Refining Reduction in the lambda calculus
Journal of Functional Programming 5(4), 1995[*]

Fairouz Kamareddine [†]
Department of Computing Science
17 Lilybank Gardens
University of Glasgow
Glasgow G12 8QQ, Scotland
email: fairouz@dcs.glasgow.ac.uk

and

Rob Nederpelt
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O.Box 513
5600 MB Eindhoven, the Netherlands
email: wsinrpn@win.tue.nl

October 23, 1996

[*]We are grateful for the discussions with Roel Bloo, Tijn Borghuis, Erik Poll and Phil Wadler and for the helpful remarks received from them. In particular, we are grateful to Phil Wadler who has recommended that we dispose of λ's and δ's as in $(x\delta)(\lambda_y)y$ and write $(x)[y]y$ instead. We are also grateful to Peter Peters for his help concerning Latex and e-mail which enabled us to keep exchanging drafts and corrections of the present paper. Last but not least, we are grateful to the anonymous referees for their useful comments.

[†]Kamareddine is grateful to the Department of Mathematics and Computing Science, Eindhoven University of Technology, for their financial support and hospitality from October 1991 to September 1992, and during several short periods since 1993.

Abstract

We introduce a λ-calculus notation which enables us to detect in a term, more β-redexes than in the usual notation. On this basis, we define an extended β-reduction which is yet a subrelation of conversion. The Church Rosser property holds for this extended reduction. Moreover, we show that we can transform generalised redexes into usual ones by a process called "term reshuffling".

Keywords: Item notation, Redexes, Church Rosser.

Contents

1 Introduction

In the λ-calculus as we know it, some redexes in a term may not be visible before other redexes have been contracted. For example, in $t \equiv ((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a$, only $(\lambda_y.\lambda_z.zd)c$, and $(\lambda_x.(\lambda_y.\lambda_z.zd)c)b$ are visible. Yet when reducing t to a normal form, a third redex must be contracted; namely $(\lambda_z.zd)a$. This third redex is not immediately visible in t (assume for the sake of argument that none of x, y, z occur free in any of a, b, c and d). To solve this problem we switch from the classical notation to what we call *item notation* where the argument occurs before the function and where parentheses are grouped in a novel way. In our item notation, t will be written as $(a)(b)[x](c)[y][z](d)z$ and we can provide t in this item notation with a bracketing structure $\{\{\ \}\{\ \}\}$ where $(-)$ and $[-]$ correspond to '{' and '}' respectively (ignoring $(d)z$). We extend the notion of a redex from being any opening bracket '{' next to a closing bracket '}', to being any pair of matching '{' and '}' which are separated by matching brackets. Figure 1 shows the possible redexes. That is, we see immediately that

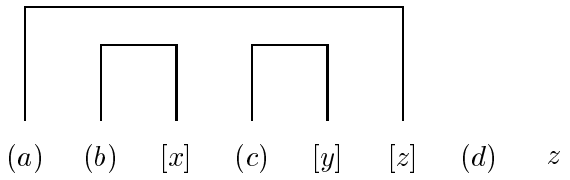


Figure 1: Redexes in item notation

the redexes in t originate from the couples $(b)[x]$, $(c)[y]$ and $(a)[z]$. This natural matching was not present in the classical notation of t. We call items of the form (a) and $[x]$, application and abstraction items respectively. With item notation, we shall refine reduction in two ways:

1. We generalise β-reduction so that any redex can be contracted and hence we can contract the redex based on $(a)[z]$ before we contract any of the redexes based on $(b)[x]$ and $(c)[y]$. That is, the β-rule changes from $(b)[v]a \rightarrow_\beta a[v := b]$ to $(b)\overline{s}[v]a \rightsquigarrow_\beta \overline{s}\{a[v := b]\}$ for $\overline{s}$ having a matching bracketing structure. I.e. $(b)\overline{s}[v]$ is a redex in our generalised sense. (Here, { and } are used for grouping purposes so that no confusion arises.) For example,

   $$\begin{array}{ll} (a)(b)[x](c)[y][z](d)z & \rightsquigarrow_\beta \\ (b)[x](c)[y]\{((d)z)[z := a]\} & \equiv \\ (b)[x](c)[y](d)a & \end{array}$$

   We show moreover, that the Church Rosser property holds for $\rightsquigarrow_\beta$

2. An alternative to the generalised notion of β-reduction can be obtained by keeping the old β-reduction and by *reshuffling* the term in hand. So we can reshuffle the term $(a)(b)[x](c)[y][z](d)z$ to $(b)[x](c)[y](a)[z](d)z$, in order to transform the bracketing structure $\{\{\ \}\{\ \}\}$ into $\{\ \}\{\ \}\{\ \}$, where all the redexes correspond to adjacent '{' and '}'. In other words, Figure 1 can be redrawn using term reshuffling in Figure 2. Such a reshuffling is more difficult to describe in classical notation. I.e. it is hard to say what exactly

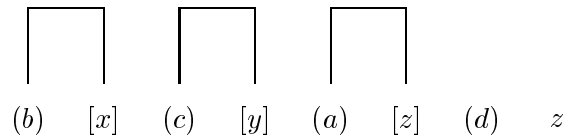


Figure 2: Term reshuffling in item notation

happened when $((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a$, is reshuffled to $(\lambda_x.(\lambda_y.(\lambda_z.zd)a)c)b$. This is another attractive feature of our item notation which we shall also describe in this paper (using TS) showing its correctness and well-behavedness. In particular, we show that for any term a, in $TS(a)$ all the application items occur next to their matching abstraction items. We show moreover, that if $a \rightsquigarrow_\beta b$ then $(\exists c)[(TS(a) \rightarrow_\beta c) \wedge TS(c) \equiv TS(b)]$.

