Beyond Beta-reduction in Church's lambda-arrow

Bloo, C.J.; Kamareddine, F.; Nederpelt Lazarom, R.P.

Published: 01/01/1994

Citation for published version (APA):
Eindhoven University of Technology

Department of Mathematics and Computing Science

Beyond β-Reduction in Church’s λ→

by

R. Bloo, F. Kamareddine and R. Nederpelt

94/20

Computing Science Note 94/20
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.
Beyond $\beta$-Reduction in Church's $\lambda \rightarrow^*$

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

and

Fairouz Kamareddine †
Department of Computing Science
17 Lilybank Gardens
University of Glasgow
Glasgow G12 8QQ, Scotland
email: fairouz@dcs.glasgow.ac.uk
fax + 44 41 -3304913

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 26, 1994

---

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

†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 various short visits in 1993 and 1994.
Abstract

In this paper, we shall write \( \lambda \) using a notation, item notation, which enables one to make more redexes visible, and shall extend \( \beta \)-reduction to all visible redexes. We will prove that \( \lambda \) written in item notation and accommodated with extended reduction, satisfies all its original properties (such as Church Rosser, Subject Reduction and Strong Normalisation). The notation itself is very simple: if \( I \) translates classical terms to our notation, then \( I(t_1t_2) \equiv (I(t_1)\delta)I(t_2) \) and \( I(\lambda e^p.t) \equiv (\rho\lambda e)I(t) \). For example, \( t \equiv ((\lambda x_7.x_4. (\lambda x_6.x_3.\lambda x_5.x_1 \rightarrow x_2.x_3x_4)x_3)x_2)x_1 \), can be written in our item notation as \( I(t) \equiv (x_1\delta)(x_2\delta)(X_4\lambda x_7)(x_3\delta)(X_3\lambda x_5)((X_1 \rightarrow X_2)\lambda x_2)(x_4\delta)x_5 \) where the visible redexes are based on all the matching \( \delta \lambda \)-couples. So here, the redexes are based on \((x_2\delta)(X_4\lambda x_7), (x_3\delta)(X_3\lambda x_5)\) and \((x_1\delta)((X_1 \rightarrow X_2)\lambda x_2)\). In classical notation however, only the redexes based on \((\lambda x_7.x_4. \vdash x_2)\) and \((\lambda x_6.x_3. \vdash x_3)\) are immediately visible. The third redex, \((\lambda x_5.x_1 \vdash x_2)\), only becomes visible when the first two redexes have been contracted. We extend \( \beta \)-reduction so that we can contract newly visible redexes even before other redexes have been contracted. So in our example above, \((x_1\delta)((X_1 \rightarrow X_2)\lambda x_2)\) can be contracted before \((x_2\delta)(X_4\lambda x_7)\) or \((x_3\delta)(X_3\lambda x_5)\). This refinement (which cannot be done in classical notation) is achieved by generalising the axiom \( \beta \) from \((t_1\delta)(\rho\lambda e)t_2 \rightarrow_\beta t_2[v := t_1] \) to \((t_1\delta)\overline{s}(\rho\lambda e)t_2 \rightarrow_\beta \overline{s}(t_2[v := t_1]) \) for \( \overline{s} \) consisting of matching \( \delta \lambda \)-couples only. Hence, as \((x_2\delta)(X_4\lambda x_7)(x_3\delta)(X_3\lambda x_5)\) consists of matching \( \delta \lambda \)-couples, we get that \( I(t) \rightarrow_\beta \overline{s}(x_2\delta)(X_4\lambda x_7)(x_3\delta)(X_3\lambda x_5)((x_4\delta)x_5)[x_5 := x_1] \).

Furthermore, with our item notation, it is possible to refine reduction by rewriting (or reshuffling) terms so that matching \( \delta \lambda \)-couples occur adjacent to each other. For example, we can rewrite \( I(t) \) above as \((x_2\delta)(X_4\lambda x_7)(x_3\delta)(X_3\lambda x_5)(x_1\delta)((X_1 \rightarrow X_2)\lambda x_2)(x_4\delta)x_5 \). We shall formalise term reshuffling and shall show that it is correct and preserves both \( \beta \)-reduction and typing.

**Keywords:** \( \beta \)-reduction, Church Rosser, Subject Reduction, Strong Normalisation.
Contents

1 Introduction ........................................... 4

2 \( \lambda_\_ \) in item notation ............................. 6
   2.1 The basic theory .................................. 6
   2.2 Properties of \( \lambda_\_ \) .......................... 8

3 Generalising reduction .................................. 9
   3.1 Extending redexes and \( \beta \)-reduction ............... 9
   3.2 Properties of \( \lambda_\_ \) with generalised reduction .. 11

4 Term reshuffling ........................................ 16
   4.1 Partitioning the term into bachelor and well-balanced segments 17
   4.2 The reshuffling procedure and its properties ............ 18

5 Conclusion ............................................. 21
1 Introduction

The notation of this paper, item notation, is a novel notation where the argument is given before the function, the type is given before the function, and where the parentheses are grouped differently than those of the classical notation. So that, if \( I \) translates classical terms into our notation, then \( (I(t_2)\delta)I(t_1) \) is written as \( (\rho\lambda_\varphi)I(t) \). Both \( (t\delta) \) and \( (\rho\lambda_\varphi) \) are called items.

Example 1.1 \( I((\lambda_\omega: X_1 \rightarrow (X_2 \rightarrow X_3)) \cdot \lambda_{\varphi: X_1 \cdot xy} \cdot z) \equiv (z\delta)(X_1 \rightarrow (X_2 \rightarrow X_3)\lambda_\varphi)(X_1\lambda_\varphi)(y\delta)z \). The items are \( (z\delta) \), \( (X_1 \rightarrow (X_2 \rightarrow X_3)\lambda_\varphi) \), and \( (X_1\lambda_\varphi) \).

Before we discuss the calculus and the properties of typing, let us see why we want to extend the notion of a redex and to refine \( \beta \)-reduction.

In the classical \( \lambda \)-calculus, the notions of redex, and of \( \beta \)-reduction are described as follows:

Definition 1.2 (Redexes and \( \beta \)-reduction in classical notation)
In the classical notation of the \( \lambda \)-calculus, a redex is of the form \((\lambda_{\psi: \varphi} t)t'\). Moreover, one-step \( \beta \)-reduction \( \rightarrow_\beta \) is the compatible relation generated out of the axiom \( \beta : (\lambda_{\psi: \varphi} t)t' \rightarrow_\beta t[v := t'] \). Many step \( \beta \)-reduction \( \rightarrow^*_\beta \) is the reflexive transitive closure of \( \rightarrow_\beta \).

With our item notation, classical redexes and \( \beta \)-reduction take the following form:

Definition 1.3 (Classical redexes and \( \beta \)-reduction in item notation)
In the item notation of the \( \lambda \)-calculus, a classical redex is of the form \((t'\delta)(p\lambda_\varphi)t\). We call the pair \((t'\delta)(p\lambda_\varphi)\) a \( \delta\lambda \)-pair, or a \( \delta\lambda \)-segment. The classical \( \beta \)-reduction axiom is: \((t'\delta)(p\lambda_\varphi)t \rightarrow_\beta t[v := t'] \). One and many step \( \beta \)-reduction are defined as in Definition 1.2.

Example 1.4 In the classical term \( t \equiv ((\lambda_{x_7: X_4} \cdot (\lambda_{x_6: X_3} \cdot \lambda_{x_5: X_1 \rightarrow X_2} \cdot x_5 x_4)) x_3) x_1 \), 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. \((\lambda_{x_5: X_3} \cdot \lambda_{x_5: X_1 \rightarrow X_2} \cdot x_5 x_4) x_3\)
2. \((\lambda_{x_7: X_4} \cdot (\lambda_{x_6: X_3} \cdot \lambda_{x_5: X_1 \rightarrow X_2} \cdot x_5 x_4)) x_2\)

In item notation \( t \) becomes \((x_1\delta)(x_2\delta)(X_4\lambda_{x_7})(x_3\delta)(X_3\lambda_{x_6})(X_1 \rightarrow X_2)\lambda_{x_3}(x_4\delta) x_5\). Here, the two classical redexes correspond to \( \delta\lambda \)-pairs as follows:

1. \((\lambda_{x_6: X_3} \cdot \lambda_{x_5: X_1 \rightarrow X_2} \cdot x_5 x_4) x_3\) corresponds to \((x_3\delta)(X_3\lambda_{x_6})\). \((X_1 \rightarrow X_2)\lambda_{x_3}(x_4\delta) x_5\) is ignored as it is easily retrievable in item notation. It is the maximal subterm of \( t \) to the right of \((X_3\lambda_{x_6})\).

2. \((\lambda_{x_7: X_4} \cdot (\lambda_{x_6: X_3} \cdot \lambda_{x_5: X_1 \rightarrow X_2} \cdot x_5 x_4)) x_2\)
Again \((x_3\delta)(X_3\lambda_{x_6})(X_1 \rightarrow X_2)\lambda_{x_3}(x_4\delta) x_5\) is ignored for the same reason as above.

There is however a third redex which is not immediately visible in the classical term; namely, \((\lambda_{x_5: X_1 \rightarrow X_2} \cdot x_5 x_4) x_1\). 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.

\[
((\lambda_{x_7: X_4} \cdot (\lambda_{x_6: X_3} \cdot \lambda_{x_5: X_1 \rightarrow X_2} \cdot x_5 x_4)) x_2) x_1 \rightarrow_\beta \\
((\lambda_{x_6: X_3} \cdot \lambda_{x_5: X_1 \rightarrow X_2} \cdot x_5 x_4)) x_3 \rightarrow_\beta \\
(\lambda_{x_5: X_1 \rightarrow X_2} \cdot x_5 x_4) x_1 \rightarrow_\beta x_1 x_4
\]
Now, even though all these three redexes 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 had contracted the first two redexes. In item notation, the third redex \((\lambda x_5 : X_1 \rightarrow X_2 : x_5 x_4) x_1\) is visible as it corresponds to the matching \((x_1 \delta)((X_1 \rightarrow X_2) \lambda x_2)\) where \((x_1 \delta)\) and \((X_1 \rightarrow X_2) \lambda x_2\) are separated by the segment \((x_2 \delta)(X_4 \lambda x_7)(x_3 \delta)(X_3 \lambda x_9)\). Hence, by extending the notion of a redex from being a \( \delta \)-item adjacent to a \( \lambda \)-item, to being a matching pair of \( \delta \)- and \( \lambda \)-items, we can make more redexes visible. This extension furthermore is simple, as in \((t_1 \delta)s(\rho \lambda_u)\), we say that \((t_1 \delta)\) and \((\rho \lambda_u)\) match if \( s \) 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. For example, in \( t \) above, \((x_1 \delta)\) and \((X_1 \rightarrow X_2) \lambda x_3\) match as \((x_2 \delta)(X_4 \lambda x_7)(x_3 \delta)(X_3 \lambda x_9)\) has the bracketing structure \([][]\) (see Figure 1 which is drawn ignoring types just for the sake of argument). With this

![Redexes in item notation](image)

Figure 1: Redexes in item notation

extension of redexes, we refine \( \beta \)-reduction in two different ways:

1. By changing \( \beta \) from \((t_1 \delta)(\rho \lambda_u)t_2 \rightarrow_\beta t_2[v := t_1]\) to \((t_1 \delta)s(\rho \lambda_u)t_2 \rightarrow_\beta s(t_2[v := t_1])\) if \((t_1 \delta)\) and \((\rho \lambda_u)\) match.

