\( M^{(\omega)} \) considered as a programming language

Karl-Heinz Niggl *

Technische Universität Ilmenau, Institut für theoretische und technische Informatik, Fachgebiet
Komplexitätstheorie, PF 100565, 98684 Ilmenau, Germany

Received 20 January 1998; received in revised form 17 September 1998
Communicated by I. Moerdijk

Abstract

The paper studies a simply typed term system \( M^{(\omega)} \) providing a primitive recursive concept of parallelism in the sense of Plotkin. The system aims at defining and computing partial continuous functionals. Some connections between denotational and operational semantics \( \rightarrow \) for \( M^{(\omega)} \) are investigated. It is shown that \( \rightarrow \) is correct with respect to the denotational semantics. Conversely, \( \rightarrow \) is complete in the sense that if a program denotes some number \( k \), then it is reducible to the numeral \( n_k \).

Restricting to the primitive recursive kernel \( PH' \) of \( M^{(\omega)} \), it is shown that \( \rightarrow \) is strongly normalising with uniquely determined normal forms. The twist is the design of fixed point style conversion rules for constants \( j_\mu \) accounting for parallelly bounded parallel search such that correctness and strong normalisation hold. Thereupon, minor alternations to \( \rightarrow \) bring about that every reduction sequence for a program of \( PH' \) terminates either in a numeral \( n_k \) if the program denotes \( k \), or in the term \((-1)0\) if the program denotes the "undefined" object. Thus, \( PH' \) can be considered a primitive recursive version of Plotkin's \( LPA \). © 1999 Elsevier Science B V

AMS classification: 68Q10, 68Q42, 68Q20, 68Q22, 68Q55, 68N17, 03D20

Keywords: \( \lambda \)-calculus with sequential search; Operational and denotational semantics; Scott domains; Parallelism; Weak and strong normalisation.

1. Introduction

Plotkin [18] presented a simple functional programming language \( LPA \) which is based on Scott's logic of computable functionals LCF [25] extended by a parallel conditional and a type level 2 functional denoted \( \exists \) continuously approximating the existential quantifier. LCF essentially consists of the simply typed \( \lambda \)-calculus extended

* E-mail: niggl@theoinf.tu-ilmenau.de

1 Partially supported by DFG, Germany. The author was affiliated with Ludwig-Maximilians-Universität München where this paper was written.
by some standard arithmetical constants and a special constant $Y_\sigma$ for each type $\sigma$, taking a "functional" of type $(\sigma \rightarrow \sigma)$ to its least fixed point. Referring to a generalised notion of computability on Scott Domains $D_\sigma$ (cf. [26, 5, 18, 22, 27, 23]), Plotkin proved that a functional in $D_\sigma$ is computable iff it is definable in $L_{PA+\exists}$.

From the point of view of complexity theory, functional programming languages such as $L_{PA+\exists}$ are not completely satisfactory, because they do not provide any intrinsic notion of time or structural complexity. This is mainly due to the presence of the constants $Y_\sigma$ and $\exists$, since both represent much too powerful means: $Y_\sigma$ corresponds to unbounded search, and $\exists$ to infinitely many parallel computations.

So the question is that of finding a restricted notion of computability on Scott domains which comprises a rich class of interesting partial computable functionals, and for which a reasonable subclass can be classified with respect to complexity. Therefore, building on [3, 7], a restricted computation model on Scott domains $D_\sigma$ via so-called parallel typed while programs (PTWP) is introduced in [12] such that a functional in $D_\sigma$ is PTWP-computable iff it is definable in a simply typed term system $M_\omega$. The latter consists of the simply typed $\lambda$-calculus extended by the standard arithmetical constants $0$, (+1), (-1), constants $\triangleright$, $\mu_I$ for the parallel conditional and parallelly bounded parallel search respectively, constants $R_{\triangleright}^{sim}$ for simultaneous partial primitive recursion, and a constant $\mu$ for sequential search.

The constant $\mu_I$ has been first considered as a scheme in [11]. For objects $f$, $g$ both of type $(i \rightarrow i)$ and objects $\bar{x}$ all of ground type $i$, the denotation of $\mu_I$ is such that $\mu_I(fg\bar{x})$ represents a parallel search for a zero of $f$ parallelly bounded in $\bar{x}$, together with a continuous evaluation strategy $g$. Thus, $\mu_I$ can be seen as a primitive recursive version of Plotkin's continuous approximation $\exists$ to the existential quantifier.

The functionals definable in the subsystem $R_\omega := M_\omega \setminus \{\mu\}$ are characterised in [12] by a proper subclass $HPPR_\omega$ of PTWP. Their complexity is analysed by a hierarchy of strictly increasing function algebras $R_n^{\omega}$ in the style of Heinermann [6]. The classes $R_n^{\omega}$ are characterised by corresponding program classes $HPPR_n^{\omega}$. Recently, these results have been considerably extended in [15] by adding to PTWP the data of partial sequences, and by adding constants for simultaneous partial sequence recursion to $M_\omega$.

Shifting the emphasis, a purely syntactical approach to measuring the computational complexity of the algorithm represented by a $R_\omega$ term is studied in [13]. Recent research [14, 16] shows that this approach allows one to uniformly integrate traditional characterisations of the Grzegorczyk classes at and above elementary level [21, 9] with resource-free characterisations of sub-elementary complexity classes [2, 1].

As for the relation of $M_\omega$ to $L_{PA+\exists}$, $M_\omega$ is reducible to Plotkin's PCF with parallel conditional [18] denoted $L_{PA}$. We conjecture that the converse does not hold.

The aim of this paper is to present $M_\omega$ as a simple functional programming language by introducing an operational semantics, and to study some connections between its denotational and operational semantics. While this language is itself rather far from the commonly used languages, we do hope that the present study will contribute to the study of these languages too, especially languages supporting parallel facilities.
The operational semantics for $\mathcal{M}^\omega$ is given by a reduction relation between terms denoted $t \rightarrow t'$. We are concerned with the relation between the behaviour of a program and the nature of its denotation. For us a program will be a closed term of ground type. The behaviour of a program, $t$ say, is determined by whether it terminates, and its value when it does, namely whether $t$ is reducible in finitely many steps to a numeral denoted $n_k$, and which one when it does.

As a first result we show that the behaviour of a terminating program determines its denotation. To establish this, we prove that $\rightarrow$ is correct in the sense that if $t \rightarrow t'$, then $t, t'$ denote the same object in any environment. The main achievement here is the design of a fixed point style conversion rule for the constant $\mu_\gamma$, accounting for parallelly bounded parallel search, such that correctness holds. Conversely, we prove that $\rightarrow$ is complete in the sense that if a program denotes some natural number $k$, then $t$ is reducible to the numeral $n_k$. In that way, the denotation of a program determines its behaviour.

Since the presence of the constant $\mu$ is mainly responsible for nontermination, it is natural to ask how matters are when restricting to $\mathcal{M}^\omega$. Indeed, we will show strong normalisation for $\mathcal{M}^\omega$, that is, every reduction sequence starting with a term of $\mathcal{M}^\omega$ is finite, and it ends with a term in normal form, that is a term to which no rule of $\rightarrow$ is applicable. Like for correctness, the twist is the design of a fixed point style conversion rule for $\mu_\gamma$ which guarantees termination. Strong normalisation implies that every reduction sequence for a program in $\mathcal{M}^\omega$ terminates either in a numeral $n_k$ if the program denotes $k$, or in a term denoting the undefined object $\bot$ if the program denotes $\bot$. Thus, the nontermination permitted in the denotational semantics for $\mathcal{M}^\omega$ is decidable, and $\bot$ can be viewed as a finite error. In that way, strong normalisation can be used to distinguish between termination and error for $\mathcal{M}^\omega$ programs.

Rounding off the study, strong normalisation is employed to show the uniqueness of the normal form for terms in $\mathcal{M}^\omega$. Following cosmetic purposes, minor alternations to the operational semantics achieve that every program in $\mathcal{M}^\omega$ reduces either to a numeral or to the error state $(0)\bot$ denoting $\bot$.

Concluding, that the kind of parallelism considered does not allow inconsistent results in parallel computations. This is due to working with the model of partial continuous functionals. Allowing inconsistent results in parallel computations as in [19, 8] would require a totally different semantical approach being far beyond our target. The underlying view of computation here is that of taking arguments and yielding results.

2. Preliminaries

Types are built from the ground type $t$ by means of $\rightarrow$. Note that by repeatedly decomposing the right hand side of $\rightarrow$, any composed type $\rho$ can be written uniquely as $\rho = (\rho_0 \rightarrow (\rho_2 \rightarrow \cdots \rightarrow (\rho_k \rightarrow t)\ldots))$ which we often write as $\rho_0 \rightarrow \rho_2 \rightarrow \cdots \rightarrow \rho_k \rightarrow t$ or just $\rho \rightarrow t$ with the convention of association to the right when
parenthesis are omitted. Each type \( \sigma \) has associated with it a Scott domain \( D_\sigma \) of partial continuous functionals of type \( \sigma \).

