The λ-cube with classes which approximate reductional equivalence*

Roel Bloo† Fairouz Kamareddine‡ Rob Nederpelt§

December 5, 1995

Abstract

We study lambda calculus and refine the notions of β-reduction and β-equality. In particular, we define the operation TS (term reshuffling) on λ-terms which reshuffles a term in such a way that more redexes become visible. Two terms are called shuffle-equivalent if they have syntactically equivalent TS-images. The shuffle-equivalence classes are shown to divide the classes of β-equal terms into smaller classes consisting of terms with similar reduction behaviour. The refinement of β-reduction from a relation on terms to a relation on shuffle classes, called shuffle-reduction, allows one to make more redexes visible and to contract these newly visible redexes. This enables one to have more freedom in choosing the reduction path of a term, which can result in smaller terms along the reduction path if a clever reduction strategy is used. Moreover, this gain in reductional breadth is not at the expense of reductional length.

We show that the λ-cube of [Barendregt 92] extended with shuffle-reduction satisfies all its properties such as Church Rosser, Strong Normalisation and Subject Reduction (the latter depends on allowing definitions in contexts).

1 Introduction

1.1 Term reshuffling and reductional equivalence

β-equality of two terms A and B is by the Church-Rosser property equivalent to the existence of a common reduct C. Nothing can be said about the nature of the two reductions $A \twoheadrightarrow_\beta C$ and $B \twoheadrightarrow_\beta C$. It can be that both reductions consist of the same number of steps, or that one of them is larger than the other. Also, the reduction behaviour of A and B can be very different, as is the case if $A \equiv \mathbf{KI\Omega}$ and $B \equiv \mathbf{KII}$. We think it is an interesting problem to characterize terms with equal reduction behaviour. In this paper we try to give an approximation to the reductional equivalence between two terms.

*We are grateful for the Netherlands Computer Science Research Foundation (SION), the Netherlands Organisation for Scientific Research (NWO), the universities of Glasgow and Eindhoven, to the Basic Action for Research ESPRIT project "Types for Proofs and Programs", and to the EPSRC Grant nb GR/K 25014 for their financial support.

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

‡Department of Computing Science, 17 Lilybank Gardens, University of Glasgow, Glasgow G12 8QQ, Scotland, *email*: fairouz@dcs.glasgow.ac.uk

§same address as Bloo. `email`: wsinrpn@win.tue.nl

Consider in a typed λ-calculus of the λ-cube as described in [Barendregt 92], the terms

$$\begin{aligned} A &\equiv (\lambda_{\beta:*}.\lambda_{y:\beta}.\lambda_{f:\beta\to\beta}.fy)\alpha x \\ B &\equiv (\lambda_{\beta:*}.(\lambda_{y:\beta}.\lambda_{f:\beta\to\beta}.fy)x)\alpha \end{aligned}$$

Both terms have the term $\lambda_{f:\alpha\to\alpha}.fx$ as a reduct, so $A =_\beta B$. However, B has two redexes whereas A has only one. Here are the redexes of B and their corresponding results in B:

1. $r_1 = (\lambda_{\beta:*}.(\lambda_{y:\beta}.\lambda_{f:\beta\to\beta}.fy)x)\alpha$ which when contracted in B results in $(\lambda_{y:\alpha}.\lambda_{f:\alpha\to\alpha}.fy)x$
2. $r_2 = (\lambda_{y:\beta}.\lambda_{f:\beta\to\beta}.fy)x$ which when contracted in B results in $(\lambda_{\beta:*}.\lambda_{f:\beta\to\beta}.fx)\alpha$

Here is the only redex in A and the result of contracting this redex in A:

1. $r'_1 = (\lambda_{\beta:*}.\lambda_{y:\beta}.\lambda_{f:\beta\to\beta}.fy)\alpha$ which when contracted in A results in $(\lambda_{y:\alpha}.\lambda_{f:\alpha\to\alpha}.fy)x$

Note that r_1 in B and r'_1 in A are both based on $(\lambda_{\beta:*}.-)\alpha$ and contracting r_1 in B results in the same term as contracting r'_1 in A.

A closer look at A however, enables us to see that in A (as in B), $\lambda_{y:-}$ will get matched with x resulting in a redex $r'_2 = (\lambda_{y:-}.-)x$. There are differences however between r_2 in B and r'_2 in A. r_2 in B is completely visible and may be contracted before r_1 in B. r'_2 on the other hand is a future redex in A. In fact, it is not a redex of A itself but a redex of a contractum of A, namely $(\lambda_{y:\alpha}.\lambda_{f:\alpha\to\alpha}.fy)x$, the result of contracting the redex r'_1 in A.

We could however guess from A itself the presence of the future redex. That is, looking at A itself, we see that λ_β is matched with α and λ_y is matched with x.

In this paper, and in order to discuss reductional equivalence between terms, redexes will be extended so that a future redex like $(\lambda_{y:-}.-)x$ will be treated as a first class redex and will be contracted in A even before the originator $(\lambda_{\beta:*}.\lambda_{y:\beta}.\lambda_{f:\beta\to\beta}.fy)\alpha$ has been contracted. Hence, with our extended notion of redexes and reduction we get in A:

$$r'_2 = (\lambda_{y:\beta}.\lambda_{f:\beta\to\beta}.fy)x \text{ which when contracted in } A \text{ results in } (\lambda_{\beta:*}.\lambda_{f:\beta\to\beta}.fx)\alpha$$

This is remarkable. Note that r'_2 is $\lambda_{y:\beta}$ matched with x (exactly as r_2 in B). Note moreover that contracting r'_2 in A gives the same result as contracting r_2 in B.

With this notion of extended redex, we can observe that there is a bijective correspondence between the (extended) redexes of A and B. That is, r_1 corresponds to r'_1 and r_2 corresponds to r'_2. Moreover, if one redex is contracted in A, the reduct is syntactically equal to the reduct which results from contracting the corresponding redex in B and vice versa. That is, r_1 and r'_1 yield the same values; similarly r_2 and r'_2 yield the same values.

If on the other hand, we consider the term $C \equiv (\lambda_{y:\beta}.(\lambda_{\beta:*}.\lambda_{f:\beta\to\beta}.fy)\alpha)x$, we see that there is no such correspondence between (extended) redexes of A and C: contracting the outermost redex in C gives the same result as contracting the mentioned extended redex in A, but contracting the innermost redex in C yields $(\lambda_{y:\beta}.\lambda_{f:\alpha\to\alpha}.fy)x$ which is not syntactically equivalent to $(\lambda_{y:\alpha}.\lambda_{f:\alpha\to\alpha}.fy)x$.

These considerations lead us to define reductional equivalence $\sim_{\text{inf}}$ informally by:

Definition 1.1 *We say that A and B are reductionally equivalent and write $A \sim_{\text{inf}} B$ iff*

1. *There is a bijective correspondence between the (extended) redexes of A and B.*

2. *Contracting an (extended) redex in A results in a value syntactically equal ($\equiv$) or reductionally equivalent ($\sim_{\inf}$) to the result of contracting the corresponding redex in B and vice versa.*

Alas however, it may not be easy to decide on the reductional equivalence of two terms. We conjecture that in general it is undecidable whether two terms are reductionally equivalent.

Conjecture 1.2 *It is undecidable whether two terms are reductionally equivalent.*

It seems interesting to find notions that are more easy to decide which approximate $\sim_{\inf}$. One approach could be to define *degrees of reductional equivalence* ($\sim_n$ with $n \geq 0$ for short) in the following way:

- $M \sim_0 N$ iff $M \equiv N$.
- $M \sim_{n+1} N$ iff there is a bijective correspondence between the (extended) redexes of M and N such that contracting one in M yields a term $\sim_m$, $m \leq n$ to the result of contracting the corresponding redex in N.

These are not very well behaved notions since the notions $\sim_n$ for $n \geq 2$ are not compatible. This can be seen as follows[1]:
$\lambda_{z:g}.(\lambda_{x:c}.\lambda_{y:d}.e)ba \sim_1 \lambda_{z:g}.(\lambda_{x:c}.(\lambda_{y:d}.e)a)b$ but
$(\lambda_{z:g}.(\lambda_{x:c}.\lambda_{y:d}.e)ba)f \sim_2 (\lambda_{z:g}.(\lambda_{x:c}.(\lambda_{y:d}.e)a)b)f$.

For this reason we follow a different approach and consider what we call *shuffle-equivalence*. Our notion of shuffle-equivalence will be decidable but it is incomparable to reductional equivalence of any degree $\sim_n$, $n \geq 0$. It is however a good approximation to $\sim_{\inf}$.

Term reshuffling (see [KN 95]) amounts to rewriting a term so that more redexes than usual become visible. Observe that extended redexes can be shuffled to "classical" (i.e., non extended) redexes without loosing reductional equivalence. This can be seen by our terms A, B, and C above. The extended redex r_2' in A becomes classical in B. We call B the reshuffled version of A. We have seen that $A \sim_{\inf} B$. C on the other hand is not equivalent to either A or B, either shuffle-wise or reductional wise.

Now to decide on the shuffle-equivalence of two terms A and B, we reshuffle both A and B and if we get in both cases the same result, then we say that A and B are shuffle-equivalent. We denote the reshuffled version of a term A by $TS(A)$; a concise definition of TS will be given in subsection 3.2. Now, we go back to the terms given above and see what we mean:

Example 1.3 Consider the terms A, B and C given above. We see that all redexes in B are visible and so $TS(B) \equiv B$. On the other hand, we see that A has a redex $(\lambda_{y:\beta}.-)x$ which becomes visible after $(\lambda_{\beta:*}.-)\alpha$ has been contracted. The idea of term reshuffling is to make this redex visible even before the other redex has been contracted. When we do so (via the function TS of Definition 3.5), we get $TS(A) \equiv B$. Now, $TS(A) \equiv TS(B)$. Moreover, we have seen that $A \sim_{\inf} B$. C on the other hand, has all its redexes visible, hence $TS(C) \equiv C$, and we see that $TS(C) \not\equiv TS(A)$ or $TS(B)$.

It will be easier to understand what the operation TS does if we change the classical notation we have been using so far. For now, let us provide another example where we have more extended redexes than above.

[1] This counterexample will be better understood if it is translated into the item-notation of Section 2.

Example 1.4 Let $A \equiv ((\lambda_{f:\Pi_{u:\alpha}.\Pi_{v:\alpha}.\alpha}.(\lambda_{x:\alpha}.\lambda_{y:\alpha}.fxy)m)+)n$.

The redexes in A are: $(\lambda_{f:\Pi_{u:\alpha}.\Pi_{v:\alpha}.\alpha}.(\lambda_{x:\alpha}.\lambda_{y:\alpha}.fxy)m)+$, $(\lambda_{x:\alpha}.\lambda_{y:\alpha}.fxy)m$ and $(\lambda_{y:\alpha}.fxy)n$. The third redex is not classical, is not immediately visible, and is not subject to contraction without having first unfolded in $\lambda_{y:\alpha}.fxy$ the two definitions that f is $+$ and x is m.

Now, take $B \equiv (\lambda_{f:\Pi_{u:\alpha}.\Pi_{v:\alpha}.\alpha}.(\lambda_{x:\alpha}.(\lambda_{y:\alpha}.fxy)n)m)+$.

The (classical) redexes in B are: $(\lambda_{f:\Pi_{u:\alpha}.\Pi_{v:\alpha}.\alpha}.(\lambda_{x:\alpha}.(\lambda_{y:\alpha}.fxy)n)m)+$, $(\lambda_{x:\alpha}.(\lambda_{y:\alpha}.fxy)n)m$ and $(\lambda_{y:\alpha}.fxy)n$. All these redexes are classical, immediately visible and subject to contraction. Moreover, $A =_\beta B$.

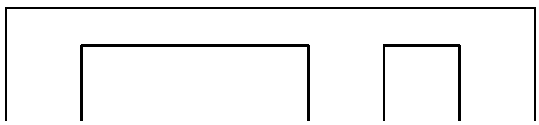
All the three redexes of A are *needed* in order to get its normal form and correspond to the redexes of B. B is already in *reshuffled form*, $B \equiv TS(B)$. Moreover, reshuffling A so that all redexes become visible results in B. I.e. $TS(A) \equiv B$. Hence as $TS(A) \equiv TS(B)$, $A \sim_{\text{inf}} B$.

