constants, $c^\rho \in \mathscr{C}$, 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 does not cause ambiguities. Instead of M^ρ, we will sometimes write $M : \rho$. We let the symbol co range over the constructors 0, S, $[]$ and cons. In a constructor term $\mathsf{co}(\vec{M})^\iota$ the terms $\vec{M}$ are called *arguments*.

β-conversion is defined as usual by

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

where $M[N/x]$ means the usual substitution of every free occurrence of x in M by N, with renaming of bound variables in M if necessary. More generally, we consider substitutions θ, that is, type preserving mappings from variables to terms, and define $M\theta$ as the simultaneous replacement in M of x by $\theta(x)$, with renaming of bound variables in M if necessary.

The operational meaning of a constant $c \in \mathscr{C}$ is determined by a number n, called the *arity* of c, such that the type of c can be written in the form $\rho_1 \to \ldots \to \rho_n \to \sigma$, and a finite set of *rewrite rules* of the form

$$c\, P_1^{\rho_1} \ldots P_n^{\rho_n} \;\mapsto\; R^\sigma$$

where $\mathsf{FV}(cP_1 \ldots P_n) \supseteq \mathsf{FV}(R)$ and the P_i are *constructor patterns*, that is, terms built from variables by constructor application only. Furthermore, we require the rules to be *left linear*, that is, a variable may occur at most once in the left-hand side of a rule, and *mutually disjoint*, that is, the left-hand sides of two different rules must be non-unifiable.

As an example, consider the constants $\mathsf{if} : \mathsf{boole} \to \rho \to \rho \to \rho$, $< : \mathsf{nat} \to \mathsf{nat} \to \mathsf{boole}$, $\mathsf{lh} : \rho^* \to \mathsf{nat}$, $\mathsf{get} : \rho^* \to \mathsf{nat} \to \rho$, $+\!\!+ : \rho^* \to \rho^* \to \rho^*$, and the rewrite rules

$$\begin{aligned}
\mathsf{if}\,\mathsf{T}\,x\,y &\;\mapsto\; x\\
\mathsf{if}\,\mathsf{F}\,x\,y &\;\mapsto\; y\\
n < 0 &\;\mapsto\; \mathsf{F}\\
0 < \mathsf{S}(m) &\;\mapsto\; \mathsf{T}\\
\mathsf{S}(n) < \mathsf{S}(m) &\;\mapsto\; n < m\\
\mathsf{lh}\,[] &\;\mapsto\; 0\\
\mathsf{lh}\,\mathsf{cons}(x,s) &\;\mapsto\; \mathsf{S}(\mathsf{lh}(s))\\
\mathsf{get}\,[]\,n &\;\mapsto\; 0^\rho \quad \text{(some closed term of type } \rho)\\
\mathsf{get}\,\mathsf{cons}(x,s)\,0 &\;\mapsto\; x
\end{aligned}$$

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press

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

These are examples of *primitive recursion in higher types*. Gödel's system T summarises 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 rewrite rules

$$\begin{aligned} \mathsf{R}_{\mathsf{nat},\rho} xy\mathsf{0} &\mapsto x \\ \mathsf{R}_{\mathsf{nat},\rho} xy\mathsf{S}(z) &\mapsto yz(\mathsf{R}_{\mathsf{nat},\rho} xyz)\,. \end{aligned}$$

Similar rules can be introduced for recursion constants for the other inductive types.

3. Proving strong normalisation using strong computability

In the following we fix a higher type rewrite system, that is, a system $\mathscr{R}$ of rewrite rules (satisfying the general conditions formulated in Section 2). We write $M \to_1 N$ (one-step reduction) if N is obtained from M by replacing one subterm occurrence of the left-hand side of a β-conversion or a substitution instance of a rewrite rule in $\mathscr{R}$ by the corresponding right-hand side. We say a term is M *strongly normalising*, $\mathsf{SN}(M)$, if M is in the accessible part of the relation $\to_1$, that is, there is no infinite reduction sequence $M \to_1 M' \to_1 M'' \to_1 \ldots$. Equivalently, SN can be defined inductively by the rule

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

We say $\mathscr{R}$ is strongly normalising if every term is strongly normalising with respect to $\mathscr{R}$.

