

# Primitive and Some/Any Proof Principles for $\beta$ -Standardisation<sup>\*</sup>

René Vestergaard

Japan Advanced Institute of Science and Technology,  
Tatsunokuchi, Ishikawa 923-1211, Japan

**Abstract.** We consider formal provability for three kinds of  $\beta$ -standardisation for the  $\lambda$ -calculus seen as a programming language and show that there are substantial and instructive short-comings in the known proofs. In order to overcome the identified problems, we employ a range of proof techniques that we have developed for reasoning formally about programming languages, including *Some/Any proof theory*. The techniques are reasonably mature and succinct and should be of general interest.

## 1 Introduction

In [16, 17], we presented the first formalised proof of a major programming-language property ( $\beta$ -confluence) that was conducted exclusively with syntactic (i.e., structural) proof principles. As such, the proof (i) pertains directly to the language in question and (ii) justifies informal practice. We also showed that, although a proof conducted in some alternative framework that is *adequate* with respect to the syntactic presentation results in a property of the right geometric shape, it would typically be logically weaker than ours because adequacy does not allow for reflection of rewriting properties per se.

In this paper, we tackle an even more advanced property,  $\beta$ -standardisation, and show that the discrepancies between syntax-based formal proofs, on the one hand, and informal proofs and formal proofs in alternative frameworks, on the other, are even bigger here. Standardisation is used for this case study, if we can call it that, because (i) it is the best-known truly advanced rewriting property, (ii) proving standardisation formally turns out to be far from as straightforward as some of the clever informal proofs might have us believe, and (iii) it allows us to account for the full range of common problems in the area and to showcase the methods we have developed to overcome them [15]. We hope and trust that the exposition will be instructive and provide technical insights of a general nature.

**Our Contributions** We make a clear distinction between the syntactic presentation of a programming language and the intended, *real* version of it, obtained as the  $\alpha$ -equivalence collapse of the syntax. We do so because it is the primitive proof principles of the former that we consistently use in informal proofs,

---

<sup>\*</sup> The article comprises 16 pages plus 6 pages of supplementary material.

whereas it is typically claimed that the obtained results pertain to the latter. This is frequently done to justify a certain amount of sloppiness in dealing with the fundamental formal problems caused by binding, cf. Appendix B. Realising that this prevents the formal substantiation of informal work, we instead use the delineation of the two universes to continue to develop the formal theory required to bridge them. This is our main contribution and it comes in many forms in the article. It basically amounts to the ease with which we conduct the proofs that are spelt out in the paper and the framework that makes it possible.

We also contribute by accounting for the limitations in the expressive power of the primitive proof principles for syntax. This centres around the use of our BCF-predicate, which formalises the so-called Barendregt Variable Convention, and showing that David’s approach to standardisation fails formally, cf. Section 6.

By “syntax” we mean first-order abstract syntax and, so, all our results are formulated and proved by first-order means, with one exception, cf. Section 4. It pertains to our ability to analyse  $\alpha$ -equivalence classes (so as to recurse over them). We contribute by accounting for the identified first vs second-order issues.

A final contribution is concerned with what we call *Some/Any proof theory*. The issues are introduced in Section 2.3 and further pursued in Section 4. The theory is still in its infancy and is very much part of our on-going work.

**Related Work** Apart from the informal work that we introduce tools and techniques to be able to formalise, there are two main lines of related work: McKinna and Pollack [10] and Gabbay and Pitts [5] (and subsequent articles). Both of these present formal proof principles for objects that look like (but are different from)  $\alpha$ -equivalence classes. Our work is the only one that uses first-order abstract syntax, which means we are faced with more details but also benefit from being able to establish properties that appear logically stronger than what the other two frameworks can accomplish and by being able to formally justify existing, informal work [15–17]. Indeed, what is clear at the moment is that we do not know in general how to formally transfer results between the three frameworks<sup>1</sup> and that the frameworks all invoke some-/any-ness albeit in different ways. Clarifying the various foundational issues must be the next step.