Now, consider A above and $C \equiv (((\lambda_{f:\Pi_{u:\alpha}.\Pi_{v:\alpha}.\alpha}.\lambda_{x:\alpha}.\lambda_{y:\alpha}.fxy)+)m)n$. Both A and C have a bijective correspondence between their extended redexes and $A =_\beta C$. $A \sim_{\text{inf}} C$ but this is hardly visible. Reshuffling A and C however, makes this claim visible. That is, $TS(A) \equiv TS(C) \equiv TS(B)$ and so all three terms A, B and C are reductionally equivalent.

The classical notation cannot extend the notion of redexes or enable reshuffling in an easy way. *Item notation* however (see [KN 93], [KN 94] and [KN 96b]), can. In item notation, complex terms of the λ-cube are of the form $(A\omega)B$ where $\omega \in \{\delta\} \cup \{\mathcal{O}_x; x$ is a variable, a $*$ or a $\Box$ and $\mathcal{O} = \lambda$ 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 δ-item next to a λ-item ([KN 96b] discusses various advantages of this notation).

Example 1.5 A of Example 1.4 reads $(n\delta)(+\delta)((\alpha\Pi_u)(\alpha\Pi_v)\alpha\lambda_f)(m\delta)(\alpha\lambda_x)(\alpha\lambda_y)(y\delta)(x\delta)f$ in item notation (see Section 2). Here, the first two redexes, the classical redexes, correspond to $\delta\lambda$-*pairs* followed by the body of the abstraction as follows: $(\lambda_{f:\Pi_{u:\alpha}.\Pi_{v:\alpha}.\alpha}.(\lambda_{x:\alpha}.\lambda_{y:\alpha}.fxy)m)+$ corresponds to $(+\delta)((\alpha\Pi_u)(\alpha\Pi_v)\alpha\lambda_f)(m\delta)(\alpha\lambda_x)(\alpha\lambda_y)(y\delta)(x\delta)f$ and $(\lambda_{x:\alpha}.\lambda_{y:\alpha}.fxy)m$ corresponds to $(m\delta)(\alpha\lambda_x)(\alpha\lambda_y)(y\delta)(x\delta)f$. Note that the so-called δ-item $(+\delta)$ and the so-called λ-item $((\alpha\Pi_u)(\alpha\Pi_v)\alpha\lambda_f)$ are adjacent, showing the presence of a redex. Similarly, note the adjacency of $(m\delta)$ and $(\alpha\lambda_x)$.

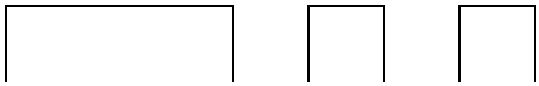
The third redex of A is obtained by matching δ and λ-items. $(\lambda_{y:\alpha}.fxy)n$ is visible as it corresponds to the matching $(n\delta)(\alpha\lambda_y)$ where $(n\delta)$ and $(\alpha\lambda_y)$ are separated by the segment $(+\delta)((\alpha\Pi_u)(\alpha\Pi_v)\alpha\lambda_f)(m\delta)(\alpha\lambda_x)$ which has the bracketing structure $[\,][\,]$ (see Figure 1 which represents A).



$(n\delta)\ \ (+\delta)\ ((\alpha\Pi_u)(\alpha\Pi_v)\alpha\lambda_f)\ (m\delta)\ (\alpha\lambda_x)(\alpha\lambda_y)\ \ (y\delta)(x\delta)f$

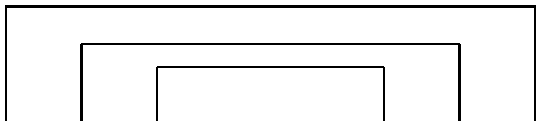
Figure 1: (Extended) redexes in item notation: A

Term reshuffling amounts to moving δ-items to occur next to their matching λ-items. Hence A of Example 1.4 is reshuffled to $(+\delta)((\alpha\Pi_u)(\alpha\Pi_v)\alpha\lambda_f)(m\delta)(\alpha\lambda_x)(n\delta)(\alpha\lambda_y)(y\delta)(x\delta)f$ and Figure 1 changes to Figure 2 (which represents B). Furthermore, Figure 3 (which represents C) also changes to Figure 2 when reshuffled.



$(+\delta)\ ((\alpha\Pi_u)(\alpha\Pi_v)\alpha\lambda_f)\ (m\delta)\ (\alpha\lambda_x)\ (n\delta)\ (\alpha\lambda_y)\ (y\delta)(x\delta)f$

Figure 2: The reshuffled term A in item notation: B



$(n\delta)\ (m\delta)\ (+\delta)\ ((\alpha\Pi_u)(\alpha\Pi_v)\alpha\lambda_f)(\alpha\lambda_x)(\alpha\lambda_y)\ (y\delta)(x\delta)f$

Figure 3: More extended redexes: C

1.2 Reduction modulo term classes

Having noted above that terms like A, B and C of Example 1.4 are shuffle-equivalent and that for both A and C, we could find their TS form which makes more redexes visible, we decide to study the classes of these shuffle-equivalent terms. For this reason, we define $[t]$, the class of t, to be $\{t' | TS(t) \equiv TS(t')\}$. Hence, A, B and C above belong to the same class.

All elements of $[A]$ are $=_\beta$ and have somehow the same redexes. We say $A \rightsquigarrow_\beta A'$ iff $\exists B \in [A] \exists B' \in [A']$ such that $B \to_\beta B'$. Note that from this definition, $\to_\beta \subseteq \rightsquigarrow_\beta \subseteq =_\beta$.

1.3 The need for definitions

Recall that in Example 1.4 when we explained the third redex of A, we said that two definitions were unfolded in $\lambda_{y:\alpha}.fxy$. It turns out that this observation is necessary in order to show that the λ-cube extended with term reshuffling and $\rightsquigarrow_\beta$ satisfies Subject Reduction. But then definitions are important on their own (see [BKN 9y], [Con 86], [Dow 91], [KBN 9-], [NGV 94] and [SP 93]). We show that the λ-cube extended with TS, $\rightsquigarrow_\beta$ 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

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

$$\begin{array}{lcll} \mathcal{I}(A) & = & A & \text{if } A \text{ is a variable or a constant} \\ \mathcal{I}(\mathcal{O}_{x:A}.B) & = & (\mathcal{I}(A)\mathcal{O}_x)\mathcal{I}(B) & \text{where } \mathcal{O} \in \{\lambda, \Pi\} \\ \mathcal{I}(AB) & = & (\mathcal{I}(B)\delta)\mathcal{I}(A) & \end{array}$$

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 λ's have different variables as subscript. Hence, we will not have $(x\delta)(A\lambda_x)(x\lambda_y)(y\lambda_y)y$, but $(x\delta)(A\lambda_u)(u\lambda_y)(y\lambda_v)v$ instead.

The systems of the λ-cube are based on a set of *pseudo-expressions* $\mathcal{T}$ (also called terms) defined by:

$$\mathcal{T} = * \mid \Box \mid V \mid (\mathcal{T}\delta)\mathcal{T} \mid (\mathcal{T}\mathcal{O}_V)\mathcal{T}$$

where $\mathcal{O}$ ranges over $\{\lambda, \Pi\}$ and V is an infinite collection of variables over which $x, y, z, \ldots$ range. $*$ 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.

A relation $\to$ on terms is compatible iff the following holds for all $\omega \in \{\delta\} \cup \{\mathcal{O}_x \mid x \in V\}$:

$$\frac{A_1 \to A_2}{(A_1\omega)B \to (A_2\omega)B} \qquad \frac{B_1 \to B_2}{(A\omega)B_1 \to (A\omega)B_2}$$

Definition 2.2 *((main) items, (main, $\delta\mathcal{O}$-)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 λ-item, Π-item and δ-item respectively); A is called the body of the item. We use $s, s_1, s_i, \ldots$ to range over items.*
- *Each pseudo-expression A is the concatenation of zero or more items and a variable or constant: $A \equiv s_1 s_2 \cdots s_n B$ where $B \in V \cup \{*, \Box\}$. These items $s_1, s_2, \ldots, s_n$ are called the* **main items** *of A, B is called the* **heart** *of A, notation $\heartsuit(A)$.*
- *A concatenation of zero or more items $s_1 s_2 \cdots s_n$ is called a* **segment**. *We use $\overline{s}, \overline{s}_1, \overline{s}_i, \ldots$ as meta-variables for segments. We write $\emptyset$ for the empty segment. The items $s_1, s_2, \ldots, s_n$ (if any) are called the* **main items** *of the segment. A concatenation of adjacent main items $s_m \cdots s_{m+k}$, is called a* **main segment**. *A* **$\delta\mathcal{O}$-segment** *is a δ-item immediately followed by an $\mathcal{O}$-item.*
- *The* **weight** *of a segment $\overline{s}$, $\mathtt{weight}(\overline{s})$, is the number of main items that compose the segment. Moreover, we define $\mathtt{weight}(\overline{s}t) = \mathtt{weight}(\overline{s})$ for $t \in V \cup \{*, \Box\}$.*

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 $\emptyset$ is a well-balanced segment.*
- *If $\overline{s}$ is well-balanced, then $(A\delta)\overline{s}(B\mathcal{O}_x)$ 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 $\mathcal{O}$-)item corresponding with an opening (resp. closing) bracket.

Definition 2.4 *(match, $\delta\mathcal{O}$- (reducible) couple, partner, partnered, bachelor)*
Let $A \in \mathcal{T}$. Let $\overline{s} \equiv s_1 \cdots s_n$ be a segment occurring in A.

- *We say that s_i and s_j* **match**, *when $1 \leq i < j \leq n$, s_i is a δ-item, s_j is an $\mathcal{O}$-item, and $s_{i+1} \cdots s_{j-1}$ is a well-balanced segment.*
- *If s_i and s_j match, we call $s_i s_j$ a* **$\delta\mathcal{O}$-couple**. *A $\delta\lambda$-couple is called a* **reducible couple**.
- *If s_i and s_j match, we call s_i and s_j* **partners** *or* **partnered** *items.*
- *All non-partnered $\mathcal{O}$- (or δ-)items s_k in A are called* **bachelor** *$\mathcal{O}$- (resp. δ-)items in A; a segment consisting of bachelor items only is called a* **bachelor segment**.

Definition 2.5 *(definitions, unfolding)*

- *If $\overline{s}$ is well-balanced not containing $\delta\Pi$-couples, then a segment $(A\delta)\overline{s}(B\lambda_x)$ occurring in a context is called a definition.*
- *Let $\overline{s}$ be a well-balanced segment, We define the unfolding of $\overline{s}$ in A, $[A]_{\overline{s}}$, inductively as follows: $[A]_\emptyset \equiv A$, $[A]_{(B\delta)\overline{s_1}(C\mathcal{O}_x)} \equiv [A[x := B]]_{\overline{s_1}}$ and $[A]_{\overline{s_1}\,\overline{s_2}} \equiv [[A]_{\overline{s_2}}]_{\overline{s_1}}$. Note that substitution takes place from right to left and that when none of the binding variables of $\overline{s}$ are free in A, then $[A]_{\overline{s}} \equiv A$.*

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 $\vdash$ be a typing relation and let $\twoheadrightarrow$ be a reduction relation whose equivalence closure is $=_\beta$.

