A domain model characterising strong normalisation

Ulrich Berger*

April 30, 2007

Abstract

Building on previous work by Coquand and Spiwack [CS06] we construct a strict domain-theoretic model for the untyped λ-calculus with constructors and recursively defined constants with the property that a term is strongly normalising if its value is not $\bot$. There are no disjointness or confluence conditions imposed on the rewrite rules, and under a mild but necessary condition completeness of the method is proven. Hence, we properly extend the results in [CS06]. As an application, we prove strong normalisation for barrecursion in higher types combined with non-deterministic choice.

1 Introduction

Modern functional programming languages like ML or Haskell, which support the definition of functions by λ-abstraction, pattern matching and recursion, can be modelled by suitable extended λ-calculi in order to analyse their fundamental properties. In this paper we study such an extension, the type-free λ-calculus with constructors, constants and rewrite rules, and present new results about a domain-theoretic method for proving strong normalisiation. The method was introduced in [Ber05a] and [Ber05b], and later improved in [CS06]. Its essential ideas are rooted in three strands of work:

(1) The adequacy theorem for PCF [Plo77]: If a closed PCF-term of base type denotes a numeral in the domain model, then it weak head reduces to that numeral.

(2) The characterisation of strongly normalising (pure) λ-terms by intersection types [Pot80].

(3) The observation that intersection types can be used to construct a domain-theoretic semantics (filter model) of λ-terms [BCDC83, vB92].

*University of Wales Swansea, Email: u.berger@swan.ac.uk

In [Ber05a], the adequacy result (1) was extended to strong normalisation for all terms by making the domain model strict: If a term is defined, i.e. does not denote $\bot$, then it is strongly normalising. The proof used the assumption that strong normalisiation holds for an underlying typed λ-calculus without rewrite rules, an assumption which Coquand and Spiwack [CS06] were able to eliminate, by adapting the method of reducibility candidates for intersection types (2) and the domain construction (3).

In this paper, we further improve the method by removing the disjointness resp. confluence condition on rewrite systems that was imposed in [Ber05a] and [CS06]. For example, rules for a non-deterministic choice operator

$$x \parallel y \to x \qquad\qquad x \parallel y \to y$$

are now allowed. The only restriction that remains is left linearity, i.e. variables must not occur more than once on the left hand side of a rewrite rule, and finite branching, i.e. every constant must have finitely many rules only. Furthermore, we prove completeness under a mild but necessary condition: If all constants are fully applied, i.e. receive as many arguments as specified by their arity, then all strongly normalising terms are defined.

In order to be able to interpret rules like those for the choice operator $\parallel$ above, our domain model accommodates finite non-determinism in the form of strict lists. We construct our model by solving a recursive domain equation involving standard operations on domains within an abstract order-theoretic setting. We believe that this is a very economic and transparent way of presenting the construction. If one wishes to make the constructive aspects if this model more explicit one could easily redefine it in the style of [CS06] as a filter model over an inductively defined set of formal compacts.

In order for our method to be useful it is important that terms are interpreted in a natural way, i.e. close to their intended meaning. For example, the terms of a simply typed system such as Gödel's system T are naturally interpreted in a hierarchy of total functionals of finite types. In the deterministic models in [Ber05a] and [CS06] this hierarchy is mimicked by interpreting types as certain subsets of the model not containing $\bot$ and showing that every constant (for example, the primitive recursion operator) is total, i.e. has a value in its respective type. Since totality is compositional, it follows that all terms are total, hence defined and therefore strongly normalising. In the deterministic model [Ber05b, CS06] the interpretation of the constants is indeed very close to the intended interpretation except that one has to take care of strictness. In our non-deterministic model one has to deal in addition with "fuzzy numbers", but still, as our case studies with non-deterministic extensions of system T and barrecursion show, the totality proofs for the constants are fairly smooth.

Related work. There exist many approaches to combining λ-calculus and term rewriting. Let us briefly point out some similarities and differences. While our rules are of the form

$$f\,\vec{P} \to M$$

where f is a constant, $\vec{P}$ is a linear constructor pattern (built form from variables and constructors without the use of λ-abstraction) and M is an arbitrary term with $\mathsf{FV}(M) \subseteq \mathsf{FV}(\vec{P})$, the algebraic λ-calculus [BT88, Dou92] allows only algebraic terms in rewrite rules, but is not restricted to patterns on the left hand side of rules (permitting for example rules like $(x+y)+z \to x+(y+z)$). Other approaches allow more general patterns, but are typed or have other syntactic restrictions [Nip91, Mil91, Bla05, BJO99]. A strictly more general approach is Jay's pattern calculus [Jay04], but little seems to be known about strong normalisation in this calculus.

2 The type free λ-calculus with constructors and function constants

Definition 2.1 (Terms). *The set* Λ *of (untyped)* λ-terms *with constructors, pairs and constants is defined as follows:*

$$\Lambda \ni M, N ::= x \mid c \mid f \mid (M, N) \mid \lambda x.M \mid MN \mid \mathbf{if}(M, N)$$

where x *ranges over a set* Var *of* variables*,* c *ranges over a set* $\mathcal{C}$ *of* constructors *which is assumed to contain at least the constructors* T *and* F*, and* f *ranges over a set* $\mathcal{F}$ *of* constants*.*

The usual notational conventions apply, e.g. if M is a term and $\vec{N} = N_1, \ldots, N_k$ is a tuple of terms, then $M\vec{N}$ stands for $(\ldots(MN_1)\ldots)M_k$.

The idea behind constructors and pairs is that they are used to construct data. For example, the natural numbers $0, 1, 2, \ldots$ can be represented by the terms 0, $(\mathrm{S}, 0)$, $(\mathrm{S}, (\mathrm{S}, 0))$, ..., where 0 and S are constructors. Pairs could be eliminated by replacing terms of the form (M, N) by $\mathrm{cons}\, M\, N$ were cons is a constructor. However, pairs have some technical advantages, and we think that they make the distinction between data and the algorithmic part of the calculus clearer. Also, **if**-terms are included just for convenience (they sometimes allow for shorter and more readable formulations of rewrite systems, for example barrecursion in Sect. 5) and could be eliminated by replacing terms of the form $\mathbf{if}(M, N)$ by $f\,\vec{x}$, where $\vec{x} = \mathsf{FV}(M, N)$ and f is a new constant with the rewrite rules $f\,\vec{x}\,\mathsf{T} \to M$ and $f\,\vec{x}\,\mathsf{F} \to N$.

Definition 2.2 (Rewriting). *A* pattern *is a term built from variables, constructors and pairing. A list of patterns is* linear *if every variable occurs at most once in it.*

A rewrite system *is determined by assigning to every constant* $f \in \mathcal{F}$ *a number* $\mathrm{arity}(f)$ *and a finite list* $\mathcal{R}_f$ *of* rules *of the form* $\vec{P} \mapsto M$*, where* $\vec{P} = [P_1, \ldots, P_{\mathrm{arity}(f)}]$ *is a linear list of patterns and* M *is a term with* $\mathsf{FV}(M) \subseteq \mathsf{FV}(\vec{P})$*. We may assume that different rules are variable disjoint (this is no restriction because the variables in a rule are considered to be bound). We will often display a rule* $\vec{P} \mapsto M \in \mathcal{R}_f$ *in the more familiar form* $f\vec{P} \to M$*.*

A term L can be contracted *to a term R as follows:*

L	R	
$(\lambda x.M)N$	$M[N/x]$	
$\mathbf{if}(M,N)\,\mathsf{T}$	M	
$\mathbf{if}(M,N)\,\mathsf{F}$	N	
$f\vec{P}\theta$	$M\theta$	$(\vec{P} \mapsto M \in \mathcal{R}_f,\ \theta$ *substitution*$)$

A term M reduces to M' ($M \to M'$) *if M' is obtained from M by contracting a subterm which is not a subterm of an* **if***-term.*

A term M is strongly normalising ($\mathsf{SN}(M)$) *if there is no infinite rewrite sequence starting with M, i.e. M is in the wellfounded part of the relation $\to$.*

As an example of a rewrite system, consider the rules

$$\begin{array}{lcl} m < 0 & \to & \mathsf{F} \\ 0 < (\mathrm{S}, n) & \to & \mathsf{T} \\ (\mathrm{S}, m) < (\mathrm{S}, n) & \to & m < n \end{array}$$

where 0 and S are constructors and $<$ is a binary constant. Note that this stands for $\mathcal{R}_< = [[m, 0] \mapsto \mathsf{F}, [0, (\mathrm{S}m)] \mapsto \mathsf{T}, [(\mathrm{S}, m), (\mathrm{S}, n)] \mapsto m < n]$. The rules for $<$ are an instance of *primitive recursion* which is captured in general by the *primitive recursion operator* R with the rules

$$\begin{array}{lcl} \mathrm{R}\ x\ y\ 0 & \to & x \\ \mathrm{R}\ x\ y\ (\mathrm{S}, z) & \to & y\ z\ (\mathrm{R}\ x\ y\ z) \end{array}$$

The rules considered so far are deterministic in the sense that different rules have non-unifiable left hand sides and hence cannot be used to contract the same term. The following rules for a *choice operator* $\parallel$ are non-deterministic:

$$\begin{array}{lcl} x \parallel y & \to & x \\ x \parallel y & \to & y \end{array}$$

While the rules for the constants $<$ and R can be immediately viewed as recursive definitions of corresponding functions, it is less clear what kind of function should be defined by the rules for the choice operator. The model defined in the next section will interpret $\parallel$ as concatenation of non-determisitic values represented by lists.

