Characterizing lambda-terms with equal reduction behavior

Technische Universiteit Eindhoven
Department of Mathematics and Computing Science

Characterizing λ-terms with equal reduction behavior

by

Fairouz Kamareddine and Roel Bloo and Rob Nederpelt

00/16

ISSN 0926-4515

Computing Science Reports 00/16
Eindhoven, October 2000

Characterizing λ-terms with equal reduction behavior*

Fairouz Kamareddine[1] and Roel Bloo[2] and Rob Nederpelt[2]

[1] Computing and Electrical Engineering, Heriot-Watt Univ., Riccarton, Edinburgh EH14 4AS, Scotland, `fairouz@cee.hw.ac.uk`

[2] Mathematics and Computing Science, Eindhoven Univ. of Technology, P.O.Box 513, 5600 MB Eindhoven, the Netherlands, `{c.j.bloo, r.p.nederpelt}@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 aim of reductional equivalence is to characterize the evaluation behavior of programs. The shuffle-equivalence classes are shown to divide the classes of β-equal strongly normalising terms (programs which lead to the same final value/output) into smaller ones consisting of terms with similar evaluation behavior. 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 allows one to make more redexes visible and to contract these newly visible redexes. Moreover, it allows 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 choosing reduction paths can be exploited to produce the shortest program evaluation paths and optimal proofs.

Keywords: specification, semantics and rewriting of programs

1 Introduction

The λ-calculus plays a major role in the semantics of programming languages through its mechanisms for modeling evaluation strategies (e.g., call by name, call by value, etc.). Due to this basic role, the λ-calculus must be informative not only of the final value of the program (the normal form of the λ-term representing the program), but also of the consecutive values before the final value is reached. In particular, if we have two programs P_1 and P_2 that return the same final value, we want to know if these programs have equivalent evaluation paths in the sense that each evaluation path from P_1 to the final value (going through all the intermediate programs), corresponds (in a strong evaluation sense) to an evaluation path from P_2 to the final value, and vice versa. This will mean that P_1 and P_2 are equivalent programs even though they are written differently. Each intermediate value a_1 along the evaluation path from one of these programs to the final value corresponds to a unique intermediate value a_2 along the evaluation path of the other program to the final value, and the number of evaluation steps to reach a_1 from the first program is equal to the number of evaluation steps to reach a_2 from the second program. Of course this does not constitute a formal definition of what we call *reductional equivalence*. Reductional equivalence is difficult to define and is also undecidable. In this paper, we attempt to formally define a decidable approximation of reductional equivalence which we call *shuffle equivalence*.

For this, we attempt to observe the reduction behavior of λ-terms (which represent programs). 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 behavior of A and B can be very different, as is the case if $A \equiv \mathbf{K I \Omega}$

* We are grateful for enlightening discussions and useful feedback and comments received from Henk Barendregt, Twan Laan and Joe Wells.

and $B \equiv \mathbf{KII}$, where $\mathbf{K}$ and $\mathbf{I}$ are the well known combinators with $\mathbf{K}xy \to_\beta x$ and $\mathbf{I}x \to_\beta x$, and $\mathbf{\Omega} \equiv (\lambda x.xx)(\lambda x.xx)$. Both A and B have the common reduct $\mathbf{I}$ but although B is *strongly normalizing* (i.e., it allows no infinite reduction path), A is not ($\mathbf{\Omega}$ infinitely reduces to itself). Our problem is to characterize terms with *equivalent reduction behavior* in the sense that reduction paths from one can be mapped to reduction paths from the other. This is different from studying the behavior of programs using bisimulation. We are concerned with the syntactic analysis of programs written as λ-terms.

In order to conduct our syntactic study of reductional equivalence of programs written as λ-terms, we need to use a λ-notation that enables us to detect more redexes in a term than can be visible in the known classical notation λ-calculus [3]. For this purpose, we will use the item notation whose advantages are discussed in [7]. [6] explains that it is not feasible to syntactically describe generalised reduction in classical notation and therefore item notation is indispensable for our study of reductional equivalence which depends on extended and generalised redexes.

In Section 2 we introduce informally reductional equivalence, potential future redexes and the motivation for the *preferred* reductionally equivalent term TS(A) of a λ-term A. TS(A) makes visible as many redexes of A as possible.

In Section 3 we introduce what is needed of the item notation and other formal machinery that will enable us to syntactically describe potential future redexes and will help us approximate reductional equivalence.

In Section 4 we introduce shuffle equivalence which we will use to approximate reductional equivalence. An important phenomenon that results from this definition is the ability to partition terms elegantly into parts which are informative as to where redexes occur.

In Section 5 we extend the usual β-reduction $\to_\beta$ on λ-terms to $\leadsto_\beta$ on classes of terms modulo shuffle equivalence. We establish that our extended reduction $\leadsto_\beta$ is a generalisation of $\to_\beta$ which enables one to have more freedom in choosing the reduction path of a term. In fact, a clever reduction algorithm might know how to choose the shortest reduction path and it is likely that this reduction path does not exist when usual reduction is used. We also establish that shuffle equivalence is a decidable approximation of reductional equivalence.

In Section 6 we compare our work with existing literature on generalising reduction and conclude that our notion of reduction in this paper subsumes all of this reported previous work.

2 Informal introduction to shuffle and reductional equivalence

2.1 Making as many redexes as possible visible

Example 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. Yet, our paper will establish that the terms* A *and* B *are reductionally equivalent. We will do so by extending the notion of redexes and by considering for each* E, *the preferred reductionally equivalent term* TS(E) *which makes visible as many redexes of* E *as possible. First, note that initially, one will only see the following redexes of* B:

- $r_1 = (\lambda_\beta.(\lambda_y.\lambda_f.fy)x)\alpha$. *Observe that* $B \xrightarrow{r_1} (\lambda_y.\lambda_f.fy)x$.
- $r_2 = (\lambda_y.\lambda_f.fy)x$. *Observe that* $B \xrightarrow{r_2} (\lambda_\beta.\lambda_f.fx)\alpha$.