The assumption above that none of x, y, z occur free in any of a, b, c, d was only for the sake of clarity in our particular example. We are not of course restricted to terms with such trivial substitutions. As for variable renaming which results from reductions and substitution, we decide not to let it blind us with details to the point that the argument of the paper becomes unclear. For this reason, we identify terms that differ only in the name of bound variables (i.e. we take terms modulo α-conversion). For example $[x]x \equiv [y]y$. Furthermore, we assume the Barendregt variable convention which is formally stated as follows:

Definition 1.1 *(BC Barendregt's Convention)*
Names of bound variables will always be chosen such that they differ from the free ones in a term. Hence, we will not have $(v)[v]v$, but $(v)[v']v'$ instead. We extend BC to meta-terms *like $\{[v']a\}[v := b]$. This avoids the danger of clash of variables, since we assume that no free variable in b occurs bound in $[v']a$. Moreover, the extended BC implies that $v \not\equiv v'$.*

1.1 The item notation and visible redexes

We shall devise a novel notation in this paper where the order in an application is inverted and where the parentheses are grouped differently than those of the classical notation. So that, if $\mathcal{I}$ translates classical terms into our notation, **item notation**, then $\mathcal{I}(a\,b)$ is written as $(\mathcal{I}(b))\mathcal{I}(a)$ and $\mathcal{I}(\lambda_v.a)$ is written as $[v]\mathcal{I}(a)$. Both (a) and $[v]$ are called **items**.

Example 1.2 $\mathcal{I}((\lambda_{xy}.xy)z) \equiv (z)[x][y](y)x$. The items are (z), $[x]$, $[y]$ and (y).

Note how the items (z) and $[x]$ occur next to each other showing the β-redex based on applying $\lambda_x.\texttt{body}$ to z. In the classical calculus, the λ_x and z are separated by `body` which may be very long. With our item notation, classical redexes and β-reduction take the following form:

Definition 1.3 *(Classical redexes and β-reduction in item notation)*
In the item notation of the λ-calculus, a classical redex is of the form $(b)[v]a$. We call the pair $(b)[v]$, a **reducible segment**. *Moreover, one-step β-reduction $\rightarrow_\beta$, is the least compatible relation generated out of the classical β-reduction axiom: $(b)[v]a \rightarrow_\beta a[v := b]$. Many step β-reduction $\twoheadrightarrow_\beta$ is the reflexive transitive closure of $\rightarrow_\beta$.*

Example 1.4 In the classical term $t \equiv ((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a$, we have the following redexes (the fact that neither y nor x appear as free variables in their respective scopes does not matter here; this is just to keep the example simple and clear):

1. $(\lambda_y.\lambda_z.zd)c$
2. $(\lambda_x.(\lambda_y.\lambda_z.zd)c)b$

Written in item notation, t becomes $(a)(b)[x](c)[y][z](d)z$. Here, the two classical redexes correspond to $(-)[-]$-pairs as follows:

1. $(\lambda_y.\lambda_z.zd)c$ corresponds to $(c)[y]$. We ignore $[z](d)z$ as it is easily retrievable in item notation. It is the maximal subterm of t to the right of $[y]$.
2. $(\lambda_x.(\lambda_y.\lambda_z.zd)c)b$ corresponds to $(b)[x]$. Again $(c)[y][z](d)z$ is ignored for the same reason as above.

There is however a third redex which is not visible in the classical term. Namely, $(\lambda_z.zd)a$. Such a redex will only be visible after we have contracted the above two redexes (we will not discuss the order here). In fact, assume we contract the second redex in the first step, and the first redex in the second step. I.e.

Classical Notation		*Item Notation*	
$(\underline{(\lambda_x.(\lambda_y.\lambda_z.zd)c)b})a$	$\to_\beta$	$(a)\underline{(b)[x]}(c)[y][z](d)z$	$\to_\beta$
$(\underline{(\lambda_y.\lambda_z.zd)c})a$	$\to_\beta$	$(a)\underline{(c)[y]}[z](d)z$	$\to_\beta$
$\underline{(\lambda_z.zd)a}$	$\to_\beta ad$	$\underline{(a)[z]}(d)z$	$\to_\beta (d)a$

Now, even though all these redexes (i.e. the first, second and third) are *needed* in order to get the normal form of t, only the first two were visible in the classical term at first sight. The third could only be seen once we have contracted the first two reductions. In item notation, the third redex $(\lambda_z.zd)a$ corresponds to $(a)[z]$ but the ( )-item and the [ ]-item are separated by the segment $(b)[x](c)[y]$. By extending the notion of a redex and of β-reduction, we can make this redex visible and we can contract it before the other redexes.

The idea is simple; we generalise the notion of a reducible segment $(b)[v]$ to a **reducible couple** being an item (b) and an item $[v]$ separated by a segment $\overline{s}$ which is a **well-balanced segment**. A well-balanced segment is a sequence of ( )- and [ ]-items which has the same structure as a matching composite of opening and closing brackets, each ( )-item corresponding to an opening bracket and each [ ]-item corresponding to a closing bracket.

1.2 The system Λ

We construct the system Λ where a term is either a variable or is of the form $s_1 s_2 \cdots s_n v$ for variable v and *items* s_i, for $1 \leq i \leq n$. An item is defined either as $[v']$ for variable v' or as (a) for a being a term. The following definitions formalise our system Λ.

Definition 1.5 *(Terms in item notation Λ)*

- *We take $\mathcal{V} = \{x, y, z, \ldots\}$ to be the set of variables and let $v, v', v'', v_1, v_2, \ldots$ range over $\mathcal{V}$.*

- *We write terms in item notation as:* $\Lambda ::= \mathcal{V} \mid (\Lambda)\Lambda \mid [\mathcal{V}]\Lambda$. *We use* $a, b, c, d, e, t, \ldots$ *to range over terms in item notation.*

Definition 1.6 *((main) items, (main, reducible) segment, body, weight)*

- *If* a *is a* Λ*-term and* $v \in \mathcal{V}$ *then* (a) *and* $[v]$ *are (application resp. abstraction)* **items**. *We use* $s, s_1, s_i, \cdots$ *as meta-variables for items.*

- *A concatenation of zero or more items is a* **segment**. *In [de Bruijn 93] an item is called a* wagon *and a segment is called a* train. *We use* $\overline{s}, \overline{s}_1, \overline{s}, \ldots$ *as meta-variables for segments.*

- *Each term* a *is the concatenation of zero or more items and a variable:* $a \equiv s_1 s_2 \ldots s_n v$. *These items* $s_1, s_2, \cdots, s_n$ *are called the* **main items** *of* a. *We call the segment* $s_1 s_2 \cdots s_n$, $\mathtt{body}(a)$. *Also, for later use, we define* $\mathtt{body}((a)) \equiv a$ *and* $\mathtt{body}([v]) \equiv \emptyset$.

- *Analogously, a segment* $\overline{s}$ *is a concatenation of zero or more items* $\overline{s} \equiv s_1 s_2 \cdots s_n$. *These items* $s_1, s_2, \cdots, s_n$ *are called the* **main items**, *this time of* $\overline{s}$.