3 Strict semantics for strong normalisation

By a *domain* we mean in this paper an algebraic bounded complete dcpo. We denote the partial ordering on a domain D by $\sqsubseteq_D$, the least element (which always exists) by $\bot_D$ (we will often omit the subscript D), and the set of compact elements by D_0. We set $D_+ = D \setminus \{\bot\}$ (the set of "defined" elements) and $D_{0+} = D_0 \setminus \{\bot\}$. If A is a set, then the flat domain $A_\bot = A \cup \{\bot\}$ has a new

bottom element $\bot$ and the elements of A as pairwise incomparable members. A function $f\colon D \to E$ is (Scott-)continuous if it is monotone and preserves directed suprema. We set $\mathrm{dom}(f) := \{d \in D \mid f(d) \neq \bot_E\}$. We call f *strict* if $f(\bot_D) = \bot_E$, i.e. $\mathrm{dom}(f) \subseteq D_+$. The domain constructions, product $D \times E$, and continuous function space $D \to E$ are defined as usual. In the following we will need their strict versions, as well as strict lists and the coalesced sum:

$$\begin{aligned} D \times_{\mathrm{s}} E &:= \{(d,e) \mid d \in D_+, e \in E_+\} \cup \{\bot\} \\ D \to_\bot E &:= \{f\colon D \to E \mid f \text{ continuous and strict}\} \\ D^* &:= \{[d_1,\ldots,d_n] \mid n \in \mathbb{N},\ d_i \in D_+\} \cup \{\bot\} \\ D^1 \oplus \ldots \oplus D^n &:= \{\mathrm{in}_i(d_i) \mid i \in \{1,\ldots,n\}, d_i \in D_+^i\} \cup \{\bot\} \end{aligned}$$

Pairs, lists and strict functions are ordered component-wise resp. pointwise. The order on the coalesced sum is defined by $\mathrm{in}_i(d) \sqsubseteq \mathrm{in}_j(e)$ iff $i = j$ and $d \sqsubseteq e$. The elements of the domain D^* should be viewed as non-deterministic or 'fuzzy' elements of D. If A is a subset of D_+, then $A^* \subseteq D^*$ is the set of all finite lists with elements in A. In the definitions below we use the fact that in order to define a strict continuous function $f\colon D \to_\bot E$ it suffices to define $f(d)$ for $d \in D_+$. Furthermore, we use the convention that $[d_1,\ldots,d_n]$ denotes $\bot$ if one of the d_i is $\bot$.

$\mathrm{strict}_\times$	:	$(D \times E) \to_\bot (D \times_{\mathrm{s}} E)$	
$\mathrm{strict}_\to$	:	$(D \to E) \to_\bot (D \to_\bot E)$	
$(+\!\!+)$	:	$D^* \to_\bot D^* \to_\bot D^*$	
concat	:	$(D^*)^* \to_\bot D^*$	
map	:	$(D \to_\bot E) \to (D^* \to_\bot E^*)$	
$(\rhd)$	:	$\{\mathsf{T},\mathsf{F}\}_\bot \to_\bot D^* \to D^*$	
$\mathrm{strict}_\times(d,e)$	=	(d,e)	$(d,e \in D_+)$
$\mathrm{strict}_\times(\bot,d) = \mathrm{strict}_\times(d,\bot)$	=	$\bot$	$(d \in D)$
$\mathrm{strict}_\to(f)(d)$	=	$f(d)$	$(d \in D_+)$
$[d_1,\ldots,d_n] +\!\!+ [e_1,\ldots,e_m]$	=	$[d_1,\ldots,d_n,e_1,\ldots,e_m]$	$(d_i,e_i \in D_+)$
concat $[\mathbf{d}_1,\ldots,\mathbf{d}_n]$	=	$\mathbf{d}_1 +\!\!+ \ldots +\!\!+ \mathbf{d}_n$	$(\mathbf{d}_i \in D_+^*)$
$\mathrm{map}(f)[d_1,\ldots,d_n]$	=	$[f(d_1),\ldots,f(d_n)]$	$(d_i \in D_+)$
$\mathsf{T} \rhd \mathbf{d}$	=	$\mathbf{d}$	$(\mathbf{d} \in D^*)$
$\mathsf{F} \rhd \mathbf{d}$	=	$[]$	$(\mathbf{d} \in D^*)$

Note that the functions map and $(\rhd)$ are non-strict in their first and second argument, respectively, since $\mathrm{map}(\bot)[] = []$ and $\mathsf{F} \rhd \bot = []$.

For ease of readability, we will use Haskell's *list comprehension* notation $[f(d) \mid d \leftarrow \mathbf{d}] = \mathrm{map}(f)(\mathbf{d})$ as well as its iterated form

$$\begin{aligned} & [f(d,d_1,\ldots,d_n) \mid d \leftarrow \mathbf{d}, d_1 \leftarrow g_1(d), \ldots, d_n \leftarrow g_n(d,d_1,\ldots,d_{n-1})] \\ = \ & [\,[f(d,d_1,\ldots,d_n) \mid d_1 \leftarrow g_1(d), \ldots, d_n \leftarrow g_n(d,\ldots,d_{n-1})] \mid d \leftarrow \mathbf{d}]. \end{aligned}$$

In the category of domains with embeddings the domain operations above are (more precisely, can be extended to) covariant continuous functors. Furthermore, since the category of domains has directed colimits, we can solve recursive domain equations (see e.g. [GHK+03]).

Definition 3.1 (Strict domain model). *For the denotational semantics of terms we use the following recursively defined domain*

$$D = \mathcal{C}_\perp \oplus (D^* \times_s D^*) \oplus (D^* \to_\perp D^*)$$

where "=" means of course "isomorphic".

The elements of D_+ are $\mathrm{in}_1(c)$ ($c \in \mathcal{C}$), $\mathrm{in}_2(\mathbf{d},\mathbf{e})$ ($\mathbf{d},\mathbf{e} \in D_+^*$), and $\mathrm{in}_3(f)$ ($f \in (D^* \to_\perp D^*)_+$). We will abbreviate $\mathrm{in}_1(c)$ by c, and extend the list comprehension notation by setting

$$[g(\mathbf{d}_1,\mathbf{d}_2) \mid (\mathbf{d}_1,\mathbf{d}_2) \leftarrow \mathbf{d}] = \mathrm{map}(g)(\mathrm{concat}(\mathrm{map}(\mathrm{split})(\mathbf{d})))$$

where $\mathrm{split}\colon D \to_\perp (D^* \times D^*)^*$ is defined by $\mathrm{split}(\mathrm{in}_2(\mathbf{d},\mathbf{e})) = [(\mathbf{d},\mathbf{e})]$ and $\mathrm{split}(d) = []$ if $d \in D_+$ is not of the form $\mathrm{in}_2(\mathbf{d},\mathbf{e})$. Furthermore, we define the strict continuous functions

$$\begin{array}{lcl}
\mathrm{pair} & : & D^* \times D^* \to_\perp D \\
\mathrm{fun} & : & (D^* \to D^*) \to_\perp D \\
\mathrm{fun}^k & : & ((D^*)^k \to D^*) \to_\perp D \quad (k \geq 1) \\
(\bullet) & : & D^* \times D^* \to_\perp D^* \\
\mathrm{pair}(\mathbf{d},\mathbf{e}) & = & \mathrm{in}_2(\mathrm{strict}_\times(\mathbf{d},\mathbf{e})) \\
\mathrm{fun}(f) = \mathrm{fun}^1(f) & = & \mathrm{in}_3(\mathrm{strict}_\to(f)) \\
\mathrm{fun}^{k+1}(f) & = & \mathrm{fun}(\lambda \mathbf{d} \in D^*.[\mathrm{fun}^k(\lambda \vec{\mathbf{e}} \in (D^*)^k.f(\mathbf{d},\vec{\mathbf{e}}))]) \\
\mathbf{d} \bullet \mathbf{e} & = & \mathrm{concat}[f(\mathbf{e}) \mid \mathrm{fun}(f) \leftarrow \mathbf{d}]
\end{array}$$

In order to define the semantics of constants we need to match linear patterns P against non-deterministic values. For a set X of variables let VEnv_X denote the pointwise ordered domain of environments η with $\mathrm{dom}(\eta) = X$, i.e. $\eta\colon X \to D^*$.

Definition 3.2 (Semantic matching). *We define* $\mathrm{match}_P\colon D^* \to \mathsf{VEnv}^*_{\mathsf{FV}(P)}$ *by*

$$\begin{array}{lcl}
\mathrm{match}_x(\mathbf{d}) & = & [[x \mapsto \mathbf{d}]] \text{ where } \mathrm{dom}\,[x \mapsto \mathbf{d}] = \{x\} \text{ and } [x \mapsto \mathbf{d}](x) = \mathbf{d} \\
\mathrm{match}_c(\mathbf{d}) & = & (c \in \mathbf{d}) \rhd [\emptyset] \\
\mathrm{match}_{(P,Q)}(\mathbf{d}) & = & [\eta \cup \eta' \mid (\mathbf{e},\mathbf{e}') \leftarrow \mathbf{d}, \eta \leftarrow \mathrm{match}_P(\mathbf{e}), \eta' \leftarrow \mathrm{match}_Q(\mathbf{e}')]
\end{array}$$

For a linear tuple $\vec{P} = P_1,\ldots,P_k$ *of patterns we set*

$$\mathrm{match}_{\vec{P}}(\mathbf{d}_1,\ldots,\mathbf{d}_k) = [\eta_1 \cup \ldots \cup \eta_k \mid \eta_1 \leftarrow \mathrm{match}_{P_1}(\mathbf{d}_1),\ldots,\eta_k \leftarrow \mathrm{match}_{P_k}(\mathbf{d}_k)].$$