**Definition 2.1.** We define \( D_\sigma \) by induction on the buildup of \( \sigma \) as follows:

- \( D_1 := (\{\perp, 0, 1, 2, \ldots\}, \subseteq) \) with \( \subseteq \) the flat ordering, i.e. \( \perp \subseteq x \) for all \( x \in D_1 \), and every two distinct natural numbers are incomparable with respect to \( \subseteq \). \( \perp \) is called undefined value or bottom element of \( D_1 \).
- \( D_{\rho \rightarrow \rho} \) is the set \( [D_\rho \rightarrow D_\rho] \) of all continuous functionals from \( D_\rho \) to \( D_\rho \) equipped with the pointwise ordering \( \subseteq \), that is, \( f \subseteq g \iff \forall x \in D_\rho. fx \subseteq gx \).

\( f \) is continuous if for all directed subsets \( X \subseteq D_\rho, f(\sqcup X) = \sqcup \{fx \mid x \in X\} \), where \( \sqcup \) denotes the least upper bound, and a subset \( X \) of a domain \( D_\sigma \) is directed if \( \forall x, y \in X \exists z \in X, x, y \subseteq z \). Note that Scott domains are models of so-called algebraic, consistently complete cpo's (cf. e.g. [18, 27, 12]).

Note that we do not have boolean values at ground level. However, we may identify true with 0, false with any non-zero number and \( \bot \) with \( \perp \). So we do not lose anything.

Referring to the Curry homoeomorphism \([D_\rho \times D_\rho \rightarrow D_1] \cong [D_\rho \rightarrow [D_\rho \rightarrow D_1]]\) (cf. [20, 4]), we will also make use of the notation \((\rho_0, \rho_2, \ldots, \rho_{k-1} \rightarrow i)\) and thus think of an object \( F \in D_\rho \) as a functional taking arguments \( X_0, \ldots, X_{k-1} \) with \( X_i \in D_\rho \) and returning a value in \( D_1 \). Accordingly, we write \( F(X_0, \ldots, X_{k-1}) = y \) instead of \( F(X_0) \ldots (X_{k-1}) = y \), regardless of \( y \) being defined or not, since \( \perp \) is an object of our ground domain. Note that Scott domains allow partial objects as both arguments and values. This is a necessary requirement if we want the computable objects to be closed under substitution which has been first considered by Platek [17].

3. The term systems \( M^{\omega} \) and \( P^{\omega} \)

In this section we define the term systems \( M^{\omega} \) and \( P^{\omega} \), together with their denotational and operational semantics.

**Definition 3.1.** Terms in \( P^{\omega} \) are built from

- countably infinite many typed variables \( x^\sigma \) for each type \( \sigma \)
- the standard arithmetical constants 0, (+1), (−1)
- a constant \( : \) of type \( (i, i, i \rightarrow i) \) for the parallel conditional, and constants \( \mu_i \) of type \( ((i \rightarrow i), (i \rightarrow i), i \rightarrow i) \), one for each \( i \), for parallely bounded parallel search
- constants \( R_i^{im} \) of type \( (i, (i, i \rightarrow i), i, i \rightarrow i) \), one for each \( i \), for simultaneous partial primitive recursion
by means of \( \lambda \)-abstraction and application.

\( M^{\omega} \) results from \( P^{\omega} \) by adding the constant \( \mu^{(i^{(i \rightarrow i)} \rightarrow i)} \) for sequential search.
Notation. For readability we use lower case letters \(x, y, z, \ldots\) to stand for ground type variables only. Furthermore, we omit type informations when being clear from the context.

3.1. Denotational semantics

Every closed term \(t^o \in \mathcal{M}^o\) is to denote an object \([t]\) in \(D^o\). To make the construction work, however, one has to define for arbitrary terms \(t\) the value \([t]_\varphi\) of \(t\) under an environment \(\varphi\) as an element in \(D^o\), thus giving the denotational semantics for \(\mathcal{M}^o\). An environment is a type respecting mapping from the set of variables into \(D^o\).

Definition 3.2. Let \(t^o \in \mathcal{M}^o\) and an environment \(\varphi\) be given. Then we define pointwise the value \([t]_\varphi\) under \(\varphi\) as follows.

- \([x^o]_\varphi := \varphi(x)\)
- \([0]_\varphi := 0; \ [(+1)]_\varphi(x) := x + 1\) if \(x \neq \bot\), and \(\bot\) else; \([(-1)]_\varphi(x) := x - 1\) if \(x \geq 1\), and \(\bot\) else
- \([\mathcal{C}]_\varphi(x, y, z) := \begin{cases} y & x = 0 \\ z & x > 0 \text{ or } (x = \bot \text{ and } y = z) \\ \bot & \text{else} \end{cases}\)
- \([\mu_t]_\varphi(f, g, \vec{x}) := \begin{cases} g(k) & \exists x_i \in \vec{x}. k \leq x_i \text{ and } f(k) = 0 \text{ and } g(l) = g(k) \\ \bot & \text{else} \end{cases}\)
- For \(t := \#^o\), using \(t_i := [\mathcal{R}^{im}_i]_\varphi(\vec{x}, \vec{f}, y, i)\), let
  \([\mathcal{R}^{im}_i]_\varphi(\vec{x}, \vec{f}, \bot, z) := \bot\)
  \([\mathcal{R}^{im}_i]_\varphi(\vec{x}, \vec{f}, 0, z) := x_i \text{ if } z < l, \text{ and } \bot \text{ else}\)
  \([\mathcal{R}^{im}_i]_\varphi(\vec{x}, \vec{f}, y + 1, z) := f_z(y, t_0, \ldots, t_{i-1}) \text{ if } z < l, \text{ and } \bot \text{ else}\)
  \([\mu]_\varphi(f) := \begin{cases} k & f(k) = 0 \text{ and } \forall l < k. f(l) > 0 \\ \bot & \text{else} \end{cases}\)
- \([ts]_\varphi(\vec{F}) := [t]_\varphi([s]_\varphi(\vec{F}))\)
- \([\lambda x.t]_\varphi(F, \vec{F}) := [t]_\varphi[\lambda x \leftarrow F](\vec{F})\)

where \(\varphi[x \leftarrow F]\) denotes the environment resulting from \(\varphi\) by altering \(\varphi\) at \(x\) to \(F\).

Note that \(\mu_t\) is a continuous operator taking continuous functions \(f, g\) to a continuous function \(\lambda x.\mu_t f \bar{g} x\). Especially, \([\mu_t]_\varphi(f, g, \vec{x})\) is well-defined, i.e. independent of the choice of \(k\). For the proof, suppose that both \(k\) and \(k'\) meet the condition above. By symmetry we assume \(k < k'\). Hence \(g(k) = g(k')\), since \(f(k) = 0\) and \(k'\) meets the condition above.

Furthermore, note that the underlying logic for partial objects is such that \(\bot = \bot\) is considered to be true. As a consequence, the first alternative in the denotation of \(\mu_t\) reads as there is a defined \(x_i\) among \(\vec{x}\) such that for some number \(k \leq x_i\), \(f(\bar{x}, k)\) is zero and for all numbers \(l < k\), either \(f(\bar{x}, l)\) is defined and non-zero or \(g(l) = g(k)\).
regardless of whether \( f(x', 1) = 0 \) or \( f(x', 1) = \perp \) or \( g(1) = \perp = g(k) \). In that way, \([\mu_t]_\phi(f, g, x')\) represents a parallel search for a generally not the least zero of \( f \) parallelly bounded in \( x' \), together with a continuous evaluation strategy \( g \). Thus, \( \mu_t \) can be seen as a primitive recursive version of Plotkin's continuous approximation to the existential quantifier \([18]\).

Finally, note that \([\rho^{\text{sim}}]_\phi(x, \tilde{f}, y)\) has been modelled as a finite function instead of a vector of ground type objects in order to avoid the use of product types.

**Definition 3.3.** A functional \( F \in D_\alpha \) is called \( \mathcal{M}^\omega \)-definable (\( \mathcal{P}^\omega \)-definable) iff there is a closed term \( t^\alpha \in \mathcal{M}^\omega \) (\( t^\alpha \in \mathcal{P}^\omega \)) such that \( F = [t]_\phi \).

We say that \( t \) defines (or denotes) \( f \) in the environment \( \phi \) if \( [t]_\phi = f \).

