On Continuous Normalization

Klaus Aehlig* and Felix Joachimski

{aehlig|joachski}@mathematatik.uni-muenchen.de
Mathematisches Institut, Ludwig-Maximilians-Universität München
Theresienstrasse 39, 80333 München, Germany

Abstract. This work aims at explaining the syntactical properties of continuous normalization, as introduced in proof theory by Mints, and further studied by Ruckert, Buchholz and Schwichtenberg.
In an extension of the untyped coinductive λ-calculus by void constructors (so-called repetition rules), a primitive recursive normalization function is defined. Compared with other formulations of continuous normalization, this definition is much simpler and therefore suitable for analysis in a coalgebraic setting. It is shown to be continuous w.r.t. the natural topology on non-wellfounded terms with the identity as modulus of continuity. The number of repetition rules is locally related to the number of β-reductions necessary to reach the normal form (as represented by the Böhm tree) and the number of applications appearing in this normal form.

Introduction

Continuous normalization has been introduced by Mints [Min78,KMS75] in order to separate cut-elimination for semiformal systems from their ordinal analysis. By introducing a logical rule of *repetition* $(\mathcal{R})\ \frac{\Gamma \vdash A}{\Gamma \vdash A}$, it is possible to describe manipulations of infinitary derivations in a finitary manner, turning the cut-elimination operator into a primitive recursive function (see [Buc91] for a concise exposition). As Mints observed, this cut-elimination operator can also be applied to non-wellfounded derivations, resulting in a continuous function on derivation trees. More recently, Schwichtenberg [Sch98] has transferred these ideas from sequent calculi to natural deduction systems and a λ-calculus with an infinitary branching rule, building on [Buc91] and previous work by Ruckert [Ruc85].

$\mathcal{R}$ = "please wait". The central idea of continuous normalization is well-known to every user of modern computer operation systems: Whenever the result of a computation cannot immediately be displayed, a procedure should at least give regular life signs (like "please wait...") to be considered productive. In the terminology of coalgebra, in particular Coquand's conception of guarded recursion [Coq94], a partial function computing non-wellfounded objects can always be rendered total by adding a void constructor $\mathcal{R}$ (for repetition) in the

* Supported by the "Graduiertenkolleg Logik in der Informatik" of the Deutsche Forschungsgemeinschaft

clauses where productivity cannot be guaranteed. This corresponds to the above mentioned repetition rule.

Normalization for diverging terms. In our example of a normalization function for λ-terms, this amounts to returning $\mathcal{R}$, whenever the head constructor of the normal form cannot immediately be read off from the argument. So instead of non-wellfounded normal forms (Böhm trees), normal forms in an extended (co)grammar are computed. While for instance the head constructor of the normal form of λxr must be λ, the head constructor of an application rs depends on whether r is an abstraction, a variable, or again an application. The continuous normalization function therefore outputs $\mathcal{R}$, before further analysing r to find out whether s will be used for a substitution (if r were an abstraction) or has to be processed next (if r is a variable). As a consequence, the result of the diverging term $(\lambda x.xx)\lambda x.xx$ is an infinite sequence of repetition rules.

Coinductive λ-calculus. According to Mints' observation, the so obtained normalization procedure can be used to normalize non-wellfounded derivations, or in our case, terms of the coinductive λ-calculus Λ^{co} (see, e.g., [KKSdV97,Joa01a]). One of the interesting features of such a calculus is the possibility to directly define a fixpoint $Y_r^{\mathrm{co}} := rY_r^{\mathrm{co}}$.

Explaining $\mathcal{R}$. It has to be ensured that not too many $\mathcal{R}$ are produced in order to retain correctness of the normalization function for those terms that actually have a finitary normal form. Our analysis will reveal that every $\mathcal{R}$ corresponds to either a β-reduction or to an application in the Böhm tree of the term. To make this precise, another constructor β is introduced, which justifies a previous $\mathcal{R}$ in the normal form and stands for one reduction step in the normalization process. For example, normalization of the fixpoint combinator $Y := \lambda f.(\lambda x.f(xx))\lambda x.f(xx)$ applied to $K := \lambda y\lambda xy$ results in

$$(YK)^\beta = \mathcal{R}\beta\mathcal{R}\beta\mathcal{R}\beta\lambda x.\ \mathcal{R}\beta\mathcal{R}\beta\lambda x.\ \mathcal{R}\beta\mathcal{R}\beta\lambda x \dots$$

while for the fixpoint combinator $\Theta := (\lambda x, f.f(xxf))\lambda x, f.f(xxf)$ we obtain

$$(\Theta K)^\beta = \mathcal{R}\mathcal{R}\beta\beta\mathcal{R}\beta\lambda x.\ \mathcal{R}\mathcal{R}\beta\beta\mathcal{R}\beta\lambda x.\ \mathcal{R}\mathcal{R}\beta\beta\mathcal{R}\beta\lambda x \dots$$

So, apart from computing the non-wellfounded normal form $\lambda x\lambda x\lambda x \dots$,[1] the result yields insight into the normalization behaviour of its argument and thus permits a more detailed analysis: It shows that unfolding ΘK requires 3 reductions

$$\begin{aligned}\Theta K &= (\lambda x, f.f(xxf))\lambda x, f.f(xxf))K \\ &\to (\lambda f.f(\Theta f))K \\ &\to K(\Theta K) \\ &\to \lambda x.\Theta K\end{aligned}$$

[1] Note that both terms have undefined Böhm trees

to obtain the next λ, while $Y_K := (\lambda x.K(xx))\lambda x.K(xx)$ only needs two:

$$(\lambda x.K(xx))\lambda x.K(xx) \to KY_K \to \lambda x Y_K .$$

Not very surprising, the directly defined fixpoint Y_K^{co} is the most efficient: $Y_K^{\mathrm{co}} = KY_K^{\mathrm{co}} \to \lambda x Y_K^{\mathrm{co}}$. In fact,

$$\begin{array}{rl} \Theta^\beta = & \mathcal{R}\beta\lambda x.\mathcal{R}(x(\mathcal{R}\mathcal{R}\beta\beta\mathcal{R}(x(\mathcal{R}\mathcal{R}\beta\beta\mathcal{R}(x(\ldots \qquad \text{while} \\ Y^\beta = & \lambda x.\mathcal{R}\beta\mathcal{R}(x(\quad \mathcal{R}\beta \;\; \mathcal{R}(x(\quad \mathcal{R}\beta \;\; \mathcal{R}(x(\ldots \\ (Y_x^{\mathrm{co}})^\beta = & \qquad\quad \mathcal{R}(x(\qquad\quad \mathcal{R}(x(\qquad\quad \mathcal{R}(x(\ldots \end{array}$$

Analysis. Compared to Schwichtenberg's presentation of continuous normalization for the λ-calculus, the algorithm is simpler and therefore better suited for a precise analysis.