We now 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 $\iota = \rho^*$; the rules are similar for boole and nat):

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

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

$\mathsf{SC}_{\rho\to\sigma}$ is defined explicitly 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) We use 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, the induction hypothesis is used.
(b) This part is obvious. □

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press

Lemma 2.

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

Proof. In order to get the proof through, we strengthen part (b) to:

(b′) $\mathsf{SN}(A) \to \mathsf{SC}_\rho(A)$ for terms A of 'variable head', that is, terms A of the form $x\vec{M}$.

We use induction on ρ:

(a) If ρ is an inductive type, the implication follows easily by a side induction on the definition of $\mathsf{SC}_\rho(M)$, possibly using the main induction hypothesis. For the case $\rho \to \sigma$, we assume $\mathsf{SC}_{\rho\to\sigma}(M)$. By induction hypothesis (b′), we have $\mathsf{SC}(x^\rho)$. Hence $\mathsf{SC}_\sigma(Mx)$. By induction hypothesis (a), $\mathsf{SN}(Mx)$. Hence $\mathsf{SN}(M)$.
(b′) Let A be a strongly normalising term with variable head. If A has an inductive type, 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 the 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 say a term is *neutral* if it is of the form $uL_1 \dots L_k$ where u is a β-redex (that is, of the form $(\lambda x\, K)L$), or a variable, or a constant of arity $\leqslant k$. 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 reducing either M or N (the latter property, which will be used in Lemma 3 below, is ensured by the above condition on the constant's arity). However, neutral terms are *not* closed under one-step reduction.

Lemma 3. A neutral term is strongly computable if and only if 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 holds by definition of SC_ι. Now let 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 the 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^σ 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 Lemmas 2 and 1(a), we may argue by induction on $\mathsf{SN}(M, N)$. Assume $(\lambda x\, M)N \to_1 K$. If the reduction has happened within M or N, we

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press

may use the induction hypothesis. If not, we must have $K = M[N/x]$, which is strongly computable by assumption. □

Theorem 1. A term containing only strongly computable constants is strongly normalising.

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. 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 theorem now follows with the empty substitution and Lemma 2. □

Using Theorem 1, one can show that Gödel's system T is strongly normalising by proving that all constants, that is, the recursors, are strongly computable. To this end, one shows that $\mathsf{R}_{\sigma^*,\rho}MNL$ is strongly computable for all strongly computable terms M, N, L of appropriate types, which can be done by induction on $\mathsf{SN}(M, N, L)$ and a side induction on the built up of L using Lemmas 1, 2, 3. We will not give the details of this proof (which is essentially due to Tait, Troelstra and Martin-Löf), but will instead prove strong normalisation for a rather restricted class of terms, which, however, will suffice to infer strong normalisation for Gödel's T and much stronger systems, when combined with the semantic method of the next section.

The set of *stratified terms* is defined inductively as follows:

— Every variable is stratified.
— A constant c is stratified if for every rule $cP_1 \dots P_n \mapsto R$ in $\mathscr{R}$ the term R is stratified.
— A composite term is stratified if all its immediate subterms are.

Clearly, a term is stratified if and only if it contains stratified constants only.

Note that stratification is a severe restriction. For example, any constant with a recursive rewrite rule, that is, a rule $cP_1 \dots P_n \mapsto R$ where c occurs in R, is not stratified.

Theorem 2. Every stratified term is strongly normalising.

Proof. We follow a similar line to that used in the proof of Theorem 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 in which M is a constant is interesting, since all other cases are as in Theorem 1. Let c be a constant of arity n. By unfolding the definition of strong computability for function types, we see that we have to show that $cM_1 \dots M_n$ is strongly computable for all strongly computable terms M_i of appropriate types. We now do a side induction on the strong normalisability 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, by the main induction hypothesis, $R\theta$ is strongly computable. □