2. By reshuffling terms so that matching \( \delta \)'s and \( \lambda \)'s occur adjacently.

We start by showing that \( \rightarrow _\beta \) (the reflexive transitive closure of \( \rightarrow _\beta \)) is a generalisation of \( \rightarrow _\beta \) (Lemma 3.7). We will then show that \( \lambda_\_ \) with \( \rightarrow _\beta \) satisfies all the desirable typing properties. In particular, we will establish that \( \lambda_\_ \) extended with \( \rightarrow _\beta \) satisfies the following:

1. Church Rosser: this says that if a program is evaluated in two different ways, then the answer stays the same (Theorem 3.10).

2. Subject Reduction: this says that if a program \( P \) is well-typed then the program obtained from evaluating some steps in \( P \) is also well-typed (Theorem 3.13).

3. Unicity of Types: this says that a well-typed program has a unique type and that two equal programs have the same type (Lemma 3.15).

4. Strong Normalisation: this says that all ways of evaluating a well-typed program terminate (Theorem 3.21).

We will furthermore show that term reshuffling is correct. In particular, we shall show that \( \lambda_\_ \) accommodated with term reshuffling \( TS \), satisfies the following:

1. Reshuffling a term, moves all \( \delta \)'s next to their matching \( \lambda \)'s (Lemma 4.9).
2. Reshuffling terms preserves $\rightarrow_\beta$. That is, if $t \sim_\beta t'$ then there exists $t''$ such that $TS(t) \rightarrow_\beta t''$ and $TS(t') \equiv TS(t'')$ (Lemma 4.11).

3. Reshuffling terms preserves types. That is, if $\Gamma \vdash t : \rho$ then $\Gamma \vdash TS(t) : \rho$ (Lemma 4.13).

2 $\lambda_-$ in item notation

In this section, we shall introduce the known $\lambda_-$ (which uses the ordinary $\beta$-reduction $\rightarrow_\beta$), and its properties. We shall write $\lambda_-$ immediately in item notation. That is, we assume a translation function $I$ from terms in classical notation to terms in item notation such that:

$I(v) = v$ if $v$ is a variable
$I(\lambda_v : p.t) = (\rho \lambda_v)I(t)$
$I(t_1 t_2) = (I(t_2)\rho)I(t_1)$

2.1 The basic theory

In Church's system $\lambda_-$, types and terms are defined as follows:

**Definition 2.1 (Types of $\lambda_-$)**

The set of types $T$ of $\lambda_-$ is defined as follows:

$T ::= V | (T \rightarrow T)$
$V ::= \{X_0, X_1, \ldots\}$

That is, types are either variables such as $X_0, X_1, X_2, \ldots$ or arrow types.

**Definition 2.2 (Terms of $\lambda_-$)**

The set of terms $\Lambda_T$ of $\lambda_-$ is defined as follows:

$\Lambda_T ::= V | (T \lambda V)\Lambda_T | (\Lambda_T \rho)\Lambda_T$
$V ::= \{x_0, x_1, \ldots\}$

In other words, a term is either a variable $x_0, x_1, x_2, \ldots$, or an abstraction or an application.

**Notation 2.3** We let $\rho, \rho', \rho_1, \rho_2, \ldots$ range over types, $\alpha, \alpha', \alpha_1, \alpha_2, \ldots$ range over type variables. Furthermore, we take $t, t', t_1, t_2, \ldots$ to range over terms and let $v, v', v_1, \ldots$ range over variables.

Parentheses moreover will be omitted when no confusion occurs.

We understand $\rho_1 \rightarrow \rho_2 \rightarrow \cdots \rho_n \rightarrow (\rho_{n-1} \rightarrow \rho_n) \cdots$.

Bound and free variables in $\lambda_-$ are defined as usual. We write $BV(t)$ and $FV(t)$ to represent the bound and free variables of $t$ respectively. Substitution moreover, is defined in the usual way. Furthermore, we take terms to be equivalent up to variable renaming. For example, we take $(\rho \lambda_{x_0})x_0 \equiv (\rho \lambda_{x_1})x_1$. We assume moreover, the Barendregt variable convention which is formally stated as follows:

**Definition 2.4 (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 $(v\theta)(\rho \lambda_v)v$, but $(v\theta)(\rho \lambda_{v'})v'$ instead.
Definition 2.5 (Compatibility)
We say that a relation $\rightarrow$ on terms is compatible iff the following holds:

\[
\begin{align*}
\frac{t \rightarrow t'}{(t\delta)t_1 \rightarrow (t'\delta)t_1} & \quad \frac{t \rightarrow t'}{(t_1\delta)t \rightarrow (t_1\delta)t'} \\
\frac{t \rightarrow t'}{(\rho\lambda_\nu)t \rightarrow (\rho\lambda_\nu)t'}
\end{align*}
\]

Basically compatibility means that if $t \rightarrow t'$ then $T[t] \rightarrow T[t']$ where $T[\ ]$ is a "term with a hole in it".

Definition 2.6 ($\beta$-reduction $\rightarrow_\beta$ in $\lambda_-$)
In $\lambda_-$, $\beta$-reduction $\rightarrow_\beta$, is the least compatible relation generated out of the following axiom:

\[(\beta) \quad (t_1\delta)(\rho\lambda_\nu)t \rightarrow_\beta t[v := t_1] \]

We take $\rightarrow_\beta$ to be the reflexive transitive closure of $\rightarrow_\beta$ and $=_\beta$ to be the least equivalence relation generated by $\rightarrow_\beta$.

Definition 2.7 ((main) items, (main, $\delta\lambda$-)segments, context, body, endvar, weight)

- If $v$ is a variable, $\rho$ is a type and $t$ is a term then $(\rho\lambda_\nu)$ and $(t\delta)$ are items (the first is called $\lambda$-item, the second $\delta$-item). We use $s, s_1, s_2, \ldots$ to range over items.

- A concatenation of zero or more items is a segment. We use $\exists, \exists_1, \exists_2, \ldots$ as metavariables for segments. We write $\emptyset$ for the empty segment.

- Each term $t$ is the concatenation of zero or more items and a variable: $t \equiv s_1 s_2 \ldots s_n v$. These items $s_1, s_2, \ldots, s_n$ are called the main items of $t$.

- Analogously, a segment $\exists$ is a concatenation of zero or more items: $\exists \equiv s_1 s_2 \ldots s_n$; again, these items $s_1, s_2, \ldots, s_n$ (if any) are called the main items, this time of $\exists$.

- A concatenation of adjacent main items (in $t$ or $\exists$), $s_m \ldots s_{m+k}$, is called a main segment (in $t$ or $\exists$).

- A context is a segment which consists of only $\lambda$-items. We use $\Gamma, \Gamma', \Gamma_1, \Gamma_2, \ldots$ to range over contexts.

- A $\delta\lambda$-segment is a $\delta$-item immediately followed by a $\lambda$-item.

- Let $t \equiv \exists_0$ be a term. Then we call $\exists$ the body of $t$, denoted $\text{body}(t)$, and $v$ the end variable of $t$, or $\text{endvar}(t)$. It follows that $t \equiv \text{body}(t) \text{ endvar}(t)$.

- The weight of a $\lambda_-$-segment $\exists$, weight($\exists$), is the number of main items that compose the segment. Moreover, we define weight($t$) = weight($\text{body}(t)$).

Definition 2.8 (Statements)
A statement is of the form $t : \rho$, $t$ and $\rho$ are called the subject and the type of the statement respectively.
Convention 2.9 In a context, we never have two occurrences of $\lambda_v$ (for the same $v$). Hence, contexts are what [Barendregt 92] calls bases.

We need the following definition over contexts:

**Definition 2.10** Let $\Gamma = (\rho_1 \lambda_{v_1})(\rho_2 \lambda_{v_2}) \cdots (\rho_k \lambda_{v_k})$ be a context. Then

1. $\text{dom}(\Gamma) = \{v_1, v_2, \ldots, v_k\}$

2. $(\rho \lambda_v) \in' \Gamma$ iff $(\rho \lambda_v)$ is an item of $\Gamma$. If $\Gamma'$ is a context such that all items of $\Gamma'$ are also items of $\Gamma$, we write $\Gamma' \subseteq \Gamma$.

3. Let $V_0$ be a set of term variables. $\Gamma \upharpoonright V_0$ (the restriction of $\Gamma$ to $V_0$) is the context which only contains the items $(\rho \lambda_v) \in' \Gamma$ for which $v \in V_0$, in the original order.

Now for the formulation of the typing rules we can use the following definitions for the derivation of so-called judgements of the form $\Gamma \vdash t : \rho$.

**Definition 2.11** *(Typing rules of $\lambda_\_*)* 
A statement $t : \rho$ is derivable in the context $\Gamma$, notation $\Gamma \vdash t : \rho$, if $t : \rho$ can be derived using the following rules:

- **(Axiom)** \( \Gamma \vdash v : \rho \) if $(\rho \lambda_v) \in' \Gamma$

- **($\to$ - elimination)** \( \frac{\Gamma \vdash t : \rho \quad \Gamma \vdash t' : (\rho \to \rho')} {\Gamma \vdash (t \delta)t' : \rho'} \)

- **($\to$ - introduction)** \( \frac{\Gamma(\rho \lambda_v) \vdash t : \rho'} {\Gamma \vdash (\rho \lambda_v)t : (\rho \to \rho')} \)

2.2 Properties of $\lambda_\_*$

Here we list the properties of $\lambda_\_*$ (that we will establish for extended reduction) without proofs. The reader can refer to [Barendregt 92] for details.

**Theorem 2.12** *(The Church Rosser Theorem)*
If $t \rightarrow_\beta t_1$ and $t \rightarrow_\beta t_2$ then there exists $t_3$ such that $t_1 \rightarrow_\beta t_3$ and $t_2 \rightarrow_\beta t_3$ \( \Box \)

**Lemma 2.13** *(Context lemma)*
1. \( \forall t, t', \rho \forall v : [\Gamma \subseteq \Gamma' \land \Gamma \vdash t : \rho \Rightarrow \Gamma' \vdash t : \rho] \)

2. \( \forall t, t', \rho \forall v : [\Gamma \vdash t : \rho \Rightarrow \text{FV}(t) \subseteq \text{dom}(\Gamma)] \)

3. \( \forall t, t', \rho \forall v : [\Gamma \vdash t : \rho \Rightarrow \Gamma \upharpoonright \text{FV}(t) \vdash t : \rho] \)

**Proof:** All by induction on the derivation $\Gamma \vdash t : \rho$. \( \Box \)

**Lemma 2.14** *(Generation lemma)*
1. \( \forall t, t', \rho \forall v : [\Gamma \vdash \rho \lambda_v : \rho \Rightarrow (\rho \lambda_v) \in' \Gamma] \)

2. \( \forall t, t', \rho \forall v : [\Gamma \vdash (t \delta)t : \rho \Rightarrow \exists \rho' [\Gamma \vdash (t' \delta)t : (\rho' \rightarrow \rho) \land \Gamma \vdash t' : \rho']] \)
3. \( \forall t \forall \rho \forall \rho', \rho' \left[ \Gamma \vdash (\rho' \lambda \nu) t : \rho \Rightarrow \exists \rho'' [\rho \equiv \rho' \rightarrow \rho'' \land \Gamma (\rho' \lambda \nu) \vdash t : \rho''] \right] \)