- The function is proved continuous w.r.t. the natural topology on non-well-founded terms that arises from the notion of equality up to k observations. The modulus of continuity is the identity.
- Lower and upper bounds for the number of $\mathcal{R}$ and β are exhibited.
- Each β is shown to correspond exactly to one reduction in the leftmost-outermost normalization strategy.

Outline of the contents. Section 1 recalls the definition of the coinductive λ-calculus with de Bruijn indices and introduces the extension by $\mathcal{R}$ and β. Section 2 presents the normalization algorithm, proves the modulus of continuity and correctness for finite normal forms. Section 3 establishes a coinductive characterization of the set of normal forms which arise by continuous normalization. This is used in section 4 to make the above remarks on the number of $\mathcal{R}$ and β precise. The appendix contains an example implementation in `Haskell`.

Acknowledgements. We are grateful to Andreas Abel for helpful comments on previous drafts of this work. Wilfried Buchholz gave valuable remarks on the history of continuous normalization.

1 The Coinductive λ-Calculus with $\mathcal{R}$ and β

The coinductive λ-calculus Λ^{co} arises by a coinductive interpretation of the defining grammar of the usual λ-calculus. Since this construction includes infinitary λ-terms, a variable's property of being new w.r.t. a given term is no longer decidable. Terms like r_0 with $r_n := x_n r_{n+1}$ (where $x_0, x_1, x_2, \ldots$ is an enumeration of all variables) may even contain all variables. It is thus reasonable to retreat into a de Bruijn discipline [Bru72] in handling free and bound variables in order to keep constructions like substitution primitive recursive.

1.1. Terms. Terms of the inductive and coinductive λ-calculus and their extensions by $\mathcal{R}$ and β are given by[2]

$$
\begin{array}{llllllll}
\Lambda & \ni r,s ::= & k & \mid rs & \mid \lambda r & & & \\
\Lambda^{\mathrm{co}} & \ni r,s ::=^{\mathrm{co}} & k & \mid rs & \mid \lambda r & & & \\
\Lambda_{\mathcal{R}} & \ni r,s ::= & k & \mid rs & \mid \lambda r & \mid \mathcal{R}r & \mid \beta r & \\
\Lambda^{\mathrm{co}}_{\mathcal{R}} & \ni r,s ::=^{\mathrm{co}} & k & \mid rs & \mid \lambda r & \mid \mathcal{R}r & \mid \beta r &
\end{array}
$$

Notation. x, y, k, l, m, n range over natural numbers. Obviously $\Lambda^{\mathrm{co}}_{\mathcal{R}} \supset \Lambda^{\mathrm{co}} \supset \Lambda \subset \Lambda_{\mathcal{R}} \subset \Lambda^{\mathrm{co}}_{\mathcal{R}}$. The notorious dot notation is applied as follows: A dot stands for a pair of parentheses that open at the dot and close as far right as syntactically possible. For instance $\lambda\lambda\lambda.2\,0.1\,0$ stands for $\lambda\lambda\lambda((2\,0)(1\,0))$, i.e., the combinator S. $\vec{r}^{\,n}$ (the superscript n will be omitted whenever reasonable) denotes a possibly empty list of terms $r_1, \ldots, r_n$. ε stands for the empty list. A comma is used for pre- and postfixing terms as well as appending lists. $\vec{0}^{\,n}$ stands for the list $0, \ldots, 0$ (n zeros).

Examples. By the guarded[3] recursive definition $Y^{\mathrm{co}}_r := rY^{\mathrm{co}}_r$ a direct implementation of a fixpoint for any term r is admissible in Λ^{co}. Less reasonable terms are $r := rr$ or $r := \lambda r$. In $\Lambda^{\mathrm{co}}_{\mathcal{R}}$ we may define

$$\bot_n ::=^{\mathrm{co}} \mathcal{R}\bot_{n+1} \mid \beta\bot_{n-1} \mid \lambda\bot_n.$$

In section 3 we will see that each $\bot_0$ arises exactly as a normal form of a term with undefined Böhm tree.

1.2. Observational equality. Define the equivalence relation $\simeq_k$ on $\Lambda^{\mathrm{co}}_{\mathcal{R}}$ inductively by

$$\frac{}{r \simeq_0 r'} \qquad \frac{}{x \simeq_l x} \qquad \frac{r, s \simeq_k r', s'}{rs \simeq_{k+1} r's'} \qquad \frac{r \simeq_k r'}{\lambda r, \mathcal{R}r, \beta r \simeq_{k+1} \lambda r', \mathcal{R}r', \beta r'}$$

Here we used the abbreviation $\vec{r}^{\,n} \simeq_k \vec{s}^{\,n} \;:\Leftrightarrow\; r_1 \simeq_k s_1 \;\wedge\; \ldots \;\wedge r_n \simeq_k s_n$.

Remarks. Obviously $r \simeq_{k+n} r'$ implies $r \simeq_k r'$ (weakening). $\simeq_k$-equivalence classes define the open sets of a topology on terms in coinductive calculi which is also induced by the metric $\mathrm{d}(r,s) := \frac{1}{1+\sup_k r \simeq_k s}$.

Equality on non-wellfounded terms is given by the bisimulation

$$r = s \;:\Longleftrightarrow\; \forall k.r \simeq_k s.$$

[2] We write $::=^{\mathrm{co}}$ to signify that the grammar should be interpreted coinductively, i.e., instead of the least we choose the greatest fixpoint of the underlying Set-endofunctor. For instance, Λ^{co} is the set of — not necessarily wellfounded — unary and binary branching trees with leaves labeled by natural numbers.

[3] In the sense of [Coq94,Gim95].

1.3. Lifting. Define by guarded[4] recursion $(-)\!\uparrow_n : \Lambda^{\mathrm{co}}_{\mathcal{R}} \longrightarrow \Lambda^{\mathrm{co}}_{\mathcal{R}}$.

$$\begin{aligned} (rs)\!\uparrow_n &:= r\!\uparrow_n s\!\uparrow_n & (\lambda r)\!\uparrow_n &:= \lambda.r\!\uparrow_{n+1} \\ (\mathcal{R}r)\!\uparrow_n &:= \mathcal{R}.r\!\uparrow_n & (\beta r)\!\uparrow_n &:= \beta.r\!\uparrow_n \\ k\!\uparrow_n &:= \begin{cases} k & \text{if } k < n \\ k+1 & \text{otherwise} \end{cases} & r\!\uparrow &:= r\!\uparrow_0 \end{aligned}$$

Remark. Lifting is continuous: $r \simeq_k s$ implies $r\!\uparrow_l \simeq_k s\!\uparrow_l$.

