On Pi-conversion in type theory

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.
Eindhoven University of Technology
Department of Mathematics and Computing Science

On Π-conversion in Type Theory

by

R. Bloo, F. Kamareddine and R. Nederpelt

94/47

ISSN 0926-4515
All rights reserved
editors: prof.dr. J.C.M. Baeten
prof.dr. M. Rem

Computing Science Report 94/47
Eindhoven, October 1994
The \(\lambda\)-cube with classes of terms modulo conversion*

Roel Bloo; Fairouz Kamareddine; Rob Nederpelt

October 7, 1994

1 Extended abstract

In this paper, we find for each term its reshuffled version where as many redexes as possible become visible and subject to contraction. We take terms modulo term reshuffling and present reduction as a relation between equivalence classes. The \(\lambda\)-cube of [Barendregt 92] extended with this reduction satisfies all its properties such as Church Rosser, Strong Normalisation and Subject Reduction (the latter depends on allowing definitions in contexts). We believe that treating terms modulo their reduction behaviour is a generalisation that is worth considering. An important phenomenon that results from studying the possible redexes is the ability to partition terms elegantly into parts which are informative as to where redexes occur.

1.1 Why term reshuffling? Every term \(A\) can be reshuffled to \(B\) where \(A\) is semantically equivalent to \(B\) and \(B\) has more visible redexes and more possible reductions.

Example 1.1 Let \(A \equiv ((\lambda x.p.(\lambda y.q.\lambda z.R.x y z)v)w)u\). The redexes in \(A\) are: \((\lambda y.q.\lambda z.R.x y z)v, (\lambda x.p.(\lambda y.q.\lambda z.R.x y z)v)w\) and \((\lambda x,R.w v z)u\) (which appears as \((\lambda x,R.x y z)[y := y][x := w])u\). The third redex is not classical nor is immediately visible, nor is subject to contraction without having first unfolded in \(\lambda x,R.x y z\) the two definitions that \(y = v\) and \(x = w\).

Now, take \(B \equiv (\lambda x.p.(\lambda y.q.(\lambda z.R.x y z)v)w)u\). The redexes in \(B\) are: \((\lambda y.q.\lambda z.R.x y z)u)w, (\lambda x.p.(\lambda y.q.\lambda z.R.x y z)u)v)w\) and \((\lambda z.R.x y z)u)\). All these redexes are classical, immediately visible and subject to contraction. Moreover, \(A =_\beta B\).

All the three redexes of \(A\) are needed in order to get its normal form and correspond to the redexes of \(B\). \(B\) is the reshuffled version of \(A\) we are looking for, notation: \(B \equiv TS(A)\).

The classical notation which we have used so far cannot extend the notion of redexes or enable reshuffling in an easy way. Our notation however, the item notation (see [KN 93] and [KN 9z]) can.

1.2 The item notation. In item notation, complex terms of the \(\lambda\)-cube are of the form \((A\omega)B\) where \(\omega \in \{\delta\} \cup \{O_{x};x\text{ is a variable and } O = \lambda \text{ or } \Pi\}\). We call \((A\omega)\) an item and \((A\delta)B\) means apply \(B\) to \(A\) (note the order). A redex starts with a \(\delta\)-item next to a \(\lambda\)-item.

---

*We are grateful for the Netherlands Computer Science Research Foundation (SION), the Netherlands Organisation for Scientific Research (NWO), the universities of Glasgow and Eindhoven and to the Basic Action for Research ESPRIT project "Types for Proofs and Programs", for their financial support.

\(^1\)Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, the Netherlands, email: bloo@win.tue.nl

\(^2\)Department of Computing Science, 17 Lilybank Gardens, University of Glasgow, Glasgow G12 8QQ, Scotland, email: fairouz@dcs.glasgow.ac.uk

\(^3\)Same address as Bloo. email: wsinrpn@win.tue.nl
Example 1.2 A of Example 1.1 is written \((u\delta)(w\delta)(P\lambda_z)(v\delta)(Q\lambda_y)(R\lambda_z)(z\delta)(y\delta)x\) in item notation. Here, the first two redexes, the classical redexes, correspond to \(\delta\lambda\)-pairs as follows: 
\((\lambda y.Q\cdot \lambda z.R.xyz)v\) corresponds to \((v\delta)(Q\lambda_y)(R\lambda_z)(z\delta)(y\delta)x\) and 
\((\lambda x.P.(\lambda y.Q\cdot \lambda z.R.xyz)v)w\) corresponds to \((w\delta)(P\lambda_z)(v\delta)(Q\lambda_y)(R\lambda_z)(z\delta)(y\delta)x\).

The third redex of \(A\) is obtained by matching \(\delta\) and \(\lambda\)-items. \((\lambda_{z;R}.xyz)u\) is visible as it corresponds to the matching \((u\delta)(R\lambda_z)\) where \((u\delta)\) and \((R\lambda_z)\) are separated by the segment \((w\delta)(P\lambda_z)(v\delta)(Q\lambda_y)\) which has the bracketing structure \[[[[\]]\]]\) (see Figure 1).

![Figure 1: Extended redexes in item notation](image)

Term reshuffling amounts to moving \(\delta\)-items to occur next to their matching \(\lambda\)-items. Hence \(A\) will be reshuffled to \((u\delta)(P\lambda_z)(v\delta)(Q\lambda_y)(u\delta)(R\lambda_z)(z\delta)(y\delta)x\) and Figure 1 changes to Figure 2. Now, \((u\delta)(R\lambda_z)(z\delta)(y\delta)x\) can be contracted in \(TS(A)\) via \(\rightarrow_\beta\). We use \(\rightarrow_\beta\) for one step shuffle reduction which is the sequence of term reshuffling followed by one-step ordinary reduction \(\rightarrow_\beta\). Hence, \(A \rightarrow_\beta (w\delta)(P\lambda_z)(v\delta)(Q\lambda_y)(u\delta)(y\delta)x\) because \(TS(A) \rightarrow_\beta (w\delta)(P\lambda_z)(v\delta)(Q\lambda_y)(u\delta)(y\delta)x\).

We generalise \(\rightarrow_\beta\) as follows: observe that if \(TS(A) \equiv TS(B)\) then there is a correspondence between the redexes of \(A\) and those of \(B\). We take \([A]\) to be \(\{B \mid TS(A) \equiv TS(B)\}\). All elements of \([A]\) are \(=_\beta\) and have somehow the same redexes. We say \([A] \rightarrow_\beta [A']\) iff \(\exists B \in [A] \exists B' \in [A']\) such that \(B \rightarrow_\beta B'\).

![Figure 2: Term reshuffling in item notation](image)

1.3 The need for definitions. Recall that in Example 1.1 when we explained the third redex of \(A\), we said that two definitions were unfolded in \(\lambda_{z;R}.xyz\). It turns out that this observation is necessary in order to show that the \(\lambda\)-cube extended with term reshuffling satisfies Subject Reduction. But then definitions are important on their own (see [BKN 9y] and [SP 93]). We show that the \(\lambda\)-cube extended with \(TS\), \(\rightarrow_\beta\) and definitions, preserves its original properties including Strong Normalisation and Subject Reduction and that term reshuffling preserves typing.
2 The item notation and the formal machinery

$I$ translates terms from classical notation to item notation such that ($O$ ranges over \{\lambda, \Pi\}): 

\[
\begin{align*}
I(A) & = A \quad \text{if } A \text{ is a variable or a constant} \\
I(Ox.A.B) & = (I(A)Ox)I(B) \quad \text{where } O \in \{\lambda, \Pi\} \\
I(AB) & = (I(B)\delta)I(A)
\end{align*}
\]

Bound and free variables and substitution are defined as usual. We write $BV(A)$ and $FV(A)$ to represent the bound and free variables of $A$ respectively. We write $A[x := B]$ to denote the term where all the free occurrences of $x$ in $A$ have been replaced by $B$. We take terms to be equivalent up to variable renaming and use $\equiv$ to denote syntactical equality of terms. We assume moreover, the Barendregt variable convention which is formally stated as follows:

Convention 2.1 (BC: Barendregt's Convention)

Names of bound variables will always be chosen such that they differ from the free ones in a term. Moreover, different $\lambda$'s have different variables as subscript. Hence, we will not have $(x\delta)(A\lambda_x)x$, but $(x\delta)(A\lambda_y)y$ instead.

The systems of the $\lambda$-cube are based on a set of pseudo-expressions $T$ defined by:

\[
T = \ast | \Box | V | (T\delta)T | (TO_V)T
\]

where $V$ is an infinite collection of variables over which $x, y, z, \ldots$ range. $\ast$ and $\Box$ are special constants called sorts over which $S, S_1, S_2, \ldots$ range. We take $A, B, C, a, b \ldots$ to range over pseudo-expressions.

Let $\omega \in \{\delta\} \cup \{O_x \mid x \in V\}$. A relation $\rightarrow$ on terms is compatible iff the following holds:

\[
\begin{align*}
A_1 \rightarrow A_2 & \quad (A_1\omega)B \rightarrow (A_2\omega)B \\
B_1 \rightarrow B_2 & \quad (A\omega)B_1 \rightarrow (A\omega)B_2
\end{align*}
\]

Definition 2.2 ((main) items, (main, $\delta O$-)segments, heart, weight)

- If $x$ is a variable and $A$ is a pseudo-expression then $(A\lambda_x), (A\Pi_x)$ and $(A\delta)$ are items (called $\lambda$-item, $\Pi$-item and $\delta$-item respectively). We use $s, s_1, s_i, \ldots$ to range over items.

- A concatenation of zero or more items is a segment. We use $\bar{s}, \bar{s_1}, \bar{s_i}, \ldots$ as meta-variables for segments. We write $\emptyset$ for the empty segment.

- Each pseudo-expression $A$ is the concatenation of zero or more items and a variable or constant: $A \equiv s_1s_2 \ldots s_nx$. These items $s_1, s_2, \ldots, s_n$ are called the main items of $A$, $x$ is called the heart of $A$, notation $\blacklozenge(A)$.