**Proof:** By induction on the derivation of \( \Gamma \vdash t : \rho \). \( \square \)

**Lemma 2.15 (Subterm lemma)**
\( \forall t \forall \rho \forall \rho' \left[ (\Gamma \vdash t : \rho \land t' \text{ is a subterm of } t) \Rightarrow \exists \rho'' [\rho' \vdash t' : \rho''] \right] \)

**Proof:** By induction on \( t \). \( \square \)

**Lemma 2.16 (Substitution lemma)**
1. \( \forall t \forall \rho \forall \alpha \in V \left[ (\Gamma \vdash \alpha := \rho' \vdash t[\alpha := \rho'] : \rho[\alpha := \rho']) \right] \)
2. \( \forall t \forall \rho \forall \rho' \left[ (\Gamma (\rho \lambda \alpha) \vdash t : \rho' \land \Gamma \vdash t' : \rho \Rightarrow \Gamma \vdash t[v := t'] : \rho'] \right] \)

**Proof:** 1: by induction on \( \Gamma \vdash t : \rho \). 2: by induction on \( \Gamma (\rho \lambda \alpha) \vdash t : \rho' \). \( \square \)

**Theorem 2.17 (Subject Reduction)**
\( \forall t \forall \rho \forall \rho' \left[ (\Gamma \vdash \rho \rightarrow \rho') \Rightarrow \Gamma \vdash t : \rho' \right] \)

**Proof:** By induction on \( \rightarrow_{\beta} \) using the Generation and Substitution lemmas for the basic case. \( \square \)

**Lemma 2.18 (Unicity of Types)**
1. \( \forall t \forall \rho \forall \rho' \left[ (\Gamma \vdash t : \rho \land \Gamma \vdash t : \rho' \Rightarrow \rho \equiv \rho') \right] \)
2. \( \forall t \forall \rho \forall \rho', \rho'' \left[ (\Gamma \vdash t : \rho \land \Gamma \vdash t : \rho' \land \Gamma \vdash t : \rho'' \Rightarrow \rho \equiv \rho'') \right] \)

**Proof:** 1 is by an easy induction on \( t \). 2 is by Church Rosser, Subject Reduction and 1. \( \square \)

**Definition 2.19 (Strongly Normalising terms with respect to \( \rightarrow_{\beta} \))**
We say that a term \( t \) is strongly normalising with respect to \( \rightarrow_{\beta} \) iff every reduction path using \( \rightarrow_{\beta} \) and starting at \( t \) terminates.

**Theorem 2.20 (Strong Normalisation with respect to \( \rightarrow_{\beta} \))**
If \( \Gamma \vdash t : \rho \) then \( t \) is strongly normalising with respect to \( \rightarrow_{\beta} \). \( \square \)

## 3 Generalising reduction

In this section we shall extend the classical notions of redexes and \( \beta \)-reduction of \( \lambda \_ \). We shall show that this extension of \( \lambda \_ \) satisfies all the listed properties in Section 2.

### 3.1 Extending redexes and \( \beta \)-reduction

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 the \( \delta \)- and \( \lambda \)-items 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 3.1 (well-balanced segments in $\lambda_-$)

- The empty segment $\emptyset$ is a well-balanced segment;
- If $s$ is a well-balanced segment, then $(\ell\delta)s(\rho\lambda_v)$ 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 $\delta$- (or $\lambda$-)item corresponding with an opening (resp. closing) bracket.

Now we can easily define what matching $\delta\lambda$-couples are, given a segment $s$. Namely, they are a main $\delta$-item and a main $\lambda$-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 3.2 (match, $\delta\lambda$- or reducible couple, partner, partnered item, bachelor item, bachelor segment)

Let $t$ be a $\lambda_-$-term. Let $s = s_1 \ldots s_n$ be a segment occurring in $t$.

- We say that $s_i$ and $s_j$ match, when $1 \leq i < j \leq n$, $s_i$ is a $\delta$-item, $s_j$ is a $\lambda$-item, and the sequence $s_{i+1}, \ldots, s_{j-1}$ forms a well-balanced segment.
- When $s_i$ and $s_j$ match, we call $s_is_j$ 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.
- All $\lambda$- (or $\delta$-)items $s_k$ in $t$ that are not partnered, are called bachelor $\lambda$- (resp. $\delta$-)items.
- A segment consisting of bachelor items only, is called a bachelor segment.
- The segment $s_{i_1} \ldots s_{i_m}$ consisting of all bachelor main $\lambda$- (or $\delta$-)items of $s$ is called the bachelor $\lambda$- (or $\delta$-)segment of $s$.

Example 3.3 In $s \equiv (\rho_1\lambda_v)(\rho_2\lambda_v)(t_1\delta)(\rho_3\lambda_v)(\rho_4\lambda_v)(t_2\delta)(t_3\delta)(t_4\delta)(\rho_5\lambda_v)(\rho_6\lambda_v)(t_5\delta)$:

- $(t_1\delta)$ matches with $(\rho_3\lambda_v)$, $(t_4\delta)$ matches with $(\rho_5\lambda_v)$ and $(t_3\delta)$ with $(\rho_6\lambda_v)$. The segments $(t_1\delta)(\rho_3\lambda_v)$ and $(t_4\delta)(\rho_5\lambda_v)$ are $\delta\lambda$-segments (and $\delta\lambda$-couples). There is another $\delta\lambda$-couple in $s$, viz. the couple of $(t_3\delta)$ and $(\rho_6\lambda_v)$.
- $(t_1\delta), (\rho_3\lambda_v), (t_3\delta), (t_4\delta), (\rho_5\lambda_v)$ and $(\rho_6\lambda_v)$, are the partnered main items of $s$. $(\rho_1\lambda_v), (\rho_2\lambda_v), (\rho_4\lambda_v), (t_2\delta)$ and $(t_5\delta)$, are bachelor items.
- $(\rho_1\lambda_v)(\rho_2\lambda_v)$ and $(\rho_4\lambda_v)(t_2\delta)$ are bachelor segments, whereas $(t_3\delta)(t_4\delta)(\rho_5\lambda_v)$ and $(t_3\delta)(t_4\delta)(\rho_3\lambda_v)(\rho_6\lambda_v)$ are non-bachelor, the latter also being a well-balanced segment.

De Bruijn uses another terminology; see e.g. [de Bruijn 93]. In his phrasing, $\delta$-items are applicators or $A$'s, and $\lambda$-items are abstractors or $T$'s. For $\delta\lambda$-segments he uses the word $AT$-pair and for $\delta\lambda$-couples he uses $AT$-couples. Void $\beta$-reduction (i.e.: the reduction $(t_1\delta)(\rho\lambda_v)t \rightarrow_\beta t$ if $v \not\in \text{FV}(t)$), he calls $AT$-removal.

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. In order to accomplish this, we change the $\beta$-reduction of Definition 2.6 to the following:
Definition 3.4 (General \( \beta \)-reduction \( \rightsquigarrow_\beta \) for \( \lambda \_ \)

General one-step \( \beta \)-reduction \( \rightsquigarrow_\beta \), is the least compatible relation generated out of the following axiom:

\[
(\text{general } \beta) \quad (t_1 \delta) \approx (\rho \lambda \nu)t \rightsquigarrow_\beta \exists (t[v:= t_1]) \quad \text{ if } \exists \text{ is well-balanced}
\]

General \( \rightsquigarrow_\beta \) is the reflexive and transitive closure of \( \rightsquigarrow_\beta \) and \( \approx_\beta \) is the least equivalence relation generated by \( \rightsquigarrow_\beta \).

Example 3.5 Take Example 1.4. As \((x_2 \delta)(X_4 \lambda x_7)(x_3 \delta)(X_3 \lambda x_6)\) is a well-balanced segment, then \((x_1 \delta)((X_1 \rightarrow X_2) \lambda x_5)\) is a \( \delta \lambda \)-couple and

\[
t \equiv (x_1 \delta)(x_2 \delta)(X_4 \lambda x_7)(x_3 \delta)(X_5 \lambda x_6)((X_1 \rightarrow X_2) \lambda x_5)(x_4 \delta)x_5 \rightsquigarrow_\beta
\]

The reducible couple \((x_1 \delta)((X_1 \rightarrow X_2) \lambda x_5)\) also has a corresponding ("generalised") redex in the traditional notation, which will appear after two one-step \( \beta \)-reductions, leading to \((\lambda x_5 x_1 \rightarrow x_2 x_5 x_4)x_1\). With \( \rightsquigarrow_\beta \), we could reduce \((\lambda x_7 x_4 \lambda x_3 \lambda x_1 \rightarrow x_2 x_5 x_4)x_3\) to \((\lambda x_7 x_4 \lambda x_3 \lambda x_1 \rightarrow x_2 x_5 x_4)x_3\). 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 [Nederpel 73]) can only be obtained tidily in a system formulated using some form of 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 3.6 The "bracketing structure" of \((\lambda x_7 x_4 \lambda x_3 \lambda x_1 \rightarrow x_2 x_5 x_4)x_3\) is compatible with \('1 [2 [3 2] 1] 3'\), where \('i'\) and \('j'\) match. In item notation however, it has the bracketing structure \(['[1][2]'\]).

We strongly believe that it is the item notation which enables us to extend reduction smoothly beyond \( \rightsquigarrow_\beta \). 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 3.7 Let \( t_1, t_2 \) be \( \lambda \_ \)-terms. If \( t_1 \rightsquigarrow_\beta t_2 \) in the sense of Definition 2.6, then \( t_1 \approx_\beta t_2 \) in the sense of Definition 3.4. Moreover, if \( t_1 \approx_\beta t_2 \) comes from contracting a \( \delta \lambda \)-segment then \( t_1 \rightsquigarrow_\beta t_2 \).

Proof: Obvious as a \( \delta \lambda \)-segment is a \( \delta \lambda \)-couple. \( \square \)

3.2 Properties of \( \lambda \_ \) with generalised reduction

If we look at Section 2.2, we see that Context, Generation, Subterm, Substitution and Unicity of Types (part 1) lemmas are not affected by our extension of Reduction. Hence, they all still hold for \( \lambda \_ \) with \( \approx_\beta \). The only three (and very important) properties that get affected by \( \approx_\beta \) are: Church Rosser (Theorem 2.12), Subject Reduction (Theorem 2.17), Unicity of Types (part 2, Lemma 2.18), and Strong Normalisation (Theorem 2.20). In this section, we shall show that these properties are preserved for \( \approx_\beta \).

The proof of the generalised Church Rosser theorem is simple. The idea is to show that if \( t \approx_\beta t' \) then \( t \equiv_\beta t' \) and to use the Church Rosser property for \( \equiv_\beta \).
Lemma 3.8 If \( t \leadsto_\beta t' \) then \( t \equiv_\beta t' \).

Proof: It suffices to consider the case \( t \equiv \overline{s(t_1 \delta)} \overline{s(\rho \lambda_v) t_2} \) where the contracted redex is based on \((t_1 \delta)(\rho \lambda_v), t' \equiv \overline{s_1(t_2[v := t_1])}, \) and \( \overline{s} \) is well-balanced (hence weight(\( \overline{s} \)) is even). We shall prove the lemma by induction on weight(\( \overline{s} \)).

- Case weight(\( \overline{s} \)) = 0 then obvious as \( \leadsto_\beta \) coincides with \( \rightarrow_\beta \) in this case.

- Assume the property holds when weight(\( \overline{s} \)) = 2n. Take \( \overline{s} \) such that weight(\( \overline{s} \)) = 2n + 2. Now, \( \overline{s} \equiv (t_3 \delta) \overline{s}(\rho \lambda_{v'} \rho' \lambda_v) \overline{s''} \) where \( \overline{s'}, \overline{s''} \) are well-balanced. Assume \( v \neq v' \) (if necessary, use renaming).

