STRONG NORMALISATION FOR APPLIED LAMBDA CALCULI

ULRICH BERGER

Department of Computer Science, University of Wales Swansea, Singleton Park, Swansea, SA2
8PP, United Kingdom
e-mail address: u.berger@swan.ac.uk

ABSTRACT. We consider the untyped lambda calculus with constructors and recursively
defined constants. We construct a domain-theoretic model such that any term not denoting
$\bot$ is strongly normalising provided all its 'stratified approximations' are. From this we
derive a general normalisation theorem for applied typed λ-calculi: If all constants have
a total value, then all typeable terms are strongly normalising. We apply this result to
extensions of Gödel's system T and system F extended by various forms of bar recursion
for which strong normalisation was hitherto unknown.

1. INTRODUCTION

Extensions of typed λ-calculi by data types and recursively defined higher-order func-
tions, often called *applied λ-calculi*, play an important role in logic and computer science.
They are used, for example, to represent formal proofs and to give computational in-
terpretations of logical and mathematical theories leading to relative consistency results
and estimates of the strengths of theories in terms of their provably recursive functions
[Göd58, Spe62, Gir71, Tro73, CU93]. They also form the theoretical backbone of functional
and type-theoretic proof/programming languages [ea86, PM93]. The most important and
often also most difficult problem in the study of applied λ-calculi is normalisation, i.e. the
question whether every term can be reduced to a normal form with respect to β-reduction
and the rewrite rules for the extended calculus. The best possible result in this connection is
strong normalisation, i.e. termination of *every* possible reduction sequence. A common pat-
tern for proving strong normalisation for an applied λ-calculus is to take an existing strong
normalisation proof for the 'pure' underlying typed λ-calculus w.r.t. β-reduction only and
adapt it to the applied calculus. A typical example is the strong normalisation proof for
Gödel's system T of primitive recursive functionals in simple types [Göd58] which can be
obtained by adapting Tait's computability method to primitive recursion [Tro73]. Similar
methods were applied to prove strong normalisation for the calculus of constructions ex-
tended by inductive types [Alt93, BJO99]. There are important extensions of system T by
stronger recursion principles, for example Spector's bar recursion [Spe62], which also have

been treated using adaptations of Tait's computability method, however at the price of considerable complications. The difficulty in proving normalisation for bar recursion and similar recursion schemes lies in the fact that these schemes do not use a recursive descent along some kind of wellfounded structural ordering, but rather rely on continuity arguments and the ability to construct, in a suitable model, infinite sequences by nonconstructive choices. Since the computability method amounts to the construction of a syntactic model (built from strongly normalising terms) which does not satisfy these requirements, one needs to enrich the model, either by introducing infinite terms [Tai71, Vog85], or by building the model from sets of terms instead of single terms [Bez85]. These modifications, which work for Spector's bar recursion, seem to fail, however, for other recursion principles which also rely on continuity and choice and which occur in recent work on computational interpretations of classical choice and related principles [BBC98, BO05, Ber04a]. An example is modified bar recursion (see (2.1) in section 2).

In this paper we present a new method for proving strong normalisation of applied lambda calculi which will allow us to deal with modified bar recursion and other forms of recursion. The method roughly proceeds as follows: Let $\mathcal{T}$ be a strongly normalising typed λ-calculus. We assume that $\mathcal{T}$ is given as a type assignment system for untyped λ-terms with constructors, and we require that $\mathcal{T}$ allows for the (nonrecursive) definition of functions by pattern matching on constructors. Let $\mathcal{R}$ be a higher-order rewrite system using pattern matching and (possibly) recursion. Then, to prove strong normalisation for $\mathcal{T} + \mathcal{R}$ we

(a) interpret the untyped terms in a strict domain model where the constants are interpreted according to $\mathcal{R}$,
(b) show that any term not denoting $\bot$ is strongly normalising,
(c) prove that all constants are total w.r.t. the notion of totality given by the typing discipline of $\mathcal{T}$

While, as we will see, (a) and (b) are always possible, (c) will depend on the given rewrite system $\mathcal{R}$. Now, by (c) (and the presumed soundness of the typing discipline w.r.t. the model) all typeable terms are total and hence $\neq \bot$. With (b) it follows that all typeable terms are strongly normalising.

The advantages of this method lie in its generality and its manifold modularity aspects: First, strong normalisation of the underlying typed λ-calculus can be proven separately. Second, steps (a) and (b) above are independent of the typing discipline and can be carried out for any rewrite system $\mathcal{R}$. The constructions and proofs involved in (a) and (b) are elementary (formalisable in primitive recursive arithmetic). Third, the logical and mathematical strengths of the typing discipline and the rewrite rules only enter into step (c). The proof of totality of the constants can usually be carried out using the intuitive argument why the given rewrite rules are 'semantically sound'. Fourth, the combination of different rewrite systems for which strong normalisation can be shown using our method preserves strong normalisation. This holds because our method only uses totality of the constants of each rewrite system separately.

Our method appears to be similar to Plotkin's adequacy proof for PCF [Plo77], and, in fact, is inspired by [Plo77]. The differences are that Plotkin intertwines the computability method for the simply typed λ-calculus with a semantic approximation argument whereas we keep these arguments separate. Also, Plotkin deals with full recursion without pattern matching and shows weak normalisation for closed terms of ground-type only whereas our

recursions are of a restricted form, but we show strong normalisation for terms of arbitrary types.

As mentioned above our method only applies to rewrite systems based on a restricted form of pattern matching. These restrictions enforce that the rewrite rules define the operational and denotational semantics of the constants in a canonical way. Hence, rules like distributivity or permutative conversions, which express optimisations rather than definitions, are not covered by our method. Strong normalisation results for higher-order rewrite rules of the latter kind are proven in [vdPS95] with a different semantic method based on another notion of strict functionals.

Our paper is organised as follows. Section 2 introduces untyped λ-calculi with constructors and constants together with higher-order rewrite rules. As our running example we discuss a few primitive recursive constants and rewrite rules for modified bar recursion. We give an informal argument why, under the assumption that higher type functions are continuous, bar recursion is sound. In section 3 we define a strict domain-theoretic model for the calculi introduced previously. We prove that any term not denoting $\bot$ is strongly normalisable provided all its 'stratified approximations' are, where, roughly speaking, the n-th stratified approximation of a term is obtained by replacing each constant by a variant for which maximally n unfoldings of the recursion equations are allowed. The untyped result is used in section 4 to prove a strong normalisation theorem for applied λ-calculi based on an abstract notion of a strongly normalising and total type system. We have kept the notion of type system as general as possible in order to prepare the ground for future applications of our method to a variety of type systems. In section 5 we consider as an example Girard/Reynold's system F of second-order polymorphic λ-calculus. We show that extending this system by higher type primitive recursion and modified bar recursion does not destroy strong normalisation. We also discuss some other higher-order rewrite systems our method applies to.

In [Ber04b] we have worked out a special case of our method, tailored for the simply typed λ-calculus. By giving up generality the definition of the model and the totality proofs then are slightly simpler. However, we feel that the greater flexibility gained by the type free approach of this paper is worth paying the price of technically slightly more involved constructions.

Acknowledgements. The comments and the constructive criticism by two anonymous referees contributed significantly to an improved presentation of this work.