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press

4. Strong normalisation based on strict semantics

We will now develop a general semantic method for proving strong normalisation of higher type rewrite systems. The main idea is to infer the *operational* property of strong normalisation directly from the totality of the constants in a suitable *denotational* model. The model is based on the domain-theoretic structure $\hat{\mathrm{C}}$ of *partial continuous functionals* (for background on domain theory see, for example, Stoltenberg-Hansen *et al.* (1993) and Abramsky and Jung (1994)), which 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 from $\hat{\mathrm{C}}(\rho)$ to $\hat{\mathrm{C}}(\sigma)$ ('$\equiv$' means 'isomorphic'), and, for an inductive type ι, $\hat{\mathrm{C}}(\iota)$ is defined by an appropriate 'recursive domain equation'. For example, $\hat{\mathrm{C}}(\rho^*)$ is defined as the least (more precisely, initial) solution of the 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 use [] to denote the canonical injection of the one point space 1 into $\hat{\mathrm{C}}(\rho^*)$, and cons for the other canonical injection. Being a Scott domain, $\hat{\mathrm{C}}(\rho)$ carries a partial order, $\sqsubseteq_\rho$, contains a least element $\bot_\rho$, and has suprema of directed subsets. Furthermore, a function $f : \hat{\mathrm{C}}(\rho) \to \hat{\mathrm{C}}(\sigma)$ is continuous if and only if it is monotone and preserves suprema of directed sets.

The *total elements* in $\hat{\mathrm{C}}(\rho)$ are defined by recursion on ρ: 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 $\hat{\mathrm{C}}(\iota)$, ι inductive, is defined inductively. For example, [] 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. 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 $c^\rho \in \mathscr{C}$ some $\alpha(c) \in \hat{\mathrm{C}}(\rho)$. Similarly, VEnv denotes the domain of all *variable environments*, that is, 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*, $[M] : \mathsf{CEnv} \to \mathsf{VEnv} \to \hat{\mathrm{C}}(\rho)$, by

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

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 $([\theta]^\alpha\eta)(x) := [\theta(x)]^\alpha\eta$.

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press

(d) $[M]^{[\zeta]^\alpha\eta}\eta = [M\zeta]^\alpha\eta$ where ζ is a 'constant substitution', that is, $\zeta(c^\rho)$ is a term of type ρ for each constant c^ρ, and $([\zeta]^\alpha\eta)(c) := [\zeta(c)]^\alpha\eta$.

Proof. The proof is by easy inductions on M. □

Now we define the semantics of constants induced by a given system $\mathscr{R}$ of rewrite rules. Let $\bot$ denote the 'undefined' variable environment, that is, $\bot(x^\rho) = \bot_\rho$ for all variables x. Hence, $\bot^{\vec{a}}_{\vec{x}}$ is an environment with $\bot^{\vec{a}}_{\vec{x}}(x_i) = a_i$ and $\bot^{\vec{a}}_{\vec{x}}(y) = \bot$ if $y \notin \vec{x}$. For a vector $\vec{P} : \vec{\rho}$ of constructor patterns and $\vec{a} \in \hat{\mathrm{C}}(\vec{\rho})$, we define the $\vec{P}$-*predecessor* of $\vec{a}$, $\mathsf{pred}_{\vec{P}}(\vec{a}) \in \mathsf{VEnv}$, by recursion on the number of constructors in $\vec{P}$. Recall that a constructor pattern is either a variable, or of the form $\mathsf{co}(\vec{Q})$ where co is a constructor and $\vec{Q}$ is a vector of variable disjoint constructor patterns. Then

$$\begin{aligned}
\mathsf{pred}_{\vec{x}}(\vec{a}) &= \bot^{\vec{a}}_{\vec{x}} \\
\mathsf{pred}_{\vec{x},\mathsf{co}(\vec{Q}),\vec{P}}(\vec{a},\mathsf{co}(\vec{b}),\vec{c}) &= \mathsf{pred}_{\vec{x},\vec{Q},\vec{P}}(\vec{a},\vec{b},\vec{c}) \\
\mathsf{pred}_{\vec{x},\mathsf{co}(\vec{Q}),\vec{P}}(\vec{a},b,\vec{c}) &= \bot \quad \text{otherwise.}
\end{aligned}$$

We say $\vec{a}$ *matches* a vector $\vec{P}$ of constructor patterns, or a rewrite rule $c\vec{P} \mapsto R$, if in the definition of $\mathsf{pred}_{\vec{P}}(\vec{a})$, the last clause has not been used. From the disjointness assumption on the rules in $\mathscr{R}$, it follows that $\vec{a} \in \hat{\mathrm{C}}$ can match at most one rule $c\vec{P} \mapsto R \in \mathscr{R}$. Therefore the operator $\Gamma_{\mathscr{R}} : \mathsf{CEnv} \to \mathsf{CEnv}$ is well defined and continuous:

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

We define the constant environment $\alpha_{\mathscr{R}}$ as the least fixed point of $\Gamma_{\mathscr{R}}$. The main aim of this section is to prove the following theorem.