In A, *the only obvious redex is:* $r'_1 = (\lambda_\beta.\lambda_y.\lambda_f.fy)\alpha$. *Observe that* $A \xrightarrow{r'_1} (\lambda_y.\lambda_f.fy)x$.
Note that r_1 *in* B *and* r'_1 *in* A *are both based on the redex* $(\lambda_\beta.-)\alpha$ *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 [1,5,8,9,11,12,14,16,17,19–21,23]. 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 behavior. Let us give here this example:

Example 2 *Consider E_1, E_2, E_3, E_4 as follows:*
$E_1 \equiv (((\lambda_f.\lambda_x.\lambda_y.fxy)+)m)n,$ $\quad E_2 \equiv ((\lambda_f.(\lambda_x.\lambda_y.fxy)m)+)n,$
$E_3 \equiv (\lambda_f.((\lambda_x.\lambda_y.fxy)m)n)+,$ $\quad E_4 \equiv (\lambda_f.(\lambda_x.(\lambda_y.fxy)n)m)+.$

Note that $E_1 =_\beta E_2 =_\beta E_3 =_\beta E_4$. The visible redexes in each of these terms are as follows:
In E_1: $(\lambda_f.\lambda_x.\lambda_y.fxy)+$.
In E_2: $(\lambda_f.(\lambda_x.\lambda_y.fxy)m)+$ and $(\lambda_x.\lambda_y.fxy)m$.
In E_3: $(\lambda_f.((\lambda_x.\lambda_y.fxy)m)n)+$ and $(\lambda_x.\lambda_y.fxy)m$.
In E_4: $(\lambda_f.(\lambda_x.(\lambda_y.fxy)n)m)+$, $(\lambda_x.(\lambda_y.fxy)n)m$ and $(\lambda_y.fxy)n$.
Moreover, one can see potential *future redexes as follows:*
In E_1: $\lambda_x.-$ will eventually be applied to m and $\lambda_y.-$ will be eventually be applied to n.
In E_2: $\lambda_y.-$ will eventually be applied to n.
In E_3: $\lambda_y.-$ will eventually be applied to n.
Note that $E_1 \to_\theta E_2 \to_\theta E_3 \to_\theta E_4$ and that by transforming E_1 to E_2 (resp. E_3 to E_4), an extra redex becomes visible. In E_4 all redexes are visible and E_4 is in θ-normal form.

Based on this, one wonders if one could classify terms according to their transformational relationship as in E_1, E_2, E_3, E_4. One has to be careful however as the example below shows that we need to base reduction on classes.

Example 3 *If $A_1 \equiv ((\lambda_x.((\lambda_y.\lambda_z.\lambda_t.E)D)C)B)A$ and $A_2 \equiv ((\lambda_x.(\lambda_y.\lambda_z.(\lambda_t.E)A)D)B)C$. Then, $A_1 \not\to_\theta A_2$ and $A_2 \not\to_\theta A_1$. But, these two terms have the same θ-normal form and are reductionally equivalent. We will show in Section 4 that classes will be able to represent this.*

In this paper, we propose to define for each term M, a term $\mathrm{TS}(M)$ (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 $\{M' \mid \mathrm{TS}(M) \equiv \mathrm{TS}(M')\}$. As $\twoheadrightarrow_\theta$ is Church Rosser and strongly normalizing, then if $M_1 =_\theta M_2$ we get $\mathrm{TS}(M_1) \equiv \mathrm{TS}(M_2)$. 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. For Examples 1 and 2, TS will help us establish that A and B are reductionally equivalent and that $E_1, \ldots, E_4$ are all reductionally equivalent.

2.2 Reductional Equivalence

In order to discuss reductional equivalence between terms, redexes will be *extended* (cf. Definition 24) so that a potential future redex like $(\lambda_y.-)x$ in A of Example 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 another redex:

$r_2' = (\lambda_y.\lambda_f.fy)x$, 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 observe that there is a bijective correspondence between the (extended) redexes of A and B of Example 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. This is illustrated by this example:

Example 4 *The reduction paths from A and B of Example 1 are as follows:*
A-$Path_1$: $(\lambda_\beta.\lambda_y.\lambda_f.fy)\alpha x \to_{r'_1} (\lambda_y.\lambda_f.fy)x \to \lambda_f.fx$
A-$Path_2$: $(\lambda_\beta.\lambda_y.\lambda_f.fy)\alpha x \to_{r'_2} (\lambda_\beta.\lambda_f.fx)\alpha \to \lambda_f.fx$
B-$Path_1$: $(\lambda_\beta.(\lambda_y.\lambda_f.fy)x)\alpha \to_{r_1} (\lambda_y.\lambda_f.fy)x \to \lambda_f.fx$
B-$Path_2$: $(\lambda_\beta.(\lambda_y.\lambda_f.fy)x)\alpha \to_{r_2} (\lambda_\beta.\lambda_f.fx)\alpha \to \lambda_f.fx$
It is clear that A and B have the same number of possible paths before reaching the normal form and that there is a bijective correspondence between the paths A-$Path_1$ and B-$Path_1$, and between A-$Path_2$ and B-$Path_2$.

These considerations lead us to define reductional equivalence $\sim_{\texttt{equi}}$ by:

Definition 5 *We say that A and B are reductionally equivalent and write $A \sim_{\texttt{equi}} B$ iff*
1. *If A or B is in normal form then A is syntactically equal to B.*
2. *There is a bijective correspondence between the (extended) redexes of A and B.*
3. *Contracting an (extended) redex in A results in a value syntactically equal ($\equiv$) or reductionally equivalent ($\sim_{\texttt{equi}}$) to the result of contracting the corresponding redex in B and vice versa.*

Example 6 *$A \sim_{\texttt{equi}} B$ for A, B as in Example 1. Also $E_1 \sim_{\texttt{equi}} E_2 \sim_{\texttt{equi}} E_3 \sim_{\texttt{equi}} E_4$ for E_1, E_2, E_3, E_4 as in Example 2. But, due to clause 2 of the above definition, it is not the case that* $\mathbf{KII} \sim_{\texttt{equi}} \mathbf{KI\Omega}$.