- *A concatenation of main items is a* **main segment**.

- *An important case of a segment is that of a* **reducible segment**, *being an application item immediately followed by an abstraction item.*

- *The* **weight** *of a* Λ*-segment* $\overline{s}$, $\mathtt{weight}(\overline{s})$, *is the number of main items that compose the segment. The weight of a* Λ*-term* a *is the weight of* $\mathtt{body}(a)$.

Example 1.7 Let the Λ-term a be defined as $[x]((x)[y]y)[z]z$ and let the segment $\overline{s}$ be $[x]((x)[y]y)[z]$. Then the main items of both a and $\overline{s}$ are $[x]$, $((x)[y]y)$ and $[z]$, being an abstraction, an application and an abstraction item respectively.

With our BC of Definition 1.1, we define substitution as follows:

Definition 1.8 *(Substitution in* Λ *with* BC*)*
If a, b *are* Λ*-terms and* $v \in \mathcal{V}$ *and if* BC *is assumed, then we define the result of substituting* b *for all the free occurrences of* v *in* a *as follows:*

$$a[v := b] =_{df} \begin{cases} b & \text{if } a \equiv v \\ v' & \text{if } a \equiv v' \not\equiv v \\ (c[v := b])\{d[v := b]\} & \text{if } a \equiv (c)d \\ [v']c[v := b] & \text{if } a \equiv [v']c \end{cases}$$

With this implicit substitution, β-reduction is given in Definition 1.3. Furthermore, in order to avoid confusion, we sometimes group terms using { and }. For example, in $\overline{s}\{\overline{s'}a[v := b]\}$, $\overline{s'}a[v := b]$ is grouped between { and } so that one understands that substitution takes place in $\overline{s'}a$ and not in $\overline{s}$. It is obvious why we could not use ( ) instead of { }. It is to be noted moreover that we do not face this problem if we use (λ_v) instead of $[v]$ and $(a\delta)$ instead of (a).

2 Generalising redexes and β-reduction

As we have argued above, a term can contain an application item (a) and an abstraction item $[v]$ such that eventually the reduction based on the segment $(a)[v]$ should take place. This reduction however, can only so far take place when (a) and $[v]$ are not separated by other items or segments. This makes it difficult to use the λ-calculus as a basis for many applications which depend heavily on manipulating the order in which reduction and substitution take place in a term. Based on this observation, we shall in this section introduce a general β-reduction which enables the manipulation of the order of reduction and substitution. This general β-reduction will be an extension of the known β-reduction in that not only the reducible segments result in firing reductions, but a more general notion which we call reducible couples.

2.1 Extending redexes from segments to couples

Why should we in the term $(a)(b)[x](c)[y][z](d)z$ not allow that the reduction based on $(a)[z]$ gets fired? There is no reason why we should not carry out some reductions before other ones. We ask the reader to convince himself of the fact that priority for firing the redex $(a)[z]$ does not affect the final result. If we look moreover at this term, we find that what separates (a) and $[z]$ is a segment with a particular structure. The same holds for the segment $(b)[v]$ separating (a) and $[v']$ in $(a)(b)[v][v']$. These are the "well-balanced" structures as we discussed before and will define below. Basically, the idea is that when one desires to start a β-reduction on the basis of two items (application and abstraction) occurring in one segment, the *matching* of these items in question is the important thing, even when they are separated by other items. I.e., the relevant question is whether they *may* together become a reducible segment after a number of β-steps. This depends solely on the structure of the intermediate segment. If such an intermediate segment is well-balanced then the application item and the abstraction item match and β-reduction based on these two items may take place. Here is the definition of well-balanced segments:

Definition 2.1 *(well-balanced segments in Λ)*

- *The empty segment $\emptyset$ is a well-balanced segment.*
- *If $\overline{s}$ is a well-balanced segment, then $(a)\overline{s}[v]$ is a well-balanced segment.*
- *The concatenation of well-balanced segments is a well-balanced segment.*

A well-balanced segment has the same structure as a matching composite of opening and closing brackets, each application (or abstraction) item corresponding with an opening (resp. closing) bracket.

Example 2.2

1. $(c)[y]$ forms a well-balanced segment as it corresponds to $\{\ \}$.
2. $(b)[x]$ forms a well-balanced segment as it corresponds to $\{\ \}$.
3. $(b)[x](c)[y]$ forms a well-balanced segment as it corresponds to $\{\ \}\{\ \}$.
4. $(a)(b)[x](c)[y][z]$ forms a well-balanced segment as it corresponds to $\{\{\ \}\{\ \}\}$; the reducible couples in this segment are $(b)[x]$, $(c)[y]$ and $(a)[z]$.

Now we can easily define what matching reducible couples are. Namely, they are an application item and an abstraction item separated by a well-balanced segment. The abstraction item and the application item of the reducible couple are said to match and each of them is called a partner or a partnered item. The items in a segment that are not partnered are called bachelor items. The following definition summarizes all this:

Definition 2.3 *(match, reducible couple, partner, partnered item, bachelor item, bachelor segment)*
Let a be a Λ-term. Let $\overline{s} \equiv s_1 \cdots s_n$ be a segment occurring in a.

- *We say that the main items s_i and s_j* **match** *in $\overline{s}$, when $1 \leq i < j \leq n$, s_i is an application item, s_j is an abstraction item, and the sequence $s_{i+1} \cdots s_{j-1}$ forms a well-balanced segment.*
- *When s_i and s_j match, we call $s_i s_j$ a* **reducible couple**.
- *When s_i and s_j match, we call both s_i and s_j the* **partners** *in the reducible couple. We also say that s_i and s_j are* **partnered** *items in $\overline{s}$ (or in a).*
- *All the application (or abstraction) items s_k that are not partnered in $\overline{s}$, are called* **bachelor** *application (resp. abstraction)items in $\overline{s}$ (or in a).*
- *A segment $\overline{s}$ consisting of only bachelor items (in $\overline{s}$), is called a* **bachelor segment**.
- *The segment $s_{i_1} \cdots s_{i_m}$ consisting of all bachelor main application (or abstraction) items of $\overline{s}$ is called the* **bachelor application** *(or* **abstraction***)* **segment** *of $\overline{s}$*

Example 2.4 Let $\overline{s} \equiv [x][y](a)[z][x'](b)(c)(d)[y'][z'](e)$. Then:

- (a) matches with $[z]$, (d) matches with $[y']$ and (c) matches with $[z']$. The segments $(a)[z]$ and $(d)[y']$ are reducible segments (also reducible couples), and there is another reducible couple in $\overline{s}$, viz. the couple of (c) and $[z']$.
- (a), $[z]$, (c), (d), $[y']$ and $[z']$, are the partnered main items of $\overline{s}$. $[x]$, $[y]$, $[x']$, (b) and (e), are bachelor items.
- $[x][y]$ and $[x'](b)$ are bachelor segments. $(c)(d)[y']$ and $(c)(d)[y'][z']$ are non-bachelor segments, the latter also being a well-balanced segment.

