A domain model characterising strong normalisation

Ulrich Berger

Department of Computer Science, University of Wales Swansea, Singleton Park, Swansea SA2 8PP, United Kingdom

Abstract

Building on previous work by Coquand and Spiwack [8] we construct a strict domain-theoretic model for the untyped λ-calculus with pattern matching and term rewriting which has 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. As an application, we prove strong normalisation for barrecursion in higher types combined with polymorphism and non-deterministic choice.

Key words: λ-calculus, term rewriting, normalisation, domain theory

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 conveniently modelled and analysed by suitable extended λ-calculi. In this paper we study the untyped λ-calculus with pattern matching and term rewriting which captures a large variety of such calculi. We construct a domain-theoretic model in which terms not denoting $\bot$ are strongly normalising. For the large class of so-called safe terms we show that the converse holds as well. Our results are based on a domain-theoretic method for proving strong normalisation introduced in [3] and [4] and later improved in [8]. The method has its roots in three strands of work:

Email address: u.berger@swansea.ac.uk (Ulrich Berger).
URL: http://www.cs.swan.ac.uk/∼csulrich/ (Ulrich Berger).

(1) The adequacy theorem for PCF [18]: if a closed PCF-term of base type denotes a numeral in the domain model, then it weakly head reduces to that numeral.
(2) The characterisation of strongly normalising (pure) λ-terms by intersection types [19].
(3) The observation that intersection types can be used to construct a domain-theoretic semantics (filter model) of λ-terms [2,25].

In [3], 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 normalisation holds for an underlying typed λ-calculus without recursive rewrite rules, an assumption which Coquand and Spiwack [8] 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 respectively confluence condition on rewrite systems that was imposed in [3] and [8]. 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 remaining restrictions are 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, i.e. the equivalence of definedness and strong normalisation, under a mild but necessary 'safety' condition (which is guaranteed, for example, by typeability).

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 within an abstract order-theoretic setting, by solving a recursive domain equation involving standard operations on domains. We believe that this is a very economic and transparent way of presenting the construction. If one wishes to make the constructive aspects of this model more explicit one could easily redefine it in the style of [8] 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 [3] and [8] 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 de-

fined and therefore strongly normalising. In the deterministic model [4,8] 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 elements", but still, as our case studies with non-deterministic and polymorphic 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 list of constructor patterns (built 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 [7,10] 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 [17,15,5,6,26]. A strictly more general approach is Jay's pattern calculus [12], but little seems to be known about strong normalisation in this calculus.

2 The λ-calculus with pattern matching and term rewriting

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 $(\ldots(MN_1)\ldots)N_k$.

The idea behind constructors and pairs is that they construct data (as in Lisp like programming languages). For example, the natural numbers $0, 1, 2, \ldots$ can be represented by the terms 0, $(\mathrm{S}, 0)$, $(\mathrm{S}, (\mathrm{S}, 0))$, $\ldots$, where 0 and S are constructors. The syntax of terms allows constructors or pairs to occur on the left hand side of an application, but, semantically, terms of the form $c\,N$ or $(M_1, M_2)\,N$ are meaningless and are interpreted by $\bot$ in our model. Consequently, such terms are excluded from our completeness results in Sect. 6. Pairs could be eliminated by replacing terms of the form (M, N) by $\mathrm{cons}\,M\,N$,

where cons is a constructor, and modifying the domain model accordingly. We include pairs since they have some technical advantages and make the distinction between data and the algorithmic part of the calculus clearer. Also, **if**-terms are included just for convenience (they 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 $\mathcal{R}$ *assigns 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 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 for a constant $<$

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

(standing for $\mathcal{R}_< = [[m, 0] \mapsto \mathsf{F}, [0, (\mathrm{S}, m)] \mapsto \mathsf{T}, [(\mathrm{S}, m), (\mathrm{S}, n)] \mapsto m < n]$). This is an instance of *primitive recursion* which is captured in general by the

primitive recursion operator R with the rules

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

The rules for $<$ and R are deterministic in the sense that different rules have non-unifiable left hand sides and hence cannot be used to contract the same term. Hence they can be viewed immediately as recursive definitions of corresponding functions. It is less clear what kind of function should be defined by the non-deterministic rules for the *choice operator*

$$\begin{aligned} x \parallel y &\to x \\ x \parallel y &\to y \end{aligned}$$

The model defined in the next section will interpret $\parallel$ as the concatenation operation on non-deterministic values represented by lists.

