Refining reduction in the lambda calculus

Citation for published version (APA):

Document status and date:
Published: 01/01/1994

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 02. Oct. 2020
Refining Reduction in the Lambda Calculus

by

F. Kamareddine and R. Nederpelt

Computing Science Note 94/18
Eindhoven, April 1994
This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

All rights reserved
editors: prof.dr.M.Rem
         prof.dr.K.M.van Hee.
Refining Reduction in the lambda calculus *

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

April 14, 1994

*We are grateful for the discussions with Roel Bloo, Tijn Borghuis and Erik Poll, and for the helpful remarks received from them. 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.

†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 in 1993 and 1994.
Capsule Review

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 

\( t = (\lambda_{x_7}(\lambda_{x_6}.\lambda_{x_5}.x_5 x_4)x_3 x_2)x_1 \), only \( (\lambda_{x_6}.\lambda_{x_5}.x_5 x_4)x_3 \), and \( (\lambda_{x_7}.\lambda_{x_6}.x_5 x_4)x_3 \) are visible. Yet when reducing \( t \) to a normal form, a third redex must be contracted; namely \( (\lambda_{x_5}.x_5 x_4)x_3 \). This third redex is not immediately visible in \( t \). 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 parenthesis are grouped in a novel way. In our item notation, \( t \) will be written as:

\( [x_1 \delta][x_2 \delta][\lambda_{x_7}][\lambda_{x_6}][x_5][x_4 \delta][x_3][x_2] \)

and we can provide this in this item notation with a bracketing structure \( [[[]]] \) where \( \delta \) and \( \lambda \) correspond to \([\) and \( ]\) respectively (ignoring \( (x_4 \delta)x_3 \)). 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. Hence we see immediately that the redexes in \( t \) originate from the \( \delta \lambda \)-couples \( (x_2 \delta)[\lambda_{x_7}], (x_3 \delta)[\lambda_{x_6}] \) and \( (x_1 \delta)[\lambda_{x_5}] \). This natural matching was not present in the classical notation of \( t \). With item notation, we shall refine reduction in two ways:

1. We shall generalise β-reduction so that any redex can be contracted and hence we can contract the redex based on \( (x_1 \delta)[\lambda_{x_5}] \) before we contract any of the redexes based on \( (x_2 \delta)[\lambda_{x_7}] \) and \( (x_3 \delta)[\lambda_{x_6}] \). That is, the β-rule changes from \( (t_1 \delta)[\lambda_{x_7}] t_2 \beta t_2[v := t_1] \) to \( (t_1 \delta)[\lambda_{x_7}] t_2 \beta s(t_2[v := t_1]) \) for \( s \) having a matching bracketing structure. I.e. \( (t_1 \delta)[\lambda_{x_7}] \) is a redex in our generalised sense. So

\[
(x_1 \delta)[x_2 \delta][\lambda_{x_7}][x_3 \delta][\lambda_{x_6}][\lambda_{x_5}][x_4 \delta][x_5] \sim_{\beta} \frac{x_1 \delta}{x_2 \delta}[\lambda_{x_7}][x_3 \delta][\lambda_{x_6}][\lambda_{x_5}][x_4 \delta][x_5] \equiv (x_2 \delta)[\lambda_{x_7}][x_3 \delta][\lambda_{x_6}][x_4 \delta][x_5]
\]

We show moreover, that the Church Rosser property holds for the generalised β-reduction.

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 \( (x_1 \delta)[x_2 \delta][\lambda_{x_7}][x_3 \delta][\lambda_{x_6}][\lambda_{x_5}][x_4 \delta][x_5] \) to \( (x_2 \delta)[\lambda_{x_7}][x_3 \delta][\lambda_{x_6}][\lambda_{x_5}][x_4 \delta][x_5] \), in order to transform the bracketing structure \( [[[]]] \) into \( [[][]] \), where all the redexes correspond to adjacent \([\) and \( ]\). Such a reshuffling is more difficult to describe in classical notation. I.e. it is hard to say what exactly happened when \( (\lambda_{x_7}(\lambda_{x_6}.\lambda_{x_5}.x_5 x_4)x_3)x_2 \), is reshuffled to \( (\lambda_{x_7}(\lambda_{x_6}.\lambda_{x_5}.x_5 x_4)x_3)x_2 \). 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 shall show that for any term \( t \), in \( TS(t) \) all the \( \delta \)-items occur next to their matching \( \lambda \)-items. We show moreover, that if \( t \sim_{\beta} t' \) then \( (\exists t''\)\( TS(t) \rightarrow_{\beta} t''\) \& \( TS(t'') \equiv TS(t') \).
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 4
  1.1 The item notation and visible redexes 4
  1.2 The system A 5

2 Generalising redexes and β-reduction 6
  2.1 Extending redexes to δλ-couples 7
  2.2 Extending β-reduction and the Church Rosser theorem 8

3 Term reshuffling 10
  3.1 Partitioning the term into bachelor and well-balanced segments 10
  3.2 The reshuffling procedure 12

4 Conclusion 14
1 Introduction

In the classical λ-calculus, redexes and β-reduction are defined as follows

**Definition 1.1 (Redexes and β-reduction in classical notation)**

In the classical notation of the λ-calculus, a redex is of the form \((\lambda_v.t)t\). Moreover, one-step β-reduction \(\rightarrow_\beta\) is the compatible relation generated out of the axiom \(\beta: (\lambda_v.t)t' \rightarrow_\beta t[v := t']\). Many step β-reduction \(\rightarrow_\beta\) is the reflexive transitive closure of \(\rightarrow_\beta\).

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 \(I\) translates classical terms into our notation, item notation, then \(I(t_1/t_2)\) is written as \((I(t_2)\delta)I(t_1)\) and \(I(\lambda_v.t)\) is written as \((\lambda_v)I(t)\). Both \((t\delta)\) and \((\lambda_v)\) are called items.

**Example 1.2** \(I((\lambda_y.x)y)z = (z\delta)(\lambda_y)(y\delta)x\). The items are \((z\delta), (\lambda_y), (y\delta)\).

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 \((t\delta)(\lambda_v)\). We call the pair \((t\delta)(\lambda_v)\), a δλ-pair, or a δλ-segment. Moreover, the classical β-reduction axiom is:

\[(t\delta)(\lambda_v)t \rightarrow t[v := t']\].

One and many step β-reduction are defined as in Definition 1.1.

**Example 1.4** In the classical term \(t = ((\lambda_x.\lambda_y.x_6x_5x_4)x_3)x_2)\), we have the following redexes (the fact that neither \(x_6\) nor \(x_7\) appear as free variables in their respective scopes does not matter here; this is just to keep the example simple and clear):

1. \((\lambda_x.\lambda_y.x_5x_4)x_3\)
2. \((\lambda_x.\lambda_y.x_5x_4)x_2\)

Written in item notation, \(t\) becomes \((x_1\delta)(x_2\delta)(\lambda_x)(\lambda_y)(x_3\delta)(\lambda_x)(x_4\delta)x_5\). Here, the two classical redexes correspond to δλ-pairs as follows:

1. \((\lambda_x.\lambda_y.x_5x_4)x_3\) corresponds to \((x_1\delta)(\lambda_x)\). We ignore \((x_4\delta)x_3\) as it is easily retrievable in item notation. It is the maximal subterm of \(t\) to the right of \((\lambda_x)\).
2. \((\lambda_x.\lambda_y.x_5x_4)x_2\) corresponds to \((x_2\delta)(\lambda_y)\). Again \((x_3\delta)(\lambda_x)(x_4\delta)x_5\) is ignored for the same reason as above.