Note that \( [t]_\phi \) does only depend on the assignment to the free variables of \( t \), usually referred to as coincidence lemma. More precisely, if \( \phi, \phi' \) are two environments which coincide on the free variables \( FV(t) \) of a term \( t \) in \( \mathcal{M}^\omega \), then \( [t]_\phi = [t]_\phi' \). Since the value of a closed term \( t \) is independent of any environment, we will therefore write \([t]_\phi \) instead of \([t]_\phi^\alpha \). We say that a closed term \( t \) of ground type is defined or undefined and mean by it \([t]_\phi \neq \perp \) or \([t]_\phi = \perp \) respectively. For short, this will be denoted \([t]_\phi \downarrow \) and \([t]_\phi \uparrow \), respectively.

Furthermore, the value of a term \( t \in \mathcal{M}^\omega \) does not depend on the names of its bound variables, i.e. if \( t \) and \( t' \) are identical up to their bound variables, then \([t]_\phi = [t']_\phi \) for all environments, usually referred to as bound renaming. As a consequence, one may always assume that a list of terms \( \tilde{s}^\alpha \) is simultaneously substitutable for a list of variables \( \tilde{x}^\alpha \) respectively in a term \( t \), where simultaneous substitution is defined as usual. We use \( t[\tilde{s}/\tilde{x}] \) to denote the result of simultaneously substituting \( \tilde{s} \) for \( \tilde{x} \) in \( t \). Furthermore, we identify terms which differ only in their bound variables.

Finally, we just quote the usual substitution lemma, stating that for all \( r, \tilde{s}, \tilde{x}^\alpha \in \mathcal{M}^\omega \) and environments \( \phi \), \([r[\tilde{s}/\tilde{x}]^\alpha]_\phi = [r]_\phi \phi_1 \ldots \phi_n = [r]_\phi \phi_1 \ldots \phi_n \).

**Defined terms often used.** \( 1 := (+1)0 \) defines \( 1 \), and \( \odot := \lambda x y z . \odot x(\odot x y 0)(\odot x 1 z) \) defines the sequential conditional, that is, \( \odot \perp y z = \perp \), \( \odot 0 y z = y \) and \( \odot (n + 1) y z = z \). \( \Omega := (-1)0 \) defines the undefined object \( \perp \), and for numbers \( k, n_k := (+1)^k 0 \) defines the numeral denoting \( k \).

### 3.2. Operational semantics

We are concerned with the relation between the behaviour of a program and the nature of its denotation. For us a program will be a closed term of ground type. Though higher type objects are allowed both as arguments and results of other higher type objects, we are mainly interested in computing values of ground type. In other words, the idea is that \( t \) is a data type and programs are to produce data. All of the other terms are just significant as subterms of programs.

Programs produce data via so-called operational semantics, roughly speaking a method of transforming a given program into a representation without "detours". Thus,
the computation concept for terms is more or less normalisation. The rules for this transformation process are as usual \( \alpha \beta \eta \) conversion and special conversion rules for each constant in \( \mathcal{M}^\omega \). A term is said to be in normal form if none of the conversion rules are applicable to it.

More formally, one defines inductively a reduction relation between terms denoted \( t \to t' \) with the meaning that \( t' \) results from \( t \) by converting one subterm of \( t \) according to \( \to \). As usual \( \to^* \) stands for the reflexive and transitive closure of \( \to \), i.e. \( t \to^* t' \) means that there are finitely many terms \( t_0, t_1, \ldots, t_n = t' \) with \( n \geq 0 \) such that \( t_0 \to t_1 \to \cdots \to t_n \). Correctness of \( \to \) reads as: if \( t \to t' \), then \( \llbracket t \rrbracket_\varphi = \llbracket t' \rrbracket_\varphi \) for any environment \( \varphi \), and completeness as: if \( t \) is a program with \( \llbracket t \rrbracket_\varphi = k \), then \( t \to^* n_k \).

The behaviour of a program, \( t \) say, is determined by whether it terminates, and its value when it does, that is whether there is a terminating reduction sequence for \( t \), and the value of the term with which it terminates. A reduction sequence for a term \( t \) is a possibly infinite sequence of one step reductions \( t = t_0 \to t_1 \to \cdots \), and we say that a reduction sequence terminates if it is finite and ends with a term in normal form.

Obviously, one cannot expect that every program terminates. This is mainly due to the presence of the constant \( \mu \) and its fixed point style conversion rule (see below). However, even if a program \( t \) denotes a value \( k \), it depends on the chosen reduction strategy whether \( t \) reduces to the numeral \( n_k \) or not. This is due to the presence of the parallel conditional. For example, if a reduction strategy for the term \( \mathcal{M}\mathcal{I}(\mathcal{M}(+1) t t') \) is such that one first tries to reduce the first component, then it will never stop, even if both \( t, t' \to^* n_k \).

Concerning the relation between the behaviour of a program and the nature of its denotation, correctness of \( \to \) guarantees that the behaviour of a terminating program determines its denotation. Conversely, completeness ensures that the denotation of a program determines its behaviour.

Programs of practical use are certainly those which terminate independently of the chosen reduction strategy. This leads to the notion of strong normalisation for a term \( t \), i.e. every reduction sequence for \( t \) terminates. It turns out that \( \mathcal{P}\mathcal{R}^\omega \) is a natural subclass of \( \mathcal{M}^\omega \) on which \( \to \) is strongly normalising. The twist here is a decidable, fixed point style conversion rule for \( \bar{\mu} \) for which correctness and termination hold.

The constant \( \bar{\mu} \) has been first considered as a scheme (\( \bar{\mu} \)) in [11]:

**Definition 3.4.** Given functions \( f \in D_{\overline{\mathcal{I}}_{-1}} \) and \( g \in D_{-1} \), then \( (\bar{\mu}, f, g) \) denotes the function in \( D_{-1} \), satisfying

\[
(\bar{\mu}, f, g) \overline{x} := \begin{cases} 
g(k) & \exists x_i \in \overline{x}, k \leq x_i \text{ and } f(\overline{x}, k) = 0 \text{ and } \\ 
\forall l < k. f(\overline{x}, l) > 0 \text{ or } g(l) = g(k) \\
\bot & \text{else.}
\end{cases}
\]

Reading \( \bar{\mu} \) as an operator of type \( (\overline{\mathcal{I}}, \mathcal{I} \to \mathcal{I}), (\mathcal{I} \to \mathcal{I}), (\overline{\mathcal{I}} \to \mathcal{I}) \), it turns out that \( \bar{\mu} \) does not allow a correct fixed point style conversion rule of the form

\[
\bar{\mu}, f, g_0, \ldots, n_k \ldots = \mathcal{M}(\mathcal{M}(+1) y \lambda v. (\overline{\mathcal{I}}, (-1) f))
\]
where \((-1) := \lambda x. \exists x \neq 0 (-1)x\) defines the strict predecessor, and \((-1)^{r}, (+1)^{r}\) are to be understood component-wise. This is caused by a double rôle of \(r\) in the denotation of a term \(\bar{\mu}_{r} f g^{r}\). For \(r\) act both as parameters of \(f\) and as bounds for the parallel search. Dropping all unnecessary type informations, \(\bar{\mu}_{r}\) brings in the scheme \((\bar{\mu}_{r})\), but decouples the double rôles of the parameters \(r\), thus providing a correct and complete fixed point style conversion rule. The operators \(\bar{\mu}_{r}\) and \(\bar{\mu}_{r}\) are equivalent in the sense below, implying that the partial primitive recursive functions in [11] are definable in \(\mathcal{PR}^{o}\).

Lemma 3.1. \(\bar{\mu}_{r}\) and \(\bar{\mu}_{r}\) are explicitly definable from each other.

Proof. It is straightforward to verify that \(\bar{\mu}_{r}\) can be explicitly defined from

\[\lambda f^{r} \lambda x^{r} \lambda y^{r} g^{r}(\bar{\mu}_{r}(\lambda y^{r} f^{r} x^{r}) g^{r} x^{r})\]

and conversely that \(\bar{\mu}_{r}\) is explicitly definable from \(\bar{\mu}_{r}\) by

\[\lambda f^{r} \lambda x^{r} \lambda y^{r} g^{r}(\bar{\mu}_{r}(\lambda y^{r} f^{r} x^{r}) g^{r} x^{r})\]

Definition 3.5. The operational semantics for \(\mathcal{M}^{o}\) is given by the following inductively defined reduction relation \(t \rightarrow t'\).

\begin{align*}
(\beta) & \quad (\lambda x^{o}. r)s^{o} \rightarrow r[s/x] \\
(\eta) & \quad \lambda x^{o}. r \rightarrow r[x] \text{ provided } x^{o} \notin \text{FV}(r) \\
(S) & \quad r \rightarrow r, \quad s \rightarrow s', \quad r \rightarrow r', \quad \lambda x^{o}. r \rightarrow \lambda x^{o}. r \\
(A) & \quad (-1)^{r} (+1)^{r} t \rightarrow t \\
(\emptyset) & \quad \emptyset 0 t_{1} t_{2} \rightarrow t_{1}, \quad \emptyset n_{k+1} t_{1} t_{2} \rightarrow t_{2} \text{ and } \emptyset n_{k} n_{k} \rightarrow n_{k} \\
(\mu) & \quad \text{For } l := \#r \text{ and a fresh variable } x \text{ let} \\
& \quad \bar{\mu}_{r} f g_{r_{0}} \ldots n_{k} \ldots r_{l-1} \rightarrow \emptyset f^{0} g^{0}(\bar{\mu}_{r} \lambda x. f^{0}(+1)x \lambda x. g^{0}(+1)x (-1)^{r}) \\
(A') & \quad \text{For } l := \#r \text{ let} \\
& \quad \mathcal{R}_{l}^{\bar{r} s} n_{k+1} n_{k} \rightarrow n_{k+1} n_{k} \ldots (\mathcal{R}_{l}^{\bar{r} s} n_{k+1} n_{k}) \ldots \text{ if } n_{z} < n_{l}, \text{ and } \Omega_{l} \text{ else} \\
(\mu) & \quad \mu f \rightarrow \emptyset f^{0} 0 (+1)(\mu \lambda x. f^{0}(+1)x) \text{ with } x \text{ fresh} \\
\end{align*}

where \(<\) denotes a term of \(\mathcal{PR}^{o}\) defining the strict extension of the characteristic function of its standard interpretation.