3 Domain semantics for strong normalisation

By a *domain* we mean 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 $D_+ := D \backslash \{\bot\}$ (the set of "defined" elements) and $D_{0+} := D_0 \backslash \{\bot\}$. For any set A 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. The category of domains and continuous functions is cartesian closed with product $(D \times E)$ and exponential (continuous function space $D \to E$) defined as expected. This justifies the informal use of λ-abstraction to define continuous functions on domains. Furthermore, every continuous function $f\colon D \to D$ has a least fixed point (depending continuously on f), which permits arbitrary (continuous) recursive definitions of continuous functions. For a continuous function $f\colon D \to E$ we set $\mathrm{dom}(f) := \{d \in D \mid f(d) \neq \bot_E\}$ and call f *strict* if $f(\bot_D) = \bot_E$, i.e. $\mathrm{dom}(f) \subseteq D_+$. In the following we also need the strict versions of product and exponential, *smash product* and *strict function space*, as well as *strict lists* and the *coalesced sum* [11]:

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

Pairs, lists and strict functions are ordered component-wise respectively 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. By ϵ we denote the empty list. 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 $f\colon D \xrightarrow{!} E$ 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 $\bot$.

$$
\begin{array}{lll}
(++) & : & D^* \xrightarrow{!} D^* \xrightarrow{!} D^* \\
\mathrm{concat} & : & (D^*)^* \xrightarrow{!} D^* \\
\mathrm{map} & : & (D \xrightarrow{!} E) \to (D^* \xrightarrow{!} E^*) \\
(\rhd) & : & \{\mathsf{T}, \mathsf{F}\}_\bot \xrightarrow{!} D^* \to D^* \\
[d_1, \ldots, d_n] ++ [e_1, \ldots, e_m] & := & [d_1, \ldots, d_n, e_1, \ldots, e_m] \\
\mathrm{concat}\ [\mathbf{d}_1, \ldots, \mathbf{d}_n] & := & \mathbf{d}_1 ++ \ \ldots \ ++ \mathbf{d}_n \\
\mathrm{map}(f)[d_1, \ldots, d_n] & := & [f(d_1), \ldots, f(d_n)] \\
\mathsf{T} \rhd \mathbf{d} & := & \mathbf{d} \\
\mathsf{F} \rhd \mathbf{d} & := & \epsilon
\end{array}
$$

Note that map and $(\rhd)$ are non-strict since $\mathrm{map}(\bot)\epsilon = \epsilon$ and $\mathsf{F} \rhd \bot = \epsilon$.

We will make frequent use of the Haskell-like *list comprehension* notation $[f(d) \mid d \leftarrow \mathbf{d}] := \mathrm{map}(f)(\mathbf{d})$ as well as of its iterated forms, for example, $[f(d,e) \mid d \leftarrow \mathbf{d}, e \leftarrow g(d)] := \mathrm{concat}[\,[f(d,e) \mid e \leftarrow g(d)] \mid d \leftarrow \mathbf{d}]$.

In the category of domains with embeddings the domain operations above are (more precisely, can be extended to) covariant continuous functors. Furthermore, since this category has directed colimits, we can solve *recursive domain equations* (see e.g. [11]).

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

$$D = \mathcal{C}_\bot \oplus (D^* \otimes D^*) \oplus (D^* \xrightarrow{!} D^*)$$

where "=" means of course "isomorphic".

In the following, the boldface letters $\mathbf{d}$, $\mathbf{e}$ range over D_+^* (i.p. $\mathbf{d}, \mathbf{e} \neq \bot$). The elements of D_+ are $\mathrm{in}_1(c)$ $(c \in \mathcal{C})$, $\mathrm{in}_2(\mathbf{d}, \mathbf{e})$, and $\mathrm{in}_3(f)$ $(f \in (D^* \xrightarrow{!} D^*)_+)$, which, for ease of readability, we write as c, $\mathrm{pair}(\mathbf{d}, \mathbf{e})$ and $\mathrm{fun}(f)$, respectively. We extend pair and fun to strict continuous functions $\mathrm{pair}\colon D^* \times D^* \xrightarrow{!} D$

and $\mathrm{fun}\colon (D^* \to D^*) \xrightarrow{!} D$ by setting $\mathrm{pair}(\bot, \mathbf{d}) = \mathrm{pair}(\mathbf{d}, \bot) := \bot$ and $\mathrm{fun}(f) := \mathrm{fun}(f')$ where f' coincides with f except that $f'(\bot) := \bot$ (of course, $\mathrm{fun}(\bot) := \bot$). We also 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 $\mathrm{split}\colon D \xrightarrow{!} (D^* \times D^*)^*$ is defined by $\mathrm{split}(\mathrm{in}_2(\mathbf{d}, \mathbf{e})) := [(\mathbf{d}, \mathbf{e})]$ and $\mathrm{split}(d) := \epsilon$ if $d \in D_+$ is not of the form $\mathrm{pair}(\mathbf{d}, \mathbf{e})$. Furthermore, we define the strict continuous functions

$$\begin{array}{lll}
\mathrm{fun}^k & : & ((D^*)^k \to D^*) \xrightarrow{!} D \quad (k \geq 1) \\
\mathrm{app} & : & D \times D^* \xrightarrow{!} D^* \\
(\bullet) & : & D^* \times D^* \xrightarrow{!} D^* \\
\mathrm{fun}^1(f) & := & \mathrm{in}_3(\mathrm{strict}_{\to}(f)) \\
\mathrm{fun}^{k+1}(f) & := & \mathrm{fun}(\lambda \mathbf{d} \in D^*.[\mathrm{fun}^k(\lambda \vec{\mathbf{e}} \in (D^*)^k.f(\mathbf{d}, \vec{\mathbf{e}}))]) \\
\mathrm{app}(\mathrm{fun}(f), \mathbf{d}) & := & f(\mathbf{d}) \\
\mathrm{app}(d, \mathbf{d}) & := & \bot, \text{ if } d \text{ is a pair or a constructor} \\
\mathbf{d} \bullet \mathbf{e} & := & [\mathrm{app}(d, \mathbf{e}) \mid d \leftarrow \mathbf{d}]
\end{array}$$