2. The type free λ-calculus with constructors and recursion

We fix a set Var of variables $x, y \ldots$. Given a set $\mathcal{CO}$ of *constructors* co and a set $\mathcal{C}$ of constants c, the set of terms $\Lambda = \Lambda(\mathcal{CO}, \mathcal{C})$ is defined by

$$\Lambda \ni M, N := x \mid \mathsf{co}(M_1, \ldots, M_k) \mid c \mid \lambda x M \mid MN$$

where k is the *arity* of co, which is fixed for each constructor. If $\mathcal{CO}$ is fixed, but $\mathcal{C}$ may vary, then we write $\Lambda(\mathcal{C})$ instead of $\Lambda(\mathcal{CO}, \mathcal{C})$. We let $\mathsf{FV}(M)$ denote the set of free variables of a term M.

The operational meaning of the constants $c \in \mathcal{C}$ is given by a a set $\mathcal{R}$ of *rewrite rules* of the form

$$cP_1 \ldots P_n \ \mapsto \ R$$

where $\mathsf{FV}(cP_1 \ldots P_n) \supseteq \mathsf{FV}(R)$ and the P_i are *constructor patterns*, i.e. terms built from variables and the constructors such that no variable occurs twice in the term $c\vec{P}$. The number n of arguments P_i is fixed for each constant c and is called the *arity* of c. We furthermore require the P_i to be mutually variable disjoint and the left hand sides of different rules to be non-unifiable. A rule of the form $c\vec{P} \mapsto R$ is called a *rule for c*.

A *conversion*, $M \mapsto N$, is either a β-conversion, $(\lambda x M)N \mapsto_\beta M[N/x]$, or an instance of a rewrite rule, i.e. $L\sigma \mapsto_{\mathcal{R}} R\sigma$ for some rewrite rule $L \mapsto R \in \mathcal{R}$ and substitution σ. We write $M \to_{\mathcal{R}} N$ 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 normalising* with respect to $\mathcal{R}$, $\mathsf{SN}_{\mathcal{R}}(M)$, if there is no infinite reduction sequence $M \to_{\mathcal{R}} M_1 \to_{\mathcal{R}} \ldots$. Equivalently, the predicate $\mathsf{SN}_{\mathcal{R}}$ is inductively generated by the single rule:

If $\mathsf{SN}_{\mathcal{R}}(N)$ for all N such that $M \to_{\mathcal{R}} N$, then $\mathsf{SN}_{\mathcal{R}}(M)$.

Our goal is to prove strong normalisation for various classes of terms (which are usually given by typing disciplines) with respect to various rewrite systems $\mathcal{R}$.

As an example consider the constructors T, F, 0, $[]$, S, cons and the constants if, $<$, lh, get and $+\!\!+$ with the rewrite rules

$$
\begin{aligned}
\mathsf{if}\,\mathsf{T}\,x\,y &\mapsto x \\
\mathsf{if}\,\mathsf{F}\,x\,y &\mapsto y \\[4pt]
n < 0 &\mapsto \mathsf{F} \\
0 < \mathsf{S}(m) &\mapsto \mathsf{T} \\
\mathsf{S}(n) < \mathsf{S}(m) &\mapsto n < m \\[4pt]
\mathsf{lh}\,[] &\mapsto 0 \\
\mathsf{lh}\,\mathsf{cons}(x, s) &\mapsto \mathsf{S}(\mathsf{lh}s) \\[4pt]
\mathsf{get}\,\mathsf{cons}(x, s)\,0 &\mapsto x \\
\mathsf{get}\,\mathsf{cons}(x, s)\,\mathsf{S}(n) &\mapsto \mathsf{get}\,s\,n \\[4pt]
[] +\!\!+ t &\mapsto t \\
\mathsf{cons}(x, s) +\!\!+ t &\mapsto \mathsf{cons}(x, s +\!\!+ t)
\end{aligned}
$$

When assigning suitable types to these constructors and constants one obtains a subsystem of Gödel's system T of primitive recursive functionals in finite types which is well-known to be strongly normalising. We are interested in stronger forms of recursion, for example bar recursion, which we discuss now. To improve readability will write M_k for $\mathsf{get}\,M\,k$, $M*N$ for $M +\!\!+ \mathsf{cons}(N, [])$ and if M then N else K for $\mathsf{if}\,M\,N\,K$. The following form of *modified bar recursion* was studied in [BBC98] and [BO05]:

$$\Phi y g s \;=\; y(\lambda k.\mathsf{if}\;k < |s|\;\mathsf{then}\;s_k\;\mathsf{else}\;gk(\lambda x.\Phi y g(s*x)))$$

In order to make sense of this equation one should think of the variables being typed as follows: $y\colon (\mathsf{nat} \to \rho) \to \mathsf{nat}$, $g\colon \mathsf{nat} \to (\rho \to \mathsf{nat}) \to \rho$, $s\colon \mathsf{list}(\rho)$, where ρ is an arbitrary type. A functional Φ satisfying the recursion equation above was used in [BBC98] and [BO05] to give a realisability interpretation of the classical (i.e. negative translated) axiom of countable dependent choice. Below we give an intuitive argument why, under the

assumption that higher type functions are continuous, the equation above is sound in the sense that it defines a total functional (i.e. maps total arguments to total values):

Let g, y, s be total arguments where $s = [x_0, \dots, x_{n-1}]$ with total x_i, and assume, for contradiction, that Φygs is undefined. Then the infinite sequence $\lambda k.\text{if } k < |s| \text{ then } x_k \text{ else } gk(\lambda x.\Phi yg(s*x))$ cannot be total, so there must be some k such that $gk(\lambda x.\Phi yg(s*x))$ is not total. Since g is total this implies that $\lambda x.\Phi yg(s*x)$ is not total, i.e. $\Phi yg(s*x_n)$ is not total for some total x_n. Repeating this argument one arrives at an infinite sequence of total elements x_k such that $\Phi yg[x_0, \dots, x_{m-1}]$ is undefined for all $m \geq n$. Since all x_k are total, $y(\lambda k.x_k)$ must be defined. Furthermore, since y is assumed to be continuous it will query its argument at numbers k smaller than some fixed number m only. But then $\Phi yg[x_0, \dots, x_{m-1}]$ must be defined as well, which is a contradiction.

In the proof of theorem 5.4 we will repeat a slight variation of this argument in a strict domain-theoretic model in order to obtain strong normalisation of system F extended by modified bar recursion. More precisely, since turning the equation above into a rewrite rule would clearly not be strongly normalising, we will have to work with the following minor variation of modified bar recursion. We replace the conditional expression on 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 $k < |s|$ before the subterm $\Phi ygh(s*x)$ may be further reduced (Vogel's trick [Vog85]).

$$
\begin{array}{rcl}
\Phi ygs & \mapsto & y(\lambda k.\Psi ygsk(k < |s|)) \\
\Psi ygsk\mathsf{T} & \mapsto & s_k \\
\Psi ygsk\mathsf{F} & \mapsto & gk(\lambda x.\Phi yg(s*x))
\end{array}
\tag{2.1}
$$

Our results will enable us to easily show that all terms that are typeable in system F (under a suitable typing of the constructors and constants) are strongly normalising with respect to the rewrite rules above.