1.4. Substitution. Define by guarded recursion $-[-] : \Lambda^{\mathrm{co}}_{\mathcal{R}} \times \Lambda^{\mathrm{co}}_{\mathcal{R}} \longrightarrow \Lambda^{\mathrm{co}}_{\mathcal{R}}$.

$$\begin{aligned} (rs)[t]_n &:= r[t]_n\, s[t]_n & (\lambda r)[t]_n &:= \lambda.r[t\!\uparrow]_{n+1} \\ (\mathcal{R}r)[t]_n &:= \mathcal{R}.r[t]_n & (\beta r)[t]_n &:= \beta.r[t]_n \\ k[t]_n &:= \begin{cases} k & \text{if } k < n \\ t & \text{if } k = n \\ k-1 & \text{otherwise,} \end{cases} & r[t] &:= r[t]_0 \end{aligned}$$

This notion of substitution is tailored for β-reduction, only. A general substitution for non-wellfounded term systems which satisfies the usual monadic laws can for instance be found in [FPT99]. For an adaptation to Λ^{co} see [Joa01b].

Proposition 1 (Continuity). $r, s \simeq_k r', s' \implies r[s]_l \simeq_k r'[s']_l$.

1.5. β-reduction in Λ. The reduction relation $\to$ is only needed and defined on the inductive calculus Λ. It is the compatible closure of elementary β-reduction $(\lambda r)s \mapsto r[s]$, i.e., defined inductively by

$$\frac{}{(\lambda r)s \to r[s]} \qquad \frac{r \to r'}{\lambda r, rs, sr \to \lambda r', r's, sr'}$$

(with pointwise reading of $\vec{r} \to \vec{s}$). $\to^*$ is the reflexive transitive closure of $\to$.

Remark. If used in Λ^{co}, the reduction $\to$ is no longer confluent: With $r := 0r$ and $s := ((\lambda 0)0)s$ we have

$$r \;{}^*\!\not\leftarrow\; s \leftarrow (\lambda r)((\lambda 0)0) \to (\lambda r)0 \to r \not\to^* s.$$

For a more precise analysis and positive confluence results see [Joa01a].

1.6. Normal forms. The set of *normal forms*

$$\mathrm{NF} \ni r, s ::=^{\mathrm{co}} x\vec{r} \mid \lambda r \mid \mathcal{R}r \mid \beta r$$

contains only terms without redexes. Note that $\mathrm{NF} \cap \Lambda$ is the usual set of normal forms of Λ. However, there are some normal terms in $\Lambda^{\mathrm{co}}_{\mathcal{R}}$ which are not captured by this cogrammar, such as $(\mathcal{R}0)0$.[5]

[4] To justify this definition, the framework of Coquand/Gimenez is sufficient. For the following definitions, we assume a more liberal metatheory such as [TT97].

[5] The complete cogrammar for *all* normal forms is $r, s ::=^{\mathrm{co}} x\vec{r} \mid \lambda r \mid (\mathcal{R}r)\vec{s} \mid (\beta r)\vec{s}$.

2 Continuous Normalization

This section defines the primitive recursive normalization function $()^\beta$. The result of r^β can be understood as the normal form of r, enriched by information on the reduction sequence that was used to reach it.

The definition of r^β takes recourse to an auxiliary function $r@\vec{s}$, which intuitively should compute the normal form of $r\vec{s}$.[6]

2.1. Definition. We define $r@\vec{s} \in \mathrm{NF}$ (with $r, \vec{s} \in \Lambda^{\mathrm{co}}_{\mathcal{R}}$) by guarded recursion, using the abbreviation $r^\beta := r@\varepsilon$.

$$
\begin{aligned}
(\lambda r)@(s,\vec{s}\,) &:= \beta.r[s]@\vec{s} \qquad & (\lambda r)@\varepsilon &:= \lambda r^\beta \\
x@\vec{s} &:= x\vec{s}^{\,\beta} & (\mathcal{R}r)@\vec{s} &:= \mathcal{R}.r@\vec{s} \\
(rs)@\vec{s} &:= \mathcal{R}.r@(s,\vec{s}\,) & (\beta r)@\vec{s} &:= \beta.r@\vec{s}
\end{aligned}
$$

The normalization function outputs $\mathcal{R}$ whenever it faces an application rs, because it cannot foresee what to do with the argument s. When it next encounters an abstraction $r = \lambda r'$, the s will be used for the substitution $r'[s]$, so the $\mathcal{R}$ is justified ex post. If on the other hand a variable $r = x$ should follow then the s will be further normalized to produce the normal form of xs. The $\mathcal{R}$ produced in the step $(rs)^\beta = \mathcal{R}.r@s$ thus accounts for the application in the normal form xs^β.

Example. $(SKK)^\beta = \mathcal{R}\mathcal{R}\beta\beta\lambda\mathcal{R}\mathcal{R}\beta\beta 0$ with $S := \lambda\lambda\lambda.2\,0.1\,0,\ K := \lambda\lambda 1$. Note that $SKK \to^2 \lambda.K0.K0 \to^2 \lambda 0$.

Remarks. It is easy to see that the β-constructor is not necessary to ensure well-definedness, but it guarantees that the modulus of continuity is the identity.

For $r \in \Lambda$, the last two clauses are not needed to compute r^β.

Finally, we note a simple property: $(r\vec{s}^{\,n})^\beta = \mathcal{R}^n.r@\vec{s}$.

2.2. Continuity. The following lemma establishes that $()^\beta$ is continuous with modulus of continuity $k \mapsto k$.

Lemma 2. $r \simeq_k r' \ \wedge\ \vec{s} \simeq_k \vec{s'} \implies r@\vec{s} \simeq_k r'@\vec{s'}$.

Proof. The claim implies in particular that $r^\beta \simeq_k r'^\beta$ for $r \simeq_k r'$. It is proved by induction on k. Of course, the case $k = 0$ is trivial. **Case** $x@\varepsilon$. Trivial. **Case** $x@(\vec{s}, s)$ with $\vec{s}, s \simeq_k \vec{s'}, s'$. By induction hypothesis $x\vec{s} \simeq_k x\vec{s'}$, so $x\vec{s}s \simeq_{k+1} x\vec{s'}s'$ and by weakening $x\vec{s}s \simeq_k x\vec{s'}s'$. **Case** $(\lambda r)@(s,\vec{s}\,)$ with $\lambda r \simeq_{k+1} \lambda r'$ and

[6] It should be noted that our main concern is the conituity of the normalization function and the additional constructors needed for this purpose. The idea of collecting arguments of an application in a list and hence having $r@\vec{s}$ as a main normalization device is suggested by the way we analyze terms and was, for example, also used by Berarducci and Böhm [BB01, p.27] in their non-continuous normalization algorithm.