In order to define the semantics of constants we need to match linear patterns P against non-deterministic values. Let VEnv denote the pointwise ordered domain of *environments*, i.e. mappings from the set of variables to D^*. We set $\eta_\epsilon(x) := \epsilon$ for all variables x and denote by $\eta[x := \mathbf{d}]$ the modified environment mapping x to $\mathbf{d}$ and all other variables y to $\eta(y)$. We also set

$$(\eta{+\!\!+}\eta')(x) := \eta(x){+\!\!+}\eta'(x).$$

However, we will use the construction $\eta{+\!\!+}\eta'$ only in situations where $\eta(x) = \epsilon$ or $\eta'(x) = \epsilon$ for every variable x.

Definition 3.2 (Semantic matching) *We define* $\mathrm{match}_P\colon D^* \xrightarrow{!} \mathsf{VEnv}^*$ *by*

$$\begin{array}{ll}
\mathrm{match}_x(\mathbf{d}) & = [\eta_\epsilon[x := \mathbf{d}]] \\
\mathrm{match}_c(\mathbf{d}) & = (c \in \mathbf{d}) \rhd [\eta_\epsilon] \\
\mathrm{match}_{(P,Q)}(\mathbf{d}) & = [\eta{+\!\!+}\eta' \mid (\mathbf{e}, \mathbf{e}') \leftarrow \mathbf{d}, \eta \leftarrow \mathrm{match}_P(\mathbf{e}), \eta' \leftarrow \mathrm{match}_Q(\mathbf{e}')]
\end{array}$$

For a linear tuple $\vec{P} = P_1, \ldots, P_k$ *of patterns we set* $\mathrm{match}_{\vec{P}}(\mathbf{d}_1, \ldots, \mathbf{d}_k) :=$

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

Clearly, if $\eta \in \mathrm{match}_P(\mathbf{d})$ and $x \notin \mathsf{FV}(P)$, then $\eta(x) = \epsilon$.

Definition 3.3 (Strict denotational semantics of terms) *The value of a term with respect to an environment, $[\![M]\!]\eta \in D^*$, is recursively defined by*

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

where $k = \mathrm{arity}(f)$. We set $[\![M]\!] := [\![M]\!]\eta_\epsilon$.

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

We prove this theorem in the next section. As remarked earlier, for applications it is important that in our model the values of constants are easy to compute and correspond closely to the intended meaning of the constants. For example, for the constants R and $\|$ introduced in Sect. 2 we have

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

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

Remark. The model in [8] 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.

Because our model is strict (and we allow non-deterministic rewrite rules) we cannot expect it to respect reduction, i.e. $M \to M'$ does in general not imply $[\![M]\!]\eta = [\![M']\!]\eta$. However, as the following lemma shows, β-conversion is respected if the argument is defined.

Lemma 3.5 *If $[\![N]\!]\eta \neq \bot$, then $[\![(\lambda x.M)\,N]\!]\eta = [\![M[N/x]]\!]\eta$.*

Proof. One first shows $[\![M[N/x]]\!]\eta = [\![M]\!]\eta[x \mapsto [\![N]\!]\eta]$, by induction on M, from which the assertion easily follows. □

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 [8] 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, we have to extend the assignment to "fuzzy" elements.

A term is called *simple* if it has neither of the following forms: $c\vec{N}$, $(M_1, M_2)\vec{N}$, $\lambda x.M$, $\mathbf{if}(M, N)$, $fN_1 \dots N_k$ where $k < \mathrm{arity}(f)$. A *reducibility candidate* is a set X of terms such that

RC1 If $M \in X$, then M is strongly normalising.
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**.

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.*

Proof. Immediate from the definitions. □

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 $((D^*)^k \xrightarrow{!} D^*)_{0+}$ (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 elements 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 $D_0 = \{\bot\}$,

$$D_{n+1} = \mathcal{C}_\bot \oplus (D^*_n \otimes D^*_n) \oplus (D^*_n \xrightarrow{!} D^*_n)$$

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* $F(\mathbf{d}) \neq \bot$, *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 $\Lambda(\mathbf{U})$:

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

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 $M \in \Lambda(\mathbf{U})$, $N \in \Lambda(\mathbf{V})$ and $\mathbf{W} \sqsubseteq \mathbf{U} \bullet \mathbf{V}$, then $MN \in \Lambda(\mathbf{W})$.*

Proof. 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 $\Lambda(\mathbf{W})$. Since M is simple, any reduction of MN must take place in M or N. In either case, the induction hypothesis applies.