- Analogously, a segment $\bar{s}$ is a concatenation of zero or more items: $\bar{s} \equiv s_1s_2 \ldots s_n$; again, these items $s_1, s_2, \ldots, s_n$ (if any) are called the main items, this time of $\bar{s}$.

- A concatenation of adjacent main items $s_ms_{m+k}$, is called a main segment.

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

- The weight of a segment $\bar{s}$, weight($\bar{s}$), is the number of main items that compose the segment. Moreover, we define weight($\check{s}z$) = weight($\check{s}$).
When one desires to start a reduction on the basis of a δ-item and a λ-item, the matching of the δ and the λ in question is the important thing, even when the items are adjacent. Well-balanced segments separate matching δ and λ-items.

**Definition 2.3 (well-balanced segments)**

- The empty segment 0 is a well-balanced segment.
- If 3 is well-balanced, then $(Aδ)3(BO_λ)$ is well-balanced.
- 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 δ- (or O-)item corresponding with an opening (resp. closing) bracket.

**Definition 2.4 (match, δO- (reducible) couple, partner, partnered, bachelor)**

Let $A ∈ T$. Let $3 = s_1⋯s_n$ be a segment occurring in $A$.

- We say that $s_i$ and $s_j$ match, when $1 ≤ i < j ≤ n$, $s_i$ is a δ-item, $s_j$ is an O-item, and $s_{i+1}⋯s_{j-1}$ is a well-balanced segment.
- If $s_i$ and $s_j$ match, we call $s_is_j$ a δO-couple. A δλ-couple is called a reducible couple.
- If $s_i$ and $s_j$ match, we call $s_i$ and $s_j$ the partners or partnered items.
- All non-partnered O- (or δ-)items $s_k$ in $A$, are called bachelor O- (resp. δ-)items.
- A segment consisting of bachelor items only, is called a bachelor segment.

**Definition 2.5 (definitions, unfolding)**

- If $3$ is well-balanced not containing δπ-couples, then a segment $(Aδ)3(Bλ_π)$ occurring in a context is called a definition.
- Let $3$ be a well-balanced segment, We define the unfolding of $3$ in $A$, $[A]_3$, inductively as follows: $[A]_0 ≡ A$, $[A]_{(Bδ)s(Aλ_x)} ≡ [A[x := δ]]_3$ and $[A]_{3′} ≡ [[A]_{3′}]_3$. Note that substitution takes place from right to left and that when none of the binding variables of $3$ are free in $A$, then $[A]_3 ≡ A$.

We now introduce some general notions concerning typing rules which are the same as the usual ones when we do not allow definitions in the context (as is the case in the λ-cube). When definitions are present however, the notions are more general. Let $⊢$ be a typing relation and let $→$ be a reduction relation whose equivalence closure is $=$.

**Definition 2.6 (declarations, pseudocontexts, ⊢', →')**

1. A declaration $d$ is a λ-item $(Aλ_δ)$. subj$(d)$, pred$(d)$ and $d$ are $x$, $A$ and $0$ respectively.
2. For a definition $d ≡ (Bδ)s(Aλ_δ)$ we define subj$(d)$, pred$(d)$, $d$ and def$(d)$ to be $x$, $A$, $3$ and $B$ respectively.
3. We use $d, d_1, d_2,⋯$ to range over declarations and definitions.
4. A pseudocontext is a concatenation of declarations and definitions such that if \((A\lambda x)\) and \((B\lambda y)\) are two different main items of the pseudocontext, then \(x \neq y\). We use \(\Gamma, \Delta, \Gamma', \Gamma_1, \Gamma_2, \ldots\) to range over pseudocontexts.

5. For \(\Gamma\) a pseudocontext we define 
   \[ \text{dom}(\Gamma) = \{x \in V \mid (A\lambda x) \text{ is a main } \lambda\text{-item in } \Gamma \text{ for some } A\}, \]
   \[ \Gamma\text{-decl} = \{s \mid s \text{ is a bachelor main } \lambda\text{-item of } \Gamma\}, \]
   \[ \Gamma\text{-def} = \{s \mid \exists (A\delta)\exists_1(B\lambda x) \text{ is a main segment of } \Gamma \text{ where } \exists_1 \text{ is well-balanced}\}, \]
   Note that \(\text{dom}(\Gamma) = \{\text{subj}(d) \mid d \in \Gamma\text{-decl} \cup \Gamma\text{-def}\}\).

6. Define \(\subseteq\) between pseudocontexts as the least reflexive transitive relation satisfying:
   - \(\Gamma \subseteq \Gamma(C\lambda x)\Delta\) if no \(\lambda\)-item in \(\Delta\) matches a \(\delta\)-item in \(\Gamma\)
   - \(\Gamma \subseteq \Gamma d\Delta\) if \(d\) is a definition
   - \(\Gamma \supseteq \Gamma(A\lambda x)\Delta \subseteq \Gamma(D\delta)\exists(A\lambda x)\Delta\) if \((A\lambda x)\) is bachelor in \(\Gamma \supseteq \Gamma(A\lambda x)\Delta\), \(\exists\) is well-balanced

7. If \(A \rightarrow B\) then \(\Gamma(A\omega)\Gamma' \rightarrow \Gamma(B\omega)\Gamma'\) for \(\omega \in \{\delta\} \cup \{\lambda_v : v \in V\}\). \(\rightarrow\) between contexts is the reflexive transitive closure of \(\rightarrow\).

**Definition 2.7** (statements, judgements, \(<\) )

1. A statement is of the form \(A : B\), \(A\) and \(B\) are called the subject and the predicate of the statement respectively.

2. When \(\Gamma\) is a pseudocontext and \(A : B\) is a statement, we call \(\Gamma \vdash A : B\) a judgement, and write \(\Gamma \vdash A : B : C\) to mean \(\Gamma \vdash A : B \land \Gamma \vdash B : C\).

3. For \(\Gamma\) a pseudocontext and \(d \in \Gamma\text{-def} \cup \Gamma\text{-decl}\), \(\Gamma\) invites \(d\), notation \(\Gamma \prec d\), iff
   - \(\Gamma d\) is a pseudocontext
   - \(\Gamma \vdash \text{pred}(d) : S\) for some sort \(S\).
   - if \(d\) is a definition then \(\Gamma \vdash \text{def}(d) : \text{pred}(d)\).

**Definition 2.8** (Definitional equality) For all legal contexts \(\Gamma\) we define the binary relation \(\Gamma \vdash \cdot =_{\text{def}} \cdot\) to be the equivalence relation generated by
   - if \(A = B\) then \(\Gamma \vdash A =_{\text{def}} B\)
   - if \(d \in \Gamma\text{-def}\) and \(A, B \in T\) such that \(B\) arises from \(A\) by substituting one particular occurrence of \(\text{subj}(d)\) in \(A\) by \(\text{def}(d)\), then \(\Gamma \vdash A =_{\text{def}} B\).

**Definition 2.9** Let \(\Gamma\) be a pseudocontext and \(A\) be a pseudo-expression.

1. Let \(d, d_1, \ldots, d_n\) be declarations and definitions. We define \(\Gamma \vdash d\) and \(\Gamma \vdash d_1 \cdots d_n\) simultaneously as follows:
   - If \(d\) is a declaration: \(\Gamma \vdash d\) iff \(\Gamma \vdash \text{subj}(d) : \text{pred}(d)\).
   - If \(d\) is a definition: \(\Gamma \vdash d\) iff \(\Gamma \vdash \text{subj}(d) : \text{pred}(d) \land \Gamma \vdash \text{def}(d) : \text{pred}(d) \land \Gamma \vdash d \land \Gamma \vdash \text{subj}(d) =_{\text{def}} \text{def}(d)\).
• \( \Gamma \vdash d_1 \cdots d_n \) iff \( \Gamma \vdash d_i \) for all \( 1 \leq i \leq n \).

2. \( \Gamma \) is called legal if \( \exists P, Q \in \mathcal{T} \) such that \( \Gamma \vdash P : Q \).

3. \( A \in \mathcal{T} \) is called a \( \Gamma^+ \)-term if \( \exists B \in \mathcal{T} [\Gamma \vdash A : B \text{ or } \Gamma \vdash B : A] \).
   
   We take \( \Gamma^+ \)-terms = \( \{ A \in \mathcal{T} | \exists B \in \mathcal{T} [\Gamma \vdash A : B \lor \Gamma \vdash B : A] \} \).
   
   \( A \in \mathcal{T} \) is called legal if \( \exists \Gamma[A \in \Gamma^+ \text{-terms}] \).

4. We say that \( A \) is strongly normalising with respect to a reduction relation \( \rightarrow \) (written \( \text{SN}_\rightarrow(A) \)) iff every \( \rightarrow \)-reduction path starting at \( A \) terminates.

**Definition 2.10** We say that two terms \( A \) and \( B \) are semantically equivalent iff \( A = B \).

In the \( \lambda \)-cube of [Barendregt 92], the only declarations allowed are of the form \( (A \lambda_x) \). Therefore, \( \Gamma \vdash d \) is of the form \( \Gamma \vdash (A \lambda_x) \) and means that \( \Gamma \vdash A : S \) for some \( S \) and that \( x \) is fresh in \( \Gamma, A \). Moreover, for any \( d \equiv (A \lambda_x), d \equiv \emptyset, \text{ subj}(d) \equiv x \) and \( \text{ pred}(d) \equiv A \). Hence, in the next definition, \( d \) is a meta-variable for declarations only, \( =_{\text{def}} \) is the same as \( =_{\beta} \) (which is independent of \( \vdash \)) and the reduction relation is \( \rightarrow_{\beta} \).