$s, \vec{s} \simeq_{k+1} s', \vec{s'}$.

$$\begin{array}{rll} s, \vec{s} \simeq_k & s', \vec{s'} & \text{by weakening} \\ r[s] \simeq_k & r'[s'] & \text{by proposition 1.4} \\ r[s]@\vec{s} \simeq_k & r'[s']@\vec{s'} & \text{by induction hypothesis} \\ \beta.r[s]@\vec{s} \simeq_{k+1} & \beta.r'[s']@\vec{s'} & \\ (\lambda r)@(s, \vec{s}) \simeq_{k+1} & (\lambda r')@(s', \vec{s'}). & \end{array}$$

Case $(rs)@\vec{s}$ with $\vec{s} \simeq_{k+1} \vec{s'}$ and $rs \simeq_{k+1} r's'$, i.e., $r, s \simeq_k r', s'$. By weakening $\vec{s} \simeq_k \vec{s'}$, so the induction hypothesis yields $r@(s, \vec{s}) \simeq_k r'@(s'\vec{s'})$. It follows that

$$r@(s, \vec{s}) = \mathcal{R}.r@(s, \vec{s}) \simeq_{k+1} \mathcal{R}.r'@(s', \vec{s'}) = r'@(s', \vec{s'}).$$

All other cases are simple applications of the induction hypothesis. □

2.3. Normalization. The next goal is to verify that the result computed by r^β is actually the normal form of r, if it is finite. Since r^β might contain some $\mathcal{R}$ and β, we have to eliminate them to prove this correctness property.

Definition. For $r \in \Lambda_\mathcal{R}$ define recursively $r^* \in \Lambda$.

$$x^* := x, \qquad (rs)^* := r^* s^*, \qquad (\lambda r)^* := \lambda r^*, \qquad (\mathcal{R}r)^* := (\beta r)^* := r^*.$$

Proposition 3. $r \in \Lambda \ \wedge \ r^\beta \in \Lambda_\mathcal{R} \implies r \to^* r^{\beta *}$.

Proof. Induction on *the size* of r^β. Cases on r. **Case** $x\vec{r}$. We compute $(x\vec{r})^\beta = \mathcal{R}^n.x\vec{r}^{\,\beta}$. Thus

$$\begin{array}{rll} x\vec{r}^{\,n} \to^* & x\vec{r}^{\,\beta *} & \text{by IH} \\ = & (\mathcal{R}^n.x\vec{r}^{\,\beta})^* & \\ = & (x\vec{r})^{\beta *}. & \end{array}$$

Case $(\lambda r)s\vec{s}^{\,n}$. Note $((\lambda r)s\vec{s}^{\,n})^\beta = \mathcal{R}^n.(\lambda r)@(s, \vec{s}) = \mathcal{R}^{n+1}.r[s]@\vec{s}$. Thus

$$\begin{array}{rll} (\lambda r)s\vec{s}^{\,n} \to & r[s]\vec{s} & \\ \to^* & (r[s]@\vec{s})^* & \text{by IH} \\ = & (\mathcal{R}^{n+1}\beta.r[s]@\vec{s})^* & \\ = & ((\lambda r)s\vec{s})^{\beta *}. & \end{array}$$

The case λr is a simple application of the induction hypothesis. □

3 Continuous Normal Forms

NF still contains many terms that are not in the image of $()^\beta$, e.g., $\beta 7$. The objective for this section is to find a precise characterization of the set CNF $:= \{r \in \Lambda^{\mathrm{co}}_\mathcal{R} \mid \exists s \in \Lambda^{\mathrm{co}}.s^\beta = r\}$ of terms that arise as normal forms under the process of continuous normalization.

3.1. Well-formedness. Intuitively, a term is well-formed, if each $\mathcal{R}$ is justified either by a β or an application. More precisely, we define $\vec{s} \vdash r$, where each of the $\vec{s}$ is accounted for by a $\mathcal{R}$. The coinductive rules for $\vec{s} \vdash r$ are

$$(\mathcal{R}_s)\ \frac{s, \vec{s} \vdash r}{\vec{s} \vdash \mathcal{R}r} \qquad (v)\ \frac{\vdash \vec{r}}{\vec{r} \vdash x\vec{r}} \qquad (\lambda)\ \frac{\vdash r}{\vdash \lambda r} \qquad (\beta)\ \frac{\vec{s} \vdash r}{s, \vec{s} \vdash \beta r}$$

r is *well-formed* if $\vdash r$. In this case obviously $r \in \mathrm{NF}$.

Notation. In $\vec{s} \vdash r$ the list $\vec{s}$ is named the *context*. The terms $\vec{r}$ in rule (v) are called the *Eigenterms* of this rule.

Example. $\vec{0}^n \vdash \bot_n$.

Lemma 4. *$\vec{s} \vdash r@\vec{s}$.*

Proof. Guarded induction along the definition of @.[7]

Case $x@\vec{s}$. By induction hypothesis $\vdash \vec{s}^{\beta}$. Thus $\vec{s} \vdash x@\vec{s}$ by (v).

Case $(\lambda r)@(s, \vec{s})$. By induction hypothesis $\vec{s} \vdash r[s]@\vec{s}$, so by rule (β): $s, \vec{s} \vdash \beta.r[s]@\vec{s} = (\lambda r)@(s, \vec{s})$.

Case $(rs)@\vec{s}$. By induction hypothesis: $s, \vec{s} \vdash r@(s, \vec{s})$, so by rule $(\mathcal{R}_s)$: $\vec{s} \vdash \mathcal{R}.r@(s, \vec{s}) = (rs)@\vec{s}$.

Case $(\lambda r)@\varepsilon$. By induction hypothesis: $\vdash r^{\beta}$, so by (λ): $\vdash \lambda r^{\beta} = (\lambda r)@\varepsilon$. □

Thus every CNF term is well-formed. For the converse implication a further analysis of derivations of $\vec{s} \vdash r$ is necessary. In particular, each $\mathcal{R}$ should be attributed to either a β or an application. The corresponding notion is that of

3.2. Well-explained terms. Define coinductively

$$(\mathcal{R}_s)\ \frac{s, \vec{s} \vDash_0 r \qquad \vDash_0 s}{\vec{s} \vDash_0 \mathcal{R}r} \qquad (v)\ \frac{}{\vec{r} \vDash_0 x\vec{r}} \qquad (\lambda)\ \frac{\vDash_0 r}{\vDash_0 \lambda r}$$

$$(\mathcal{R})\ \frac{\vec{s} \vDash_{n+1} r}{\vec{s} \vDash_n \mathcal{R}r} \qquad (\beta)\ \frac{\vec{s} \vDash_n r}{\vec{s} \vDash_{n+1} \beta r}$$

r is *well-explained* if $\vDash_0 r$.

In a well-explained term we have a precise explanation for each $\mathcal{R}$: either it is introduced by $(\mathcal{R})$, in which case it will be canceled later by a β; or it is introduced by $(\mathcal{R}_s)$ and the well-explained term s will occur in an application.

[7] This means that the non-wellfounded derivation of $\vec{s} \vdash r@\vec{s}$ is constructed coiteratively by guarded recursion.