3. A domain-theoretic model for strong normalisation

By a *domain* we mean a Scott domain, i.e. a consistently complete algebraic domain with an effective base [GLSH93, AJ94]. The least element of a domain D is denoted $\bot_D$ (or $\bot$, if no confusion is possible) and $\sqsubseteq_D$ (or $\sqsubseteq$) denotes the domain ordering. $D \to E$ denotes the domain of continuous functions from D to E. Note that $\bot_{D\to E} = \lambda a \in D.\bot_E$. If X is an effectively given countable set, then $X_\bot$ denotes the flat domain $X \cup \{\bot\}$ where all elements of X are maximal and $\bot(\notin X)$ is the least element. For any domain D we let D^X be the domain of all functions from X to D, ordered pointwise. For a continuous function $f\colon D^k \to E$ we define the *strict version*, $\mathtt{strict}(f)\colon D^k \to E$, by $\mathtt{strict}(f)(\vec{a}) := f(\vec{a})$ if $\bot \notin \vec{a}$, $\mathtt{strict}(f)(\vec{a}) := \bot$ otherwise. Clearly $\mathtt{strict}(f)$ is again continuous and $\mathtt{strict}(f) \sqsubseteq f$. Moreover, $\mathtt{strict}(.)$ is itself a continuous function on the domain $D^k \to E$. We will also use $\mathsf{Maybe}(D) := D + 1 = \{\mathsf{Just}(d) \mid d \in D\} \cup \{\mathsf{Nothing}\} \cup \{\bot\}$ where 1 denotes some 1-element domain and, in general, $D_1 + \dots + D_k$ denotes the usual separated sum of domains which has as carrier the disjoint union of the D_i plus a new bottom element and is ordered in the expected way.

Given a system $\mathcal{CO}$ of constructors we define the domain D by the recursive domain equation

$$
D = \Sigma_{\mathsf{co}\in\mathcal{CO}}D^{\mathsf{arity}(\mathsf{co})} + (D \to D)
$$

The existence of a solution to such an equation is guaranteed by the fact that in the category of domains and embedding/projection pairs the separated sum and the continuous function space construction are continuous functors (co-variant in all arguments) and all such functors have initial fixed points (up to isomorphism). By the definition of D every element of $D \setminus \{\bot\}$ is either of the form $\mathtt{co}(\vec{a})$ with $\vec{a} \in D^{\mathsf{arity(co)}}$, or of the form $\mathtt{abst}(f)$ with $f \in D \to D$, and there are continuous functions $\mathtt{abst} \colon (D \to D) \to D$, $\mathtt{app} \colon D^2 \to D$ and $\mathtt{case} \colon D \to (\mathcal{CO} \cup \{\mathtt{abst}\})_{\bot}$ as well as for each constructor $\mathtt{co}$ of arity k continuous functions $\mathtt{co} \colon D^k \to D$ and $\mathtt{co}_i^{-1} \colon D \to D$ $(i = 1, \dots, k)$ such that

(i) $\mathtt{case}(\mathtt{co}(\vec{a})) = \mathtt{co}$, $\mathtt{case}(\mathtt{abst}(f)) = \mathtt{abst}$,

(ii) $\mathtt{co}_i^{-1}(\mathtt{co}(\vec{a})) = a_i$,

(iii) $\mathtt{app}(\mathtt{abst}(f), b) = f(b)$.

If $f \in D^k \to D$, then $\mathtt{abst}(f)$ stands for $\mathtt{abst}(\lambda a_1 \in D. \dots \mathtt{abst}(\lambda a_k \in D.f(a_1, \dots, a_k)))$. Similarly, $\mathtt{app}(a, b_1, \dots, b_k)$ abbreviates $\mathtt{app}(\dots \mathtt{app}(a, b_1) \dots, b_k)$. We define for each term $M \in \Lambda(\mathcal{C})$ the *strict denotational semantics* $[M] \colon D^{\mathcal{C}} \to D^{\mathsf{Var}} \to D$ by

$$
\begin{aligned}
[x]^\alpha \eta &= \eta(x) \\
[c]^\alpha \eta &= \alpha(c) \\
[\lambda x\, M]^\alpha \eta &= \mathtt{strict}(\mathtt{abst})(\lambda a \in D.[M]^\alpha \eta_x^a) \\
[MN]^\alpha \eta &= \mathtt{strict}(\mathtt{app})([M]^\alpha \eta [N]^\alpha \eta) \\
[\mathtt{co}(M_1, \dots, M_k)]^\alpha \eta &= \mathtt{strict}(\mathtt{co})([M_1]^\alpha \eta, \dots, [M_k]^\alpha \eta)
\end{aligned}
$$

The soundness of this definition rests on the fact that domains and continuous functions form a cartesian closed category.

Lemma 3.1. *Let $M, N \in \Lambda(\mathcal{C})$, $\alpha \in D^{\mathcal{C}}$, $\eta \in D^{\mathsf{Var}}$, $\theta \colon \mathcal{C} \to \mathcal{C}'$, $\alpha' \in D^{\mathcal{C}'}$.*

(a) *If $\alpha(c) = \bot_D$ for some constant c in M, then $[M]^\alpha = \bot_{D^{\mathsf{Var}}}$.*

(b) *$[M[N/x]]^\alpha \eta = [M]^\alpha \eta_x^a$ where $a := [N]^\alpha \eta$.*

(c) *$[M\theta]^{\alpha'} = [M]^{\alpha' \circ \theta}$ $(M\theta := M[\theta(c)/c \mid c \in \mathcal{C}])$.*

(d) *$[(\lambda x M)N] \sqsubseteq [M[N/x]]$.*

Proof. (a-c) are proved by easy inductions on M. (d) follows from (b): Let $a := [N]^\alpha \eta$. Then $[(\lambda x M)N]^\alpha \eta \sqsubseteq \mathtt{app}(\mathtt{abst}(\lambda b \in D.[M]^\alpha \eta_x^b), a) = [M]^\alpha \eta_x^a = [M[N/x]]^\alpha \eta$. $\square$

Next we define the constant assignment $\alpha_{\mathcal{R}} \in D^{\mathcal{C}}$ naturally associated with a rewrite system $\mathcal{R}$. The values $\alpha_{\mathcal{R}}(c) \in D$ are defined by a simultaneous recursion, i.e. $\alpha_{\mathcal{R}}$ is the least fixed point of a certain continuous operator on the domain $D^{\mathcal{C}}$. For a constant c without any rule in $\mathcal{R}$ we set $\alpha_{\mathcal{R}}(c) := \bot$. The definition of $\alpha_{\mathcal{R}}(c)$ for constants with at least one rule requires some preparation. For every vector $\vec{P} = P_1, \dots, P_k$ of variable disjoint constructor patterns we define a continuous 'inverse' $\vec{P}^{-1} \colon D^k \to \mathsf{Maybe}(D^{\mathsf{Var}})$. The definition is by recursion on the number of constructors in $\vec{P}$. $\vec{x}^{-1}(\vec{a}) := \mathsf{Just}(\bot_{\vec{x}}^{\vec{a}})$, where $\bot_{\vec{x}}^{\vec{a}}(x_i) = a_i$, and

$$(\vec{x}, \mathtt{co}(Q_1, \dots, Q_n), \vec{P})^{-1}(\vec{a}, b, \vec{c}) :=$$

$$
\begin{cases}
(\vec{x}, Q_1, \dots, Q_n, \vec{P})^{-1}(\vec{a}, \mathtt{co}_1^{-1}(b), \dots, \mathtt{co}_n^{-1}(b), \vec{c}) & \text{if } \mathtt{case}(b) = \mathtt{co} \\
\mathsf{Nothing} & \text{if } \mathtt{case}(b) \in (\mathcal{CO} \setminus \{\mathtt{co}\}) \cup \{\mathtt{abst}\} \\
\bot & \text{if } \mathtt{case}(b) = \bot
\end{cases}
$$

Lemma 3.2.

