An approximation of reductional equivalence*

Fairouz Kamareddine

Department of Computing Science
University of Glasgow
17 Lilybank Gardens
Glasgow G12 8QQ, Scotland
fairouz@dcs.gla.ac.uk

Roel Bloo and Rob Nederpelt

Mathematics and Computing Science,
Eindhoven University of Technology,
P.O.Box 513,
5600 MB Eindhoven, the Netherlands,
{bloo, wsinrpn}@win.tue.nl

Abstract

We define an equivalence relation on λ-terms called *shuffle-equivalence* which attempts to capture the notion of reductional equivalence on strongly normalizing terms. The shuffle-equivalence classes are shown to divide the classes of β-equal terms into smaller ones consisting of terms with similar reduction behaviour. We refine β-reduction from a relation on terms to a relation on shuffle-equivalence classes, called *shuffle-reduction*, and show that this refinement captures existing generalisations of β-reduction. Shuffle-reduction moreover, apart from allowing one to make more redexes visible and to contract these newly visible redexes, 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. This can benefit both programming languages and theorem provers since this flexibility and freedom in chosing reduction paths can be exploited to produce the shortest program evaluation paths and optimal proofs.

1 Introduction

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 reduction paths $A \twoheadrightarrow_\beta C$ and $B \twoheadrightarrow_\beta C$. It can be that both paths 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. We are interested in terms A and B that have a common reduct C such that the reductional behaviour from A to C is equivalent to the reductional behaviour from B to C. This is different from reduction paths. In this paper we try to give an approximation to the reductional equivalence between two strongly normalizing terms.

1.1 Making as many redexes as possible visible

Example 1.1 Consider the terms: $A \equiv (\lambda_\beta.\lambda_y.\lambda_f.fy)\alpha x$ and $B \equiv (\lambda_\beta.(\lambda_y.\lambda_f.fy)x)\alpha$. Both terms have the term $\lambda_f.fx$ 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 reducts in B:

1. $r_1 = (\lambda_\beta.(\lambda_y.\lambda_f.fy)x)\alpha$ which when contracted in B results in $(\lambda_y.\lambda_f.fy)x$

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

2. $r_2 = (\lambda_y.\lambda_f.fy)x$ which when contracted in B results in $(\lambda_\beta.\lambda_f.fx)\alpha$

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

1. $r_1' = (\lambda_\beta.\lambda_y.\lambda_f.fy)\alpha$ which when contracted in A results in $(\lambda_y.\lambda_f.fy)x$

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 enables us to see that in A (as in B), λ_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 $(\lambda_y.\lambda_f.fy)x$, the result of contracting the redex r_1' in A.

We could 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.

This has been noted by many researchers and hence rules like $(\lambda_x.N)PQ \to_\theta (\lambda_x.NQ)P$ have been introduced in many articles with different purposes [1, 4, 6, 8, 10, 11, 13, 15, 16, 18, 19, 20, 22] Such rules enable one to rewrite A so that both redexes become visible in A. Note that: $A \equiv (\lambda_\beta.\lambda_y.\lambda_f.fy)\alpha x \to_\theta (\lambda_\beta.(\lambda_y.\lambda_f.fy)x)\alpha \equiv B$.

These transformations are rather powerful in that they can group together terms with equal reductional behaviour. Let us give here this example:

Example 1.2 Take D, E, F, G with $D \equiv ((\lambda_f.(\lambda_x.\lambda_y.fxy)m)+)n$, $E \equiv (\lambda_f.(\lambda_x.(\lambda_y.fxy)n)m)+$, $F \equiv (((\lambda_f.\lambda_x.\lambda_y.fxy)+)m)n$ and $G \equiv (\lambda_f.((\lambda_x.\lambda_y.fxy)m)n)+$. In D, $(\lambda_f.(\lambda_x.\lambda_y.fxy)m)+$ and $(\lambda_x.\lambda_y.fxy)m$ are the visible redexes with a potential for $(\lambda_y.-)n$. In E, $(\lambda_f.(\lambda_x.(\lambda_y.fxy)n)m)+$, $(\lambda_x.(\lambda_y.fxy)n)m$ and $(\lambda_y.fxy)n$ are all redexes. F has the unique redex $(\lambda_f.\lambda_x.\lambda_y.fxy)+$. Moreover, $D =_\beta E =_\beta F =_\beta G$. Note that $F \to_\theta D \to_\theta G \to_\theta E$ and one sees that, by transforming F to D, an extra redex becomes visible and by transforming G to E the same happens, and that in E all redexes are visible.

Based on this, one wonders if one could classify terms according to their transformational relationship as in F, D, G, E. One has to be careful however:

Example 1.3 Let $E_1 \twoheadrightarrow_\theta E_2$ and $F_1 \twoheadrightarrow_\theta F_2$ (meaning E_1 and E_2 (respectively F_1 and F_2) can be transformed to have the same redexes as θ-reduction is Church Rosser and strongly normalizing). It is not the case that $(F_2\delta)E_1 \twoheadrightarrow_\theta (F_1\delta)E_2$ or $(F_1\delta)E_2 \twoheadrightarrow_\theta (F_2\delta)E_1$ despite the fact that $(F_1\delta)E_2$ and $(F_2\delta)E_1$ can be transformed to have the same redexes.