Definition 2.6 *(declarations, pseudocontexts, $\subseteq'$, $\twoheadrightarrow$)*

1. *A declaration d is a λ-item $(A\lambda_x)$. $\mathtt{subj}(d)$, $\mathtt{pred}(d)$ and $\underline{d}$ are x, A and $\emptyset$ respectively.*
2. *For a definition $d \equiv (B\delta)\overline{s}(A\lambda_x)$ we define $\mathtt{subj}(d)$, $\mathtt{pred}(d)$, $\underline{d}$ and $\mathtt{def}(d)$ to be x, A, $\overline{s}$ and B respectively.*
3. *We use $d, d_1, d_2, \ldots$ 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 $x \not\equiv y$. We use $\Gamma, \Delta, \Gamma', \Gamma_1, \Gamma_2, \ldots$ to range over pseudocontexts.*
5. *For Γ a pseudocontext we define*

   $dom(\Gamma) = \{x \in V \mid (A\lambda_x)$ *is a main λ-item in Γ for some $A\}$,*

   $\Gamma\text{-}\mathtt{decl} = \{s \mid s$ *is a bachelor main λ-item of $\Gamma\}$,*

   $\Gamma\text{-}\mathtt{def} = \{\overline{s} \mid \overline{s} \equiv (A\delta)\overline{s}_1(B\lambda_x)$ *is a main segment of Γ where $\overline{s}_1$ is well-balanced $\}$,*

   Note that $dom(\Gamma) = \{\mathtt{subj}(d) \mid d \in \Gamma\text{-}\mathtt{decl} \cup \Gamma\text{-}\mathtt{def}\}$.
6. *Define $\subseteq'$ between pseudocontexts as the least reflexive transitive relation satisfying:*
   - *$\Gamma\Delta \subseteq' \Gamma(C\lambda_x)\Delta$ if no λ-item in Δ matches a δ-item in Γ*
   - *$\Gamma\underline{d}\Delta \subseteq' \Gamma d\Delta$ if d is a definition*

- $\Gamma\overline{s}(A\lambda_x)\Delta \subsetneq' \Gamma(D\delta)\overline{s}(A\lambda_x)\Delta$ *if* $(A\lambda_x)$ *is bachelor in* $\Gamma\overline{s}(A\lambda_x)\Delta$, $\overline{s}$ *is well-balanced*

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

Definition 2.7 *(Definitional equality) For all contexts* Γ *we define the binary relation* $\Gamma \vdash \cdot =_{\mathtt{def}} \cdot$ *to be the equivalence relation generated by*

- *if* $A =_\beta B$ *then* $\Gamma \vdash A =_{\mathtt{def}} B$
- *if* $d \in \Gamma\text{-}\mathtt{def}$ *and* $A, B \in \mathcal{T}$ *such that* B *arises from* A *by substituting one particular occurrence of* $\mathtt{subj}(d)$ *in* A *by* $\mathtt{def}(d)$, *then* $\Gamma \vdash A =_{\mathtt{def}} B$.

Definition 2.8 *(statements, judgements,* $\prec$*)*

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* Γ *is a pseudocontext and* $A : B$ *is a statement, we call* $\Gamma \vdash A : B$ *a judgement, meaning* $A : B$ *is derivable from the context* Γ, *and we write* $\Gamma \vdash A : B : C$ *to mean* $\Gamma \vdash A : B \wedge \Gamma \vdash B : C$.
3. *For* Γ *a pseudocontext and* $d \in \Gamma\text{-}\mathtt{def} \cup \Gamma\text{-}\mathtt{decl}$, Γ *invites* d, *notation* $\Gamma \prec d$, *iff*
   - Γd *is a pseudocontext*
   - $\Gamma\underline{d} \vdash \mathtt{pred}(d) : S$ *for some sort* S.
   - *if* d *is a definition then* $\Gamma\underline{d} \vdash \mathtt{def}(d) : \mathtt{pred}(d)$ *and* $FV(\mathtt{def}(d)) \subseteq dom(\Gamma)$

Remark 2.9 Note that binding variables in $\underline{d}$ may occur free in $\mathtt{pred}(d)$ but not in $\mathtt{def}(d)$ if $\Gamma \prec d$.

Definition 2.10 *Let* Γ *be a pseudocontext and* A *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* $\Gamma \vdash \mathtt{subj}(d) : \mathtt{pred}(d)$.
   - *If* d *is a definition:* $\Gamma \vdash d$ *iff* $\Gamma \vdash \mathtt{subj}(d) : \mathtt{pred}(d) \wedge \Gamma \vdash \mathtt{def}(d) : \mathtt{pred}(d) \wedge \Gamma \vdash \underline{d} \wedge \Gamma \vdash \mathtt{subj}(d) =_{\mathtt{def}} \mathtt{def}(d)$.
   - $\Gamma \vdash d_1 \cdots d_n$ *iff* $\Gamma \vdash d_i$ *for all* $1 \leq i \leq n$.
2. Γ *is called legal if* $\exists P, Q \in \mathcal{T}$ *such that* $\Gamma \vdash P : Q$.
3. $A \in \mathcal{T}$ *is called a* $\Gamma^\vdash$*-term if* $\exists B \in \mathcal{T}[\Gamma \vdash A : B$ *or* $\Gamma \vdash B : A]$.
   We take $\Gamma^\vdash\text{-terms} = \{A \in \mathcal{T} \mid \exists B \in \mathcal{T}[\Gamma \vdash A : B \vee \Gamma \vdash B : A]\}$.
   $A \in \mathcal{T}$ *is called legal if* $\exists \Gamma[A \in \Gamma^\vdash\text{-terms}]$.
4. *We say that* A *is strongly normalising with respect to a reduction relation* $\to$ *(written* $SN_\to(A)$*) iff every* $\to$*-reduction path starting at* A *terminates.*

Definition 2.11 *We say that two terms* A *and* B *are semantically equivalent iff* $A =_\beta B$.

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

Definition 2.12 *(Axioms and rules of the λ-cube: d is a declaration, $=_{\mathtt{def}}$ is $=_\beta$)*

(axiom) $\quad <> \vdash * : \Box$

(start rule) $\quad \dfrac{\Gamma \prec d}{\Gamma d \vdash \mathtt{subj}(d) : \mathtt{pred}(d)}$

(weakening rule) $\quad \dfrac{\Gamma \prec d \qquad \Gamma\underline{d} \vdash D : E}{\Gamma d \vdash D : E}$

(application rule) $\quad \dfrac{\Gamma \vdash F : (A\Pi_x)B \qquad \Gamma \vdash a : A}{\Gamma \vdash (a\delta)F : B[x := a]}$

(abstraction rule) $\quad \dfrac{\Gamma(A\lambda_x) \vdash b : B \qquad \Gamma \vdash (A\Pi_x)B : S}{\Gamma \vdash (A\lambda_x)b : (A\Pi_x)B}$

(conversion rule) $\quad \dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : S \qquad \Gamma \vdash B =_{\mathtt{def}} B'}{\Gamma \vdash A : B'}$

(formation rule) $\quad \dfrac{\Gamma \vdash A : S_1 \qquad \Gamma(A\lambda_x) \vdash B : S_2}{\Gamma \vdash (A\Pi_x)B : S_2}$ *if (S_1, S_2) is a rule*

Each of the eight systems of the λ-cube is obtained by taking the (S_1, S_2) rules allowed from a subset of $\{(*,*), (*,\Box), (\Box,*), (\Box,\Box)\}$. 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 $(*,\Box)$, $(\Box,*)$ and $(\Box,\Box)$ for (S_1, S_2). Here is the table which presents the eight systems of the λ-cube:

System	*Set of specific rules*			
$\lambda_\rightarrow$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
λP	$(*,*)$		$(*,\Box)$	
$\lambda P2$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
$\lambda P\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda P\omega = \lambda C$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

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 function is correct in the sense that $A =_\beta TS(A)$, i.e., A and $TS(A)$ are semantically equivalent. In Section 4, we show that this term reshuffling preserves reduction in the sense that if $A \rightarrow_\beta B$ then $TS(A) \rightsquigarrow_\beta TS(B)$ and

$\exists B' \in [B]\ [TS(A) \to_\beta B']$. In Section 5, we show that this term reshuffling preserves typing in the sense that if $\Gamma \vdash^{\mathtt{sh}} A : B$ then $\Gamma \vdash^{\mathtt{sh}} TS(A) : B$.

Let us go back to the definition of $\delta\mathcal{O}$-couples. Recall that if $\overline{s} \equiv s_1 \cdots s_m$ for $m > 1$ where $s_1 s_m$ is a $\delta\mathcal{O}$-couple then $s_2 \cdots s_{m-1}$ is a well-balanced segment, s_1 is the δ-item of the $\delta\mathcal{O}$-couple and s_m is its $\mathcal{O}$-item. Now, we can move s_1 in $\overline{s}$ so that it occurs adjacently to s_m. That is, we may rewrite $\overline{s}$ as $s_2 \cdots s_{m-1} s_1 s_m$. As legal terms and contexts of the λ-cube contain no $\delta\Pi$-couples, we focus only on $\delta\lambda$-couples.

Example 3.1 The term $A \equiv (u\delta)(w\delta)(P\lambda_x)(v\delta)(Q\lambda_y)(R\lambda_z)(z\delta)(y\delta)x$ reshuffles to $TS(A) \equiv (w\delta)(P\lambda_x)(v\delta)(Q\lambda_y)(u\delta)(R\lambda_z)(z\delta)(y\delta)x$ by moving the item $(u\delta)$ to the right. Such a reshuffling is not easy to describe in the classical notation. That is, it is difficult to describe how $((\lambda_{x:P}.(\lambda_{y:Q}.\lambda_{z:R}.xyz)v)w)u$ 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 $\overline{s}$ be a well-balanced segment not containing $\delta\Pi$-couples.*

1. $[A]_{\overline{s}} =_\beta \overline{s}A$.
2. *If none of the binding variables of $\overline{s}$ is free in A, then $[A]_{\overline{s}} \equiv A$ and for any segment $\overline{s_1}$, $\overline{s_1}(A\delta)\overline{s}B =_\beta \overline{s_1}\,\overline{s}(A\delta)B$.*

Proof: *$[A]_{\overline{s}} =_\beta \overline{s}A$ is by induction on $\mathtt{weight}(\overline{s})$; the other statement is now obvious.* □

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 $\overline{s}$ be a segment. Then the following holds:*

1. *Each bachelor main $\mathcal{O}$-item in $\overline{s}$ precedes each bachelor main δ-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 $\delta\mathcal{O}$-couples, leaves behind a $\underbrace{\mathcal{O}\cdots\mathcal{O}}_{n}\underbrace{\delta\cdots\delta}_{m}$-segment, consisting of all bachelor main $\mathcal{O}$- and δ-items.*

Proof: *1 is by induction on $\mathtt{weight}(\overline{s'})$ for $\overline{s} \equiv \overline{s'}(B\mathcal{O}_x)\overline{s''}$ and $(B\mathcal{O}_x)$ bachelor in $\overline{s}$. 2 and 3 are by induction on $\mathtt{weight}(\overline{s})$.* □

Note that we have assumed $\emptyset$ well-balanced. We assume it moreover non-bachelor. The following lemma is informative on the form of the terms:

Lemma 3.4 *Every term has one of the following three forms:*

- $(A\mathcal{O}_x)B$
- $(A_1\delta)\cdots(A_n\delta)C$, *where* $C \in V \cup \{*, \Box\}$ *and* $n \geq 0$
- $(A_1\delta)\cdots(A_n\delta)(B\delta)(C\mathcal{O}_x)D$, *where* $n \geq 0$

3.2 A reshuffling procedure and its properties

Definition 3.5 *The reshuffling function TS is defined such that:*

$$\begin{array}{lll} TS((B\mathcal{O}_x)C) & =_{df} & (TS(B)\mathcal{O}_x)TS(C) \\ TS((B_1\delta)\cdots(B_n\delta)A) & =_{df} & (TS(B_1)\delta)\cdots(TS(B_n)\delta)A \qquad \text{if } A \in V \cup \{*, \Box\} \\ TS((B_1\delta)\cdots(B_n\delta)(C\delta)(D\mathcal{O}_x)E) & =_{df} & (TS(C)\delta)(TS(D)\mathcal{O}_x)TS((B_1\delta)\cdots(B_n\delta)E) \end{array}$$

Note that the second and third clauses also apply for $n = 0$.

TS can be viewed to work in the following way: if a term starts with a main bachelor $\mathcal{O}$-item, the body of this item is reshuffled and the rest of the term is reshuffled. If a term does not start with a (bachelor) $\mathcal{O}$-item, then either it consists of (possibly zero) bachelor main δ-items followed by the heart of the term, in which case only the bodies of the main items are reshuffled, or it starts with some (possibly zero) bachelor δ-items followed by a well-balanced segment. Then TS looks for the leftmost $\delta\lambda$-segment and shifts all preceeding δ-items (bachelor as well as partnered) to the right of it. The bodies of the $\delta\lambda$-segment are reshuffled and the new, longer, term to the right of the $\delta\lambda$-segment is reshuffled.