There is however a third redex which is not visible in the classical term. Namely, \((\lambda_x.\lambda_y.x_5x_4)x_1\). 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.

\[
\begin{align*}
(\lambda_x.\lambda_y.x_5x_4)x_2) & \rightarrow_\beta x_1 \\
(\lambda_x.\lambda_y.x_5x_4)x_3 & \rightarrow_\beta x_1 \\
(\lambda_x.\lambda_y.x_5x_4)x_1 & \rightarrow_\beta x_1 \\
\end{align*}
\]

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 \((\lambda_x.x_5.x_4)x_1\) corresponds to \((x_1\delta)(\lambda_x.x_5.x_4)\) but the \(\delta\)-item and \(\lambda\)-item are separated by the segment \((x_2\delta)(\lambda_x.x_3)(x_5)\) \((\lambda_{x_5})\). By extending the notion of a redex and of \(\beta\)-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 \(\delta\lambda\)-segment \((t_1\delta)(\lambda_{v'})\) to a \(\delta\lambda\)-couple being an item \((t_1\delta)\) and an item \((\lambda_{v'})\) separated by a segment \(\bar{s}\) which is a well-balanced segment. A well-balanced segment is a sequence of \(\delta\)- and \(\lambda\)-items which has the same structure as a matching composite of opening and closing brackets, each \(\delta\)-item corresponding to an opening bracket and each \(\lambda\)-item corresponding to a closing bracket. We will show that this generalised \(\beta\)-reduction is Church-Rosser. We show moreover that with our item notation, it is possible to shuffle a term so that \(\delta\lambda\)-couples become \(\delta\lambda\)-pairs. I.e., so that \((x_1\delta)(x_2\delta)(\lambda_{x_3})(\lambda_{x_4})(\lambda_{x_5})(x_6)\) becomes \((x_2\delta)(\lambda_{x_7})(x_3\delta)(\lambda_{x_8})(x_1\delta)(\lambda_{x_9})(x_4\delta)(x_5)\) where the three redexes correspond to the three \(\delta\lambda\)-pairs \((x_2\delta)(\lambda_{x_7}), (x_3\delta)(\lambda_{x_8})\) and \((x_1\delta)(\lambda_{x_9})\). Such a reshuffling is not easy to describe in classical notation. We shall show that term reshuffling is correct and preserves \(\beta\)-reduction.

1.2 The system \(\Lambda\)

We construct the system \(\Lambda\) out of a set of variables \(\mathcal{V}\) and a set of operators \(\Omega\) where a term is either a variable or is of the form \(s_1s_2\ldots s_n.v\) for variable \(v\) and items \(s_i\), for \(1 \leq i \leq n\). An item is defined either as \((\lambda_{v'})\) for variable \(v'\) or as \((t\delta)\) for \(t\) being a term. Both \(\delta\) and \(\lambda_{v'}\) are called operators; the first is an application operator, the second is an abstraction operator. The following definitions formalise our system \(\Lambda\).

**Definition 1.5** (Terms in item notation \(\Lambda\))

- We take \(\mathcal{V} = \{x_i; n \geq 1\}\) to be the set of variables and assume that if \(i \neq j\), then \(x_i \neq x_j\) for all \(i, j \geq 1\). We let \(v, v', v''\) range over \(\mathcal{V}\).
- We take \(\{\lambda_{v}; v \in \mathcal{V}\}\) to be the set of the abstraction operators (or \(\lambda\)-operators). We use \(\lambda_{v}, \lambda'_{v}, \lambda''_{v}, \ldots\) to range over these operators.
- We assume an application operator or \(\delta\)-operator \(\delta \notin \{\lambda_{v}; v \in \mathcal{V}\}\) and we use \(\omega, \omega', \omega'', \omega_1, \omega_2, \ldots\) to range over \(\Omega = \{\delta\} \cup \{\lambda_{v}; v \in \mathcal{V}\}\).
- We write terms in item notation as: \(\Lambda ::= \mathcal{V} \mid (\lambda\delta)\Lambda \mid (\lambda\Lambda)\Lambda\). We use \(t, t_1, t_2, \ldots\) to range over terms in item notation.

**Definition 1.6** ((main) items, (main, \(\delta\lambda\)) segment, body, endvar, \(\delta\lambda\)-segment, weight)

- If \(t\) is a \(\Lambda\)-term and \(v \in \mathcal{V}\) then \((t\delta)\) and \((\lambda_{v})\) are items, the former being a \(\delta\)-item and the latter a \(\lambda\)-item or a \(\lambda\)-item. We use \(s, s_1, s_2, \ldots\) 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 \(\bar{s}, \bar{s}_1, \bar{s}_2, \ldots\) as meta-variables for segments.
- Each term \(t\) is the concatenation of zero or more items and a variable: \(t = s_1s_2\ldots s_n.v\). These items \(s_1, s_2, \ldots, s_n\) are called the main items of \(t\). We call the segment \(s_1s_2\ldots s_n\), body\((t)\), and the variable \(v\), endvar\((t)\).
• Analogously, a segment \( \bar{s} \) is a concatenation of zero or more items \( \bar{s} \equiv s_1 s_2 \cdots s_n \). These items \( s_1, s_2, \cdots, s_n \) are called the main items, this time of \( \bar{s} \).

• A concatenation of main items is a main segment.

• An important case of a segment is that of a \( \delta \lambda \)-segment, being a \( \delta \)-item immediately followed by a \( \lambda \)-item. Sometimes, we call the \( \delta \lambda \)-segment, a \( \delta \lambda \)-pair.

• The weight of a \( \Lambda \)-segment \( \bar{s} \), \( \text{weight}(\bar{s}) \), is the number of main items that compose the segment. The weight of a \( \Lambda \)-term \( t \) is the weight of \( \text{body}(t) \).

Example 1.7 Let the \( \Lambda \)-term \( t \) be defined as \( (\lambda_{x_2})((x_2\delta)(\lambda_{x_3})x_3\delta)(\lambda_{x_4})x_4 \) and let the segment \( \bar{s} \) be \( (\lambda_{x_2})(x_2\delta)(\lambda_{x_3})x_3\delta)(\lambda_{x_4}) \). Then the main items of both \( t \) and \( \bar{s} \) are \( (\lambda_{x_2}), ((x_2\delta)(\lambda_{x_3})x_3\delta) \) and \( (\lambda_{x_4}) \), being a \( \lambda \)-item, a \( \delta \)-item, and a \( \lambda \)-item.

We identify terms that differ only in the name of bound variables (i.e. we take terms modulo \( \alpha \)-conversion). For example \( (\lambda_{x_1})x_1 \equiv (\lambda_{x_2})x_2 \). Furthermore, we assume the Barendregt variable convention which is formally stated as follows:

Definition 1.8 (BC Barendregt’s Convention for \( \Lambda \))

Names of bound variables will always be chosen such that they differ from the free ones in a term. Hence, we will not have \( (v\delta)(\lambda_v)v \), but \( (v\delta)(\lambda_{v'})v' \) instead. We extend BC to meta-terms like \( ((\lambda_v)t_2)[v := t'] \). This avoids the danger of clash of variables, since we assume that no free variable in \( t' \) occurs bound in \( (\lambda_v)t_2 \). Moreover, the extended BC implies that \( v \neq v' \).