Definition 3.3 (Strict denotational semantics of terms). *The value* $[\![M]\!]\eta \in D^*$ *of a term with respect to an environment* $\eta\colon \mathrm{dom}(\eta) \to D^*$, *where* $\mathrm{dom}(\eta) \supseteq \mathsf{FV}(M)$, *is recursively defined by*

$$\begin{array}{lcl}
[\![x]\!]\eta & = & \eta(x) \\
[\![c]\!]_ & = & [c] \\
[\![(M,N)]\!]\eta & = & [\mathrm{pair}([\![M]\!]\eta,[\![N]\!]\eta)] \\
[\![MN]\!]\eta & = & [\![M]\!]\eta \bullet [\![N]\!]\eta \\
[\![\lambda x.M]\!]\eta & = & [\mathrm{fun}(\lambda \mathbf{d} \in D^*.[\![M]\!]\eta[x := \mathbf{d}])] \\
[\![\mathbf{if}(M,N)\,]\!]\eta & = & [\mathrm{fun}(\lambda \mathbf{d} \in D^*.(\mathsf{T} \in \mathbf{d} \rhd [\![M]\!]\eta) +\!\!+ (\mathsf{F} \in \mathbf{d} \rhd [\![N]\!]\eta))] \\
[\![f]\!]_ & = & [\mathrm{fun}^k(\lambda \vec{\mathbf{d}} \in (D^*)^k. \\
 & & \quad \mathrm{concat}[\,[\![M]\!]\eta \mid (\vec{P} \mapsto M) \leftarrow \mathcal{R}_f,\ \eta \leftarrow \mathrm{match}_{\vec{P}}(\vec{\mathbf{d}})\,]\,)]
\end{array}$$

where $k = \mathrm{arity}(f)$. *We set* $[\![M]\!] = [\![M]\!]\emptyset$ *if* M *is closed.*

Theorem 3.4 (Strong normalisation theorem). *If* $[\![M]\!] \neq \bot$, *then* M *is strongly normalising.*

We prove this theorem in the next section. In order for this theorem to be useful it is important that the value of a term corresponds to its intuitive meaning and can be calculated easily. For example, the values of the constants R and $\|$ introduced in Sect. 2 satisfy

$$\begin{aligned} [\![\mathrm{R}]\!] \bullet \mathbf{d}_1 \bullet \mathbf{d}_2 \bullet \mathbf{d}_3 &= ((0 \in \mathbf{d}_3) \rhd \mathbf{d}_1) \mathbin{+\!\!+} \\ &\quad \mathrm{concat}\,[\mathbf{d}_2 \bullet \mathbf{e} \bullet ([\![\mathrm{R}]\!] \bullet \mathbf{d}_1 \bullet \mathbf{d}_2 \bullet \mathbf{e}) \mid (\mathrm{S}, \mathbf{e}) \leftarrow \mathbf{d}_3] \\ [\![\,\|\,]\!] \bullet \mathbf{d} \bullet \mathbf{e} &= \mathbf{d} \mathbin{+\!\!+} \mathbf{e} \end{aligned}$$

for all $\mathbf{d}, \mathbf{d}_i, \mathbf{e} \in D_+^*$.

Remark. The model in [CS06] contains a top element used to interpret terms of the form $f\,\vec{M}$ that do not match any rule for f. In our model the role of the top element is taken over by the empty list.

4 Proof of the strong normalisation theorem

To prove Thm. 3.4 we use a version of the technique of reducibility candidates, where the usual role of types is taken over by elements of D_{0+}. The definitions are analogous to those in [CS06] except that we assign reducibility candidates to abstract (instead of formal) compact domain elements and replace structural recursion by recursion on the *rank* of compacts. Furthermore have to extend the assignment to "fuzzy" compacts.

A term is called *simple* if it is neither a constructor nor a pair nor a λ-abstraction nor an **if**-term nor of the form $f\vec{N}_1 \ldots N_k$ where $k < \mathrm{arity}(f)$. A *reducibility candidate* is a set X of terms such that

RC1 $X \subseteq \mathsf{SN}$.

RC2 If $M \in X$ and $M \to M'$, then $M' \in X$.

RC3 If M is simple and $\forall M'\,(M \to M' \Rightarrow M' \in X)$, then $M \in X$.

Let X and Y be sets of terms.

$X \to Y := \{M \mid \forall N\,(N \in X \Rightarrow MN \in Y)\}$.

$X \times Y := \{(M, N) \mid M \in X, N \in Y\}$ $(\subseteq \Lambda)$.

$\mathbf{RC3}(X) :=$ the closure of X under the rule **RC3** above.

Lemma 4.1. *If* X, Y *are reducibility candidates, then* $X \to Y$ *is a reducibility candidate. If* $\mathcal{X}$ *is a nonempty set of reducibility candidates, then* $\bigcap \mathcal{X}$ *is a reducibility candidate. Furthermore, if* $X \subseteq \Lambda$ *satisfies* **RC1** *and* **RC2**, *then* $\mathbf{RC3}(X)$ *is a reducibility candidate.*

In the following we let U, V, W range over D_{0+}, $\mathbf{U}, \mathbf{V}, \mathbf{W}$ over D^*_{0+}, and F, G over $((D^*)^k \to_\perp D^*)_{0+}$ (for various $k \geq 1$). We write $\mathbf{U} \subseteq \mathbf{V}$ if every member of $\mathbf{U}$ is a member of $\mathbf{V}$. Note that the element of D_{0+} are of the form c, $\mathrm{pair}(\mathbf{U}, \mathbf{V})$, or $\mathrm{fun}(F)$.

The next step is to assign to each $U \in D_{0+}$ a reducibility candidate $\Lambda(U)$. The definition of $\Lambda(U)$ proceeds by recursion on the the first stage in the construction of D where U appears. More precisely, let $D_0 = \{\perp\}$,

$$D_{n+1} = \mathcal{C}_\perp \oplus (D^*_n \times_{\mathrm{s}} D^*_n) \oplus (D^*_n \to_\perp D^*_n)$$

and $D = \lim_{n\in\mathbb{N}} D_n$. Let $\epsilon_n \colon D_n \to D$ be the embeddings implicit in the limit construction. For every $U \in D_{0+}$ there is an n such that U is in the image of ϵ_n. We call the least such n the *rank of* U and denote it by $\mathrm{rk}(U)$. We also set

$$\mathrm{rk}(\mathbf{U}) = \sup\{\mathrm{rk}(U) \mid U \in \mathbf{U}\}$$

(where $\sup \emptyset = 0$).

Lemma 4.2. $\mathrm{rk}(\mathbf{U}_i) < \mathrm{rk}(\mathrm{pair}(\mathbf{U}_1, \mathbf{U}_2))$. *Furthermore, if* $F(\mathbf{d}) \neq \perp$, *then* $\mathrm{rk}(F(\mathbf{d})) < \mathrm{rk}(\mathrm{fun}(F))$ *and* $F(\mathbf{d}) = F(\mathbf{U})$ *for some* $\mathbf{U} \sqsubseteq \mathbf{d}$ *with* $\mathrm{rk}(\mathbf{U}) < \mathrm{rk}(\mathrm{fun}(F))$.

Proof. Immediate, from the definition of $\mathrm{rk}(\mathbf{U})$ and from general properties of compact elements. □

By recursion on $\mathrm{rk}(U)$ and $\mathrm{rk}(\mathbf{U})$ we define sets of terms $\Lambda(U)$ and $\Lambda(\mathbf{U})$:

$$\begin{array}{lcl}
\Lambda(c) & = & \mathbf{RC3}(c) \\
\Lambda(\mathrm{pair}(\mathbf{U}, \mathbf{V})) & = & \mathbf{RC3}(\Lambda(\mathbf{U}) \times \Lambda(\mathbf{V})) \\
\Lambda(\mathrm{fun}(F)) & = & \bigcap\{\Lambda(\mathbf{U}) \to \Lambda(F(\mathbf{U})) \mid \mathbf{U} \in \mathrm{dom}(F), \mathrm{rk}(\mathbf{U}) < \mathrm{rk}(\mathrm{fun}(F))\} \\
\Lambda(\mathbf{U}) & = & \mathbf{RC3}(\bigcup\{\Lambda(U) \mid U \in \mathbf{U}\})
\end{array}$$

If $\vec{N} = N_1, \ldots, N_k$ and $\vec{\mathbf{U}} = \mathbf{U}_1, \ldots, \mathbf{U}_k$, then $\vec{N} \in \Lambda(\vec{\mathbf{U}})$ means $N_i \in \Lambda(\mathbf{U}_i)$ for all $i \in \{1, \ldots, k\}$.

Lemma 4.3. $\Lambda(U)$ *and* $\Lambda(\mathbf{U})$ *are reducibility candidates.*

Proof. Direct, by Lemma 4.1. □

Lemma 4.4. *If* $\mathbf{U} \subseteq \mathbf{V}$, *then* $\Lambda(\mathbf{U}) \subseteq \Lambda(\mathbf{V})$.

Proof. Immediate, by definition of $\Lambda(\mathbf{U})$. □

Lemma 4.5. *If* $\mathbf{U} \sqsubseteq \mathbf{V}$, *then* $\Lambda(\mathbf{U}) \supseteq \Lambda(\mathbf{V})$.

Proof. By induction on $\mathrm{rk}(V)$ we show that $U \sqsubseteq V$ implies $\Lambda(U) \supseteq \Lambda(V)$ (obviously, this suffices). The case c is trivial and the case $\mathrm{pair}(\mathbf{U}, \mathbf{V})$ is immediate by induction hypothesis. Now assume $\mathrm{fun}(F) \sqsubseteq \mathrm{fun}(G)$. Let $M \in \Lambda(\mathrm{fun}(G))$, let $\mathbf{U} \in \mathrm{dom}(F)$ and let $N \in \Lambda(\mathbf{U})$. It suffices to show that $MN \in \Lambda(F(\mathbf{U}))$. Since $F \sqsubseteq G$ we have $\mathbf{U} \in \mathrm{dom}(G)$ and therefore, by

Lemma 4.2, $\mathrm{rk}(G(\mathbf{U})) < \mathrm{rk}(\mathrm{fun}(G))$. Since $F(\mathbf{U}) \sqsubseteq G(\mathbf{U})$, we know, by induction hypothesis, that $\Lambda(F(\mathbf{U})) \supseteq \Lambda(G(\mathbf{U}))$. Therefore, it suffices to show $MN \in \Lambda(G(\mathbf{U}))$. By Lemma 4.2, we have $G(\mathbf{U}) = G(\mathbf{V})$ for some $\mathbf{V} \sqsubseteq \mathbf{U}$ with $\mathrm{rk}(\mathbf{V}) < \mathrm{rk}(\mathrm{fun}(G))$. By induction hypothesis, $N \in \Lambda(\mathbf{V})$. Since $M \in \Lambda(\mathrm{fun}(G))$, it follows $MN \in \Lambda(G(\mathbf{V})) = \Lambda(G(\mathbf{U}))$. □

Lemma 4.6. *If* $\mathbf{U} \bullet \mathbf{V} \neq \bot$*, then* $\Lambda(\mathbf{U}) \subseteq \Lambda(\mathbf{V}) \to \Lambda(\mathbf{U} \bullet \mathbf{V})$.