**Definition 2.11** (Axioms and rules of the \( \lambda \)-cube: \( d \) is a declaration, \( =_{\text{def}} \) is \( =_{\beta} \))

\[
\begin{align*}
\text{(axiom)} & \quad \Gamma \vdash * : \Box \\
\text{(start rule)} & \quad \frac{\Gamma \vdash d}{\Gamma \vdash \text{ subj}(d) : \text{ pred}(d)} \\
\text{(weakening rule)} & \quad \frac{\Gamma \vdash d}{\Gamma \vdash D : E} \\
\text{(application rule)} & \quad \frac{\Gamma \vdash F : (A \Pi_x)B \quad \Gamma \vdash a : A}{\Gamma \vdash (a \beta)F : B[x := a]} \\
\text{(abstraction rule)} & \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash (A \lambda_x) : S}{\Gamma \vdash (A \lambda_x)b : (A \Pi_x)B} \\
\text{(conversion rule)} & \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : S \quad \Gamma \vdash B =_{\text{def}} B'}{\Gamma \vdash A : B'} \\
\text{(formation rule)} & \quad \frac{\Gamma \vdash A : S_1 \quad \Gamma \vdash (A \lambda_x) : S_2}{\Gamma \vdash (A \Pi_x)B : S_2 \text{ if } (S_1, S_2) \text{ is a rule}}
\end{align*}
\]

Each of the eight systems of the \( \lambda \)-cube is obtained by taking the \((S_1, S_2)\) rules allowed from a subset of \{(*, *), (*, \emptyset), (\emptyset, *), (\emptyset, \emptyset)\}. The basic system is the one where \((S_1, S_2) = (*, *)\) is the only possible choice. All other systems have this version of the formation rules, plus one or more other combinations of \((*, \emptyset), (\emptyset, *)\) and \((\emptyset, \emptyset)\) for \((S_1, S_2)\). Here is the table which presents the eight systems of the \( \lambda \)-cube:
### System 

<table>
<thead>
<tr>
<th>System</th>
<th>Set of specific rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda )</td>
<td>((<em>,</em>))</td>
</tr>
<tr>
<td>( \lambda 2 )</td>
<td>((<em>,</em>))</td>
</tr>
<tr>
<td>( \lambda P )</td>
<td>((<em>,</em>))</td>
</tr>
<tr>
<td>( \lambda P 2 )</td>
<td>((<em>,</em>))</td>
</tr>
<tr>
<td>( \lambda \omega )</td>
<td>((<em>,</em>))</td>
</tr>
<tr>
<td>( \lambda P \omega )</td>
<td>((<em>,</em>))</td>
</tr>
<tr>
<td>( \lambda P \omega = \lambda C )</td>
<td>((<em>,</em>))</td>
</tr>
</tbody>
</table>

3 Term reshuffling

In this section we rewrite terms so that all the newly visible redexes can be subject to \( \rightarrow_\beta \). We shall show in this section that this term rewriting is correct in the sense that \( A =_\beta TS(A) \). In Section 4, we show that this term reshuffling preserves reduction in the sense that if \( A \sim_\beta B \) then \( TS(A) \sim_\beta TS(B) \) and \( \exists B' \in [B] [TS(A) \rightarrow_\beta B'] \). In Section 5, we show that this term reshuffling preserves typing in the sense that if \( \Gamma \vdash_\text{th} A : B \) then \( \Gamma \vdash_\text{th} TS(A) : B \).

Let us go back to the definition of \( \delta O \)-couples. Recall that if \( \bar{s} \equiv s_1 \cdots s_m \) for \( m > 1 \) where \( s_1 s_m \) is a \( \delta O \)-couple then \( s_2 \cdots s_{m-1} \) is a well-balanced segment, \( s_1 \) is the \( \delta \)-item of the \( \delta O \)-couple and \( s_m \) is its \( O \)-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 \( s_2 \cdots s_{m-1} s_1 s_m \). As legal terms and contexts of the \( \lambda \)-cube contain no \( \delta \Pi \)-couples, we focus only on \( \delta \lambda \)-couples.

**Example 3.1** The term \( A \equiv (u\delta)(w\delta)(P\lambda x)(v\delta)(Q\lambda y)(R\lambda z)(z\delta)(y\delta)x \) is reshuffled to \( TS(A) \equiv (w\delta)(P\lambda z)(v\delta)(Q\lambda y)(u\delta)(R\lambda z)(z\delta)(y\delta)x \) by moving the item \( (w\delta) \) to the right. Such a reshuffling is not easy to describe in the classical notation. That is, it is difficult to describe how \(((\lambda x.P.(\lambda y.Q.\lambda z.R.xyz)u)v)w \) is reshuffled to \( (\lambda x.P.(\lambda y.Q.\lambda z.R.xyz)u)v)w \).

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 3.2** Let \( \bar{s} \) be a well-balanced segment.

1. If none of its binding variables is free in \( A \), then \( [A]_{\bar{s}} \equiv A \) and \( \bar{s}_1(A\delta)\bar{s}B =_\beta \bar{s}_1 \bar{s}(A\delta)B \).

2. If \( \bar{s} \) contains no \( \delta \Pi \)-couples then \( [A]_{\bar{s}} =_\beta \bar{s}A \).

Proof: \( [A]_{\bar{s}} =_\beta \bar{s}A \) is obvious and the rest is by induction on \( \text{weight}(\bar{s}) \). \( \square \)

To reshuffle terms, we study the classes of partnered and bachelor items in a term.

3.1 Partitioning the term into bachelor and well-balanced segments

With Definition 2.4, we may categorize the main items of a term \( A \) into different parts to which the partnered or bachelor items belong:

**Lemma 3.3** Let \( \bar{s} \) be the body of a term \( A \). Then the following holds:

1. Each bachelor main \( O \)-item in \( \bar{s} \) precedes each bachelor main \( \delta \)-item in \( \bar{s} \).
2. The removal from \( \mathfrak{s} \) of all bachelor main items, leaves behind a well-balanced segment.

3. The removal from \( \mathfrak{s} \) of all main \( \delta\Omega \)-couples, leaves behind a \( \underbrace{\Omega \cdots \Omega \delta \cdots \delta}_{n \times m} \)-segment, consisting of all bachelor main \( \Omega \)- and \( \delta \)-items.

**Proof:** 1 is by induction on weight(\( \mathfrak{s} \)) for \( \mathfrak{s} \equiv \mathfrak{s}'(B\Omega x)s'' \) and \( (B\Omega x) \) bachelor in \( \mathfrak{s} \). 2 and 3 are by induction on weight(\( \mathfrak{s} \)). \( \square \)

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

**Corollary 3.4** For each non-empty segment \( \mathfrak{s} \), there is a unique partitioning in segments \( \overline{s_0}, \overline{s_1}, \ldots, \overline{s_n} \), such that

1. \( \mathfrak{s} \equiv \overline{s_0} \overline{s_1} \cdots \overline{s_n} \),
2. For all \( 0 \leq i \leq n, \overline{s_i} \) is well-balanced for even \( i \) and \( \overline{s_i} \) is bachelor in \( \mathfrak{s} \) for odd \( i \).
3. For all \( 0 \leq i, j \leq n \): if \( \overline{s_i} \) contains bachelor \( \Omega \)-items and \( \overline{s_j} \) contains bachelor \( \delta \)-items then \( i \leq j \).
4. \( \overline{s_{2n}} \neq \emptyset \) for \( n > 0 \). \( \square \)

**Example 3.5** The partitioning of \( (A\lambda x)(B\lambda y)(C\delta)(D\Pi z)(E\lambda u)(F\delta)(a\delta)(b\delta)(c\lambda u)(d\lambda w)(e\delta) \) is:

- well-balanced segment \( \overline{s_0} \equiv \emptyset \),
- bachelor segment \( \overline{s_1} \equiv (A\lambda x)(B\lambda y) \),
- well-balanced segment \( \overline{s_2} \equiv (C\delta)(D\Pi z) \),
- bachelor segment \( \overline{s_3} \equiv (E\lambda u)(F\delta) \),
- well-balanced segment \( \overline{s_4} \equiv (a\delta)(b\delta)(c\lambda u)(d\lambda w) \),
- bachelor segment \( \overline{s_5} \equiv (e\delta) \).

### 3.2 A reshuffling procedure and its properties

**Definition 3.6** The reshuffling function \( TS \) is defined such that:

\[
\begin{align*}
TS(\mathfrak{s}x) &= \text{df} \quad TS(\mathfrak{s})x \\
TS((A\Omega x)s) &= \text{df} \quad (TS(A)\Omega x)TS(\mathfrak{s}) & \text{if } (A\Omega x) \text{ is bachelor in } \mathfrak{s} \\
TS((A_1\delta) \cdots (A_n\delta)x) &= \text{df} \quad (TS(A_1)\delta) \cdots (TS(A_n)\delta) & \text{if } \mathfrak{s} \text{ is well-balanced,} \\
TS(((\delta\Omega)(B\Omega x))) &= \text{df} \quad TS(\mathfrak{s})(TS(A)\delta)(TS(B)\Omega x) & \text{if } \mathfrak{s} \text{ is well-balanced}
\end{align*}
\]

With term reshuffling, well-balanced segments must be rewritten so that \( \delta\Omega \)-couples become \( \delta\Omega \)-segments and all bachelor main \( \delta \)-items are moved to the right of all well-balanced segments. Hence, for any \( A \), \( TS(A) \equiv \overline{s_0} \overline{s_1}x \) where \( \overline{s_1} \) consists of all bachelor main \( \delta \)-items of \( A \). \( \overline{s_0} \) is of the form \( \overline{s_2} \overline{s_3} \cdots \overline{s_n} \) where \( \overline{s_i} \) is either a \( \delta\Omega \)-segment or a bachelor main \( \Omega \)-item.
Take $P \equiv (A_1 \ldots \lambda_z)(B \delta)(C_1 \ldots \lambda_x)(D \lambda_y)(F \delta)(G \lambda_u)(H \delta)(I \delta)x$. Can we move all the bachelor main $O$-items (to the left or right)? The answer is no. For example, $D$ may contain variables bound by the $\lambda_x$ and we cannot rewrite $P$ as $(A_1 \ldots \lambda_y)(B \delta)(C_1 \ldots \lambda_x)(F \delta)(G \lambda_u)(H \delta)(I \delta)x$ (assuming $A \ldots I$ are already reshuffled). Moreover, in $P$, $B$ and $C$ may contain variables bound by $\lambda_z$ so $\lambda_z$ cannot move to the right of $(B \delta)(C \lambda_z)$. Hence, in a term $A$, all main bachelor $O$-items will occur in the same position as in $TS(A)$. Now, let us show the properties of $TS$.