Case M is not simple. Then $M \in \Lambda(U)$ for some $U \in \mathbf{U}$. Since, by assumption, $\mathbf{U} \bullet \mathbf{V} \neq \bot$, U is of the form $\mathrm{fun}(F)$ with $\mathbf{V} \in \mathrm{dom}(F)$ and $F(\mathbf{V}) \subseteq \mathbf{U} \bullet \mathbf{V}$. By Lemma 4.2, 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 4.5, $N \in \Lambda(\mathbf{V}')$. Hence, $MN \in \Lambda(F(\mathbf{V}')) = \Lambda(F(\mathbf{V})) \subseteq \Lambda(\mathbf{U} \bullet \mathbf{V}) \subseteq \Lambda(\mathbf{W})$, by Lemmas 4.4 and 4.5. □

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 *(a) If $c \in \Lambda(\mathbf{U})$, then $c \in \mathbf{U}$.*
(b) If $(M_1, M_2) \in \Lambda(\mathbf{U})$, then $M_1 \in \Lambda(\mathbf{U}_1)$ and $M_2 \in \Lambda(\mathbf{U}_2)$ for some $\mathrm{pair}(\mathbf{U}_1, \mathbf{U}_2) \in \mathbf{U}$.
(c) $\Lambda(\mathbf{U})$ does not contain terms of the form $c\,N\,\vec{N}$ or $(M_1, M_2)\,N\,\vec{N}$.

Proof. The three items are proven simultaneously by induction on $\mathrm{rk}(\mathbf{U})$. To prove (a), assume $c \in \Lambda(\mathbf{U})$. Since c is not simple, $c \in \Lambda(U)$ for some $U \in \mathbf{U}$ which cannot be a pair and cannot be of the form $\mathrm{fun}(F)$ either, since then

we could choose $\mathbf{V} \in \mathrm{dom}(F)$ and $x \in \Lambda(\mathbf{V})$ and would have $c\,x \in \Lambda(F(\mathbf{V}))$ contradicting the induction hypothesis (c), since $\mathrm{rk}(F(\mathbf{V})) < \mathrm{rk}(U)$ and since $c\,x$ is not simple. Hence $U = c$. Part (b) is proven similarly. Finally, to prove (c), assume $M \in \Lambda(\mathbf{U})$ where M is of the form $c\,N\,\vec{N}$ or $(M_1, M_2)\,N\,\vec{N}$. Since M is not simple, $M \in \Lambda(U)$ for some $U \in \mathbf{U}$. Continuing with a similar argument as in (a) a contradiction follows. $\Box$

Lemma 4.10 *Suppose that for all* $\mathbf{U} \in \mathrm{dom}(F)$, $M \in \Lambda(F(\mathbf{U}))$ *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 $\mathsf{T} \in \mathbf{U}$, by Lemma 4.9 (a), and we are done. The case $K = \mathsf{F}$ is similar. $\Box$

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

Lemma 4.11 *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 $\eta := \eta_\epsilon[x := \mathbf{U}]$. If P is a constructor c, then, by assumption, $c \in \Lambda(\mathbf{U})$. By Lemma 4.9 (a), $c \in \mathbf{U}$ and therefore $\eta_\epsilon \in \mathrm{match}_c(\mathbf{U})$. For the induction step, assume $(P\theta, Q\theta) \in \Lambda(\mathbf{U})$. By Lemma 4.9 (b), $P\theta \in \Lambda(\mathbf{U}_1)$ and $Q\theta \in \Lambda(\mathbf{U}_2)$ for some $\mathrm{pair}(\mathbf{U}_1, \mathbf{U}_2) \in \mathbf{U}$. 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 $\eta := \eta_1 +\!\!+ \eta_2 \in \mathrm{match}_{(P,Q)}(\mathbf{U})$ and $\theta \in \Lambda(\eta)$. $\Box$

Proof of Theorem 3.4. Assume $[\![M]\!]\eta \neq \bot$. Then $\mathbf{U} \sqsubseteq [\![M]\!]\eta$ for some $\mathbf{U} \in D^*_{0+}$ and, by continuity, we may assume $\eta(x) \in D^*_{0+}$ for all variables x. 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 $\mathbf{U} \sqsubseteq [\![M]\!]\eta$ implies $M \in \Lambda(\mathbf{U})$ we need to prove a more general claim involving substitutions. 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. Since reducibility candidates contain all variables it suffices to prove the following

Claim. If $\theta \in \Lambda(\eta)$ and $\mathbf{U} \sqsubseteq [\![M]\!]^n\eta$, 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 $\theta(x) \in \Lambda(\mathbf{U})$, 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 $\mathbf{U} \sqsubseteq [\![(M, N)]\!]^{n+1}\eta = [\mathrm{pair}([\![M]\!]^n\eta, [\![N]\!]^n\eta)]$ we have $\mathbf{U} = [\mathrm{pair}(\mathbf{U}_0, \mathbf{U}_1)]$ 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, $\mathbf{U} \sqsubseteq [\![MN]\!]^{n+1}\eta = [\![M]\!]^n\eta \bullet [\![N]\!]^n\eta$. Therefore, $\mathbf{U}_0 \sqsubseteq [\![M]\!]^n\eta$ and $\mathbf{U}_1 \sqsubseteq [\![N]\!]^n\eta$ 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 Lemma 4.6, $(MN)\theta \in \Lambda(\mathbf{U})$.