Proof. We show: If $M \in \Lambda(\mathbf{U})$ and $N \in \Lambda(\mathbf{V})$, then $MN \in \Lambda(\mathbf{U} \bullet \mathbf{V})$, by induction on $\mathsf{SN}(M, N)$.

Case M is simple: Then MN is simple, too. Hence, it suffices to show that all reducts of MN are in $\Lambda(\mathbf{U} \bullet \mathbf{V})$. Clearly, any reduction of MN must take place in M or in N. In either case, the induction hypothesis applies.

Case M is not simple: Then $M \in \Lambda(U)$ for some $U \in \mathbf{U}$. If U is of the form $\mathrm{fun}(F)$ with $\mathbf{V} \in \mathrm{dom}(F)$, then there is some $\mathbf{V}' \sqsubseteq \mathbf{V}$ with $\mathrm{rk}(\mathbf{V}') < \mathrm{rk}(\mathrm{fun}(F))$ and $F(\mathbf{V}') = F(\mathbf{V})$, by Lemma 4.2. By Lemma 4.5, $N \in \Lambda(\mathbf{V}')$. Hence, $MN \in \Lambda(F(\mathbf{V}')) = \Lambda(F(\mathbf{V})) \subseteq \Lambda(\mathbf{U} \bullet \mathbf{V})$, by Lemma 4.4. If U is a constructor or a pair, then M is a constructor or a pair as well and therefore MN is simple. Hence, it suffices to show that all reducts of MN are in $\Lambda(\mathbf{U} \bullet \mathbf{V})$. Since M is a constant or a pair, any reduction of MN must take place in M or in N. In either case, the induction hypothesis applies. □

Lemma 4.7. *If* $F \in ((D^*)^k \to D^*)_{0+}$ *(where* $k \geq 1$*) and* $M \in \Lambda$ *are such that* $M\vec{N} \in \Lambda(F(\vec{\mathbf{U}}))$ *for all* $\vec{\mathbf{U}} \in \mathrm{dom}(F)$ *and* $\vec{N} \in \Lambda(\vec{\mathbf{U}})$*, then* $M \in \Lambda(\mathrm{fun}^k(F))$.

Proof. Induction on k. For $k = 1$, this holds by definition of $\Lambda(\mathrm{fun}(F))$. Now assume $MN\vec{N} \in \Lambda(F(\mathbf{U}, \vec{\mathbf{U}}))$ for all $\mathbf{U}, \vec{\mathbf{U}} \in \mathrm{dom}(F)$ and $N \in \Lambda(\mathbf{U})$, $\vec{N} \in \Lambda(\vec{\mathbf{U}})$. Then for each $\mathbf{U}$ and $N \in \Lambda(\mathbf{U})$, the term MN and the function $F'(\mathbf{U}) := \lambda\vec{\mathbf{U}} \in (D^*).F(\mathbf{U}, \vec{\mathbf{U}})$ satisfy the hypothesis of the lemma and therefore $MN \in \Lambda(\mathrm{fun}^k(F'(\mathbf{U})))$. It follows that $M \in \Lambda(\mathrm{fun}(\lambda\mathbf{U}.[\mathrm{fun}^k(F'(\mathbf{U}))]))$. Since $\mathrm{fun}^{k+1}(F) = \mathrm{fun}(\lambda\mathbf{U}.[\mathrm{fun}^k(F'(\mathbf{U}))])$, we are done. □

Lemma 4.8. *If* $M[N/x] \in \Lambda(F(\mathbf{U}))$ *for all* $\mathbf{U} \in \mathrm{dom}(F)$ *and all* $N \in \Lambda(\mathbf{U})$*, then* $\lambda x.M \in \Lambda(\mathrm{fun}(F))$.

Proof. Induction on $\mathsf{SN}(M)$. Assume that M fulfils the assumption of the lemma. By definition of $\Lambda(\mathrm{fun}(F))$ it suffices to show that $(\lambda x.M)N \in \Lambda(F(\mathbf{U}))$ for all $\mathbf{U} \in \mathrm{dom}(F)$ and $N \in \Lambda(\mathbf{U})$. We show this by a side induction on $\mathsf{SN}(N)$. Since $(\lambda x.M)N$ is simple, it suffices to show that all reducts of this term are in $\Lambda(F(\mathbf{U}))$. If the reduction takes place in M, then we may use the main induction hypothesis, since any reduct of M will fulfil the assumption of the lemma as well (because reduction is closed under substitution and reducibility candidates are closed under reduction). If the reduction takes place in N, then we use the side induction hypothesis. Otherwise the reduct is $M[N/x]$ in which case we apply the assumption on M. □

Lemma 4.9. *If* $M \in \Lambda(F(\mathbf{U}))$ *whenever* $\mathsf{T} \in \mathbf{U}$*, and* $N \in \Lambda(F(\mathbf{U}))$ *whenever* $\mathsf{F} \in \mathbf{U}$*, then* $\mathbf{if}(M, N) \in \Lambda(\mathrm{fun}(F))$.

Proof. It suffices to show that for $\mathbf{U} \in \mathrm{dom}(F)$ and $K \in \Lambda(\mathbf{U})$, $\mathbf{if}(M,N)\,K \in \Lambda(F(\mathbf{U}))$. We show this by induction on $\mathsf{SN}(K)$. Because $\mathbf{if}(M,N)\,K$ is simple, it suffices to show that all reducts are in $\Lambda(F(\mathbf{U}))$. If K is reduced, then we can apply the induction hypothesis. If $K = \mathsf{T}$ (and $\mathbf{if}(M,N)\,\mathsf{T} \to M$), then $\mathsf{T} \in \mathbf{U}$, since the term T is not simple, and we are done. The case $K = \mathsf{F}$ is similar. □

If θ is a substitution and η an environment, then we write $\theta \in \Lambda(\eta)$ if $\mathrm{dom}(\theta) \supseteq \mathrm{dom}(\eta)$ and for all $x \in \mathrm{dom}(\eta)$, $\eta(x) \in D_{0+}$ and $\theta(x) \in \Lambda(\eta(x))$.

Lemma 4.10. *If $P\theta \in \Lambda(\mathbf{U})$, then $\theta \in \Lambda(\eta)$ for some $\eta \in \mathrm{match}_P(\mathbf{U})$.*

Proof. Induction on P. If P is a variable, then set $\eta := [x \mapsto \mathbf{U}]$. If P is a constructor c, then, by assumption, $c \in \Lambda(\mathbf{U}) = \mathbf{RC3}(c)$. Hence $c \in \mathbf{U}$ and therefore $\emptyset \in \mathrm{match}_c(\mathbf{U})$. For the induction step, assume $(P\theta, P\theta) \in \Lambda(\mathbf{U})$. Since $(P\theta, Q\theta)$ is not simple, $(P\theta, Q\theta) \in \Lambda(U)$ for some $U \in \mathbf{U}$. Clearly, U must be of the form $\mathrm{pair}(\mathbf{U}_1, \mathbf{U}_2)$ where $P\theta \in \mathbf{U}_1$ and $P\theta \in \mathbf{U}_2$. By induction hypothesis, there are $\eta_1 \in \mathrm{match}_P(\mathbf{U}_1)$ and $\eta_2 \in \mathrm{match}_Q(\mathbf{U}_2)$ such that $\theta \in \Lambda(\eta_1)$ and $\theta \in \Lambda(\eta_2)$. Hence $\eta := \eta_1 \cup \eta_2 \in \mathrm{match}_{(P,Q)}(U)$ and $\theta \in \Lambda(\eta)$. □

Proof of Theorem 3.4. Assume $[\![M]\!] \neq \bot$. Then $\mathbf{U} \sqsubseteq [\![M]\!]$ for some $\mathbf{U} \in D^*_{0+}$. Below, we show that this implies $M \in \Lambda(\mathbf{U})$. Since, by Lemma 4.3, $\Lambda(\mathbf{U})$ is a reducibility candidate, it follows that M is strongly normalising.

In order to show that $\mathbf{U} \sqsubseteq [\![M]\!]$ implies $M \in \Lambda(\mathbf{U})$ we need to prove a more general claim involving open terms, substitutions and environments. We also need to exploit the fact that the value of a term is defined as the least fixed point of a continuous function, which entails that $[\![M]\!]\eta$ is the supremum of an increasing sequence

$$\bot = [\![M]\!]^0\eta \sqsubseteq [\![M]\!]^1\eta \sqsubseteq [\![M]\!]^2\eta \sqsubseteq \ldots$$

where the definition of $[\![M]\!]^{n+1}\eta$ is obtained by replacing in the definition of $[\![M]\!]\eta$ every occurrence of $[\![\cdot]\!]$ on the left hand side of an equation by $[\![\cdot]\!]^{n+1}$ and on the right hand side by $[\![\cdot]\!]^n$ (for example $[\![MN]\!]^{n+1}\eta = [\![M]\!]^n\eta \bullet [\![N]\!]^n\eta$). Furthermore, since $\mathbf{U}$ is compact, $\mathbf{U} \sqsubseteq [\![M]\!]\eta$ implies that $\mathbf{U} \sqsubseteq [\![M]\!]^n\eta$ for some n. Therefore, it suffices to prove the following

> *Claim.* Let η be an environment with $\mathrm{dom}(\eta) \supseteq \mathsf{FV}(M)$ and let θ be a substitution such that $\theta \in \Lambda(\eta)$. Assume further $\mathbf{U} \sqsubseteq [\![M]\!]^n\eta$. Then $M\theta \in \Lambda(\mathbf{U})$.