Theorem 3. If $[M]^{\alpha_{\mathscr{R}}}\eta \neq \bot$, then M is strongly normalising.

This theorem, together with Lemma 5 (a) and the fact that total objects are different from $\bot$, immediately yields the following (very practical) sufficient condition for strong normalisation.

Theorem 4. If $\alpha_{\mathscr{R}}(c)$ is total for every constant in M, then M is strongly normalising.

We will prove Theorem 3 through a succession of lemmas and definitions.

Lemma 6. Let $\vec{P}$ be a vector of constructor patterns in which no variable occurs twice (as required in the left-hand sides of rewrite rules). Let $\eta \in \mathsf{VEnv}$ and set $\vec{a} := [\vec{P}]\eta$. If $\bot \notin \vec{a}$, then $\vec{a}$ matches $\vec{P}$ and $\mathsf{pred}_{\vec{P}}(\vec{a}) \sqsubseteq \eta$, where environments are ordered pointwise.

Proof. The proof is by an easy induction on the number of constructors in $\vec{P}$. □

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

Proof. We use induction on M, where we assume, without loss of generality, that $[M]^{\alpha_{\mathscr{R}}}\eta \neq \bot$.

— *Case*: $(\lambda x M)N \to_1 M[N/x]$. Set $a := [N]^{\alpha_{\mathscr{R}}}\eta$. Then

$$[(\lambda x M)N]^{\alpha_{\mathscr{R}}}\eta \sqsubseteq ([\lambda x M]^{\alpha_{\mathscr{R}}}\eta)(a) = [M]^{\alpha_{\mathscr{R}}}\eta^a_x = [M[N/x]]^{\alpha_{\mathscr{R}}}\eta\,,$$

by Lemma 5 (c).

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press

— *Case*: $c\vec{P}\theta \to_1 R\theta$ where $c\vec{P} \mapsto R \in \mathscr{R}$. Set $\vec{a} := [\vec{P}\theta]^{\alpha_{\mathscr{R}}}\eta = [\vec{P}]([\theta]^{\alpha_{\mathscr{R}}}\eta)$, by Lemma 5 (c). We must have $\bot \notin \vec{a}$ since otherwise $[c\vec{P}\theta]^{\alpha_{\mathscr{R}}}\eta = \bot$, because of strictness. Hence, by Lemmas 6 and 5 (c), $[c\vec{P}\theta]^{\alpha_{\mathscr{R}}}\eta \sqsubseteq \alpha_{\mathscr{R}}(c)(\vec{a}) = [R]^{\alpha_{\mathscr{R}}}\mathsf{pred}_{\vec{P}}(\vec{a}) \sqsubseteq [R]^{\alpha_{\mathscr{R}}}([\theta]^{\alpha_{\mathscr{R}}}\eta) = [R\theta]^{\alpha_{\mathscr{R}}}\eta$.

All other cases (rewriting of a proper subterm) follow immediately from the induction hypothesis and the strict interpretation of function and constructor application. □

We now introduce a stratified variant $\mathscr{R}_\omega$ of $\mathscr{R}$. Let $\mathscr{C}$ be the set of constants of $\mathscr{R}$. We define a new set of constants $\mathscr{C}_\omega := \{c_n^\rho \mid c^\rho \in \mathscr{C},\ n \in \mathbb{N}\}$ (c_n is just c with label n attached). For any $\mathscr{C}$-term M let $M_{[n]}$ be the $\mathscr{C}_\omega$-term obtained from M by replacing every occurring constant c by c_n. We set

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

Clearly, $\mathscr{R}_\omega$ inherits from $\mathscr{R}$ the linearity and disjointness properties formulated in Section 2.

Lemma 8. $\mathscr{R}_\omega$ is strongly normalising.

Proof. By induction on n, one shows that all constants c_n and hence all $\mathscr{C}_\omega$-terms are stratified. Therefore Theorem 2 applies. □

We write $A \preceq M$ if M is a $\mathscr{C}$-term and A is a $\mathscr{C}_\omega$-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 9. If $A \preceq M$ and A contains no constant of the form c_0, then to every $\mathscr{C}$-term M' such that $M \to_1 M'$ there is a $\mathscr{C}_\omega$-term A' such that $A \to_1 A'$ and $A' \preceq M'$.