Note that partnered δ-items which were to the left of the $\delta\lambda$-segment are now still partnered and next to their matching λ-items, and there are no bachelor δ-items to the left of a $\delta\lambda$-segment.

Hence, for any A, $TS(A)$ is of the form $\overline{s_0}\,\overline{s_1}A'$ where $A' \in V \cup \{*, \Box\}$, $\overline{s_1}$ consists of all bachelor main δ-items of A and $\overline{s_0}$ is of the form $\overline{s_2}\,\overline{s_3}\cdots\overline{s_n}$ where $\overline{s_i}$ is either a $\delta\mathcal{O}$-segment or a bachelor main $\mathcal{O}$-item.

As an example, the term

$$(w\lambda_x)(w\delta)\ (\overset{+}{x}\delta)(\overset{\bullet}{y}\delta)\ ((x\delta)\ (\overset{-}{y}\delta)(\overset{-}{z}\lambda_u)u\overset{\bullet}{\lambda}_v)\ (\overset{\times}{v}\delta)(\overset{''}{x}\delta)(\overset{''}{v}\lambda_w)(\overset{\times}{w}\lambda_t)(\overset{+}{y}\lambda_s)\ (s\delta)t$$

will be reshuffled to the term

$$(w\lambda_x)\ (\overset{\bullet}{y}\delta)\ ((\overset{-}{y}\delta)(\overset{-}{z}\lambda_u)\ (x\delta)u\ \overset{\bullet}{\lambda}_v)\ (\overset{''}{x}\delta)(\overset{''}{v}\lambda_w)(\overset{\times}{v}\delta)(\overset{\times}{w}\lambda_t)(\overset{+}{x}\delta)(\overset{+}{y}\lambda_s)\ (w\delta)(s\delta)t$$

One might wonder why TS moves bachelor δ-items but doesn't move bachelor $\mathcal{O}$-items. Consider $P \equiv (A\lambda_z)(B\delta)(C\lambda_x)(D\lambda_y)x$. Can we move a bachelor main $\mathcal{O}$-item (to the left or right)? The answer is no. For example, D may contain variables bound by the λ_x and we cannot rewrite P as $(A\lambda_z)(D\lambda_y)(B\delta)(C\lambda_x)x$. Moreover, in P, B and C may contain variables bound by λ_z so λ_z cannot move to the right of $(B\delta)(C\lambda_x)$. Hence, in a term, the order of the main $\mathcal{O}$-items is fixed and cannot in general be changed without changing the meaning of the term. Now, let us show the properties of TS.

Lemma 3.6 (Decidability of TS) *For any A, B, it is decidable whether $TS(A) \equiv TS(B)$.*

Proof: *This is obvious as $\equiv$ is decidable.* □

Lemma 3.7

1. *For all pseudo-expressions M, $TS(M)$ is well defined.*
2. *$FV(M) = FV(TS(M))$*
3. *If $\overline{s}$ is well-balanced, then $TS((A_1\delta)\cdots(A_n\delta)\overline{s}B) \equiv TS(\overline{s}(A_1\delta)\cdots(A_n\delta)B)$.*

Proof: *1. Every time at most one case of the definition of $TS(M)$ is applicable, and weights of the resulting terms to which TS is applied become smaller or TS disappears. 2. Induction on the structure of M. 3. By induction on* $\mathtt{weight}(\overline{s})$. □

Lemma 3.8 *For a term A, $TS(A) \equiv \overline{s_0}\,\overline{s_1}\heartsuit(A)$, where $\overline{s_1}$ consists of the term reshufflings of all bachelor main δ-items of A and $\overline{s_0}$ is a sequence of term reshufflings of main $\delta\mathcal{O}$-segments and bachelor main $\mathcal{O}$-items.*

Proof: *Induction on* $\mathtt{weight}(A)$.

- $A \equiv (B\mathcal{O}_x)C$, *use IH on* C.
- $A \equiv (B_1\delta)\cdots(B_n\delta)C$, $C \in V \cup \{*, \Box\}$. *Then* $\overline{s_0}$ *is empty.*
- $A \equiv (B_1\delta)\cdots(B_n\delta)(C\delta)(D\mathcal{O}_x)E$.

  Then $TS(A) \equiv (TS(C)\delta)(TS(D)\mathcal{O}_x)TS((B_1\delta)\cdots(B_n\delta)E)$. *By the induction hypothesis* $TS((B_1\delta)\cdots(B_n\delta)E)$ *is of the form* $\overline{s_0}\,\overline{s_1}\heartsuit(E) \equiv \overline{s_0}\,\overline{s_1}\heartsuit(A)$. □

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. *If A doesn't contain partnered* Π*-items then* $A =_\beta TS(A)$

Proof: *1. By induction on the structure of A.*

2. Induction on the number of symbols in A, using 1.

3. By induction on the number of symbols in A. If $A \equiv (A_1\delta)\cdots(A_n\delta)A'$ where $A' \in V \cup \{*, \Box\}$ or $A \equiv (A_1\mathcal{O}_x)A_2$ then use the induction hypothesis.

If $A \equiv (A_1\delta)\cdots(A_n\delta)(B\delta)(C\lambda_x)D$ then

$$\begin{array}{lcll} TS(A) & \equiv & (TS(B)\delta)(TS(C)\lambda_x)TS((A_1\delta)\cdots(A_n\delta)D) & \overset{IH}{=_\beta} \\ (B\delta)(C\lambda_x)(A_1\delta)\cdots(A_n\delta)D & =_\beta & ((A_1\delta)\cdots(A_n\delta)D)[x := B] & \overset{x\notin FV(A_i)}{=_\beta} \\ (A_1\delta)\cdots(A_n\delta)D[x := B] & =_\beta & (A_1\delta)\cdots(A_n\delta)(B\delta)(C\lambda_x)D & \end{array}$$

□

Corollary 3.10 *For all pseudo-expressions A, B without partnered* Π*-items: $TS(A) =_\beta TS(B)$ iff $A =_\beta B$.* □

Remark 3.11 Our notion of term reshuffling is related to the canonical forms and σ-equivalence of [Reg 92] and [Reg 94]. The difference is that Regnier studies untyped λ-terms and a Curry-style type system whereas we study Church-style type systems. Therefore, terms like $(A\delta)(B\lambda_x)(C\lambda_y)D$ and $(C\lambda_y)(A\delta)(B\lambda_x)D$ which have the same canonical form when the types C, B are omitted, cannot have the same term reshuffling due to the possibility of corruption of variable bindings.

4 Equivalence classes and shuffle $\beta\Pi$-reduction

Definition 4.1 *(Shuffle Class, shuffle-reduction, extended redexes and $\hookrightarrow_\beta$)*

- *For a pseudo-expression A, we define $[A]$, the shuffle class of A, to be $\{B \mid TS(A) \equiv TS(B)\}$.*
- *One-step shuffle-reduction $\rightsquigarrow_\beta$ is the least compatible relation generated by:*

  $$A \rightsquigarrow_\beta A' \text{ iff } \exists B \in [A] \exists B' \in [A'][B \to_\beta B']$$

  Note that $\rightsquigarrow_\beta$ is compatible and transitive because $\to_\beta$ is. Many-step shuffle-reduction $\rightsquigarrow\!\!\!\!\rightarrow_\beta$ is the reflexive and transitive closure of $\rightsquigarrow_\beta$ and $\approx_\beta$ is the least equivalence relation generated by $\rightsquigarrow\!\!\!\!\rightarrow_\beta$.
- *An extended redex starts with the δ-item of a* **$\delta\lambda$-couple** *(i.e. is of the form $(C\delta)\overline{s}(B\lambda_x)A$ where $\overline{s}$ is well-balanced).*
- *$\hookrightarrow_\beta$ is the least compatible relation generated by $(B_1\delta)\overline{s}(B_2\lambda_x)B_3 \hookrightarrow_\beta \overline{s}(B_3[x := B_1])$ for $\overline{s}$ well-balanced, that is, $\hookrightarrow_\beta$-reduction contracts an (extended) redex. $\hookleftarrow\!\!\!\!\twoheadrightarrow_\beta$ is the reflexive and transitive closure of $\hookrightarrow_\beta$ and $\sim_\beta$ be the least equivalence relation closed under $\hookleftarrow\!\!\!\!\twoheadrightarrow_\beta$.*

$\hookleftarrow\!\!\!\!\twoheadrightarrow_\beta$ has been used in [BKN 9y] and [KN 95]. We will use [BKN 9y] to obtain Strong Normalisation for the present paper.

Example 4.2 Let $A \equiv (z\delta)(w\delta)(u\lambda_x)(x\lambda_y)y$. Then $[A] = \{A, (w\delta)(u\lambda_x)(z\delta)(x\lambda_y)y\}$. Moreover, $A \rightsquigarrow_\beta (w\delta)(u\lambda_x)z$ and $A \rightsquigarrow_\beta (z\delta)(w\lambda_y)y$.

Lemma 4.3 *$TS(A) \hookrightarrow_\beta B$ iff $TS(A) \to_\beta B$.* **Proof:** *This is a direct consequence of 3.8*

Lemma 4.4 *If $A \rightsquigarrow_\beta B$ then for all $A' \in [A]$, for all $B' \in [B]$, $A' \rightsquigarrow_\beta B'$.*

Proof: *As $A \rightsquigarrow_\beta B$ then $\exists A_1 \in [A] \exists B_1 \in [B][A_1 \to_\beta B_1]$. Let $A', B' \in [A], [B]$ respectively. Then $A_1 \in [A']$, $B_1 \in [B']$, $A_1 \to_\beta B_1$. So $A' \rightsquigarrow_\beta B'$.* □

Corollary 4.5 $\quad A \rightsquigarrow_\beta B \Leftrightarrow TS(A) \rightsquigarrow_\beta TS(B)$

Remark 4.6 It is not in general true that $A \rightsquigarrow\!\!\!\!\rightarrow_\beta B \Rightarrow \exists A' \in [A] \exists B' \in [B][A' \twoheadrightarrow_\beta B']$. This can be seen by the following counterexample:

Let $A \equiv ((\alpha\lambda_u)(\alpha\lambda_v)v\delta)((\alpha\Pi_u)(\alpha\Pi_v)\alpha\lambda_x)(w\delta)(w\delta)x$ and $B \equiv (w\delta)(\alpha\lambda_u)w$. Then $A \rightsquigarrow_\beta (w\delta)(w\delta)(\alpha\lambda_u)(\alpha\lambda_v)v \rightsquigarrow_\beta B$. But $[A]$ has three elements, namely: A, $(w\delta)((\alpha\lambda_u)(\alpha\lambda_v)v\delta)((\alpha\Pi_u)(\alpha\Pi_v)\alpha\lambda_x)(w\delta)x$ and $(w\delta)(w\delta)((\alpha\lambda_u)(\alpha\lambda_v)v\delta)((\alpha\Pi_u)(\alpha\Pi_v)\alpha\lambda_x)x$, $[B] = \{B\}$ and if $A' \in [A]$ then the only $\to_\beta$ reduct of A' is $(w\delta)(w\delta)(\alpha\lambda_u)(\alpha\lambda_v)v$, which doesn't $\to_\beta$-reduce to B. In Lemma 4.12 however, we show that there is a correspondance between $\rightsquigarrow\!\!\!\!\rightarrow_\beta$ on classes and $\twoheadrightarrow_\beta$ on terms.