With our BC, we define substitution as follows:

Definition 1.9 (Substitution in \( \Lambda \) with \( \text{BC} \))

If \( t, t' \) are \( \Lambda \)-terms and \( v \in \mathcal{V} \) and if \( \text{BC} \) is assumed, then we define the result of substituting \( t' \) for all the free occurrences of \( v \) in \( t \) as follows:

\[
[\lambda_{v'}]([v := t'] \equiv [v := t]_{t_2}) \rightarrow [v := t'] \quad \text{if } t \equiv v
\]

\[
[t[v := t']] = \begin{cases} 
  t' & \text{if } t \equiv v \\
  t'[v := t'] & \text{if } t \equiv v' \neq v \\
  (t_1[v := t']t_2)[v := t'] & \text{if } t \equiv (t_1t_2) \\
  (\lambda_v)t_2[v := t'] & \text{if } t \equiv (\lambda_v)t_2 
\end{cases}
\]

With this implicit substitution, \( \beta \)-reduction is defined as follows:

Definition 1.10 (\( \beta \)-reduction \( \rightarrow_\beta \) in \( \Lambda \))

In \( \Lambda \), \( \beta \)-reduction \( \rightarrow_\beta \) is the least compatible relation generated out of the following axiom:

\[
(\beta) \quad (t'\delta)(\lambda_v)t' \rightarrow_\beta t'[v := t'']
\]

2 Generalising redexes and \( \beta \)-reduction

As we have argued above, a term can contain a \( \delta \)-item \( (t_1\delta) \) and a \( \lambda \)-item \( (\lambda_v) \) such that eventually the reduction based on the \( \delta \lambda \)-segment \( (t_1\delta)(\lambda_v) \) should take place. This reduction however, can only so far take place when \( (t_1\delta) \) and \( (\lambda_v) \) are not separated by other items or segments. This makes it difficult to use the \( \lambda \)-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 $\beta$-reduction which enables the manipulation of the order of reduction and substitution. This general $\beta$-reduction will be an extension of the known $\beta$-reduction in that not only the $\delta\lambda$-segments result in firing reductions, but a more general notion of $\delta\lambda$-segments which we call $\delta\lambda$-couples.

2.1 Extending redexes to $\delta\lambda$-couples

Why should we in the term $(x_1 \delta)(x_2 \delta)(\lambda_{x_1})(x_3 \delta)(\lambda_{x_2})x\delta x_5$ not allow that the reduction based on $(x_1 \delta)(\lambda_{x_1})$ gets fired? There is no reason why we should not carry out some reductions before others. We ask the reader to convince himself of the fact that priority for the mentioned $\delta\lambda$-firing does not affect the final result. If we look moreover at this term, we find that what separates $(x_1 \delta)$ and $(\lambda_{x_2})$ is a segment with a particular structure. The same holds for the segment $(t_1 \delta)(\lambda_{t_1})$ separating $(t_2 \delta)$ and $(\lambda_{t_2})$ in $(t_1 \delta)(t_2 \delta)(\lambda_{t_2})(\lambda_{t_2})$. 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 $\beta$-reduction on the basis of a certain $\delta$-item and a $\lambda$-item occurring in one segment, the matching of the $\delta$ and the $\lambda$ 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 $\delta\lambda$-segment after a number of $\beta$-steps. This depends solely on the structure of the intermediate segment. If such an intermediate segment is well-balanced then the $\delta$-item and the $\lambda$-item match and $\beta$-reduction based on these two items may take place. Here is the definition of well-balanced segments:

**Definition 2.1** (well-balanced segments in $\Lambda$)

- An empty segment is a well-balanced segment.
- If $\Sigma$ is a well-balanced segment, then $(\delta)^n(\Sigma)$ 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 $\delta$- (or $\lambda$-)item corresponding with an opening (resp. closing) bracket.

**Example 2.2**

1. $(x_3 \delta)(\lambda_{x_2})$ forms a well-balanced segment as it corresponds to $[ ]$.
2. $(x_2 \delta)(\lambda_{x_1})$ forms a well-balanced segment as it corresponds to $[ ]$.
3. $(x_2 \delta)(\lambda_{x_1})(x_3 \delta)(\lambda_{x_2})$ forms a well-balanced segment as it corresponds to $[[ ]]$.
4. $(x_1 \delta)(x_2 \delta)(\lambda_{x_1})(x_3 \delta)(\lambda_{x_2})$ forms a well-balanced segment as it corresponds to $[[ ][ ]]$; the $\delta\lambda$-couples in this segment are $(x_2 \delta)(\lambda_{x_1})$, $(x_3 \delta)(\lambda_{x_2})$ and $(x_1 \delta)(\lambda_{x_2})$.

Now we can easily define what matching $\delta\lambda$-couples are. Namely, they are a $\delta$-item and a $\lambda$-item separated by a well-balanced segment. Such couples are reducible couples. The $\delta$-item and $\lambda$-item of the $\delta\lambda$-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, δλ- or reducible couple, partner, partnered item, bachelor item, bachelor segment)

Let \( t \) be a \( \lambda \)-term. Let \( \overline{s} \equiv s_1 \cdots s_n \) be a segment occurring in \( t \).

- 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 a \( \delta \)-item, \( s_j \) is a \( \lambda \)-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 \( \delta\lambda \)-couple or reducible couple.

- When \( s_i \) and \( s_j \) match, we call both \( s_i \) and \( s_j \) the partners in the \( \delta\lambda \)-couple. We also say that \( s_i \) and \( s_j \) are partnered items in \( \overline{s} \) (or in \( t \)).

- All \( \delta \)- (or \( \delta \)) items \( s_k \) that are not partnered in \( \overline{s} \), are called bachelor \( \delta \) (resp. \( \delta \)) items in \( \overline{s} \) (or in \( t \)).

- A segment \( \overline{s} \) consisting of only bachelor items (in \( \overline{s} \)), is called a bachelor segment.

- The segment \( s_1 \cdots s_n \) consisting of all bachelor main \( \delta \) (or \( \delta \)) items of \( \overline{s} \) is called the bachelor \( \delta \) (or \( \delta \)) segment of \( \overline{s} \).

Example 2.4 Let \( \overline{s} \equiv (\lambda v_1)(\lambda v_2)(t_3\delta)(\lambda v_3)(\lambda v_4)(t_7\delta)(t_9\delta)(\lambda v_5)(\lambda v_6)(t_{11}\delta) \). Then:

- \((t_3\delta)\) matches with \((\lambda v_1)\), \((t_7\delta)\) matches with \((\lambda v_7)\) and \((t_9\delta)\) matches with \((\lambda v_9)\). The segments \((t_3\delta)\) and \((t_7\delta)\) are \( \delta\lambda \)-segments (also \( \delta\lambda \)-couples), and there is another \( \delta\lambda \)-couple in \( \overline{s} \), viz. the couple of \((t_3\delta)\) and \((\lambda v_9)\).