Remark 7 *Note that the above clauses cannot handle the following two situations:*
- *Forbidding the reductional equivalence of terms such as: $(\lambda_x.\mathbf{I})\mathbf{I}$ and $(\lambda_x.\mathbf{I})\mathbf{K}$.*
- *Ensuring the composionality of reductional equivalence in the sense that if $A_1 \sim_{\texttt{equi}} A_2$ then $A_1B \sim_{\texttt{equi}} A_2B$ and $\lambda_x.A_1 \sim_{\texttt{equi}} \lambda_x.A_2$. For example, if $A_1 \equiv \lambda_z.(\lambda_x.y)z$ and $A_2 \equiv \lambda_z.(\lambda_x.y)\mathbf{I}$, then $A_1 \sim_{\texttt{equi}} A_2$ but $A_1(\mathbf{II}) \not\sim_{\texttt{equi}} A_2(\mathbf{II})$.*

In order to handle the above situations, we need to add the following fourth clause to Definition 5:

4. Arguments of corresponding (extended) redexes are syntactically equal or reductionally equivalent.

Clause 4 above solves the problems raised by the above two situations. We will not however be concerned with these situations in this paper and we will not hence include clause 4 in Definition 5.

We conjecture that in general it is undecidable whether two terms are reductionally equivalent.

Conjecture 8 *It is undecidable whether two terms are reductionally equivalent.*

2.3 Shuffle-equivalence, shuffle-reduction and summary of results

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 [8]. First, the term-reshuffling of [8] 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 since 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. The extended redex r'_2 in A becomes classical in B. We call B the reshuffled version of A. We have seen that $A \sim_{\texttt{equi}} B$.

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); TS will be defined in Section 4.

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 [8] (which turns out to be the θ-normal form). Furthermore, we depart from [8] 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 $\{M' \,|\, \mathrm{TS}(M) \equiv \mathrm{TS}(M')\}$. Hence, E_1, E_2, E_3 and E_4 above belong to the same class. All elements of $[M]$ are β-equal and have in some sense 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 (cf. Fact 41) that on strongly normalizing terms, two shuffle-equivalent terms are reductionally equivalent, that shuffle-equivalence is decidable and does not coincide with reductional equivalence (which we conjecture to be undecidable).

Certainly, our notion of shuffle-equivalence captures existing extensions of reductions. For example, if $A \to_\theta B$ then A and B are shuffle-equivalent. The converse is not true (cf. Example 23).

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 existing 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 (cf. Lemma 26) that both $\to_\beta$ and the generalised reduction $\hookrightarrow_\beta$ of [4] are captured by $\rightsquigarrow_\beta$.

3 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 9 *It is easy to see that* $A \equiv (u\delta)(w\delta)(\lambda_x)(v\delta)(\lambda_y)(\lambda_z)(z\delta)(y\delta)x$ *in item notation. By moving the item* $(u\delta)$ *to the right until it is next to its 'matching partner'* (λ_z), *this* A *reshuffles to* $\mathrm{TS}(A) \equiv (w\delta)(\lambda_x)(v\delta)(\lambda_y)(u\delta)(\lambda_z)(z\delta)(y\delta)x$.
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 10 E_1 *of Example 2 reads* $(n\delta)(m\delta)(+\delta)(\lambda_f)(\lambda_x)(\lambda_y)(y\delta)(x\delta)f$ *in item notation. Here, the (classical) redex corresponds to a '$\delta\lambda$-pair' followed by the body of the abstraction, as follows:*
$(\lambda_f.(\lambda_x.\lambda_y.fxy)m)+$ *corresponds to* $(+\delta)(\lambda_f)(\lambda_x)(\lambda_y)(y\delta)(x\delta)f$.
Note that the δ-item $(+\delta)$ *and the λ-item* (λ_f) *are now adjacent, which is characteristic for the presence of a classical redex in item notation. (Cf. Figure 1 which represents* E_1*). The second and third redexes of* E_1 *are obtained by matching δ and λ-items which are* not *adjacent:*

- $(\lambda_y.fxy)n$ *is visible as it corresponds to the matching* $(n\delta)(\lambda_y)$ *where* $(n\delta)$ *and* (λ_y) *are separated by the segment* $(m\delta)(+\delta)(\lambda_f)(\lambda_x)$ *which has the bracketing structure* $[\,[\,]\,]$.
- $(\lambda_x.\lambda_y.fxy)m$ *is visible as it corresponds to the matching* $(m\delta)(\lambda_x)$ *where* $(m\delta)$ *and* (λ_x) *are separated by the segment* $(+\delta)(\lambda_f)$.

As above, we will use obvious notions throughout like **partner**, **match**, **bachelor**, etc. In Figure 2, $(+\delta)$ 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. The adjacent item pair $(+\delta)(\lambda_f)$ is called a **$\delta\lambda$-pair** and the non-adjacent items $(n\delta)(\lambda_y)$ form a **$\delta\lambda$-couple**.

Term reshuffling amounts to moving δ-items to occur next to their matching λ-items. Hence E_1 of Example 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 4 (which represents E_4). Furthermore, Figures 2 and 3 (which represent E_2 and E_3) also change to Figure 4.

According to our shuffle-equivalence, E_1, E_2, E_3 and E_4 belong to the same class and are $\sim_{\mathrm{equi}}$.