We prove the claim by induction on n. The induction base is trivial since the hypothesis $\mathbf{U} \sqsubseteq [\![M]\!]^0\eta$ is false (because $\mathbf{U} \neq \bot$). In the step we consider the different forms of M:

Case x: By assumption, $\mathbf{U} \sqsubseteq \eta(x)$ and $\theta(x) \in \Lambda(\eta(x))$. Hence $\theta(x) \in \Lambda(\eta(x))$, by Lemma 4.5.

Case c: By assumption, $\mathbf{U} \sqsubseteq [c]$. Hence $\mathbf{U} = [c]$, and therefore $c \in \Lambda(\mathbf{U})$.

Case (M,N): Since $\bot \neq \mathbf{U} \sqsubseteq [\![(M,N)]\!]^{n+1}\eta$ we have $[\![M]\!]^n\eta \neq \bot$ and $[\![N]\!]^n\eta \neq \bot$, by strictness, and $[\![(M,N)]\!]^{n+1}\eta = [\mathrm{pair}([\![M]\!]^n\eta, [\![N]\!]^n\eta)]$. Hence

$\mathbf{U} = [\mathrm{pair}(\mathbf{U}_0, \mathbf{U}_1)]$ with $\mathbf{U}_0 \sqsubseteq [\![M]\!]^n\eta$ and $\mathbf{U}_1 \sqsubseteq [\![N]\!]^n\eta$. By induction hypothesis, $M\theta \in \Lambda(\mathbf{U}_0)$ and $N\theta \in \Lambda(\mathbf{U}_1)$. Hence $(M\theta, N\theta) \in \Lambda(\mathbf{U}_0) \times \Lambda(\mathbf{U}_1) \subseteq \Lambda(\mathbf{U})$.

Case MN: By assumption, $\bot \neq \mathbf{U} \sqsubseteq [\![(M,N)]\!]^{n+1}\eta = [\![M]\!]^n\eta \bullet [\![N]\!]^n\eta$. Hence $\mathbf{U}_0 \sqsubseteq [\![M]\!]^n\eta$ and $\mathbf{U}_1 \sqsubseteq [\![N]\!]^n\eta \neq \bot$ for some $\mathbf{U}_0, \mathbf{U}_1 \in D^*_{0+}$ with $\mathbf{U} \sqsubseteq \mathbf{U}_0 \bullet \mathbf{U}_1$, by strictness and continuity of $\bullet$. By induction hypothesis, $M\theta \in \Lambda(\mathbf{U}_0)$ and $N\theta \in \Lambda(\mathbf{U}_1)$. By Lemmas 4.5 and 4.6, $(MN)\theta \in \Lambda(\mathbf{U}_0 \bullet \mathbf{U}_1) \subseteq \Lambda(\mathbf{U})$.

Case $\lambda x.M$: By assumption, $\mathbf{U} \sqsubseteq [\![\lambda x.M]\!]^{n+1}\eta$. Hence $\mathbf{U} = [\mathrm{fun}(F)]$ for some $F \in (D^* \to D^*)_{0+}$. By Lemma 4.8, and since w.l.o.g. $(\lambda x.M)\theta = \lambda x.(M\theta)$, it suffices to show that $M\theta[N/x] \in \Lambda(F(\mathbf{V}))$ for all $\mathbf{V} \in \mathrm{dom}(F)$ and $N \in \Lambda(\mathbf{V})$. Hence, assume $N \in \Lambda(\mathbf{V})$. Since $\mathrm{fun}(F) \sqsubseteq [\![\lambda x.M]\!]^{n+1}\eta$, we have $F(\mathbf{V}) \sqsubseteq [\![M]\!]^n\eta[x := \mathbf{V}]$. Therefore, we may apply the induction hypothesis to $\eta[x := \mathbf{V}]$, $\theta[N/x]$, $F(\mathbf{V})$ and M, which gives us $M\theta[N/x] \in \Lambda(F(\mathbf{V}))$.

Case $\mathbf{if}(M,N)$: By assumption, $\mathbf{U} \sqsubseteq [\![\mathbf{if}(M,N)]\!]^{n+1}\eta$. Therefore, $\mathbf{U} = [\mathrm{fun}(F)]$ and

$$F(\mathbf{d}) \quad \sqsubseteq \quad (\mathsf{T} \in \mathbf{d} \rhd [\![M]\!]^n\eta) \mathbin{+\!\!+} (\mathsf{F} \in \mathbf{d} \rhd [\![N]\!]^n\eta))]$$

for all $\mathbf{d} \in D^*$. We use Lemma 4.9 to prove that $\mathbf{if}(M\theta, N\theta) \in \Lambda(\mathrm{fun}(F))$. Let $\mathsf{T} \in \mathbf{V}$. Then $\mathbf{W} \sqsubseteq [\![M]\!]^n\eta$ for some $\mathbf{W} \subseteq F(\mathbf{V})$. By induction hypothesis and Lemma 4.4 we have $M\theta \in F(\mathbf{V})$. The case $\mathsf{F} \in \mathbf{V}$ is similar.

Case f: By assumption, $\mathbf{U} \sqsubseteq [\![f]\!]^{n+1}$. Therefore, $\mathbf{U} = [\mathrm{fun}^k(F)]$, where $k = \mathrm{arity}(f)$, and

$$(*) \quad F(\vec{\mathbf{d}}) \quad \sqsubseteq \quad \mathrm{concat}[\,[\![M]\!]\eta \mid (\vec{P} \mapsto M) \leftarrow \mathcal{R}_f,\ \eta \leftarrow \mathrm{match}_{\vec{P}}(\vec{\mathbf{d}})\,])]$$

for all $\vec{\mathbf{d}} \in (D^*)^k$. We use Lemma 4.7 to show $f \in \Lambda(\mathrm{fun}^k(F))$. Let $\vec{\mathbf{U}} \in \mathrm{dom}(F)$ and $\vec{N} \in \Lambda(\vec{\mathbf{U}})$. We show $M\vec{N} \in \Lambda(F(\vec{\mathbf{U}}))$ by a side induction on $\mathsf{SN}(\vec{N})$. Since $f\vec{N}$ is simple, it suffices to show that $K \in \Lambda(F(\vec{\mathbf{U}}))$ for all terms K such that $f\vec{N} \to K$. If K is obtained by reducing one of the N_i, then we can apply the side induction hypothesis. If however $f\vec{N} \to M\theta = K$ because $\vec{N} = \vec{P}\theta$ for some rule $\vec{P} \mapsto M \in \mathcal{R}_f$, then we use Lemma 4.10 to obtain some $\eta \in \mathrm{match}_{\vec{P}}(\vec{\mathbf{U}})$ such that $\theta \in \Lambda(\eta)$. By $(*)$, there exists $\mathbf{V} \subseteq F(\vec{\mathbf{U}})$ such that $\mathbf{V} \sqsubseteq [\![M]\!]^n\eta$. By the main induction hypothesis, it follows $M\theta \in \Lambda(\mathbf{V}) \subseteq \Lambda(F(\vec{\mathbf{U}}))$. □

5 Strong normalisation for total simply typed system

We now apply our normalisation proof method to simply typed systems over the base types of booleans and integers with the non-deterministic version of Gödel's T as a concrete example. The method also works for simple types over arbitrary positive inductive base types and polymorphic types [Ber05b], and has been applied in [CS06] to dependent types.

Definition 5.1 (Simply typed systems). *The set* $\mathcal{T}$ *of* simple types *is defined as follows:*

$$\mathcal{T} \ni \rho, \sigma ::= o \mid \iota \mid \rho \to \sigma$$

where o and ι are the base types of booleans and natural numbers, respectively.

A simply typed system *is given by a rewrite system and a typing relation $f : \rho$ between function constants and types.*

Given a simply typed system, the typing judgements $\Gamma \vdash M : \rho$, where $\Gamma = x_1 : \rho_1, \ldots, x_n : \rho_n$ is an unordered context, are derived by the following rules:

$$\Gamma \vdash \mathsf{T} : o \qquad \Gamma \vdash \mathsf{F} : o \qquad \Gamma \vdash 0 : \iota \qquad \frac{\Gamma \vdash M : \iota}{\Gamma \vdash (\mathrm{S}, M) : \iota}$$

$$\Gamma,\ x : \rho \vdash x : \rho \qquad \frac{\Gamma,\ x : \rho \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \rho \to \sigma} \qquad \frac{\Gamma \vdash M : \rho \to \sigma \quad \Gamma \vdash N : \rho}{\Gamma \vdash MN : \sigma}$$

$$\frac{f : \rho}{\Gamma \vdash f : \rho} \qquad \frac{\Gamma \vdash M : \rho \quad \Gamma \vdash N : \rho}{\Gamma \vdash \mathbf{if}(M, N)\ : o \to \rho}$$

Definition 5.2 (Denotational semantics of types). *For every type ρ we define a set $[\![\rho]\!] \subseteq D_+$: $[\![o]\!] = \{\mathsf{T}, \mathsf{F}\}$. $[\![\iota]\!]$ is defined as the least subset of D_+ that contains 0 and with $d_1, \ldots, d_n$ also $([\mathrm{S}], [d_1, \ldots, d_n])$. For function types we set $[\![\rho \to \sigma]\!] = \{\mathrm{fun}(f) \mid f([\![\rho]\!]^*) \subseteq [\![\sigma]\!]^*\}$.*

Lemma 5.3. *For any $\mathbf{d} \in D_+^*$, $\mathbf{d} \in [\![\rho_1 \to \ldots \to \rho_k \to \sigma]\!]^*$ iff alle elements of $\mathbf{d}$ are of the form $\mathrm{fun}^k(f)$ and $\mathbf{d} \bullet \mathbf{e}_1 \bullet \ldots \bullet \mathbf{e}_k \in [\![\sigma]\!]^*$ for all $\mathbf{e}_i \in [\![\rho_i]\!]^*$.*