- \((t_3\delta)\), \((\lambda v_9)\), \((t_7\delta)\), \((t_9\delta)\), \((\lambda v_7)\) and \((\lambda v_9)\), are the partnered main items of \( \overline{s} \). \((\lambda v_1)\), \((\lambda v_2)\), \((\lambda v_4)\), \((t_7\delta)\) and \((t_{11}\delta)\) are bachelor items.

- \((\lambda v_1)\) and \((\lambda v_4)\) are bachelor segments. \((t_7\delta)\) and \((t_9\delta)\) are bachelor segments, \((t_7\delta)\) and \((t_9\delta)\) are non-bachelor segments, the latter also being a well-balanced segment.

Remark 2.5

Observe that a \( \delta\lambda \)-segment is a \( \delta\lambda \)-couple and that the \( \delta \)-item and the \( \lambda \)-item in a \( \delta\lambda \)-couple are separated by zero, one or more \( \delta\lambda \)-couples.

2.2 Extending \( \beta \)-reduction and the Church Rosser theorem

Having argued above that \( \beta \)-reduction should not be restricted to the \( \delta\lambda \)-segments but may take into account other candidates, we can extend our notion of \( \beta \)-reduction in this vein. That is to say, we may allow \( \delta\lambda \)-couples to have the same "reduction rights" as \( \delta\lambda \)-segments. That is, the \( \beta \)-reduction of Definition 1.10 changes to the following:

Definition 2.6 (General \( \beta \)-reduction \( \overset{\beta}{\rightarrow} \) in \( \Lambda \))

General \( \beta \)-reduction \( \overset{\beta}{\rightarrow} \), is the least compatible relation generated out of the following axiom:

\[
(\text{general } \beta) \quad (t^i\delta)\overline{s}(\lambda v)\overset{\beta}{\rightarrow} t^i\delta \overline{s}[v := t^i]\quad \text{if } \overline{s} \text{ is well-balanced}
\]

Example 2.7 Take Example 1.4. As \((x_2\delta)(\lambda x_2)(x_3\delta)(\lambda x_3)\) is a well-balanced segment, then \((x_1\delta)(\lambda x_3)\) is a \( \delta\lambda \)-couple and

\[
t = (x_2\delta)(x_2\delta)(\lambda x_7)(x_3\delta)(\lambda x_8)(\lambda x_9)(x_4\delta)x_5 \overset{\beta i}{\rightarrow} (x_2\delta)(\lambda x_7)(x_3\delta)(\lambda x_8)(x_4\delta)x_5[\lambda x_5 := x_1] \equiv (x_2\delta)(\lambda x_7)(x_3\delta)(\lambda x_8)(x_4\delta)x_1
\]
The reducible couple \((x_1 \delta)(x_2 \delta)\) also has a corresponding ("generalized") redex in the traditional notation, which will appear after two one-step \(\beta\)-reductions, leading to \((x_2 \delta x_5 x_4) x_1\). With our generalised one-step \(\beta\)-reduction we could reduce \(((\lambda x_7 (x_6 \lambda x_5 x_3 x_2) x_1) x_2) x_1\) to \((x_7 (x_6 \delta x_1 x_4) x_3) x_2\). This reduction is difficult to carry out in the classical \(\lambda\)-calculus. The item notation enables a new and important sort of reduction which has not yet been studied in relation to the standard \(\lambda\)-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 \(((\lambda x_7 (x_6 \lambda x_5 . . . ) x_3 x_2) x_1)\), is compatible with \('[1 [2 [3] 2 ] ) 3', where \('[i' and \('[j' 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 \(\rightarrow_{\beta}\). Because a well-balanced segment may be empty, the general \(\beta\)-reduction rule presented above is really an extension of the classical \(\beta\)-reduction rule.

**Lemma 2.9** Let \(t_1, t_2\) be \(\lambda\)-terms. If \(t_1 \rightarrow_{\beta} t_2\) then \(t_1 \rightarrow_{\beta} t_2\). Moreover, if \(t_1 \rightarrow_{\beta} t_2\) comes from contracting a \(\delta\lambda\)-segment then \(t_1 \rightarrow_{\beta} t_2\).

**Proof:** Obvious as a \(\delta\lambda\)-segment by Remark 2.5. \(\square\)

The proof of the Church Rosser theorem is simple. The idea is to show that if \(t \rightarrow_{\beta} t'\) then \(t =_{\beta} t'\) (where \(=_{\beta}\) is the least equivalence relation closed under \(\rightarrow_{\beta}\) and \(\rightarrow_{\beta}\) the reflexive transitive closure of \(\rightarrow_{\beta}\)) and to use the Church Rosser property for \(=_{\beta}\).

**Lemma 2.10** If \(t \rightarrow_{\beta} t'\) then \(t =_{\beta} t'\).

**Proof:** It suffices to consider the case \(t \equiv t_1, t_2\) where the contracted redex is based on \(t_1 \delta (\lambda x_4) t_2\) and \(\delta\) is balanced (hence \(\text{weight}(\delta) = 2n\)). We shall prove the lemma by induction on \(\text{weight}(\delta)\).

- *Case* \(\text{weight}(\delta) = 0\) then obvious as \(\rightarrow_{\beta}\) coincides with \(\rightarrow_{\beta}\) in this case.
- *Assume* the property holds when \(\text{weight}(\delta) = 2n\). Take \(s\) such that \(\text{weight}(s) = 2n + 2\). Now, \(s \equiv (t_3 \delta (x_6 \lambda x_5 x_3 x_2 x_1) t_2)\) where \(\delta\) is any well-balanced.
  
  \[\begin{array}{l}
  \text{As } s \equiv \overline{t}(v := t_1) t_3 \rightarrow_{\beta} \overline{t}(v := t_1) [v' := t_3], \text{ we get by IH and compatibility that } t' =_{\beta} \overline{t}(v := t_1) [v' := t_3], \text{ with } t_3 \equiv \overline{t}(v := t_3) \text{ and } \delta\text{ is balanced.}
  
  \text{Moreover, } t \equiv \overline{t}(v := t_1) [v' := t_3] =_{\beta} \overline{t}(v := t_1) [v' := t_3].
  
  \text{Now, } t_4 =_{\beta} \overline{t}(v := t_4) [v' := t_4]. \text{ But by BC, } v, v' \notin \text{FV}(t_4), \text{ hence by IH and substitution, } t =_{\beta} \overline{t}(v := t_4) [v' := t_4].
  
  \text{Therefore } t =_{\beta} t'.
  \end{array}\]

**Corollary 2.11** If \(t \rightarrow_{\beta} t'\) then \(t =_{\beta} t'\).

**Theorem 2.12** The general \(\beta\)-reduction is Church-Rosser. i.e. If \(t \rightarrow_{\beta} t_1\) and \(t \rightarrow_{\beta} t_2\), then there exists \(t_3\) such that \(t_1 \rightarrow_{\beta} t_2\) and \(t_2 \rightarrow_{\beta} t_3\).