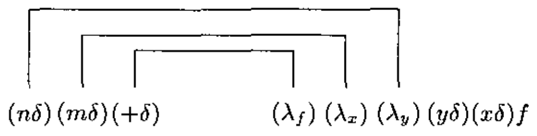
$(n\delta)\,(m\delta)\,(+\delta)\quad(\lambda_f)\,(\lambda_x)\,(\lambda_y)\,(y\delta)(x\delta)f$

Fig. 1. More extended redexes: E_1

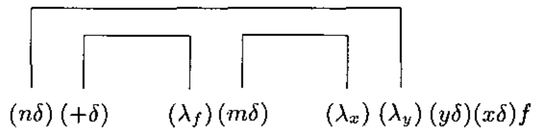
$(n\delta)\,(+\delta)\quad(\lambda_f)\,(m\delta)\quad(\lambda_x)\,(\lambda_y)\,(y\delta)(x\delta)f$

Fig. 2. (Extended) redexes in item notation: E_2

In item notation, 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 each s_i is either a λ-item or a δ-item, and $x \in V$. These items $s_1, s_2, \ldots, s_n$ are called the **main items** of A, x is called the **heart** of A, notation $\heartsuit(A)$.[1] 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. Well-balanced segments are given inductively by: (i) $\emptyset$ is w-b, (ii) if $\overline{s}$ is w-b then $(A\delta)\overline{s}(\lambda_x)$ is w-b, (iii) if $\overline{s_1}, \overline{s_2}, \ldots \overline{s_n}$ are w-b, then the concatenation $\overline{s_1}\ \overline{s_2}, \cdots \overline{s_n}$ is w-b.

In Figures 1, 2, 3, and 4, all segments that occur under a hat are w-b.

Lemma 11 *Every term has one of the following three forms: (i) $(A_1\delta)\cdots(A_n\delta)x$, where $x \in V$ and $n \geq 0$, (ii) $(\lambda_x)B$, and (iii) $(A_1\delta)\cdots(A_n\delta)(B\delta)(\lambda_x)D$, where $n \geq 0$.*

Definition 12 *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 the usual Barendregt variable convention BC and definition of compatibility (cf. [3]). 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.

4 Shuffle-equivalence

In this section we follow [8] 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 $A =_\beta \mathrm{TS}(A)$, i.e., A

[1] Note that the term *head variable* used in [2] is a special case of our notion of heart. The head variable of a term in head normal form is the heart of the term. It is not the case however that the heart of a term is always a head variable.

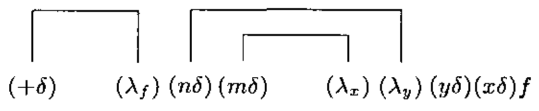
$(+\delta)\quad(\lambda_f)\,(n\delta)\,(m\delta)\quad(\lambda_x)\,(\lambda_y)\,(y\delta)(x\delta)f$

Fig. 3. The term E_3 in item notation

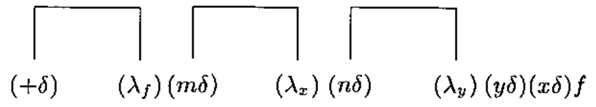

Fig. 4. The reshuffled term E_2 in item notation: E_4

and $\mathrm{TS}(A)$ are semantically equivalent (cf. Lemma 18). Furthermore, we show that shuffle-equivalence is stronger than $\to_\theta$ and than the term-reshuffling of [8] (cf. Lemma 22 and Example 23) and that shuffle-equivalence classes are decidable (cf. Lemma 21). In Section 5 (cf. Lemma 40), we show that shuffle-equivalence is indeed a (decidable) approximation of reductional equivalence.

Definition 13 *The reshuffling function* TS *is defined such that:*

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

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

Note that in this definition we use the notions of the previous section: if a term starts with a λ_x-item then this item is bachelor. If a term starts with a partnered δ-item then the last clause above applies. If a term starts with a bachelor δ-item then the term starts with a positive number of bachelor δ-items followed by either a variable in which case the second clause applies or a well-balanced segment in which case the last clause applies. With term reshuffling, well-balanced segments are rewritten so that $\delta\lambda$-couples become $\delta\lambda$-pair and all bachelor main δ-items are moved to the right of all well-balanced main segments.

Example 14 *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, $\mathrm{TS}(A)$ 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 $\delta\lambda$-pair or a bachelor main λ-item. Now, the following lemmas show some properties of TS.

Lemma 15 (Decidability of TS) *For any A, B, it is decidable whether* $\mathrm{TS}(A) \equiv \mathrm{TS}(B)$.

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

Lemma 16

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

Proof: *1. Every time at most one case of the definition of* $\mathrm{TS}(M)$ *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* `weight`$(\overline{s})$. □

Lemma 17 *For a term A,* $\mathrm{TS}(A) \equiv \overline{s_0}\,\overline{s_1}\heartsuit(A)$*, 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: *By induction on* `weight`(A).

- *Case $A \equiv (\lambda_x)C$, use IH on C. Case $A \equiv (B_1\delta)\cdots(B_n\delta)x$, $x \in V$, then $\overline{s_0}$ is empty.*
- *$A \equiv (B_1\delta)\cdots(B_n\delta)(C\delta)(\lambda_x)E$. Then* $\mathrm{TS}(A) \equiv (\mathrm{TS}(C)\delta)(\lambda_x)\mathrm{TS}((B_1\delta)\cdots(B_n\delta)E)$*. By the induction hypothesis* $\mathrm{TS}((B_1\delta)\cdots(B_n\delta)E)$ *is of the form* $\overline{s_0}\,\overline{s_1}\heartsuit(E) \equiv \overline{s_0}\,\overline{s_1}\heartsuit(A)$. □