  - As \( \overline{s}(t_2[v := t_1]) \leadsto_\beta \overline{s'}(\overline{s''}(t_2[v := t_1])[v' := t_3]) \), we get by IH and compatibility that \( t' \equiv_\beta \overline{s}(\overline{s''}(t_2[v := t_1])[v' := t_3])(t_2[v := t_1][v' := t_3]) \equiv t \).

  - Moreover, \( t \equiv \overline{s_1(t_1 \delta)} (t_3 \delta) \overline{s}(\rho' \lambda_v \rho \lambda_v) t_2 \leadsto_\beta \overline{s_1(t_1 \delta)} \overline{s''}(\overline{s''}(\rho \lambda_v) t_2[v' := t_3]) \equiv BC \overline{s_1(t_1 \delta)} \overline{s''}(\overline{s''}(v' := t_3))(\rho \lambda_v)(t_2[v' := t_3]) \equiv t'' \). Hence by IH, \( t \equiv_\beta t'' \).

  - Now, \( t'' \leadsto_\beta \overline{s_1(s''(v' := t_3))}(t_2[v := t_3])[v := t_1] \). But by BC, \( v, v' \notin FV(t_1) \cup FV(t_2) \). Hence, by IH and substitution, \( t'' \equiv_\beta \overline{s_1(s''(v' := t_3))}(t_2[v := t_3])[v := t_1][v' := t_3]) \equiv t''' \).

Therefore, \( t \equiv_\beta t'', t'' \equiv_\beta t''' \) and \( t' \equiv_\beta t''' \), hence \( t \equiv_\beta t' \).

Corollary 3.9 If \( t \equiv_\beta t' \) then \( t \equiv_\beta t'' \).

Theorem 3.10 (The general Church Rosser theorem)
If \( t \equiv_\beta t_1 \) and \( t \equiv_\beta t_2 \), then there exists \( t_3 \) such that \( t_1 \equiv_\beta t_3 \) and \( t_2 \equiv_\beta t_3 \).

Proof: As \( t \equiv_\beta t_1 \) and \( t \equiv_\beta t_2 \) then by Corollary 3.9, \( t \equiv_\beta t_1 \) and \( t \equiv_\beta t_2 \). Hence, \( t_1 \equiv_\beta t_2 \) and by the Church Rosser property for the classical lambda calculus, there exists \( t_3 \) such that \( t_1 \equiv_\beta t_3 \) and \( t_2 \equiv_\beta t_3 \). But, \( t' \equiv_\beta t'' \) implies \( t' \equiv_\beta t''' \). Hence the Church-Rosser theorem holds for the general \( \beta \)-reduction.

For the proof of Subject Reduction, we need the following “shuffle lemma”.

Lemma 3.11 \( \Gamma \vdash \overline{s_1(t_1 \delta)} \overline{s_2(t_2 \delta)} : \rho \Rightarrow \Gamma \vdash \overline{s_1(s_2(t_1 \delta))} t_2 : \rho \) where \( \overline{s_2} \) is well-balanced and the binding variables in \( \overline{s_2} \) are not free in \( t_1 \).

Proof: By induction on weight(\( \overline{s_2} \)).

- case weight(\( \overline{s_2} \)) = 0 then nothing to prove.

- case weight(\( \overline{s_2} \)) = 2, say \( \overline{s_2} \equiv (t_3 \delta)(\rho_1 \lambda_v) \). We use induction on weight(\( \overline{s_1} \)).

  - Suppose weight(\( \overline{s_1} \)) = 0.

    \( \Rightarrow \) suppose \( \Gamma \vdash (t_3 \delta)(t_3 \delta)(\rho_1 \lambda_v) t_2 : \rho \).

    Using the Generation lemma three times, we obtain:

    \[ \begin{align*}
    \Gamma & \vdash (t_3 \delta)(\rho_1 \lambda_v) t_2 : \rho' \rightarrow \rho \\
    \Gamma & \vdash t_1 : \rho' \\
    \Gamma & \vdash (\rho_1 \lambda_v) t_2 : \rho_1 \rightarrow (\rho' \rightarrow \rho) \\
    \Gamma & \vdash t_2 : \rho_1 \\
    \Gamma & \vdash (\rho_1 \lambda_v) t_2 : \rho' \rightarrow \rho
    \end{align*} \]
Hence
\[ \Gamma(\rho_1 \lambda_\nu) \vdash (t_1 \delta) t_2 : \rho \] (Context, \( \rightarrow \)-elimination, (1), (3)) (4)

Now, \( \Gamma \vdash (p_1 \lambda_\nu)(t_1 \delta) t_2 : p_1 \rightarrow p \) (\( \rightarrow \)-introduction, (4)) (5)

And so, \( \Gamma \vdash (t_3 \delta)(p_1 \lambda_\nu)(t_1 \delta) t_2 : \rho \) (\( \rightarrow \)-elimination, (5), (2))

\( \Rightarrow \) Suppose \( \Gamma \vdash (t_3 \delta)(p_1 \lambda_\nu)(t_1 \delta) t_2 : p \).

Using the Generation lemma three times we obtain:
\[
\begin{align*}
\Gamma & \vdash (p_1 \lambda_\nu)(t_1 \delta) t_2 : p_1 \\
\Gamma & \vdash t_3 : p_1 \\
\Gamma & \vdash (p_1 \lambda_\nu)(t_1 \delta) t_2 : p \\
\Gamma & \vdash t_2 : p' \rightarrow p \\
\Gamma & \vdash (t_1 \delta)(t_1 \delta)(p_1 \lambda_\nu)(t_1 \delta) t_2 : \rho \\
\end{align*}
\]

\( \Rightarrow \) Suppose \( \text{weight(st)} = 1/1 + 1 \).

\( \Rightarrow \) Case \( \overrightarrow{s_1} \equiv (t_1 \delta) \overrightarrow{s_3} \) then
\[
\begin{align*}
\Gamma & \vdash (t_2 \delta) \overrightarrow{s_1}(t_1 \delta)(t_1 \delta)(p_1 \lambda_\nu)(t_1 \delta) t_2 : \rho \Leftrightarrow \text{Generation; \( \rightarrow \)-elimination} \\
\Gamma & \vdash (t_1 \delta)(p_1 \lambda_\nu)(t_1 \delta) t_2 : p' \rightarrow p \land \Gamma \vdash t_4 : p' \Leftrightarrow \text{IH} \\
\Gamma & \vdash (t_1 \delta)(p_1 \lambda_\nu)(t_1 \delta) t_2 : p' \rightarrow p \land \Gamma \vdash t_4 : p' \Leftrightarrow \text{\( \rightarrow \)-elimination; Generation} \\
\Gamma & \vdash (t_3 \delta) (p_1 \lambda_\nu)(t_1 \delta) t_2 : \rho \\
\end{align*}
\]

\( \Rightarrow \) Case \( \overrightarrow{s_1} \equiv (p_2 \lambda_\nu) \overrightarrow{s_3} \) then
\[
\begin{align*}
\Gamma & \vdash (t_2 \delta) \overrightarrow{s_1}(t_1 \delta)(t_1 \delta)(p_1 \lambda_\nu)(t_1 \delta) t_2 : \rho \Leftrightarrow \text{Generation; \( \rightarrow \)-introduction} \\
\Gamma & \vdash (t_1 \delta)(p_2 \lambda_\nu)(t_1 \delta) t_2 : p_3 \land \rho \equiv p_2 \rightarrow p_3 \Leftrightarrow \text{IH} \\
\Gamma & \vdash (p_2 \lambda_\nu)(t_1 \delta)(p_1 \lambda_\nu)(t_1 \delta) t_2 : p_3 \land \rho \equiv p_2 \rightarrow p_3 \Leftrightarrow \text{\( \rightarrow \)-elimination; Generation} \\
\Gamma & \vdash (p_2 \lambda_\nu)(t_1 \delta)(p_1 \lambda_\nu)(t_1 \delta) t_2 : \rho \\
\end{align*}
\]

\( \Rightarrow \) Case \( \text{weight}(\overrightarrow{s_2}) = 2(n + 1), u \geq 1 \). If \( \overrightarrow{s_2} \equiv (t_3 \delta) \overrightarrow{s_3}))(p_1 \lambda_\nu) \overrightarrow{s_4} \) where \( \overrightarrow{s_3}, \overrightarrow{s_4} \) are well-balanced and \( \text{IH} \) holds for them, then:
\[
\begin{align*}
\Gamma & \vdash (t_2 \delta) \overrightarrow{s_1}(t_1 \delta)(t_1 \delta)(p_2 \lambda_\nu)(t_1 \delta) t_2 : \rho \Leftrightarrow \text{IH} \\
\Gamma & \vdash (t_1 \delta)(t_3 \delta)(t_1 \delta)(p_2 \lambda_\nu)(t_1 \delta) t_2 : \rho \Leftrightarrow \text{IH} \\
\Gamma & \vdash (t_1 \delta)(t_3 \delta)(p_1 \lambda_\nu)(t_1 \delta) t_2 : \rho \Leftrightarrow \text{IH} \\
\Gamma & \vdash (t_3 \delta) \overrightarrow{s_1}(t_1 \delta)(t_1 \delta) t_2 : \rho \Leftrightarrow \text{IH} \\
\Gamma & \vdash (t_3 \delta) \overrightarrow{s_3}(p_1 \lambda_\nu)(t_1 \delta) t_2 : \rho. \\
\end{align*}
\]

**Remark 3.12** Note that in Lemma 3.11 above, we insisted on the condition that the binding variables in \( \overrightarrow{s_2} \) are not free in \( t_1 \) in order to avoid cases such as moving \((v \delta)\) in \((t_0 \delta)(p_\lambda_\nu)(v \delta)\) to the left of \((t_0 \delta)(p_\lambda_\nu)\).

Now we can prove Subject Reduction for generalised \( \beta \)-reduction.

**Theorem 3.13** (Generalised Subject Reduction)
If \( \Gamma \vdash t : p \) and \( t \approx_\beta t' \) then \( \Gamma \vdash t' : p \).

**Proof:** By induction on \( \approx_\beta \).

\[ \blacksquare \]

13
• Basic Case: \((t_1 \delta) \beta_2 (\rho_1 \lambda_v) t_2 \rightsquigarrow \beta \beta_2 (t_2[v := t_1])\) and \(\Gamma \vdash (t_1 \delta) \beta_2 (\rho_1 \lambda_v) t_2 : \rho\).

\[
\Gamma \vdash (t_1 \delta) \beta_2 (\rho_1 \lambda_v) t_2 : \rho \Rightarrow \text{Lemma 3.11}
\]

\[
\Gamma \vdash \beta_2 (t_2) (\rho_1 \lambda_v) t_2 : \rho \Rightarrow \text{Lemma 2.17}
\]

\[
\Gamma \vdash \beta_2 (t_2[v := t_1]) : \rho.
\]

• The reflexivity, transitivity and compatibility cases are easy.

For Unicity of Types, we just need the following lemma:

**Lemma 3.14** If \(t \approx_\beta t'\) then \(t =_\beta t'\).

**Proof:** By induction on \(t \approx_\beta t'\) using Corollary 3.9.

**Lemma 3.15** (Generalised Unicity of Types)

1. \(\forall t \forall \rho, \rho'[\Gamma \vdash t : \rho \land \Gamma \vdash t' : \rho' \Rightarrow \rho \equiv \rho']\)

2. \(\forall t, t' \forall \rho, \rho'[\Gamma \vdash t : \rho \land \Gamma \vdash t' : \rho' \land t \approx_\beta t' \Rightarrow \rho \equiv \rho']\)

**Proof:** The proof of 1 is the same for Lemma 2.18. The proof of 2 is also carried from Lemma 2.18 using Lemma 3.14 above.