**This Paper** To make the formal contributions of this paper accessible, we use some amount of space to introduce the basic concepts and terminology and to account for already published results that we need. Please find this material in Section 2, with substantial supplementary material in Appendices A, B, and C. In Section 3, we formally establish Semi-Standardisation for the real  $\lambda$ -calculus. Semi-Standardisation is used in Sections 4 and 5 to formally establish Mitschke’s and Plotkin’s notions of (full) standardisation for the real  $\lambda$ -calculus. Section 6, on the other hand, shows that David’s approach fails formally. All higher-level proofs are included in full, with some to be found in Appendix D. All the lower-level results, such as substitution, substitutivity, and commutation lemmas, and

<sup>1</sup> For example, the equivalent of Assumption 3 is inconsistent with the Gabbay-Pitts framework, raising questions about the differences in the effectiveness of the set-ups.

their proofs have been suppressed but can be found elsewhere [15]. The full set of proofs would properly amount to around 100 type-set pages. We note that the material contained in Section 4 is new and cannot be found in [15].

## 2 Basics

Notation-wise, we write relations on equivalence classes full-lined,  $\longrightarrow$ , and on terms dashed,  $\dashrightarrow$ . We indicate reflexive, transitive closure by a double-headed arrow,  $\longleftrightarrow$ , and reflexive, transitive, symmetric closure as a long equality,  $\equiv$ . Relation composition is written with infix  $;$ . Specially-formed arrows will be used.

### 2.1 Syntax and Primitive Proof Principles of the $\lambda$ -Calculus

The  $\lambda$ -calculus is the archetypical functional programming language when presented with first-order abstract syntax over variables of the following kind.

**Assumption 1**  $\mathcal{VN}$  is a single-sorted set of objects called variable names.

**Assumption 2**  $\mathcal{VN}$ -equality,  $=_{\mathcal{VN}}$ , is decidable.

**Assumption 3** There exists a total, computable function on finite subsets of  $\mathcal{VN}$ ,  $\text{Fresh}(-) : \mathcal{P}_{\text{fin}}(\mathcal{VN}) \longrightarrow \mathcal{VN}$ , such that:  $\text{Fresh}(\text{VN}) \notin \text{VN}$ .

Concretely, we typically take variables as words over the Latin alphabet although we remain abstract here. We use  $x, y, z$  (possibly annotated) as meta-variables over  $\mathcal{VN}$  and, with an abuse of notation, also as object-level variables.

**Definition 1.**  $A^{\text{var}} ::= \mathcal{VN} \mid A^{\text{var}} A^{\text{var}} \mid \lambda \mathcal{VN}. A^{\text{var}}$

As the  $A^{\text{var}}$ -terms (ranged over by  $e$ , possibly annotated) are inductively defined, they come equipped with a range of primitive proof principles [1, 2]: structural induction, case-splitting, and recursion, and the ability to define relations inductively over the terms and to rule-induct over the result, cf. Appendix A.

### 2.2 Orthonormal Reduction and Renaming-Free Substitution

As shown previously [15–17], the root cause of the hitherto formal inapplicability of structural proof principles has been the use of Curry-style, or *capture-avoiding*, substitution, cf. our lengthy discussion in Appendix B. Consequently, we base our presentation of the  $\lambda$ -calculus on *renaming-free* substitution, cf. Figure 1, and ultimately show that there only really is one  $\lambda$ -calculus, cf. Theorem 1.

**Definition 2.** The capturing binders of free occurrences of  $x$  are:

$$\begin{aligned} \text{Capt}_x(y) &= \emptyset \\ \text{Capt}_x(e_1 e_2) &= \text{Capt}_x(e_1) \cup \text{Capt}_x(e_2) \\ \text{Capt}_x(\lambda y. e) &= \begin{cases} \{y\} \cup \text{Capt}_x(e) & \text{if } x \in \text{FV}(\lambda y. e) \\ \emptyset & \text{otherwise} \end{cases} \end{aligned}$$