**Proof:** As \(t \rightarrow_{\beta} t_1\) and \(t \rightarrow_{\beta} t_2\) then by Corollary 2.11, \(t =_{\beta} t_1\) and \(t =_{\beta} t_2\). Hence, \(t_1 =_{\beta} t_2\) and by the Church Rosser property for the classical lambda calculus, there exists \(t_3\) such that \(t_1 \rightarrow_{\beta} t_3\) and \(t_2 \rightarrow_{\beta} t_3\). But, \(t \rightarrow_{\beta} t'\) implies \(t \rightarrow_{\beta} t'\). Hence the Church-Rosser theorem holds for the general \(\beta\)-reduction. \(\square\)
3 Term reshuffling

Let us go back to the definition of $\delta\lambda$-couples. Recall that if $\bar{s} \equiv s_1 \cdots s_m$ for $n > 1$ where $s_1 s_m$ is a $\delta\lambda$-couple then $s_2 \cdots s_{m-1}$ is a well-balanced segment, $s_1 \equiv (\ell_1 d)$ is the $\delta$-item of the $\delta\lambda$-couple and $s_m \equiv (\lambda_d)$ is its $\lambda$-item. Now, we can move $s_1$ in $\bar{s}$ so that it occurs adjacent to $s_m$. That is, we may rewrite $\bar{s}$ as $s_2 \cdots s_{m-1} s_1 s_m$.

**Example 3.1** In our item notation, the term $(x_1 d)(x_2 d)(x_3 \delta)(x_2 \delta)(\lambda_{x_2})(x_4 \delta)x_5$ can be easily rewritten as $(x_2 \delta)(x_1 d)(x_3 \delta)(\lambda_{x_2})(x_4 \delta)x_5$ by moving the item $(x_1 d)$ to the right. Hence, we can rewrite (or reshuffle) a term so that all $\delta$-items stand next to their matching $\lambda$-items. This means that we can keep the old $\beta$-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 $((\lambda_{x_2},x_{x_3},x_3 x_4)x_3)x_2$, is rewritten as $(\lambda_{x_2},x_{x_3},x_3 x_4)x_3)x_2$. This is another advantage of our item notation.

Note furthermore that in $A$, 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 $v^\circ$ is a free occurrence of $v$ in $s\overline{t}$, then $v^\circ$ is free in $\overline{s}t$.

**Proof:** By $BC$ as $\lambda_v$ does not occur in $\overline{s}t$. \hfill $\Box$

**Example 3.3** Note that in Example 3.1, reshuffling does not affect the “meaning” of the term. In fact, in $t \equiv (x_1 d)(x_2 \delta)(x_3 \delta)(\lambda_{x_2})(x_4 \delta)x_5$, the free variable $x_1$ cannot be captured by $\lambda_{x_2}$ or $\lambda_{x_3}$. Moreover, $t$ is equivalent, semantically and procedurally, to $(x_2 \delta)(x_1 d)(x_3 \delta)(\lambda_{x_2})(x_4 \delta)x_5$.

That is, the $\delta$-items of $\delta\lambda$-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 $\delta$-items of $\delta\lambda$-couples in a term to occupy positions adjacent to their $\lambda$-partners, term reshuffling. This term reshuffling should be such that all the $\delta$-items of well-balanced segments in a term are shifted to the right until they meet their $\lambda$-partners. Before we define term reshuffling, we need to understand better the structure of terms. Therefore the following section.

3.1 Partitioning the term into bachelor and well-balanced segments

**Convention 3.4** In what follows, we will use $(\ell \omega)$ to refer to $(\ell \delta)$ when $\omega = \delta$ and to $(\ell \lambda_d)$ or $(\ell \lambda_u)$ when $\omega \equiv \lambda_u$. I.e. we extend $A$ with $\varepsilon$ and we take $\varepsilon$ to represent the empty term. At the moment, this is only to enable us to use a uniform notation for terms in our definitions. (However, this opens the possibility for an extension to typed $\lambda$-calculus with items $(t \lambda_u)$, where $t$ is the type of $v$.)

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 $\delta$- and $\lambda$-items which are partners, hence “coupled” to a matching one).
2. The “bachelors” (i.e. the bachelor $\lambda$-items and bachelor $\delta$-items).
Lemma 3.5
Let $\bar{s}$ be the body of a term $t$. Then the following holds:

1. Each bachelor main $\lambda$-item in $\bar{s}$ precedes each bachelor main $\delta$-item in $\bar{s}$.
2. The removal from $\bar{s}$ of all bachelor main items, leaves behind a well-balanced segment.
3. The removal from $\bar{s}$ of all main $\delta\lambda$-couples, leaves behind a $\underbrace{\lambda_1 \lambda_2 \cdots \lambda_n}_{n} \underbrace{\delta_1 \delta_2 \cdots \delta_m}_{m}$-segment, consisting of all bachelor main $\lambda$- and $\delta$-items.
4. If $\bar{s} \equiv \bar{s}_1(t',\delta)\bar{s}_2(\lambda_n)\bar{s}_0$ where $(\lambda_n)$ and $(t',\delta)$ match, then $\bar{s}_2$ is well-balanced.

Proof: 1 is by induction on weight($\bar{s}$) for $\bar{s} \equiv \bar{s}'(\lambda_n)\bar{s}'$ and $(\lambda_n)$ bachelor in $\bar{s}$. 2 and 3 are by induction on weight($\bar{s}$). 4 is by induction on weight($\bar{s}_2$).

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

Corollary 3.6 For each non-empty segment $\bar{s}$, there is a unique partitioning in segments $\bar{s}_0, \bar{s}_1, \cdots, \bar{s}_n$, such that

1. $\bar{s} \equiv \bar{s}_0 \bar{s}_1 \cdots \bar{s}_n$,
2. For all $0 \leq i \leq n$, $\bar{s}_i$ is well-balanced in $\bar{s}$ for even $i$ and $\bar{s}_i$ is bachelor in $\bar{s}$ for odd $i$.
3. each bachelor $\lambda$-segment $\bar{s}_j$ precedes each bachelor $\delta$-segment $\bar{s}_k$ in $\bar{s}$.
4. If $i \geq 1$ then $\bar{s}_{2i} \not\equiv \varepsilon$.

Example 3.7
The segment $\bar{s} \equiv (\lambda_{v_1})(\lambda_{v_2})(t_3\delta)(\lambda_{v_3})(\lambda_{v_4})(t_6\delta)(t_7\delta)(t_8\delta)(\lambda_{v_5})(\lambda_{v_6})(t_{11}\delta)$, has the following partitioning:

- well-balanced segment $\bar{s}_0 \equiv \varepsilon$
- bachelor segment $\bar{s}_1 \equiv (\lambda_{v_1})(\lambda_{v_2})$
- well-balanced segment $\bar{s}_2 \equiv (t_3\delta)(\lambda_{v_3})$
- bachelor segment $\bar{s}_3 \equiv (\lambda_{v_4})(t_6\delta)$
- well-balanced segment $\bar{s}_4 \equiv (t_7\delta)(t_8\delta)(\lambda_{v_5})(\lambda_{v_6})$
- bachelor segment $\bar{s}_5 \equiv (t_{11}\delta)$. 