Lemma 3.7

1. For all pseudo-expressions $M$, $TS(M)$ is well defined.

2. If $\bar{s}$ is well-balanced, then $TS(\bar{s}A) \equiv TS(\bar{s})TS(A)$ and $TS(\bar{s})$ is well-balanced.

Proof: 1. Every time a rule $TS(M)$ is used, weights of the resulting terms become shorter or $TS$ disappears. 2. By induction on $\bar{s}$. Let IH stands for Induction Hypothesis.

Lemma 3.8 For a term $A$, $TS(A) \equiv TS(\bar{s})TS((B_1 \delta) \cdots (B_n \delta)x$ where $x \equiv \forall(A)$, $\bar{s}$ consists of the term reshufflings of all bachelor main $\delta$-items of $A$ and $\bar{s}$ is a sequence of term reshufflings of main $\delta O$-segments and bachelor main $O$-items.

Proof: Induction on weight($A$).

- $A \equiv x$, then nothing to prove.

- $A \equiv (B_0 \delta)C$ or $A \equiv \bar{s}C$ where $\bar{s}$ well-balanced, use IH on $C$.

- $A \equiv (B_1 \delta) \cdots (B_n \delta)\bar{s}C$ where $\bar{s}$ well-balanced and ($\bar{s} \not= \emptyset$ or $(C \equiv x$ and $n > 0$)). $TS(A) \equiv TS(\bar{s})TS((B_1 \delta) \cdots (B_n \delta)C)$. By lemma 3.7, $\bar{s}$ is well-balanced and hence by IH on $\bar{s}$, $TS(\bar{s})$ is a sequence of $\delta O$-segments, if $\bar{s} \not= \emptyset$ then by IH on $(B_1 \delta) \cdots (B_n \delta)C$ we are done, else $A \equiv x$ and by IH, $TS((B_1 \delta) \cdots (B_n \delta)C) \equiv (TS(B_1) \delta) \cdots (TS(B_n) \delta)x$ which also has the required format. □

Lemma 3.9 For all pseudo-expressions $A, B$ and variable $x$:

1. $TS(A) \equiv TS(TS(A))$

2. $TS(A[x := B]) \equiv TS(TS(A)[x := TS(B)])$

3. $A =_\beta TS(A)$

Proof: 1. By induction on the structure of $A$. Case $A \equiv x$, then $A \equiv TS(A)$. Case $A \equiv (B_0 \delta)C$, or $A \equiv (B_1 \delta) \cdots (B_n \delta)\bar{s}C$ where $\bar{s}$ well-balanced and ($\bar{s} \not= \emptyset$ or $(C \equiv x$ and $n > 0$)), use IH. 2. By induction on the structure of $A$, using 1. 3. By induction on the number of symbols in $A$. □

Corollary 3.10 For all pseudo-expressions $A, B$: $TS(A) =_\beta TS(B)$ iff $A =_\beta B$. □

Definition 3.11 (Shuffle Class)

For a pseudo-expression $A$, we define $[A]$ to be $\{ B \mid TS(A) \equiv TS(B) \}$.

Lemma 3.12 for all $A \in [B]$, $A$ is semantically equivalent to $B$. □
4 Equivalence classes and shuffle $\beta\Pi$-reduction

**Definition 4.1 (Extended redexes and $\beta$-reduction in item notation)**
An extended redex is a $\delta\lambda$-couple (i.e. is of the form $(C\delta)\overline{s}(B \lambda z)A$ where $\overline{s}$ is well-balanced).

General one-step $\beta$-reduction $\sim_\beta$ is the least compatible relation generated out of:

$$ A \sim_\beta A' \iff \exists B \in [A] \exists B' \in [A'][B \rightarrow_\beta B'] $$

Note that $\sim_\beta$ is compatible and transitive because $\rightarrow_\beta$ is. General $\sim_\beta$ is the reflexive and transitive closure of $\sim_\beta$ and $\approx_\beta$ is the least equivalence relation generated by $\sim_\beta$.

**Example 4.2** Let $A \equiv (z\delta)(w\delta)(u \lambda z)(x \lambda y)y$. Then $[A] = \{A, (w\delta)(u \lambda z)(z\delta)(x \lambda y)y\}$. Moreover, $A \sim_\beta A'$ for any $A' \in \{(w\delta)(u \lambda z)(z\delta)(w \lambda y)y\}$.

**Lemma 4.3** If $A \sim_\beta B$ then for all $A' \in [A]$, for all $B' \in [B]$, $A' \approx_\beta B'$.

**Proof:** As $A \sim_\beta B$ then $\exists A_1 \in [A], \exists B_1 \in [B] \rightarrow_\beta B_1$. Let $A', B' \in [A], [B]$ respectively. Hence $A_1 \in [A'], B_1 \in [B']$, $A_1 \rightarrow_\beta B_1$. So $A' \approx_\beta B'$.

**Remark 4.4** Now it is not in general true that $A \approx_\beta B \Rightarrow \exists A' \in [A] \exists B' \in [B] [A' \rightarrow_\beta B']$.

This can be seen by the following counterexample:

$$ A \equiv ((a \lambda u)(a \lambda v)\delta)((a \Pi u)(a \Pi v)a \lambda x)(w \delta)(w \delta)x \sim_\beta (w \delta)(w \delta)(a \lambda u)(a \lambda v)u \sim_\beta B \equiv (w \delta)(w \delta)u. $$

But $[A]$ has the elements $A, (w \delta)(a \lambda u)(a \lambda v)u$. If $A' \in [A]$ then the only $\rightarrow_\beta$ reduct of $A'$ is $(w \delta)(w \delta)(a \lambda u)(a \lambda v)u$, which doesn’t $\rightarrow_\beta$-reduce to $B$. In Lemma 4.12 however, we show that there is a correspondence between $\approx_\beta$ on classes and $\rightarrow_\beta$ on terms.

Let $\rightarrow_\beta$ be the least compatible relation generated by $(B_1 \delta)\overline{s}(B_2 \lambda z)B_3 \rightarrow_\beta \overline{s}(B_3[z := B_1])$ for $\overline{s}$ is well-balanced. Let $\sim_\beta$ be its reflexive and transitive closure and $\sim_\beta$ be the least equivalence relation closed under $\sim_\beta$. $\sim_\beta$ has been used in [BKN 9y]. We will use [BKN 9y] to obtain Strong Normalisation for the present paper.

**Lemma 4.5** $\rightarrow_\beta \subseteq \approx_\beta \subseteq \sim_\beta$.

**Proof:** It suffices to show $(A \delta)(B \lambda z)C \rightarrow_\beta C[z := A]$ and $(A \delta)\overline{s}(B \lambda z)C \sim_\beta \overline{s}C[z := A]$. But $(A \delta)(B \lambda z)C \equiv (A \delta)\emptyset(B \lambda z)C \rightarrow_\beta \emptyset C[z := A] \equiv C[z := A]$, and $(A \delta)\overline{s}(B \lambda z)C \in [\overline{s}(A \delta)(B \lambda z)C], \overline{s}(A \delta)(B \lambda z)C \rightarrow_\beta \overline{s}C[z := A]$ hence $(A \delta)\overline{s}(B \lambda z)C \sim_\beta \overline{s}C[z := A]$.

**Lemma 4.6** If $A \sim_\beta B$ then $TS(A) \sim_\beta TS(B)$.

**Proof:** We show $\sim_\beta$. By Lemma 3.9, $[TS(A)] =_\beta [A]$; so $A \sim_\beta B \Rightarrow TS(A) \sim_\beta TS(B)$.

**Remark 4.7** Note that this lemma does not hold for $\approx_\beta$. Take for example $A$ and $B$ where $A \equiv ((z \lambda u)(z \lambda v)\delta)(v \lambda x)(y \delta)(y \delta)x$ and $B \equiv (y \delta)(y \delta)(z \lambda u)(z \lambda v)v$. It is obvious that $A \approx_\beta B$ yet $TS(A) \equiv A \not\rightarrow_\beta TS(B) \equiv (y \delta)(z \lambda u)(y \delta)(z \lambda v)v$.

**Lemma 4.8** If $(B \delta)\overline{s}(C \lambda z)D$ is an extended redex in $A$, then $TS(B)\delta(TS(C)\lambda z)TS(D)$ is a classical redex in $TS(A)$, and if $(B \delta)(C \lambda z)D$ is a classical redex in $TS(A)$ then there exist $B', C', D'$'s such that $TS(B') \equiv B, TS(C') \equiv C, TS(D') \equiv D$ and $(B' \delta)\overline{s}(C' \lambda z)D'$ is an extended redex in $A$.

**Proof:** by induction on the number of symbols in $A$.
Lemma 4.9 If $A \sim_{\beta} B$ or $A \rightarrow_{\beta} B$ then $A =_{\beta} B$.

Proof: For $\sim_{\beta}$: say $A' \in [A], B' \in [B], A' \rightarrow_{\beta} B'$. Then by lemma 3.9: $A =_{\beta} TS(A) \equiv TS(A') =_{\beta} A' =_{\beta} B' =_{\beta} TS(B') \equiv TS(B) =_{\beta} B$.

For $\rightarrow_{\beta}$: it suffices to consider the case $A =_{\beta} TS(A) \equiv TS(A') =_{\beta} A' =_{\beta} B' =_{\beta} TS(B') \equiv TS(B) =_{\beta} B$.

We shall prove the lemma by induction on $weight(\bar{s})$.

For $\rightarrow_{\beta}$: it suffices to consider the case $A =_{\beta} TS(A)$.

Proof: By induction on $weight(\bar{s})$.

Case $weight(\bar{s}) = 0$ then obvious as $\rightarrow_{\beta}$ coincides with $\rightarrow_{\beta}$ in this case. Assume the property holds when $weight(\bar{s}) = 2n$. Take $s$ such that $I/eight(s) = 2n + 2$.

Assume $x \neq y$ (if necessary, use renaming).