In this paper, we propose to define for each term M, a term $TS(M)$ (the θ-normal form of M) that makes as many redexes as possible visible. We consider the equivalence class of a term M to be $\{M' \mid TS(M) \equiv TS(M')\}$. As $\twoheadrightarrow_\theta$ is Church Rosser and strongly normalizing, then if $M_1 =_\theta M_2$ we get $TS(M_1) \equiv TS(M_2)$. We set out to show that the notion of equivalence classes modulo TS helps us to capture reductional equivalence on terms that are strongly normalizing.

1.2 Reductional Equivalence

In order to discuss reductional equivalence between terms, redexes will be extended so that a future redex like $(\lambda_y.-)x$ in A of Example 1.1 will be treated as a first class redex and will be contracted in A even before the originator $(\lambda_\beta.\lambda_y.\lambda_f.fy)\alpha$ has been contracted. Hence, with our extended notion of redexes and reduction we get in A:

$r_2' = (\lambda_y.\lambda_f.fy)x$ which when contracted in A results in $(\lambda_\beta.\lambda_f.fx)\alpha$

Note that r_2' is λ_y 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 of Example 1.1. 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.

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

Definition 1.4 *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 ($\sim_{\text{inf}}$) to the result of contracting the corresponding redex in B and vice versa.*
3. *Arguments of corresponding (extended) redexes are $\equiv$ or reductionally equivalent.*

Example 1.5 Note that $A \sim_{\text{inf}} B$ for A, B as in Example 1.1. Also $D \sim_{\text{inf}} E \sim_{\text{inf}} F \sim_{\text{inf}} G$ for D, E, F, G as in Example 1.2.

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 1.6 *It is undecidable whether two terms are reductionally equivalent.*

It seems interesting to find notions that are more easy to decide which approximate $\sim_{\text{inf}}$. One approach suggested by Henk Barendregt, in personal communications, 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 well behaved notions since the notions $\sim_n$ for $n \geq 2$ are not compatible. 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 $\sim_{\text{inf}}$ on strongly normalizing terms.

1.3 Shuffle-equivalence and shuffle-reduction

We settle in this paper for a new notion that we call *shuffle-equivalence*. Shuffle-equivalence is particularly related to θ and to the notion of term-reshuffling of [6]. First, the term-reshuffling of [6] is the θ-normal form. Moreover, shuffle-equivalence equates terms modulo term-reshuffling. Hence, if $A \to_\theta B$ or A term-reshuffles to B then A and B are shuffle-equivalent. There are many cases however where A and B are shuffle-equivalent without A and B being θ-related. Shuffle-equivalence looks for the class of all terms that have the same term-reshuffling.

In order to give an intuition why we take classes modulo term-reshuffling, observe that extended redexes can be shuffled to "classical" (i.e., non extended) redexes without losing reductional equivalence. This can be seen by our terms A and B of Example 1.1. The extended redex r_2' in A becomes classical in B. We call B the reshuffled version of A. We have seen that $A \sim_{\text{inf}} B$.

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.

It will be easier to understand what the operation TS does if we change the classical notation we have been using so far.

So we depart from those researchers who use θ, by using an extended form of θ based on the term-reshuffling of [6] (which turns out to be the θ-normal form). Furthermore, we depart from [6] by working with the equivalence classes modulo term-reshuffling rather than the terms themselves. Our motivation for doing so is that those equivalence classes capture as much as possible the notion of reductional equivalence. We define $[M]$, the class of M, to be $\{M'|TS(M) \equiv TS(M')\}$. Hence, D, E F and G above belong to the same class. All elements of $[M]$ are $=_\beta$ and have somehow the same redexes. We believe this is the closest *decidable* approximation that exists so far to the *undecidable* notion of reductional equivalence. In particular, we establish (see Fact 4.18) that on strongly normalizing terms, two shuffle-equivalent terms are reductionally equivalent, that a shuffle-equivalence class is decidable and that shuffle-equivalence does not coincide with reductional equivalence (which we conjecture to be undecidable).

Certainly, our notion of shuffle-equivalence captures (and is stronger than) already existing extensions of reductions. For example, $A \to_\theta B \Longrightarrow A$ and B are shuffle-equivalent. The converse is not true (see Example 3.10).

Once we have an approximation to reductional equivalence, we will extend the notion of β-reduction to apply to classes rather than terms. As classes capture already extensions of reductions such as θ, term reshuffling, etc., β-reduction over classes will capture all these notions. We say A *shuffle-reduces to* A' and we write $A \rightsquigarrow_\beta A'$ iff $\exists B \in [A] \exists B' \in [A']$ such that $B \to_\beta B'$. We show (see Lemma 4.3) that both $\to_\beta$ and the generalised reduction $\hookrightarrow_\beta$ of [3] are captured by $\rightsquigarrow_\beta$.