3.2 The reshuffling procedure

Definition 3.8 $TS$ and $T$ are defined mutually recursively such that:

$$
TS(\varepsilon) = \varepsilon
$$

$$
TS(\overline{x}y) = TS(\overline{x})y
$$

$$
TS((t_1 \omega_1) \cdots (t_n \omega_n)) = (TS(t_1) \omega_1) \cdots (TS(t_n) \omega_n) \text{ if } (t_1 \omega_1) \cdots (t_n \omega_n) \text{ is bachelor}
$$

$$
TS(\overline{x}) = T(\varepsilon, \overline{x})
$$

$$
TS(\overline{x_0} \cdots \overline{x_n}) = TS(\overline{x_0}) \cdots TS(\overline{x_n})
$$

If $\overline{x_0} \cdots \overline{x_n}$ is the unique partitioning of Corollary 3.6

$$
T(\overline{s(t_0)^1} \lambda_v \overline{s^2}) = (t_0)(\lambda_v)T(\overline{s}, \overline{s^2})
$$

$$
T(\overline{s(t_0)^1} \lambda_v \overline{s^2}) = T(\overline{s(TS(t_0)^1)}, \overline{s^2})
$$

$$
T(\varepsilon, \varepsilon) = \varepsilon
$$

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

Lemma 3.9

1. If $\overline{x}$ is well-balanced, then $T(\overline{s_1}, \overline{s_2}) = TS(\overline{x})T(\overline{s_1}, \overline{s_2})$.

2. If $\overline{x}$ is well-balanced and none of its binding variables are free in $t$, then $TS((t_0)\overline{s} \lambda_v \overline{s^1}) = TS(\overline{s(t_0)} \lambda_v \overline{s^1})$.

3. If $\overline{x}$ contains no items which are partner in $t$ then $TS(\overline{\omega t}) = TS(\overline{x})TS(t)$.

4. If $\overline{x}$ is well-balanced or is bachelor in $\overline{x}$ then $TS(\overline{\omega t}) = TS(\overline{x})TS(t)$.

Proof: 1: by induction on weight($\overline{x}$). Case weight($\overline{x}$) = 0 then obvious. Case $\overline{x} = (t_0)\overline{s^2} \lambda_v \overline{s^1}$ then

$$
T(\overline{s_1(t_0)^1} \lambda_v \overline{s^2}) = TS(\overline{s_1(TS(t_0)^1)}, \lambda_v \overline{s^2}) \equiv TH
$$

$$
TS(\overline{s})T(\overline{s_1(TS(t_0)^1)}, \lambda_v \overline{s^2}) \equiv TH
$$

$$
TS(\overline{s(TS(t_0)^1)}, \lambda_v \overline{s^2}) = TS(\overline{s}s(TS(t_0)^1)), \lambda_v \overline{s^2}) \equiv TH
$$

$$
T(\overline{s(t_0)^1} \lambda_v \overline{s^2}) = TS(\overline{s(TS(t_0)^1)}, \lambda_v \overline{s^2}) \equiv TH
$$

2: using 1, 3: let $t \equiv \overline{s_0} s_1 \cdots s_n$ and $\overline{x} \equiv \overline{s_0} \cdots s_n$ be partitionings. Use cases on $\overline{s_0}$ being empty or not and on $s_n$ being bachelor or well-balanced. 4: This is a corollary of 3 above in case $\overline{x}$ is bachelor. If $\overline{x}$ is well-balanced then let $t \equiv s_0 ... s_n$. If $\overline{s_0} \equiv \varepsilon$, nothing to prove, else note from 1, that $TS(\overline{s \overline{s_0}}) \equiv TS(\overline{s})TS(\overline{s_0})$.

The following lemma shows that $TS(t)$ changes all $\delta \lambda$-couples of $t$ to $\delta \lambda$-segments.

Lemma 3.10 For every subterm $t'$ of a term $t$, the following holds:

1. $TS(t')$ is well-defined.

2. If $\overline{x} \equiv (t_0 \delta) \overline{s} \lambda_v$ is a subsegment of $t'$ and $(t_0 \delta)$ matches $\lambda_v$, then $TS(\overline{x}) = TS(\overline{s})TS(t_0 \delta)(\lambda_v)$.

3. If $\overline{x} \equiv (t_0 \omega_1) \cdots (t_n \omega_n)$ is a bachelor subsegment of $t'$, then $TS(\overline{x}) \equiv TS(t_0 \omega_1) \cdots TS(t_n \omega_n)$ is a bachelor subsegment of $TS(t')$.

4. If $\overline{x}$ is a subsegment of $t'$ which is well-balanced, then $TS(\overline{x})$ is well-balanced.
Proof: By induction on $t$.

- Case $t \equiv v$ then $t$ is the unique subterm of $t$, and all 1...4 hold.

- Assume $t \equiv (t_1 \omega) t_2$ where III holds for $t_1$ and $t_2$. Let $t'$ be a subterm of $t$. If $t'$ is a subterm of $t_1$ or $t_2$ then use III. If $t' \equiv t$ then:
  - Case $(t_1 \omega)$ is bachelor then $TS(t) \equiv \text{Lemma 3.9} (TS(t_1) \omega) TS(t_2)$. Here all 1...4 hold by III on $t_1$ and $t_2$.
  - Case $(t_1 \delta)$ matches $(\lambda_v)$ in $t$. i.e., \( t \equiv (t_1 \delta) \overline{s}(\lambda_v) t_3 \) then $TS(t) \equiv \text{Lemma 3.9} TS(s)(TS(t_1) \delta)(\lambda_v) TS(t_3)$. Now use III to show 1...4.

Lemma 3.11 For all variables $v$ and terms $t, t'$ we have:

\[
TS(t) \equiv TS(TS(t)) \quad \text{and} \quad TS(t'[v := t']) \equiv TS(TS(t))[v := TS(t')].
\]

Proof: By induction on $t$ we show that for all subterms $t''$ of $t$, $TS(t'') \equiv TS(TS(t''))$ and $TS(t'[v := t']) \equiv TS(TS(t'))[v := TS(t')]$. 

Note that if $t \rightarrow_\beta t'$ and if all the $\delta\lambda$-couples in $t$ are $\delta\lambda$-segments, then it is not necessary that all the $\delta\lambda$-couples of $t'$ are $\delta\lambda$-segments. In other words, we can have $TS(t_1) \rightarrow_\beta t_2$ where $t_2 \not\equiv TS(t_2)$. For example, $(x_1 \delta)(x_2 \delta)((\lambda_x) \epsilon \delta)(\lambda_{x_2}) x_3 \rightarrow_\beta (x_1 \delta)(x_2 \delta)(\lambda_{x_2}) x_4$. Following this remark, we show that in a sense, term reshuffling preserves $\beta$-reduction.

Lemma 3.12 If $t, t' \in \Lambda$ and $t \sim_\beta t'$ then $(\exists t'')[TS(t) \rightarrow_\beta t'' \land TS(t'') \equiv TS(t')]$. In other words, the following diagram commutes:

\[
\begin{array}{ccc}
T S & \longrightarrow_\beta & T S \\
T S(t) & \longrightarrow & T S(t'') \\
T S(t''[v := t']) & \equiv & T S(t')
\end{array}
\]

Proof: By induction on the general $\sim_\beta$. As the compatibility case is easy, we will only consider the case where $t \equiv \overline{s}(t_1 \delta) \overline{s}(\lambda_v) t_3 \sim_\beta t' \equiv \overline{s} \overline{s}_3[v := t_1]$. Here, we use induction on the number $n$ of bachelor $\delta$-items of $\overline{s}$ that are partnered in $t_3$. Recall that $\overline{s}$ is well-balanced.