Now we come to the proof of Strong Normalisation. For this, we need the following definition:

**Definition 3.16**

• We say that \(t \in \Lambda_T\) is strongly normalising with respect to \(\rightsquigarrow_\beta\) iff every reduction path (with respect to \(\rightsquigarrow_\beta\)) starting at \(t\), terminates.

• We define \(\text{SN} = \{t \in \Lambda_T : t\ is \ strongly \ normalising \ with \ respect \ to \ \rightsquigarrow_\beta\}\).

• For \(A, B \subseteq \Lambda_T\) we define \(A \rightarrow B = \{t \in \Lambda_T : \forall t' \in A[(t' \delta) t \in B]\}\).

• We define \([\cdot] : T \rightarrow \text{Power Set of } \Lambda_T\) as follows:

\[
[a] = \text{SN}
\]

\[
[\rho \rightarrow \rho'] = [\rho] \rightarrow [\rho']
\]

• We call \(X \subseteq \text{SN}\) saturated iff:

1. \(\forall n \geq 0, t_1, \ldots, t_n \in \text{SN}, v \in V[(t_1 \delta) \cdots (t_n \delta) v \in X]\).

2. \(\forall n \geq 0, t_1, \ldots, t_n \in \text{SN}, \rho \in \mathcal{T}, \exists \text{ well-balanced}, t' \in \Lambda_T \}

\[
[(t_1 \delta) \cdots (t_n \delta) \beta(t'[v := t]) \in X \Rightarrow (t_1 \delta) \cdots (t_n \delta)(t' \delta \beta(\rho \lambda_v) t' \in X].
\]

• We define \(\text{SAT} = \{X \subseteq \Lambda_T : X \text{ saturated}\}\).

Those familiar with the proof of Strong Normalisation of \(\lambda_\_\_\), will notice that we have accommodated \(\rightsquigarrow_\beta\) in the definition of \(\text{SN}\) and that in the second condition of a saturated set, we have accommodated extended redexes. The accommodation of saturated sets with extended redexes is not necessary, the proof can go without it. Furthermore, the following is the crucial lemma which highlights the difference between \(\rightsquigarrow_\beta\) and \(\rightsquigarrow_\beta\). Once this lemma is established, the proof of Strong Normalisation proceeds similarly to that of ordinary \(\rightarrow_\beta\).
Lemma 3.17

1. $SN \in SAT$.
2. $A, B \in SAT \Rightarrow A \rightarrow B \in SAT$.
3. $\rho \in T \Rightarrow [\rho] \in SAT$.

Proof:

1. $SN \subseteq SN$ and if $t_1, \ldots, t_n \in SN, v \in V$ then similarly $(t_1\delta) \cdots (t_n\delta) v \in SN$.
   Now, if $t, t_1, \ldots, t_n \in SN, \rho \in T, S$ is well-balanced and $t' \in \Lambda_T$ such that
   $(t_1\delta) \cdots (t_n\delta) \overline{s}(t'[v := t]) \in SN$ then also $(t_1\delta) \cdots (t_n\delta)(\rho \lambda_v)t' \in SN$:
   - Reductions inside $t', t, S$ or one of the $t_i$ must terminate since these terms are $SN$ (subterms of $SN$-terms are themselves $SN$, $t'[v := t]$ is $SN \Rightarrow t'$ is $SN$).
   - A reduction path of $(t_1\delta) \cdots (t_n\delta)(\rho \lambda_v)t'$ goes to $(t'_1\delta) \cdots (t'_n\delta)(\rho \lambda_v)t''$ with $t' \rightarrow_{\rho} t''$ etc. and then to $(t'_1\delta) \cdots (t'_n\delta) \overline{s}(t''[v := t'])$; since
     $(t_1\delta) \cdots (t_n\delta) \overline{s}(t'[v := t]) \in SN$ also $(t'_1\delta) \cdots (t'_n\delta) \overline{s}(t''[v := t']) \in SN$.

2. Suppose $A, B \in SAT$.
   - As $v \in A$ for all $v \in V$, we see: $t \in A \rightarrow B \Rightarrow (v\delta)t \in B \Rightarrow (v\delta)t \in SN \Rightarrow t \in SN$. So $A \rightarrow B \subseteq SN$.
   - If $t_1, \ldots, t_n \in SN, v \in V$ then for all $t \in A$, as $t \in SN$ and $B \in SAT$, we get that
     $(\overline{t}\delta)(t_1\delta) \cdots (t_n\delta)v \in B$. Hence $(t_1\delta) \cdots (t_n\delta)v \in A \rightarrow B$ which proves condition 1 of saturation.
   - As to condition 2, suppose $t, t_1, \ldots, t_n \in SN, t' \in \Lambda_T, S$ is well-balanced, $\rho$ a type
     and $(t_1\delta) \cdots (t_n\delta) \overline{s}(t'[v := t]) \in A \rightarrow B$.
     Let $t'' \in A$. Then $(t''\delta)(t_1\delta) \cdots (t_n\delta) \overline{s}(t''[v := t]) \in B$, by definition of $A \rightarrow B$.
     Hence $(t''\delta)(t_1\delta) \cdots (t_n\delta) \overline{s}(\rho \lambda_v)t'' \in B$ since $B \in SAT, t'' \in A \subseteq SN$.
     This means $(t_1\delta) \cdots (t_n\delta)(\rho \lambda_v)t'' \in A \rightarrow B$.

3. Easy induction on the generation of $\rho$ using 1 and 2. \hfill \square

Corollary 3.18 For all $\rho \in T$, we have $[\rho] \neq \emptyset$ and $[\rho] \subseteq SN$.

Proof: Note that no saturated set is empty (use $SN \neq \emptyset$ and condition 1 of saturated sets). \hfill \square

Definition 3.19

- A valuation is a map $g : V \rightarrow \Lambda_T$
- If $g$ is a valuation then $[\cdot]_g$ is defined inductively as follows:
  
  $[v]_g = g(v)$
  $[(t\delta)t']_g = ([t]_g[t']_g)$
  $[(\rho \lambda_v)]_g = ([\rho]_g[\lambda_v]_g(v := v))$

  where $g(v := N)$ is the valuation that assigns $g(v')$ to $v' \neq v$ and $N$ to $v$. Note that $[t]_g$
  substitutes $g(v')$ for $v'$ in $t$ for all free variables $v'$ of $t$. For example, $[(\rho \lambda_x)(y\delta)x]_g =
  (\rho \lambda_x)(g(y\delta)x)$.  

15
• $\models$ is defined as follows:

\[
\begin{align*}
g \models t : \rho & \iff \llbracket t \rrbracket_\rho \in [\rho] \\
g \models \Gamma & \iff \text{for all } (\rho \lambda v) \in \Gamma, \text{ we have } g \models v : \rho \\
\Gamma \models t : \rho & \iff \text{for all valuations } g, \text{ if } g \models \Gamma \text{ then } g \models t : \rho
\end{align*}
\]

**Lemma 3.20** (Soundness)
If $\Gamma \vdash t : \rho$ then $\Gamma \models t : \rho$.

**Proof:** By a straightforward induction on the derivation of $\Gamma \vdash t : \rho$. We only treat the $\rightarrow$-introduction.

Suppose $\Gamma \vdash (\rho \lambda v)t : \rho \rightarrow \rho'$ out of $\Gamma(\rho \lambda v) \vdash t : \rho'$.

Suppose $g \models \Gamma$ in order to show $g \models (\rho \lambda v)t : \rho \rightarrow \rho'$ (i.e. for all $t' \in [\rho] : (t' \delta)(\rho \lambda v)t_\delta \in [\rho']$).

Let $t' \in [\rho]$. Then $g(v := v') \models \Gamma(\rho \lambda v)$, so by the induction hypothesis $\llbracket t' \delta \rrbracket_{g(v := v')} \in [\rho']$.

Since $(t' \delta)(\rho \lambda v)t_\delta \equiv (t' \delta)(\rho \lambda v)t_\delta[\rho(v := v') \rightarrow \beta]$, $g(v := v') \models [\rho] \subseteq SN$ and $[\rho] \subseteq SAT$, also $(t' \delta)(\rho \lambda v)t_\delta \in [\rho']$. \hfill $\Box$

**Theorem 3.21** (Strong Normalisation with respect to $\rightarrow_\beta$)
If $\Gamma \vdash t : \rho$ then $t$ is strongly normalising with respect to $\rightarrow_\beta$.

**Proof:** Suppose $\Gamma \vdash t : \rho$. Define $g(v) = v$. Then $g \models \Gamma$ (because $[\rho] \subseteq SAT$, so $V \subseteq [\rho]$). Hence by soundness $\llbracket t \rrbracket_\rho \subseteq [\rho] \subseteq SN$. But $\llbracket t \rrbracket_\rho \models t$. \hfill $\Box$

4 Term reshuffling

In this section we shall rewrite terms so that all the newly visible redaxes (obtained as a result of our item notation), can be subject to the ordinary classical $\beta$-reduction $\rightarrow_\beta$. We shall show that this term rewriting is correct and preserves both reduction (be it only in a certain sense) and typing.

Let us go back to the definition of $\delta \lambda$-couples. Recall that if $\overline{s} = s_1 \cdots s_m$ for $m > 1$ where $s_1s_m$ is a $\delta \lambda$-couple then $s_2 \cdots s_{m-1}$ is a well-balanced segment, $s_1 \equiv (t_1 \delta)$ is the $\delta$-item of the $\delta \lambda$-couple and $s_m \equiv (\rho \lambda v)$ is its $\lambda$-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 $s_2 \cdots s_{m-1}s_1s_m$.

**Example 4.1** The term $(x_1 \delta)(x_2 \delta)(x_4 \lambda x_7)(x_3 \delta)(x_3 \lambda x_6)(x_1 \delta)(x_1 \rightarrow X_2 X_3 \lambda x_5)(x_4 \delta)x_5$ can be easily rewritten as $(x_2 \delta)(X_4 \lambda x_7)(x_3 \delta)(X_3 \lambda x_6)(x_1 \delta)(X_1 \rightarrow X_2 X_3 \lambda x_5)(x_4 \delta)x_5$ by moving the item $(x_1 \delta)$ 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 redaxes 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 $((\lambda_{x_7} : X_4)(\lambda_{x_6} : x_3) \lambda_{x_5} : X_1 \rightarrow X_2) x_3 x_4) x_2)$ is written as $(\lambda_{x_7} : X_4)(\lambda_{x_6} : x_3)(\lambda_{x_5} : X_1 \rightarrow X_2) x_3 x_4) x_2)$. This is another advantage of our item notation.

Note furthermore that the shuffling is not problematic because we use 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 4.2** If $v^e$ is a free occurrence of $v$ in $s \overline{t}$, then $v^o$ is free in $s \overline{t}$.