Definition 5.4 (Total simply typed systems). *A simply typed system is* total *if $[\![f]\!] \in [\![\rho]\!]^*$ for every typing $f : \rho$.*

Definition 5.5 (Semantic typing). *We define a semantic analogue to the proof-theoretic typing judgements. For a context $\Gamma = x_1 : \rho_1, \ldots, x_n : \rho_n$ and an environment η, let $\eta \in [\![\Gamma]\!]^*$ mean that $\{x_1, \ldots, x_n\} \subseteq \mathrm{dom}(\eta)$ and $\eta(x_i) \in [\![\rho_i]\!]^*$ for all i. We define*

$$\Gamma \models M : \rho \ \equiv\ \forall \eta \in [\![\Gamma]\!]^*\ (\ [\![M]\!]\eta \in [\![\rho]\!]^*\ )$$

Theorem 5.6 (Soundness of total simply typed systems). *For every total simply typed system it holds that $\Gamma \vdash M : \rho$ implies $\Gamma \models M : \rho$.*

Proof. Straightforward induction on the derivation of $\Gamma \vdash M : \rho$. □

Theorem 5.7. *Every total simply typed system is strongly normalising.*

Proof. By Theorem 5.6 and Theorem 3.4. □

Definition 5.8 (System T and its non-deterministic extension NT). Gödel's system T *is the simply typed system with the constant R and the rewrite rules given above as well as the typing $\mathrm{R} : \rho \to (\iota \to \rho \to \rho) \to \iota \to \rho$ for every type $\rho \in \mathcal{T}$. We also allow inessential extensions by constants like $<$ and the rewrite rules given in the introduction and the typing $<: \iota \to \iota \to o$.*

The system **NT** *is obtained by adding the constant $\|$ with the rewrite rules given in the introduction and the typing $\| : \rho \to \rho \to \rho$ for every type $\rho \in \mathcal{T}$.*

Theorem 5.9. *System* **NT** *is total and therefore strongly normalising.*

Proof. By Theorem 5.7 it suffices to show that all constants are total. Looking at the equations at the end of Sect. 3 and by Lemma 5.3 it is obvious that $[\![\parallel]\!] \in [\![\rho \to \rho \to \rho]\!]^*$. In order to see that $[\![\mathrm{R}]\!] \in [\![\rho \to (\iota \to \rho \to \rho) \to \iota \to \rho]\!]^*$, it suffices to show that $[\![\mathrm{R}]\!] \bullet \mathbf{d}_1 \bullet \mathbf{d}_2 \bullet \mathbf{d}_3 \in [\![\rho]\!]^*$ for all $(\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3) \in [\![\rho]\!]^* \times [\![\iota \to \rho \to \rho]\!]^* \times [\![\iota]\!]^*$. This can be proven easily by induction on the number of occurrences of the constructor S in $\mathbf{d}_3 \in [\![\iota]\!]^*$. □

Remarks. 1. The addition of the nondeterministic choice operator to a strongly normalising system does in general *not* preserve strong normalisation. For example, consider the constant $f : \iota \to \iota \to \iota \to \iota$ with the rule

$$f\,0\,1\,x \to f\,x\,x\,x$$

It is easy to see that adding f to Gödel's system T yields a strongly normalising system (using confluence and strong normalisation of system T). However, further adding the choice operator yields Toyama's well-known counterexample [Toy87]

$$f\,0\,1\,(0 \parallel 1) \to f\,(0 \parallel 1)\,(0 \parallel 1)\,(0 \parallel 1) \to^2 f\,0\,1\,(0 \parallel 1)$$

2. Kristiansen [Kri06] suggests to use fragments of the nondeterministic extension of Gödel's T above to characterise the nondeterministic polynomial hierarchy, respectively to define a higher type analogue of it.

3. For simply typed systems with non-overlapping rules (i.e. mutually non-unifiable left-hand sides) the base type ι can be interpreted as the least set containing 0 and [], and with d also $([\mathrm{S}], [d])$. This results in a semantics of types very similar to the one in [Ber05b] and [CS06].

Let us now put our method to a more serious test.

Definition 5.10 (Non-deterministic barrecursion NT). *Let* **NBR** *denote the following extension of system* **NT** *by (a suitable variant of) Spector's bar-recursion in simple types [Spe62, Tai71]. We add to system* **NT** *a constant* Φ *with the rewrite rule*

$$\begin{aligned}\Phi\ G\ H\ Y\ \alpha\ n \quad &\to \\ &\mathbf{if}(G\ \alpha\ n, H\ \alpha\ n(\lambda x.\Phi\ G\ H\ Y\ \alpha^x_n)\ (\mathrm{S}, n)))\,(Y\ \alpha < n)\end{aligned}$$

where α^x_n *is shorthand for* $\lambda k.\mathbf{if}(\alpha\ k, x)\,(k < n)$*, and the typing*

$$\Phi \quad : \quad \tau_G \to \tau_H \to \tau_Y \to (\iota \to \rho) \to \iota \to \sigma$$

where $\tau_G = (\iota \to \rho) \to \iota \to \sigma$*,* $\tau_H = (\iota \to \rho) \to \iota \to (\rho \to \sigma) \to \sigma$*,* $\tau_Y = (\iota \to \rho) \to o$*, and* ρ, σ *are arbitrary types.*

Theorem 5.11. *System* **NBR** *is strongly normalising.*

Proof. By Theorems 5.7 and 5.9, it suffices to prove that $[\![\Phi]\!]$ is total, i.e. in $[\![\tau_G \to \tau_H \to \tau_Y \to (\iota \to \rho) \to \iota \to \sigma]\!]^*$.

Every $d \in [\![\iota]\!]$ represents a finite set $\mathbb{N}(d) \subseteq \mathbb{N}$: $\mathbb{N}(0) = \{0\}$, $\mathbb{N}(\mathrm{pair}([\mathrm{S}], \mathbf{n})) = \{i+1 \mid i \in \mathbb{N}(\mathbf{n})\}$, where $\mathbb{N}(\mathbf{n}) = \bigcup\{\mathbb{N}(d) \mid d \in \mathbb{N}(\mathbf{n})\}$. We use this to lift any binary relation $\Box$ on $\mathbb{N}$ to a binary relation $\overset{\exists}{\Box}$ on $[\![\iota]\!]^*$

$$\mathbf{m} \overset{\exists}{\Box} \mathbf{n} \;\Leftrightarrow\; \exists i \in \mathbb{N}(\mathbf{m})\, \exists j \in \mathbb{N}(\mathbf{n})\, (i \Box j)$$

We view $\overset{\exists}{\Box}$ as a binary boolean function on $[\![\iota]\!]^*$. Furthermore, we define $\mathbf{n} + i \in [\![\iota]\!]^*$ ($\mathbf{n} \in [\![\iota]\!]^*$, $i \in \mathbb{N}$), by $\mathbf{n} + 0 = \mathbf{n}$ and $\mathbf{n} + (i+1) = [\mathrm{pair}([\mathrm{S}], \mathbf{n} + i)]$. It is easy to see that

$$[\![<]\!] \bullet \mathbf{m} \bullet \mathbf{n} \;=\; (\mathbf{m} \overset{\exists}{<} \mathbf{n}) \rhd [\mathsf{T}] \mathbin{+\!\!+} (\mathbf{m} \overset{\exists}{\geq} \mathbf{n}) \rhd [\mathsf{F}]$$

and, fixing $\mathbf{G} \in [\![\tau_G]\!]^*$, $\mathbf{H} \in [\![\tau_H]\!]^*$, $\mathbf{Y} \in [\![\tau_Y]\!]^*$, we have for $\mathbf{\Phi} = [\![\Phi]\!] \bullet \mathbf{G} \bullet \mathbf{H} \bullet \mathbf{Y}$

$$\begin{array}{rcl} \mathbf{\Phi} \bullet \alpha \bullet \mathbf{n} & = & (\mathsf{T} \in \mathbf{b}) \rhd \mathbf{G} \bullet \alpha \bullet \mathbf{n} \mathbin{+\!\!+} \\ & & (\mathsf{F} \in \mathbf{b}) \rhd \mathbf{H} \bullet \alpha \bullet \mathbf{n} \bullet [\mathrm{fun}(\lambda \mathbf{d}.\mathbf{\Phi} \bullet \mathrm{ext}(\alpha, \mathbf{d}, \mathbf{n}) \bullet (\mathbf{n}+1)] \end{array}$$

where $\mathbf{b} = [\![<]\!] \bullet (\mathbf{Y} \bullet \alpha) \bullet \mathbf{n}$ (see the equations at the end of Sect. 3) and

$$\mathrm{ext}(\alpha, \mathbf{d}, \mathbf{n}) = [\mathrm{fun}(\lambda \mathbf{m}.((\mathbf{m} \overset{\exists}{<} \mathbf{n}) \rhd \alpha \bullet \mathbf{m}) \mathbin{+\!\!+} ((\mathbf{m} \overset{\exists}{\geq} \mathbf{n}) \rhd \mathbf{d}))].$$

We define a binary relation $\gg$ on $[\![\iota \to \rho]\!]^* \times [\![\mathsf{nat}]\!]^*$:

$$(\alpha, \mathbf{n}) \gg (\beta, \mathbf{m}) \Leftrightarrow \mathbf{Y} \bullet \alpha \overset{\exists}{\geq} \mathbf{n} \wedge \mathbf{m} = \mathbf{n} + 1 \wedge \exists \mathbf{x} \in [\![\rho]\!]^*\, (\beta = \mathrm{ext}(\alpha, \mathbf{x}, \mathbf{n}))$$

We show that $\gg$ is wellfounded. Assume we had an infinite decreasing sequence

$$(\alpha_0, \mathbf{n}_0) \gg (\alpha_1, \mathbf{n}_0 + 1) \gg \ldots \gg (\alpha_i, \mathbf{n}_0 + i) \gg \ldots$$