Remark 2.5 Note that a reducible *segment* is a reducible couple and that the application and abstraction items in a reducible *couple* are separated by zero or more reducible couples.

2.2 Extending β-reduction and the Church Rosser theorem

Having argued above that β-reduction should not be restricted to the reducible segments but may take into account other candidates, we can extend our notion of β-reduction in this vein. That is to say, we may allow reducible *couples* to have the same "reduction rights" as reducible *segments*. That is, the β-reduction of Definition 1.3 changes to the following:

Definition 2.6 *(Extended redexes and general β-reduction $\rightsquigarrow_\beta$ in Λ)*
An extended redex if of the form $(b)\overline{s}[v]a$, where $\overline{s}$ is well-balanced. We call $(b)\overline{s}[v]a$ a reducible couple. Moreover, one-step general β-reduction $\rightsquigarrow_\beta$, is the least compatible relation generated out of the following axiom:

$$(general\ \ \beta) \qquad (b)\overline{s}[v]a \rightsquigarrow_\beta \overline{s}\{a[v := b]\} \qquad \text{if } \overline{s} \text{ is well-balanced}$$

Many step general β-reduction $\rightsquigarrow\!\!\!\!\rightarrow_\beta$ is the reflexive transitive closure of $\rightsquigarrow_\beta$.

Example 2.7 Take Example 1.4. As $(b)[x](c)[y]$ is a well-balanced segment, then $(a)[z]$ is a reducible couple and

$$\begin{array}{ll} t \equiv (a)(b)[x](c)[y][z](d)z & \rightsquigarrow_\beta \\ (b)[x](c)[y]\{((d)z)[z := a]\} & \equiv \\ (b)[x](c)[y](d)a & \end{array}$$

The *reducible couple* $(a)[z]$ also has a corresponding ("generalized") redex in the traditional notation, which will appear after two one-step β-reductions, leading to $(\lambda_z.zd)a$. With our generalised one-step β-reduction we could reduce $((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a$ to $(\lambda_x.(\lambda_y.ad)c)b$. This reduction is difficult to carry out in the classical λ-calculus. The item notation enables a new and important sort of reduction which has not yet been studied in relation to the standard λ-calculus up to date. We believe that this generalised reduction (introduced in [Nederpelt 73]) can only be obtained tidily in a system formulated using our item notation. In fact, one is to compare the bracketing structure of the classical term t of Example 1.4, with the bracketing structure of the corresponding term in item notation:

Example 2.8 The "bracketing structure" of the maximal main segment of t of Example 1.4, (that is, of $((\lambda_x.(\lambda_y.\lambda_z.--)c)b)a$), is compatible with '$\{_1\ \{_2\ \{_3\ \}_2\ \}_1\ \}_3$', where '$\{_i$' and '$\}_i$' match. In item notation however, t has the bracketing structure $\{\{\ \}\{\ \}\}$.

We strongly believe that it is the item notation which enables us to extend reduction smoothly beyond the existing $\rightarrow\!\!\!\!\rightarrow_\beta$. Because a well-balanced segment may be empty, the general β-reduction rule presented above is really an extension of the classical β-reduction rule.

Lemma 2.9 *Let a, b be Λ-terms. If $a \rightarrow_\beta b$ then $a \rightsquigarrow_\beta b$. Moreover, if $a \rightsquigarrow_\beta b$ comes from contracting a reducible segment then $a \rightarrow_\beta b$.*

Proof: *Obvious as a reducible segment is a reducible couple by Remark 2.5.* □

The proof of the Church Rosser theorem is simple. The idea is to show that if $a \rightsquigarrow\!\!\!\!\rightarrow_\beta b$ then $a =_\beta b$ (where $=_\beta$ is the least equivalence relation closed under $\rightarrow_\beta$ and $\rightsquigarrow\!\!\!\!\rightarrow_\beta$ the reflexive transitive closure of $\rightsquigarrow_\beta$) and to use the Church Rosser property for $=_\beta$.

Lemma 2.10 *If $a \rightsquigarrow_\beta b$ then $a =_\beta b$.*

Proof: *It suffices to consider the case $a \equiv \overline{s_1}(d)\overline{s}[v]c$ where the contracted redex is based on $(d)[v]$, $b \equiv \overline{s_1}\,\overline{s}\{c[v := d]\}$, and $\overline{s}$ is balanced (hence $\mathtt{weight}(\overline{s})$ is even). We shall prove the lemma by induction on $\mathtt{weight}(\overline{s})$.*

- *Case $\mathtt{weight}(\overline{s}) = 0$ then obvious as $\rightsquigarrow_\beta$ coincides with $\rightarrow_\beta$ in this case.*
- *Assume the property holds when $\mathtt{weight}(\overline{s}) = 2n$. Take $\overline{s}$ such that $\mathtt{weight}(\overline{s}) = 2n+2$. Now, $\overline{s} \equiv (e)\overline{s'}[v']\overline{s''}$ where $\overline{s'}$, $\overline{s''}$ are well-balanced.*

– *As* $\overline{s}\{c[v := d]\} \rightsquigarrow_\beta \overline{s'}\{\overline{s''}\{c[v := d]\}[v' := e]\}$, *we get by IH and compatibility that* $b =_\beta \overline{s_1}\,\overline{s'}\{\overline{s''}\{c[v := d]\}[v' := e]\} \equiv \overline{s_1}\,\overline{s'}\,\{\overline{s''}[v' := e]\}\{c[v := d][v' := e]\}$.

– *Moreover,* $a \equiv \overline{s_1}(d)(e)\overline{s'}[v']\overline{s''}[v]c \rightsquigarrow_\beta \overline{s_1}(d)\overline{s'}\{\overline{s''}[v]c[v' := e]\} \equiv^{BC} \overline{s_1}(d)\overline{s'}\{\overline{s''}[v' := e]\}[v]\{c[v' := e]\} \equiv f$. *Hence by IH,* $a =_\beta f$.