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

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

for all $\mathbf{d} \in D^*$. We use Lemma 4.10 to prove that $\mathbf{if}(M\theta, N\theta) \in \Lambda(\mathrm{fun}(F))$. Let $\mathbf{V} \in \mathrm{dom}(F)$ and $\mathsf{T} \in \mathbf{V}$. Then $\mathbf{W} \sqsubseteq [\![M]\!]^n\eta$ for some $\mathbf{W} \subseteq F(\mathbf{V})$. By induction hypothesis and Lemma 4.4 we have $M\theta \in \Lambda(F(\mathbf{V}))$. 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

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

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 $f\vec{N} \in \Lambda(F(\vec{\mathbf{U}}))$, 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 4.11 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 $M\theta \in \Lambda(\mathbf{V}) \subseteq \Lambda(F(\vec{\mathbf{U}}))$. $\square$

5 Strong normalisation for total typed systems

We now apply our normalisation proof method to polymorphically typed systems over the base types of booleans and integers, with the non-deterministic polymorphic version of Gödel's T and its extension by barrecursion as concrete examples. The method also works for types over arbitrary positive inductive base types [4], and has been applied in [8] to dependent types.

Definition 5.1 (Typed systems) Types *are defined by the grammar*

$$\rho, \sigma ::= o \mid \iota \mid p \mid \rho \to \sigma \mid \forall p\,.\rho$$

where o and ι are the base types of booleans and natural numbers, respectively, and p ranges over a set of type variables.

A typing $\mathcal{T}$ *assigns to every function constant f a set $\mathcal{T}_f$ of tuples of types $(\rho_1, \ldots, \rho_k, \sigma)$. We will usually write $f : \vec{\rho} \to \sigma$ instead of $(\vec{\rho}, \sigma) \in \mathcal{T}_f$.*

Given a typing $\mathcal{T}$, 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:*

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

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

$$\frac{\Gamma \vdash M : \rho}{\Gamma \vdash M : \forall p\,.\rho}\ (p \text{ not free in } \Gamma) \qquad \frac{\Gamma \vdash M : \forall p\,.\rho}{\Gamma \vdash M : \rho[\sigma/p]}$$

$$\frac{\Gamma \vdash M_i : \rho_i \quad (i = 1, \ldots, k)}{\Gamma \vdash f\,M_1\,\ldots\,M_k : \sigma}\ (f : \rho_1, \ldots, \rho_k \to \sigma) \qquad \frac{\Gamma \vdash M : \rho \quad \Gamma \vdash N : \rho}{\Gamma \vdash \mathbf{if}(M, N)\ : o \to \rho}$$

A typed system *is a rewrite system together with a typing.*

Definition 5.2 (Denotational semantics of types) *Set* $[\![o]\!] := \{\mathsf{T}, \mathsf{F}\} \subseteq D_+$ *and define* $[\![\iota]\!]$ *as the least subset of* D_+ *that contains* 0 *and with* $d_1, \ldots, d_n$ *also* $([\mathrm{S}], [d_1, \ldots, d_n])$*. Furthermore, define for* $A, B \subseteq D_+$

$$A \to B := \{\mathrm{fun}(f) \mid f \in D^* \xrightarrow{!} D^*, f(A^*) \subseteq B^*\} \subseteq D_+.$$

For every type ρ *and every assignment* t *of subsets of* D_+ *to type variables we define* $[\![\rho]\!]_t \subseteq D_+$*:* $[\![p]\!]_t = t(p)$*,* $[\![\tau]\!]_t = [\![\tau]\!]$ *for* $\tau \in \{o, \iota\}$*,* $[\![\rho \to \sigma]\!]_t = [\![\rho]\!]_t \to [\![\sigma]\!]_t$*,* $[\![\forall p\,.\rho]\!]_t = \bigcap_{A \subseteq D_+} [\![\rho]\!]_{t[p:=A]}$*. We set* $[\![\rho]\!] := [\![\rho]\!]_{t_\emptyset}$ *where* $t_\emptyset(p) := \emptyset$ *for all type variables* p*.*

Lemma 5.3 $\mathbf{d} \in (A_1 \to \ldots \to A_k \to B)^*$ *iff all elements of* $\mathbf{d}$ *are of the form* $\mathrm{fun}^k(f)$ *and* $\mathbf{d} \bullet \mathbf{e}_1 \bullet \ldots \bullet \mathbf{e}_k \in B^*$ *for all* $\mathbf{e}_i \in A_i^*$*.*

Definition 5.4 (Total typed systems) *A typed system is* total *if* $[\![f]\!] \in [\![\rho_1 \to \ldots \to \rho_k \to \sigma]\!]_t^*$ *for every typing* $f : \rho_1, \ldots, \rho_k \to \sigma$ *and every type assignment* t*.*

We define a semantic analogue to the proof-theoretic typing judgements.