As $\bar{s}(E[x := C]) \rightarrow_{\beta} \bar{s}(\bar{s}^r(E[x := C])[y := C'])$, we get by IH and compatibility that $B =_{\beta} B' =_{\beta} \bar{s}^r(E[x := C])[y := C'] \equiv \bar{s}^r(E[x := C])[y := C'] \equiv B''$.

Moreover, $A =_{\beta} C\delta\bar{s}(D'y)\bar{s}^r(D\lambda x)E \rightarrow_{\beta} C\delta\bar{s}(D'y)\bar{s}^r(D\lambda x)E[y := C'] \equiv BC$.

Now, $B' \rightarrow_{\beta} C\delta\bar{s}(D'y)\bar{s}^r(D\lambda x)E[y := C'] \equiv B'$. Hence by IH, $A =_{\beta} B'$.

Therefore, $A =_{\beta} B', B' =_{\beta} B''$ and $B =_{\beta} B''$, hence $A =_{\beta} B$.

Corollary 4.10 Let $\rightarrow$ be $\sim_{\beta}$ or $\rightarrow_{\beta}$.

1. If $A \rightarrow B$ then $A =_{\beta} B$.

2. $A \equiv_{\beta} B$ iff $A =_{\beta} B$.

Theorem 4.11 (The general Church Rosser theorem for $\sim_{\beta}$) Let $\rightarrow$ be $\sim_{\beta}$ or $\rightarrow_{\beta}$.

If $A \rightarrow B$ and $A \rightarrow C$, then there exists $D$ such that $B \rightarrow D$ and $C \rightarrow D$.

Proof: As $A \rightarrow B$ and $A \rightarrow C$ then by Corollary 4.10, $A =_{\beta} B$ and $A =_{\beta} C$. Hence, $B =_{\beta} C$ and by CR for $\rightarrow_{\beta}$, there exists such that $B \rightarrow_{\beta} D$ and $C \rightarrow_{\beta} D$. But, $A \rightarrow_{\beta} B$ implies $A \rightarrow_{\beta} B$. Hence we are done.

We can have $TS(C) \rightarrow_{\beta} D$ where $D \neq TS(D)$. Consider for example the terms $C \equiv ((z\lambda u)(z\lambda v)(w\lambda x)(y\delta)(y\delta)z x d E$ and $D \equiv (y\delta)(y\delta)(z\lambda u)(z\lambda v)$. Then $TS(A) \equiv C \rightarrow_{\beta} D$ whereas $TS(D) \equiv (y\delta)(z\lambda u)(y\delta)(z\lambda v)$. But we still can show that in a certain sense, term reshuffling preserves $\beta$-reduction.

Lemma 4.12 If $A, B \in T$ and $A \sim_{\beta} B$ then $(\exists B' \in [B])[TS(A) \rightarrow_{\beta} B']$. In other words, the following diagram commutes:

$$\begin{array}{c}
A \\
\downarrow \beta \\
TS(A) \\
\downarrow \beta \\
B' \in [B]
\end{array}$$

Proof: Prove by induction on the structure of $A'$ that if $A' \rightarrow_{\beta} B' \in [B]$, then for some $B''$, $TS(A') \rightarrow_{\beta} B'' \in [B]$.

Corollary 4.13 If $A \sim_{\beta} B$ then there exist $A_0, A_1, \ldots, A_n$ such that $[A \equiv A_0] \land (TS(A_0) \rightarrow_{\beta} A_1) \land (TS(A_1) \rightarrow_{\beta} A_2) \land \cdots \land (TS(A_{n-1}) \rightarrow_{\beta} A_n \in [B])]$.

Proof: By induction on $\sim_{\beta}$.
5 The \(\lambda\)-cube with equivalence classes, definitions and shuffle \(\beta\Pi\)-reduction

If we extend the \(\lambda\)-cube with \(\sim_\beta\) then Subject Reduction fails. That is: \(\Gamma \vdash A : B\) and \(A \sim_\beta A' \neq \Gamma \vdash A' : B'\).

Example 5.1 (SR does not hold in \(\lambda_2\) using \(\sim_\beta\))
\((*\lambda_\beta)(\beta_\lambda y') \vdash_{\lambda_2} (y'\delta)(\beta_\delta)(*\lambda_\alpha)(\alpha_\lambda y)(y'\delta)(\alpha_\lambda z) : \beta.
Moreover, \((y'\delta)(\beta_\delta)(*\lambda_\alpha)(\alpha_\lambda y)(y'\delta)(\alpha_\lambda z) \sim_\beta (\beta_\delta)(*\lambda_\alpha)(\alpha_\lambda z)\).

Yet, \((*\lambda_\beta)(\beta_\lambda y') \vdash_{\lambda_2} (y'\delta)(\beta_\delta)(*\lambda_\alpha)(\alpha_\lambda z) : \beta.
Even, \((*\lambda_\beta)(\beta_\lambda y') \vdash_{\lambda_2} (y'\delta)(\beta_\delta)(*\lambda_\alpha)(\alpha_\lambda z) : \tau\) for any \(\tau\).

This is because \((\alpha_\lambda z) : (\alpha_\Pi\lambda)\alpha\) and \(y : \beta\) yet \(\alpha\) and \(\beta\) are unrelated and hence we fail in firing the application rule to find the type of \((y'\delta)(\alpha_\lambda z)\). Looking closer however, one finds that \((\beta_\delta)(*\lambda_\alpha)\) is defining \(\alpha\) to be \(\beta\), yet no such information can be used to combine \((\alpha_\Pi\lambda)\alpha\) with \(\beta\). Definitions take such information into account. Finally note that failure of SR in \(\lambda_2\), means its failure in \(\lambda_2P, \lambda_2\omega\) and \(\lambda C\).

Example 5.2 (SR does not hold in \(\lambda P\) using \(\sim_\beta\))
\((*\lambda_\beta)(\sigma \lambda y)((\sigma \Pi \lambda) \alpha) \vdash_{\lambda P} ((t \delta)Q \lambda y)((t \delta)Q \lambda y)((t \delta)Q \lambda z)Z : (t \delta)Q.

And \((N \delta)(t \delta)(\sigma \lambda y)((t \delta)Q \lambda y)((t \delta)Q \lambda z)Z \sim_\beta (t \delta)(\sigma \lambda y)((N \delta)(t \delta)Q \lambda z)Z\).

Now, \(N : (t \delta)Q, t : \sigma, y : (t \delta)Q, z : \sigma, (t \delta)Q \neq (t \delta)Q)\).
\((*\lambda_\beta)(\sigma \lambda y)((\sigma \Pi \lambda) \alpha) \vdash_{\lambda P} ((t \delta)(\sigma \lambda y)((N \delta)(t \delta)Q \lambda z)Z) : \tau\) for any \(\tau\).

Here again the reason of failure is similar to the above example. At one stage, we need to match \((t \delta)Q\) with \((t \delta)Q\) but this is not possible even though we do have the definition segment: \((t \delta)(\sigma \lambda y)\) which defines \(x\) to be \(t\). All this calls for the need to use these definitions. Finally note that failure of SR in \(\lambda P\), means its failure in \(\lambda P2, \lambda P\omega\) and \(\lambda C\).

We conjecture that Subject Reduction is valid for \(\lambda_\omega\) and \(\lambda C\) with \(\sim_\beta\) and that the proof goes as in [BKN 9y] for \(\sim_\beta\).

We extend the \(\lambda\)-cube with definitions, \(\sim_\beta\) and equivalence classes modulo \(TS\). Contexts now consist of declarations \((A_\lambda x)\) and definitions. We take the typing rules \(\vdash_{\text{sh}}\) to be exactly those \(\vdash\) with the addition of the definition rule:
\[
\frac{\Gamma, \Delta \vdash_{\text{sh}} C : D}{\Gamma, \Delta \vdash_{\text{sh}} d \vdash_{\text{sh}} C : [D]_d \quad \text{if } d \text{ is a definition}}
\]

Now, we proceed to show the properties of \(\vdash_{\text{sh}}\).

Lemma 5.3 (Free variable lemma for \(\vdash_{\text{sh}}\))
Let \(\Gamma\) be a legal context such that \(\Gamma \vdash_{\text{sh}} B : C\). Then the following holds:

1. If \(d\) and \(d'\) are two different elements of \(\Gamma - \text{decl} \cup \Gamma - \text{def}\), then \(\text{subj}(d) \neq \text{subj}(d')\).
2. \(\text{FV}(B), \text{FV}(C) \subseteq \text{dom}(\Gamma)\).
3. For \(s_1\) a main item of \(\Gamma\), \(\text{FV}(s_1) \subseteq \{\text{subj}(d) \mid d \in \Gamma - \text{decl} \cup \Gamma - \text{def}, d \text{ is to the left of } s_1 \text{ in } \Gamma\}\).

Proof: All by induction on the derivation of \(\Gamma \vdash_{\text{sh}} B : C\).

Lemma 5.4 (Start Lemma for \(\vdash_{\text{sh}}\))
Let \(\Gamma\) be a legal context. Then \(\Gamma \vdash_{\text{sh}} \emptyset : \emptyset\) and \(\forall d : \emptyset \in \emptyset \in \Gamma \vdash_{\text{sh}} d\).

Proof: \(\Gamma\) is legal \(\Rightarrow \exists B, C[\Gamma \vdash_{\text{sh}} B : C]\); use induction on the derivation \(\Gamma \vdash_{\text{sh}} B : C\).
Lemma 5.5 (Transitivity Lemma for $t_{\text{sh}}$)

Let $\Gamma$ and $\Delta$ be legal contexts. Then: $[\Gamma \vdash_{t_{\text{sh}}} \Delta \wedge \Delta \vdash_{t_{\text{sh}}} A : B] \Rightarrow [\Gamma \vdash_{t_{\text{sh}}} A : B].$

Proof: Induction on the derivation $\Delta \vdash_{t_{\text{sh}}} A : B$ using the start lemma. By the compatibility of $\Gamma \vdash_{t_{\text{sh}}} C = \text{def } D$ it follows that if $d \in \Delta$ and $D$ arises from $C$ by $\text{def}(d)$, then $\Gamma \vdash_{t_{\text{sh}}} C = \text{def } D$ and hence $\Delta \vdash_{t_{\text{sh}}} C = \text{def } D$, now use induction on the derivation of $\Delta \vdash_{t_{\text{sh}}} A : B$. $\square$

Lemma 5.6 (Definition-shuffling for $t_{\text{sh}}$) Let $d$ be a definition.