1.4 Comparison with previous work

The last few years have seen an explosion in new notions of reductions which can be used for various purposes. Attempts at generalising reduction can be summarized by four axioms:

(θ) $((\lambda_x.N)P)Q \to (\lambda_x.NQ)P$, (γ) $(\lambda_x.\lambda_y.N)P \to \lambda_y.(\lambda_x.N)P$,

(γ_C) $((\lambda_x.\lambda_y.N)P)Q \to (\lambda_y.(\lambda_x.N)P)Q$, (g) $((\lambda_x.\lambda_y.N)P)Q \to (\lambda_x.N[y := Q])P$

These rules attempt to make more redexes visible and to contract non-visible redexes. g is a combination of a θ-step with a g-step. γ_C e.g., makes sure that λ_y and Q form a redex even before the redex based on λ_x and P is contracted. By compatibility, γ implies γ_C. Moreover, $((\lambda_x.\lambda_y.N)P)Q \to_\theta (\lambda_x.(\lambda_y.N)Q)P$ and hence both θ and γ_C put λ adjacently next to its matching argument. θ moves the argument next to its matching λ whereas γ_C moves the λ next to its matching argument. θ can be equally applied to explicitly and implicitly typed systems. The transfer of γ or γ_C to explicitly typed systems is not straightforward however, since in these systems, the type of y may be affected by the reducible pair λ_x, P. E.g., it is fine to write $((\lambda_{x:*}.\lambda_{y:x}.y)z)u \to_\theta (\lambda_{x:*}.(\lambda_{y:x}.y)u)z$ but not to write $((\lambda_{x:*}.\lambda_{y:x}.y)z)u \to_{\gamma_C} (\lambda_{y:x}.(\lambda_{x:*}.y)z)u$. Hence, we study θ-like rules in this paper. Now, we discuss where generalised reduction has been used[1] (cf. [12, 8]).

[18] introduces the notion of a *premier redex* which is similar to the redex based on λ_y and Q above (which we call *generalised redex*). [19] 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 Strongly Normalizing (SN). [22] also introduces reductions similar to those of [19]. Furthermore, [10] uses θ (and other reductions) to show that typability in ML is equivalent to acyclic semi-unification.

[1] We are grateful to Joe Wells for enlightening discussions on this subject.

[20] uses a reduction which has some common themes with θ. Nederpelt's thesis in [17] and [4] use θ whereas [13] uses γ to reduce the problem of β-strong normalization to the problem of weak normalization (WN) for related reductions. [11] uses θ and γ to reduce typability in the rank-2 restriction of the 2nd order λ-calculus to the problem of acyclic semi-unification. [15, 23, 21, 14] use related reductions to reduce SN to WN and [9] uses similar notions in SN proofs. [1] uses θ (called "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. [6] uses a more extended version of θ (called *term-reshuffling*) and of g (called *generalised reduction*) where Q and N are not only separated by the redex $(\lambda_x.N)P$ but by many redexes (ordinary and generalised).

After looking carefully at all these attempts, we realised that none of the extensions of reductions introduced so far can play as general a role as approximating reductional equivalence. Of course all these notions have influenced our choice of the relation which we consider to best approximate reductional equivalence (i.e. shuffle equivalence).

2 The formal machinery

The classical notation cannot extend the notion of redexes or enable reshuffling in an easy way. *Item notation* however can ([7] discusses various advantages of this notation). Let V be an infinite collection of variables over which $x, y, z, \ldots$ range. In item notation, terms of the λ-calculus are: $\mathcal{T} ::= V \,|\, (\mathcal{T}\delta)\mathcal{T} \,|\, (\lambda_V)\mathcal{T}$. We take $A, B, C, \ldots$ to range over $\mathcal{T}$. We call $(A\delta)$ a **δ-item**, A the **body** of the item and $(A\delta)B$ means apply B to A (note the order). (λ_x) is called a **λ-item**. A redex starts with a δ-item (i.e., $(A\delta)$) next to a λ-item (i.e., (λ_x)).

Example 2.1 $A \equiv (u\delta)(w\delta)(\lambda_x)(v\delta)(\lambda_y)(\lambda_z)(z\delta)(y\delta)x$, by moving the item $(u\delta)$ to the right until it is next to its matching partner (λ_z), reshuffles to $TS(A) \equiv (w\delta)(\lambda_x)(v\delta)(\lambda_y)(u\delta)(\lambda_z)(z\delta)(y\delta)x$ Such a reshuffling in item notation is clearer than reshuffling in classical notation where the term $((\lambda_x.(\lambda_y.\lambda_z.xyz)v)w)u$ is reshuffled to $(\lambda_x.(\lambda_y.(\lambda_z.xyz)u)v)w$.

Note furthermore that the shuffling is not problematic because we use the Barendregt Convention (see below) 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.

Example 2.2 D of Example 1.2 reads $(n\delta)(+\delta)(\lambda_f)(m\delta)(\lambda_x)(\lambda_y)(y\delta)(x\delta)f$ in item notation. Here, the two (classical) redexes correspond to *$\delta\lambda$-pairs* followed by the body of the abstraction as follows: $(\lambda_f.(\lambda_x.\lambda_y.fxy)m)+$ corresponds to $(+\delta)(\lambda_f)(m\delta)(\lambda_x)(\lambda_y)(y\delta)(x\delta)f$ and $(\lambda_x.\lambda_y.fxy)m$ corresponds to $(m\delta)(\lambda_x)(\lambda_y)(y\delta)(x\delta)f$. Note that the δ-item $(+\delta)$ and the λ-item $((\lambda_f)$ are adjacent, showing the presence of a redex. Similarly, note the adjacency of $(m\delta)$ and (λ_x).