When converting a term \(\mathcal{R}_{l}^{\bar{r} s} t_{i}\), it does not suffice to require that \(t\) is of a successor form \((+1)^{r} t').\) This is due to the underlying partiality. Consider e.g. the terms \(T := \mathcal{R}_{l}^{\bar{r} s} r(\lambda y.0)(+1)\Omega, 0\) and \(T' := (\lambda y.0)\Omega,(\mathcal{R}_{l}^{\bar{r} s} r(\lambda y.0)\Omega, 0).\) Clearly, \(T \rightarrow T'\), but \([T]_{\Omega} = \bot\) and \([T']_{\Omega} = 0\) in any environment \(\phi.\) Hence for correctness of \(\rightarrow\) one has to ensure by purely syntactical means that \(t\) denotes a defined object. The way chosen is to require that \(t\) is a numeral. The same applies to the rules for \(\bar{\mu}_{r}\) and \(\emptyset.\)
Furthermore, note that the use of the predecessor term \((-1)\) instead of \((-1)\) within the rule \((\mu)\) would lead to nontermination, since \((-1)0 \rightarrow^* 0\).

As usual \(t \rightarrow t'\) implies \(FV(t) \supseteq FV(t')\), and \(t, t'\) are of the same type. Of course, there is no explicit bound renaming, for this would lead to nontermination. Rather, \(\alpha\) conversion is implicitly given within the \(\beta\) rule and serves only to rearrange, if necessary, the bound variables in \(r\) such that \(s\) is substitutable for \(x\) in \(r\).

**4. Correctness and completeness for \(\rightarrow\) on \(M^\omega\)**

**Theorem 4.1 (Correctness).** If \(t \rightarrow t'\), then \(\llbracket t \rrbracket_\varphi = \llbracket t' \rrbracket_\varphi\) for all \(\varphi\).

**Proof.** We proceed by induction of the definition of \(t \rightarrow t'\). All cases are obvious by the induction hypothesis, except possibly the rules \((\beta), (\eta), (\mu)\), and \((\mu)\).

The case \(t := (\lambda x.r)s \rightarrow r[s/x] := t'\) follows from \(\llbracket t \rrbracket_\varphi = \llbracket \lambda x.r \rrbracket_\varphi \llbracket s \rrbracket_\varphi = \llbracket r \rrbracket_\varphi[x \rightarrow \llbracket s \rrbracket_\varphi] = \llbracket t' \rrbracket_\varphi\), using the substitution lemma. Similarly, the case \(\lambda x.rx \rightarrow r\) follows from the coincidence lemma. For readability, in the remaining cases we identify constants and the defined term \(\supseteq\) with their denotations.

**Crucial case** \(t = \mu t.f gr_0 \ldots n_k \ldots r_{l-1} \) and \(t' = \supseteq f 0 g 0 (\mu t.f^l g^l (-1) \bar{x})\) where \(h^l := \lambda x.h(\pm 1)x\) for some fresh \(x\), and for all terms \(h^l\). It suffices to show

\[ \forall f, g \in D_{\leq n}, \bar{x} \in D_{\bar{x}}, \bar{x} \neq \bar{\perp} \rightarrow \mu t.f g \bar{x} = \supseteq f 0 g 0 (\mu t.f^l g^l (-1) \bar{x}) \]

where for all \(h \in D_{\leq n}\), \(h^l\) denotes the shift of \(g\) satisfying \(h^l(\perp) = h(\perp)\) and \(h^l(n) = h(n)\). So assume arbitrary objects \(f, g, \bar{x}\) with \(\bar{x} \neq \bar{\perp}\).

**Subcase** \(\mu t.f g \bar{x} \downarrow\). So there is a number \(k\) and a defined component \(x_i\) such that

\[ k \leq x_i, \ f(k) = 0, \ \text{and} \ \forall l < k. \ f(l) > 0 \ \text{or} \ g(l) = g(k). \]  

If \(k = 0\), then (2) implies \(LHS = g0 = RHS\). Suppose that \(k > 0\). Then we conclude from (2) that \(k - 1 \leq (-1)x_i, \ f^l(k - 1) = 0, \ \text{and} \ \forall l < k - 1. \ f^l(l) > 0 \ \text{or} \ g^l(l) = g^l(k - 1)\). Hence \(\mu t.f^l g^l (-1) \bar{x} = g^l(k - 1) = g(k) = LHS\). Thus if \(f0 > 0\), then \(RHS = \mu t.f^l g^l (-1) \bar{x} = LHS\). Otherwise if \(f0 = 0 \ \text{or} \ f0 \uparrow\), then (2) and \(k > 0\) imply \(g0 = g(k)\). Thus, in either case we obtain \(RHS = LHS\) from \(\mu t.f^l g^l (-1) \bar{x} = g(k) = g0\) and the definition of \(\supseteq\).

**Subcase** \(\mu t.f g \bar{x} \uparrow\). Hence for all defined components \(x_i\) and numbers \(k \leq x_i\),

\[ \neg(f(k) = 0) \ \text{or} \ \neg(\forall l < k. f(l) > 0 \ \text{or} \ g(l) = g(k)). \]

Since one of the components \(x_i\) is assumed to be defined, we know in particular

\[ f0 > 0 \ \text{or} \ f0 \uparrow. \]

We argue indirectly and assume \(RHS \downarrow\). Hence \(\mu t.f^l g^l (-1) \bar{x} \downarrow\) by (4), implying that there is a defined component \((-1)x_i\) and a number \(k \leq (-1)x_i\) such that

\[ f(k + 1) = 0 \ \text{and} \ \forall l < k. f(l + 1) > 0 \ \text{or} \ g(l + 1) = g(k + 1). \]
Since \( k + 1 \leq x_i \) and \( f(k + 1) = 0 \), (3) implies the existence of a number \( l \leq k \) satisfying \( \neg (f(l) > 0) \) or \( g(l) = g(k + 1) \). Now (5) forces \( l \) to be 0, hence \( \neg (f'0 > 0) \) and \( g0 \neq g(k + 1) \). From (4) we therefore obtain \( f0 \uparrow \). Hence the assumption RHS \( \downarrow \) gives \( g0 = \mu f1^{\prime} g1^{\prime} (-1) \bar{x} = g(k + 1) \), contradicting \( g0 \neq g(k + 1) \). Therefore RHS \( \uparrow \), concluding the proof of (1).

Case \( t := \mu f \rightarrow \sigma f0 0 \, (+1) \mu f1^{\prime} =: t' \). It suffices to show

\[
\forall f \in D_{\rightarrow, \mu}, \mu f = \sigma f0 0 \, (+1) \mu f1^{\prime}.
\]

(6)

Assume an arbitrary \( f \in D_{\rightarrow, \mu} \). Subcase \( \mu f \downarrow \). Then there is a number \( k \) such that \( f(k) = 0 \) and \( \forall l < k. f(l) > 0 \). If \( k = 0 \), then LHS = 0 = RHS. Otherwise \( k > 0 \) and \( \mu f1^{\prime} = k - 1 \), implying LHS = \( k \) = RHS. Subcase \( \mu f \uparrow \). Hence for every number \( k \),

\[
\neg (f(k) = 0) \text{ or } \neg (\forall l < k. f(l) > 0).
\]

(7)

If RHS were defined, then (7) would give \( f0 > 0 \), \( \mu f1^{\prime} = k \) for some \( k \) so that

\[
f(k + 1) = 0 \text{ and } \forall l < k. f(l + 1) > 0.
\]

(8)

Since \( f(k + 1) = 0 \), (7) would imply the existence of an \( l \leq k \) satisfying \( \neg (f(l) > 0) \).