Lemma 4.7 *$\twoheadrightarrow_\beta \subseteq \hookleftarrow\!\!\!\!\twoheadrightarrow_\beta \subseteq \rightsquigarrow\!\!\!\!\rightarrow_\beta$.*

Proof: *It suffices to show $(A\delta)(B\lambda_x)C \hookrightarrow_\beta C[x := A]$ and $(A\delta)\overline{s}(B\lambda_x)C \rightsquigarrow_\beta \overline{s}C[x := A]$. But $(A\delta)(B\lambda_x)C \equiv (A\delta)\emptyset(B\lambda_x)C \hookrightarrow_\beta \emptyset C[x := A] \equiv C[x := A]$, by induction on $\mathtt{weight}(\overline{s})$ we can show that $(A\delta)\overline{s}(B\lambda_x)C \in [\overline{s}(A\delta)(B\lambda_x)C]$, and since $\overline{s}(A\delta)(B\lambda_x)C \to_\beta \overline{s}C[x := A]$ we have $(A\delta)\overline{s}(B\lambda_x)C \rightsquigarrow_\beta \overline{s}C[x := A]$.* □

Remark 4.8 Note that $A \hookrightarrow_\beta B \not\Rightarrow TS(A) \hookrightarrow\!\!\!\!\rightarrow_\beta TS(B)$ nor do we have $A \rightarrow_\beta B \Rightarrow TS(A) \twoheadrightarrow_\beta TS(B)$. Take for example A and B where $A \equiv ((z\lambda_u)(z\lambda_v)v\delta)(v\lambda_x)(y\delta)(y\delta)x$ and $B \equiv (y\delta)(y\delta)(z\lambda_u)(z\lambda_v)v$. It is obvious that $A \rightarrow_\beta B$ (hence $A \hookrightarrow_\beta B$) yet $TS(A) \equiv A \not\twoheadrightarrow_\beta$ and $\not\twoheadrightarrow_\beta TS(B) \equiv (y\delta)(z\lambda_u)(y\delta)(z\lambda_v)v$.

Lemma 4.9 *If $A \rightsquigarrow_\beta B$ or $A \hookrightarrow_\beta B$ then $A =_\beta B$.*

Proof: *For $\rightsquigarrow_\beta$: 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 $\hookrightarrow_\beta$: it suffices to consider the case $A \equiv \overline{s_1}(C\delta)\overline{s}(D\lambda_x)E$ where the contracted redex is based on $(C\delta)(D\lambda_x)$, $B \equiv \overline{s_1}\,\overline{s}(E[x := C])$, and $\overline{s}$ is well-balanced (hence $\mathtt{weight}(\overline{s})$ is even). We shall prove the lemma by induction on $\mathtt{weight}(\overline{s})$. If $\mathtt{weight}(\overline{s}) = 0$ then it is obvious as $\hookrightarrow_\beta$ coincides with $\rightarrow_\beta$ in this case. Assume the property holds when $\mathtt{weight}(\overline{s}) = 2n$. Take $\overline{s}$ such that $\mathtt{weight}(\overline{s}) = 2n + 2$. Now, $\overline{s} \equiv (C'\delta)\overline{s'}(D'\lambda_y)\overline{s''}$ where $\overline{s'}$, $\overline{s''}$ are well-balanced. Assume $x \not\equiv y$ (if necessary, use renaming).

- *As $\overline{s}(E[x := C]) \hookrightarrow_\beta \overline{s'}(\overline{s''}(E[x := C])[y := C'])$, we get by IH and compatibility that $B =_\beta \overline{s_1}\,\overline{s'}(\overline{s''}(E[x := C])[y := C']) \equiv \overline{s_1}\,\overline{s'}(\overline{s''}[y := C'])(E[x := C][y := C']) \equiv B''$.*
- *Moreover, $A \equiv \overline{s_1}(C\delta)(C'\delta)\overline{s'}(D'\lambda_y)\overline{s''}(D\lambda_x)E \hookrightarrow_\beta \overline{s_1}(C\delta)\overline{s'}(\overline{s''}(D\lambda_x)E[y := C']) \equiv^{BC} \overline{s_1}(C\delta)\overline{s'}(\overline{s''}[y := C'])(D[y := C']\lambda_x)(E[y := C']) \equiv B'$. Hence by IH, $A =_\beta B'$.*
- *Now, $B' \rightsquigarrow_\beta \overline{s_1}\overline{s'}(\overline{s''}[y := C'])(E[y := C'][x := C])$.*
  But by BC, x, $y \notin FV(C) \cup FV(C')$. Hence, by IH and substitution,
  $B' =_\beta \overline{s_1}\overline{s'}(\overline{s''}[y := C'])(E[x := C][y := C']) \equiv B''$.

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

Corollary 4.10

1. *If $A \rightsquigarrow\!\!\!\!\rightarrow_\beta B$ or $A \hookrightarrow\!\!\!\!\rightarrow_\beta B$ then $A =_\beta B$.*
2. *$A \approx_\beta B$ iff $A =_\beta B$ iff $A \sim_\beta B$ iff $TS(A) =_\beta TS(B)$.* □

Theorem 4.11 *(The general Church Rosser theorem for $\rightsquigarrow\!\!\!\!\rightarrow_\beta$) Let $\twoheadrightarrow$ be $\rightsquigarrow\!\!\!\!\rightarrow_\beta$ or $\hookrightarrow\!\!\!\!\rightarrow_\beta$.*
If $A \twoheadrightarrow B$ and $A \twoheadrightarrow C$, then there exists D such that $B \twoheadrightarrow D$ and $C \twoheadrightarrow D$.

Proof: *As $A \twoheadrightarrow B$ and $A \twoheadrightarrow C$ then by Corollary 4.10, $A =_\beta B$ and $A =_\beta C$. Hence, $B =_\beta C$ and by CR for $\twoheadrightarrow_\beta$, there exists D such that $B \twoheadrightarrow_\beta D$ and $C \twoheadrightarrow_\beta D$. But, $M \twoheadrightarrow_\beta N$ implies $M \twoheadrightarrow N$. Hence we are done.* □

As we noted in Remark 4.8, we can have $TS(C) \rightarrow_\beta D$ where $D \not\equiv TS(D)$. But we still can show that in a certain sense, term reshuffling preserves β-reduction.

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

$$\begin{array}{ccc} A & \xrightarrow{\qquad\qquad} & B \\ \Big| & & \Big| \\ TS(A) & \xrightarrow{\qquad\qquad} & B' \in [B] \end{array}$$

Proof: *We prove by induction on the structure of A' that if $A' \to_\beta B' \in [B]$, then for some B'', $TS(A') \to_\beta B'' \in [B]$. The compatibility cases are easy, distinguish cases according to the definition of TS. If $A' \equiv (C\delta)(D\lambda_x)E$ and $B' \equiv E[x := C] \in [B]$ then $TS(A') \equiv (TS(C)\delta)(TS(D)\lambda_x)TS(E) \to_\beta TS(E)[x := TS(C)]$ and by Lemma 3.9, $TS(TS(E)[x := TS(C)]) \equiv TS(E[x := C]) \in [B]$.*

□

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

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

Lemma 4.14 *Let $A \in SN_{\rightsquigarrow_\beta}$. Then for all $A' \in [A]$, $A' \sim_{\inf} A$.*

Proof: *It is sufficient to show that $(B\delta)\overline{s}C$ is reductionally equivalent to $\overline{s}(B\delta)C$ if $\overline{s}$ is well-balanced and $(B\delta)\overline{s}C \in SN_{\rightsquigarrow_\beta}$. We prove this by induction on the maximal length of $\rightsquigarrow_\beta$-reduction paths of $(B\delta)\overline{s}C$.*

If $(B\delta)\overline{s}C$ is in normalform then $\overline{s} \equiv \emptyset$ so $(B\delta)\overline{s}C \equiv \overline{s}(B\delta)C$. If $(B\delta)\overline{s}C$ is not in normalform then contraction of some redex yields a term which is either of the form $(B'\delta)\overline{s'}C'$ (if the redex was inside B, $\overline{s}$ or C) or of the form $\overline{s}C'$ if the redex consisted of $(B\delta)$ and its partnered item.

Then in the first case $\overline{s}(B\delta)C$ can reduce to $\overline{s'}(B'\delta)C'$ by contracting the corresponding redex, now by the induction hypothesis $(B'\delta)\overline{s'}C'$ is reductionally equivalent to $\overline{s'}(B'\delta)C'$. In the second case, $\overline{s}(B\delta)C$ also reduces to $\overline{s}C'$.

Hence $(B\delta)\overline{s}C$ is reductionally equivalent to $\overline{s}(B\delta)C$. □

Hence we have provided a relation between terms which approximates reductional equivalence. Here are some facts on this relation and on reductional equivalence:

1. Let $A \in SN_{\rightsquigarrow_\beta}$. If $TS(A) \equiv TS(B)$ then $A \sim_{\inf} B$ (Lemma 4.14).
2. $TS(A) \equiv TS(B) \not\Longleftrightarrow A \sim_{\inf} B$ (Example 4.15).
3. $A \sim_{\inf} B$ does not imply $TS(A) \equiv TS(B)$ (Example 4.16 below).
4. $TS(A) \equiv TS(B)$ is decidable (Lemma 3.6).
5. $A \sim_{\inf} B$ is not decidable (Conjecture 1.2).

Example 4.15 Take the terms A and B where $A \equiv (a\delta)(b\delta)(c\lambda_x)(d\lambda_y)((e\lambda_z)(z\delta)z\delta)(e\lambda_z)(z\delta)z$ and $B \equiv (b\delta)(c\lambda_x)(a\delta)(d\lambda_y)((e\lambda_z)(z\delta)z\delta)(e\lambda_z)(z\delta)z$. These terms read in classical notation $(\lambda_{x:c}.\lambda_{y:d}.\mathbf{\Omega})ba$ respectively $(\lambda_{x:c}.(\lambda_{y:d}.\mathbf{\Omega})a)b$. Now, $TS(A) \equiv TS(B)$ but $A \not\sim_{\inf} B$. This example shows that one cannot drop the assumption that A is strongly normalising.

Example 4.16 *Let $A \equiv ((a\delta)(b\lambda_x)x\delta)(c\lambda_y)y$ and $B \equiv (a\delta)(b\lambda_x)(x\delta)(c\lambda_y)y$. $A \sim_{\inf} B$ but $TS(A) \not\equiv TS(B)$. The same holds for the terms $(a\delta)(b\lambda_y)(y\delta)y$ and $(a\delta)(b\lambda_y)(y\delta)a$.*

We shall now show that due to the fact that shuffle-reduction on classes makes more redexes visible, it allows for smaller terms during reductions.

Example 4.17 Let $M \equiv (\lambda_{x:u}.\lambda_{y:u}.y(Cxx\cdots x))B(\lambda_{z:u}.u)$ where B is a BIG term. Then $M \to_\beta (\lambda_{y:u}.y(CBB\cdots B))(\lambda_{z:u}.u) \to_\beta (\lambda_{z:u}.u)(CBB\cdots B) \to_\beta u$ and u is in normal form. Now the first and second reducts both contain the segment $CBB\cdots B$, so they are very, very long terms. Shuffle reduction however allows us to reduce M in the following way: $TS(M) \equiv (\lambda_{x:u}.(\lambda_{y:u}.y(Cxx\cdots x))\lambda_{z:u}.u)B \to_\beta (\lambda_{x:u}.(\lambda_{z:u}.u)(Cxx\cdots x))B \to_\beta (\lambda_{x:u}.u)B \to_\beta u$, and in this reduction all the terms are of smaller size than M! So shuffle reduction might allow us to define clever strategies that reduce terms via paths of relatively small terms. Note also that the length of the reduction path to normal form doesn't change.

5 The λ-cube with equivalence classes, definitions and shuffle $\beta\Pi$-reduction

If we extend the λ-cube with $\twoheadrightarrow_\beta$ then Subject Reduction fails. That is: $\Gamma \vdash A : B$ and $A \twoheadrightarrow_\beta A' \not\Rightarrow \Gamma \vdash A' : B$.

Example 5.1 *(SR does not hold in $\lambda 2$ using $\twoheadrightarrow_\beta$)*
$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} (y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x : \beta$.
Moreover, $(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x \rightsquigarrow_\beta (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x$.
Yet, $(*\lambda_\beta)(\beta\lambda_{y'}) \not\vdash_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x : \beta$.
Even, $(*\lambda_\beta)(\beta\lambda_{y'}) \not\vdash_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x : \tau$ for any τ.
This is because $(\alpha\lambda_x)x : (\alpha\Pi_x)\alpha$ and $y : \beta$ yet α and β are unrelated and hence we fail in firing the application rule to find the type of $(y'\delta)(\alpha\lambda_x)x$. Looking closer however, one finds that $(\beta\delta)(*\lambda_\alpha)$ is defining α to be β, yet no such information can be used to combine $(\alpha\Pi_x)\alpha$ with β. Definitions take such information into account. Finally note that failure of SR in $\lambda 2$, means its failure in $\lambda P2, \lambda\omega$ and λC.