The third redex of D is obtained by matching δ and λ-items. $(\lambda_y.fxy)n$ is visible as it corresponds to the matching $(n\delta)(\lambda_y)$ where $(n\delta)$ and (λ_y) are separated by the segment $(+\delta)(\lambda_f)(m\delta)(\lambda_x)$ which has the bracketing structure $[\,][\,]$ (see Figure 1 which represents D). We will use obvious notions throughout like **partner**, **bachelor**, **match**, etc. In Figure 1, $(+\delta)$ and (λ_f) match or are partnered. So are the items $(n\delta)$ and (λ_y). $(y\delta)$ and $(x\delta)$ on the other hand are bachelor. $(+\delta)(\lambda_f)$ is called a **$\delta\lambda$-pair** and $(n\delta)(\lambda_y)$ is a **$\delta\lambda$-couple**. Term reshuffling amounts to moving δ-items to occur next to their matching λ-items. Hence D of Example 1.2 is reshuffled to $(+\delta)(\lambda_f)(m\delta)(\lambda_x)(n\delta)(\lambda_y)(y\delta)(x\delta)f$ and Figure 1 changes to Figure 2 (which represents E). Furthermore, Figures 3 and 4 (which represent F and G) also change to Figure 2.

According to our shuffle-equivalence, D, E, F and G belong to the same class and are $\sim_{\text{inf}}$.

Each term A is the concatenation of zero or more items and a variable: $A \equiv s_1 s_2 \cdots s_n x$ where $x \in V$. These items $s_1, s_2, \ldots, s_n$ are called the **main items** of A, x is called the **heart** of A,

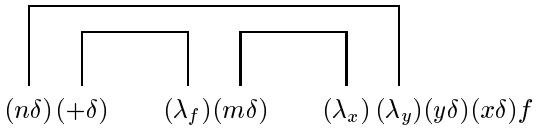

Figure 1: (Extended) redexes in item notation: D

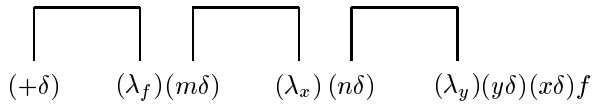

Figure 2: The reshuffled term D in item notation: E

notation $\heartsuit(A)$. We use $s, s_1, s_i, \ldots$ to range over items. 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 **$\delta\lambda$-segment** is a δ-item immediately followed by a λ-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 $\mathtt{weight}(\overline{s}x) = \mathtt{weight}(\overline{s})$ for $x \in V$.

In reduction, the *matching* of the δ and the λ in question is the important thing. **Well-balanced segments (w-b)** separate matching δ and λ-items. They are: The empty segment $\emptyset$, if $\overline{s}$ is w-b then $(A\delta)\overline{s}(\lambda_x)$ is w-b, the concatenation of w-b segments is w-b. Hence in Figures 1, 2, 3 and 4 all segments that occur under a hat are w-b.

Lemma 2.3 *Every term has one of the following three forms:*

- $(\lambda_x)B$
- $(A_1\delta)\cdots(A_n\delta)x$, *where* $x \in V$
- $(A_1\delta)\cdots(A_n\delta)(B\delta)(\lambda_x)D$, *where* $n \geq 0$

Definition 2.4 *We say that two terms A and B are semantically equivalent iff $A =_\beta B$.*

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. Note that in item notation, the scope of a λ-item is anything to the right of it. 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 usual Barendregt variable convention BC and usual definition of compatibility (see [2]). We say that A is strongly normalizing with respect to a reduction relation $\rightarrow$ (written $\mathrm{SN}_\rightarrow(A)$) iff every $\rightarrow$-reduction path starting at A terminates.

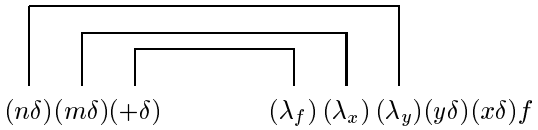

Figure 3: More extended redexes: F

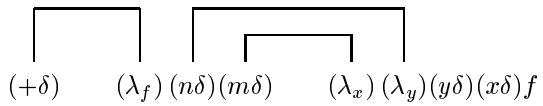

Figure 4: The term G in item notation

3 Shuffle-equivalence

In this section we follow [6] and rewrite terms so that all the newly visible redexes can be subject to $\rightarrow_\beta$. We shall show that this term rewriting function is correct in the sense that $A =_\beta TS(A)$, i.e., A and $TS(A)$ are semantically equivalent (see Lemma 3.5). Furthermore, we show that shuffle-equivalence is stronger than $\rightarrow_\theta$ and than the term-reshuffling of [6] (Lemma 3.9 and Example 3.10) and that shuffle-equivalence classes are decidable (Lemma 3.8). In Section 4 (Lemma 4.17), we show that shuffle-equivalence is indeed a (decidable) approximation of reductional equivalence.

Definition 3.1 *The reshuffling function TS is defined such that:*

$$\begin{array}{lll} TS((\lambda_x)C) & =_{df} & (\lambda_x)TS(C) \\ TS((B_1\delta)\cdots(B_n\delta)x) & =_{df} & (TS(B_1)\delta)\cdots(TS(B_n)\delta)x \qquad \textit{if } x \in V \\ TS((B_1\delta)\cdots(B_n\delta)(C\delta)(\lambda_x)E) & =_{df} & (TS(C)\delta)(\lambda_x)TS((B_1\delta)\cdots(B_n\delta)E) \end{array}$$

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

As an example, the term $(\lambda_x)(w\delta)\ (\overset{+}{x}\delta)(\overset{\bullet}{y}\delta)\ (\overset{\bullet}{\lambda_v})\ (\overset{\times}{v}\delta)(\overset{''}{x}\delta)(\overset{''}{\lambda_w})(\overset{\times}{\lambda_t})(\overset{+}{\lambda_s})\ (s\delta)t$

will be reshuffled to the term $(\lambda_x)\ (\overset{\bullet}{y}\delta)\ (\overset{\bullet}{\lambda_v})\ (\overset{''}{x}\delta)(\overset{''}{\lambda_w})(\overset{\times}{v}\delta)(\overset{\times}{\lambda_t})(\overset{+}{x}\delta)(\overset{+}{\lambda_s})\ (w\delta)(s\delta)t$

It can be seen that for any A, $TS(A)$ is of the form $\overline{s_0}\,\overline{s_1}x$ where $x \in V$, $\overline{s_1}$ 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 $\delta\lambda$-segment or a bachelor main λ-item. Now, let us show some properties of TS.