$$\begin{array}{c}
y[x := e] = \begin{cases} e & \text{if } x = y \\ y & \text{otherwise} \end{cases} \\
(e_1 e_2)[x := e] = e_1[x := e] e_2[x := e] \\
(\lambda y. e_0)[x := e] = \begin{cases} \lambda y. e_0[x := e] & \text{if } x \neq y \wedge y \notin \text{FV}(e) \\ \lambda y. e_0 & \text{otherwise} \end{cases} \\
\hline
\frac{y \notin \text{Capt}_x(e) \cup \text{FV}(e)}{\lambda x. e \xrightarrow{y}_{i\alpha} \lambda y. e[x := y]} (i\alpha) & \frac{e \xrightarrow{y}_{i\alpha} e'}{\lambda x. e \xrightarrow{y}_{i\alpha} \lambda x. e'} (L_{i\alpha}) \\
\frac{e_1 \xrightarrow{y}_{i\alpha} e'_1}{e_1 e_2 \xrightarrow{y}_{i\alpha} e'_1 e_2} (Al_{i\alpha}) & \frac{e_2 \xrightarrow{y}_{i\alpha} e'_2}{e_1 e_2 \xrightarrow{y}_{i\alpha} e_1 e'_2} (Ar_{i\alpha}) \\
\hline
\frac{\text{Capt}_x(e_1) \cap \text{FV}(e_2) = \emptyset}{(\lambda x. e_1) e_2 \xrightarrow{\beta} e_1[x := e_2]} (\beta) & \frac{e \xrightarrow{\beta} e'}{\lambda x. e \xrightarrow{\beta} \lambda x. e'} (L_\beta) \\
\frac{e_1 \xrightarrow{\beta} e'_1}{e_1 e_2 \xrightarrow{\beta} e'_1 e_2} (Al_\beta) & \frac{e_2 \xrightarrow{\beta} e'_2}{e_1 e_2 \xrightarrow{\beta} e_1 e'_2} (Ar_\beta)
\end{array}$$

**Fig. 1.** Renaming-free substitution and  $\alpha$ - and  $\beta$ -reduction

**Definition 3 (The  $\lambda^{\text{var}}$ -Calculus).** *The terms of the  $\lambda^{\text{var}}$ -calculus are  $\Lambda^{\text{var}}$ , cf. Definition 1. The  $\beta$ - and indexed  $\alpha$ -relations of  $\lambda^{\text{var}}$ :  $\xrightarrow{\beta}$  and  $\xrightarrow{y}_{i\alpha}$  are given in Figure 1. Plain  $\alpha$ -reduction is given as:  $e_1 \xrightarrow{\alpha} e_2 \Leftrightarrow^{\text{def}} \exists y. e_1 \xrightarrow{y}_{i\alpha} e_2$ .*

The predicate guarding our use of substitution,  $\text{Capt}_x(e_1) \cap \text{FV}(e_2) = \emptyset$  coincides with the notion of *is free for*. Our use of the predicate is justified by the following lemma, which says that Curry-style substitution's use of (implicit) renaming can be accounted for by (explicit, renaming-free)  $\alpha$ -reduction.<sup>2</sup>

**Lemma 1.** *Cf. Appendix B and Figure 1:*

$$\begin{array}{c}
e_a[x := e]_{\text{Cu}} = e_b \\
\Downarrow \\
\exists! e'_a. e_a \xrightarrow{\alpha} e'_a \wedge e'_a[x := e] = e_b
\end{array}$$

While the lemma might appear obvious, the interesting fact is that it can be proved by (constructive) structural means [15]. As implicit renaming is involved, this is not so obvious (although the details are out-of-scope of this paper). The lemma implies that the various definitions of the  $\lambda$ -calculus are closely related.

**Lemma 2.**  $(\xrightarrow{\alpha} \subseteq \xrightarrow{\alpha}_{\text{Cu}} \subseteq \xrightarrow{\alpha} \wedge \xrightarrow{\beta} \subseteq \xrightarrow{\beta}_{\text{Cu}} \subseteq \xrightarrow{\beta}) \wedge (\xrightarrow{\beta} \subseteq \xrightarrow{\beta}_{\text{Cu}} \subseteq \xrightarrow{\beta})$

<sup>2</sup> For precise definitions, please see Appendix B; see [15] for the details.

$$\boxed{
\begin{array}{c}
\frac{}{x \dashrightarrow_{\beta} x} \quad \frac{e \dashrightarrow_{\beta} e'}{\lambda x.e \dashrightarrow_{\beta} \lambda x.e'} \quad \frac{e_1 \dashrightarrow_{\beta} e'_1 \quad e_2 \dashrightarrow_{\beta} e'_2}{e_1 e_2 \dashrightarrow_{\beta} e'_1 e'_2} \\
\frac{e_1 \dashrightarrow_{\beta} e'_1 \quad e_2 \dashrightarrow_{\beta} e'_2 \quad \text{FV}(e'_2) \cap \text{Capt}_x(e'_1) = \emptyset}{(\lambda x.e_1)e_2 \dashrightarrow_{\beta} e'_1[x := e'_2]} \quad (\beta'')
\end{array}
}$$

**Fig. 2.** The parallel  $\beta$ -relation for  $\lambda^{\text{var}}$

The Cu-indexed relations use Curry-style substitution, cf. Appendix B.