(a) *If $\vec{P}^{-1}(\vec{a}) = \mathsf{Just}(\eta)$ and $\vec{P}$ and $\vec{Q}$ are non-unifiable, then $\vec{Q}^{-1}(\vec{a}) \in \{\bot, \mathsf{Nothing}\}$.*

(b) $\vec{P}^{-1}([\vec{P}\sigma]^{\alpha}\eta) = [\sigma]^{\alpha}\eta$ *where* $[\sigma]^{\alpha}\eta(x) := [\sigma(x)]^{\alpha}\eta$.

Proof. Easy inductions on the number of constructors in $\vec{P}$. $\square$

By lemma 3.2 (a), the condition that the left hand sides of different rules for the same constant are non-unifiable implies that for every constant c of $\mathcal{R}$-arity k and every $\vec{a} \in D^k$ there is at most one rule $c\vec{P} \mapsto R \in \mathcal{R}$ such that $\vec{P}^{-1}(\vec{a}) = \mathsf{Just}(\eta)$ for some $\eta \in D^{\mathsf{Var}}$. This guarantees the soundness of the following definition of the values of a constant c with at least one rule in $\mathcal{R}$: $\alpha_{\mathcal{R}}(c) := \mathsf{abst}(f)$ where $f \colon D^k \to D$ is defined (recursively) by

$$f(\vec{a}) = \begin{cases} [R]^{\alpha_{\mathcal{R}}}\eta & \text{if } c\vec{P} \mapsto R \in \mathcal{R} \text{ and } \vec{P}^{-1}(\vec{a}) = \mathsf{Just}(\eta) \\ \mathtt{dummy} & \text{if } \vec{P}^{-1}(\vec{a}) = \mathsf{Nothing} \text{ for all } c\vec{P} \mapsto R \in \mathcal{R} \\ \bot & \text{otherwise} \end{cases}$$

Here $\mathtt{dummy}$ is some fixed element of D whose value will be irrelevant in this section. However, when applying our construction to a particular type system (Section 5), we will have to choose $\mathtt{dummy}$ in such a way that it lies in the intersection of all denotations of types (note that $\mathtt{dummy}$ is independent of the type that might be associated with the constant c).

We set

$$[M]^{\mathcal{R}} := [M]^{\alpha_{\mathcal{R}}}$$

Lemma 3.3. *If* $M \to N$, *then* $[M]^{\mathcal{R}}\eta \sqsubseteq [N]^{\mathcal{R}}\eta$.

Proof. Induction on M.

If $M \mapsto_{\beta} N$, then we use lemma 3.1 (d).

Consider the case of a constant conversion, i.e. $c\vec{P}\sigma \to R\sigma$. Set $\vec{a} := [\vec{P}\sigma]^{\mathcal{R}}\eta = [\vec{P}]^{\mathcal{R}}\eta'$ where $\eta'(x) := [\sigma(x)]^{\mathcal{R}}\eta$, by lemma 3.1 (b). By lemma 3.2 (b), $\vec{P}^{-1}(\vec{a}) = \mathsf{Just}(\eta')$. Therefore, $[c\vec{P}\sigma]^{\mathcal{R}}\eta \sqsubseteq \mathsf{app}(\alpha_{\mathcal{R}}(c), \vec{a}) \sqsubseteq [R]^{\mathcal{R}}\eta' = [R\sigma]^{\mathcal{R}}\eta$, again by lemma 3.1 (b).

All other cases (conversion of a proper subterm) follow immediately from the induction hypothesis and the fact that the functions $\mathtt{strict(abst)}$, $\mathtt{strict(app)}$ and $\mathtt{strict(constr)}$ are monotone. $\square$

The key to our first normalisation result is the approximation of a given rewrite system by a 'stratified' rewrite system, that is a rewrite system where no recursion occurs. More precisely, let $\mathcal{R}$ be a rewrite system for a given term system $\Lambda(\mathcal{C})$ and define inductively a constant $c \in \mathcal{C}$ to be *stratified* (w.r.t. $\mathcal{R}$) if for every rule $cP_1 \ldots P_n \mapsto R \in \mathcal{R}$ the term R is stratified, i.e. contains stratified constants only. Roughly speaking, stratified rewrite systems allow nothing more than to define functions by pattern matching and case analysis on constructors. $\mathcal{R}$ is called stratified if all constants are stratified w.r.t. $\mathcal{R}$.

Let $\mathcal{R}$ be an arbitrary rewrite system for a system of constants $\mathcal{C}$. For every constant $c \in \mathcal{C}$ and each $n \in \mathbb{N}$ let c_n be a new constant and set $\mathcal{C}_{\omega} := \{c_n \mid c \in \mathcal{C}, n \in \mathbb{N}\}$. For every term $M \in \Lambda(\mathcal{C})$ and $n \in \mathbb{N}$ let $M_{[n]} \in \Lambda(\mathcal{C}_{\omega})$ be the term obtained from M by replacing every constant c by c_n. We define a stratified rewrite system for $\mathcal{C}_{\omega}$ by

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

In the following we let $M, N, \ldots$ range over $\Lambda(\mathcal{C})$ while $A, B, \ldots$ range over $\Lambda(\mathcal{C}_{\omega})$. We write $A \preceq M$ if replacing in A each constant c_n by c yields M. In particular $M_{[n]} \preceq M$.

Lemma 3.4. *If* $A \preceq M$ *and* A *contains no constant of the form* c_0, *then to every* $\mathcal{C}$-*term* N *such that* $M \to_{\mathcal{R}} N$ *there is a* $\mathcal{C}_{\omega}$-*term* A *such that* $A \to_{\mathcal{R}_{\omega}} B$ *and* $B \preceq N$.

Proof. Easy induction on M. $\square$

Lemma 3.5. $[M]^{\mathcal{R}} = \bigsqcup_n [M_{[n]}]^{\mathcal{R}_\omega}$.

Proof. By definition, $\alpha_{\mathcal{R}}$ is the least fixed point of the continuous functional $\Gamma_{\mathcal{R}} : D^{\mathcal{C}} \to D^{\mathcal{C}}$ defined by $\Gamma_{\mathcal{R}}(\alpha)(c) := \bot$ if there is no rule for c in $\mathcal{R}$, otherwise $\Gamma_{\mathcal{R}}(\alpha)(c) := \mathtt{abst}(f)$ where $f : D^k \to D$ is defined by

$$f(\vec{a}) := \begin{cases} [R]^{\alpha}\eta & \text{if } c\vec{P} \mapsto R \in \mathcal{R} \text{ and } \vec{P}^{-1}(\vec{a}) = \mathsf{Just}(\eta) \\ \mathtt{dummy} & \text{if } \vec{P}^{-1}(\vec{a}) = \mathsf{Nothing} \text{ for all } c\vec{P} \mapsto R \in \mathcal{R} \\ \bot & \text{otherwise} \end{cases}$$

Set $\alpha_n(c) := \alpha_{\mathcal{R}_\omega}(c_n)$. We show

$$\alpha_n = \Gamma_{\mathcal{R}}^n(\bot) \tag{3.1}$$

by induction on n. For $n = 0$ both sides are $\bot$ (the left hand side $= \bot$ because there are no rules for constants of the form c_0). If there is no rule for c in $\mathcal{R}$, then both sides of (3.1) are again $\bot$. Let now c be a constant with at least one rule in $\mathcal{R}$. By induction hypothesis we have $\Gamma_{\mathcal{R}}^{n+1}(\bot)(c) = \mathtt{abst}(f_n)$ where

$$f_n(\vec{a}) := \begin{cases} [R]^{\alpha_n}\eta & \text{if } c\vec{P} \mapsto R \in \mathcal{R} \text{ and } \vec{P}^{-1}(\vec{a}) = \mathsf{Just}(\eta) \\ \mathtt{dummy} & \text{if } \vec{P}^{-1}(\vec{a}) = \mathsf{Nothing} \text{ for all } c\vec{P} \mapsto R \in \mathcal{R} \\ \bot & \text{otherwise} \end{cases}$$