1. If $\Gamma d \vdash_{t_{\text{sh}}} C = \text{def } D$ then $\Gamma \vdash (\text{def}(d) \delta)(\text{pred}(d) \lambda_{\text{subj}(d)}) \Delta \vdash_{t_{\text{sh}}} C = \text{def } D$.

2. If $\Gamma d \vdash_{t_{\text{sh}}} C : D$ then $\Gamma \vdash (\text{def}(d) \delta)(\text{pred}(d) \lambda_{\text{subj}(d)}) \Delta \vdash_{t_{\text{sh}}} C : D$.

Proof: 1. is by induction on the generation of $\Gamma d \vdash_{t_{\text{sh}}} C = \text{def } D$. 2. is by induction on the derivation of $\Gamma d \vdash_{t_{\text{sh}}} C : D$ using 1. for conversion. $\square$

Lemma 5.7 (Thinning for $t_{\text{sh}}$)

1. If $\Gamma_1 \Gamma_2 \vdash_{t_{\text{sh}}} A = \text{def } B$, $\Gamma_1 \Delta \Gamma_2$ is a legal context, then $\Gamma_1 \Gamma_2 \vdash_{t_{\text{sh}}} A = \text{def } B$.

2. If $\Gamma$ and $\Delta$ are legal contexts such that $\Gamma \subseteq \Delta$ and $\Gamma \vdash_{t_{\text{sh}}} A : B$, then $\Delta \vdash_{t_{\text{sh}}} A : B$.

Proof: 1. is by induction on the derivation $\Gamma_1 \Gamma_2 \vdash_{t_{\text{sh}}} A = \text{def } B$. 2. is done by showing:

- If $\Gamma_1 \Delta \vdash_{t_{\text{sh}}} A : B$, $\Gamma \vdash_{t_{\text{sh}}} C : S$, $x$ is fresh, and no $\lambda$-item in $\Gamma$, then also $\Gamma(C\lambda_x) \vdash_{t_{\text{sh}}} A : B$. We show this by induction on the derivation $\Gamma \vdash_{t_{\text{sh}}} A : B$ using 1. for conversion.

- If $\Gamma_3 \Delta \vdash_{t_{\text{sh}}} A : B$, $\Gamma_3 \vdash_{t_{\text{sh}}} C : D$, $[C]_{\Delta} \equiv C$, $x$ is fresh, $\exists$ is well-balanced, then also $\Gamma(C\delta)\exists(D\lambda_x) \Delta \vdash_{t_{\text{sh}}} A : B$. We show this by induction on $\Gamma_3 \Delta \vdash_{t_{\text{sh}}} A : B$. In the case of (start) where $\Gamma(A\lambda_y)\exists(B\lambda_x) \vdash_{t_{\text{sh}}} A$ comes from $\Gamma_3 \vdash_{t_{\text{sh}}} A : B : S$, $[A]_{\Delta} \equiv A$, $x$ fresh, then $[A]_{\Delta}(C\delta)\exists(D\lambda_x) \equiv A$ because $x$ fresh and $\Gamma(C\delta)\exists(D\lambda_x) \vdash_{t_{\text{sh}}} A : B : S$ by IH.

- If $\Gamma_3(\lambda\lambda_x) \Delta \vdash_{t_{\text{sh}}} B : C$, $(\lambda\lambda_x)$ bachelor, $\exists$ well-balanced, $\Gamma_3 \vdash_{t_{\text{sh}}} D : A$, $[D]_{\Delta} \equiv D$, then $\Gamma(D\delta)\exists(\lambda\lambda_x) \Delta \vdash_{t_{\text{sh}}} B : C$. We show this by induction on $\Gamma_3(\lambda\lambda_x) \Delta \vdash_{t_{\text{sh}}} B : C$. $\square$

Lemma 5.8 (Substitution lemma for $t_{\text{sh}}$) Let $d$ be a definition.

1. If $\Gamma d \vdash_{t_{\text{sh}}} A = \text{def } B$, $A$ and $B$ are $\Gamma d$-legal terms, then $\Gamma \vdash_{t_{\text{sh}}} A[\text{subj}(d) := \text{def}(d)] = \text{def } B[\text{subj}(d) := \text{def}(d)]$

2. If $B$ is a $\Gamma d$-legal term, then $\Gamma \vdash_{t_{\text{sh}}} B = \text{def } [B]_d$

3. If $\Gamma(A\delta)\exists(B\lambda_x) \Delta \vdash_{t_{\text{sh}}} C : D$ then $\Gamma \exists(\Delta[x := A]) \vdash_{t_{\text{sh}}} C[x := A] : D[x := A]$

4. If $\Gamma(B\lambda_x) \Delta \vdash_{t_{\text{sh}}} C : D$, $\Gamma \vdash_{t_{\text{sh}}} A : B$, $(B\lambda_x)$ bachelor in $\Gamma$, then $\Gamma \Delta[x := A] \vdash_{t_{\text{sh}}} C[x := A] : D[x := A]$

5. If $\Gamma d \vdash_{t_{\text{sh}}} C : D$, then $\Gamma(\Delta[x := A]) \vdash_{t_{\text{sh}}} C[x := A] : D[x := A]$

Proof: 1. Induction to the derivation rules of $=\text{def }$. 2. Induction on the structure of $B$. 3. Induction to the derivation rules, use 1., 2. and the thinning lemma. 4. Idem. 5. Corollary of 3. $\square$
Lemma 5.9 (Generation Lemma for \( \vdash_{\text{sh}} \))

1. If \( \Gamma \vdash_{\text{sh}} x : A \) then for some \( B \): \( (B \lambda x) \in \Gamma, \Gamma \vdash_{\text{sh}} B : S, \Gamma \vdash_{\text{sh}} A = \text{def} B \) and \( \Gamma \vdash_{\text{sh}} A : S' \) for some sort \( S' \).

2. If \( \Gamma \vdash_{\text{sh}} (A \lambda x)B : C \) then for some \( D \) and sort \( S' \): \( \Gamma (A \lambda x)B : D, \Gamma \vdash_{\text{sh}} (A \Pi x)D : S, \Gamma \vdash_{\text{sh}} (A \Pi x)D = \text{def} C \) and if \( (A \Pi x)D \neq C \) then \( \Gamma \vdash_{\text{sh}} C : S' \) for some sort \( S' \).

3. If \( \Gamma \vdash_{\text{sh}} (A \Pi x)B : C \) then for some sorts \( S_1, S_2 \): \( \Gamma \vdash_{\text{sh}} A : S_1, \Gamma \vdash_{\text{sh}} B : S_2, (S_1, S_2) \in \mathcal{R}, \Gamma \vdash_{\text{sh}} C = \text{def} S_2 \) and if \( S_2 \neq C \) then \( \Gamma \vdash_{\text{sh}} C : S \) for some sort \( S \).

4. If \( \Gamma \vdash_{\text{sh}} (A \delta)B : C, (A \delta) \) bachelor in \( B \), then for some terms \( D, E \), variable \( x \): \( \Gamma \vdash_{\text{sh}} A : D, \Gamma \vdash_{\text{sh}} B : (D \Pi x)E, \Gamma \vdash_{\text{sh}} E[x := A] = \text{def} C \) and if \( E[x := A] \neq C \) then \( \Gamma \vdash_{\text{sh}} C : S \) for some sort \( S \).

5. If \( \Gamma \vdash_{\text{sh}} \exists A : B \), then \( \Gamma \exists \vdash_{\text{sh}} A : B \)

Proof: 1., 2., 3. and 4. follow by a tedious but straightforward induction on the derivations (use the thinning lemma). As to 5., use induction on \( \text{weight}(\exists) \).

Corollary 5.10 (Correctness of Types)

If \( \Gamma \vdash_{\text{sh}} A : B \) then \( B \equiv \Box \) or \( \Gamma \vdash_{\text{sh}} B : S \) for some sort \( S \).

Proof: By induction to the derivation rules. The interesting cases are the definition and application rules. In case \( \Gamma \vdash_{\text{sh}} dA : [B]d \) as a consequence of \( \Gamma d \vdash_{\text{sh}} A : B \), then by IH \( B \equiv \Box \) or \( \Gamma d \vdash_{\text{sh}} B : S \) for some sort \( S \). In the first case also \( [B]d \equiv \Box \), in the second case by the Substitution Lemma \( \Gamma \vdash_{\text{sh}} [B]d : [S]d \equiv S \).

In case \( \Gamma \vdash_{\text{sh}} (a \delta)F : B[x := a] \) as a consequence of \( \Gamma \vdash_{\text{sh}} F : (A \Pi x)B, \Gamma \vdash_{\text{sh}} a : A \), then by the induction hypothesis \( \Gamma \vdash_{\text{sh}} (A \Pi x)B : S \) for some sort \( S \) and hence by Generation \( \Gamma (A \lambda x) \vdash_{\text{sh}} B : S \). Then by Thinning \( \Gamma (a \delta)(A \lambda x) \vdash_{\text{sh}} B : S \), so by the definition rule \( \Gamma \vdash_{\text{sh}} (a \delta)(A \lambda x)B : S[x := a] \equiv S \).

Now, we prove SR for \( \vdash_{\text{sh}} \) using \( \rightarrow_{\beta} \) rather than \( \rightarrow_{\beta} \).

Theorem 5.11 (Subject Reduction for \( \vdash_{\text{sh}} \) and \( \rightarrow_{\beta} \))

\( \Gamma \vdash_{\text{sh}} A : B \) and \( A \rightarrow_{\beta} A' \) then \( \Gamma \vdash_{\text{sh}} A' : B \).