**Definition 4 (The Real  $\lambda$ -Calculus).**

- $\Lambda \stackrel{\text{def}}{=} \Lambda^{\text{var}} / \equiv_{\alpha}$
- $[-] \stackrel{\text{def}}{=} \Lambda^{\text{var}} \longrightarrow \Lambda$   
 $e \mapsto \{e' \mid e \equiv_{\alpha} e'\}$
- $[e_1] \longrightarrow_{\beta} [e_2] \Leftrightarrow^{\text{def}} e_1 \equiv_{\alpha}; \dashrightarrow_{\beta}; \equiv_{\alpha} e_2$

We see that  $\beta$ -reduction is a unique notion, albeit non-uniquely axiomatised.

**Theorem 1.**  $[e] \longrightarrow_{\beta} [e'] \Leftrightarrow e \dashrightarrow_{\alpha, \beta} e' \Leftrightarrow e \dashrightarrow_{\alpha \text{Cu}, \beta \text{Cu}} e' \Leftrightarrow [e] \longrightarrow_{\beta \text{Hi}} [e']$

The Hi-index stands for Hindley’s use of Curry’s relations to define reduction on  $\alpha$ -equivalence classes [6], cf. Appendix B. The choice of which  $\lambda$ -calculus presentation to work with therefore comes down to a choice of proof principles (where our presentation wins hands-down). On a related note, we see that we can define more expressive relations than just  $\dashrightarrow_{\beta}$  such as, e.g., parallel reduction.

**Definition 5.** *Figure 2 defines the parallel  $\beta$ -relation. We also define:*

$$[e] \dashrightarrow_{\beta} [e'] \Leftrightarrow^{\text{def}} e \equiv_{\alpha}; \dashrightarrow_{\beta}; \equiv_{\alpha} e'$$

**Proposition 1.**  $(\dashrightarrow_{\beta} \subseteq \dashrightarrow_{\beta} \subseteq \dashrightarrow_{\beta}) \wedge (\longrightarrow_{\beta} \subseteq \dashrightarrow_{\beta} \subseteq \longrightarrow_{\beta})$

### 2.3 Resolving $\alpha$ -Equivalence

We will need to be able to reduce  $\alpha$ -equivalence to  $\Lambda^{\text{var}}$ -equality and, so, we briefly recount an earlier proof of this fact (which makes it clear that the property is not as trivial as could be expected) [15, 17].<sup>3</sup> Using  $\varepsilon$  for the empty vector and writing  $\vec{z}_i$  for the vector of the variable names  $z_i$ , Figure 3 shows how to completely fresh-name a  $\Lambda^{\text{var}}$ -term, with fresh-naming defined as follows.

**Definition 6.**  $e \dashrightarrow_{\alpha_0} e' \Leftrightarrow^{\text{def}} \exists z.z \notin \text{Var}(e) \wedge e \dashrightarrow_{i\alpha} e'$

Fresh-naming  $\alpha$ -reduction is relevant because it commutes with all notions of  $\beta$ -reduction (with the proviso that the resulting  $\alpha$ -reductions need not be fresh-naming). As for complete fresh-naming, we note that it is structural-recursive.

<sup>3</sup> We follow [15] as it improves on [17] by using inside-out, rather than outside-in, complete fresh-naming; doing so guarantees computability primitively.

$$\frac{x \xrightarrow{\varepsilon} \mathfrak{A}_{i\alpha_0^{i_0}} x \quad \frac{e \xrightarrow{\vec{z}_i} \mathfrak{A}_{i\alpha_0^{i_0}} e' \quad z \notin \{z_i\} \quad \{z, z_i\} \cap \text{Var}(\lambda x.e) = \emptyset}{\lambda x.e \xrightarrow{\vec{z}_i} \mathfrak{A}_{i\alpha_0^{i_0}} \lambda z.e'[x := z]}}}{e_1 \xrightarrow{\vec{x}_i} \mathfrak{A}_{i\alpha_0^{i_0}} e'_1 \quad e_2 \xrightarrow{\vec{y}_i} \mathfrak{A}_{i\alpha_0^{i_0}} e'_2 \quad \{x_i\} \cap \{y_i\} = \{x_i\} \cap \text{Var}(e_2) = \{y_i\} \cap \text{Var}(e_1) = \emptyset} e_1 e_2 \xrightarrow{\vec{x}_i \vec{y}_i} \mathfrak{A}_{i\alpha_0^{i_0}} e'_1 e'_2}$$