Example 5.2 *(SR does not hold in λP using $\twoheadrightarrow_\beta$)*
$(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q) * \lambda_Q)((t\delta)Q\lambda_N) \vdash_{\lambda P} (N\delta)(t\delta)(\sigma\lambda_x)((x\delta)Q\lambda_y)(y\delta)((x\delta)Q\lambda_Z)Z : (t\delta)Q$.
And $(N\delta)(t\delta)(\sigma\lambda_x)((x\delta)Q\lambda_y)(y\delta)((x\delta)Q\lambda_Z)Z \twoheadrightarrow_\beta (t\delta)(\sigma\lambda_x)(N\delta)((x\delta)Q\lambda_Z)Z$
Now, $N : (t\delta)Q, y : (x\delta)Q, (t\delta)Q \neq (x\delta)Q$.
$(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q) * \lambda_Q)((t\delta)Q\lambda_N) \not\vdash_{\lambda P} (t\delta)(\sigma\lambda_x)(N\delta)((x\delta)Q\lambda_Z)Z : \tau$ for any τ.
Here again the reason of failure is similar to the above example. At one stage, we need to match $(x\delta)Q$ with $(t\delta)Q$ but this is not possible even though we do have the definition segment: $(t\delta)(\sigma\lambda_x)$ which defines x to be t. All this calls for the need to use these definitions. Finally note that failure of SR in λP, means its failure in $\lambda P2, \lambda P\underline{\omega}$ and λC.

We conjecture that Subject Reduction is valid for $\lambda_\to$ and $\lambda\underline{\omega}$ with $\twoheadrightarrow_\beta$ and that the proof is similar to the one in [BKN 9y] for $\hookrightarrow\!\!\!\!\twoheadrightarrow_\beta$.

We extend the λ-cube with definitions, $\twoheadrightarrow_\beta$ and equivalence classes modulo TS. Contexts now consist of declarations $(A\lambda_x)$ and definitions. We take the typing rules $\vdash^{\mathtt{sh}}$ to be exactly those of $\vdash$ with the addition of the definition rule:

$$\text{(def rule)} \qquad \frac{\Gamma d \vdash^{\mathtt{sh}} C : D}{\Gamma \vdash^{\mathtt{sh}} dC : [D]_d} \qquad \text{if } d \text{ is a definition}$$

¿From the point of view of efficiency, it may seem unsatisfactory that in the (def rule) definitions are being unfolded in D, since this will usually mean a size explosion of the predicate. The unfolding is not necessary for non-topsorts (i.e. for $D \not\equiv \Box$) however:

Lemma 5.3 *The following rule is a derived rule:*

$$\textit{(derived def rule)} \qquad \frac{\Gamma d \vdash^e C : D \qquad \Gamma d \vdash^e D : S}{\Gamma \vdash^e dC : dD} \text{ if } d \text{ is a definition}$$

Proof: *If* $\Gamma d \vdash^e C : D$ *then by the (def rule),* $\Gamma \vdash^e dC : [D]_d$*; if* $\Gamma d \vdash^e D : S$ *then by the (def rule)* $\Gamma \vdash^e dD : S$*. Now by conversion* $\Gamma \vdash^e dC : dD$ *since* $\Gamma \vdash^e dD =_{\mathtt{def}} [D]_d$*.*

If D is a sort then of course unfolding d in D is not inefficient since d will disappear.

Due to the possibility of using the (def rule) to type a redex, by using the (derived def rule), in some cases it is even possible to circumvent a size explosion: suppose we want to derive in λC a type for the term $(B\delta)(*\lambda_\beta)(\beta\lambda_x)((\beta\Pi_y)\beta\lambda_f)(x\delta)f$.

In λC without definions, we will have to derive first the type $(*\Pi_\beta)(\beta\Pi_x)((\beta\Pi_y)\beta\Pi_f)\beta$ for the subterm $(*\lambda_\beta)(\beta\lambda_x)((\beta\Pi_x)\beta\lambda_f)(x\delta)f$, and by the application rule we will finally derive the type $(B\Pi_x)((B\Pi_y)B\Pi_f)B$. Note that due to the last applied application rule the term B has been copied four times, which could make the resulting type very large.

Using our type system extended with definitions however, we would first derive the type $(\beta\Pi_x)((\beta\Pi_y)\beta\Pi_f)\beta$ for the term $(\beta\lambda_x)((\beta\Pi_y)\beta\lambda_f)(x\delta)f$, and then by the derived definition rule we would derive the type $(B\delta)(*\lambda_\beta)(\beta\Pi_x)((\beta\Pi_y)\beta\Pi_f)\beta$ and avoid the substitution of B for β. This is a further evidence for the advantage of using definitions.

Now, we proceed to show the properties of $\vdash^{\mathtt{sh}}$.

Lemma 5.4 *(Free Variable Lemma for* $\vdash^{\mathtt{sh}}$*)*
Let Γ *be a legal context such that* $\Gamma \vdash^{\mathtt{sh}} B : C$*. Then the following holds:*

1. *If* d *and* d' *are two different elements of* Γ-$\mathtt{decl} \cup \Gamma$-$\mathtt{def}$*, then* $\mathtt{subj}(d) \not\equiv \mathtt{subj}(d')$*.*
2. $FV(B), FV(C) \subseteq dom(\Gamma)$*.*
3. *If* $\Gamma = \Gamma_1 s_1 \Gamma_2$ *then* $FV(s_1) \subseteq dom(\Gamma_1)$*.*

Proof: *All by induction on the derivation of* $\Gamma \vdash^{\mathtt{sh}} B : C$*.* □

Lemma 5.5 *(Start Lemma for* $\vdash^{\mathtt{sh}}$*)*
Let Γ *be a legal context. Then* $\Gamma \vdash^{\mathtt{sh}} * : \Box$ *and* $\forall d \in \Gamma[\Gamma \vdash^{\mathtt{sh}} d]$*.*

Proof: Γ *is legal* $\Rightarrow \exists B, C[\Gamma \vdash^{\mathtt{sh}} B : C]$*; use induction on the derivation* $\Gamma \vdash^{\mathtt{sh}} B : C$*.* □

Lemma 5.6 *(Transitivity Lemma for* $\vdash^{\mathtt{sh}}$*)*
Let Γ *and* Δ *be legal contexts and define* $\Gamma \vdash^{\mathtt{sh}} \Delta$ *as usual. Then we have:*
$[\Gamma \vdash^{\mathtt{sh}} \Delta \wedge \Delta \vdash^{\mathtt{sh}} A : B] \Rightarrow \Gamma \vdash^{\mathtt{sh}} A : B$*.*

Proof: *Induction on the derivation* $\Delta \vdash^{\mathtt{sh}} A : B$*. Note that by the compatibility of* $\Gamma \vdash^{\mathtt{sh}} C =_{\mathtt{def}} D$ *it follows that if* $d \in \Delta$ *and* D *arises from* C *by substituting one particular free occurrence of* $\mathtt{subj}(d)$ *in* C *by* $\mathtt{def}(d)$*, then* $\Gamma \vdash^{\mathtt{sh}} C =_{\mathtt{def}} D$ *and hence* $\Delta \vdash^{\mathtt{sh}} C =_{\mathtt{def}} D$ *implies* $\Gamma \vdash^{\mathtt{sh}} C =_{\mathtt{def}} D$*.* □

Lemma 5.7 *(Definition-shuffling for* $\vdash^{\mathtt{sh}}$*) Let* d *be a definition.*

1. *If* $\Gamma d \Delta \vdash^{\mathtt{sh}} C =_{\mathtt{def}} D$ *then* $\Gamma \underline{d}(\mathtt{def}(d)\delta)(\mathtt{pred}(d)\lambda_{\mathtt{subj}(d)})\Delta \vdash^{\mathtt{sh}} C =_{\mathtt{def}} D$*.*
2. *If* $\Gamma d \Delta \vdash^{\mathtt{sh}} C : D$ *then* $\Gamma \underline{d}(\mathtt{def}(d)\delta)(\mathtt{pred}(d)\lambda_{\mathtt{subj}(d)})\Delta \vdash^{\mathtt{sh}} C : D$*.*

Proof: *1. is by induction on the generation of* $\Gamma d \Delta \vdash^{\mathtt{sh}} C =_{\mathtt{def}} D$*. 2. is by induction on the derivation of* $\Gamma d \Delta \vdash^{\mathtt{sh}} C : D$ *using 1. for conversion.* □

Lemma 5.8 *(Thinning for $\vdash^{\mathtt{sh}}$)*

1. *If $\Gamma_1\Gamma_2 \vdash^{\mathtt{sh}} A =_{\mathtt{def}} B$, $\Gamma_1\Delta\Gamma_2$ is a legal context, then $\Gamma_1\Delta\Gamma_2 \vdash^{\mathtt{sh}} A =_{\mathtt{def}} B$.*
2. *If Γ and Δ are legal contexts such that $\Gamma \subseteq' \Delta$ and $\Gamma \vdash^{\mathtt{sh}} A : B$, then $\Delta \vdash^{\mathtt{sh}} A : B$.*

Proof: *1. is by induction on the derivation $\Gamma_1\Gamma_2 \vdash^{\mathtt{sh}} A =_{\mathtt{def}} B$. 2. is done by showing:*

- *If $\Gamma\Delta \vdash^{\mathtt{sh}} A : B$, $\Gamma \vdash^{\mathtt{sh}} C : S$, x is fresh, and no λ-item in Δ is partnered by a δ-item in Γ, then also $\Gamma(C\lambda_x)\Delta \vdash^{\mathtt{sh}} A : B$. We show this by induction on the derivation $\Gamma\Delta \vdash^{\mathtt{sh}} A : B$ using 1. for conversion.*
- *If $\Gamma\overline{s}\Delta \vdash^{\mathtt{sh}} A : B$, $\Gamma\overline{s} \vdash^{\mathtt{sh}} C : D : S$, $FV(C) \subseteq dom(\Gamma)$, x is fresh, $\overline{s}$ is well-balanced, then also $\Gamma(C\delta)\overline{s}(D\lambda_x)\Delta \vdash^{\mathtt{sh}} A : B$. We show this by induction on $\Gamma\overline{s}\Delta \vdash^{\mathtt{sh}} A : B$. In the case of (start) for instance where $\Gamma(A\delta)\overline{s}(B\lambda_y) \vdash^{\mathtt{sh}} y : A$ comes from $\Gamma\overline{s} \vdash^{\mathtt{sh}} A : B : S$, y fresh and $FV(A) \subseteq dom(\Gamma)$, then $\Gamma(C\delta)\overline{s}(D\lambda_x) \vdash^{\mathtt{sh}} A : B : S$ by IH so again by (start), $\Gamma(C\delta)(A\delta)\overline{s}(B\lambda_y)(D\lambda_x) \vdash^{\mathtt{sh}} x : A$.*
- *If $\Gamma\overline{s}(A\lambda_x)\Delta \vdash^{\mathtt{sh}} B : C, (A\lambda_x)$ bachelor, $\overline{s}$ well-balanced, $\Gamma\overline{s} \vdash^{\mathtt{sh}} D : A$, $FV(D) \subseteq dom(\Gamma$, then $\Gamma(D\delta)\overline{s}(A\lambda_x)\Delta \vdash^{\mathtt{sh}} B : C$. We show this by induction on $\Gamma\overline{s}(A\lambda_x)\Delta \vdash^{\mathtt{sh}} B : C$.* □

Lemma 5.9 *(Substitution lemma for $\vdash^{\mathtt{sh}}$) Let d be a definition.*

1. *If $\Gamma d\Delta \vdash^{\mathtt{sh}} A =_{\mathtt{def}} B$, A and B are $\Gamma d\Delta$-legal terms, then $\Gamma\underline{d}\Delta[\mathtt{subj}(d) := \mathtt{def}(d)] \vdash^{\mathtt{sh}} A[\mathtt{subj}(d) := \mathtt{def}(d)] =_{\mathtt{def}} B[\mathtt{subj}(d) := \mathtt{def}(d)]$*
2. *If B is a Γd-legal term, then $\Gamma d \vdash^{\mathtt{sh}} B =_{\mathtt{def}} [B]_d$*
3. *If $\Gamma(A\delta)\overline{s}(B\lambda_x)\Delta \vdash^{\mathtt{sh}} C : D$ then $\Gamma\overline{s}(\Delta[x := A]) \vdash^{\mathtt{sh}} C[x := A] : D[x := A]$*
4. *If $\Gamma(B\lambda_x)\Delta \vdash^{\mathtt{sh}} C : D$, $\Gamma \vdash^{\mathtt{sh}} A : B$, $(B\lambda_x)$ bachelor in Γ, then $\Gamma\Delta[x := A] \vdash^{\mathtt{sh}} C[x := A] : D[x := A]$*
5. *If $\Gamma d\Delta \vdash^{\mathtt{sh}} C : D$, then $\Gamma[\Delta]_d \vdash^{\mathtt{sh}} [C]_d : [D]_d$*