Lemma 3.2 (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 3.3

1.For all terms 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 3.4 *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 $\delta\lambda$-segments and bachelor main λ-items.*

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

- $A \equiv (\lambda_x)C$, *use IH on C.*
- $A \equiv (B_1\delta)\cdots(B_n\delta)x$, $x \in V$. *Then $\overline{s_0}$ is empty.*
- $A \equiv (B_1\delta)\cdots(B_n\delta)(C\delta)(\lambda_x)E$.

  Then $TS(A) \equiv (TS(C)\delta)(\lambda_x)TS((B_1\delta)\cdots(B_n\delta)E)$. 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 3.5 *For all terms 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.
2. Induction on the number of symbols in A, using 1.
3. By induction on the number of symbols in A. If $A \equiv (A_1\delta)\cdots(A_n\delta)x$ where $x \in V$ or $A \equiv (\lambda_x)A_2$ then use the induction hypothesis. If $A \equiv (A_1\delta)\cdots(A_n\delta)(B\delta)(\lambda_x)D$ then

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

□

Corollary 3.6 *For all terms A, B, $TS(A) =_\beta TS(B)$ iff $A =_\beta B$.* □

Definition 3.7 *(Shuffle-class and shuffle-equivalence) For a term A, we define $[A]$, the shuffle class of A, to be $\{B \mid TS(A) \equiv TS(B)\}$. We say that A and B are shuffle-equivalent if $[A] = [B]$.*

Lemma 3.8 *For any A, B, it is decidable whether $A \in [B]$. Moreover, if $A \in [B]$ then $A =_\beta B$.*
Proof *Follows from Lemma 3.2 and Corollary 3.6.* □

The following shows that shuffle-equivalence contains $\to_\theta$ and term-reshuffling of [6].

Lemma 3.9 *If $A \to_\theta B$ or A term-reshuffles to B in the sense of [6] then A and B are shuffle-equivalent.*
Proof: *TS formalizes term-reshuffling of [6] and the latter captures $\to_\theta$.* □

The other way round does not always hold however:

Example 3.10 $A \equiv (A_1\delta)(A_2\delta)(A_3\delta)(\lambda_x)(\lambda_y)(\lambda_z)A_4$ and $B \equiv (A_2\delta)(A_3\delta)(\lambda_x)(\lambda_y)(A_1\delta)(\lambda_z)A_4$ are not related by $\to_\theta$ ($\exists C \not\equiv A$ however such that $B \to_\theta C$ and $A \twoheadrightarrow_\theta C$). Moreover, neither A is $TS(B)$ nor B is $TS(A)$, but $TS(A) \equiv TS(B)$.

4 Shuffle-reduction

In this section, we introduce shuffle-reduction $\rightsquigarrow_\beta$, show that it is Church-Rosser and that shuffle-equivalence preserves reduction in the sense that if $A \to_\beta B$ then $A \rightsquigarrow_\beta B$. We show that shuffle-equivalence implies reductional equivalence (on SN terms) and that shuffle-reduction on classes makes more redexes visible and allows for smaller terms during reductions.

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

- *One-step shuffle-reduction $\rightsquigarrow_\beta$ is the least compatible relation generated by:*

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

  Many-step shuffle-reduction $\rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta$ is the reflexive and transitive closure of $\rightsquigarrow_\beta$ and $\approx_\beta$ is the least equivalence relation generated by $\rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta$.

- *An extended redex starts with the δ-item of a* **$\delta\lambda$-couple** *(i.e. is of the form $(C\delta)\overline{s}(\lambda_x)A$ where $\overline{s}$ is well-balanced).*

- $\hookrightarrow_\beta$ *is the least compatible relation generated by* $(B_1\delta)\overline{s}(\lambda_x)B_2 \hookrightarrow_\beta \overline{s}(B_2[x := B_1])$ *for* $\overline{s}$ *well-balanced, that is,* $\hookrightarrow_\beta$*-reduction contracts an (extended) redex.* $\hookrightarrow\!\!\!\!\rightarrow_\beta$ *is the reflexive and transitive closure of* $\hookrightarrow_\beta$ *and* $\sim_\beta$ *the least equivalence relation closed under* $\hookrightarrow\!\!\!\!\rightarrow_\beta$*.*

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

$\hookrightarrow\!\!\!\!\rightarrow_\beta$ has been used in [3, 6] where it was shown to be more general than other generalised notions of reduction introduced in the literature (such as (g) of Subsection 1.4). Here, we show that $\rightsquigarrow\!\!\!\!\rightarrow_\beta$ is more general than $\hookrightarrow\!\!\!\!\rightarrow_\beta$. Shuffle-reduction captures classical β-reduction and the generalised reduction of [3, 6]:

Lemma 4.3 $\rightarrow_\beta \subset \hookrightarrow_\beta \subset \rightsquigarrow_\beta$.