Definition 5.5 (Semantic typing) *For a context* $\Gamma = x_1 : \rho_1, \ldots, x_n : \rho_n$*, an environment* η *and a type assignment* t*, let* $\eta \in [\![\Gamma]\!]_t^*$ *mean* $\eta(x_i) \in [\![\rho_i]\!]_t^*$ *for all* i*. We define*

$$\Gamma \models M : \rho \;:\Leftrightarrow\; \forall t, \eta \;(\; \eta \in [\![\Gamma]\!]_t^* \Rightarrow [\![M]\!]\eta \in [\![\rho]\!]_t^* \;).$$

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

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

Theorem 5.7 *Every total typed system is strongly normalising. That is, if* $\Gamma \vdash M : \rho$ *for some context* Γ *and some type* ρ*, then* M *is strongly normalising.*

Proof. Assume $\Gamma \vdash M : \rho$. By Theorem 5.6, $\Gamma \models M : \rho$. Clearly, $\eta_\epsilon \in [\![\Gamma]\!]_{t_\emptyset}^*$. Hence $[\![M]\!] \in [\![\rho]\!]^*$, in particular, $[\![M]\!] \neq \bot$. By Theorem 3.4 it follows that M is strongly normalising. $\square$

Definition 5.8 (System NT) *The polymorphic version of Gödel's system* **T** *of primitive recursion in simple types is the typed system with the constant* R *and the rewrite rules given in Sect. 2 as well as the typing (for all types* ρ*)*

$$\mathrm{R} : \rho, (\iota \to \rho \to \rho), \iota \to \rho.$$

We also allow inessential extensions by constants like $<$ *with the rewrite rules given in Sect. 2 and the typing* $<: \iota, \iota \to o$*.*

The system **NT** *is obtained by adding the non-deterministic constant* $\|$ *with the rewrite rules given in Sect. 2 and the typing (for all types* ρ*)*

$$\| : \rho, \rho \to \rho.$$

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 Lemma 5.3 it is obvious that $[\![\|]\!] \in [\![\rho \to \rho \to \rho]\!]^*_t$. In order to see that $[\![\mathrm{R}]\!] \in [\![\rho \to (\iota \to \rho \to \rho) \to \iota \to \rho]\!]^*_t$, it suffices, by Lemma 5.3, to show that $[\![\mathrm{R}]\!] \bullet \mathbf{d}_1 \bullet \mathbf{d}_2 \bullet \mathbf{d}_3 \in A^*$ for all $A \subseteq D_+$ and $(\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3) \in A^* \times ([\![\iota]\!] \to A \to A)^* \times [\![\iota]\!]^*$. This can be proven easily by induction on the number of occurrences of the constructor S in $\mathbf{d}_3 \in [\![\iota]\!]^*$. $\square$

Remarks. 1. The addition of the non-deterministic choice operator to a strongly normalising system does in general *not* preserve strong normalisation. For example, consider the constant $f : \iota, \iota, \iota \to \iota$ 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 [22]

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

2. Kristiansen [14] suggests to use monomorphic fragments of the non-deterministic extension of system **T** above to characterise the non-deterministic polynomial hierarchy, respectively to define a higher type analogue of it.

3. For typed systems with non-overlapping rules (i.e. pairwise non-unifiable left-hand sides) the base type ι can be interpreted as the least set containing 0 and ϵ, and with d also $([\mathrm{S}], [d])$. This results in a semantics of types very similar to the one in [4] and [8].

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

Definition 5.10 (System NBR) *The system* **NBR** *of non-deterministic polymorphic barrecursion is the extension of system* **NT** *by a constant* Φ *with the rewrite rule*

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

where α_n^x *is shorthand for* $\lambda k.\mathbf{if}(\alpha\ k, x)\,(k < n)$*, and the typing (for all types* ρ, σ*)*

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

where $\tau_G := (\iota \to \rho) \to \iota \to \sigma$*,* $\tau_H := (\iota \to \rho) \to \iota \to (\rho \to \sigma) \to \sigma$*, and* $\tau_Y := (\iota \to \rho) \to o$*.* Φ *is a polymorphic variant of Spector's barrecursion in simple types [20,21].*

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 $[\![\tau_G \to \tau_H \to \tau_Y \to (\iota \to \rho) \to \iota \to \sigma]\!]^*_t$.