- Case $n = 0$ then

\[
\begin{align*}
TS(\overline{s}(t_1) \overline{s}(\lambda_v) t_3) & \equiv \text{Lemma 3.9} \ (4) \\
TS(\overline{s}TS((t_1) \overline{s}(\lambda_v)) TS(t_3)) & \equiv \text{Lemma 3.9} \ (2.4) \\
TS(\overline{s}TS(\overline{s}(TS(t_1) \delta)(\lambda_v) TS(t_3)) & \equiv \overline{s} \overline{s}_3[v := t_1] \\
TS(t'') & \equiv \text{Lemma 3.11} \\
TS(\overline{s}TS(\overline{s}_3[v := t_1])) & \equiv \end{align*}
\]
Corollary 3.13 If \( t \sim_{\beta} t' \) then there exists exists \( t_1, t_2, \ldots, t_n \) such that:
\[
[(TS(t) \rightarrow_{\beta} t_1) \land (TS(t_1) \rightarrow_{\beta} t_2) \land \cdots \land (TS(t_{n-1}) \rightarrow_{\beta} t_n) \land (TS(t_n) \equiv TS(t'))].
\]

Proof: By induction on \( \sim_{\beta} \).

- Case \( t \sim_{\beta} t' \) use Lemma 3.12.
- Case \( t \sim_{\beta} t \) then obvious \( (n = 1 \land t_1 \equiv TS(t)) \).
- Case \( t \sim_{\beta} t' \land t' \sim_{\beta} t'' \), then by III, there exists \( t_1, t_2, \ldots, t_n, t_1', t_2', \ldots, t_m' \) such that
  \[
  (TS(t') \rightarrow_{\beta} t_1') \land (TS(t_1') \rightarrow_{\beta} t_2') \land \cdots \land (TS(t_{n-1}') \rightarrow_{\beta} t_n') \land (TS(t_n') \equiv TS(t'')).
  \]
  Hence, \( (TS(t') \rightarrow_{\beta} t_1') \land \cdots \land (TS(t_{n-1}') \rightarrow_{\beta} t_n') \land (TS(t_n') \equiv TS(t'')) \land \sim_{\beta} t_1 \rightarrow_{\beta} t_2 \land \sim_{\beta} t_2 \rightarrow_{\beta} t_3 \land \cdots \land \sim_{\beta} t_n \rightarrow_{\beta} t_{n+1} \land (TS(t_{n+1}') \equiv TS(t'')) \).

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, \( t \equiv ((\lambda x_1)(\lambda x_2)(\lambda x_3)(x_1 \delta)(x_1 \delta)(x_2 \delta)(x_2 \delta)(x_3 \delta)(x_3 \delta)(x_4 \delta)(x_4 \delta)(x_5 \delta)(x_5 \delta)(x_6 \delta)(x_6 \delta)(x_7 \delta)(x_7 \delta) \sim_{\beta} t' \equiv (\lambda x_1)(x_1 \delta)(\lambda x_2)(x_1 \delta)(\lambda x_3)(x_1 \delta)(\lambda x_4)(x_1 \delta)(\lambda x_5)(x_1 \delta)(\lambda x_6)(x_1 \delta)(\lambda x_7)(x_1 \delta) \) and does not satisfy Lemma 3.12. There is however \( t_1 \) and \( t_2 \) such that \( t_1 \equiv ((\lambda x_1)(x_1 \delta)(x_1 \delta)(\lambda x_2)(\lambda x_3) x_2 \) and \( t_2 \equiv t' \) such that \( TS(t) \rightarrow_{\beta} t_1 \land TS(t_1) \rightarrow_{\beta} t_2 \land \sim_{\beta} t_2 \rightarrow_{\beta} t_3 \land \sim_{\beta} t_3 \rightarrow_{\beta} t_4 \land \cdots \land \sim_{\beta} t_7 \rightarrow_{\beta} t_8 \land (TS(t_8') \equiv TS(t'')). \]

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 \( t \equiv ((\lambda x_1)(\lambda x_2)(\lambda x_3)(x_4)(x_5)(x_6)(x_7)(x_8) \sim_{\beta} t' \equiv (\lambda x_1)(\lambda x_2)(\lambda x_3)(x_4)(x_5)(x_6)(x_7)(x_8) \) we may want to substitute \( x_1 \) for \( x_3 \) in \( \lambda x_4(\lambda x_5)(x_4)(x_5) x_2 \) before \( x_3 \) has been substituted for \( x_4 \) and \( x_5 \) has been substituted for \( x_7 \). i.e. we need to carry out the reduction which substitutes \( x_2 \) for \( x_8 \) while the other two reductions which substitute \( x_3 \) and \( x_2 \) for \( x_8 \) and \( x_7 \) respectively are frozen.
2. Having a term like \( t \) above, we want to be able to discuss its \textit{needed} redexes for reasons that are explained in [BKKs 87]. In fact, the needed redexes in a term \( t' \) as introduced in that paper are those redexes that are contracted in every reduction of \( t' \) 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 \( \beta \)-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 \( \beta \)-reduction. Both the generalised reduction and the reshuffling of the term are difficult to describe in the classical notation.

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 [NK 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 92].

References


<table>
<thead>
<tr>
<th>Computing Science Notes</th>
<th>Department of Mathematics and Computing Science</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>In this series appeared:</strong></td>
<td><strong>Eindhoven University of Technology</strong></td>
</tr>
<tr>
<td>91/02 R.P. Nederpelt H.C.M. de Swart</td>
<td>Implication. A survey of the different logical analyses &quot;if...then...&quot;, p. 26.</td>
</tr>
<tr>
<td>91/03 J.P. Katoen L.A.M. Schoenmakers</td>
<td>Parallel Programs for the Recognition of P-invariant Segments, p. 16.</td>
</tr>
<tr>
<td>91/05 D. de Reus</td>
<td>An Implementation Model for GOOD, p. 18.</td>
</tr>
<tr>
<td>91/06 K.M. van Hee</td>
<td>SPECIFICATIEMETHODEN, een overzicht, p. 20.</td>
</tr>
<tr>
<td>91/07 E. Poll</td>
<td>CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.</td>
</tr>
<tr>
<td>91/12 E. van der Sluis</td>
<td>A parallel local search algorithm for the travelling salesman problem, p. 12.</td>
</tr>
<tr>
<td>91/14 P. Lemmens</td>
<td>The PDB Hypermedia Package. Why and how it was built, p. 63.</td>
</tr>
<tr>
<td>91/16 A.J.J.M. Marcelis</td>
<td>An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.</td>
</tr>
<tr>
<td>No.</td>
<td>Authors</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>91/18</td>
<td>Rik van Geldrop</td>
</tr>
<tr>
<td>91/19</td>
<td>Erik Poll</td>
</tr>
<tr>
<td>91/23</td>
<td>K.M. van Hee, L.J. Somers, M. Voorhoeve</td>
</tr>
<tr>
<td>91/24</td>
<td>A.T.M. Aerts, D. de Reus</td>
</tr>
<tr>
<td>91/25</td>
<td>P. Zhou, J. Hooiman, R. Kuiper</td>
</tr>
<tr>
<td>91/27</td>
<td>F. de Boer, C. Palamidessi</td>
</tr>
<tr>
<td>91/28</td>
<td>F. de Boer</td>
</tr>
<tr>
<td>91/30</td>
<td>J.C.M. Baeten, F.W. Vaandrager</td>
</tr>
<tr>
<td>91/31</td>
<td>H. ten Eikelder</td>
</tr>
<tr>
<td>91/33</td>
<td>W. v.d. Aalst</td>
</tr>
<tr>
<td>91/34</td>
<td>J. Coenen</td>
</tr>
<tr>
<td>Volume</td>
<td>Authors</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>92/01</td>
<td>J. Coenen, J. Zwiers, W.-P. de Roever</td>
</tr>
<tr>
<td>92/02</td>
<td>J. Coenen, J. Hooman</td>
</tr>
<tr>
<td>92/03</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
</tr>
<tr>
<td>92/05</td>
<td>J.P.H.W.v.d.Eijnde</td>
</tr>
<tr>
<td>92/06</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
</tr>
<tr>
<td>92/07</td>
<td>R.P. Nederpelt</td>
</tr>
<tr>
<td>92/10</td>
<td>P.M.P. Rambags</td>
</tr>
<tr>
<td>92/13</td>
<td>F. Kamareddine</td>
</tr>
<tr>
<td>92/14</td>
<td>J.C.M. Baeten</td>
</tr>
<tr>
<td>92/15</td>
<td>F. Kamareddine</td>
</tr>
<tr>
<td>92/16</td>
<td>R.R. Seljoe</td>
</tr>
<tr>
<td>92/17</td>
<td>W.M.P. van der Aalst</td>
</tr>
<tr>
<td>92/18</td>
<td>R.Nederpelt, F. Kamareddine</td>
</tr>
<tr>
<td>92/20</td>
<td>F.Kamareddine</td>
</tr>
</tbody>
</table>
92/21 F.Kamareddine
Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.