Example. A derivation of $Y^\beta = \lambda\phi$ with $\phi := \mathcal{R}\beta\mathcal{R}.0\phi$ is shown on the right.

$$
{\scriptstyle(\lambda)}\dfrac{{\scriptstyle(\mathcal{R})}\dfrac{{\scriptstyle(\beta)}\dfrac{{\scriptstyle(\mathcal{R}_\phi)}\dfrac{{\scriptstyle(v)}\dfrac{}{\phi \vDash_0 0\phi} \qquad {\scriptstyle(\mathcal{R})}\dfrac{{\scriptstyle(\beta)}\dfrac{\vdots}{\vDash_1 \beta\mathcal{R}.0\phi}}{\vDash_0 \phi}}{\vDash_0 \mathcal{R}.0\phi}}{\vDash_1 \beta\mathcal{R}.0\phi}}{\vDash_0 \phi}}{\vDash_0 \lambda\phi}
$$

Proposition 5. $\vdash \vec{s} \;\wedge\; \vec{s} \vDash_n r \implies \vec{0}^n, \vec{s} \vdash r$

Proof. Guarded induction on $\vec{r} \vDash_n r$.[8]

Case $(\mathcal{R}_s)$: $\vec{s} \vDash_0 \mathcal{R}r$ has been derived from $s, \vec{s} \vDash_0 r$ and $\vDash_0 s$. By induction hypothesis $\vdash s$, so, again by induction hypothesis $s, \vec{s} \vdash r$ and rule $(\mathcal{R}_s)$ yields $\vec{s} \vdash \mathcal{R}r$.

Case (v): $\vec{s} \vDash_0 x\vec{s}$. By assumption $\vdash \vec{s}$, so rule (v) proves $\vec{s} \vdash x\vec{s}$.

Case $(\mathcal{R})$: $\vec{s} \vDash_n \mathcal{R}r$ has been concluded from $\vec{s} \vDash_{n+1} r$. By induction hypothesis $0, \vec{0}^{\,n}, \vec{s} \vdash r$, so by rule $(\mathcal{R}_0)$ $\vec{0}, \vec{s} \vdash \mathcal{R}r$.

Case (β): $\vec{s} \vDash_{n+1} \beta r$ from $\vec{s} \vDash_n r$. By induction hypothesis $\vec{0}^{\,n}, \vec{s} \vdash r$, so rule (β) shows $0, \vec{0}, \vec{s} \vdash \beta r$.

Case (λ). Simple application of the induction hypothesis. □

Definition. We write $\vec{r} \,|\, \vec{s} \vdash r$ if $\vec{r}, \vec{s} \vdash r$ and $\vec{s}$ are Eigenterms of the derivation and $\vec{r}$ are not. We write $\vec{r} \,|\, \vec{s} \vdash^* r$ if all contexts of the derivation of $\vec{r}, \vec{s} \vdash r$ can be consistently split that way.

Theorem 6 (Clairvoyance).
If $\vec{r} \vdash r$ then there is a partition $\vec{r} = (\vec{s}, \vec{t})$ such that $\vec{s} \,|\, \vec{t} \vdash^* r$.

Proof. Using classical logic,[9] each term either is or is not an Eigenterm of the derivation. Noting that every term on the right of an Eigenterm in any context of the derivation has to be an Eigenterm as well (in fact, even of the same variable rule), we get the claim. □

Example. $\vec{0}^n \,|\, \vdash^* \bot_n$.

Lemma 7. $\vec{r}^{\,n} \,|\, \vec{s} \vdash^* r \implies \vec{s} \vDash_n r$.

Proof. Guarded induction on $\vec{r}^{\,n} \,|\, \vec{s} \vdash^* r$.

Case $(\mathcal{R}_s)$ with $\vec{s}^{\,n} \,|\, \vec{t} \vdash \mathcal{R}r$ from $s, \vec{s} \,|\, \vec{t} \vdash^* r$. By induction hypothesis $\vec{t} \vDash_{n+1} r$, so rule $(\mathcal{R})$ yields $\vec{t} \vDash_n \mathcal{R}r$.

Case $(\mathcal{R}_s)$ with $\,|\, \vec{s} \vdash^* \mathcal{R}r$ from $\,|\, s, \vec{s} \vdash^* r$. As s is an Eigenterm of a rule (v) which contains $\vdash^* s$ as a premise, we can apply the induction hypothesis to obtain $\vDash_0 s$. Also $s, \vec{s} \vDash_0 r$ by induction hypothesis, so rule $(\mathcal{R}_s)$ proves $\vec{s} \vDash_0 \mathcal{R}r$.

Case (β): $r, \vec{r}^{\,n} \,|\, \vec{s} \vdash^* \beta r$ from $\vec{r} \,|\, \vec{s} \vdash^* r$. By induction hypothesis $\vec{r} \vDash_n r$, so rule (β) yields $\vec{r} \vDash_{n+1} \beta r$.

The two remaining cases are simple. □

---

[8] See previous footnote. The formulation “guarded induction on ...” provides the additional information that for finite derivations of $\vDash_n$ a well-founded derivation of $\vdash$ is obtained.

[9] Note that a constructive proof of this theorem cannot exist, because its underlying algorithm would allow to decide the halting problem. Moreover, this is the only place in the article where non-constructive reasoning is invoked.

Example. $\vDash_n \bot_n$ using just $(\mathcal{R}), (\beta), (\lambda)$.

Corollary 8 (Equivalence). $\vdash r \iff \vDash_0 r$.

3.3. Completeness. With the equivalent reformulation of $\vdash$ by $\vDash_0$ at hand, we can now show that every well-formed term arises as a continuous normal form. The witness of this existence is computed by the function T.

Definition. Define a term $T(\vec{s} \vDash_n r) \in \Lambda^{\mathrm{co}}$ by guarded recursion on the derivation of $\vec{s} \vDash_n r$. We use the notation of the definition of $\vDash_n$.

$$\begin{array}{lll}
(\mathcal{R}_s) & T(\vec{s} \vDash_0 \mathcal{R}r) & := T(s, \vec{s} \vDash_0 r)T(\vDash_0 s) \\
(v) & T(\vec{r} \vDash_0 x\vec{r}) & := x \\
(\lambda) & T(\vDash_0 \lambda r) & := \lambda T(\vDash_0 r) \\
(\beta) & T(\vec{s} \vDash_{n+1} \beta r) & := \lambda.T(\vec{s} \vDash_n r){\uparrow} \\
(\mathcal{R}) & T(\vec{s} \vDash_n \mathcal{R}r) & := T(\vec{s} \vDash_{n+1} r)0
\end{array}$$

Example. For the derivation of $\vDash_0 Y^\beta$ in subsection 3.2 we obtain $T(\vDash_0 Y^\beta) = \lambda.\lambda(1.\lambda(2.\lambda(\ldots)2)1)0$

Proposition 9 (Completeness). $\vec{t}^{\,\beta} = \vec{s} \Longrightarrow T(\vec{s} \vDash_n r)@(\vec{0}^{\,n}, \vec{t}) = r$.