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

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

Fixing $A, B \subseteq D_+$ and

$$\begin{aligned}
&\mathbf{G} \in ((([\![\iota]\!] \to A) \to [\![\iota]\!] \to B)^* \\
&\mathbf{H} \in ((([\![\iota]\!] \to A) \to [\![\iota]\!] \to (A \to B) \to B)^* \\
&\mathbf{Y} \in ((([\![\iota]\!] \to A) \to [\![o]\!])^*
\end{aligned}$$

we have for $\mathbf{\Phi} := [\![\Phi]\!] \bullet \mathbf{G} \bullet \mathbf{H} \bullet \mathbf{Y}$

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

where $\mathbf{b} := [\![<]\!] \bullet (\mathbf{Y} \bullet \alpha) \bullet \mathbf{n}$ and

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

We need to show that $\mathbf{\Phi} \bullet \alpha \bullet \mathbf{n} \in B^*$ for all $\alpha \in ([\![\iota]\!] \to A)^*$ and $\mathbf{n} \in [\![\iota]\!]^*$.

Let us define a binary relation $\gg$ on $([\![\iota]\!] \to A)^* \times [\![\iota]\!]^*$:

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

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 $\mathbf{m} = \epsilon$, then $\alpha_\infty \bullet \mathbf{m} = \epsilon = \alpha_k \bullet \mathbf{m}$. Hence assume $\mathbf{m} \neq \epsilon$. 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 $\mathbf{\Phi} \bullet \alpha \bullet \mathbf{n} \in B^*$ for all $\alpha \in ([\![\iota]\!] \to A)^*$ and $\mathbf{n} \in [\![\iota]\!]^*$, by $\gg$-induction on $(\alpha, \mathbf{n})$. Clearly, it is enough to show that if $\mathsf{F} \in \mathbf{b}$, where $\mathbf{b} := [\![<]\!] \bullet (\mathbf{Y} \bullet \alpha) \bullet \mathbf{n}$, then for every $\mathbf{x} \in A^*$ we have $\mathbf{\Phi} \bullet (\mathrm{ext}(\alpha, \mathbf{x}, \mathbf{n}) \bullet (\mathbf{n} + 1) \in B^*$. 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. $\square$

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 shows 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 phenomenon is that the rewrite rule for Y requires one argument which is not present in the term Y. Further counterexamples to completeness are terms of the form $c\, N\, \vec{N}$ and $(M_1, M_2)\, N\, \vec{N}$, which have an undefined value, but are strongly normalising provided the component terms are. We now show that if we exclude these situations, the converse of Theorem 3.4 holds.

Definition 6.1 (safe terms) *A term is called* syntactically safe *if (1) constants f occur only 'fully applied', i.e. in contexts $f\,N_1 \dots N_k$ where $k \geq \mathrm{arity}(f)$, and (2) no constructor or pair occurs as the left hand side of an application. A term M is* safe *with respect to a rewrite system if all terms M' with $M \to^* M'$ are safe.*

Theorem 6.2 (Completeness) *If M is safe, 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.

Claim. If M is safe and strongly normalising, then

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

We prove the claim by main induction on $\mathsf{SN}(M)$ and side induction on (the structure of) M. Note that if P is a variable x, then 2. holds trivially, since we can set $\theta(x) := M$. 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 safe term:

$$x\,\vec{K},\ c, (M_1, M_2),\ \lambda x.M,\ (\lambda x.M)\,N\,\vec{K},\ \mathbf{if}(M,N),\ \mathbf{if}(M,N)\,L\,\vec{K},\ f\,\vec{N}\,\vec{K}$$

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

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

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

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

Case $\lambda x.M$. By side induction hypothesis, $[\![M]\!] \neq \bot$. It follows

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

Since $\mathrm{match}_P[\mathrm{fun}(\dots)] = \epsilon$ if P is a constructor or a pair, part 2 is trivial.

Case $(\lambda x.M)\,N\,\vec{K}$. By side induction hypothesis, $[\![M, N, \vec{K}]\!] \neq \bot$. Hence, by Lemma 3.5, $[\![(\lambda x.M)\,N\,\vec{K}]\!] = [\![M[N/x]\,\vec{K}]\!]$. 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)

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

and

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

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

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

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

Therefore, for part 1, it suffices to show that both arguments of the $+\!\!+$ expression are $\neq \bot$. If $\mathsf{T} \notin \mathbf{d}$, then $\mathsf{T} \in \mathbf{d} \rhd [\![M\,\vec{K}]\!] = \epsilon$. Otherwise, $\mathsf{T} \in \mathbf{d} \rhd [\![M\,\vec{K}]\!] = [\![M\,\vec{K}]\!]$. Furthermore, $\eta_\epsilon \in \mathrm{match}_{\mathsf{T}}(\mathbf{d})$ 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, $[\![M\,\vec{K}]\!] \neq \bot$. For $\mathsf{F} \in \mathbf{d} \rhd [\![N\,\vec{K}]\!]$ the argument is similar. To prove 2, let $\eta \in \mathrm{match}_P([\![\mathbf{if}(M, N)\,L\,\vec{K}]\!])$. Since P is a constructor or a pair we have, w.l.o.g., $\eta \in \mathrm{match}_P(\mathsf{T} \in \mathbf{d} \rhd [\![M\,\vec{K}]\!])$, hence, necessarily, $\mathsf{T} \in \mathbf{d}$ and $\eta \in \mathrm{match}_P([\![M\,\vec{K}]\!])$. 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 $\vec{\mathbf{d}} := [\![\vec{N}]\!]$ and $\vec{\mathbf{e}} := [\![\vec{K}]\!]$, which, by side induction hypothesis, are all $\neq \bot$. We have