– *Now,* $f \rightsquigarrow_\beta \overline{s_1}\overline{s'}\{\overline{s''}[v' := e]\}\{c[v' := e][v := d]\}$. *But by BC,* $v, v' \notin FV(d) \cup FV(e)$. *Hence, by IH and substitution,* $f =_\beta \overline{s_1}\overline{s'}\{\overline{s''}[v' := e]\}\{c[v := d][v' := e]\}$.

Therefore $a =_\beta b$. □

Corollary 2.11 *If* $a \twoheadrightsquigarrow_\beta b$ *then* $a =_\beta b$. □

Theorem 2.12 *The general* β*-reduction is Church-Rosser. I.e. If* $a \twoheadrightsquigarrow_\beta b$ *and* $a \twoheadrightsquigarrow_\beta c$, *then there exists* d *such that* $b \twoheadrightsquigarrow_\beta d$ *and* $c \twoheadrightsquigarrow_\beta d$.

Proof: *As* $a \twoheadrightsquigarrow_\beta b$ *and* $a \twoheadrightsquigarrow_\beta c$ *then by Corollary 2.11,* $a =_\beta b$ *and* $a =_\beta c$. *Hence,* $b =_\beta c$ *and by the Church Rosser property for the classical lambda calculus, there exists* d *such that* $b \twoheadrightarrow_\beta d$ *and* $c \twoheadrightarrow_\beta d$. *But, by Lemma 2.9,* $a \twoheadrightarrow_\beta b$ *implies* $a \twoheadrightsquigarrow_\beta b$. *Hence the Church-Rosser theorem holds for the general* β*-reduction.* □

3 Term reshuffling

Let us go back to the definition of reducible couples. Recall that if $\overline{s} \equiv s_1 \cdots s_m$ for $m > 1$ where $s_1 s_m$ is a reducible couple then $s_2 \cdots s_{m-1}$ is a well-balanced segment, $s_1 \equiv (a)$ is the abstraction item of the reducible couple and $s_m \equiv [v]$ is its application item. Now, we can move s_1 in $\overline{s}$ so that it occurs adjacently to s_m. That is, we may rewrite $\overline{s}$ as $s_2 \cdots s_{m-1} s_1 s_m$.

Example 3.1 In our item notation, the term $(a)(b)[x](c)[y][z](d)z$ can be easily rewritten as $(b)[x](c)[y](a)[z](d)z$ by moving the item (a) to the right. Hence, we can rewrite (or *reshuffle*) a term so that all application items stand next to their matching abstraction items. This means that we can keep the old β-axiom and we can contract redexes in any order. Such an action of reshuffling is not easy to describe in the classical notation. That is, it is difficult to describe how $((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a$, is rewritten as $(\lambda_x.(\lambda_y.(\lambda_z.zd)a)c)b$. This is another advantage of our item notation.

Note furthermore that in Λ, the shuffling is not problematic due to the Barendregt Convention which means that no free variable will become unnecessarily bound after reshuffling due to the fact that names of bound and free variables are distinct.

Lemma 3.2 *If* v° *is a free occurrence of* v *in* $s\overline{s_1}\,a$, *then* v° *is free in* $\overline{s_1}s\,a$.

Proof: *By BC as* $[v]$ *does not occur in* $s\overline{s_1}a$. □

Example 3.3 Note that in Example 3.1, reshuffling does not affect the "meaning" of the term. In fact, in $t \equiv (a)(b)[x](c)[y][z](d)z$, the free variable a cannot be captured by $[x]$ or $[y]$. Moreover, t is equivalent, semantically and procedurally, to $(b)[x](c)[y](a)[z](d)z$.

That is, the application items of reducible couples can occupy different positions in a term, without disturbing the meaning of the term, both semantically and procedurally. We call this process of moving application items of reducible couples in a term to occupy positions adjacent to their abstraction partners, *term reshuffling*. This term reshuffling should be such that *all* the application items of well-balanced segments in a term are shifted to the right until they meet their partners. Before we define term reshuffling, we need to understand better the structure of terms. Therefore the following section.

3.1 Partitioning terms into bachelor and well-balanced segments

With Definition 2.3 and Example 2.4, we may categorize the main items of a term t into different classes:

1. The "partnered" items (i.e. the application and abstraction items which are partners, hence "coupled" to a matching one).
2. The "bachelors" (i.e. the bachelor abstraction items and bachelor application items).

Lemma 3.4 *Let $\overline{s}$ be the body of a term a. Then the following holds:*

1. *Each bachelor main abstraction item in $\overline{s}$ precedes each bachelor main application item in $\overline{s}$.*
2. *The removal from $\overline{s}$ of all bachelor main items, leaves behind a* well-balanced *segment.*
3. *The removal from $\overline{s}$ of all main reducible couples, leaves behind $[v_1]\dots[v_n](a_1)\dots(a_m)$, the segment consisting of all bachelor main abstraction and application items.*
4. *If $\overline{s} \equiv \overline{s_1}(b)\overline{s_2}[v]\overline{s_3}$ where $[v]$ and (b) match, then $\overline{s_2}$ is well-balanced.*

Proof: *1 is by induction on* $\mathtt{weight}(\overline{s'})$ *for $\overline{s} \equiv \overline{s'}[v]\overline{s''}$ and $[v]$ bachelor in $\overline{s}$. 2 and 3 are by induction on* $\mathtt{weight}(\overline{s})$. *4 is by induction on* $\mathtt{weight}(\overline{s_2})$. □

Note that we have assumed $\emptyset$ well-balanced. We assume it moreover non-bachelor.

Corollary 3.5 *For each non-empty segment $\overline{s}$, there is a unique* partitioning *in segments $\overline{s_0}, \overline{s_1}, \cdots, \overline{s_n}$, such that*

1. $\overline{s} \equiv \overline{s_0}\,\overline{s_1}\cdots\overline{s_n}$,
2. *For all $0 \leq i \leq n$, $\overline{s_i}$ is well-balanced in $\overline{s}$ for even i and $\overline{s_i}$ is bachelor in $\overline{s}$ for odd i.*
3. *If $\overline{s_i}$ and $\overline{s_j}$ for $0 \leq i, j \leq n$ are bachelor abstraction resp. application segments, then $\overline{s_i}$ precedes $\overline{s_j}$ in $\overline{s}$.*
4. *If $i \geq 1$ then $\overline{s_{2i}} \not\equiv \emptyset$.* □

This is actually a very nice corollary. It tells us a lot about the structure of our terms.