(note that the definitions of the functions f and f_n above and the definition of f after lemma 3.2 differ in the constant environment under which the term R is evaluated). Since by lemma 3.1 (c), $[R]^{\alpha_n}\eta = [R_{[n]}]^{\mathcal{R}_\omega}\eta$ ($\alpha_n = \alpha_{\mathcal{R}} \circ \theta_n$ where $\theta_n(c) := c_n$) it follows $\Gamma_{\mathcal{R}}^{n+1}(\bot)(c) = \alpha_{\mathcal{R}_\omega}(c_n) = \alpha_n(c)$. Now, since $\alpha_{\mathcal{R}}$ is the directed supremum of the $\Gamma_{\mathcal{R}}^n(\bot)$ it follows, by continuity of the evaluation function $[M]$, equation (3.1) and lemma 3.1 (c),

$$[M]^{\mathcal{R}} = \bigsqcup_n [M]^{\Gamma_{\mathcal{R}}^n(\bot)} = \bigsqcup_n [M]^{\alpha_n} = \bigsqcup_n [M_{[n]}]^{\alpha_\omega}$$

$\square$

Theorem 3.6. *If $[M]^{\mathcal{R}} \neq \bot$ and all $M_{[n]}$ are strongly normalising w.r.t. $\mathcal{R}_\omega$, then M is strongly normalising w.r.t. $\mathcal{R}$.*

Proof. Assume $[M]^{\mathcal{R}} \neq \bot$. By continuity we have $[M]^{\Gamma_{\mathcal{R}}^n(\bot)} \neq \bot$ for some n. By lemma 3.5 it follows $[M_{[n]}]^{\alpha_\omega}\eta \neq \bot$ for some n. Since, by assumption, $M_{[n]}$ is strongly normalising w.r.t. $\mathcal{R}_\omega$ it suffices to show:

$$\text{If } \mathsf{SN}_{\mathcal{R}_\omega}(A), [A]^{\alpha} \neq \bot \text{ and } A \preceq M, \text{ then } \mathsf{SN}_{\mathcal{R}}(M). \tag{3.2}$$

We show this by induction on $\mathsf{SN}_{\mathcal{R}_\omega}(A)$. Assume the hypotheses of (3.2). We need to show that all one step reducts of M are strongly normalising. So, assume $M \to_{\mathcal{R}} N$. Since $[A]^{\alpha} \neq \bot$ we know, by lemma 3.1 (a), that A contains no constant of the form c_0. By lemma 3.4 it follows that $A \to_{\mathcal{R}_\omega} B$ with $B \preceq N$ for some B. By lemma 3.3 (applied to $\mathcal{R}_\omega$), $[B]^{\alpha} \neq \bot$, hence we can apply the induction hypothesis to B and N. $\square$

4. STRONG NORMALISATION VIA TYPING

In many cases the premises of theorem 3.6 can be proven for terms that are typeable in a certain type system. Our main example will be second-order polymorphism (system F), but any type system that meets a few natural conditions would do as well. These conditions are isolated in the following notions of an (abstract) *strongly normalising type system* and the notion of *totality* of a type system with respect to a *type interpretation*. The former notion is the requirement that all typeable terms are strongly normalising for β-conversion plus any stratified type-sound rewrite system. For a given type system this slightly stronger notion of strong normalisation can usually be obtained by a simple modification of the known proof of strong normalisation for β-conversion (for example Girard's candidate method).

Concerning strong normalisation we clearly may restrict our attention to the set $\Lambda_0(\mathcal{C})$ of closed terms in $\Lambda(\mathcal{C})$.

A *type system* consists of a set $\mathcal{T}$ of *types* and a family of ternary relations $\vdash_{\mathcal{C}} \subseteq \mathcal{T}^{\mathcal{C}} \times \Lambda_0(\mathcal{C}) \times \mathcal{T}$ (indexed by constant systems $\mathcal{C}$) which is stable under type respecting constant substitutions, that is,

$$\text{if } \Delta \vdash_{\mathcal{C}} M : \rho \text{ and } \Delta = \Delta' \circ \theta, \text{ then } \Delta' \vdash_{\mathcal{C}'} M\theta : \rho,$$

for any $\Delta \in \mathcal{T}^{\mathcal{C}}$, $M \in \Lambda_0(\mathcal{T})$, $\rho \in \mathcal{T}$, $\theta \colon \mathcal{C} \to \mathcal{C}'$ and $\Delta' \in \mathcal{T}^{\mathcal{C}'}$.

A term $M \in \Lambda_0(\mathcal{C})$ is called *typeable* w.r.t. $\Delta \in \mathcal{T}^{\mathcal{C}}$ if $\Delta \vdash_{\mathcal{C}} M : \rho$ for some $\rho \in \mathcal{T}$. A rewrite system $\mathcal{R}$ for $\Lambda(\mathcal{C})$ is *type-sound* w.r.t. $\Delta \in \mathcal{T}^{\mathcal{C}}$ if $\Delta \vdash_{\mathcal{C}} \lambda\vec{x}L : \rho$ implies $\Delta \vdash_{\mathcal{C}} \lambda\vec{x}R : \rho$ for every rule $L \mapsto R \in \mathcal{R}$ with $\mathsf{FV}(L) = \vec{x}$ and $\rho \in \mathcal{T}$. The type system $\mathcal{T}, \vdash$ is called *strongly normalising* if for any set of constants $\mathcal{C}$, any type assignment $\Delta \in \mathcal{T}^{\mathcal{C}}$ and any stratified type-sound rewrite system $\mathcal{R}$ for $\Lambda(\mathcal{C})$ all typeable closed terms are strongly normalising w.r.t. $\mathcal{R}$.

Next, we consider possible semantics of types in the model D introduced in the previous section. Since the value of a closed term does not depend on a variable assignment, we may, for closed terms M, set $[M]^{\alpha} := [M]^{\alpha}\eta$ where $\eta \in D^{\mathsf{Var}}$ is arbitrary, for example $\eta := \bot$. A *type interpretation* for $\mathcal{T}$ is a mapping that assigns to every type $\rho \in \mathcal{T}$ a subset $[\![\rho]\!]$ of $D \setminus \{\bot\}$ such that whenever $\Delta \vdash_{\mathcal{C}} M : \rho$ and $\alpha(c) \in [\![\Delta(c)]\!]$ for every $c \in \mathcal{C}$, then $[M]^{\alpha} \in [\![\rho]\!]$.

In the following we will call $\alpha \in D^{\mathcal{C}}$ *total* if $\alpha(c) \in [\![\Delta(c)]\!]$ for all $c \in \mathcal{C}$. Similarly we will call $a \in D$ *total* if $a \in [\![\rho]\!]$ provided ρ is clear from the context.