Proof. The proof is by an easy induction on M. □

Let $\alpha_{\mathscr{R}_\omega}$ be the $\mathscr{C}_\omega$-constant environment determined by $\mathscr{R}_\omega$. Hence

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

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

Lemma 10. $\alpha_{\mathscr{R}}(c) = \bigsqcup_{n\in\omega} \alpha_{\mathscr{R}_\omega}(c_n)$ and $\alpha_{\mathscr{R}_\omega}(c_n) \sqsubseteq \alpha_{\mathscr{R}_\omega}(c_{n+1})$ for every constant $c \in \mathscr{C}$.

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

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

□

Lemma 11. $[M]^{\alpha_{\mathscr{R}}}\eta = \bigsqcup_{n\in\omega}[M_{[n]}]^{\alpha_{\mathscr{R}_\omega}}\eta$ for every $\mathscr{C}$-term M and every variable environment η.

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press

Proof. Set, as in the proof of Lemma 10, $\alpha_n(c) := \alpha_{\mathscr{R}_\omega}(c_n)$. By Lemma 10, we have $\alpha_{\mathscr{R}} = \bigsqcup_{n\in\omega} \alpha_n$ where $\alpha_n \sqsubseteq \alpha_{n+1}$. Hence, because $[M]$ is a continuous function,

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

Now we are ready to prove Theorem 3. Let M be a ($\mathscr{C}$-)term such that $[M]^{\alpha_{\mathscr{R}}}\eta \neq \bot$. By Lemma 11, it follows that there is some n such that $[M_{[n]}]^{\alpha_{\mathscr{R}_\omega}}\eta \neq \bot$. Clearly, $M_{[n]} \preceq M$. Therefore it suffices to show that whenever $A \preceq N$ and $[A]^{\alpha_{\mathscr{R}_\omega}}\eta \neq \bot$, we have N is strongly normalising. We prove this by induction on the strong normalisability of the $\mathscr{C}_\omega$-term A, using Theorem 2. We need to show that all one-step reducts of N are strongly normalising. So, assume $N \to_1 N'$. Since $[A]^{\alpha_{\mathscr{R}_\omega}}\eta \neq \bot$, we know, by Lemma 5(b), that A does not contain a constant of the form c_0. It follows with Lemma 9 that $A \to_1 A'$ with $A' \preceq N'$ for some $\mathscr{C}_\omega$-term A'. By Lemma 7, $[A']^{\alpha_{\mathscr{R}_\omega}}\eta \neq \bot$, so we can apply the induction hypothesis to A' and N'.

5. Applications

In the light of Theorem 4, strong normalisation of Gödel's system T is almost obvious. It suffices to show that the primitive recursion operators $\mathsf{R}_{\iota,\rho}$ have a total value. Let, for example, $f := \alpha_{\mathscr{R}_T}(\mathsf{R}_{\mathsf{nat},\rho})$. Then for total arguments $n\colon\mathsf{nat}$, $a\colon\rho$ and $g\colon\mathsf{nat}\to\rho\to\rho$

$$f(a,g,0) = a,\ \ f(a,g,n+1) = g(n,f(a,g,n))\,.$$

Hence, by a straightforward induction on n, it follows that $f(a,g,n)$ is total for all total arguments a,g,n, that is, f is total. For the variants of barrecursion in higher types, which we describe next, the strong normalisation proof via Theorem 4 will be very similar.

The rewrite systems we consider now are extensions of system T, which we assume to include the constants if, $<$, lh, get and $+\!\!+$ introduced in Section 2. To improve readability, we write ρ^ω for $\mathsf{nat}\to\rho$, $\mathsf{if}\,B\,\mathsf{then}\,M\,\mathsf{else}\,N$ for $\mathsf{if}\,BMN$, $|M|$ for $\mathsf{lh}\,M$, $\widehat{M}$ for $\mathsf{get}\,M$ and $M{*}N$ for $M +\!\!+ \langle N\rangle$ where $\langle N\rangle := \mathsf{cons}(N,[])$. *Spector's barrecursion 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:

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

Turning this into a rewrite rule would clearly not be strongly normalising. 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 $\widehat{ys} < |s|$ before the subterm $\Phi ygh(s{*}x)$ may be reduced further (Vogel's trick):

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

We call the rewrite system above BR.