Example 3.6 $\overline{s} \equiv [x][y](a)[z][x'](b)(c)(d)[y'][z'](e)$, has the following partitioning:

- well-balanced segment $\overline{s_0} \equiv \emptyset$
- bachelor segment $\overline{s_1} \equiv [x][y]$,
- well-balanced segment $\overline{s_2} \equiv (a)[z]$,
- bachelor segment $\overline{s_3} \equiv [x'](b)$,
- well-balanced segment $\overline{s_4} \equiv (c)(d)[y'][z']$,
- bachelor segment $\overline{s_5} \equiv (e)$.

3.2 The reshuffling procedure

Definition 3.7 *TS and T are defined mutually recursively such that:*

$$
\begin{array}{lll}
TS(\emptyset) & =_{df} & \emptyset \\
TS(\overline{s}v) & =_{df} & TS(\overline{s})v \\
TS(s_1 \cdots s_n) & =_{df} & TS(s_1) \cdots TS(s_n) \quad \textit{if } s_1 \cdots s_n \textit{ is bachelor} \\
TS((a)) & =_{df} & (TS(a)) \\
TS([v]) & =_{df} & [v] \\
TS(\overline{s}) & =_{df} & T(\emptyset, \overline{s}) \quad \textit{if } \overline{s} \textit{ is well-balanced} \\
TS(\overline{s_0} \cdots \overline{s_n}) & =_{df} & TS(\overline{s_0}) \cdots TS(\overline{s_n}) \quad \textit{If } \overline{s_0} \cdots \overline{s_n}\textit{, is the unique} \\
 & & \textit{partitioning of Corollary 3.5} \\
T(\overline{s}(a), [v]\overline{s'}) & =_{df} & (a)[v]T(\overline{s}, \overline{s'}) \\
T(\overline{s}, (a)\overline{s'}) & =_{df} & T(\overline{s}(TS(a)), \overline{s'}) \\
T(\emptyset, \emptyset) & =_{df} & \emptyset
\end{array}
$$

Note that in this definition, we use $\overline{s}$ bachelor to mean $\overline{s}$ bachelor in $\overline{s}$.

Lemma 3.8

1. *If $\overline{s}$ is well-balanced, then $T(\overline{s_1}, \overline{s}\,\overline{s_2}) \equiv TS(\overline{s})T(\overline{s_1}, \overline{s_2})$.*
2. *If $\overline{s}$ is well-balanced and none of its binding variables are free in a, then $TS((a)\overline{s}[v]\overline{s'}) \equiv TS(\overline{s}(a)[v]\overline{s'})$.*
3. *If $\overline{s}$ contains no items which are partnered in a then $TS(\overline{s}a) \equiv TS(\overline{s})TS(a)$.*
4. *If $\overline{s}$ is well-balanced or is bachelor in $\overline{s}a$ then $TS(\overline{s}a) \equiv TS(\overline{s})TS(a)$.*

Proof: *1: by induction on* $\texttt{weight}(\overline{s})$. *Case* $\texttt{weight}(\overline{s}) = 0$ *then obvious. Case* $\overline{s} \equiv (a)\overline{s'}[v]\overline{s''}$ *then* $\overline{s'}$ *and* $\overline{s''}$ *are well-balanced and*

$$
\begin{array}{lcll}
T(\overline{s_1}, (a)\overline{s'}[v]\overline{s''}\,\overline{s_2}) & \equiv & T(\overline{s_1}(TS(a)), \overline{s'}[v]\overline{s''}\,\overline{s_2}) & \equiv^{IH} \\
TS(\overline{s'})T(\overline{s_1}(TS(a)), [v]\overline{s''}\,\overline{s_2}) & \equiv & TS(\overline{s'})(TS(a))[v]T(\overline{s_1}, \overline{s''}\,\overline{s_2}) & \equiv^{IH} \\
TS(\overline{s'})(TS(a))[v]TS(\overline{s''})T(\overline{s_1}, \overline{s_2}) & \equiv & TS(\overline{s'})T((TS(a)), [v]\overline{s''})T(\overline{s_1}, \overline{s_2}) & \equiv^{IH} \\
T((TS(a)), \overline{s'}[v]\overline{s''})T(\overline{s_1}, \overline{s_2}) & \equiv & TS((a)\overline{s'}[v]\overline{s''})T(\overline{s_1}, \overline{s_2}) &
\end{array}
$$

2: using 1. 3: let $t \equiv \overline{s_0} \cdots \overline{s_n}v$ and $\overline{s} \equiv \overline{s_0'} \cdots \overline{s_m'}$ be partitionings. Use cases on $\overline{s_0}$ being empty or not and on $\overline{s_m'}$ being bachelor or well-balanced. 4: This is a corollary of 3 above. □

The next lemma shows that $TS(a)$ changes all reducible couples of a to reducible segments.

Lemma 3.9 *For every subterm b of a term a, the following holds:*

1. *$TS(b)$ is well-defined.*
2. *If $\overline{s} \equiv (c)\overline{s'}[v]$ is a subsegment of b and (c) matches $[v]$, then $TS(\overline{s}) \equiv TS(\overline{s'})(TS(c))[v]$.*
3. *If $\overline{s} \equiv s_1 \cdots s_n$ is a bachelor subsegment of b, then $TS(\overline{s}) \equiv TS(s_1) \cdots TS(s_n)$ is a bachelor subsegment of $TS(b)$.*
4. *If $\overline{s}$ is a subsegment of b which is well-balanced, then $TS(\overline{s})$ is well-balanced.*

Proof: *By induction on a.*

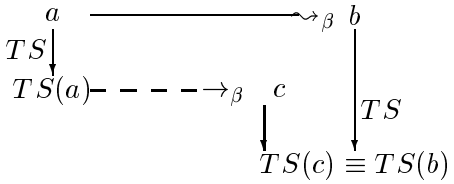
- *Case $a \equiv v$ then a is the unique subterm of a and all $1 \cdots 4$ hold.*
- *Assume $a \equiv sc$ where IH holds for $\mathtt{body}(s)$ and for c. Let b be a subterm of a. If b is a subterm of $\mathtt{body}(s)$ or of c then use IH. If $b \equiv a$ then:*
  - *Case s is bachelor then $TS(a) \equiv^{Lemma\ 3.8} TS(s)TS(c)$. Here all $1 \cdots 4$ hold by IH on $\mathtt{body}(s)$ and c.*
  - *Case $s \equiv (d)$ matches $[v]$ in a. I.e. $a \equiv (d)\overline{s}[v]e$ then $TS(a) \equiv^{Lemma\ 3.8} TS(\overline{s})(TS(d))[v]TS(e)$. Now use IH to show $1 \cdots 4$.* □

Lemma 3.10 *For all variables v and terms a, b we have:*
$TS(a) \equiv TS(TS(a))$ and $TS(a[v := b]) \equiv TS(TS(a)[v := TS(b)])$.