Proof: By simultaneous induction on the derivation rules:

1. If \( \Gamma \vdash_{\text{sh}} A : B \) and \( \Gamma \rightarrow_{\beta} \Gamma' \) then \( \Gamma' \vdash_{\text{sh}} A : B \)

2. If \( \Gamma \vdash_{\text{sh}} A : B \) and \( A \rightarrow_{\beta} A' \) then \( \Gamma \vdash_{\text{sh}} A' : B \)

Lemma 5.12 If \( \Gamma \vdash_{\text{sh}} A : B \) and \( A' \in [A] \), \( \Gamma' \) results from \( \Gamma \) by substituting some main items \( (C \omega) \) by \( (C' \omega) \) where \( C' \in [C] \), then \( \Gamma' \vdash_{\text{sh}} A' : B \).

Proof: Induction on the derivation rules. We treat two cases:

- (start): \( \Gamma (A \delta)d(B \in C) \vdash_{\text{sh}} B \) as a consequence of \( \Gamma d \vdash_{\text{sh}} A : B \) and \( \Gamma d \vdash_{\text{sh}} B : S \).

We must show \( \Gamma (A' \delta)d'(B' \in C) \vdash_{\text{sh}} B \). By the induction hypothesis \( \Gamma d' \vdash_{\text{sh}} A' : B \), \( \Gamma d' \vdash_{\text{sh}} B' : S \), by Lemma 3.12 \( B = \beta B' \) so by conversion \( \Gamma d' \vdash_{\text{sh}} A' : B' \), hence by the start rule \( \Gamma (A' \delta)d'(B' \in C) \vdash_{\text{sh}} B' \) and by conversion \( \Gamma (A' \delta)d'(B' \in C) \vdash_{\text{sh}} B \).
• (definition): $\Gamma \vdash^{\text{sh}} dA : [B]_d$ as a consequence of $\Gamma d \vdash^{\text{sh}} A : B$.

By the induction hypothesis $\Gamma' d' \vdash^{\text{sh}} A' : B$, where $d'$ is the items of $d'$ in the order of $d$. Now by Lemma 5.6 $\Gamma' d' \vdash^{\text{sh}} dA' : [B]_{d'}$. By the induction hypothesis also $\Gamma' d' \vdash^{\text{sh}} A' : B$, hence $\Gamma' \vdash^{\text{sh}} dA' : [B]_{d'}$, so by Lemma 5.10 $[B]_{d'} = o \text{ or } \Gamma' \vdash^{\text{sh}} [B]_{d'} : S$ for some sort $S$. In the first case also $[B]_{d'} = o$ and we are done, in the second case by Lemma 3.12 $[B]_{d'} = \gamma [B]_{d'}$ so by conversion $\Gamma' \vdash^{\text{sh}} dA' : [B]_{d'}$.

Corollary 5.13 (TS preserves typing)

1. $\Gamma \vdash^{\text{sh}} A : B \iff \Gamma \vdash^{\text{sh}} TS(A) : B$.
2. If $\Gamma \vdash^{\text{sh}} A : B$ and $A' \in [A]$, $B' \in [B]$ then $\Gamma \vdash^{\text{sh}} A' : B'$.

Proof:

1. By lemma 5.12 as $A \in [TS(A)]$ and $TS(A) \in [A]$. 
2. By lemma 5.12 using the Generation Corollary and conversion.

Here is now the proof of SR using $\vdash^{\text{sh}}$ and $\rightarrow^\beta$. 

Corollary 5.14 (Subject Reduction for $\vdash^{\text{sh}}$ and $\rightarrow^\beta$)

If $\Gamma \vdash^{\text{sh}} A : B$ and $A \rightarrow^\beta A'$ then $\Gamma \vdash^{\text{sh}} A' : B$.

Proof: We prove $\Gamma \vdash^{\text{sh}} A : B$, $A \rightarrow^\beta A' \Longrightarrow \Gamma \vdash^{\text{sh}} A' : B$.

By Corollary 5.13 $\Gamma \vdash^{\text{sh}} TS(A) : B$, by Lemma 4.12 there is a term $C$ such that $TS(A) \rightarrow^\beta C$ and $C \in [A]$, now by Theorem 5.11 $\Gamma \vdash^{\text{sh}} C : B$ and by Lemma 5.13 $\Gamma \vdash^{\text{sh}} A' : B$.

Now we shall prove Strong Normalisation for the $\lambda$-cube with definitions and shuffle $\beta$-reduction. The proof is based on Strong Normalisation of the $\lambda$-cube extended with definitions and generalised reduction as in [BKN 9y].

Lemma 5.15 If $\Gamma \vdash^{\text{sh}} A : B$ then $\Gamma \vdash^e A : B$, where $\vdash^e$ is the typing relation of systems of the $\lambda$-cube extended with definitions and generalised reduction.

Proof: Induction on the derivation rules of $\vdash^{\text{sh}}$. All rules are trivial since they are also rules in $\vdash^e$.

Corollary 5.16 If $A$ is a $\vdash^{\text{sh}}$-legal term then $A$ is strongly normalising with respect to $\rightarrow^\beta$.

Proof: If $A$ is $\vdash^{\text{sh}}$-legal then $A$ is $\vdash^e$-legal by Lemma 5.15 and hence $A$ is strongly normalising with respect to $\rightarrow^\beta$ (see [BKN 9y]).

Definition 5.17 For a $\vdash^{\text{sh}}$-legal term $A$, define $\text{height}(A)$ to be the maximal length of a $\rightarrow^\beta$-reduction path starting with $A$.

Lemma 5.18

1. If $A$ is legal and $A \rightarrow^\beta B$, then $\text{height}(A) > \text{height}(B)$.
2. If $A$ is legal and $A' \in [A]$, then $\text{height}(A') = \text{height}(A)$.
3. If $A$ is legal and $A \rightarrow^\beta B$, then $\text{height}(A) > \text{height}(B)$.

Proof: Long but straightforward.

Corollary 5.19 Every legal term is strongly normalising with respect to $\rightarrow^\beta$.
6 Conclusion

Reshuffling terms and enabling newly visible redexes to be contracted before other ones, and studying the classes of terms that are semantically equivalent, may act as a powerful tool in the study of some programming languages. For example, there is a need to make as many needed redexes visible as possible (see [BKKS 87]). In fact, even though the notion of a needed redex is undecidable, much work has been carried out in order to study some classes of needed redexes (as in [BKKS 87] and [Gardner 94]). Moreover, 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]. Moreover, we may avoid explosion if we had the choice of making more redexes visible and the ability of contracting a visible redex before other ones:

Example 6.1 Let $M = (\lambda z:u.(\lambda y:u.y)(Cxx\ldots x))B(\lambda z:u.u)$ where $B$ is a BIG term. Then $M \rightarrow_\beta (\lambda y:u.y(CBB\ldots B))\lambda z:u.u \rightarrow_\beta (\lambda z:u.u)(CBB\ldots B) \rightarrow_\beta u$ and $u$ is in normal form. Now the first and second reducts both contain the segment $CBB\ldots B$, so they are very, very long terms. Shuffle reduction however allows us to reduce $M$ in the following way: $TS(M) = (\lambda z:u.(\lambda y:u.y)(Cxx\ldots x))\lambda z:u.u)B \rightarrow_\beta (\lambda z:u.u)(Cxx\ldots x))B \rightarrow_\beta (\lambda z:u.u)B \rightarrow_\beta u$, and in this reduction all the terms are of equal or smaller size than $M!$ So shuffle reduction might allow us to define clever strategies that reduce terms via paths of relatively small terms.