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press

Berardi, Bezem and Coquand's variant of barrecursion (Berardi *et al.* 1998), which is also discussed in Berger and Oliva (2005), is given by

$$\Phi : (\rho^\omega \to \mathsf{nat}) \to (\mathsf{nat} \to (\rho \to \mathsf{nat}) \to \rho) \to \rho^* \to \mathsf{nat}$$
$$\Phi y g s = y(\lambda k.\mathsf{if}\ k < |s|\ \mathsf{then}\ \mathsf{get}\, s\, k\ \mathsf{else}\ g k (\lambda x.\Phi y g (s{*}x))).$$

Applying Vogel's trick again, we obtain the rewrite system

$$\begin{aligned} \Phi y g s &\mapsto y(\lambda k.\Psi y g s k (k < |s|)) \\ \Psi y g s k \mathsf{T} &\mapsto \mathsf{get}\, s\, k \\ \Psi y g s k \mathsf{F} &\mapsto g k (\lambda x.\Phi y g (s{*}x)), \end{aligned}$$

which we call *MBR* (modified barrecursion), following Berger and Oliva (2005).

We now apply Theorem 4 to prove strong normalisation for barrecursion.

Theorem 5. Gödel's system T extended by *BR* and *MBR* is strongly normalising.

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 that for *total* arguments y, g, s, k (in $\hat{\mathrm{C}}$) satisfy the equations

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

By a continuity argument, one shows that for every total y, the 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 well founded. Now the totality of $\varphi(y,g,s)$ and $\psi(y,g,s)$ for total y, g, s can be proved easily by $\gg_y$-induction on s. Strong normalisation then follows from Theorem 4. □

Remark. In Tait (1971), Vogel (1976), Luckhardt (1973) and Bezem (1985), strong normalisation is proved for *BR* formulated in a combinatorial calculus. Our result is slightly stronger since we work in a λ-calculus framework that allows more reductions. Strong normalisation for *MBR* is completely new. Further interesting rewrite rules that Theorem 4 applies to are realisers of the negative- and A-translations of the axiom schemes of countable choice (Berardi *et al.* 1998) and open induction (Berger 2004).

6. Conclusions and further work

We have introduced a semantic method for proving strong normalisation of higher type rewrite systems by showing that, in a suitable strict domain-theoretic model, any term not denoting $\bot$ is strongly normalisable (Theorem 3). Since the evaluation functional for terms is total, this reduces the strong normalisation problem to the totality problem for constants (Theorem 4), which, in many applications, is easily solvable. Using this method,

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press

we have obtained new strong normalisation results for variants of barrecursion in higher types (Theorem 5) and related systems.

Our work can be extended in a number of ways. For example, one can enforce certain reduction strategies by restricting the application of rewrite rules to certain contexts. These restrictions will then have to be reflected in the model by making the semantic operations strict only at those arguments that correspond to these contexts. In the case of call-by-name reduction, for example, only constructors need to be made strict since function application is strict in its left argument anyway. One may also allow a formal non-strict if-then-else construct making Vogel's trick unnecessary. Furthermore, it is possible to extend our method to stronger type systems such as polymorphic types (Berger 2005). Also, extensions to dependent types should not pose big problems since our rewrite rules are always confluent (Tait's well-known method of proving confluence via the diamond lemma for parallel reduction applies).

The rewrite systems we have investigated in this paper are, on the one hand, very general (there is no syntactic restriction imposed on the right-hand sides of rules) and hence able to capture mathematically very strong recursion schemes. On the other hand, there are restrictions (the left-hand sides of rules must be linear and mutually non-unifiable constructor patterns) that confine them to be used for *defining* functionals, and not for performing *ad hoc* transformations like $x + (y + z) \mapsto (x + y) + z$. It would be interesting to combine our results with other work on the termination of higher type rewrite systems that does allow rules of the latter kind, but are subject to other restrictions such as, for example, Jouannaud and Okada's *general schema* (Barbanera *et al.* 1997; Blanqui *et al.* 1999).

Acknowledgements

I am grateful to the anonymous referees for valuable comments and suggestions concerning the presentation of the results.

References

Abramsky, S. and Jung, A. (1994) Domain Theory. In: Abramsky, S., Gabbay, D. M. and Maibaum, T. S. E. (eds.) *Handbook of Logic in Computer Science* **3**, Clarendon Press 1–168.