**Proof:** By BC as $\lambda_v$ does not occur in $s \overline{t}$. \hfill $\Box$
Example 4.3 Note that in Example 4.1, reshuffling does not affect the “meaning” of the term. In fact, in $t \equiv (x_1 \delta)(x_2 \delta)(x_3 \delta)(x_4 \lambda x_7)(x_5 \lambda x_8)((X_1 \rightarrow X_2)\lambda x_7)(x_4 \delta)x_5$, the free variable $x_7$ cannot be captured by $\lambda x_7$ or $\lambda x_8$. Moreover, $t$ is equivalent, semantically and procedurally, to $(x_2 \delta)(x_4 \lambda x_7)(x_3 \delta)(x_5 \lambda x_8)(x_1 \delta)((X_1 \rightarrow X_2)\lambda x_7)(x_4 \delta)x_5$.

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. To do this however, we must study the classes of partnered and bachelor items in a term.

4.1 Partitioning the term into bachelor and well-balanced segments

With Definition 3.2, 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 4.4 Let $\overline{s}$ be the body of a term $t$. Then the following holds:

1. Each bachelor main $\lambda$-item in $\overline{s}$ precedes each bachelor main $\delta$-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 $\delta \lambda$-couples, leaves behind a $\lambda \ldots \lambda \delta \ldots \delta$-segment, consisting of all bachelor main $\lambda$- and $\delta$-items.

4. If $\overline{s} \equiv \overline{s_1}(t\delta)\overline{s_2}((\rho \lambda)\overline{s_3})$ where $(\rho \lambda)$ and $(t\delta)$ match, then $\overline{s_2}$ is well-balanced.

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

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

Corollary 4.5 For each non-empty segment $\overline{s}$, there is a unique partitioning in segments $\overline{s_1}, \overline{s_2}, \ldots, \overline{s_n}$, such that

1. $\overline{s} \equiv \overline{s_0} \overline{s_1} \ldots \overline{s_n}$,

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. Each bachelor $\lambda$-segment $\overline{s_i}$ precedes each bachelor $\delta$-segment $\overline{s_k}$ in $\overline{s}$.

4. $\overline{s_2n} \neq \emptyset$ for $n > 0$. \[\square\]

Example 4.6 $\overline{s} \equiv (p_1 \lambda v_1)(p_2 \lambda v_2)(t_1 \delta)(p_3 \lambda v_3)(p_4 \lambda v_4)(t_2 \delta)(t_3 \delta)(t_4 \delta)(p_5 \lambda v_5)(p_6 \lambda v_6)(t_5 \delta)$ has the following partitioning:

- well-balanced segment $\overline{s_0} \equiv \emptyset$,

- bachelor segment $\overline{s_1} \equiv (p_1 \lambda v_1)(p_2 \lambda v_2)$. 
• well-balanced segment $\overline{xy} \equiv (t_1 \delta)(\rho_3 \lambda_v)$,
• bachelor segment $\overline{xy} \equiv (t_2 \delta)\lambda_v$,
• well-balanced segment $\overline{xyz} \equiv (t_3 \delta)(t_4 \delta)(\rho_5 \lambda_v)(\rho_6 \lambda_v)$,
• bachelor segment $\overline{xyz} \equiv (t_5 \delta)$.

4.2 The reshuffling procedure and its properties

In what follows, we use $\omega_1, \omega_2, \ldots$ to range over $\{\delta\} \cup \{\lambda_v; v \in V\}$, and we shall use $A_1, A_2, \ldots$ to range over both terms and types (i.e. over $\Lambda_T \cup T$).

**Definition 4.7** $TS$ and $T$ are defined mutually recursively such that:

- $TS(\phi) = \phi$
- $TS(\phi \overline{xy}) = TS(\phi) \overline{xy}$
- $TS((A_1 \omega_1) \cdots (A_n \omega_n)) = TS(A_1 \omega_1) \cdots TS(A_n \omega_n)$ if $(A_1 \omega_1) \cdots (A_n \omega_n)$ is bachelor
- $TS(\overline{xy}) = T(\emptyset, \overline{xy})$
- $TS(\overline{x_1} \cdots \overline{x_n}) = TS(\overline{x_1}) \cdots TS(\overline{x_n})$

If $\overline{x_1} \cdots \overline{x_n}$, is the unique partitioning of Corollary 4.5

$T(\overline{s}(t\delta), (\rho \lambda_v)\overline{s}) = TS(\overline{s}(t\delta), (\rho \lambda_v)\overline{s})$

$T(\overline{s}, (t\delta)\overline{s}) = T(\overline{s}(TS(t\delta)), \overline{s})$

$T(\emptyset, \emptyset) = \emptyset$

**Lemma 4.8**

1. If $\overline{s}$ is well-balanced, then $T(\overline{s}_1, \overline{s}_2) \equiv T S(\overline{s} ) T(\overline{s}_1, \overline{s}_2)$.
2. If $(t\delta)$ matches $(\rho \lambda_v)$ in $\overline{s} \equiv (t\delta)\overline{s}(\rho \lambda_v)\overline{s'}$ then $TS(\overline{s'}) \equiv TS(\overline{s}(t\delta),(\rho \lambda_v)\overline{s'})$.
3. If $\overline{s}$ contains no items which are partnered in $t$ then $TS(\overline{s}) \equiv TS(\overline{s})TS(t)$.
4. If $\overline{s}$ is bachelor in $\overline{st}$ or is well-balanced, then $TS(\overline{s}t) \equiv TS(\overline{s})TS(t)$.

**Proof:** 1: by induction on weight($\overline{s}$). Case weight($\overline{s}$) = 0 then obvious.

Case $\overline{s} \equiv (t\delta)\overline{s}(\rho \lambda_v)\overline{s'}$ then

- $T(\overline{s}_1, (t\delta)\overline{s}(\rho \lambda_v)\overline{s'}) \equiv T(\overline{s}_1(\overline{s}(t\delta),(\rho \lambda_v)\overline{s'})) \equiv IH$
- $T S(\overline{s})T(\overline{s}_1(\overline{s}(t\delta),(\rho \lambda_v)\overline{s'})) \equiv TS(\overline{s})(\overline{s}_1(\overline{s}(t\delta),(\rho \lambda_v)\overline{s'})) \equiv IH$

2: using 1. 3: let $t \equiv \overline{s}_0 \cdots \overline{s}_n \lambda_v$ and $\overline{s} \equiv \overline{s}_0 \cdots \overline{s'}$ be partitionings. Use cases on $\overline{s}_0$ being empty or not and on $\overline{s'}$ being bachelor or well-balanced. 4: This is a corollary of 3 above. □

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

---

1 Note here that, from BC, no binding variables of $\overline{s}$ are free in $t$. 

18
Lemma 4.9 For every subterm t' of a term t, the following holds:

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

2. If \( \overline{s}' \equiv (t''x_\delta)(\rho\lambda_x) \) is a subsegment of t' where \( \overline{s} \) is well-balanced, then \( TS(\overline{s}) \equiv TS(\overline{s}'')(TS(t''x_\delta)(\rho\lambda_x)) \).

3. If \( \delta \equiv (A_1\omega_1) \cdots (A_n\omega_n) \) is bachelor in t', then \( TS(\delta) \equiv (TS(A_1)\omega_1) \cdots (TS(A_n)\omega_n) \) is bachelor in TS(t').

4. If s is a subsegment of t' which is well-balanced, then TS(s) is well-balanced.

Proof: By induction on t.

- Case t \( \equiv v \) then t is the unique subterm of t and all 1 \cdots 4 hold.
- Assume t \( \equiv (A\omega)t_2 \) where IH holds for A if A \( \equiv t_1 \) and for t_2. Let t' be a subterm of t. If t' is a subterm of t_1 (for A \( \equiv t_1 \)) or t_2 then use IH. If t' \( \equiv t \) then:
  - Case (A\omega) is bachelor then \( TS(t) \equiv TS(t) \equiv (TS(A)\omega)TS(t_2) \). Here all 1 \cdots 4 hold by IH on A and t_2.
  - Case A \( \equiv t_1 \land (t_1x_\delta) \) matches (\( \rho\lambda_x \)) in t. I.e. t \( \equiv (t_1x_\delta)(\rho\lambda_x)t_3 \) then \( TS(t) \equiv TS(t)(t_1x_\delta)(\rho\lambda_x)TS(t_3) \). Now use IH to show 1 \cdots 4. \( \square \)

Lemma 4.10 For all variables v and terms t, t' we have:

\[ TS(t) \equiv TS(TS(t)) \] and \( TS(t[v := t']) \equiv TS(TS(t)[v := TS(t')]) \).

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 \( \neq TS(t_2) \). For example, \((x_1x_\delta)(x_2x_\delta)(\rho\lambda_x_2)(\rho\lambda_x_2)x_3 \rightarrow_\beta (x_1x_\delta)(x_2x_\delta)(\rho\lambda_x_3)(\rho\lambda_x_4)x_4 \).

Following this remark, we show that in a sense, term reshuffling preserves \( \beta \)-reduction.

Lemma 4.11 If t, t' \( \in \lambda \) and t \( \sim_\beta t' \) then \( (TS(t) \rightarrow_\beta t'') \land TS(t'') \equiv TS(t') \). In other words, the following diagram commutes:

\[
\begin{array}{c}
\text{t} \\
TS \\
TST(t) \rightarrow_\beta t'' \\
TS(t'') \equiv TS(t')
\end{array}
\]

Proof: By induction on the general \( \sim_\beta \).

- Case t \( \equiv \overline{s}(t_1x_\delta)(\rho\lambda_x)t_3 \sim_\beta t' \equiv \overline{s}(t_3[v := t_1]) \), we use induction on the number n of bachelor \( \delta \)-items of \( \overline{s} \) that are partnered in t_3. Recall that \( \delta \) is well-balanced.
Case $n = 0$ then

$$\begin{align*}
TS(\overline{s}_1(t)\overline{s}(\rho\lambda_\nu)_{t_3}) & \equiv \text{Lemma 4.8 (3,14)} \\
TS(\overline{s}_1)TS((t_1)\overline{s}(\rho\lambda_\nu))TS(t_3) & \equiv \text{Lemma 4.8 (2,4)} \\
TS(\overline{s}_1)TS(\overline{s})(TS(1)\delta(\rho\lambda_\nu)TS(t_3)) & \to_\beta \\
TS(\overline{s}_1)TS(\overline{s})(TS(t_3)[v := TS(t_1)]) & \equiv t". \\
TS(t") & \equiv \text{Lemmas 4.0, 4.10, 4.8 (3,4)} \\
TS(\overline{s}_1)TS(\overline{s})TS(TS(t_2)[v := :TS(t_1)]) & \equiv \text{Lemma 4.10} \\
TS(\overline{s}_1)TS(\overline{s})TS(t_3[v := :t_1]) & \equiv \\
TS(\overline{s}_1)TS(t_3[v := :t_1]) & \equiv \\

\end{align*}$$

Assume the property holds for $n$ and let us show it for the case where $\overline{s}_1$ contains $n + 1$ $\delta$-items which match $\lambda$-items of $t_3$. Let $(t")$ be the leftmost such $\delta$-item of $\overline{s}_1$. Take $\overline{s} \equiv \overline{s}_1(t")s_1^{(t")}$ and $t_3 \equiv \overline{s}_2(\rho\lambda\nu)t_2$ where $(t")$ matches $((\rho\lambda\nu))$. By Lemma 4.4, $(t")s_1^{(t")}(t_1)\overline{s}(\rho\lambda\nu)s_2^{(t")}(\rho\lambda\nu)$ is well-balanced. Moreover, no item of $\overline{s}_1$ has a partner in $(t")s_1^{(t")}(t_1)\overline{s}(\rho\lambda\nu)t_3$.