Proof: $(A\delta)(\lambda_x)C \equiv (A\delta)\emptyset(\lambda_x)C \hookrightarrow_\beta \emptyset C[x := A] \equiv C[x := A]$. *Also, by Lemma 3.3, we know that* $(A\delta)\overline{s}(\lambda_x)C \in [\overline{s}(A\delta)(\lambda_x)C]$*, and since* $\overline{s}(A\delta)(\lambda_x)C \rightarrow_\beta \overline{s}C[x := A]$ *we have* $(A\delta)\overline{s}(\lambda_x)C \rightsquigarrow_\beta \overline{s}C[x := A]$*. It is easy to show that these inclusions are strict.* □

Corollary 4.4 $\twoheadrightarrow_\beta \subset \hookrightarrow\!\!\!\!\rightarrow_\beta \subset \rightsquigarrow\!\!\!\!\rightarrow_\beta$.

Remark 4.5 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 $A \equiv ((\lambda_u)(\lambda_v)v\delta)(\lambda_x)(w\delta)(w\delta)x$ and $B \equiv (w\delta)(\lambda_u)w$. Then $A \rightsquigarrow_\beta (w\delta)(w\delta)(\lambda_u)(\lambda_v)v \rightsquigarrow_\beta B$. But $[A]$ has three elements, namely: A, $(w\delta)((\lambda_u)(\lambda_v)v\delta)(\lambda_x)(w\delta)x$ and $(w\delta)(w\delta)((\lambda_u)(\lambda_v)v\delta)(\lambda_x)x$, $[B] = \{B\}$ and if $A' \in [A]$ then the only $\rightarrow_\beta$ reduct of A' is $(w\delta)(w\delta)(\lambda_u)(\lambda_v)v$, which doesn't $\rightarrow_\beta$-reduce to B. In Lemma 4.13 however, we show that there is a correspondence between $\rightsquigarrow\!\!\!\!\rightarrow_\beta$ on classes and $\twoheadrightarrow_\beta$ on terms.

Lemma 4.6 $TS(A) \hookrightarrow_\beta B$ *iff* $TS(A) \rightarrow_\beta B$.

Proof: *This is a direct consequence of Lemma 3.4* □

Lemma 4.7 *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* $\exists A_1 \in [A]\exists B_1 \in [B][A_1 \rightarrow_\beta B_1]$*. Let* $A', B' \in [A], [B]$ *respectively. Then* $A_1 \in [A']$*,* $B_1 \in [B']$*,* $A_1 \rightarrow_\beta B_1$*. So* $A' \rightsquigarrow_\beta B'$. □

Corollary 4.8 $A \rightsquigarrow_\beta B \Leftrightarrow TS(A) \rightsquigarrow_\beta TS(B)$

Remark 4.9 Note that $A \hookrightarrow_\beta B \not\Rightarrow TS(A) \hookrightarrow\!\!\!\!\rightarrow_\beta TS(B)$ nor do we have $A \rightarrow_\beta B \Rightarrow TS(A) \rightarrow \rightarrow_\beta TS(B)$. 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 $TS(A) \equiv A \not\twoheadrightarrow_\beta$ nor $\not\twoheadrightarrow_\beta TS(B) \equiv (y\delta)(z\lambda_u)(y\delta)(z\lambda_v)v$.

The following lemma helps establish that $\rightsquigarrow_\beta$ is Church-Rosser:

Lemma 4.10 *If* $A \rightsquigarrow_\beta B$ *then* $A =_\beta B$.

Proof: *Say* $A' \in [A]$*,* $B' \in [B]$*,* $A' \rightarrow_\beta B'$*. Then by lemma 3.5:* $A =_\beta TS(A) \equiv TS(A') =_\beta A' =_\beta B' =_\beta TS(B') \equiv TS(B) =_\beta B$. □

Corollary 4.11

1. If $A \rightsquigarrow\!\!\!\!\rightarrow_\beta B$ *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 4.12 *(The general Church Rosser theorem for $\rightsquigarrow\!\!\!\!\rightarrow_\beta$)*
If $A \rightsquigarrow\!\!\!\!\rightarrow_\beta B$ and $A \rightsquigarrow\!\!\!\!\rightarrow_\beta C$, then there exists D such that $B \rightsquigarrow\!\!\!\!\rightarrow_\beta D$ and $C \rightsquigarrow\!\!\!\!\rightarrow_\beta D$.

Proof: *As $A \rightsquigarrow\!\!\!\!\rightarrow_\beta B$ and $A \rightsquigarrow\!\!\!\!\rightarrow_\beta C$ then by Corollary 4.11, $A =_\beta B$ 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 $M \rightsquigarrow\!\!\!\!\rightarrow_\beta N$. Hence we are done.* □

As we noted in Remark 4.9, we can have $TS(C) \rightarrow_\beta D$ where $D \not\equiv TS(D)$. Nevertheless, term reshuffling preserves β-reduction. This is a generalisation of the result in [6] to equivalence classes.