By (8) any such \( l \) would have to be 0. Hence \( \neg (f0 > 0) \), contradicting \( f0 > 0 \).

Therefore RHS \( \uparrow \) as required. \( \square \)

**Corollary 4.2.** Every \( \mu f \) has the following fixed point operator \( Y_\sigma \) (cf. \([18]\))

\[
M_\gamma := Y_\gamma \lambda x^\sigma f^\sigma g^\sigma \overline{x}^\sigma \sigma \text{test}_l(\overline{x})(\overline{\sigma f0 g0 \, (x f1^{\prime} g1^{\prime} (-1) \bar{x})}) \Omega,
\]

where \( l := \# \text{test}_l := \lambda x_0 \ldots x_l .1 \lor (\forall D x_{l-1} \ldots (\forall D x_1 D x_0) \ldots ) \) for \( l \geq 2 \), \( \text{test}_1 := \lambda x_0. D x_0 \), with \( D := \lambda x. \exists x \, 00 \), and \( \forall := \lambda x y. \exists x0 (\overline{\gamma} y0 \Omega) \).

**Proof.** For readability, we identify constants and the defined term \( \overline{\sigma} \) with their denotations. Note that \( \forall \) defines the parallel or which, infix written, has the meaning

\[
x[\forall y] = \begin{cases} 
0 & \text{if } x = 0 \text{ or } y = 0 \\
1 & \text{if } x > 0 \text{ and } y > 0 \\
\bot & \text{else.}
\end{cases}
\]

Hence \([\text{test}_l](\overline{x}) \in \{0, \bot\} \) with \([\text{test}_l](\overline{x}) = 0 \) iff \( \overline{x} \neq \bot \). Let \( x \) be a fixed point of the functional \( F := \sigma f0 g0 \, (x f1^{\prime} g1^{\prime} (-1) \bar{x}) \). Hence for all objects \( f, g, \overline{x} \) of appropriate types,

\[
\forall x f g \overline{x} = \sigma \text{test}_l(\overline{x})(\overline{\sigma f0 g0 \, (x f1^{\prime} g1^{\prime} (-1) \bar{x})}) \bot.
\]

(9)

We show \( \forall x f g \overline{x} = \mu f g \overline{x} \) for all objects \( \overline{x}, f, g \) by induction on \( \gamma \overline{x} \), where \( \gamma \) denotes the following coding function for sequences \( \overline{x} \) of ground type objects: \( \gamma \bot := 0 \), \( \gamma x := x + 1 \) for \( x \neq \bot \), and \( \gamma \overline{x} := \{\gamma x_0, \ldots, \gamma x_{k-1}\} \) for \( \overline{x} \) in \( D_{\gamma} \). Furthermore, \( \langle \ldots \rangle \)
is a standard coding function for sequences of natural numbers satisfying $\langle \bar{n} \rangle = 0 \rightarrow \bar{n} = \vec{0}$.

**Base case** $\vec{r}\vec{x}^1 = 0$. Hence $\vec{x} = \vec{1}$, and $[\text{test}_1][\vec{x}] = \bot$. Therefore $\vec{x} f g \vec{x} = \bot = \mu_x f g \vec{x}$ by (9). **Step case** $\vec{r}\vec{x}^1 > 0$. Hence $\vec{x} \neq \vec{1}$, $[\text{test}_1][\vec{x}] = 0$, and $x f^1 g^1(-1)\vec{x} = x f^1 g^1(1)\vec{x}$ by the I.H. Furthermore, (1) in the previous proof and $\vec{x} \neq \vec{1}$ give

$$\mu_x f g \vec{x} = : f 0 g 0 (\mu_x f^1 g^1(1)\vec{x}). \quad (10)$$

Hence the claim follows from (9), (10), and the induction hypothesis. \(\square\)

The previous corollary and Lemma 3.1 show that both $\mu^*_{\neg}$ and $\mu^*_{!}$ are by far weaker than the existential quantifier $\exists$ which is not definable in Plotkin's PCF with parallel conditional denoted $\mathcal{L}_{PA}$ (cf. [18]). It follows that $\mathcal{M}^*$ and $\mathcal{P}^*$ are reducible to $\mathcal{L}_{PA}$.

Our next aim is to show that $\neg \rightarrow$ is complete. For doing so, we employ a powerful method from proof theory due to W.W. Tait [28] which is simply a proof of the required property by induction on the structure of $\mathcal{M}^*$-terms, requiring a suitable induction hypothesis on higher types.

**Definition 4.1.** For every term $t^\rho$ we define inductively what it means that $t$ is computable (or $\text{Comp}_\rho(t)$ for short) by the following three clauses.

(C1) A program $t'$ is computable iff $t \rightarrow^* n_k$ whenever $[t] = k$.

(C2) If $t' \rightarrow t$ is a closed term, then $r$ is computable iff $\text{Comp}_\rho(rs)$ whenever $s^\rho$ is a closed computable term.

(C3) If $t^\rho$ is an open term, then $r$ is computable iff $\text{Comp}_\rho(r[s/x^\rho])$ whenever $s^\rho$ are closed computable terms, and $\text{FV}(r) \subseteq \vec{s}^\rho$.

The following theorem is also referred to as weak computability, in contrast to the notion of strong computability discussed later when restricting to $\mathcal{P}^\rho$.

**Theorem 4.3** (Completeness). Every term $t \in \mathcal{M}^*$ is computable.

**Proof.** We proceed by induction on the structure of $t$. In order to illustrate the general strategy of reducing such terms, we will do a little bit more than needed in the cases where $t$ begins with one of the constants $\neg, \mu^*, \mu, \mathcal{R}^{\neg\mu}$.

**Case** $t = x^\rho$. Obviously, $t$ is computable, since every instantiation of it by a closed computable term is computable.

**Case** $t = r^\rho \neg s^\rho$. Assume that $\text{FV}(t) \subseteq \vec{s}^\rho$, and let $\vec{s}^\rho$ be a list of closed computable terms. Hence $\text{Comp}_{\rho}(r[t[\vec{s}/\vec{x}]])$ and $\text{Comp}_{\rho}(s[t[\vec{s}/\vec{x}]])$ by the induction hypothesis. From (C2) we obtain $\text{Comp}_{\rho}(t[t[\vec{s}/\vec{x}]])$, hence $\text{Comp}_{\rho}(t(t[\vec{s}/\vec{x}]))$ by (C3).

**Case** $t = \lambda x^\rho r^\rho$. It suffices to show $t[N/x]s^\rho \rightarrow^* n_k$ whenever $\vec{N}^\beta, \vec{s}^\rho, \vec{s}^\rho$ are closed computable terms such that $[t[N/x]s^\rho] = k$, where $\text{FV}(t) \subseteq \vec{s}^\rho$ and $\rho = \bar{\rho} \rightarrow 1$. Since

$t[N/x]s^\rho \rightarrow (r[N/x])s^\rho x^\rho \bar{s}^\rho = r[N,s/x,x^\rho]s^\rho$
we know $\langle r[N,s,x',x'] \rangle = k$ by correctness. Thus, since $r$ is computable by the induction hypothesis, we obtain $r[N,s,x',x'] \rightarrow^* n_k$, and so does $t[N/x]s$.

The case $t = 0$ is obvious, and the case $t = (-1)$ is treated similarly to the case $t = (+1)$. So let $r$ be a computable program $r$ such that $\langle (+1)r \rangle = k$. Hence $k > 0$ and $\langle r \rangle = k - 1$. Therefore $r \rightarrow^* n_{k-1}$, implying $(+1)r \rightarrow^* n_k$.

Case $t = :I$. We have to show $\forall r_0 r_1 r_2 \rightarrow^* n_k$ whenever $r_0, r_1, r_2$ are computable programs such that $\langle r_0 r_1 r_2 \rangle = k$. We reduce in parallel $r_0, r_1, r_2$, and by assumption, correctness, and the induction hypothesis, we must encounter one of the following situations:

\begin{itemize}
  \item $r_0 r_1 r_2 \rightarrow^* 0 r_1' r_2' \rightarrow r_1' \rightarrow^* n_k$ or
  \item $r_0 r_1 r_2 \rightarrow^* n_{m+1} r_1' r_2' \rightarrow r_2' \rightarrow^* n_k$ or
  \item $r_0 r_1 r_2 \rightarrow^* r_0' n_k \rightarrow n_k$.
\end{itemize}

Case $t = :U$. We show $\mu t f g r \rightarrow^* n_k$ whenever $f, g, r'$ are closed computable terms such that $\langle (1)t f g r \rangle = k$. Given such $f, g, r'$, then there is a defined $r_i$ and an $m \leq \langle r_i \rangle$ satisfying

$$\langle f n_i \rangle = 0 \quad \text{and} \quad \forall l < m. \langle f n_l \rangle > 0 \quad \text{or} \quad \langle g n_i \rangle = \langle g n_m \rangle.$$  

To describe the change of components during the reduction process, we define updates $h^i$ and $r^i$ for arbitrary terms $h^{i-1}$ and lists $r$ of ground type terms respectively by

$$h^0 := h, \quad \text{and} \quad h^{i+1} := \lambda x. h^i(+1)x \quad \text{with} \quad x \text{ new},$$

$$r^0 := r, \quad \text{and} \quad r^{i+1} := (-1)r^i.$$  

It is straightforward to show:

If $h, r$ are all computable, then so are all of the updates $h^i$ and $r^i$.  