Lemma 18 *For all terms A, B and variable x:*
1. $\mathrm{TS}(A) \equiv \mathrm{TS}(\mathrm{TS}(A))$, *2.* $\mathrm{TS}(A[x := B]) \equiv \mathrm{TS}(\mathrm{TS}(A)[x := \mathrm{TS}(B)])$, *and 3.* $A =_\beta \mathrm{TS}(A)$.
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 $\mathrm{TS}(A) \equiv (\mathrm{TS}(B)\delta)(\lambda_x)\mathrm{TS}((A_1\delta)\cdots(A_n\delta)D) \overset{IH}{=}_\beta$
$(B\delta)(\lambda_x)(A_1\delta)\cdots(A_n\delta)D =_\beta ((A_1\delta)\cdots(A_n\delta)D)[x := B] \overset{x\notin FV(A_i)}{=}_\beta$
$(A_1\delta)\cdots(A_n\delta)D[x := B] =_\beta (A_1\delta)\cdots(A_n\delta)(B\delta)(\lambda_x)D$ □

Corollary 19 *For all terms A, B,* $\mathrm{TS}(A) =_\beta \mathrm{TS}(B)$ *iff $A =_\beta B$.* □

Definition 20 (Shuffle-class and shuffle-equivalence) *For a term A, we define $[A]$, the shuffle class of A, to be $\{B \mid \mathrm{TS}(A) \equiv \mathrm{TS}(B)\}$. We say that A and B are shuffle-equivalent iff $[A] = [B]$.*

Lemma 21 *For any A, B, it is decidable whether $A \in [B]$. Moreover, if $A \in [B]$ then $A =_\beta B$.*
Proof: *Follows from Lemma 15 and Corollary 19.* □

The following shows that shuffle-equivalence contains $\rightarrow_\theta$ and the term-reshuffling of [8].

Lemma 22 *If $A \rightarrow_\theta B$ or A term-reshuffles to B in the sense of [8] then A and B are shuffle-equivalent.*
Proof: TS *formalizes term-reshuffling of [8] and the latter captures $\rightarrow_\theta$.* □

The other way round does not always hold however:

Example 23 *Let $A \equiv (A_1\delta)(A_2\delta)(A_3\delta)(\lambda_x)(\lambda_y)(\lambda_z)A_4$ and $B \equiv (A_2\delta)(A_3\delta)(\lambda_x)(\lambda_y)(A_1\delta)(\lambda_z)A_4$. A and B are shuffle-equivalent but are not related by $\rightarrow_\theta$ (there exists $C \not\equiv A$ however such that $B \rightarrow_\theta C$ and $A \twoheadrightarrow_\theta C$). Moreover, neither $A \equiv \mathrm{TS}(B)$ nor $B \equiv \mathrm{TS}(A)$, but* $\mathrm{TS}(A) \equiv \mathrm{TS}(B)$.

5 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 $A \rightarrow_\beta B$ then $A \rightsquigarrow_\beta B$. We show that shuffle-equivalence implies reductional equivalence on SN terms (Definition 5) and that shuffle-reduction on classes makes more redexes visible and allows for smaller terms during reductions.

Definition 24 (Shuffle-reduction, extended redexes and $\hookrightarrow_\beta$) – *One-step shuffle-reduction $\rightsquigarrow_\beta$ is the least compatible relation generated by:*
$A \rightsquigarrow_\beta B$ *iff* $\exists A' \in [A] \exists B' \in [B][A' \rightarrow_\beta B']$
Many-step shuffle-reduction $\twoheadrightarrow\!\!\!\!\sim_\beta$ is the reflexive and transitive closure of $\rightsquigarrow_\beta$ and $\approx_\beta$ is the least equivalence relation generated by $\twoheadrightarrow\!\!\!\!\sim_\beta$.
- *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 25 *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 [4,8] where it was shown to be more general than other generalised notions of reduction introduced in the literature (such as (g) of Section 6). The following lemma shows that $\rightsquigarrow_\beta$ is more general than $\hookrightarrow_\beta$ (the generalised reduction of [4,8]) and that it captures classical β-reduction.

Lemma 26 $\to_\beta \subset \hookrightarrow_\beta \subset \rightsquigarrow_\beta$.

Proof: *It suffices to show* $(A\delta)(\lambda_x)C \hookrightarrow_\beta C[x := A]$ *and* $(A\delta)\overline{s}(\lambda_x)C \rightsquigarrow_\beta \overline{s}C[x := A]$. $(A\delta)(\lambda_x)C \equiv (A\delta)\emptyset(\lambda_x)C \hookrightarrow_\beta \emptyset C[x := A] \equiv C[x := A]$. *Also, by Lemma 16, we know that* $(A\delta)\overline{s}(\lambda_x)C \in [\overline{s}(A\delta)(\lambda_x)C]$, *and since* $\overline{s}(A\delta)(\lambda_x)C \to_\beta \overline{s}C[x := A]$ *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. For example, if* $A_1 \equiv (A\delta)(B\delta)(\lambda_x)(C\delta)(D\delta)(\lambda_y)(\lambda_z)(\lambda_t)E$ *and* $A_2 \equiv (C\delta)(B\delta)(\lambda_x)(D\delta)(\lambda_y)(\lambda_z)(E[t := A])$ *(which have respectively the bracketing structures* [[ ] [[ ]]] *and* [[ ] [ ]]*), then* $A_1 \rightsquigarrow_\beta A_2$ *but* $A_1 \not\hookrightarrow_\beta A_2$. *Similarly,* $(A\delta)(B\delta)(\lambda_x)(\lambda_y)C \hookrightarrow_\beta (B\delta)(\lambda_x)C[y := A]$ *but* $(A\delta)(B\delta)(\lambda_x)(\lambda_y)C \not\to_\beta (B\delta)(\lambda_x)C[y := A]$. □

Corollary 27 $\twoheadrightarrow_\beta \subset \lhook\joinrel\twoheadrightarrow_\beta \subset \rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta$. □

Remark 28 It is not in general true that $A \rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta B \Rightarrow \exists A' \in [A]\exists B' \in [B][A' \twoheadrightarrow_\beta B']$. 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: 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 $\to_\beta$ reduct of A' is $(w\delta)(w\delta)(\lambda_u)(\lambda_v)v$, which doesn't $\to_\beta$-reduce to B. In Lemma 36 however, we find a correspondence between $\rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta$ on classes and $\twoheadrightarrow_\beta$ on terms.