Proof: *1. Induction to the derivation rules of $=_{\mathtt{def}}$. 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.* □

Lemma 5.10 *(Generation Lemma for $\vdash^{\mathtt{sh}}$)*

1. *If $\Gamma \vdash^{\mathtt{sh}} x : A$ then for some B: $(B\lambda_x) \in \Gamma$, $\Gamma \vdash^{\mathtt{sh}} B : S$, $\Gamma \vdash^{\mathtt{sh}} A =_{\mathtt{def}} B$ and $\Gamma \vdash^{\mathtt{sh}} A : S'$ for some sort S'.*
2. *If $\Gamma \vdash^{\mathtt{sh}} (A\lambda_x)B : C$ then for some D and sort S: $\Gamma(A\lambda_x) \vdash^{\mathtt{sh}} B : D$, $\Gamma \vdash^{\mathtt{sh}} (A\Pi_x)D : S$, $\Gamma \vdash^{\mathtt{sh}} (A\Pi_x)D =_{\mathtt{def}} C$ and if $(A\Pi_x)D \not\equiv C$ then $\Gamma \vdash^{\mathtt{sh}} C : S'$ for some sort S'.*
3. *If $\Gamma \vdash^{\mathtt{sh}} (A\Pi_x)B : C$ then for some sorts S_1, S_2: $\Gamma \vdash^{\mathtt{sh}} A : S_1$, $\Gamma \vdash^{\mathtt{sh}} B : S_2$, $(S_1, S_2) \in \mathcal{R}$, $\Gamma \vdash^{\mathtt{sh}} C =_{\mathtt{def}} S_2$ and if $S_2 \not\equiv C$ then $\Gamma \vdash^{\mathtt{sh}} C : S$ for some sort S.*
4. *If $\Gamma \vdash^{\mathtt{sh}} (A\delta)B : C$, $(A\delta)$ bachelor in B , then for some terms D, E, variable x: $\Gamma \vdash^{\mathtt{sh}} A : D$, $\Gamma \vdash^{\mathtt{sh}} B : (D\Pi_x)E$, $\Gamma \vdash^{\mathtt{sh}} E[x := A] =_{\mathtt{def}} C$ and if $E[x := A] \not\equiv C$ then $\Gamma \vdash^{\mathtt{sh}} C : S$ for some sort S.*

5. *If* $\Gamma \vdash^{\mathtt{sh}} \overline{s}A : B$, *then* $\Gamma\overline{s} \vdash^{\mathtt{sh}} A : B$

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* $\mathtt{weight}(\overline{s})$. □

Corollary 5.11 *(Correctness of Types)*
If $\Gamma \vdash^{\mathtt{sh}} A : B$ *then* $B \equiv \Box$ *or* $\Gamma \vdash^{\mathtt{sh}} B : S$ *for some sort* S.
Proof: *By induction to the derivation rules. The interesting cases are the definition and application rules. In case* $\Gamma \vdash^{\mathtt{sh}} dA : [B]_d$ *as a consequence of* $\Gamma d \vdash^{\mathtt{sh}} A : B$, *then by IH* $B \equiv \Box$ *or* $\Gamma d \vdash^{\mathtt{sh}} B : S$ *for some sort* S. *In the first case also* $[B]_d \equiv \Box$, *in the second case by the Substitution Lemma* $\Gamma \vdash^{\mathtt{sh}} [B]_d : [S]_d \equiv S$.
In case $\Gamma \vdash^{\mathtt{sh}} (a\delta)F : B[x := a]$ *as a consequence of* $\Gamma \vdash^{\mathtt{sh}} F : (A\Pi_x)B$, $\Gamma \vdash^{\mathtt{sh}} a : A$, *then by the induction hypothesis* $\Gamma \vdash^{\mathtt{sh}} (A\Pi_x)B : S$ *for some sort* S *and hence by Generation* $\Gamma(A\lambda_x) \vdash^{\mathtt{sh}} B : S$. *Then by Thinning* $\Gamma(a\delta)(A\lambda_x) \vdash^{\mathtt{sh}} B : S$, *so by the definition rule* $\Gamma \vdash^{\mathtt{sh}} (a\delta)(A\Pi_x)B : S[x := a] \equiv S$ □

Now, firstly we prove SR for $\vdash^{\mathtt{sh}}$ using $\twoheadrightarrow_\beta$ rather than $\rightsquigarrow_\beta$.

Theorem 5.12 *(Subject Reduction for* $\vdash^{\mathtt{sh}}$ *and* $\twoheadrightarrow_\beta$*)*
If $\Gamma \vdash^{\mathtt{sh}} A : B$ *and* $A \twoheadrightarrow_\beta A'$ *then* $\Gamma \vdash^{\mathtt{sh}} A' : B$.
Proof: *By simultaneous induction on the derivation rules:*

1. *If* $\Gamma \vdash^{\mathtt{sh}} A : B$ *and* $\Gamma \to_\beta \Gamma'$ *then* $\Gamma' \vdash^{\mathtt{sh}} A : B$
2. *IF* $\Gamma \vdash^{\mathtt{sh}} A : B$ *and* $A \to_\beta A'$ *then* $\Gamma \vdash^{\mathtt{sh}} A' : B$ □

Lemma 5.13 *If* $\Gamma \vdash^{\mathtt{sh}} A : B$ *and* $A' \in [A]$, Γ' *results from* Γ *by substituting some main items* $(C\omega)$ *by* $(C'\omega)$ *where* $C' \in [C]$, *then* $\Gamma' \vdash^{\mathtt{sh}} A' : B$.
Proof: *Induction on the derivation rules. We treat two cases:*

- *(start):* $\Gamma(A\delta)\underline{d}(B\mathcal{O}_x) \vdash^{\mathtt{sh}} x : B$ *as a consequence of* $\Gamma\underline{d} \vdash^{\mathtt{sh}} A : B$, $\Gamma\underline{d} \vdash^{\mathtt{sh}} B : S$ *and* $FV(A) \subseteq dom(\Gamma)$.

  We must show $\Gamma'(A'\delta)\underline{d'}(B'\mathcal{O}_x) \vdash^{\mathtt{sh}} x : B$. *By the induction hypothesis* $\Gamma'\underline{d'} \vdash^{\mathtt{sh}} A' : B$, $\Gamma'\underline{d'} \vdash^{\mathtt{sh}} B' : S$, *by Lemma 3.9,* $B =_\beta B'$ *so by conversion* $\Gamma'\underline{d'} \vdash^{\mathtt{sh}} A' : B'$, *by Lemma 3.7* $FV(A') = FV(A) \subseteq dom(\Gamma)$, *hence by the start rule* $\Gamma'(A'\delta)\underline{d'}(B'\mathcal{O}_x) \vdash^{\mathtt{sh}} x : B'$ *and by conversion* $\Gamma'(A'\delta)\underline{d'}(B'\mathcal{O}_x) \vdash^{\mathtt{sh}} x : B$.

- *(definition):* $\Gamma \vdash^{\mathtt{sh}} dA : [B]_d$ *as a consequence of* $\Gamma d \vdash^{\mathtt{sh}} A : B$.

  By the induction hypothesis $\Gamma' d'' \vdash^{\mathtt{sh}} A' : B$, *where* d'' *is the items of* d' *in the order of* d. *Now by Lemma 5.7* $\Gamma' d' \vdash^{\mathtt{sh}} A' : B$ *and by the definition rule* $\Gamma' \vdash^{\mathtt{sh}} d'A' : [B]_{d'}$. *By the induction hypothesis also* $\Gamma' d \vdash^{\mathtt{sh}} A : B$, *hence* $\Gamma' \vdash^{\mathtt{sh}} dA : [B]_d$, *so by Lemma 5.11* $[B]_d \equiv \Box$ *or* $\Gamma' \vdash^{\mathtt{sh}} [B]_d : S$ *for some sort* S. *In the first case also* $[B]_{d'} \equiv \Box$ *and we are done, in the second case by Lemma 3.9* $[B]_d =_\beta [B]_{d'}$ *so by conversion* $\Gamma' \vdash^{\mathtt{sh}} d'A' : [B]_d$.
  □

Corollary 5.14 *(*TS *preserves typing)*

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

Proof:

1. *By lemma 5.13 as* $A \in [TS(A)]$ *and* $TS(A) \in [A]$.

2. *By lemma 5.13 using Correctness of Types and conversion.* □

Here is now the proof of SR using $\vdash^{\mathtt{sh}}$ and $\rightsquigarrow\!\!\!\!\rightarrow_\beta$.

Corollary 5.15 *(Subject Reduction for* $\vdash^{\mathtt{sh}}$ *and* $\rightsquigarrow\!\!\!\!\rightarrow_\beta$*)*
If $\Gamma \vdash^{\mathtt{sh}} A : B$ *and* $A \rightsquigarrow\!\!\!\!\rightarrow_\beta A'$ *then* $\Gamma \vdash^{\mathtt{sh}} A' : B$.
Proof: *We prove* $\Gamma \vdash^{\mathtt{sh}} A : B$, $A \rightsquigarrow_\beta A' \Longrightarrow \Gamma \vdash^{\mathtt{sh}} A' : B$.
By Corollary 5.14 $\Gamma \vdash^{\mathtt{sh}} TS(A) : B$, *by Lemma 4.12 there is a term* C *such that* $TS(A) \rightarrow_\beta C$ *and* $C \in [A']$, *now by Theorem 5.12* $\Gamma \vdash^{\mathtt{sh}} C : B$ *and by Lemma 5.14* $\Gamma \vdash^{\mathtt{sh}} A' : B$. □

Lemma 5.16 (Unicity of Types for $\vdash^{\mathtt{sh}}$**)**

1. $\Gamma \vdash^{\mathtt{sh}} A : B \wedge \Gamma \vdash^{\mathtt{sh}} A : B' \Rightarrow \Gamma \vdash^{\mathtt{sh}} B =_{\mathtt{def}} B'$

2. $\Gamma \vdash^{\mathtt{sh}} A : B \wedge \Gamma \vdash^{\mathtt{sh}} A' : B' \wedge A =_\beta A' \Rightarrow \Gamma \vdash^{\mathtt{sh}} B =_{\mathtt{def}} B'$

Proof:

1. *By induction on the structure of* A *using the Generation Lemma.*

2. *By Church-Rosser and Subject Reduction using 1.* □

Remark 5.17 We didn't prove the property $\Gamma \vdash^{\mathtt{sh}} B : S$, $\Gamma \vdash^{\mathtt{sh}} A : B'$, $B =_\beta B' \Rightarrow \Gamma \vdash^{\mathtt{sh}} B' : S$. It seems difficult to prove because if $\Gamma \vdash^{\mathtt{sh}} B' : S'$ then by Unicity of Types $\Gamma \vdash^{\mathtt{sh}} S =_{\mathtt{def}} S'$ and it is unclear whether $S \equiv S'$.

Also, it would be interesting whether $\Gamma \vdash^{\mathtt{sh}} A : B$, $\Gamma \vdash^{\mathtt{sh}} A' : B'$, $\Gamma \vdash^{\mathtt{sh}} A =_{\mathtt{def}} A'$ implies $\Gamma \vdash^{\mathtt{sh}} B =_{\mathtt{def}} B'$, but to prove this we face similar problems. We claim that one can prove it by showing first that $\Gamma \vdash^{\mathtt{sh}} A : B$ implies $\Gamma \vdash^{\mathtt{sh}} [A]_\Gamma : [B]_\Gamma$, where $[A]_\Gamma$ means all definitions in Γ to be unfolded in A.

We don't need these properties for our theory however.