Proof: *By induction on a we show that for all subterms c of a, $TS(c) \equiv TS(TS(c))$ and $TS(c[v := b]) \equiv TS(TS(c)[v := TS(b)])$.* □

Note that if $a \to_\beta b$ and if all the reducible couples in a are reducible segments, then it is not necessary that all the reducible couples of b are reducible segments. Furthermore, if $a \rightsquigarrow_\beta b$ then it is not necessary that $TS(a) \rightsquigarrow_\beta TS(b)$ (nor even $TS(a) \to_\beta TS(b)$). For example, $d \equiv (a)(b)[x]\underline{([y]y)[z]}z \rightsquigarrow_\beta (a)(b)[x][y]y$ but $TS(d) \equiv d \not\rightsquigarrow_\beta (b)[x](a)[y]y$. Following this remark, we show that in a sense, term reshuffling preserves β-reduction.

Lemma 3.11 *If $a, b \in \Lambda$ and $a \rightsquigarrow_\beta b$ then $(\exists c)[(TS(a) \to_\beta c) \wedge TS(c) \equiv TS(b)]$. In other words, the following diagram commutes:*



Proof: *By induction on the general $\rightsquigarrow_\beta$. As the compatibility case is easy, we will only consider the case where $a \equiv \overline{s'}(d)\overline{s}[v]e \rightsquigarrow_\beta b \equiv \overline{s'}\overline{s}\{e[v := d]\}$. Here, we use induction on the number n of bachelor application items of $\overline{s'}$ that are partnered in e. Recall that $\overline{s}$ is well-balanced.*

- *Case $n = 0$ then*

$$
\begin{array}{ll}
TS(\overline{s'}(d)\overline{s}[v]e) & \equiv^{Lemma\ 3.8\ (4)} \\
TS(\overline{s'})TS((d)\overline{s}[v])TS(e) & \equiv^{Lemma\ 3.8\ (2,4)} \\
TS(\overline{s'})TS(\overline{s})(TS(d))[v]TS(e) & \to_\beta \\
TS(\overline{s'})TS(\overline{s})\{TS(e)[v := TS(d)]\} & \equiv c. \\
TS(c) & \equiv^{Lemmas\ 3.9,\ 3.10,\ 3.8\ (4)} \\
TS(\overline{s'})TS(\overline{s})TS(TS(e)[v := TS(d)]) & \equiv^{Lemma\ 3.10} \\
TS(\overline{s'})TS(\overline{s})TS(e[v := d]) & \equiv \\
TS(\overline{s'}\ \overline{s}(e[v := d])) &
\end{array}
$$

- *Assume the property holds for n and let us show it for the case where $\overline{s'}$ contains $n+1$ application items which match abstraction items of e. Let (c) be the leftmost such application item of $\overline{s'}$. Take $\overline{s'} \equiv \overline{s'_1}(c)\overline{s''_1}$ and $e \equiv \overline{s'_2}[v']f$ where (c) matches $[v']$. By Lemma 3.4, $(c)\overline{s''_1}(d)\overline{s}[v]\overline{s'_2}[v']$ is well-balanced. Moreover, no item of $\overline{s'_1}$ has a partner in $(c)\overline{s''_1}(d)\overline{s}[v]e$. As $\overline{s''_1}(d)\overline{s}[v]\overline{s'_2}(c)[v']f \rightsquigarrow_\beta \overline{s''_1}\overline{s}\{\overline{s'_2}(c)[v']f[v := d]\}$, we find by IH, g such that*
  $TS(\overline{s''_1}(d)\overline{s}[v]\overline{s'_2}(c)[v']f) \rightarrow_\beta g \wedge TS(g) \equiv TS(\overline{s''_1}\overline{s}\{\overline{s'_2}(c)[v']f[v := d]\})$.
  Now, $TS(\overline{s'_1})g$ is the wanted term because:
  $TS(a) \equiv^{Lemma\ 3.8\ (4)} TS(\overline{s'_1})TS((c)\overline{s''_1}(d)\overline{s}[v]\overline{s'_2}[v']f) \equiv^{Lemma\ 3.8\ (2)}$
  $TS(\overline{s'_1})TS(\overline{s''_1}(d)\overline{s}[v]\overline{s'_2}(c)[v']f) \rightarrow_\beta TS(\overline{s'_1})g$
  and $TS(TS(\overline{s'_1})g) \equiv^{Lemma\ 3.10}$
  $TS(\overline{s'_1})TS(\overline{s''_1}\overline{s}\{\overline{s'_2}(c)[v']f[v := d]\}) \equiv^{Lemma\ 3.8\ (2),\ BC}$
  $TS(\overline{s'_1})TS((c)\overline{s''_1}\overline{s}\{\overline{s'_2}[v := d]\}[v']\{f[v := d]\}) \equiv^{Lemma\ 3.8\ (4)}$
  $TS(\overline{s'_1}(c)\overline{s''_1}\overline{s}\{\overline{s'_2}[v']f[v := d]\}) \equiv TS(b)$. □

Corollary 3.12 *If $a \twoheadrightarrow_\beta b$ then there exists $a_0, a_1 \cdots a_n$ such that:*
$[(a \equiv a_0) \wedge (TS(a_0) \rightarrow_\beta a_1) \wedge (TS(a_1) \rightarrow_\beta a_2) \wedge \cdots \wedge (TS(a_{n-1}) \rightarrow_\beta a_n) \wedge (TS(a_n) \equiv TS(b))]$.