As $\overline{s}_1(t_1)\overline{s}(\rho\lambda\nu)s_2^{(t")}(t_1)\overline{s}(\rho\lambda\nu)t_2 \to_\beta \overline{s}_1(t_2)\overline{s}(\rho\lambda\nu)t_2[v := t_1])$, we find by III, $t"$ such that

$$\begin{align*}
TS(\overline{s}_1(t_2)\overline{s}(\rho\lambda\nu)s_2^{(t")}(\rho\lambda\nu)t_3) & \to_\beta tmm \\
TS(tmm) & \equiv TS(\overline{s}_1(t_2)\overline{s}(\rho\lambda\nu)t_2[v := t_1]) \\

\end{align*}$$

Now, $TS(\overline{s}_1)tt"$ is the wanted term because:

$$\begin{align*}
TS(t) & \equiv \text{Lemma 4.8 (4)} \cdot \text{TS}(\overline{s}_1)TS(((t")s_1^{(t")}(t_1)\overline{s}(\rho\lambda\nu)s_2^{(t")}(\rho\lambda\nu)t_2) \equiv \text{Lemma 4.8 (2)} \\
TS(\overline{s}_1)TS(\overline{s}(t_1)\overline{s}(\rho\lambda\nu)s_2^{(t")}(\rho\lambda\nu)t_2) & \to_\beta TS(\overline{s}_1)tmm \\
and TS(\overline{s}_1)T(\overline{s}_2) & \equiv \text{Lemma 4.10} \\
TS(\overline{s}_1)TS(\overline{s}_2) & \equiv \text{Lemma 4.8 (2), BC} \\
TS(\overline{s}_1)TS(((t")s_1^{(t")}(s_2[v := t_1])((\rho\lambda\nu)(t_2[v := t_1])) & \equiv \text{Lemma 4.8 (4), \overline{s}_1^{(t")}s_2 well-balanced} \\
TS(\overline{s}_1(t")s_1^{(t")}(s_2[v := t_1]) & \equiv TS(t") \\

\end{align*}$$

The proof of compatibility is technical. The difficult case is: $t \equiv (t_1)\overline{s}t_2$ and $t \to_\beta t_2'$. Distinguish the cases: $t_1$ is bachelor or non-bachelor in $t$.

\begin{corollary}
If $t \to_\beta t'$ then there exist $t_0, t_1, \cdots, t_n$ such that

$$[(t_0) \to_\beta t_1) \land (TS(t_0) \to_\beta t_2) \land \cdots \land (TS(t_{n-1}) \to_\beta t_n) \land (TS(t_n) \equiv TS(t'))]$$

Proof: By induction on $\to_\beta$.

- Case $t \not\to_\beta t'$ use Lemma 4.11.
- Case $t \not\to_\beta t$ then obvious ($n = 1 \land t_0 \equiv t \equiv TS(t)$).
- Case $t \not\to_\beta t'' \land t'' \not\to_\beta t'''$, then by III, there exist $t_0, t_1, \cdots, t_n, t_0', t_1', \cdots, t'_n$ such that

$$[(t' \equiv t_1) \land (TS(t_0) \to_\beta t_1) \land (TS(t_1) \to_\beta t_2) \land \cdots \land (TS(t_{n-1}) \to_\beta t_n) \land (TS(t_n) \equiv TS(t')) \land (t'' \equiv t_0') \land (TS(t_0') \to_\beta t_1') \land (TS(t_1') \to_\beta t_2') \land \cdots \land (TS(t_{n-1}') \to_\beta t_n') \land (TS(t_n') \equiv TS(t'''))]$$

Hence, $(t' \equiv t_0) \land (TS(t_0) \to_\beta t_1) \land (TS(t_1) \to_\beta t_2) \land \cdots \land (TS(t_{n-1}) \to_\beta t_n) \land (TS(t_n) \to_\beta t_1) \land \cdots \land (TS(t_{n-1}) \to_\beta t_n') \land (TS(t_n') \equiv TS(t'''))$.

Note that for the basic and reflexive cases, $n = 1$ for sure. For the transitive case, this may not be so. For example, $t \equiv (\rho\lambda x_1)((\rho\lambda x_2)(x_1\delta)(x_2\delta)(x_1\delta)x_1 \equiv \overline{s_2})$ $t' \equiv (\rho\lambda x_2)(x_1\delta)(\rho\lambda x_2)x_1$ yet $t \not\to_\beta t'$ does not imply there exists $t''$ such that $TS(t) \to_\beta t'' \land TS(t'') \equiv TS(t')$. There is however $t_1 \equiv (\rho\lambda x_2)(x_1\delta)(\rho\lambda x_2)(\rho\lambda x_3)x_1$ and $t_2 \equiv t'$ such that $TS(t) \to_\beta t_1 \land TS(t_1) \to_\beta t_2$ and $TS(t_2) \equiv TS(t')$.

\end{corollary}
Finally, we show that term reshuffling preserves typing:

**Lemma 4.13** If $\Gamma \vdash t : \rho$ then $\Gamma \vdash TS(t) : \rho$.

*Proof:* By induction on $t$.

- **Case** $t \equiv v$, then nothing to prove.

- **Case** $t \equiv (\rho^I \lambda_\nu)t'$ then

  \[
  \begin{align*}
  \Gamma \vdash (\rho^I \lambda_\nu)t' : \rho & \quad \Rightarrow \text{Generation} \\
  \Gamma(\rho^I \lambda_\nu) \vdash t : \rho'' \land \rho \equiv \rho' \rightarrow \rho'' & \quad \Rightarrow \text{IH} \\
  \Gamma(\rho^I \lambda_\nu) \vdash TS(t') : \rho'' \land \rho \equiv \rho' \rightarrow \rho'' & \quad \Rightarrow \rightarrow \text{-elimination, Lemma 4.8(3)} \\
  \Gamma \vdash TS((\rho^I \lambda_\nu)t') : \rho & 
  \end{align*}
  \]

- **Case** $t \equiv (t' \delta)t''$ then

  \(-\text{Case} (t' \delta) \text{ is bachelor in } t \text{ then}\)

  \[
  \begin{align*}
  \Gamma \vdash (t' \delta)t'' : \rho & \quad \Rightarrow \text{Generation} \\
  \Gamma \vdash t' : \rho' \land \Gamma \vdash t'' : \rho' \rightarrow \rho & \quad \Rightarrow \text{IH} \\
  \Gamma \vdash TS(t') : \rho' \land \Gamma \vdash TS(t'') : \rho' \rightarrow \rho & \quad \Rightarrow \rightarrow \text{-elimination, Lemma 4.8(3)} \\
  \Gamma \vdash (TS(t') \delta)TS(t'') : \rho & 
  \end{align*}
  \]

  \(-\text{Case} (t' \delta) \text{ is partnered in } t, \text{ then } t \equiv (t' \delta)(\rho^I \lambda_\nu)t_1 \text{ where } \vec{s} \text{ is well-balanced, and no binding variables of } \vec{s} \text{ are free in } t' \).

  \[
  \begin{align*}
  \Gamma \vdash (t' \delta)\vec{s}(\rho^I \lambda_\nu)t_1 : \rho & \quad \Rightarrow \text{Generation} \\
  \Gamma \vdash t' : \rho' \land \Gamma \vdash \vec{s}(\rho^I \lambda_\nu)t_1 : \rho' \rightarrow \rho & \quad \Rightarrow \text{IH} \\
  \Gamma \vdash TS(t') : \rho' \land \Gamma \vdash TS(\vec{s}(\rho^I \lambda_\nu)t_1) : \rho' \rightarrow \rho & \quad \Rightarrow \rightarrow \text{-elimination, Lemma 4.8(4)} \\
  \Gamma \vdash (TS(t') \delta)TS(\vec{s})(\rho^I \lambda_\nu)TS(t_1) : \rho & \quad \Rightarrow \text{Lemma 4.8(4)} \\
  \Gamma \vdash TS(\vec{s})(TS(t') \delta)(\rho^I \lambda_\nu)TS(t_1) : \rho & \quad \Rightarrow \text{Lemma 4.8(4)} \\
  \Gamma \vdash TS(\vec{s}(t' \delta)(\rho^I \lambda_\nu)t_1) : \rho & \quad \Rightarrow \text{Lemma 4.8(2)} \\
  \Gamma \vdash TS((t' \delta)\vec{s}(\rho^I \lambda_\nu)t_1) : \rho & \quad \Rightarrow \Gamma \vdash TS(t) : \rho 
  \end{align*}
  \]

\section{Conclusion}

In this paper, we observed that if we change slightly the classical $\lambda$-notation, then we can make more redexes visible. This is useful and is in line with current research on the needed redexes (for normal forms) as in [BKKS 87]. Making more redexes visible will work to our advantage if we could also contract these redexes before other ones. For example, in lazy evaluation ([Launchbury 93]), some redexes get frozen while other ones are being contracted. Now, if we had the ability of choosing which redex to contract out of all visible redexes, rather than waiting for some redex to be evaluated before we can proceed with the rest, then we can say that we have achieved a flexible system where we have control over what to contract rather than letting reductions force themselves in some order. This may lead to some advantages concerning optimal reductions as in [Lévy 80].
With our notation, and our new \( \beta \)-reduction, we achieve this flexibility and freedom of choice. Moreover, we do not lose any of the original properties. We have shown in fact that what we provide is a more general \( \beta \)-reduction where more redexes are visible and where all the original properties (using ordinary classical reduction) still hold for our general reduction. We believe this to be an important breakthrough which may lead to new reduction strategies that may explain various programming principles (such as lazy evaluation) in an elegant way.

We have shown further that, using item notation (which makes more redexes visible), one is able to stick to the old \( \beta \)-reduction and just do a simple reshuffling so that these newly visible redexes can be contracted before other redexes. We have shown that this reshuffling (which is very simple and can only be enabled in our notation), is correct. In fact, reshuffling does really make all redexes subject to immediate contraction and preserves typing. So, if \( t \) has type \( \rho \) then the reshuffled version of \( t \) also has type \( \rho \). It is moreover the case that if \( t \sim_{\beta} t' \) using our extended reduction, then \( TS(t) \) can be transformed into \( TS(t') \) using classical reduction and intermediate term reshuffling.

The work carried out in this paper will have many applications. We mentioned the semantics of lazy evaluation and the new reduction strategies which may lead to further optimal results. These points are under investigation. The new notation moreover deserves attention. [KN 93] and [NK 94] have shown many of its advantages for formulating and generalising type theory and for rendering substitution explicit in the \( \lambda \)-calculus. Further advantages are also studied in [KN 92].

References


<table>
<thead>
<tr>
<th>Issue</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>91/02</td>
<td>R.P. Nederpel, 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</td>
<td>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</td>
<td>D. de Reus</td>
<td>An Implementation Model for GOOD, p. 18.</td>
</tr>
<tr>
<td>91/06</td>
<td>K.M. van Hee</td>
<td>SPECIFICATIEMETHODEN, een overzicht, p. 20.</td>
</tr>
<tr>
<td>91/07</td>
<td>E. Poll</td>
<td>CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.</td>
</tr>
<tr>
<td>91/12</td>
<td>E. van der Sluis</td>
<td>A parallel local search algorithm for the travelling salesman problem, p. 12.</td>
</tr>
<tr>
<td>91/14</td>
<td>P. Lemmens</td>
<td>The PDB Hypermedia Package. Why and how it was built, p. 63.</td>
</tr>
<tr>
<td>91/16</td>
<td>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>
</tbody>
</table>
91/17 A.T.M. Aerts  
P.M.E. de Bra  
K.M. van Hee  
Transforming Functional Database Schemes to Relational Representations, p. 21.