As to the reduction sequence for $\mu t f g r$, we first reduce in parallel all of the programs $r$, and by (11) there is at least an $r_j$ satisfying $r_j \rightarrow^* n_{p_j}$. Hence for each such program,

$$\mu t f g r \rightarrow^* \mu t f g r_0 \ldots n_{p_j} \ldots r'_{j-1}.$$  

Let $s := r_0 \ldots n_{p_j} \ldots r'_{j-1}$. For each such $p_j$ we apply the $\mu t$ rule $p_j + 1$ times, and end up therefore with the following term

$$t_j := : 0 f 0 g 0 ( : 0 f^1 0 g 1 0 ( : \ldots ( : 0 f^p 0 g^p 0 ( \mu t f^{p_j+1} g^{p_j+1} s^{p_j+1} ) \ldots ) ) ).$$

By (11) and the definition of the updates we know

$$\langle f^m 0 \rangle = 0 \quad \text{and} \quad \forall l < m. \langle f^l 0 \rangle > 0 \quad \text{or} \quad \langle g^l 0 \rangle = \langle g^m 0 \rangle = k.$$  

Therefore, by (12) and $m \leq p_j$ there is a term $t_j$ such that $m \leq p_j$ and

$$f^m 0 \rightarrow^* 0, g^m 0 \rightarrow^* n_k, \forall l < m. f^l 0 \rightarrow^* n_{k_l+1} \text{ for some } k_l \text{ or } g^l 0 \rightarrow^* n_k.$$
Thus, by reducing in parallel all of the terms $f^l 0'$, $g^l 0'$ for $l \leq p_j$, we end up with

$$t_j \rightarrow^* f^l 0' g^l 0' (f^{m-1} 0' g^{m-1} 0' (\ldots (f^{p_j} 0' g^{p_j} 0' (\ldots))$$

with $f^l 0' = n_{k+l}$ or $g^l 0' = n_k$ for all $l < m$. Hence $t_j \rightarrow^* n_k$ by the $\rightarrow$ rules. All in all, this proves that $\mu f g r \rightarrow^* n_k$ as desired.

**Case $t = \mu$.** We have to show $\mu f \rightarrow^* n_k$ whenever $f$ is a closed computable term satisfying $\llbracket \mu f \rrbracket = k$. Given such $f$, we know $\llbracket fn_k \rrbracket = 0$ and $\forall l < k. \llbracket fn_l \rrbracket > 0$. Using the updates notation and (12), this implies

$$f^k 0 \rightarrow^* 0,$n_l \rightarrow^* n_{k+l+1}$$

for some $k_l$.

We follow here a *sequential strategy*, and we begin with the step

$$\mu f \rightarrow^* f 0 0 (1) \mu f^k.$$ 

Then we try to reduce $f 0$ to a numeral. If we succeed in it with $f 0 \rightarrow^* 0$, then we convert the resulting term $\rightarrow 0 0 (1) \mu f^k$ to 0, and stop there. Otherwise if $f 0 \rightarrow^* n_{l+1}$ for some $l$, then we convert the resulting term $\rightarrow n_{l+1} 0 (1) \mu f^k$ to $(1) \mu f^k$, and we continue with carrying out the next step for $(1) \mu f^k$. The strategy fails if no such update $f^l 0$ is reducible to 0. However, by assumption we will end up after finitely many steps with

$$\mu f \rightarrow^* (1)^{k_l} (\mu f^k)$$

$$\rightarrow (1)^{k_l} (\rightarrow f^0 0 (1) \mu f^{k+1})$$

$$\rightarrow^* (1)^{k_l} (\rightarrow 0 0 (1) \mu f^{k+1})$$

$$\rightarrow (1)^{k_l} 0 = n_k.$$  

**Case $t = R^m_{I}$ where $l := \#I$.** It suffices to show $R^m_{I} R s N I \rightarrow^* n_k$ whenever $R, s, N, I$ are closed computable terms satisfying $\llbracket R^m_{I} R s N I \rrbracket = k$. First we try to reduce successively the components $I, N$ to numerals. If we succeed in doing so, we end up with a term to which one of the $R$ rules is applicable, and then we go on with trying to reduce the resulting term. The strategy fails if either $I$ or $N$ is not reducible to a numeral. However, by assumption we know $\llbracket I \rrbracket = z$ for some $z < l$, and $\llbracket N \rrbracket = m$ for some $m$. Hence

$$R^m_{I} R s N I \rightarrow^* R^m_{I} R s n_m n_z =: t'. $$

It suffices to show that for every $m$ and $z < l$ such that $\llbracket t' \rrbracket = k$, $t'$ is reducible to $n_k$. The proof is by induction on $m$. In the *base case* $m = 0$ we have

$$t' \rightarrow_r r_z \rightarrow^* n_k$$

where the latter follows from $\text{Comp}_r(r_z)$ and $\llbracket r_z \rrbracket = k$ by correctness. As for the *step case*, we have

$$t' \rightarrow_r s_z n_{m-1} (R^m_{I} R s n_{m-1} 0) \cdots (R^m_{I} R s n_{m-1} n_{l-1}) \rightarrow^* n_k$$
where the latter follows from correctness, the computability of \( s_i \), and the fact that all of the components \( \mathcal{R}_i \) are closed computable terms by the induction hypothesis. 

5. Strong normalisation for \( \rightarrow \) on \( \mathcal{P} \mathcal{R}^{\infty} \)

As \( \mu \) is mainly responsible for nontermination, one might ask how matters are when restricting to \( \mathcal{P} \mathcal{R}^{\infty} \). Indeed, strong normalisation holds for \( \mathcal{P} \mathcal{R}^{\infty} \). For the proof, we employ a generalisation of Tait’s method introduced in [29] based on so-called strong computability predicates.

Strong normalisation for \( \mathcal{P} \mathcal{R}^{\infty} \) implies that every reduction sequence for a program in \( \mathcal{P} \mathcal{R}^{\infty} \) terminates either in a numeral \( n_k \) if the program denotes \( k \), or in a term denoting \( \bot \) if the program denotes \( \bot \). Thus, the nontermination permitted in the denotational semantics for \( \mathcal{P} \mathcal{R}^{\infty} \) is decidable, and \( \bot \) can be viewed as a finite error. In that way, strong normalisation can be used to distinguish between termination and error for \( \mathcal{P} \mathcal{R}^{\infty} \) programs.

**Definition 5.1.** For every term \( t \) we define inductively what it means that \( t \) is strongly computable (or \( SC_\sigma(t) \) for short) by the following two clauses.

(S1) A term \( t \) is strongly computable if it is strongly normalisable, i.e. every reduction sequence for \( t \) terminates.

(S2) A term \( r^\sigma \rightarrow^\rho \) is strongly computable if \( SC_\rho(rs) \) whenever \( SC_\sigma(s) \).

A term \( r^\sigma \) is strongly computable under substitution if \( SC_\sigma(r[s/x]) \) whenever \( s^\rho \) are strongly computable terms, and \( FV(r) \subseteq s^\rho \).

The main endeavour will be to show that every term is strongly computable under substitution, for this trivially implies that every term is strongly computable. For the proof, we shall do well to first provide some basic relations between \( \rightarrow \) and substitution, \( \rightarrow \) and strong computability, and between the latter and strong normalisation.

**Lemma 5.2.** (a) If \( s \rightarrow s' \) respectively, then \( t[s/x] \rightarrow^* t[s'/x] \).

(b) If \( t \rightarrow t' \), then \( t[s/x] \rightarrow^* t'[s'/x] \).

(c) If \( t \rightarrow t' \) and \( s \rightarrow s' \) respectively, then \( t[s/x] \rightarrow^* t'[s'/x] \).

**Proof.** (a) is proved by a straightforward induction on the structure of \( t \), and (c) is an immediate consequence of (a) and (b). The proof of (b) is by induction on the definition of \( t \rightarrow t' \). All cases are obvious by the induction hypothesis or part (a), except possibly the \( \beta \) rule. So let \( t := (\lambda x^\sigma r)s^\sigma \rightarrow t[s/x] =: t' \). Since \( t[s/x] = (\lambda x^\sigma r[s/x])s[s/x] = t'[s/x] \), we obtain

\[
(t[s/x]) = (r[s/x][x^\sigma]) \rightarrow^* r[s[s/x][x^\sigma]] = t'[s/x]
\]
where (*) follows from the tacit assumption that $\overline{s}$ is substitutable for $\overline{x}$ in $t$, hence $\lambda x^\sigma \notin \text{FV}(\overline{s})$, and since we may assume $x \notin \overline{x}$ by bound renaming.