Lemma 4.13 *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:*

$$\begin{array}{ccc} A & \longrightarrow\!\!\!\!\!\rightsquigarrow_\beta & B \\ \Big| & & \vdots \\ TS(A) & - - - - - \rightarrow_\beta & B' \in [B] \end{array}$$

Proof: *We prove by induction on the structure of A' that if $A' \rightarrow_\beta B' \in [B]$, then for some B'', $TS(A') \rightarrow_\beta B'' \in [B]$. The compatibility cases are easy, distinguish cases according to the definition of TS. If $A' \equiv (C\delta)(\lambda_x)E$ and $B' \equiv E[x := C] \in [B]$ then $TS(A') \equiv (TS(C)\delta)(\lambda_x)TS(E) \rightarrow_\beta TS(E)[x := TS(C)]$ and by Lemma 3.5, $TS(TS(E)[x := TS(C)]) \equiv TS(E[x := C]) \in [B]$.*

□

Corollary 4.14 *If $A \rightsquigarrow\!\!\!\!\rightarrow_\beta B$ then there exist $A_0, A_1, \ldots, A_n$ such that*
$[(A \equiv A_0) \wedge (TS(A_0) \rightarrow_\beta A_1) \wedge (TS(A_1) \rightarrow_\beta A_2) \wedge \cdots \wedge (TS(A_{n-1}) \rightarrow_\beta A_n \in [B])]$

Proof: *By induction on $\rightsquigarrow\!\!\!\!\rightarrow_\beta$.* □

Now we show that shuffle-equivalence preserves strong normalization:

Lemma 4.15 *Let $A \in SN_{\rightsquigarrow_\beta}$. Then for all $A' \in [A]$, $A' \in SN_{\rightsquigarrow_\beta}$.*

Proof: *$\forall B, A' \rightsquigarrow_\beta B$ implies $A \rightsquigarrow_\beta B$ by Lemma 4.7 Hence, A' must be $\in SN_{\rightsquigarrow_\beta}$.* □

Moreover, shuffle-reduction preserves β-strong normalization:

Lemma 4.16 *$A \in SN_{\rightsquigarrow_\beta} \Longleftrightarrow A \in SN_{\rightarrow_\beta}$.*

Proof: *As $\rightarrow_\beta \subset \rightsquigarrow_\beta$, $\Longrightarrow$ is immediate. $\Longleftarrow$ is by using a result of [19] which states that the length of the longest reduction of a term is invariant by σ-equivalence and by noting that shuffle-reduction is isomorphic to β-reduction modulo σ-equivalence. Another way of showing $\Longleftarrow$ is by induction on $(d(A), A)$ ordered by the lexicographic product ordering where $d(A)$ denotes the maximum length of a β-reduction of A to its β-normal form. Case $d(A) = 0$ is trivial. Case $d(A) \neq 0$, we use induction on the structure of A as given in Lemma 2.3. The interesting case when $A \equiv (A_1\delta)\cdots(A_n\delta)(B\delta)(\lambda_x)D$, for $n \geq 0$ can be done by noting that $(A_1\delta)\cdots(A_n\delta)(B\delta)(\lambda_x)D \rightarrow_\beta (A_1\delta)\cdots(A_n\delta)\{D[x := B]\}$ which satisfies IH. Another way is to follow the lines of [5].* □

Now we show that shuffle-equivalence for SN terms implies reductional equivalence:

Lemma 4.17 *Let $A \in SN_{\rightsquigarrow_\beta}$. Then for all $A' \in [A]$, $A' \sim_{\text{inf}} A$.*

Proof: *It is sufficient to show that $(B\delta)\overline{s}C \sim_{\text{inf}} \overline{s}(B\delta)C$ 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 normal form 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:

Fact 4.18 *The following holds:*

1. *Let $A \in SN_{\leadsto_\beta}$. $TS(A) \equiv TS(B) \Longrightarrow A \sim_{\text{inf}} B$ (Lemma 4.17).*
2. *$TS(A) \equiv TS(B) \not\Longrightarrow A \sim_{\text{inf}} B$ (Example 4.19).*
3. *$A \sim_{\text{inf}} B \not\Longrightarrow TS(A) \equiv TS(B)$ (Example 4.20 below).*
4. *$TS(A) \equiv TS(B)$ is decidable (Lemma 3.2).*
5. *$A \sim_{\text{inf}} B$ is not decidable (Conjecture 1.6).*
6. *Let $A \in SN_{\leadsto_\beta}$. Then for all $A' \in [A]$, $A' \in SN_{\leadsto_\beta}$.*

Example 4.19 Take the terms A and B where $A \equiv (a\delta)(b\delta)(\lambda_x)(\lambda_y)((\lambda_z)(z\delta)z\delta)(\lambda_z)(z\delta)z$ and $B \equiv (b\delta)(\lambda_x)(a\delta)(\lambda_y)((\lambda_z)(z\delta)z\delta)(\lambda_z)(z\delta)z$. These terms read in classical notation $(\lambda_x.\lambda_y.\mathbf{\Omega})ba$ respectively $(\lambda_x.(\lambda_y.\mathbf{\Omega})a)b$ where $\Omega \equiv (\lambda_z.zz)(\lambda_z.zz)$. Now, $TS(A) \equiv TS(B)$ but $A \not\sim_{\text{inf}} B$ since contracting Ω will not result in syntactically equivalent terms. This shows that one cannot drop the assumption that A is strongly normalizing.