Theorem 4.1. *Let $\mathcal{T}, \vdash$ be a strongly normalising type system and $[\![\cdot]\!] \colon \mathcal{T} \to \mathcal{P}(D \setminus \{\bot\})$ a type interpretation. Let $\Delta \in \mathcal{T}^{\mathcal{C}}$ be a type assignment and $\mathcal{R}$ a type-sound rewrite system such that $\alpha_{\mathcal{R}}$ is total. Then all typeable closed terms are strongly normalising w.r.t. $\mathcal{R}$.*

Proof. Assume $\Delta \vdash_{\mathcal{C}} M : \rho$. Since $\alpha_{\mathcal{R}}$ is total it follows that $[M]^{\alpha_{\mathcal{R}}}$ is total, i.e. a member of $[\![\rho]\!]$, and hence different from $\bot$. Define the constant substitutions $\theta_n \colon \mathcal{C} \to \mathcal{C}_\omega$ ($n \in \mathbb{N}$) and $\theta' \colon \mathcal{C}_\omega \to \mathcal{C}$ by $\theta_n(c) := c_n$ and $\theta'(c_n) := c$. Note that $M\theta_n = M_{[n]}$. Set $\Delta' := \Delta \circ \theta'$. Since $\Delta = \Delta' \circ \theta_n$ it follows $\Delta' \vdash_{\mathcal{C}_\omega} M\theta_n : \rho$ for every $n \in \mathbb{N}$. Furthermore, the stratified rewrite system $\mathcal{R}_\omega$ is type-sound w.r.t. $\Delta' \in \mathcal{T}^{\mathcal{C}_\omega}$. This can be seen as follows: Assume $\Delta' \vdash \lambda\vec{x}L : \rho$ for some rule $L \mapsto R \in \mathcal{R}_\omega$ with $\mathsf{FV}(L) = \vec{x}$. Since $\Delta' = \Delta \circ \theta'$ it follows $\Delta \vdash \lambda\vec{x}L\theta' : \rho$. But $L\theta' \mapsto R\theta' \in \mathcal{R}$. Hence $\Delta \vdash \lambda\vec{x}R\theta' : \rho$, since $\mathcal{R}$ is type-sound. Since $L \mapsto R \in \mathcal{R}_\omega$ we have $R = R\theta'\theta_n$ for some n. But this implies $\Delta' \vdash \lambda\vec{x}R : \rho$. Since we have shown that $\mathcal{R}_\omega$ is type-sound, and since $M\theta_n$ is typeable and $\mathcal{T}, \vdash$ is assumed to be strongly normalising, it follows that $M\theta_n$ is strongly normalising w.r.t. $\mathcal{R}_\omega$ for every $n \in \mathbb{N}$. Hence both premises of theorem 3.6 are satisfied and we may conclude that M is strongly normalising w.r.t. $\mathcal{R}$. $\square$

5. Applications

As an example of an applied λ-calculus we consider system F extended by lists and constants with rewrite rules based on pattern matching. We consider the same set of constructors $\mathcal{CO} = \{\mathsf{T}, \mathsf{F}, 0, [], \mathsf{S}, \mathsf{cons}\}$ as in section 2, but leave the set of constants $\mathcal{C}$ unspecified for the moment. Given a set TV of *type variables* $p, p_1, \ldots$, the set of (open) *types* is defined by the grammar

$$\rho, \sigma := p \mid \mathsf{boole} \mid \mathsf{nat} \mid \mathsf{list}(\rho) \mid \rho \to \sigma \mid \forall p\, \rho$$

A *context* is a mapping Γ from a finite set $\mathsf{dom}(\Gamma)$ of object variables to the set of types, written as $x_1 \colon \rho_1, \ldots, x_n \colon \rho_n$. The rules for the inductive definition of the typing judgments $\Delta, \Gamma \vdash_{\mathcal{C}} M \colon \rho$, where $\Delta \in \mathcal{T}^{\mathcal{C}}$, Γ is a context, $M \in \Lambda(\mathcal{C})$ and $\rho \in \mathcal{T}$, are displayed in Figure 1.

$$\frac{}{\Delta, \Gamma, x \colon \rho \vdash_{\mathcal{C}} x \colon \rho} \qquad \frac{c \in \mathcal{C}}{\Delta, \Gamma \vdash_{\mathcal{C}} c \colon \Delta(c)}$$

$$\frac{\Delta, \Gamma, x \colon \rho \vdash_{\mathcal{C}} M \colon \sigma}{\Delta, \Gamma \vdash_{\mathcal{C}} \lambda x.M \colon \rho \to \sigma} \qquad \frac{\Delta, \Gamma \vdash_{\mathcal{C}} M \colon \rho \to \sigma \qquad \Delta, \Gamma \vdash_{\mathcal{C}} N \colon \rho}{\Delta, \Gamma \vdash_{\mathcal{C}} MN \colon \sigma}$$

$$\frac{\Delta, \Gamma \vdash_{\mathcal{C}} M \colon \rho}{\Delta, \Gamma \vdash_{\mathcal{C}} M \colon \forall p\, \rho} \; (p \text{ not free in } \Gamma) \qquad \frac{\Delta, \Gamma \vdash_{\mathcal{C}} M \colon \forall p\, \rho}{\Delta, \Gamma \vdash_{\mathcal{C}} M \colon \rho[\sigma/p]}$$

$$\frac{}{\Delta, \Gamma \vdash_{\mathcal{C}} \mathsf{T} \colon \mathsf{boole}} \qquad \frac{}{\Delta, \Gamma \vdash_{\mathcal{C}} \mathsf{F} \colon \mathsf{boole}}$$

$$\frac{}{\Delta, \Gamma \vdash_{\mathcal{C}} 0 \colon \mathsf{nat}} \qquad \frac{\Delta, \Gamma \vdash_{\mathcal{C}} M \colon \mathsf{nat}}{\Delta, \Gamma \vdash_{\mathcal{C}} \mathsf{S}(M) \colon \mathsf{nat}}$$

$$\frac{}{\Delta, \Gamma \vdash_{\mathcal{C}} [] \colon \mathsf{list}(\rho)} \qquad \frac{\Delta, \Gamma \vdash_{\mathcal{C}} M \colon \rho \qquad \Delta, \Gamma \vdash_{\mathcal{C}} N \colon \mathsf{list}(\rho)}{\Delta, \Gamma \vdash_{\mathcal{C}} \mathsf{cons}(M, N) \colon \mathsf{list}(\rho)}$$

Figure 1: The typing rules for extended System F

We let $\mathcal{T}_0$ be the set of closed types and define

$$\Delta \vdash_{\mathcal{C}}^{F} M \colon \rho \quad :\Leftrightarrow \quad \Delta, \emptyset \vdash_{\mathcal{C}} M \colon \rho \text{ and } \rho \in \mathcal{T}_0$$

Proposition 5.1. $(\mathcal{T}_0, \vdash^{F})$ *is a strongly normalising type system.*

Proof. Clearly $(\mathcal{T}_0, \vdash^{F})$ is a type system. In order to see that it is strongly normalising one easily adapts the proof of strong normalisation for system F as given, for example in [Bar92] (which is based on Girard's proof [Gir71] in the λ-calculus version due to Tait [Tai75]), so as to accommodate stratified rewrite systems. We leave details to the reader. A corresponding proof for simple types is worked out in detail in [Ber04b]. $\qquad \square$