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

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 $\eta = [\![\theta]\!]$. Hence

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

Since $f\,\vec{N}\,\vec{K} \to^+ M\theta\,\vec{K}$, we know that $M\theta\,\vec{K}$ is safe and, by main induction hypothesis, $[\![M\theta\,\vec{K}]\!] \neq \bot$. To prove part 2, let $\eta \in \mathrm{match}_P([\![f\,\vec{N}\,\vec{K}]\!])$. By the above, and since we may assume that P is a constructor or a pair, it follows that $\eta \in [\![M\theta\,\vec{K}]\!]$ 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 $\eta = [\![\theta']\!]$. $\square$

As a corollary to the Completeness Theorem we obtain a characterisation of strong normalisation for type correct systems.

Definition 6.3 *A* type correct system *is a typed system such that for every function constant f, whenever $f : \rho_1, \ldots, \rho_k \to \sigma$, then $k \geq \text{arity}(f)$, and for every rewrite rule $f\vec{P} \to M$ and context Γ, if $\Gamma \vdash P_i : \rho_i$ for all $i = 1, \ldots, k$, then $\Gamma \vdash M : \sigma$.*

Lemma 6.4 *Type correct systems are safe, i.e. every typeable term is safe.*

Proof. By an easy induction on the typing relation one proves: if $\Gamma \vdash M : \rho$, then (1) M is syntactically safe, (2) if ρ is a function type, then M is neither a pair nor a constructor, (3) if $M \to M'$, then $\Gamma \vdash M' : \rho$ (subject reduction). From (1) and (3) it follows that all typeable terms are safe. □

Corollary 6.5 *In a type correct system a typeable term is strongly normalising if and only if it does not denote $\bot$.*

7 Conclusion

We defined a strict domain-theoretic model for extensions of the untyped λ-calculus with pattern matching and rewrite rules. We showed that a term is strongly normalising if it does not denote $\bot$. For safe 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 be recast in the more elementary style of [8]. However, in both settings the normalisation proof is non-elementary due to the impredicative definition of reducibility candidates. Yet there exist elementary normalisation proofs for intersection types [9,24], and recently Abel [1] gave such a proof for a system very similar to the one in [8], using techniques by Joachimski and Matthes [13]. It is quite likely that Abel's proof can be extended to our non-deterministic setting.

From our semantic characterisation of strongly normalising untyped λ-terms we derived a normalisation theorem for total polymorphic typed λ-calculi and applied it to prove strong normalisation for a non-deterministic polymorphic version of Spector's barrecursion in higher types. In fact, it is quite often the case that typeable terms can easily be proven to be total, because totality is a compositional and modular property which suffices to be proven 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 domain-theoretic method, the modularity result for complete, i.e. strongly normalising and confluent rewrite systems by Toyama, Klop and Barendregt [23] can be extended to the combination of rewriting and λ-calculus as studied here.

References

[1] A. Abel. Syntactical normalization for intersection types with term rewriting rules. In *Fourth International Workshop on Higher-Order Rewriting, HOR'07, Paris, France, 25 June 2007*, 2007.

[2] 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.

[3] 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 *Lecture Notes in Computer Science*, pages 23–34. Springer, 2005.

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

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

[6] F. Blanqui, J-P. Jouannaud, and M. Okada. The calculus of algebraic constructions. In P. Narendran and M. Rusinowitch, editors, *Proceedings of RTA'99*, volume 1631 of *Lecture Notes in Computer Science*, pages 301–316. Springer, 1999.

[7] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings of the Third Annual IEEE Symposium on Logic in Computer Science (LICS'88)*, pages 82–90. IEEE Computer Society Press, 1988.

[8] 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.

[9] R. David. Normalization without reducibility. *Annals of Pure and Applied Logic*, 107:121–130, 2001.

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

[11] 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.

[12] C.B. Jay. The pattern calculus. *ACM Transactions on Programming Languages and Systems*, 26(6):911–937, 2004.

[13] F. Joachimski and R. Matthes. Short proofs of normalization. *Archive for Mathematical Logic*, 42(1):59–87, 2003.

[14] 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 *Lecture Notes in Computer Science*, pages 279–288. Springer, 2006.

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

[16] E. Moggi. Notions of Computation and Monads. *Information and Computation*, 93(1):55–92, 1991.

[17] T. Nipkow. Higher-order critical pairs. In *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science (LICS'91)*, pages 342–349. IEEE Computer Society Press, 1991.

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

[19] 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.

[20] 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, 1962.

[21] 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.

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

[23] 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.

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

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

[26] S. van Bakel, F. Barbanera, and M. Fernández. Polymorphic Intersection Type Assignment for Rewrite Systems with Abstraction and β-rule. In T. Coquand, P. Dybjer, B. Nordström, and J. Smith, editors, *Types for Proofs and Programs, International Workshop TYPES'99*, volume 1956 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2000.