References


| Computing Science Reports | Department of Mathematics and Computing Science  
Eindhoven University of Technology |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>In this series appeared:</strong></td>
<td></td>
</tr>
</tbody>
</table>
| 91/02 R.P. Nederpelt  
H.C.M. de Swart | Implication. A survey of the different logical analyses "if...,then...", p. 26. |
| 91/03 J.P. Katoen  
L.A.M. Schoenmakers | Parallel Programs for the Recognition of P-invariant Segments, p. 16. |
| 91/04 E. v.d. Sluis  
| 91/05 D. de Reus | An Implementation Model for GOOD, p. 18. |
| 91/06 K.M. van Hee | SPECIFICATIEMETHODEN, een overzicht, p. 20. |
| 91/07 E.Poll | CPO-models for second order lambda calculus with recursive types and subtyping, p. 49. |
| 91/10 R.C.Backhouse  
P.J. de Bruin  
P. Hoogendijk  
G. Malcolm  
E. Voemans  
| 91/11 R.C. Backhouse  
P.J. de Bruin  
G. Malcolm  
E. Voemans  
J. van der Woude | Relational Catamorphism, p. 31. |
| 91/12 E. van der Sluis | A parallel local search algorithm for the travelling salesman problem, p. 12. |
| 91/14 P. Lemmens | The PDB Hypermedia Package. Why and how it was built, p. 63. |
| 91/15 A.T.M. Aerts  
<p>| 91/16 A.J.J.M. Marcelis | An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25. |</p>
<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>91/18</td>
<td>Rik van Geldrop</td>
<td>Transformational Query Solving, p. 35.</td>
</tr>
<tr>
<td>91/19</td>
<td>Erik Poll</td>
<td>Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.</td>
</tr>
<tr>
<td>91/23</td>
<td>K.M. van Hee, L.J. Somers, M. Voorhoeve</td>
<td>Z and high level Petri nets, p. 16.</td>
</tr>
<tr>
<td>91/24</td>
<td>A.T.M. Aerts, D. de Reus</td>
<td>Formal semantics for BRM with examples, p. 25.</td>
</tr>
<tr>
<td>91/25</td>
<td>P. Zhou, J. Hooman, R. Kuiper</td>
<td>A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and completeness, p. 52.</td>
</tr>
<tr>
<td>91/27</td>
<td>F. de Boer, C. Palamidessi</td>
<td>Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.</td>
</tr>
<tr>
<td>91/28</td>
<td>F. de Boer</td>
<td>A compositional proof system for dynamic process creation, p. 24.</td>
</tr>
<tr>
<td>91/30</td>
<td>J.C.M. Baeten, F.W. Vaandrager</td>
<td>An Algebra for Process Creation, p. 29.</td>
</tr>
<tr>
<td>91/31</td>
<td>H. ten Eikelder</td>
<td>Some algorithms to decide the equivalence of recursive types, p. 26.</td>
</tr>
<tr>
<td>91/33</td>
<td>W. v.d. Aalst</td>
<td>The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.</td>
</tr>
<tr>
<td>91/34</td>
<td>J. Coenen</td>
<td>Specifying fault tolerant programs in deontic logic, p. 15.</td>
</tr>
<tr>
<td>Year</td>
<td>Author(s)</td>
<td>Title</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>91/35</td>
<td>F.S. de Boer, J.W. Klop, C. Palamidessi</td>
<td>Asynchronous communication in process algebra</td>
</tr>
<tr>
<td>92/01</td>
<td>J. Coenen, J. Zwiers, W.-P. de Roever</td>
<td>A note on compositional refinement</td>
</tr>
<tr>
<td>92/02</td>
<td>J. Coenen, J. Hooman</td>
<td>A compositional semantics for fault tolerant real-time systems</td>
</tr>
<tr>
<td>92/03</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Real space process algebra</td>
</tr>
<tr>
<td>92/04</td>
<td>J.P.H.W.v.d.Eijnde</td>
<td>Program derivation in acyclic graphs and related problems</td>
</tr>
<tr>
<td>92/05</td>
<td>J.P.H.W.v.d.Eijnde</td>
<td>Conservative fixpoint functions on a graph</td>
</tr>
<tr>
<td>92/06</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Discrete time process algebra</td>
</tr>
<tr>
<td>92/07</td>
<td>R.P. Nederpelt, F. Kamareddine</td>
<td>The fine-structure of lambda calculus</td>
</tr>
<tr>
<td>92/08</td>
<td>R.P. Nederpelt, F. Kamareddine</td>
<td>On stepwise explicit substitution</td>
</tr>
<tr>
<td>92/09</td>
<td>R.C. Backhouse, W.M.P. van der Aalst</td>
<td>Calculating the Warshall/Floyd path algorithm</td>
</tr>
<tr>
<td>92/10</td>
<td>P.M.P. Rambags, J.S.C.P.v.d.Woude</td>
<td>Composition and decomposition in a CPN model</td>
</tr>
<tr>
<td>92/11</td>
<td>R.C. Backhouse, J.A. Bergstra</td>
<td>Demonic operators and monotype factors</td>
</tr>
<tr>
<td>92/12</td>
<td>F. Kamareddine, S.A. Smolka</td>
<td>Set theory and nominalisation, Part I</td>
</tr>
<tr>
<td>92/13</td>
<td>F. Kamareddine, S.A. Smolka</td>
<td>Set theory and nominalisation, Part II</td>
</tr>
<tr>
<td>92/14</td>
<td>J.C.M. Baeten, J.A. Bergstra, S.A. Smolka</td>
<td>The total order assumption</td>
</tr>
<tr>
<td>92/15</td>
<td>F. Kamareddine, S.A. Smolka</td>
<td>A system at the cross-roads of functional and logic programming</td>
</tr>
<tr>
<td>92/16</td>
<td>R.R. Seljé, W.M.P. van der Aalst</td>
<td>Integrity checking in deductive databases; an exposition</td>
</tr>
<tr>
<td>92/17</td>
<td>W.M.P. van der Aalst</td>
<td>Interval timed coloured Petri nets and their analysis</td>
</tr>
<tr>
<td>92/18</td>
<td>R.P. Nederpelt, F. Kamareddine</td>
<td>A unified approach to Type Theory through a refined lambda-calculus</td>
</tr>
<tr>
<td>92/19</td>
<td>J.C.M. Baeten, J.A. Bergstra, S.A. Smolka</td>
<td>Axiomatizing Probabilistic Processes: ACP with Generative Probabilities</td>
</tr>
<tr>
<td>92/20</td>
<td>F. Kamareddine, S.A. Smolka</td>
<td>Are Types for Natural Language?</td>
</tr>
</tbody>
</table>
Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.

A useful lambda notation, p. 17.

Nominalization, Predication and Type Containment, p. 40.

Bottom-up Abstract Interpretation of Logic Programs, p. 33.

A Programming Logic for Fo, p. 15.

A modelling method using MOVIE and SimCon/ExSpect, p. 15.

A taxonomy of keyword pattern matching algorithms, p. 50.

Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.

A continuous version of the Prisoner's Dilemma, p. 17

Quick sort for linked lists, p. 8.

Deterministic and randomized local search, p. 78.

A congruence theorem for structured operational semantics with predicates, p. 18.

On the unavoidability of metastable behaviour, p. 29

Exercises in Multiprogramming, p. 97

A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.


Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.


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 $\lambda$-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.

<table>
<thead>
<tr>
<th>Paper ID</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>93/33</td>
<td>L. Loyens and J. Moonen</td>
<td>ILIAS, a sequential language for parallel matrix computations, p. 20.</td>
</tr>
<tr>
<td>93/34</td>
<td>J.C.M. Baeten and J.A. Bergstra</td>
<td>Real Time Process Algebra with Infinitesimals, p.39.</td>
</tr>
<tr>
<td>93/36</td>
<td>J.C.M. Baeten and J.A. Bergstra</td>
<td>Non Interleaving Process Algebra, p. 17.</td>
</tr>
<tr>
<td>93/38</td>
<td>C. Verhoef</td>
<td>A general conservative extension theorem in process algebra, p. 17.</td>
</tr>
<tr>
<td>93/41</td>
<td>A. Bijlsma</td>
<td>Temporal operators viewed as predicate transformers, p. 11.</td>
</tr>
<tr>
<td>93/42</td>
<td>P.M.P. Rambags</td>
<td>Automatic Verification of Regular Protocols in P/T Nets, p. 23.</td>
</tr>
<tr>
<td>93/43</td>
<td>B.W. Watson</td>
<td>A taxonomy of finite automata construction algorithms, p. 87.</td>
</tr>
<tr>
<td>93/44</td>
<td>B.W. Watson</td>
<td>A taxonomy of finite automata minimization algorithms, p. 23.</td>
</tr>
<tr>
<td>93/48</td>
<td>R. Gerth</td>
<td>Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.</td>
</tr>
<tr>
<td>Paper ID</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>94/01</td>
<td>P. America, M. van der Kammen, R.P. Nederpelt, O.S. van Roosmalen, H.C.M. de Swan</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 $\Pi$-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>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>94/18</td>
<td>F. Kamareddine, R. Nederpelt</td>
<td>Refining Reduction in the Lambda Calculus, p. 15.</td>
</tr>
<tr>
<td>94/19</td>
<td>B.W. Watson</td>
<td>The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.</td>
</tr>
<tr>
<td>94/20</td>
<td>R. Bloo, F. Kamareddine, R. Nederpelt</td>
<td>Beyond ( \beta )-Reduction in Church's ( \lambda \rightarrow ), p. 22.</td>
</tr>
<tr>
<td>94/22</td>
<td>B.W. Watson</td>
<td>The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.</td>
</tr>
<tr>
<td>94/23</td>
<td>S. Mauw, M.A. Reniers</td>
<td>An algebraic semantics of Message Sequence Charts, p. 43.</td>
</tr>
<tr>
<td>94/24</td>
<td>D. Dams, O. Grumbel, R. Geth</td>
<td>Abstract Interpretation of Reactive Systems: Abstractions Preserving ( \forall \text{CTL*}, \exists \text{CTL*} ) and ( \text{CTL*} ), p. 28.</td>
</tr>
<tr>
<td>94/25</td>
<td>T. Kloks</td>
<td>( K_{13} )-free and ( W_4 )-free graphs, p. 10.</td>
</tr>
<tr>
<td>94/29</td>
<td>J. Hooman</td>
<td>Correctness of Real Time Systems by Construction, p. 22.</td>
</tr>
<tr>
<td>94/30</td>
<td>J.C.M. Baeten, J.A. Bergstra, Gh. Ţăfănescu</td>
<td>Process Algebra with Feedback, p. 22.</td>
</tr>
<tr>
<td>94/31</td>
<td>B.W. Watson, R.E. Watson</td>
<td>A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.</td>
</tr>
<tr>
<td>94/33</td>
<td>T. Laan</td>
<td>A formalization of the Ramified Type Theory, p. 40.</td>
</tr>
<tr>
<td>94/34</td>
<td>R. Bloo, F. Kamareddine, R. Nederpelt</td>
<td>The Barendregt Cube with Definitions and Generalised Reduction, p. 37.</td>
</tr>
<tr>
<td>94/35</td>
<td>J.C.M. Baeten, S. Mauw</td>
<td>Delayed choice: an operator for joining Message Sequence Charts, p. 15.</td>
</tr>
<tr>
<td>94/36</td>
<td>F. Kamareddine, R. Nederpelt</td>
<td>Canonical typing and ( \Pi )-conversion in the Barendregt Cube, p. 19.</td>
</tr>
</tbody>
</table>
94/37 T. Basten
   R. Bol
   M. Voorhoeve
   Simulating and Analyzing Railway Interlockings in ExSpec, p. 30.

94/38 A. Bijlsma
   C.S. Scholten
   Point-free substitution, p. 10.

94/39 A. Blokhuis
   T. Kloks
   On the equivalence covering number of splitgraphs, p. 4.

94/40 D. Alstein
   Distributed Consensus and Hard Real-Time Systems, p. 34.

94/41 T. Kloks
   D. Kratsch
   Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.

94/42 J. Engelfriet
   Concatenation of Graphs, p. 7.

94/43 R.C. Backhouse
   Bijsterveld
   Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.

94/44 E. Brinksma
   J. Davies
   R. Gerth
   S. Graf
   W. Janssen
   B. Jonsson
   S. Katz
   G. Lowe
   M. Poel
   A. Pnucli
   C. Rump
   J. Zwiers
   Verifying Sequentially Consistent Memory, p. 160

94/45 G.J. Houben
   Tutorial voor de ExSpec-bibliotheek voor "Administratieve Logistiek", p. 43.

94/46 R. Bloo
   F. Kamareddine
   R. Nederpelt
   The λ-cube with classes of terms modulo conversion, p. 16.