Our next task is to interpret types in the domain D defined in section 3 (for the specific set of constructors $\mathcal{CO}$ above). We define the element $\mathtt{dummy} \in D$, which was left unspecified in section 3, recursively by

$$\mathtt{dummy} = \mathtt{abst}(\lambda a \in D.\mathtt{dummy})$$

We set $\widetilde{\mathbb{B}} := \{\mathsf{T}, \mathsf{F}\} \cup \mathtt{abst}(D)$, where $\mathtt{abst}(D) := \{\mathtt{abst}(d) \mid d \in D\}$, and $\widetilde{\mathbb{N}} :=$ the least subset of D that contains $\{0\} \cup \mathtt{abst}(D)$ and is closed under the constructor S. Furthermore, for a subset $A \subseteq D$ we set $\widetilde{\mathsf{list}}(A) :=$ the least subset of D that contains $\{\mathtt{nil}\} \cup \mathtt{abst}(D)$ and contains with d also $\mathtt{cons}(a, d)$ for every $a \in A$. Finally, for $A, B \subseteq D$ we set $A \to B :=$ $\{\mathtt{abst}(f) \in D \mid \forall a \in D \, (a \in A \to f(a) \in B)\}$. Set

$$\widetilde{\mathcal{P}}(D) := \{A \subseteq D \mid \mathtt{dummy} \in A, \bot \notin A\}$$

Note that $\widetilde{\mathcal{P}}(D)$ contains $\widetilde{\mathbb{B}}, \widetilde{\mathbb{N}}$ and is closed under arbitrary nonempty intersections and under the operations $A \mapsto \widetilde{\mathsf{list}}(A)$ and $(A, B) \mapsto A \to B$ (it is precisely the latter closure condition together with the requirement that the intersection of all types has to be nonempty that leads to the somewhat mysterious definition of $\mathtt{dummy}$).

For every type $\rho \in \mathcal{T}$ and type variable assignment $\tau \colon \mathsf{TV} \to \widetilde{\mathcal{P}}(D)$ we define $[\![\rho]\!]\tau \in$ $\widetilde{\mathcal{P}}(D)$ by recursion on ρ:

$$
\begin{aligned}
[\![p]\!]\tau &= \tau(p) \\
[\![\mathsf{boole}]\!]\tau &= \widetilde{\mathbb{B}} \\
[\![\mathsf{nat}]\!]\tau &= \widetilde{\mathbb{N}} \\
[\![\mathsf{list}(\rho)]\!]\tau &= \widetilde{\mathsf{list}}([\![\rho]\!]\tau) \\
[\![\rho \to \sigma]\!]\tau &= [\![\rho]\!]\tau \to [\![\sigma]\!]\tau \\
[\![\forall p\, \rho]\!]\tau &= \bigcap_{A \in \widetilde{\mathcal{P}}(D)} [\![\rho]\!]\tau_p^A
\end{aligned}
$$

Lemma 5.2. $[\![.]\!]$, *restricted to closed types, is a type interpretation for* $(\mathcal{T}_0, \vdash^F)$.

Proof. Call $\eta \in D^{\mathsf{Var}}$ total for a type assignment $\tau \colon \mathsf{TV} \to \widetilde{\mathcal{P}}(D)$ and a context Γ if $\eta(x) \in$ $[\![\Gamma(\rho)]\!]\tau$ for every $x \in \mathsf{dom}(\Gamma)$. By a straightforward induction on typing derivations one shows that if $\Delta, \Gamma \vdash_{\mathcal{C}} M \colon \rho$, then $[M]^\alpha \eta \in [\![\rho]\!]\tau$ for all τ and all α, η that are total for Δ, Γ. $\qquad \square$

Now let $\mathcal{C}$ consist of the constants $\mathsf{if}, <, \mathsf{lh}, \mathsf{get}, +\!\!+, \Phi, \Psi$ and a constant for every higher-type primitive recursive functional. Let **MBR** be the rewrite system consisting of the rewrite rules of section 2 and the usual rewrite rules for primitive recursion. The typing Δ for the constants is as expected. For example, writing $c \colon \rho$ for $\Delta(c) = \rho$,

$$
\begin{aligned}
\mathsf{if} &\;:\; \forall p \,.\mathsf{boole} \to p \to p \\
< &\;:\; \mathsf{nat} \to \mathsf{nat} \to \mathsf{boole} \\
\mathsf{lh} &\;:\; \forall p \,.\mathsf{list}(p) \to \mathsf{nat} \\
\mathsf{get} &\;:\; \forall p \,.\mathsf{list}(p) \to \mathsf{nat} \to p \\
+\!\!+ &\;:\; \forall p \,.\mathsf{list}(p) \to \mathsf{list}(p) \to \mathsf{list}(p) \\
\Phi &\;:\; \forall p \,.((\mathsf{nat} \to p) \to \mathsf{nat}) \to ((p \to \mathsf{nat}) \to p) \to \mathsf{list}(p) \to \mathsf{nat} \\
\Psi &\;:\; \forall p \,.((\mathsf{nat} \to p) \to \mathsf{nat}) \to ((p \to \mathsf{nat}) \to p) \to \mathsf{list}(p) \to \mathsf{nat} \to \mathsf{boole} \to p
\end{aligned}
$$

Lemma 5.3. **MBR** *is type-sound for* $\vdash^F$ *and* Δ.

Proof. Immediate, by inspection of the rewrite rules. $\qquad \square$

Theorem 5.4. *System F extended by Gödel primitive recursion and modified bar recursion is strongly normalising.*

Proof. By theorem 4.1, proposition 5.1 and lemmas 5.2 and 5.3 it suffices to show that $\alpha_{\mathbf{MBR}}$ is total. The proof of totality of $\alpha_{\mathbf{MBR}}(c)$ for constants $c \in \{\mathsf{if}, <, \mathsf{lh}, \mathsf{get}, +\!+\}$ and, more generally, any primitive recursive constant is easy and left to the reader. In the following we will write c instead of $\alpha_{\mathbf{MBR}}(c)$, $\widehat{a}(\vec{b})$ instead of $\mathtt{strict}(\mathtt{app})(a, \vec{b})$ and $\widehat{\lambda}x.e$ instead of $\mathtt{strict}(\mathtt{abst})(\lambda x.e)$. We will also write $[x_0, \dots, x_{n-1}]$ for $\mathtt{cons}(x_0, \dots, \mathtt{cons}(x_{n-1}, \mathtt{nil}))$ and call such objects *proper lists*. According to the rewrite rules for Φ and Ψ we have $\Phi = \mathtt{abst}(\varphi)$ and $\Psi = \mathtt{abst}(\psi)$ where for total arguments y, g, s, k

$$
\begin{aligned}
\varphi(y, g, s) &= \widehat{y}(\widehat{\lambda}k.\psi(y, g, s, k, k < \mathsf{lh}(s))) \\
\psi(y, g, s, k, \mathsf{T}) &= \mathsf{get}(g, k) \\
\psi(y, g, s, k, \mathsf{F}) &= \widehat{g}(k, \widehat{\lambda}x.\varphi(y, g, s * x))) \\
\psi(y, g, s, k, b) &= \mathtt{dummy}, \text{ if } \mathtt{case}(b) \in (\mathcal{CO} \setminus \{\mathtt{co}\}) \cup \{\mathtt{abst}\}
\end{aligned}
$$