92/22 R. Nederpel
F.Kamareddine
A useful lambda notation, p. 17.

92/23 F.Kamareddine
E.Klein
Nominalization, Predication and Type Containment, p. 40.

92/24 M.Codish
D.Dams
Eyal Yardeni
Bottom-up Abstract Interpretation of Logic Programs, p. 33.

92/25 E.Poll
A Programming Logic for FOP, p. 15.

92/26 T.H.W.Beelen
W.J.J.Stut
P.A.C.Verkoulen
A modelling method using MOVIE and SimCon/ExSpect, p. 15.

92/27 B. Watson
G. Zwaan
A taxonomy of keyword pattern matching algorithms, p. 50.

93/01 R. van Geldrop
Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.

93/02 T. Verhoeff
A continuous version of the Prisoner’s Dilemma, p. 17

93/03 T. Verhoeff
Quicksort for linked lists, p. 8.

93/04 E.H.L. Aarts
J.H.M. Korst
P.J. Zwietering
Deterministic and randomized local search, p. 78.

93/05 J.C.M. Baeten
C. Verhoeof
A congruence theorem for structured operational semantics with predicates, p. 18.

93/06 J.P. Veltkamp
On the unavoidability of metastable behaviour, p. 29

93/07 P.D. Moerland
Exercises in Multiprogramming, p. 97

93/08 J. Verhoosel
A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.

93/09 K.M. van Hee

93/10 K.M. van Hee
Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.

93/11 K.M. van Hee

93/12 K.M. van Hee

93/13 K.M. van Hee


A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27.

Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.

A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.

The Design of an Online Help Facility for ExSpect, p. 21.


A Typechecker for Bijective Pure Type Systems, p. 28.

Relational Algebra and Equational Proofs, p. 23.

Pure Type Systems with Definitions, p. 38.


Multi-dimensional Petri nets, p. 25.

Finding all minimal separators of a graph, p. 11.

A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.

GOLD, a Graph Oriented Language for Databases, p. 42.

On Vertex Ranking for Permutation and Other Graphs, p. 11.

Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.

93/33 L. Loyens and J. Moonen  
ILIAS, a sequential language for parallel matrix computations, p. 20.

93/34 J.C.M. Baeten and J.A. Bergstra  
Real Time Process Algebra with Infinitesimals, p. 39.

93/35 W. Ferrer and P. Severi  
Abstract Reduction and Topology, p. 28.

93/36 J.C.M. Baeten and J.A. Bergstra  
Non Interleaving Process Algebra, p. 17.

93/37 J. Brunekreef  
Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.

93/38 J-P. Katoen  R. Koymans  S. Mauw  
A general conservative extension theorem in process algebra, p. 17.

93/39 W.P.M. Nuijten  E.H.L. Aarts  D.A.A. van Erp Taalman Kip  K.M. van Hee  
Job Shop Scheduling by Constraint Satisfaction, p. 22.

93/40 P.D.V. van der Stok  M.M.M.P.J. Claessen  D. Alstein  

93/41 A. Bijlsma  
Temporal operators viewed as predicate transformers, p. 11.

93/42 P.M.P. Rambags  
Automatic Verification of Regular Protocols in P/T Nets, p. 23.

93/43 B.W. Watson  
A taxonomy of finite automata construction algorithms, p. 87.

93/44 B.W. Watson  
A taxonomy of finite automata minimization algorithms, p. 23.

93/45 E.J. Luit  J.M.M. Martin  
A precise clock synchronization protocol, p.

93/46 T. Kloks  D. Kratsch  J. Spinrad  

93/47 W. v.d. Aalst  P. De Bra  G.J. Houben  Y. Komatzky  

93/48 R. Gerth  
Verifying Sequentially Consistently Memory using Interface Refinement, p. 20.
<table>
<thead>
<tr>
<th>Paper No.</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>94/01</td>
<td>P. America, M. van der Kammen, R.P. Nederpelt, O.S. van Roosmalen, H.C.M. de Swart</td>
<td>The object-oriented paradigm, p. 28.</td>
</tr>
<tr>
<td>94/02</td>
<td>F. Kamarcddine, R.P. Nederpelt</td>
<td>Canonical typing and Π-conversion, p. 51.</td>
</tr>
<tr>
<td>94/04</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.</td>
</tr>
<tr>
<td>94/06</td>
<td>T. Basten, T. Kunz, J. Black, M. Coffin, D. Taylor</td>
<td>Time and the Order of Abstract Events in Distributed Computations, p. 29.</td>
</tr>
<tr>
<td>94/08</td>
<td>O.S. van Roosmalen</td>
<td>A Hierarchical Diagrammatic Representation of Class Structure, p. 22.</td>
</tr>
<tr>
<td>94/09</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Process Algebra with Partial Choice, p. 16.</td>
</tr>
<tr>
<td>94/10</td>
<td>T. verhoeff</td>
<td>The testing Paradigm Applied to Network Structure. p. 31.</td>
</tr>
<tr>
<td>94/13</td>
<td>R. Seljée</td>
<td>A New Method for Integrity Constraint checking in Deductive Databases, p. 34.</td>
</tr>
<tr>
<td>94/14</td>
<td>W. Peremans</td>
<td>Ups and Downs of Type Theory, p. 9.</td>
</tr>
<tr>
<td>94/16</td>
<td>R.C. Backhouse, H. Doornbos</td>
<td>Mathematical Induction Made Calculational, p. 36.</td>
</tr>
</tbody>
</table>