Example 4.20 *Let $A \equiv ((a\delta)(\lambda_x)x\delta)(\lambda_y)y$ and $B \equiv (a\delta)(\lambda_x)(x\delta)(\lambda_y)y$. $A \sim_{\text{inf}} B$ but $TS(A) \not\equiv TS(B)$. The same holds for the terms $(a\delta)(\lambda_y)(y\delta)y$ and $(a\delta)(\lambda_y)(y\delta)a$.*

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 4.21 Let $M \equiv (\lambda_x.\lambda_y.y(Cxx\cdots x))B(\lambda_z.u)$ where B is a BIG term. Then $M \to_\beta (\lambda_y.y(CBB\cdots B))(\lambda_z.u) \to_\beta (\lambda_z.u)(CBB\cdots B) \to_\beta u$ and u is in normal form. Now the first and second reducts both contain $CBB\cdots B$, so they are very long terms. Shuffle reduction allows us to reduce M in the following way: $TS(M) \equiv (\lambda_x.(\lambda_y.y(Cxx\cdots x))\lambda_z.u)B \to_\beta (\lambda_x.(\lambda_z.u)(Cxx\cdots x))B \to_\beta (\lambda_x.u)B \to_\beta u$, 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 Conclusion

Many new notions of reductions in the λ-calculus have recently been proposed. Most of these notions reduce one term to another with the the same *reductional behaviour*. The only difference is that some potential future redexes are visible in the new term when they were not in the old.

This paper unifies all this work by looking for the class of reductionally equivalent terms. Such a class is conjectured to be undecidable but a decidable approximation of it is found which does indeed capture the existing new notions of reduction. We believe that our shuffle-equivalence is the closest decidable approximation to reductional equivalence. Moreover, if $A \to B$ where $\to$ is a new notion of reduction given by the existing accounts (such as those of Moggi, Ariola etal, Regnier, Kfoury and Wells, Vidal, Kamareddine and Nederpelt, etc) then A and B belong to the same shuffle-equivalence class under our apoproach of this paper. Furthermore, shuffle-equivalence classes partition β-equivalence classes into smaller parts.

In addition to the many notions of reduction where $A \to B$ implies A and B have the same reductional behaviour, we find many extensions of β-reduction $\to_e$ where if $A \to_\beta B$ then $A \to_e B$

and where the equivalence relation generated by $\rightarrow_e$ is just β-equality. These extensions make more redexes visible and hence allow for more flexibility in reducing a term.

We propose a generalisation of these extensions which we call shuffle-reduction. Shuffle-reduction does indeed accommodate the existing accounts and achieve their goals. In particular, we show that using shuffle-reduction we indeed may avoid size explosion without the cost of a longer reduction path, that it has the Church-Rosser property, and that the equivalence relation generated by shuffle-reduction is just β-equality.

We used the item-notation to give a clearer description of shuffle-equivalence and shuffle-reduction. We think that the item-notation is a good candidate for answering the two questions posed in the conclusions of [19] concerning the existence of a syntax for terms realising shuffle-equivalence (which Regnier [19] calls σ-equivalence).

References

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

[2] H. Barendregt. λ-calculi with types. *Handbook of Logic in Computer Science*, II, 1992.

[3] R. Bloo, F. Kamareddine, and R. Nederpelt. The Barendregt Cube with Definitions and Generalised Reduction. *Information and Computation*, 126 (2):123–143, 1996.

[4] P. de Groote. The conservation theorem revisited. *Int'l Conf. Typed Lambda Calculi and Applications LNCS*, 664, 1993.

[5] F. Kamareddine. A reduction relation for which postponement of k-contractions, conservation and preservation of strong normalisation holds. *Submitted*.

[6] F. Kamareddine and R. Nederpelt. Generalising reduction in the λ-calculus. *Journal of Functional Programming*, 5(4):637–651, 1995.

[7] F. Kamareddine and R. Nederpelt. A useful λ-notation. *Theoretical Computer Science*, 155:85–109, 1996.

[8] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. Proceedings of PLILP'96. *LNCS*, 1140:378–392, 1996. To appear.

[9] M. Karr. Delayability in proofs of strong normalizability in the typed λ-calculus. *Mathematical Foundations of Computer Software, LNCS*, 185, 1985.

[10] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. An analysis of ML typability. *ACM*, 41(2):368–398, 1994.

[11] A.J. Kfoury and J.B. Wells. 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.

[12] A.J. Kfoury and J.B. Wells. Addendum to new notions of reduction and non-semantic proofs of β-strong normalisation in typed λ-calculi. Technical report, Boston University, 1995.

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

[14] Z. Khasidashvili. The longest perpetual reductions in orthogonal expression reduction systems. *Proc. of the* 3^{rd} *International Conference on Logical Foundations of Computer Science, Logic at St Petersburg*, 813, 1994.

[15] J. W. Klop. Combinatory Reduction Systems. *Mathematical Center Tracts*, 27, 1980.

[16] E. Moggi. Computational λ-calculus and monads. *LICS'89*, 89.

[17] R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer. *Selected papers on Automath*. North-Holland, Amsterdam, 1994.

[18] L. Regnier. *Lambda calcul et réseaux*. PhD thesis, Paris 7, 1992.

[19] L. Regnier. Une équivalence sur les lambda termes. *Theoretical Computer Science*, 126:281–292, 1994.

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

[21] M. Sϕrensen. Strong normalisation from weak normalisation in typed λ-calculi. *Information and Computation*. To appear.

[22] D. Vidal. *Nouvelles notions de réduction en lambda calcul*. PhD thesis, Université de Nancy 1, 1989.

[23] H. Xi. On weak and strong normalisations. Technical Report 96-187, Carnegie Mellon University, 1996.