Corollary 10. $T(\vDash_0 r)^\beta = r$.

In particular, each $\bot_n$ is a normal form of a term with undefined Böhm tree.

Proof (of the Proposition). Guarded induction on $r \vDash_n \vec{s}$.[10] **Case** (v): $\vec{s} \vDash_0 x\vec{s}$. By the premise $\vec{t}^{\,\beta} = \vec{s}$, so $T(\vec{s} \vDash_0 x\vec{s})@\vec{t} = x\vec{t}^{\,\beta} = x\vec{s}$. **Case** $(\mathcal{R}_s)$: $\vec{s} \vDash_0 \mathcal{R}r$ from $s, \vec{s} \vDash_0 r$ and $\vDash_0 s$. By induction hypothesis $T(\vDash_0 s)^\beta = s$. Thus with $t := T(\vDash_0 s)$, by induction hypothesis, $T(s, \vec{s} \vDash_0 r)@(t, \vec{t}) = r$, hence

$$T(\vec{s} \vDash_0 \mathcal{R}r)@\vec{t} = (T(s, \vec{s} \vDash_0 r)t)@\vec{t} = \mathcal{R}.T(s, \vec{s} \vDash_0 r)@(t, \vec{t}) = \mathcal{R}r.$$

Case $(\mathcal{R})$: $\vec{s} \vDash_n \mathcal{R}r$ from $\vec{s} \vDash_{n+1} \mathcal{R}$. With $0^\beta = 0$ we get

$$\begin{aligned}
T(\vec{s} \vDash_n \mathcal{R}r)@(\vec{0}^{\,n}, \vec{t}) &= (T(\vec{s} \vDash_{n+1} r)0)@(\vec{0}^{\,n}, \vec{t}) \\
&= \mathcal{R}.T(\vec{s} \vDash_{n+1} r)@(\vec{0}^{\,n+1}, \vec{t}) \\
&= \mathcal{R}r & \text{by IH.}
\end{aligned}$$

Case (β): $\vec{s} \vDash_{n+1} \beta r$ from $\vec{s} \vDash_n r$.

$$\begin{aligned}
T(\vec{s} \vDash_{n+1} \beta r)@(0, \vec{0}^{\,n}, \vec{t}) &= (\lambda.T(\vec{s} \vDash_n r){\uparrow})@(0, \vec{0}^{\,n}, \vec{t}) \\
&= \beta.(T(\vec{s} \vDash_n r){\uparrow})[0]@(\vec{0}^{\,n}, \vec{t}) \\
&= \beta.T(\vec{s} \vDash_n r)@(\vec{0}^{\,n}, \vec{t}) \\
&= \beta r & \text{by IH.}
\end{aligned}$$

Case (λ): $\vDash_0 \lambda r$ from $\vDash_0 r$. Trivial. □

[10] More precisely, the asserted equality amounts to showing $\simeq_k$ for all k and this is done by induction on k. The formulation chosen here is hopefully more reminiscent of traditional inductive equality proofs.

4 Analysis

Using well-formedness we can now provide a precise analysis of each $\mathcal{R}$ and β in the normalization process.

4.1. Bounds for $\mathcal{R}$. The first goal is to show that each $\mathcal{R}$ produced during of continuous normalization corresponds to either a β-reduction or an application in the normal form. In order to prove this for possibly non-wellfounded normal forms, we need the concept of paths.

Definition (Paths). A *path* is a list of natural numbers, i.e., $\zeta ::= \varepsilon \mid k \cdot \zeta$. The set of paths of a term r is given inductively by

$$\frac{}{\varepsilon \in r} \qquad \frac{\zeta \in r_k}{k \cdot \zeta \in x\vec{r}} \qquad \frac{\zeta \in r}{k \cdot \zeta \in \lambda r} \qquad \frac{\zeta \in r}{k \cdot \zeta \in \mathcal{R} r} \qquad \frac{\zeta \in r}{k \cdot \zeta \in \beta r}$$

ζ *is complete in* r (written $\zeta \in_c r$) iff $\varepsilon \in r$ is used only for abstractions λs and variable eliminations $x\vec{r}$. In other words, at the end of a complete path there are no pending terms to be applied.

(Only) for valid paths $\zeta \in r$ we define the number $\mathcal{R}_\zeta r$ of $\mathcal{R}$s, the number $\beta_\zeta r$ of βs and the number $A_\zeta r$ of applications in the path by

$$\begin{array}{llllll}
\mathcal{R}_{k\cdot\zeta}\lambda r & := \mathcal{R}_\zeta r & \beta_{k\cdot\zeta}\lambda r & := \beta_\zeta r & A_{k\cdot\zeta}\lambda r & := A_\zeta r \\
\mathcal{R}_{k\cdot\zeta}\mathcal{R} r & := 1 + \mathcal{R}_\zeta r & \beta_{k\cdot\zeta}\mathcal{R} r & := \beta_\zeta r & A_{k\cdot\zeta}\mathcal{R} r & := A_\zeta r \\
\mathcal{R}_{k\cdot\zeta}\beta r & := \mathcal{R}_\zeta r & \beta_{k\cdot\zeta}\beta r & := 1 + \beta_\zeta r & A_{k\cdot\zeta}\mathcal{R} r & := A_\zeta r \\
\mathcal{R}_{k\cdot\zeta}(x\vec{r}) & := \mathcal{R}_\zeta r_k & \beta_{k\cdot\zeta}(x\vec{r}) & := \beta_\zeta r_k & A_{k\cdot\zeta}(x\vec{r}^{\,n}) & := n + A_\zeta r_k \\
\mathcal{R}_\varepsilon r & := 0 & \beta_\varepsilon r & := 0 & A_\varepsilon(x\vec{r}^{\,n}) & := n \\
 & & & & A_\varepsilon r & := 0 \text{ otherwise}
\end{array}$$

Lemma 11. *$\vec{r}^{\,n} \vdash s \implies n + \mathcal{R}_\zeta s \geq \beta_\zeta s + A_\zeta s$ with equality for $\zeta \in_c s$.*

Proof. Induction on $\zeta \in s$.
Case ε. If s is not of the form $x\vec{r}$ then the claim is trivial, because $n + \mathcal{R}_\varepsilon s = n \geq 0 = \beta_\varepsilon s + A_\varepsilon s$. For a variable elimination $\vec{r}^{\,n} \vdash x\vec{r}$ we compute

$$n + \mathcal{R}_\varepsilon(x\vec{r}) = n + 0 = n = \beta_\varepsilon(x\vec{r}) + A_\varepsilon(x\vec{r}).$$