Lemma 29 $\mathrm{TS}(A) \hookrightarrow_\beta B$ *iff* $\mathrm{TS}(A) \to_\beta B$.

Proof: *This is a direct consequence of Lemma 17.* □

Lemma 30 *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 31 $A \rightsquigarrow_\beta B$ *iff* $\mathrm{TS}(A) \rightsquigarrow_\beta \mathrm{TS}(B)$.

Remark 32 Note that $A \hookrightarrow_\beta B \not\Rightarrow \mathrm{TS}(A) \lhook\joinrel\twoheadrightarrow_\beta \mathrm{TS}(B)$ nor do we have $A \to_\beta B \Rightarrow \mathrm{TS}(A) \twoheadrightarrow_\beta \mathrm{TS}(B)$. 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 $A \to_\beta B$ (hence $A \hookrightarrow_\beta B$) yet $\mathrm{TS}(A) \equiv A \not\twoheadrightarrow_\beta$ nor $\not\twoheadrightarrow_\beta$ $\mathrm{TS}(B) \equiv (y\delta)(z\lambda_u)(y\delta)(z\lambda_v)v$.

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

Lemma 33 *If* $A \rightsquigarrow_\beta B$ *then* $A =_\beta B$.

Proof: *Say* $A' \in [A]$, $B' \in [B]$, $A' \to_\beta B'$. *Then by Lemma 18:* $A =_\beta \mathrm{TS}(A) \equiv \mathrm{TS}(A') =_\beta A' =_\beta B' =_\beta \mathrm{TS}(B') \equiv \mathrm{TS}(B) =_\beta B$. □

Corollary 34

1. *If* $A \rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta B$ *then* $A =_\beta B$. 2. $A \approx_\beta B$ *iff* $A =_\beta B$ *iff* $A \sim_\beta B$ *iff* $\mathrm{TS}(A) =_\beta \mathrm{TS}(B)$. □

Theorem 35 *(The general Church Rosser theorem for* $\rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta$*)*
If $A \rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta B$ *and* $A \rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta C$, *then there exists* D *such that* $B \rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta D$ *and* $C \rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta D$.

Proof: *As* $A \rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta B$ *and* $A \rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta C$ *then by Corollary 34,* $A =_\beta B$ *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* $M \rightsquigarrow\!\!\!\!\twoheadrightarrow_\beta N$. *Hence we are done.* □

As we noted in Remark 32, we can have $\mathrm{TS}(C) \to_\beta D$ where $D \not\equiv \mathrm{TS}(D)$. Nevertheless, term reshuffling preserves β-reduction. This is a generalisation of the result in [8] to equivalence classes.

Lemma 36 *If* $A, B \in \mathcal{T}$ *and* $A \rightsquigarrow_\beta B$ *then* $(\exists B' \in [B])[\mathrm{TS}(A) \to_\beta B']$. *In other words, the following diagram commutes:*

$$\begin{tikzcd}
A \arrow[rrr, rightsquigarrow, "\beta"] \arrow[d, dash] & & & B \arrow[d, dotted, dash] \\
\mathrm{TS}(A) \arrow[rrr, dashed, "\beta"] & & & B' \in [B]
\end{tikzcd}$$

Proof: *We prove by induction on the structure of A' that if $A' \to_\beta B' \in [B]$, then for some B'', $\mathrm{TS}(A') \to_\beta B'' \in [B]$. The compatibility cases are easy, distinguish cases according to the definition of* TS. *If $A' \equiv (C\delta)(\lambda_x)E$ and $B' \equiv E[x := C] \in [B]$ then $\mathrm{TS}(A') \equiv (\mathrm{TS}(C)\delta)(\lambda_x)\mathrm{TS}(E) \to_\beta \mathrm{TS}(E)[x := \mathrm{TS}(C)]$ and by Lemma 18, $\mathrm{TS}(\mathrm{TS}(E)[x := \mathrm{TS}(C)]) \equiv \mathrm{TS}(E[x := C]) \in [B]$.* □

Corollary 37 *If $A \rightsquigarrow\!\!\!\!\to_\beta B$ then there exist $A_0, A_1, \ldots, A_n$ such that*
$(A \equiv A_0) \wedge (\mathrm{TS}(A_0) \to_\beta A_1) \wedge (\mathrm{TS}(A_1) \to_\beta A_2) \wedge \cdots \wedge (\mathrm{TS}(A_{n-1}) \to_\beta A_n \in [B])$
Proof: *By induction on $\rightsquigarrow\!\!\!\!\to_\beta$.* □

Now we show that shuffle-equivalence preserves strong normalization:

Lemma 38 *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 30 Hence, A' must be $\in SN_{\rightsquigarrow_\beta}$.* □

Moreover, shuffle-reduction preserves β-strong normalization:

Lemma 39 *$A \in SN_{\rightsquigarrow_\beta}$ iff $A \in SN_{\to_\beta}$.*
Proof: *As $\to_\beta \subset \rightsquigarrow_\beta$, $\Longrightarrow$ is immediate. $\Longleftarrow$ is by using a result of [20] 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 11. 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 $(A_1\delta)\cdots(A_n\delta)(B\delta)(\lambda_x)D \to_\beta (A_1\delta)\cdots(A_n\delta)\{D[x := B]\}$ which satisfies IH. Another way is to follow the lines of [6].* □