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

Lemma 5.18 *If* $\Gamma \vdash^{\mathtt{sh}} A : B$ *then* $\Gamma \vdash^e A : B$, *where* $\vdash^e$ *is the typing relation of systems of the* λ*-cube extended with definitions and generalised reduction.*
Proof: *Induction on the derivation rules of* $\vdash^{\mathtt{sh}}$. *All rules are trivial since they are also rules in* $\vdash^e$. □

Corollary 5.19 *If* A *is a* $\vdash^{\mathtt{sh}}$*-legal term then* A *is strongly normalising with respect to* $\hookrightarrow\!\!\!\!\rightarrow_\beta$.
Proof: *If* A *is* $\vdash^{\mathtt{sh}}$*-legal then* A *is* $\vdash^e$*-legal by Lemma 5.18 and hence* A *is strongly normalising with respect to* $\hookrightarrow\!\!\!\!\rightarrow_\beta$ *(see [BKN 9y]).* □

Definition 5.20 *For a* $\vdash^{\mathtt{sh}}$*-legal term* A, *define the natural number* $\mathtt{height}(A)$ *to be the maximal length of a* $\hookrightarrow\!\!\!\!\rightarrow_\beta$*-reduction path starting with* A.

Lemma 5.21

1. *If A is legal and $A \hookrightarrow_\beta B$, then* $\mathtt{height}(A) > \mathtt{height}(B)$.
2. *If A is legal and $A' \in [A]$, then* $\mathtt{height}(A') = \mathtt{height}(A)$.
3. *If A is legal and $A \rightsquigarrow_\beta B$, then* $\mathtt{height}(A) > \mathtt{height}(B)$.

Proof: *Long but straightforward.* □

Corollary 5.22 *Every legal term is strongly normalising with respect to $\rightsquigarrow\!\!\!\!\rightarrow_\beta$.* □

Fact 5.23 *Subtyping does not hold for $\vdash^{\mathtt{sh}}$. Consider the following derivable judgement:*

$$(*\lambda_\alpha) \vdash^{\mathtt{sh}} (\alpha\delta)(*\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z : (\alpha\Pi_y)\alpha$$

The subterm $(*\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z$ is not typable: suppose $\Gamma \vdash^{\mathtt{sh}} (*\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z : A$, then by the Generation Lemma, $\Gamma' \vdash^{\mathtt{sh}} z : \alpha'$ where $\Gamma' \equiv \Gamma(*\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)$ and α' satisfies $\Gamma' \vdash^{\mathtt{sh}} \alpha =_{\mathtt{def}} \alpha'$ and $\Gamma' \vdash^{\mathtt{sh}} \alpha' : S$.

Since Γ cannot contain bachelor δ-items, we know that $(*\lambda_\beta)$ is not partnered in Γ', hence $\Gamma' \not\vdash^{\mathtt{sh}} \alpha =_{\mathtt{def}} \beta$. But since $(y\delta)(\alpha\lambda_z) \in \Gamma'$-$\mathtt{def}$ we know that $\Gamma(*\lambda_\beta)(\beta\lambda_y) \vdash^{\mathtt{sh}} y : \alpha : S$, also $\Gamma(*\lambda_\beta)(\beta\lambda_y) \vdash^{\mathtt{sh}} y : \beta$ so by Unicity of Types, $\Gamma(*\lambda_\beta)(\beta\lambda_y) \vdash^{\mathtt{sh}} \alpha =_{\mathtt{def}} \beta$, contradiction.

The reason for failure of subtyping is that when we typed the term $(\alpha\delta)(*\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z$, we used the context $(*\lambda_\alpha)(\alpha\delta)(*\lambda_\beta)$ to type $(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z$. In this context, β is defined to be α. Now, to type $(*\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z$, the definition $(\alpha\delta)(*\lambda_\beta)$ cannot be used. Hence, we don't have all the information necessary to derive the type of $(*\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z$. We do however have a partial result concerning subtyping:

Lemma 5.24 (Restricted Subtyping) *If $\Gamma \vdash^{\mathtt{sh}} A : B$, A' is a subterm of A such that all bachelor items in A' are also bachelor in A, then A' is legal.*

Proof: *We prove by induction on the derivations: if A' is a subterm of Γ or A such that all bachelor items in A' are also bachelor items in Γ respectively A, then A' is legal.*

Note that in the case of the (def rule) subterms $\overline{s_2}C$ where $d \equiv \overline{s_1}\,\overline{s_2}$ and $\overline{s_1}$ is not the empty segment, do not satisfy the restrictions, since at least one item of $\overline{s_2}$ is bachelor in $\overline{s_2}C$ but partnered in dC. □

Subterms satisfying the bachelor restriction as in Lemma 5.24 above, seem to be more important than those not satisfying the bachelor restriction. The reason for this is that the latter terms have an extra abstraction (the newly bachelor λ-item) and hence are Π-types which makes them more involved, whereas the subterm property is usefull because it tells something about less involved terms.

6 Conclusion and Comparison

We have proposed an extension of β-reduction called shuffle-reduction, which makes more redexes visible and hence allows for more flexibility in reducing a term. It seems a feasible approximation of the informal notion of reductional equivalence.

We used the item-notation to give a clearer description of term shuffling and shuffle-reduction and to be able to add nested definitions to typing systems. We think that the

item-notation is a good candidate for answering the two questions posed in the conclusions of [Reg 94] concerning the existence of a syntax for terms realising shuffle-equivalence (which Regnier calls σ-equivalence, see below).

Shuffle reduction is shown to behave well with respect to several aspects of the typed λ-calculi of the Barendregt cube. As far as reduction is concerned, shuffle-reduction has the Church-Rosser property, shuffle-equivalence classes partition β-equivalence classes into smaller parts and the equivalence relation generated by shuffle-reduction is just β-equality.

Furthermore the typing systems with shuffle-reduction are shown to have the same nice properties as the typing systems with β-reduction possess, providing that they are extended with definitions.

We showed that using shuffle-reduction we indeed may avoid size explosion without the cost of a longer reduction path.

Before closing, it is worth mentioning where reductions related to our generalised notion have been used elsewhere. At the time of writing this paper, we were unaware of many related work and we are grateful to Joe Wells who has compiled most of the following details. We will be short in what follows but we refer to [KW 95b] which discusses the subject in detail.

Here are two rules related to our term reshuffling:

$$(\theta) \qquad (Q\delta)(P\delta)(\lambda_x)N \to (P\delta)(\lambda_x)(Q\delta)N$$

$$(\gamma) \qquad (P\delta)(\lambda_x)(\lambda_y)N \to (\lambda_y)(P\delta)(\lambda_x)N$$

It is obvious that θ may move the δ-item $(Q\delta)$ next to a λ-item in N if $N \equiv (\lambda_y)M$, and hence the δ-couple $(Q\delta)(\lambda_y)$ becomes a δ-pair making the generalised redex a classical one (visible) and subject to contraction. The rule γ is unrelated to what we do here yet has almost always been used with θ for technical reasons. Furthermore, the transfer of rule γ to explicitly typed lambda calculus is not straightforward, since the type of y may be affected by the reducible pair $(P\delta)(\lambda_x)$. This is our reason for avoiding γ. In fact, in explicitly typed λ-calculi, γ does not return reductionally equivalent terms.

Regnier's notion of 'premier redex' (see [Reg 92]) is the same as our notion of generalised redex on untyped terms. We study it for Church-style type systems whereas Regnier studies Curry-style type systems. [Reg 94] uses θ and γ (and calls the combination σ) to show that the perpetual reduction strategy finds the longest reduction path when the term is SN. [Vid 89] also introduces reductions similar to those of [Reg 94]. Furthermore, [KTU 94] uses θ (and other reductions) to show that typability in ML is equivalent to acyclic semi-unification. [SF 92] uses a reduction which has some common themes to θ. [dG 93] uses a restricted version of θ and [KW 95a] uses γ to reduce the problem of strong normalisation for β-reduction to the problem of weak normalisation for related reductions. [KW 94] uses amongst other things, θ and γ to reduce typability in the rank-2 restriction of system F to the problem of acyclic semi-unification. [AFM 95] uses θ (which they call "let-C") as a part of an analysis of how to implement sharing in a real language interpreter in a way that directly corresponds to a formal calculus.

7 Acknowledgements

We are grateful for the useful discussions with Henk Barendregt, Bob Constable, Jan-Willem Klop and Joe Wells.

References

[AFM 95] Ariola, Z.M. Felleisen, M. Maraist, J. Odersky, M. and Wadler, P., A call by need lambda calculus, *Conf. Rec. 22nd Ann. ACM Symp. Princ. Program. Lang. ACM*, 1995.

[Barendregt 92] Barendregt, H., Lambda calculi with types, *Handbook of Logic in Computer Science*, volume II, ed. Abramsky S., Gabbay D.M., Maibaum T.S.E., Oxford University Press, 1992.

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

[BKN 9y] Bloo, R., Kamareddine, F., Nederpelt, R., The Barendregt Cube with Definitions and Generalised Reduction, Computing Science Note, University of Glasgow, Computing Science department, 1994. To appear in *Information and Computation*.

[Con 86] Constable, R.L. et al., *Implementing Mathematics with the Nuprl proof development system*, Prentice Hall 1986.

[Dow 91] Dowek, G. et al. The Coq proof assistant version 5.6, users guide, rapport de recherche 134, INRIA, 1991.

[Gardner 94] Gardner, P., Discovering Needed Reductions Using Type Theory, TACS, 1994.

[dG 93] de Groote, P., The conservation theorem revisited, *Int'l Conf. Typed Lambda Calculi and Applications*, vol. 664 of LNCS, 163-178, Springer-Verlag, 1993.

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

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

[KN 95] Kamareddine, F., and Nederpelt, R.P., Generalising reduction in the λ-calculus, *Journal of Functional Programming 5 (4)*, 1995.

[KN 96a] Kamareddine, F., and Nederpelt, R.P., On Π-conversion in the Barendregt Cube, *Journal of Functional Programming 6 (2)*, 1996.

[KN 96b] Kamareddine, F., and Nederpelt, R.P., A useful λ-notation, *Theoretical Computer Science 155*, 1996.

[KBN 9-] Kamareddine, F., Bloo, R., and Nederpelt, R.P., Definitions and Π-reductions in type theory, submitted.

[KTU 94] Kfoury, A.J., Tiuryn, J. and Urzyczyn, P., An analysis of ML typability, J. ACM 41(2), 368-398, 1994.

[KW 94] Kfoury, A.J. and Wells, J.B., A direct algorithm for type inference in the rank-2 fragment of the second order λ-calculus, *Proc. 1994 ACM Conf. LISP Funct. Program.*, 1994.

[KW 95a] Kfoury, A.J. and Wells, J.B., New notions of reductions and non-semantic proofs of β-strong normalisation in typed λ-calculi, *LICS*, 1995.

[KW 95b] Kfoury, A.J. and Wells, J.B., Addendum to new notions of reduction and non-semantic proofs of β-strong normalisation in typed λ-calculi, Boston University.

[Launchbury 93] Launchbury, J., A natural semantics of lazy evaluation, *ACM POPL 93*, 144-154, 1993.

[Lévy 80] Lévy, J.-J. Optimal reductions in the lambda calculus, in *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, J. Seldin and R. Hindley eds, Academic Press, 1980.

[NGV 94] Nederpelt, R.P., Geuvers, J.H., and de Vrijer, R.C., (eds) *Selected papers on Automath*, Studies in Logic and the Foundations of Mathematics, 133, North Holland, 1994.

[Reg 92] Regnier, L., Lambda calcul et réseaux, Thèse de doctorat de l'université Paris 7, 1992.

[Reg 94] Regnier, L., Une équivalence sur les lambda termes, *Theoretical Computer Sci. 126*, 281-292, 1994.

[SF 92] Sabry, A., and Felleisen, M., Reasoning about programs in continuation-passing style, *Proc. 1992 ACM Conf. LISP Funct. Program.*, 288-298, 1992.

[SP 93] Severi, P., and Poll, E., Pure Type Systems with Definitions, Computing Science Note 93/24, Eindhoven University of Technology, Department of Mathematics and Computing Science, 1993.

[Vid 89] Vidal, D., *Nouvelles notions de réduction en lambda calcul*, Thèse de doctorat, Université de Nancy 1, 1989.