Case $l \cdot \zeta$. Subcase $(\mathcal{R}_s)$: $\vec{r} \vdash \mathcal{R} r$ from $s, \vec{r} \vdash r$.

$$\begin{aligned}
n + \mathcal{R}_{l\cdot\zeta}\mathcal{R} r &= n + 1 + \mathcal{R}_\zeta r \\
&\geq \beta_\zeta r + A_\zeta r && \text{by IH} \\
&= \beta_{l\cdot\zeta}\mathcal{R} r + A_{l\cdot\zeta}\mathcal{R} r.
\end{aligned}$$

Subcase (v): $\vec{r}^{\,n} \vdash x\vec{r}$ from $\vdash \vec{r}$.

$$\begin{aligned}
n + \mathcal{R}_{l\cdot\zeta}(x\vec{r}) &= n + \mathcal{R}_\zeta r_l \\
&\geq n + \beta_\zeta r_l + A_\zeta r_l && \text{by IH} \\
&= \beta_{l\cdot\zeta}(x\vec{r}) + A_{l\cdot\zeta}(x\vec{r}^{\,n}).
\end{aligned}$$

Subcase (β): $r, \vec{r}^{\,n} \vdash \beta r$ from $\vec{r} \vdash r$.

$$\begin{aligned} n{+}1 + \mathcal{R}_{l\cdot\zeta}\beta r &= 1 + n + \mathcal{R}_\zeta r \\ &\geq 1 + \beta_\zeta r + A_\zeta r && \text{by IH} \\ &= \beta_{l\cdot\zeta}\beta r + A_{l\cdot\zeta}\beta r. \end{aligned}$$

Subcase (λ). Simple application of the induction hypothesis. □

As an instance of this lemma we obtain for well-formed s that the number of $\mathcal{R}$s on each complete path is precisely the number of βs plus the number of applications.

4.2. Bounds for β. Next we show that each β produced during continuous normalization corresponds to exactly one β-reduction step in the leftmost-outermost normalization strategy, as given inductively by

$$(v)\ \frac{\vec{r} \rightsquigarrow_{\vec{n}} \vec{s}}{x\vec{r} \rightsquigarrow_{\Sigma\vec{n}} x\vec{s}} \qquad (\lambda)\ \frac{r \rightsquigarrow_n s}{\lambda r \rightsquigarrow_n \lambda s} \qquad (\beta)\ \frac{r[s]\vec{s} \rightsquigarrow_n t}{(\lambda r)s\vec{s} \rightsquigarrow_{n+1} t} \qquad (r)\ \frac{}{r \rightsquigarrow_0 r}$$

($\vec{r} \rightsquigarrow_{\vec{n}} \vec{s}$ is read pointwise) The index counts the number of β-reduction steps performed.

Remark. By standardization, $r \to^* s \in \mathrm{NF} \cap \Lambda$ implies that there exists an n with $r \rightsquigarrow_n s$.

Definition. Let $\rhd_\mathcal{R}$ and $\rhd_\beta$ be the compatible closures of $\mathcal{R}r \rhd_\mathcal{R} r$ and $\beta r \rhd_\beta r$. $\rhd^n_\mathcal{R}$ stands for n steps of $\rhd_\mathcal{R}$. Let $\rhd^k_n$ contain reduction sequences with k $\rhd_\mathcal{R}$ and n $\rhd_\beta$-reductions (mixed ad libitum). Furthermore, we set $\rhd_n := \rhd^n_n$.

Lemma 12. $r \rightsquigarrow_n s \implies r^\beta \rhd_n s^\beta$.

Proof. Induction on $\rightsquigarrow_n$. **Case** $(\lambda r)s\vec{s}^{\,k} \rightsquigarrow_{n+1} t$.

$$\begin{aligned} ((\lambda r)s\vec{s})^\beta &= \mathcal{R}^k\mathcal{R}.(\lambda r)@(s,\vec{s}) \\ &= \mathcal{R}^k\mathcal{R}\beta.r[s]@\vec{s} \\ &\rhd_\mathcal{R} \mathcal{R}^k\beta.r[s]@\vec{s} \\ &\rhd_\beta \mathcal{R}^k.r[s]@\vec{s} \\ &= (r[s]\vec{s})^\beta \\ &\rhd_n t^\beta && \text{by IH.} \end{aligned}$$

Case $x\vec{r} \rightsquigarrow_{\Sigma\vec{n}} x\vec{s}$.

$$\begin{aligned} (x\vec{r}^{\,k})^\beta &= \mathcal{R}^k.x\vec{r}^{\,\beta} \\ &\rhd_{\Sigma\vec{n}} \mathcal{R}^k.x\vec{s}^{\,\beta} && \text{by IH} \\ &= (x\vec{s})^\beta. \end{aligned}$$

Cases $(\lambda), (r)$. Simple. □

4.3. Weakly normalizing terms. If a term r actually has a finite normal form, then we can slightly strengthen the claim of the last lemma.

Definition. For $r \in \Lambda \cap \mathrm{NF}$ we define recursively

$$|x\vec{r}^{\,n}| := n + \Sigma\,|\vec{r}\,|, \qquad |\lambda r| := |r|.$$

Proposition 13. $r \in \Lambda \cap \mathrm{NF} \implies r^\beta \rhd_{\mathcal{R}}^{|r|} r.$

Proof. Trivial by remark 2.1. □

Corollary 14. $r \rightsquigarrow_n s \in \mathrm{NF} \cap \Lambda \implies r^\beta \rhd_n^{n+|s|} s.$

Hence, for every weakly normalizing term r, the normal form is computed by r^β and we have precise information on the number of steps in the standard reduction sequence leading to it.

Conclusions

By adopting a coinductive viewpoint on non-wellfounded terms it has become possible to simplify the formulation of continuous normalization. In particular, it is not necessary to introduce explicit substitution operators, because substitution is itself an admissible function in the coinductive λ-calculus. Yet it would be interesting to make the connection to previous presentations precise by adding normalization rules for an explicit substitution operator.

An extension to infinitary term systems with term formers corresponding to the infinitary ω-rule along the lines of [Sch98] is conceivable. Previous work on confluence of non-wellfounded infinitely branching term systems with permutative reductions [Joa01a] suggests that such a study is feasible, although the details concerning continuous normalization have not been worked out yet.

References

[BB01] A. Beraducci and Corrado Böhm. General recursion on second order term algebras. In *Rewriting Techniques and Applications*, volume 2051 of *Lecuture notes in Computer Science*, pages 15–30. Springer, 2001.

[Bru72] N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[Buc91] Wilfried Buchholz. Notation systems for infinitary derivations. *Archive for Mathematical Logic*, 30:277–296, 1991.

[Coq94] Thierry Coquand. Infinite objects in type theory. In H. Barendregt and T. Nipkow, editors, *Proc. 1st Types 1993 (Nijmengen)*, volume 806 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 1994. Available from `http://www.cs.chalmers.se/~coquand/`.