More precisely, since Φ and Ψ have universal types, we chose an arbitrary set $A \in \widetilde{\mathcal{P}}(D)$ and take $y \in (\widetilde{\mathbb{N}} \to A) \to \widetilde{\mathbb{N}}$, $g \in (A \to \widetilde{\mathbb{N}}) \to A$, $s \in \widetilde{\mathsf{list}}(A)$, and show $\varphi(y, g, s) \in \widetilde{\mathbb{N}}$. We may assume that s is a proper list, i.e. $s = [x_0, \dots, x_{n-1}]$ with $x_i \in A$ ($i < n$), because for other $s \in \widetilde{\mathsf{list}}(A)$ we clearly have $\varphi(y, g, s) = \mathtt{dummy} \in \widetilde{\mathbb{N}}$. Assume $\varphi(y, g, s) \notin \widetilde{\mathbb{N}}$. Then there must be some $k \in \widetilde{\mathbb{N}}$ such that $k < \mathsf{lh}(s) = \mathsf{F}$ and $\psi(y, g, s, k, \mathsf{F}) \notin A$. The latter can only happen if $\varphi(y, g, [x_0, \dots, x_{n-1}, x_n]) \notin \widetilde{\mathbb{N}}$ for some $x_n \in A$. Repeating this argument one obtains an infinite sequence of elements $x_n, x_{n+1}, \dots$ such that each $x_i \in A$ and $\varphi(y, g, [x_0, \dots, x_m]) \notin \widetilde{\mathbb{N}}$ for any $m \geq n$. Define a continuous function $f \colon D \to D$

$$
f(k) := \begin{cases} x_m & \text{if } k = \mathsf{S}^m(0) \\ \mathtt{dummy} & \text{if } k = \mathsf{S}^m(\mathtt{abst}(a)) \text{ for some } m \in \mathbb{N} \text{ and } a \in D \\ \bot & \text{otherwise} \end{cases}
$$

Clearly $\mathtt{abst}(f) \in \widetilde{\mathbb{N}} \to A$. Hence $\widehat{y}(\mathtt{abst}(f)) \in \widetilde{\mathbb{N}}$. Since $\lambda a.\widehat{y}(a)$ is continuous and $\widetilde{\mathbb{N}}$ is an open subset of D there is a finite (compact) approximation $f_0 \neq \bot$ of f such that $\widehat{y}(\mathtt{abst}(f_0)) = \widehat{y}(\mathtt{abst}(f))$. From the first equation for ψ it follows that there is some $m \geq n$ such that for $s' := [x_0, \dots, x_m]$ we have $\mathtt{abst}(f_0) \sqsubseteq \widehat{\lambda}k.\psi(y, g, s', k, k < \mathsf{lh}(s'))$. Therefore $\varphi(y, g, s') = \widehat{y}(\widehat{\lambda}k.\psi(y, g, s', k, k < \mathsf{lh}(s'))) \in \widetilde{\mathbb{N}}$ which is a contradiction. $\qquad\square$

We conclude with a brief discussion of other rewrite systems which have been used to interpret strong classical analytical principles and for which our method works as well. In [BBC98] the following recursion was considered, which can be viewed as a more efficient 'demand driven' variant of modified bar recursion. As with modified bar recursion we use an auxiliary constant replacing the if-then-else construct used in [BBC98, Ber04a]:

$$
\begin{aligned}
\Phi ygs &\mapsto y(\lambda n.\Psi ygs(n \in \mathsf{dom}(s))) \\
\Psi ygs\mathsf{T} &\mapsto f[n] \\
\Psi ygs\mathsf{F} &\mapsto gn(\lambda z.\Phi yg(s*(n, z)))
\end{aligned} \tag{5.1}
$$

where $y \colon (\mathsf{nat} \to \rho) \to \mathsf{nat}$, $g \colon \mathsf{nat} \to (\rho \to \mathsf{nat}) \to \rho$ and $s \colon (\mathsf{nat} \times \rho)^*$ is to be viewed as the graph of a finite function with $n \in \mathsf{dom}(s)$ and $s[n]$ having the expected meanings. In [Ber04a] it was shown that (5.1) can be derived from the following principle of *open*

recursion (again formulated with an auxiliary constant):

$$
\begin{aligned}
\Phi y\alpha &\mapsto y\alpha(\lambda n, z.\Psi y\alpha n z(z \prec \alpha n)) \\
\Psi y\alpha n z\mathsf{T} &\mapsto \lambda\gamma.\Phi y(\overline{\alpha}n{*}z@\gamma) \\
\Psi y\alpha n z\mathsf{F} &\mapsto \lambda\gamma.0
\end{aligned} \tag{5.2}
$$

Here $\prec\colon \rho \times \rho \to \mathsf{boole}$ is (the graph of) a wellfounded relation, $\alpha, \gamma\colon \mathsf{nat} \to \rho$, $\overline{\alpha}n = [\alpha 0, \dots, \alpha(n-1)]$ and $s@\gamma = \lambda k.\mathsf{if}\ k < |s|\ \mathsf{then}\ s_k\ \mathsf{else}\ \gamma k$. So, Φ is recursively called with arguments of the form $\overline{\alpha}n{*}z@\gamma$, which are lexicographically smaller than α. Note, however, that the lexicographic ordering on infinite sequences is not wellfounded. Both recursions, (5.1) and (5.2) have the proof-theoretic strengths of full second order arithmetic. Their significance rests on the fact that they can be used to give rather direct realisability interpretations of strong classical theories: (5.1) realises classical countable choice [BBC98], while (5.1) realises *open induction* [Ber04a], a principle closely related to Nash-William minimal-bad-sequence argument [NW63].

By theorem 4.1 and the results of this section, the strong normalisability of system F plus the recursions above boils down to showing that the interpretations of (5.1) and (5.2) are total. The totality of (5.2) can be shown by open induction. To prove totality of (5.1) it is easiest to use the reduction to (5.2) given in [Ber04a].

References

[AJ94]   S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.

[Alt93]   T. Altenkirch. *Constructions, Inductive Types and Strong Normalization*. PhD thesis, The University of Edinburgh, Department of Computer Science, Edinburgh, 1993.

[Bar92]   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.

[BBC98]   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.

[Ber04a]   U. Berger. A computational interpretation of open induction. In F. Titsworth, editor, *Proceedings of the Ninetenth Annual IEEE Symposium on Logic in Computer Science*, pages 326–334. IEEE Computer Society, 2004.

[Ber04b]   U. Berger. Strong normalization proofs based on continuous semantics, 2004.

[Bez85]   M. Bezem. Strong normalization of barrecursive terms without using infinite terms. *Archive for Mathematical Logic*, 25:175–181, 1985.

[BJO99]   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.

[BO05]   U. Berger and P. Oliva. Modified bar recursion and classical dependent choice. In *Logic Colloquium 2001*. Springer, 2005.

[CU93]   S. Cook and A. Urquhart. Functional interpretations of feasibly constructive arithmetic. *Annals of Pure and Applied Logic*, 63:103–200, 1993.

[ea86]   R.L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice–Hall, New Jersey, 1986.

[Gir71]   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.

[GLSH93]   E. Griffor, I. Lindström, and V. Stoltenberg-Hansen. *Mathematical theory of domains*. Cambridge University Press, 1993.

[Göd58]   K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958.

[NW63]   C.St.J.A. Nash-Williams. On well-quasi-ordering finite trees. *Proc. Cambridge Phil. Soc.*, 59:833–835, 1963.

[Plo77]   G.D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[PM93]   C. Paulin-Mohring. Inductive definitions in the system Coq; rules and properties. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, pages 328–345. LNCS Vol. 664, 1993.

[Spe62]   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.

[Tai71]   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.

[Tai75]   W.W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Logic Colloquium Boston 1971/72*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer Verlag, Berlin, Heidelberg, New York, 1975.

[Tro73]   A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer, 1973.

[vdPS95] 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.

[Vog85]   H. Vogel. Ein starker Normalisationssatz für die barrekursiven Funktionale. *Archive for Mathematical Logic*, 18:81–84, 1985.