91/18 Rik van Geldrop  
Transformational Query Solving, p. 35.

91/19 Erik Poll  
Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.

91/20 A.E. Eiben  
R.V. Schuwer  

91/21 J. Coenen  
W.-P. de Roeveer  
J. Zwiers  
Assertional Data Reification Proofs: Survey and Perspective, p. 18.

91/22 G. Wolf  

91/23 K.M. van Hee  
L.J. Somers  
M. Voorhoeve  
Z and high level Petri nets, p. 16.

91/24 A.T.M. Aerts  
D. de Reus  
Formal semantics for BRM with examples, p. 25.

91/25 P. Zhou  
J. Hooman  
R. Kuiper  
A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and completeness, p. 52.

91/26 P. de Bra  
G.J. Houben  
J. Paredaens  
The GOOD based hypertext reference model, p. 12.

91/27 F. de Boer  
C. Palamidessi  
Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.

91/28 F. de Boer  
A compositional proof system for dynamic process creation, p. 24.

91/29 H. Ten Eikelder  
R. van Geldrop  
Correctness of Acceptor Schemes for Regular Languages, p. 31.

91/30 J.C.M. Baeten  
F.W. Vaandrager  
An Algebra for Process Creation, p. 29.

91/31 H. ten Eikelder  
Some algorithms to decide the equivalence of recursive types, p. 26.

91/32 P. Struik  

91/33 W. v.d. Aalst  
The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.

91/34 J. Coenen  
Specifying fault tolerant programs in deontic logic, p. 15.
<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>91/35</td>
<td>F.S. de Boer</td>
<td>Asynchronous communication in process algebra, p. 20.</td>
</tr>
<tr>
<td></td>
<td>J.W. Klop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C. Palamidessi</td>
<td></td>
</tr>
<tr>
<td>92/01</td>
<td>J. Coenen</td>
<td>A note on compositional refinement, p. 27.</td>
</tr>
<tr>
<td></td>
<td>J. Zwiers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W.-P. de Roever</td>
<td></td>
</tr>
<tr>
<td>92/02</td>
<td>J. Coenen</td>
<td>A compositional semantics for fault tolerant real-time systems, p. 18.</td>
</tr>
<tr>
<td></td>
<td>J. Hooman</td>
<td></td>
</tr>
<tr>
<td>92/03</td>
<td>J.C.M. Baeten</td>
<td>Real space process algebra, p. 42.</td>
</tr>
<tr>
<td></td>
<td>J.A. Bergstra</td>
<td></td>
</tr>
<tr>
<td>92/05</td>
<td>J.P.H.W.v.d.Eijnde</td>
<td>Conservative fixpoint functions on a graph, p. 25.</td>
</tr>
<tr>
<td>92/06</td>
<td>J.C.M. Baeten</td>
<td>Discrete time process algebra, p. 45.</td>
</tr>
<tr>
<td></td>
<td>J.A. Bergstra</td>
<td></td>
</tr>
<tr>
<td>92/07</td>
<td>R.P. Nederpelt</td>
<td>The fine-structure of lambda calculus, p. 110.</td>
</tr>
<tr>
<td>92/08</td>
<td>R.P. Nederpelt</td>
<td>On stepwise explicit substitution, p. 30.</td>
</tr>
<tr>
<td></td>
<td>F. Kamareddine</td>
<td></td>
</tr>
<tr>
<td>92/10</td>
<td>P.M.P. Rambags</td>
<td>Composition and decomposition in a CPN model, p. 55.</td>
</tr>
<tr>
<td>92/11</td>
<td>R.C. Backhouse</td>
<td>Demonic operators and monotype factors, p. 29.</td>
</tr>
<tr>
<td></td>
<td>J.S.C.P.v.d.Woude</td>
<td></td>
</tr>
<tr>
<td>92/13</td>
<td>F. Kamareddine</td>
<td>Set theory and nominalisation, Part II, p. 22.</td>
</tr>
<tr>
<td>92/14</td>
<td>J.C.M. Baeten</td>
<td>The total order assumption, p. 10.</td>
</tr>
<tr>
<td>92/15</td>
<td>F. Kamareddine</td>
<td>A system at the cross-roads of functional and logic programming, p. 36</td>
</tr>
<tr>
<td>92/16</td>
<td>R.R. Seljée</td>
<td>Integrity checking in deductive databases; an exposition, p. 32.</td>
</tr>
<tr>
<td>92/17</td>
<td>W.M.P. van der Aalst</td>
<td>Interval timed coloured Petri nets and their analysis, p. 20.</td>
</tr>
<tr>
<td>92/18</td>
<td>R.Nederpelt</td>
<td>A unified approach to Type Theory through a refined lambda-calculus, p. 30</td>
</tr>
<tr>
<td></td>
<td>F. Kamareddine</td>
<td></td>
</tr>
<tr>
<td>92/19</td>
<td>J.C.M.Baeten</td>
<td>Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36</td>
</tr>
<tr>
<td></td>
<td>J.A.Bergstra</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S.A.Smolkka</td>
<td></td>
</tr>
<tr>
<td>92/20</td>
<td>F.Kamareddine</td>
<td>Are Types for Natural Language? P. 32.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>92/21</td>
<td>F. Kamareddine</td>
<td>Non well-foundedness and type freeness can unify the interpretation of functional application.</td>
</tr>
<tr>
<td>92/22</td>
<td>R. Nederpelt</td>
<td>A useful lambda notation</td>
</tr>
<tr>
<td></td>
<td>F. Kamareddine</td>
<td></td>
</tr>
<tr>
<td>92/23</td>
<td>F. Kamareddine</td>
<td>Nominalization, Predication and Type Containment</td>
</tr>
<tr>
<td></td>
<td>E. Klein</td>
<td></td>
</tr>
<tr>
<td>92/24</td>
<td>M. Codish</td>
<td>Botum-up Abstract Interpretation of Logic Programs</td>
</tr>
<tr>
<td></td>
<td>D. Dams</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eyal Yardeni</td>
<td></td>
</tr>
<tr>
<td>92/25</td>
<td>E. Poll</td>
<td>A Programming Logic for Fo</td>
</tr>
<tr>
<td>92/26</td>
<td>T. H. W. Beelen</td>
<td>A modelling method using MOVIE and SimCon/ExSpect</td>
</tr>
<tr>
<td></td>
<td>W. J. J. Stut</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P. A. C. Verkoulen</td>
<td></td>
</tr>
<tr>
<td>92/27</td>
<td>B. Watson</td>
<td>A taxonomy of keyword pattern matching algorithms</td>
</tr>
<tr>
<td></td>
<td>G. Zwaan</td>
<td></td>
</tr>
<tr>
<td>93/01</td>
<td>R. van Geldrop</td>
<td>Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods</td>
</tr>
<tr>
<td>93/02</td>
<td>T. Verhoef</td>
<td>A continuous version of the Prisoner's Dilemma</td>
</tr>
<tr>
<td>93/03</td>
<td>T. Verhoef</td>
<td>Quicksort for linked lists</td>
</tr>
<tr>
<td>93/04</td>
<td>E. H. L. Aants</td>
<td>Deterministic and randomized local search</td>
</tr>
<tr>
<td></td>
<td>J. H. M. Korst</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P. J. Zwietcring</td>
<td></td>
</tr>
<tr>
<td>93/05</td>
<td>J. C. M. Baeten</td>
<td>A congruence theorem for structured operational semantics with predicates</td>
</tr>
<tr>
<td></td>
<td>C. Verhoef</td>
<td></td>
</tr>
<tr>
<td>93/06</td>
<td>J. P. Veltkamp</td>
<td>On the unavoidability of metastable behaviour</td>
</tr>
<tr>
<td>93/07</td>
<td>P. D. Moerland</td>
<td>Exercises in Multiprogramming</td>
</tr>
<tr>
<td>93/08</td>
<td>J. Verhoosel</td>
<td>A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS</td>
</tr>
<tr>
<td>93/09</td>
<td>K. M. van Hee</td>
<td>Systems Engineering: a Formal Approach</td>
</tr>
<tr>
<td>93/10</td>
<td>K. M. van Hee</td>
<td>Systems Engineering: a Formal Approach</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Part II: Frameworks</td>
</tr>
<tr>
<td>93/11</td>
<td>K. M. van Hee</td>
<td>Systems Engineering: a Formal Approach</td>
</tr>
<tr>
<td>93/12</td>
<td>K. M. van Hee</td>
<td>Systems Engineering: a Formal Approach</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Part IV: Analysis Methods</td>
</tr>
<tr>
<td>93/13</td>
<td>K. M. van Hee</td>
<td>Systems Engineering: a Formal Approach</td>
</tr>
<tr>
<td>Paper ID</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>93/16</td>
<td>H. Schepers, J. Hooman</td>
<td>A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27</td>
</tr>
<tr>
<td>93/17</td>
<td>D. Alstein, P. van der Stok</td>
<td>Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.</td>
</tr>
<tr>
<td>93/18</td>
<td>C. Verhoef</td>
<td>A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.</td>
</tr>
<tr>
<td>93/19</td>
<td>G-J. Houben</td>
<td>The Design of an Online Help Facility for ExSpect, p.21.</td>
</tr>
<tr>
<td>93/22</td>
<td>E. Poll</td>
<td>A Typechecker for Bijective Pure Type Systems, p. 28.</td>
</tr>
<tr>
<td>93/23</td>
<td>E. de Kogel</td>
<td>Relational Algebra and Equational Proofs, p. 23.</td>
</tr>
<tr>
<td>93/24</td>
<td>E. Poll and Paula Severi</td>
<td>Pure Type Systems with Definitions, p. 38.</td>
</tr>
<tr>
<td>93/26</td>
<td>W.M.P. van der Aalst</td>
<td>Multi-dimensional Petri nets, p. 25.</td>
</tr>
<tr>
<td>93/27</td>
<td>T. Kloks and D. Kratsch</td>
<td>Finding all minimal separators of a graph, p. 11.</td>
</tr>
<tr>
<td>93/28</td>
<td>F. Kamareddine and R. Nederpelt</td>
<td>A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.</td>
</tr>
<tr>
<td>93/29</td>
<td>R. Post and P. De Bra</td>
<td>GOLD, a Graph Oriented Language for Databases, p. 42.</td>
</tr>
<tr>
<td>93/30</td>
<td>J. Deogun, T. Kloks, D. Kratsch, H. Müller</td>
<td>On Vertex Ranking for Permutation and Other Graphs, p. 11.</td>
</tr>
<tr>
<td>93/31</td>
<td>W. Körver</td>
<td>Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.</td>
</tr>
</tbody>
</table>
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. Brunekevef  
Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.

93/38 C. Verhoef  
A general conservative extension theorem in process algebra, p. 17.

93/39 W.P.M. Nuijten  
Job Shop Scheduling by Constraint Satisfaction, p. 22.

93/40 P.D.V. van der Stok  

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  
A precise clock synchronization protocol, p.

93/46 T. Kloks  

93/47 W. v.d. Aalst  

93/48 R. Gerth  
Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.
<table>
<thead>
<tr>
<th>Paper Number</th>
<th>Authors</th>
<th>Title and Pages</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. Kamareddine, 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>
| 94/18 | F. Kamareddine  
      | R. Nederpelt  |
|-------|-------------|
| Refining Reduction in the Lambda Calculus, p. 15. |

<table>
<thead>
<tr>
<th>94/19</th>
<th>B.W. Watson</th>
</tr>
</thead>
<tbody>
<tr>
<td>The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.</td>
<td></td>
</tr>
</tbody>
</table>