It suffices to find $k \in \mathbb{N}$ such that $\mathbf{Y} \bullet \alpha_k \overset{\exists}{\geq} \mathbf{n}_0 + k$ does *not* hold. Define $\alpha_\infty = [\mathrm{fun}(f)]$ where $f(\mathbf{m}) = \alpha_{\sup(\mathbb{N}(\mathbf{m}))+1} \bullet \mathbf{m}$ for $\mathbf{m} \in [\![\iota]\!]^*$, and $f(\mathbf{d}) = \bot$ for $\mathbf{d} \in D^* \setminus [\![\iota]\!]^*$. f is continuous because $[\![\iota]\!]^*$ is an open subset of D^*. By continuity of $\bullet$ there exists a number $k > \sup \mathbb{N}(\mathbf{Y} \bullet \alpha_\infty)$ such that

$$\forall \beta \in [\![\iota \to \rho]\!]^* (\alpha_\infty =_k \beta \Rightarrow \mathbf{Y} \bullet \alpha_\infty = \mathbf{Y} \bullet \beta)$$

where $\alpha =_k \beta :\Leftrightarrow \forall \mathbf{m} \in [\![\iota]\!]^*\, (\sup \mathbb{N}(\mathbf{m}) < k \Rightarrow \alpha \bullet \mathbf{m} = \beta \bullet \mathbf{m})$. It suffices to show that $\alpha_\infty =_k \alpha_k$, since then $\mathbf{Y} \bullet \alpha_k = \mathbf{Y} \bullet \alpha_\infty$, which is *not* $\overset{\exists}{\geq} \mathbf{n}_0 + k$, because $k > \sup \mathbb{N}(\mathbf{Y} \bullet \alpha_\infty)$.

Now let us prove that $\alpha_\infty =_k \alpha_k$. Let $\mathbf{m} \in [\![\iota]\!]^*$ such that $l := \sup \mathbb{N}(\mathbf{m}) < k$. If $\mathbf{m} = []$, then $l = 0$ and therefore $\alpha_\infty \bullet \mathbf{m} = \alpha_1 \bullet \mathbf{m} = [] = \alpha_k \bullet \mathbf{m}$. Hence assume $\mathbf{m} \neq []$. First, note that if $i > l$, then $\mathbf{m} \overset{\exists}{<} \mathbf{n}_0 + i$ holds (because $\mathbf{m}$ and $\mathbf{n}_0$ are both nonempty), while $\mathbf{m} \overset{\exists}{\geq} \mathbf{n}_0 + i$ does *not* hold. Consequently $\alpha_{i+1} \bullet \mathbf{m} = \alpha_i \bullet \mathbf{m}$. It follows, more generally, that $\alpha_j \bullet \mathbf{m} = \alpha_i \bullet \mathbf{m}$, for all

$i, j > l$. In particular, $\alpha_\infty \bullet \mathbf{m} = \alpha_{l+1} \bullet \mathbf{m} = \alpha_k \bullet \mathbf{m}$. This completes the proof of the wellfoundedness of $\gg$.

It is now straightforward to prove that $\mathbf{\Phi}\bullet\alpha\bullet\mathbf{n} \in [\![\sigma]\!]^*$ for all $\alpha \in [\![\iota \to \rho]\!]^*$ and $\mathbf{n} \in [\![\iota]\!]^*$, by $\gg$-induction on $(\alpha, \mathbf{n})$: Clearly, it is enough to show that if $\mathsf{F} \in \mathbf{b} = [\![<]\!]\bullet(\mathbf{Y}\bullet\alpha)\bullet\mathbf{n}$, then for every $\mathbf{x} \in [\![\rho]\!]^*$ we have $\mathbf{\Phi}\bullet(\mathrm{ext}(\alpha, \mathbf{x}, \mathbf{n})\bullet(\mathbf{n}{+}1) \in [\![\sigma]\!]^*$. But, if $\mathsf{F} \in \mathbf{b}$, then $\mathbf{Y}\bullet\alpha \overset{\exists}{\geq} \mathbf{n}$ and therefore $(\alpha, \mathbf{n}) \gg (\mathrm{ext}(\alpha, \mathbf{x}, \mathbf{n}), \mathbf{n}{+}1)$. Hence, the induction hypothesis applies. □

6 Completeness

It is natural to ask whether our normalisation proof method is complete, i.e. whether the converse of Theorem 3.4 holds. The following trivial counterexample show that this is not the case for arbitrary terms: Consider the constant Y (fixed point operator) with the rewrite rule $Y\,x \to x\,(Y\,x)$. Because of the strictness of our model we have $[\![Y]\!] = \bot$, even though the term Y is in normal form and hence strongly normalising. The reason for this phenomena is that the rewrite rule for Y requires one argument which is not present in the term Y. We now show that if we exclude this situation, the converse of Theorem 3.4 holds.

Definition 6.1 (Full terms). *A term M is called* full *if M and all rewrite rules concerning M contain constants f only 'fully applied", i.e. in contexts $f\,M_1 \ldots M_k$ where $k \geq \mathrm{arity}(f)$.*

Being full is a mild condition, since every term can be made full by η-expansion. Clearly, the class of full terms is closed under reduction.

Theorem 6.2 (Completeness). *If M is full, then $[\![M]\!] \neq \bot$ if and only if M is strongly normalising.*

Proof. Because of Theorem 3.4, only the "if" direction needs to be shown. Let η_0 be the environment defined by $\eta_0(x) = []$ for all variables x.

Claim. If M is full and strongly normalising, then

1. $[\![M]\!]\eta_0 \neq \bot$.
2. For every linear pattern P and every $\eta \in \mathrm{match}_P([\![M]\!]\eta_0)$ there exists a substitution θ with $\mathrm{dom}(\theta) = \mathsf{FV}(P)$ such that $M \to^* P\theta$ and $\eta = [\![\theta]\!]\eta_0$, i.e. $\eta(x) = [\![\theta]\!]\eta_0(x)$.

We prove the claim by main induction on $\mathsf{SN}(M)$ and side induction on (the structure of) M. Note that if $[\![M]\!]\eta_0 \neq \bot$, then $\mathrm{match}_P([\![M]\!]\eta_0) \neq \bot$ (in fact, $\mathrm{match}_P(\mathbf{d}) \neq \bot$ for *all* $\mathbf{d} \in D_+^*$). Furthermore, if P is a variable x, then 2. holds trivially, since we can set $\theta = [x \mapsto M]$. Hence, in the proof of 2. it suffices to consider patterns P which are constructors or pairs.

We do a case analysis on the following possible forms of a full term:

$$x\,\vec{K}, \quad c, \quad c\,M\,\vec{K}, \quad (M_1, M_2), \quad (M_1, M_2)\,L\,\vec{K},$$

$$\lambda x.M, \quad (\lambda x.M)\, N\, \vec{K}, \quad \mathbf{if}(M,N)\,, \quad \mathbf{if}(M,N)\, L\, \vec{K}, \quad f\, \vec{N}\, \vec{K}$$

where $\vec{K}$ is a (possibly empty) list of terms, $\vec{N} = N_1, \ldots, N_{\mathrm{arity}(f)}$, and the terms M, N, L, $\vec{K}$, and $\vec{N}$ are all full. In the following, $[\![M_1, \ldots, M_k]\!]\eta_0 \neq \bot$ is shorthand for $[\![M_i]\!]\eta_0$ for all i.

Case $x\,\vec{K}$: By side induction hypothesis, $[\![\vec{K}]\!]\eta_0 \neq \bot$. Hence $[\![x\,\vec{K}]\!]\eta_0 = [] \bullet [\![\vec{K}]\!]\eta_0 = [] \neq \bot$, proving the first part. Since $\mathrm{match}_P([]) = []$, the second part holds trivially.

Case c: $[\![c]\!]\eta_0 = c \neq \bot$. If $\eta \in \mathrm{match}_P(c)$, then $P = c$ and we can choose $\theta = \emptyset$.

Case $c\,M\,\vec{K}$: By side induction hypothesis, $[\![M, \vec{K}]\!]\eta_0 \neq \bot$. Hence $[\![c\,M\,\vec{K}]\!]\eta_0 = [c] \bullet [\![M, \vec{K}]\!]\eta_0 = [] \neq \bot$ and the second part holds trivially.

Case (M_1, M_2): By side induction hypothesis, $\mathbf{d}_i = [\![M_i]\!]\eta_0 \neq \bot$. Hence $[\![(M_1, M_2)]\!]\eta_0 = [\mathrm{pair}(\mathbf{d}_1, \mathbf{d}_2)] \neq \bot$. Let $\eta \in \mathrm{match}_P[\mathrm{pair}(\mathbf{d}_1, \mathbf{d}_2)]$. Then $P = (P_1, P_2)$ and $\eta = \eta_1 \cup \eta_2$ with $\eta_i \in \mathrm{match}_{P_i}(\mathbf{d}_i)$. By side induction hypothesis, $M_i \to^* \theta_i$ and $\eta_i = [\![\theta_i]\!]\eta_0$. Then, with $\theta = \theta_1 \cup \theta_2$, $(M_1, M_2) \to^* (P_1, P_2)\theta$ and $\eta = [\![\theta]\!]\eta_0$.

Case $(M_1, M_2)\,M\,\vec{K}$: Similar to the case $c\,M\,\vec{K}$.

Case $\lambda x.M$: By side induction hypothesis, $[\![M]\!]\eta_0 \neq \bot$. Since $\eta_0 = \eta_0[x \mapsto []]$ it follows

$$[\![\lambda x.M]\!]\eta_0 = [\mathrm{fun}(\lambda \mathbf{d} \in D^*.[\![M]\!]\eta[x := \mathbf{d}])] \neq \bot.$$

Since $\mathrm{match}_P[\mathrm{fun}(\ldots)] = []$, if P is a constructor or a pair, the second part is trivial.