McAllester, D., Kucan, J. and Otth, D. F. (1995) A Proof of Strong Normalization for F2, Fω and Beyond. *Information and Computation* **121** 1–14.

Barbanera, F., Fernandez, M. and Geuvers, H. (1997) Modularity of strong normalization in the algebraic-λ-cube. *Journal of Functional Programming* **7** (6) 613–660.

Barendregt, H. (1992) Lambda calculi with types. In: Abramsky, S., Gabbay, D. M. and Maibaum, T. S. E. (eds.) *Handbook of Logic in Computer Science* **2**, Clarendon Press 117–309.

Berardi, S., Bezem, M. and Coquand, T. (1998) On the computational content of the axiom of choice. *Journal of Symbolic Logic* **63** (2) 600–622.

Berger, U. (2004) A computational interpretation of open induction. In: Titsworth, F. (ed.) *Lics '04: Proceedings of the Nineteenth Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society 326–334.

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press

Berger, U. (2005) Strong normalization for applied lambda calculi. *Logical Methods in Computer Science* **1** (2) 1–14.

Berger, U. and Oliva, P. (2005) Modified bar recursion and classical dependent choice. In: Baaz, M., Friedman, S. D. and Kraijcek, J. (eds.) Logic Colloquium '01: Proceedings of the Annual European Summer Meeting of the Association for Symbolic Logic. *Lecture Notes in Logic* **20**, Association of Symbolic Logic 89–107.

Bezem, M. (1985) Strong normalisation of barrecursive terms without using infinite terms. *Archive for Mathematical Logic* **25** 175–181.

Blanqui, F., Jouannaud, J-P. and Okada, M. (1999) The calculus of algebraic constructions. In: Narendran, P. and Rusinowitch, M. (eds.) RTA '99: Proceedings of Tenth International Conference on Rewriting Techniques and Applications. *Springer-Verlag Lecture Notes in Computer Science* **1631** 301–316.

Coquand, T. (1985) *Une théorie des constructions*, Ph.D. thesis, Université Paris VII.

Geuvers, H. and Nederhof, M. J. (1991) A modular proof of strong normalization for the calculus of constructions. *Journal of Functional Programming* **1** (2) 155–189.

Girard, J-Y. (1972) 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: Fenstad, J. E. (ed.) *Proceedings of the Second Scandinavian Logic Symposium*, North–Holland 63–92.

Luckhardt, H. (1973) Extensional Gödel Functional Interpretation – A Consistency Proof of Classical Analysis. *Springer-Verlag Lecture Notes in Mathematics* **306**.

Matthes, R. (2001) Monotone inductive and coinductive constructors of rank 2. In: Fribourg, L. (ed.) Computer Science Logic (Proceedings of the Fifteenth CSL Conference). *Springer-Verlag Lecture Notes in Computer Science* **2142** 600–615.

Plotkin, G. D. (1977) LCF considered as a programming language. *Theoretical Computer Science* **5** 223–255.

Spector, C. (1962) Provably recursive functionals of analysis: A consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In: Dekker, F. D. E. (ed.) *Recursive Function Theory: Proc. Symposia in Pure Mathematics* **5**, American Mathematical Society 1–27.

Stoltenberg-Hansen, V., Griffor, E. and Lindström, I. (1993) *Mathematical theory of domains*, Cambridge University Press.

Tait, W. W. (1971) Normal form theorem for barrecursive functions of finite type. In: Fenstad, J. E. (ed.) *Proceedings of the Second Scandinavian Logic Symposium*, North–Holland 353–367.

Troelstra, A. S. (1973) Metamathematical Investigation of Intuitionistic Arithmetic and Analysis. *Springer-Verlag Lecture Notes in Mathematics* **344**.

van de Pol, J. and Schwichtenberg, H. (1995) Strict functionals for termination proofs. In: Dezani-Ciancaglini, M. and Plotkin, G. (eds.) Typed Lambda Calculi and Applications. *Springer-Verlag Lecture Notes in Computer Science* **902** 350–364.

Vogel, H. (1976) Ein starker Normalisationssatz für die barrekursiven Funktionale. *Archive for Mathematical Logic* **18** 81–84.

https://doi.org/10.1017/S0960129506005457 Published online by Cambridge University Press