Proof: *By induction on $\twoheadrightarrow_\beta$.*

- *Case $a \rightsquigarrow_\beta b$ use Lemma 3.11.*
- *Case $a \twoheadrightarrow_\beta a$ then obvious ($n = 1 \wedge a_0 \equiv a \wedge a_1 \equiv TS(a)$).*
- *Case $a \twoheadrightarrow_\beta c \wedge c \twoheadrightarrow_\beta b$, then by IH, there exists $a_0, a_1, \cdots, a_n, b_0, b_1, \cdots b_m$ such that $(a \equiv a_0) \wedge (TS(a_0) \rightarrow_\beta a_1) \wedge (TS(a_1) \rightarrow_\beta a_2) \wedge \cdots \wedge (TS(a_{n-1}) \rightarrow_\beta a_n) \wedge (TS(a_n) \equiv TS(c)) \wedge (c \equiv b_0) \wedge (TS(b_0) \rightarrow_\beta b_1) \wedge (TS(b_1) \rightarrow_\beta b_2) \wedge \cdots \wedge (TS(b_{m-1}) \rightarrow_\beta b_m) \wedge (TS(b_m) \equiv TS(b))$. Hence, $(a \equiv a_0) \wedge (TS(a_0) \rightarrow_\beta a_1) \wedge \cdots \wedge (TS(a_{n-1}) \rightarrow_\beta a_n) \wedge (TS(a_n) \rightarrow_\beta b_1) \wedge \cdots \wedge (TS(b_{m-1}) \rightarrow_\beta b_m) \wedge (TS(b_m) \equiv TS(b))$.*
- *The compatibility case is easy.*

Note that in the basic case and the reflexive case we get $n = 1$ for sure. In the transitive case, this may not be the case. For example, $a \equiv [x]([y][z]x)[u](x)(x)u \twoheadrightarrow_\beta b \equiv [x](x)[y]x$ and does not satisfy Lemma 3.11. There are however a_0, a_1 and a_2 such that $a_0 \equiv a$, $a_1 \equiv [x](x)(x)[y][z]x$ and $a_2 \equiv b$ such that $TS(a_0) \rightarrow_\beta a_1 \wedge TS(a_1) \rightarrow_\beta a_2$ and $TS(a_2) \equiv TS(b)$. □

4 Conclusion

Classical reduction is in certain respects unattractive, not only because we should be able to discuss the existing redexes in a term at first sight, but also for at least two more reasons:

1. It may be the case that in some applications, we may want to contract a certain redex before other ones. This is the case for example in lazy evaluation where we may be interested in freezing some redexes but in working with the term as usual. That is for example, in the term $t \equiv ((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a$, we may want to substitute a for z in $(\lambda_x.(\lambda_y.zd)c)b$ before c has been substituted for y and b has been substituted for x. I.e. we need to carry out the reduction which substitutes a for z while the other two reductions which substitute c and b for y and x respectively are frozen.

2. Having a term like t above, we want to be able to discuss its *needed* redexes for reasons that are explained in [BKKS 87]. In fact, the needed redexes in a term a as introduced in that paper are those redexes that are contracted in every reduction of a to normal form. Now, [BKKS 87] provides many results which are important especially for the implementor in that it frees him from having to stick to either an inefficient but terminating normal order strategy, or an efficient but non terminating applicative strategy. Our approach enhances the work of [BKKS 87].

In this paper, we presented a notation which enables us to extend the classical notion of a redex and of β-reduction. This extension helps us to see more needed redexes than in classical calculus. The notation moreover allows us to reshuffle the term in hand so that more redexes can become visible and be contracted even using the old β-reduction. Both the generalised reduction and the reshuffling of the term are difficult to describe in the classical notation. Another attractive feature of our notation, is the ability to partition terms into bachelor and well-balanced segments (see Corollary 3.5). Such a partitioning, we believe, can play an important role in the study of reduction strategies. This is under investigation at present.

The notation presented in this paper has further advantages than generalising reduction and term reshuffling. These advantages are studied in our articles mentioned in the bibliography. Of these advantages however, we mention the ability to describe substitution explicitly as in [KN 93] and of generalising type systems as in [KN 94]. There is moreover the advantage of being able to make normal order reduction more efficient than in the classical calculus. The reason for this being that when searching for the leftmost outermost redex in a term, we need to make less recursive calls in item notation than in classical notation because in item notation, a term has a more linear structure. This and other advantages are investigated further in [KN 9z].

Finally, our discussion of reduction in this paper has been in terms of Curried functions. One could also explain the main idea using multi-argument functions and Currying/unCurrying. More specifically, one could transform an application by:

$$(\lambda_x.\lambda_y.\lambda_z.e)abc = (\lambda_{<x,y,z>}.e) < a,b,c >= (\lambda_{<z,x,y>}.e) < c,a,b >= (\lambda_z.\lambda_x.\lambda_y.e)cab.$$

This idea is based on isomorphisms (discussed in [R 91]) such as:

$$A \to B \to C = A \times B \to C = B \times A \to C = B \to A \to C$$

Of course, it would be nice to re-explore our approach in terms of multi-argument functions. This will also lead to the concept of simultaneous substitution which deserves attention.

References

[BKKS 87] Barendregt, H.P., Kennaway, J.R., Klop, J.W., and Sleep M.R., Needed reduction and spine strategies for the λ-calculus, *Information and Computation 75 (3)*, 1191-231, 1987.

[de Bruijn 93] Bruijn, N.G. de, Algorithmic definition of lambda-typed lambda calculus, in Huet, G. and Plotkin, G. eds. *Logical Environments*, 131-146, Cambridge University Press, 1993.

[KN 93] Kamareddine, F., and Nederpelt, R.P., On stepwise explicit substitution, *International Journal of Foundations of Computer Science 4 (3)*, 197-240, 1993.

[KN 94] Kamareddine, F. and Nederpelt, R.P., A unified approach to type theory through a refined λ-calculus, *Theoretical Computer Science 136*, 183-216, 1994.

[KN 9z] Kamareddine, F., and Nederpelt, R.P., *The beauty of the λ-calculus*, in preparation.

[Nederpelt 73] Nederpelt, R.P., *Strong normalisation in a typed lambda calculus with lambda structured types*, Ph.D. thesis, Eindhoven University of Technology, Department of Mathematics and Computer Science, 1973, to appear in Nederpelt, R.P., Geuvers, J.H. and de Vrijer, R.C.: *Selected Papers on Automath*, North Holland, 1994.

[R 91] Rittri, M., Using types as search keys in function libraries, *Journal of Functional Programming 1, 1*, 71-90, 1991.