Case $(\lambda x.M)\,N\,\vec{K}$: We first remark that in general, if $[\![N]\!]\eta \neq \bot$, then $[\![(\lambda x.M)\,N]\!]\eta = [\![M[N/x]]\!]$ (one first shows $[\![M[N/x]]\!]\eta = [\![M]\!]\eta[x \mapsto [\![N]\!]\eta]$, by induction on M, from which the assertion easily follows). By side induction hypothesis, we have $[\![M, N, \vec{K}]\!]\eta_0 \neq \bot$. Hence, by the above remark, $[\![(\lambda x.M)\,N\,\vec{K}]\!]\eta_0 = [\![M[N/x]\,\vec{K}]\!]\eta_0$. Since $(\lambda x.M)\,N\,\vec{K} \to M[N/x]\,\vec{K}$, the main induction hypothesis applies and we are done.

Case $\mathbf{if}(M,N)$: Since (for arbitrary M, N and η)

$$[\![\mathbf{if}(M,N)\,]\!]\eta = [\mathrm{fun}(\lambda \mathbf{d} \in D^*.(\mathsf{T} \in \mathbf{d} \rhd [\![M]\!]\eta) \mathbin{+\!\!+} (\mathsf{F} \in \mathbf{d} \rhd [\![N]\!]\eta))]$$

and

$$(\mathsf{T} \in [] \rhd [\![M]\!]\eta) \mathbin{+\!\!+} (\mathsf{F} \in [] \rhd [\![N]\!]\eta) = []$$

we have $[\![\mathbf{if}(M,N)\,]\!]\eta \neq \bot$ and $\mathrm{match}_P([\![\mathbf{if}(M,N)\,]\!]\eta) = []$, if P is a constructor or a pair. Hence, we are done.

Case $\mathbf{if}(M,N)\,L\,\vec{K}$: By side induction hypothesis, $\mathbf{d} = [\![L]\!]\eta_0 \neq \bot$. Clearly,

$$[\![\mathbf{if}(M,N)\,L\,\vec{K}]\!]\eta_0 = (\mathsf{T} \in \mathbf{d} \rhd [\![M\,\vec{K}]\!]\eta_0) \mathbin{+\!\!+} (\mathsf{F} \in \mathbf{d} \rhd [\![N\,\vec{K}]\!]\eta_0).$$

Therefore, for part 1, it suffices to show that both arguments of the $+\!\!+$ expression are $\neq \bot$. If $\mathsf{T} \notin \mathbf{d}$, then $\mathsf{T} \in \mathbf{d} \rhd [\![M\,\vec{K}]\!]\eta_0 = []$. Otherwise, $\mathsf{T} \in \mathbf{d} \rhd [\![M\,\vec{K}]\!]\eta_0 = [\![M\,\vec{K}]\!]\eta_0$. Furthermore, $\emptyset \in \mathrm{match}_{\mathsf{T}}(\mathbf{d})$ and therefore, by

side induction hypothesis, $L \to^* \mathsf{T}$. It follows $\mathbf{if}(M,N)\,L\,\vec{K} \to^* M\,\vec{K}$. Therefore, by main induction hypothesis, $[\![M\,\vec{K}]\!]\eta_0 \neq \bot$. For $\mathsf{T} \in \mathbf{d} \rhd [\![M\,\vec{K}]\!]\eta_0$ the argument is similar. To prove 2., let $\eta \in \mathrm{match}_P([\![\mathbf{if}(M,N)\,L\,\vec{K}]\!]\eta_0)$. Since P is a constructor or a pair we have, w.l.o.g., $\eta \in \mathrm{match}_P(\mathsf{T} \in \mathbf{d} \rhd [\![M\,\vec{K}]\!]\eta_0)$, hence, necessarily, $\mathsf{T} \in \mathbf{d}$ and $\eta \in \mathrm{match}_P([\![M\,\vec{K}]\!]\eta_0)$. With the same argument as before, it follows $\mathbf{if}(M,N)\,L\,\vec{K} \to^* M\,\vec{K}$, and therefore the main induction hypothesis applies.

Case $f\,\vec{N}\,\vec{K}$: Set $\vec{\mathbf{d}} = [\![\vec{N}]\!]\eta_0$, $\vec{\mathbf{e}} = [\![\vec{K}]\!]\eta_0$, which, by side induction hypothesis, are all $\neq \bot$. We have

$$[\![f\,\vec{N}\,\vec{K}]\!]\eta_0 = [\![f]\!] \bullet \vec{\mathbf{d}} \bullet \vec{\mathbf{e}} = \mathrm{concat}[[\![M]\!]\eta \bullet \vec{\mathbf{e}} \mid (\vec{P} \mapsto M) \leftarrow \mathcal{R}_f,\ \eta \leftarrow \mathrm{match}_{\vec{P}}(\vec{\mathbf{d}})].$$

Hence, for part 1., it suffices to show that $[\![M]\!]\eta \bullet \vec{\mathbf{e}} \neq \bot$ for all $\vec{P} \mapsto M \in \mathcal{R}_f$ and $\eta \in \mathrm{match}_{\vec{P}}(\vec{\mathbf{d}})$. By side induction hypothesis, there is a substitution θ such that $\vec{N} \to^* \vec{P}\theta$ and $\eta = [\![\theta]\!]\eta_0$. Hence

$$[\![M]\!]\eta \bullet \vec{\mathbf{e}} = [\![M]\!][\![\theta]\!]\eta_0 \bullet [\![\vec{K}]\!]\eta_0 = [\![M\theta\,\vec{K}]\!]\eta_0.$$

Since $f\,\vec{N}\,\vec{K} \to M\theta\,\vec{K}$, we know, by main induction hypothesis, $[\![M\theta\,\vec{K}]\!]\eta_0 \neq \bot$. To prove part 2., Let $\eta \in \mathrm{match}_P([\![f\,\vec{N}\,\vec{K}]\!]\eta_0)$. By the above, and since we may assume that P is a constructor or a pair, it follows that $\eta \in [\![M\theta\,\vec{K}]\!]\eta_0$ for some rule $\vec{P} \mapsto M \in \mathcal{R}_f$ and substitution θ with $\vec{N} \to^* \vec{P}\theta$. Since $f\,\vec{N}\,\vec{K} \to M\theta\,\vec{K}$, it follows, by main induction hypothesis, that there is a substitution θ' with $M\theta\,\vec{K} \to^* P\theta'$ and $\eta = [\![\theta']\!]\eta_0$. □

7 Conclusion

We defined a strict domain-theoretic model for extensions of the type free λ-calculus with constructors, constants and rewrite rules. We showed that a term is strongly normalising if it does not denote $\bot$ in this model. For full terms we also proved the reverse implication. The only restrictions on the rewrite rules were left linearity and finite branching, while disjointness or confluence conditions could be dropped, thanks to the non-deterministic nature of the model. Our construction, although presented in an abstract order-theoretic setting, could be easily recast in the more elementary style of [CS06]. However, the definition of the reducibility candidates is impredicative in both settings. We leave it as an open problem whether Theorem 3.4 can be prove predicatively, possibly along the lines of [Val01].

When applying our method to a typed system one usually proves totality of typeable terms, i.e. the fact that every term has a value in the denotation of its type. Totality of terms is usually easy to establish because it is a compositional and modular property and therefore it suffices to proven it for each constant separately. However this also means that unlike definedness, totality cannot be equivalent to strong normalisation, since the latter is neither compositional (if two terms are strongly normalisable their application need not be)

nor modular (the combination of two strongly normalising rewrite system need not be strongly normalising, as Toyama's counterexample shows). It is an interesting question whether using our method the modularity result for complete, i.e. strongly normalising and confluent rewrite systems by Toyama, Klop and Barendregt [TKB95] can be extended to our setting.

References

[BCDC83] H. Barendregt, , M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *The Journal of Symbolic Logic*, 48(4):931–940, 1983.

[Ber05a] U. Berger. Continuous semantics for strong normalization. In S.B. Cooper, B. Löwe, and L. Torenvliet, editors, *CiE 2005: New Computational Paradigms*, volume 3526 of *LNCS*, pages 23–34. Springer, 2005.

[Ber05b] U. Berger. Strong normalization for applied lambda calculi. *Logical Methods in Computer Science*, 1(2):1–14, 2005.

[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, 1999.

[Bla05] F. Blanqui. Definitions by rewriting in the calculus of constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005.

[BT88] Val Breazu-Tannen. Combining algebra and higher-order types. In Yuri Gurevich, editor, *Proceedings of the Third Annual IEEE Symp. on Logic in Computer Science, LICS 1988*, pages 82–90. IEEE Computer Society Press, July 1988.

[CS06] T. Coquand and A. Spiwack. A proof of strong normalisation using domain theory. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 307–316. IEEE Computer Society Press, 2006.

[Dou92] D.J. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. *Information and Computation*, 101:251–267, 1992. Daniel J. Dougherty.

[GHK+03] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott. *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2003.

[Jay04] C.B. Jay. The pattern calculus. *ACM Trans. Program. Lang. Syst*, 26(6):911–937, 2004.

[Kri06] L. Kristiansen. Complexity-theoretic hierarchies. In A. Beckmann, U. Berger, B. Löwe, and J.V. Tucker, editors, *CiE 2006: Logical Approaches to Computational Barriers*, volume 3988 of *LNCS*, pages 279–288. Springer, 2006.

[Mil91] D. Miller. A logic programming language with lambda–abstraction, function variables and simple unification. *Journal of Logic and Computation*, 2(4):497–536, 1991.

[Nip91] T. Nipkow. Higher-order critical pairs. In R. Vemuri, editor, *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 342–349, Los Alamitos, 1991. IEEE Computer Society Press.

[Plo77] G.D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci*, 5:223–255, 1977.

[Pot80] G. Pottinger. A type assignment for the strongly normalisable terms. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, 1980.

[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, 1971.

[TKB95] Y. Toyama, J.W. Klop, and H.P. Barendregt. Termination for direct sums of left-linear complete term rewriting systems. *Journal of the Association for Computing Machinery*, 42(6):1275–1304, 1995.

[Toy87] Y. Toyama. Counterexample to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.

[Val01] S. Valentini. An elementary proof of strong normalization for intersection types. *Archive for Mathematical Logic*, 40:475–488, 2001.

[vB92] S. van Bakel. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, 102:135–163, 1992.