Lemma 5.2. If $t \rightarrow t'$ and $\text{SC}_\sigma(t)$, then $\text{SC}_\sigma(t')$.

Proof. By definition it suffices to show that $t'(\overline{s})$ is strongly normalisable whenever $\overline{s}^\sigma$ are strongly computable terms, where $\sigma - \tilde{\sigma} \rightarrow t$. By assumption and rule (S) we know $t\overline{s} \rightarrow t'(\overline{s})$. Hence every infinite reduction sequence for $t'(\overline{s})$ would provide an infinite reduction sequence for $t\overline{s}$, contradicting $\text{SC}_\sigma(t)$.

Lemma 5.3. (a) Every variable $x^\sigma$ is strongly computable.

(b) Every strongly computable term is strongly normalisable.

Proof. We proceed by simultaneous induction on $\sigma$. The base case for (a) is obvious, since $x^\sigma$ is in normal form. The base case for (b) holds by definition. As for the step case, assume $\sigma = \rho \rightarrow \tau$. For (a), it suffices to show that $x\rho\overline{s}$ is strongly normalisable whenever $\overline{s}^\rho, \overline{s}^\tau$ are strongly computable terms, where $\tau - \tilde{\tau} \rightarrow t$. But this is obviously true, for every infinite reduction sequence for $x\rho\overline{s}$ would provide an infinite reduction sequence for $s$ or some $s_i$, contradicting the induction hypothesis (b) for $s, \overline{s}$.

Concerning the step case for (b), suppose that $\text{SC}_\tau(t)$. Hence $\text{SC}_\rho(\lambda x^\rho \cdot r^\rho)$ by the induction hypothesis (a). As $\tau x$ is strongly normalisable by the induction hypothesis (b), it follows that every reduction sequence for $t$ terminates.

Theorem 5.4. Every term $t \in \mathcal{P}_\mathcal{N}^\omega$ is strongly computable under substitution.

Proof. We proceed by induction on the structure of $t \in \mathcal{P}_\mathcal{N}^\omega$. The case $t = x^\sigma$ is obvious.

Case $t = r^\rho, ts^\tau$. Given strongly computable terms $\overline{s}$, the induction hypothesis yields $\text{SC}_{\rho \rightarrow \tau}(r[\overline{s}/\overline{x}])$ and $\text{SC}_\sigma(s[\overline{s}/\overline{x}])$. Hence $\text{SC}_\tau(t[\overline{s}/\overline{x}])$ by definition.

Case $t = \lambda x^\rho \cdot r^\rho$. It suffices to show that every reduction sequence for $t[\overline{N}/\overline{x}]\overline{s}$ terminates whenever $\overline{N}, \overline{s}^\rho, \overline{s}^\tau$ are strongly computable terms, where $\rho = \overline{\rho} \rightarrow \overline{t}$. We argue indirectly and assume an infinite reduction sequence

$$(\lambda x^\rho \cdot r[\overline{N}/\overline{x}])\overline{s} = t_0 \rightarrow t_1 \rightarrow \ldots$$

As $\text{SC}_\rho(\overline{N}/\overline{x})$ by the induction hypothesis, Lemma 5.3(b) gives a $t_i$ such that

$$t_i = (\lambda x^\sigma r'')s'\overline{s}'' \rightarrow r'[s'/x^\sigma]\overline{s}'' = t_{i+1}$$

with $r[\overline{N}/\overline{x}], s, \overline{s} \rightarrow^* r', s', \overline{s}'$ respectively. Therefore, all of the terms $r', s', \overline{s}'$ are strongly computable by Lemma 5.2. As $r[\overline{N}, s/\overline{x}, x^\sigma] \rightarrow^* r'[s'/x^\sigma]$ and $\text{SC}_\sigma(r[\overline{N}, s/\overline{x}, x^\sigma])$ by the induction hypothesis, we conclude $\text{SC}_\rho(r'[s'/x^\sigma])$ from Lemma 5.2, and hence $\text{SC}_\rho(t_{i+1})$ by definition. Thus, by Lemma 5.3 the reduction sequence for $t_{i+1}$ is finite, contradicting the assumption on the given reduction sequence.
In the remaining cases where $t$ is a constant $c^{\beta \rightarrow i}$ it suffices to show that every reduction sequence for $c^t$ terminates whenever $c^t$ are strongly computable terms. The case $t = 0$ is obvious.

**Case** $t = (+1)$. Assume a strongly computable term $r'$, hence every reduction sequence for $r$ terminates by definition. Thus, every reduction sequence for $(+1)r$ terminates, too.

**Case** $t = (-1)$. Given a reduction sequence for $(-1)r$ with a strongly computable term $r$, we are done by definition if all reductions take place within $r$. Otherwise there is a first member $(-1)(+1)r' \rightarrow r'$ and $r \rightarrow^* (+1)r'$. Hence we are done by Lemma 5.2 and Lemma 5.3.

**Case** $t = :0$. Given $SC_i(Q), SC_i(r_0), SC_i(r_1), SC_i(r_2)$, and a reduction sequence

$$s_0 \rightarrow s_1 \rightarrow \ldots$$

we are done by the assumption and Lemma 5.3(b) if all reductions take place within the components $r_i$ only. Otherwise there is a first member $s_i$ such that

$$s_i = :0' r'_0 r'_1 r'_2 \rightarrow s_{i+1} \in \{r'_1, r'_2\}$$

by one of the $:0$ rules, where $r_0, r_1, r_2, \rightarrow^* r'_0, r'_1, r'_2$ respectively. Hence by Lemma 5.2 the reduction sequence for $s_{i+1}$ is finite, completing the $:0$ case.

**Case** $t = \mu$. We have to show that $\mu_t f g r$ is strongly computable whenever $f, g, r$ are. Here and in the subsequent case we will benefit from the denotational semantics for $\mathcal{R}_\omega$ by employing the coding function $\Gamma,^\top$ used in the proof of Corollary 4.2. More precisely, given an arbitrary but fixed environment $\varphi$, we proceed by induction on $m := \max_i [r_i]_\varphi$, showing that for all strongly computable terms $r, f, g$, every reduction sequence for $\mu_t f g r$ terminates. So consider an arbitrary reduction sequence

$$\mu_r f g r = s_0 \rightarrow s_1 \rightarrow \ldots$$

with strongly computable terms $r, f, g$. In the base case $m = 0$ we conclude that none of the terms $r$ is reducible to a numeral, for $r \rightarrow^* n_k$ implies $[r_i]_\varphi = k$ by correctness, and hence $m > 0$. Therefore all reductions in the given sequence are in reducts of the components $f, g, r$. Thus, we are done by Lemma 5.2 and the assumption on $f, g, r$. As for the step case $m > 0$, by the argument above it suffices to consider the case that there is a member $s_j$ in the given sequence such that

$$s_j = \mu_r F G R \rightarrow :0' F(0) G(0) (\mu_t F(1) G(1) R(1)) = s_{j+1}$$

with $f, g, r \rightarrow^* F, G, R$, respectively, and $R_i = n_k$ for some $i, k$ — recall the update notation introduced in the completeness proof. Using Lemma 5.2 and the assumption on $f, g, r$ we obtain that all of the terms $F, G, R$ are strongly computable. This implies (see completeness proof) that all of the terms $F0, G0, F1, G1, R1$ are strongly computable, too. Since $:0'$ is already proved to be strongly computable, it therefore suffices to show that $\mu_r F(1) G(1) R(1)$ is strongly computable. But this follows from the induction hypothesis, for $[r_i]_\varphi = [R_i]_\varphi$ by correctness, and hence $m - 1 = \max_i [r_i]_\varphi$. \top
Case \( t = R^\text{sim}_{\tau} \) with \( l := \#i \). We have to show that \( R^\text{sim}_{\tau}t_Nl \) is strongly computable whenever all of the terms \( t, s, N, I \) are. Given an arbitrary but fixed environment \( \phi \), we proceed by induction on \( m := r[N]_{\tau} \), showing that for all strongly computable terms \( t, s, N, I \), every reduction sequence for \( R^\text{sim}_{\tau}t_Nl \) terminates. So consider any reduction sequence
\[
R^\text{sim}_{\tau}t_Nl = t_0 \rightarrow t_1 \rightarrow \ldots
\]
with strongly computable terms \( t, s, N, I \). In the base case \( m = 0 \) we argue as in the previous case that \( N \) is not reducible to a numeral, hence there cannot be an application of one of the \( R \) rules to the outermost \( R^\text{sim}_{\tau} \)-term in the given reduction sequence. Thus, we are done by Lemma 5.2 and the assumption on the components \( t, s, N, I \). As for the step case \( m > 0 \), by Lemma 5.2 and the assumption on the components \( t, s, N, I \) it suffices to consider the case that there is a member \( t_i \) in the given sequence such that
\[
t_i = R^\text{sim}_{\tau}t'_i s'_i n_k n_z \rightarrow_\Lambda t_{i+1} \in \{ r'_i, s'_{n_k-1} \ldots (R^\text{sim}_{\tau}t'_i s'_i n_k n_z), \ldots, \Omega_i \}
\]
with \( t, s, N, I \rightarrow^* t', s', n_k, n_z \) respectively, and \( n_z < n_i \). By Lemma 5.2 all of the terms \( t', s', n_k, n_z \) are strongly computable. It suffices to show that \( t_{i+1} \) is strongly computable. But this follows from the strong computability of \( t', s' \), correctness and the induction hypothesis. \( \square \)