[FPT99] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding (extended abstract). In *Proc. 14th LICS 1999 (Trento)*, pages 193–202. IEEE Computer Science Press, 1999. Available from `http://www.dcs.ed.ac.uk/home/gdp/publications/`.

[Gim95] Eduardo Gimenez. Codifying guarded definitions with recursive schemes. In J. Smith, B. Nordström, and P. Dybjer, editors, *Proc. Types'94 (Bastad)*, volume 996 of *Lecture Notes in Computer Science*, pages 35–59. Springer, 1995. Available from `http://pauillac.inria.fr/~gimenez/papers.html`.

[Joa01a] Felix Joachimski. Confluence of the coinductive lambda-calculus. Submitted to *Theoretical Computer Science*, available from `http://www.mathematik.uni-muenchen.de/~joachski`, September 2001.

[Joa01b] Felix Joachimski. *Reduction Properties of ΠIE-Systems*. PhD thesis, LMU München, 2001. Available from `http://www.mathematik.uni-muenchen.de/~joachski`.

[KKSdV97] Richard Kennaway, Jan-Willem Klop, Ronan Sleep, and Fer-Jan de Vries. Infinitary lambda calculus. *Theoretical Computer Science*, 175(1):93–125, 1997.
Available from `http://www.sys.uea.ac.uk/~jrk/`.

[KMS75] G. Kreisel, G.E. Mints, and S.G. Simpson. The use of abstract language in elementary metamathematics: Some pedagogic examples. In R. Parikh, editor, *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 38–131. Springer, 1975.

[Min78] Grigori E. Mints. Finite investigations of transfinite derivations. *Journal of Soviet Mathematics*, 10:548–596, 1978. Translated from: Zap. Nauchn. Semin. LOMI 49 (1975). Cited after Grigori Mints. *Selected papers in Proof Theory*. Studies in Proof Theory. Bibliopolis, 1992.

[Ruc85] Martin Ruckert. *Church–Rosser Theorem und Normalisierung für Termkalküle mit unendlichen Termen unter Einschluß permutativer Reduktionen*. PhD thesis, Mathematisches Institut der LMU München, 1985.

[Sch98] Helmut Schwichtenberg. Finite notations for infinite terms. *Annals of Pure and Applied Logic*, 94:201–222, 1998. Available from `http://www.mathematik.uni-muenchen.de/~schwicht`.

[TT97] Alastair Telford and David Turner. Ensuring Streams Flow. In Michael Johnson, editor, *Proc. 6^{th} AMAST 1997 (Sydney)*, volume 1349 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1997. Available from `http://www.cs.ukc.ac.uk/people/staff/ajt/`.

Appendix: Implementation in `Haskell`

```
data Term = Var Integer | App Term Term | Lam Term |
            Rep Term | Bet Term deriving Show

lift :: Term -> Integer -> Term
lift (Var k)   n = Var (k + if k < n then 0 else 1)
lift (App r s) n = App (lift r n) (lift s n)
lift (Lam r)   n = Lam (lift r (n + 1))
lift (Rep r)   n = Rep (lift r n)
lift (Bet r)   n = Bet (lift r n)

subst :: Term -> Term -> Integer -> Term
subst (Var k)    s n | k == n = s
subst (Var k)    s n = Var (k - if k < n then 0 else 1)
```

```
subst (App r r') s n = App (subst r s n) (subst r' s n)
subst (Lam r)    s n = Lam (subst r (lift s 0) (n + 1))
subst (Rep r)    s n = Rep (subst r s n)
subst (Bet r)    s n = Bet (subst r s n)

beta :: Term -> Term
beta r = app r []

app :: Term -> [Term] -> Term
app (Lam r) (s:l) = Bet (app (subst r s 0) l)
app (Lam r)  []   = Lam (beta r)
app (Rep r)   l   = Rep (app r l)
app (Bet r)   l   = Bet (app r l)
app (Var k)   l   = foldl App (Var k) (map beta l)
app (App r s) l   = Rep (app r (s:l))
```

Examples.

```
s = Lam (Lam (Lam (Var 2 `App` (Var 0) `App` (Var 1 `App` (Var 0)))))
k = Lam (Lam (Var 1))
y = Lam (Lam (Var 1 `App` (Var 0 `App` (Var 0))) `App`
        (Lam (Var 1 `App` (Var 0 `App` (Var 0)))))
yco r = r `App` (yco r)
theta = Lam (Lam (Var 0 `App` (Var 1 `App` (Var 1) `App` (Var 0)))) `App`
        Lam (Lam (Var 0 `App` (Var 1 `App` (Var 1) `App` (Var 0))))
church n = Lam (Lam (iterate (App (Var 1)) (Var 0) !! n))
```

Here are some test runs of the program:

```
Main> beta (s `App` k `App` k)
  Rep (Rep (Bet (Bet (Lam (Rep (Rep (Bet (Bet (Var 0)))))))))
Main> beta (yco k)
  Rep (Bet (Lam (Rep (Bet (Lam (Rep (Bet (Lam (Rep (Bet (Lam
               (Rep (Bet (Lam (Rep (Bet (Lam (Rep {Interrupted!}
Main> beta (y `App` k)
  Rep (Bet (Rep (Bet (Rep (Bet (Lam (Rep (Bet (Rep (Bet (Lam
          (Rep (Bet (Rep (Bet (Lam (Rep (Bet (Rep {Interrupted!}
Main> beta (theta `App` k)
  Rep (Rep (Bet (Bet (Rep (Bet
      (Lam (Rep (Rep (Bet (Bet (Rep (Bet (Lam
          (Rep (Rep (Bet (Bet (Rep (Bet (Lam {Interrupted!}
Main> beta (church 2 `App` (church 3))
  Rep (Bet (Lam (Rep (Bet (Lam (Rep (Rep (Bet (Bet (Rep (App (Var 1)
      (Rep (App (Var 1) (Rep (App (Var 1) (Rep (Rep (Bet (Bet (Rep
      (App (Var 1) (Rep (App (Var 1) (Rep (App (Var 1) (Rep (Rep
      (Bet (Bet (Rep (App (Var 1) (Rep (App (Var 1) (Rep (App (Var 1)
      (Var 0))) [..])
Main> beta (church 3 `App` (church 2))
  Rep (Bet (Lam (Rep (Bet (Lam (Rep (Rep (Bet (Bet (Rep (Rep (Bet
      (Bet (Rep (App (Var 1) (Rep (App (Var 1) (Rep (Rep (Bet (Bet (Rep
      (App (Var 1) (Rep (App (Var 1) (Rep (Rep (Bet (Bet (Rep (Rep (Bet
      (Bet (Rep (App (Var 1) (Rep (App (Var 1) (Rep (Rep (Bet (Bet (Rep
      (App (Var 1) (Rep (App (Var 1) (Var 0))) [..] )
```