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

Lemma 40 *Let $A \in SN_{\rightsquigarrow_\beta}$. Then for all $A' \in [A]$, $A' \sim_{\mathrm{equi}} A$.*
Proof: *It is sufficient to show that $(B\delta)\overline{s}C \sim_{\mathrm{equi}} \overline{s}(B\delta)C$ 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 normal form 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 41 *The following holds:*
1. *Let $A \in SN_{\rightsquigarrow_\beta}$. If $\mathrm{TS}(A) \equiv \mathrm{TS}(B)$ then $A \sim_{\mathrm{equi}} B$ (Lemma 40).*
2. *$\mathrm{TS}(A) \equiv \mathrm{TS}(B)$ does not necessarily imply $A \sim_{\mathrm{equi}} B$ (Example 42).*
3. *$A \sim_{\mathrm{equi}} B$ does not necessarily imply $\mathrm{TS}(A) \equiv \mathrm{TS}(B)$ (Example 43 below).*
4. *$\mathrm{TS}(A) \equiv \mathrm{TS}(B)$ is decidable (Lemma 15).*
5. *Let $A \in SN_{\rightsquigarrow_\beta}$. Then for all $A' \in [A]$, $A' \in SN_{\rightsquigarrow_\beta}$.*

Example 42 *Let $A \equiv (a\delta)(b\delta)(\lambda_x)(\lambda_y)((\lambda_z)(z\delta)z\delta)(\lambda_z)(z\delta)z$ (i.e., $(\lambda_x.\lambda_y.\boldsymbol{\Omega})ba$) and $B \equiv (b\delta)(\lambda_x)(a\delta)(\lambda_y)((\lambda_z)(z\delta)z\delta)(\lambda_z)(z\delta)z$ (i.e., $(\lambda_x.(\lambda_y.\boldsymbol{\Omega})a)b$) for $\boldsymbol{\Omega} \equiv (\lambda_z.zz)(\lambda_z.zz)$. Now, $\mathrm{TS}(A) \equiv \mathrm{TS}(B)$ but $A \not\sim_{\mathrm{equi}} B$ since contracting $\boldsymbol{\Omega}$ will not result in syntactically equivalent terms. This shows in 1. of Fact 41 that one cannot drop the assumption that A is strongly normalizing.*

Example 43 *Let $A \equiv ((a\delta)(\lambda_x)x\delta)(\lambda_y)y$ and $B \equiv (a\delta)(\lambda_x)(x\delta)(\lambda_y)y$. We have $A \sim_{\mathrm{equi}} B$ but $\mathrm{TS}(A) \not\equiv \mathrm{TS}(B)$. The same holds for the terms $(a\delta)(\lambda_y)(y\delta)y$ and $(a\delta)(\lambda_y)(y\delta)a$. This shows that the converse of 1. in Fact 41 does not hold.*

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

Example 44 *Let $M \equiv (\lambda_x.\lambda_y.y(Cxx\cdots x))B(\lambda_z.u)$ where B is a BIG term and u is in normal form, $z \notin FV(u)$. Then $M \to_\beta (\lambda_y.y(CBB\cdots B))(\lambda_z.u) \to_\beta (\lambda_z.u)(CBB\cdots B) \to_\beta u$. Now the first and second reducts both contain $CBB\cdots B$, so they are very long. Shuffle reduction allows us to reduce M in such a way that all the new terms are of smaller size than M. This can be seen as follows: $\mathrm{TS}(M) \equiv (\lambda_x.(\lambda_y.y(Cxx\cdots x))\lambda_z.u)B \to_\beta (\lambda_x.(\lambda_z.u)(Cxx\cdots x))B \to_\beta (\lambda_x.u)B \to_\beta u$. So shuffle reduction might allow us to define clever strategies that reduce terms via paths of relatively small terms.*

6 Comparison with previous work and Conclusion

The last decade has 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 β-step. γ_C 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 (which imply g-like rules).

Now, we discuss where generalised reduction has been used (cf. [13,9]).

[19] introduces the notion of a *premier redex* which is similar to the redex based on λ_y and Q in the rule (θ) above (which we call *generalised redex*). [20] 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). [23] also introduces reductions similar to those of [20]. Furthermore, [11] uses θ (and other reductions) to show that typability in ML is equivalent to acyclic semi-unification. [21] uses a reduction which has some common themes with θ. Nederpelt's thesis in [18] and [5] use θ whereas [14] uses γ to reduce the problem of β-strong normalization to the problem of weak normalization (WN) for related reductions. [12] 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. [16,24,22,15] use related reductions to reduce SN to WN and [10] 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. [8] 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 shuffle equivalence which we consider to be the best approximation of reductional equivalence.

This paper unifies all this previous work by looking for the class of reductionally equivalent terms. Such a class is conjectured to be undecidable but a decidable approximation of it which does indeed capture the existing new notions of reduction, is given in this paper. Our shuffle-equivalence is the closest decidable approximation to reductional equivalence up to now. 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 approach 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 behavior, we find many extensions $\to_e$ of β-reduction $\to_\beta$ where if $A \to_\beta B$ then $A \to_e B$ and where the equivalence relation generated by $\to_e$ is just β-equality. These extensions make more redexes visible and hence allow for more flexibility in reducing a term.

We propose shuffle-reduction as a generalisation of these extensions. 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 [20] concerning the existence of a syntax for terms realising shuffle-equivalence (which Regnier [20] calls σ-equivalence).