**Corollary 5.5.** Every term in \( \mathcal{P} \mathcal{R}^{\omega} \) is strongly normalisable.

Having shown that every term in \( \mathcal{P} \mathcal{R}^{\omega} \) reduces to a normal form, independently of the chosen reduction strategy, one might ask whether each reduction strategy for a given term yields the same term. In fact, the normal form of a term is uniquely determined. The proof consists of two steps: in the first step we will establish the so-called weak Church-Rosser property or local confluence for \( \rightarrow \) on \( \mathcal{M}^{\omega} \), and in the second step we employ a technique due to Newman [10], showing that local confluence and strong normalisation for \( \mathcal{P} \mathcal{R}^{\omega} \) imply the uniqueness of the normal form.

**Lemma 5.6** (Local confluence for \( \mathcal{M}^{\omega} \)). If \( t \rightarrow t' \) and \( t \rightarrow t'' \), then one can find a term \( t''' \) such that \( t' \rightarrow^* t''' \) and \( t'' \rightarrow^* t''' \).

**Proof.** Given \( t \rightarrow t' \) and \( t \rightarrow t'' \), we proceed by induction on the definition of \( t \rightarrow t' \).

**Case** \( t = (\lambda x^aindered{.}r)s \). By symmetry we assume that \( t' = t[s/x] \), leaving two possibilities for \( t'' \). If \( t'' = (\lambda x^aindered{.}r)'s \) with \( r \rightarrow r' \), then Lemma 5.1(b) gives \( r[s/x] \rightarrow^* r'[s/x] \). Hence \( t''' := r[s/x] \) will do. Otherwise if \( t'' = (\lambda x^aindered{.}r)'s \) with \( s \rightarrow s' \), then Lemma 5.1(a) implies \( r[s/x] \rightarrow^* r'[s'/x] \). Again we define \( t''' := r[s'/x] \).

**Case** \( t = \lambda x. r x \) with \( x \notin \text{FV}(r) \). By symmetry we assume \( t' = r \), leaving only the possibility \( t'' = \lambda x. r' x \) with \( r \rightarrow r' \). As \( x \notin \text{FV}(r) \subset \text{FV}(r') \), we define \( t''' := r' \).

**Case** \( t = r s \) where \( r \) is no lambda abstraction. By symmetry we assume \( t' = r' \) with \( r \rightarrow r' \), leaving two possibilities for \( t'' \). If \( t'' = r'' s \) with \( r \rightarrow r'' \), the induction
hypothesis yields a suitable term $r'''$, hence $t''' := r''' s$ will do. Otherwise if $t'' = r s'$ with $s \rightarrow s'$, we define $t''' := r' s'$.

Case $t = \lambda x. r$ where $t$ is no $\eta$ redex. Then $t' = \lambda x. r'$ with $r \rightarrow r'$, and $t'' = \lambda x. r''$ with $r \rightarrow r''$. The induction hypothesis provides a suitable term $r'''$, and we take $t''' := \lambda x. r'''$.

Case $t = \mathcal{N} t_0 t_1 t_2$. If $t', t''$ result from $t$ by reducing distinct subterms of the same component of $t$, then we are done by the induction hypothesis. Otherwise if $t' = \mathcal{N} t'_0 t'_1 t'_2$ with $t_i \rightarrow t'_i$ for exactly one $i$, and $t'' = \mathcal{N} t''_0 t''_1 t''_2$ with $t_j \rightarrow t''_j$ for exactly one $j$ with $i \neq j$, then $t'''$ results from $t$ by replacing $t_i, t_j$ with $t'_i, t''_j$ respectively. All other cases are by symmetry such that $t'$ results from $t$ by one of the $\mathcal{N}$ rules, and $t''$ by reducing a component $t_i$. Hence $t' \in \{t_1, t_2\}$, and $t'' = \mathcal{N} t''_0 t''_1 t''_2$ with $t_i \rightarrow t''_i$ for exactly one $i$. If $t_0 = 0$, then $t''' := t'_i$ will do. If $t_0 = n_{k+1}$, then we take $t''' := t'_2$.

Otherwise $t_1 = t_2 = n_k$ for some $k$, and we define $t''' := t'_i$.

Case $t = \mu f g r$. If $t', t''$ result from $t$ by reducing components of $t$, then we argue as above. Otherwise we assume by symmetry that $t' = \mathcal{N} f' g 0 (\mu f' \downarrow g' F')$ and $t'' = \mu f G R$, where $F, G, R$ result from $f, g, r$ by reducing exactly one component. In either case, $t''' := \mathcal{N} F 0 G 0 (\mu f \downarrow G \downarrow R)$ will do, since $f, g, r \rightarrow F, G, R$ respectively implies $f', g', r' \rightarrow F', G', R'$ respectively.

The cases $t = \mathcal{N} t_0 t_1 t_2$ and $t = \mu f$ are treated in a similar way. □

**Theorem 5.7** (Uniqueness of the normal form). If $t$ in $\mathcal{P}$ is such that $t \rightarrow^* t'$ and $t \rightarrow^* t''$ where both $t'$ and $t''$ are in normal form, then $t'$ and $t''$ are identical.

**Proof.** A term is called ambiguous if it has at least two distinct normal forms. It suffices to show that $(\ast)$ if $r \in \mathcal{P}$ is ambiguous, then one can find an ambiguous term $r' \in \mathcal{P}$ satisfying $r \rightarrow r'$. For if $t \in \mathcal{P}$ were ambiguous, then $(\ast)$ would give rise to an infinite reduction sequence for $t$, contradicting Corollary 5.5. For the proof of $(\ast)$, suppose that $r \rightarrow^* r'$ and $r \rightarrow^* r''$ where $r', r''$ are distinct terms in normal form. Hence $r \rightarrow r_1 \rightarrow^* r'$ and $r \rightarrow r_2 \rightarrow^* r''$. Lemma 5.6 and Corollary 5.5 yield a term $r'''$ in normal form satisfying $r_1 \rightarrow^* r'''$ and $r_2 \rightarrow^* r'''$. Since $r'$ is distinct from $r''$, we assume by symmetry that $r'$ is distinct from $r'''$. Hence $r_1$ is ambiguous and $r \rightarrow r_1$. □

By the previous results, every reduction sequence for a given program in $\mathcal{P}$ terminates in a unique normal term which is either a numeral $n_k$ if the program denotes $k$, or it is any term denoting $\perp$ if the program denotes $\perp$. So one might ask whether one could extend $\rightarrow$ for the cosmetic purpose that then every undefined program reduces to a most simplified normal form. One way to do it is to extend the language by $\perp$ itself and then appropriately arrange the reduction rules as done in [24]. This would mean to treat the object $\perp$ in the model as a finite error or under specification, in contrast to [18] where the undefined value is to be interpreted as a nonterminating computation.

The way, however, we favour is to think of the term $\Omega := (-1)0$ as the normal form of undefined programs, since $\Omega$ is denoting $\perp$, and it is already in our language.
Concerning the additional conversion rules, one has only to add the rules:

- \( \vdash \Omega_i n_k n_1 \rightarrow \Omega_i \) whenever \( n_k \neq n_1 \)
- \( (-1)\Omega_i \rightarrow \Omega_i \)
- \( j_{ij} f g \Omega_i \rightarrow \Omega_i \)
- \( \mathcal{A}_i^\text{sim} \rightarrow \Omega_i \), and \( \mathcal{A}_i^\text{sim} \rightarrow \Omega_i \).

No doubt, these alternations leave untouched correctness, strong normalisation for \( \mathcal{P} \mathcal{R}^\mu \), and uniqueness of the normal form. Moreover, every reduction sequence for a program in \( \mathcal{P} \mathcal{R}^\mu \) will then terminate either in a numeral or in the error state \( \Omega_i \).

Acknowledgements

I would like to thank the referees for their thorough proofreading and their helpful suggestions concerning the presentation of the material.

References