References

1. Z.M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A call by need lambda calculus. *22nd ACM Symposium on Principles of Programming Languages*, 1995.
2. H. P. Barendregt. *The Lambda Calculus: Its syntax and Semantics.* North-Holland, revised edition, 1984.
3. H.P. Barendregt. λ-calculi with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume II, pages 118–310. Oxford University Press, 1992.
4. R. Bloo, F. Kamareddine, and R. P. Nederpelt. The Barendregt Cube with Definitions and Generalised Reduction. *Information and Computation*, 126 (2):123–143, 1996.
5. P. de Groote. The conservation theorem revisited. In *International Conference on Typed Lambda Calculi and Applications, LNCS*, volume 664. Springer-Verlag, 1993.
6. F. Kamareddine. Postponement, conservation and preservation of strong normalisation for generalised reduction. *Logic and Computation*, 10(5), 2000.
7. F. Kamareddine and R. Nederpelt. A useful λ-notation. *Theoretical Computer Science*, 155:85–109, 1996.
8. F. Kamareddine and R. P. Nederpelt. Generalising reduction in the λ-calculus. *Journal of Functional Programming*, 5(4):637–651, 1995.
9. F. Kamareddine, A. Ríos, and J.B. Wells. Calculi of generalised β_e-reduction and explicit substitution: Type free and simply typed versions. *Journal of Functional and Logic Programming*, 1998.
10. M. Karr. Delayability in proofs of strong normalizability in the typed λ-calculus. In *Mathematical Foundations of Computer Software, LNCS*, volume 185. Springer-Verlag, 1985.
11. A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. An analysis of ML typability. *ACM*, 41(2):368–398, 1994.
12. A.J. Kfoury and J.B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second order λ-calculus. *Proceedings of the 1994 ACM Conference on LISP and Functional Programming*, 1994.
13. 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.
14. 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.
15. Z. Khasidashvili. The longest perpetual reductions in orthogonal expression reduction systems. *Proc. of the 3rd International Conference on Logical Foundations of Computer Science, Logic at St Petersburg*, 813, 1994.
16. J. W. Klop. Combinatory Reduction Systems. *Mathematical Center Tracts*, 27, 1980. CWI.
17. E. Moggi. Computational λ-calculus and monads. *LICS'89*, 1989.
18. R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer. *Selected papers on Automath.* North-Holland, Amsterdam, 1994.
19. L. Regnier. *Lambda calcul et réseaux.* PhD thesis, University Paris 7, 1992.
20. L. Regnier. Une équivalence sur les lambda termes. *Theoretical Computer Science*, 126:281–292, 1994.
21. A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Proceedings of the 1992 ACM Conference on LISP and Functional Programming*, pages 288–298, 1992.
22. M. M. Sørensen. Strong normalisation from weak normalisation in typed λ-calculi. *Information and Computation*, 133(1), 1997.
23. D. Vidal. *Nouvelles notions de réduction en lambda calcul.* PhD thesis, Université de Nancy 1, 1989.
24. H. Xi. On weak and strong normalisations. Technical Report 96-187, Carnegie Mellon University, 1996.

		edited by R.C. Backhouse, p. 180
98/09	K.M. van Hee and H.A. Reijers	An analytical method for assessing business processes, p. 29.
98/10	T. Basten and J. Hooman	Process Algebra in PVS
98/11	J. Zwanenburg	The Proof-assistemt Yarrow, p. 15
98/12	Ninth ACM Conference on Hypertext and Hypermedia Hypertext '98 Pittsburgh, USA, June 20-24, 1998 Proceedings of the second workshop on Adaptive Hypertext and Hypermedia.	Edited by P. Brusilovsky and P. De Bra, p. 95.
98/13	J.F. Groote, F. Monin and J. v.d. Pol	Checking verifications of protocols and distributed systems by computer. Extended version of a tutorial at CONCUR'98, p. 27.
98/14	T. Verhoeff (artikel volgt)	
99/01	V. Bos and J.J.T. Kleijn	Structured Operational Semantics of χ , p. 27
99/02	H.M.W. Verbeek, T. Basten and W.M.P. van der Aalst	Diagnosing Workflow Processes using Woflan, p. 44
99/03	R.C. Backhouse and P. Hoogendijk	Final Dialgebras: From Categories to Allegories, p. 26
99/04	S. Andova	Process Algebra with Interleaving Probabilistic Parallel Composition, p. 81
99/05	M. Franssen, R.C. Veltkamp and W. Wesselink	Efficient Evaluation of Triangular B-splines, p. 13
99/06	T. Basten and W. v.d. Aalst	Inheritance of Workflows: An Approach to tackling problems related to change, p. 66
99/07	P. Brusilovsky and P. De Bra	Second Workshop on Adaptive Systems and User Modeling on the World Wide Web, p. 119.
99/08	D. Bosnacki, S. Mauw, and T. Willemse	Proceedings of the first international syposium on Visual Formal Methods - VFM'99
99/09	J. v.d. Pol, J. Hooman and E. de Jong	Requirements Specification and Analysis of Command and Control Systems
99/10	T.A.C. Willemse	The Analysis of a Conveyor Belt System, a case study in Hybrid Systems and timed μ CRL, p. 44.
99/11	J.C.M. Baeten and C.A. Middelburg	Process Algebra with Timing: Real Time and Discrete Time, p. 50.
99/12	S. Andova	Process Algebra with Probabilistic Choice, p. 38.
99/13	K.M. van Hee, R.A. van der Toorn, J. van der Woude and P.A.C. Verkoulen	A Framework for Component Based Software Architectures, p. 19
99/14	A. Engels and S. Mauw	Why men (and octopuses) cannot juggle a four ball cascade, p. 10
99/15	J.F. Groote, W.H. Hesselink, S. Mauw, R. Vermeulen	An algorithm for the asynchronous *Write-All* problem based on process collision*, p. 11.
99/16	G.J. Houben, P. Lemmens	A Software Architecture for Generating Hypermedia Applications for Ad-Hoc Database Output, p. 13.
99/17	T. Basten, W.M.P. v.d. Aalst	Inheritance of Behavior, p.83
99/18	J.C.M. Baeten and T. Basten	Partial-Order Process Algebra (and its Relation to Petri Nets), p. 79
99/19	J.C.M. Baeten and C.A. Middelburg	Real Time Process Algebra with Time-dependent Conditions, p.33.
99/20	Proceedings Conferentie Informatiewetenschap 1999 Centrum voor Wiskunde en Informatica 12 november 1999, p.98	edited by P. de Bra and L. Hardman
00/01	J.C.M. Baeten and J.A. Bergstra	Mode Transfer in process Algebra, p. 14
00/02	J.C.M. Baeten	Process Algebra with Explicit Termination, p. 17.
00/03	S. Mauw and M.A. Reniers	A process algebra for interworkings, p. 63.
00/04	R. Bloo, J. Hooman and E. de Jong	Semantical Aspects of an Architecture for Distributed Embedded Systems*, p. 47.