**Fig. 3.** Indexed, one-step, inside-out, complete fresh-naming

$$\frac{\frac{\text{BCF}(e) \quad x \notin \text{BV}(e)}{\text{BCF}(\lambda x.e)} \quad \text{BCF}(x)}{\text{BCF}(e_1) \quad \text{BCF}(e_2) \quad \text{Var}(e_1) \cap \text{BV}(e_2) = \text{Var}(e_2) \cap \text{BV}(e_1) = \emptyset} \text{BCF}(e_1 e_2)}$$

**Fig. 4.** A term using distinct variable names is a *Barendregt Conventional Form*.

**Proposition 2.**  $e \xrightarrow{\vec{z}_i} \mathfrak{A}_{i\alpha_0^{i_0}} -$ , cf. Figure 3, is total, computable, and functional when the  $z_i$  are fresh with respect to  $e$ , distinct, and of the right amount.

Apart from this, the relevant properties of complete fresh-naming are as follows.

**Lemma 3.** Cf. Figures 3 and 4:  $e \xrightarrow{\vec{z}_i} \mathfrak{A}_{i\alpha_0^{i_0}} e' \Rightarrow \text{BCF}(e') \wedge e \xrightarrow{\alpha} e'$

With that, we can ultimately show that we can resolve  $\alpha$ -equivalence using either *some* or *any* suitable vector of fresh variables. Please refer to Appendix C for our diagram notation.

**Lemma 4.**

$$\left( \exists \vec{z}_i . \text{“}z_i \text{ fresh, distinct, and of right amount”} \wedge \begin{array}{c} \bullet \text{=====} \bullet \\ \searrow \quad \alpha \quad \swarrow \\ \vec{z}_i \quad \circ \quad \vec{z}_i \end{array} \right)$$

$\wedge$

$$\left( \forall \vec{z}_i . \text{“}z_i \text{ fresh, distinct, and of right amount”} \Rightarrow \begin{array}{c} \bullet \text{=====} \bullet \\ \searrow \quad \alpha \quad \swarrow \\ \vec{z}_i \quad \circ \quad \vec{z}_i \end{array} \right)$$

Unfortunately, neither of these is directly provable. If  $\vec{z}_i$  is existentially quantified, we are faced with two main problems:

**Non-Extensionality:** Having computed  $e \xrightarrow{\vec{z}_i} \mathfrak{A}_{i\alpha_0^{i_0}}$  does not allow us to fresh-name  $\lambda x.e$  because  $x$  and the  $z_i$  can conflict (and this *is* a real problem).

**Lacking Transitivity:** The transitive case of the property’s proof is blocked:

$$\begin{array}{c}
 M_1 \text{ ===== } M_2 \text{ ===== } M_3 \\
 \begin{array}{ccc}
 \nearrow & \alpha & \nwarrow \\
 \vec{x}_i & N_1 & \vec{x}_i \\
 \nwarrow & & \nearrow \\
 & & N_2 \\
 \nearrow & \alpha & \nwarrow \\
 \vec{y}_i & N_2 & \vec{y}_i
 \end{array}
 \end{array} \tag{1}$$

We cannot relate  $N_1$  and  $N_2$  because  $\vec{x}_i$  and  $\vec{y}_i$  are merely given non-constructively by the induction hypothesis and need not coincide.<sup>4</sup>

If  $\vec{z}_i$  is universally quantified, we gain extensionality but still lack transitivity because we would only be able to use any  $\vec{z}_i$  that is fresh with respect to  $M_1$  and  $M_3$ , as allowed, as well as (the arbitrary)  $M_2$ . The insight we need is that the two predicates of Lemma 4 are equivalent (irrespective of their truth-status).

**Lemma 5.**

$$\begin{array}{c}
 \left( \exists \vec{z}_i . \text{“}z_i \text{ fresh, distinct, and of right amount”} \wedge \begin{array}{c} \bullet \text{ ===== } \bullet \\ \nearrow \quad \alpha \quad \nwarrow \\ \vec{z}_i \quad \circ \quad \vec{z}_i \end{array} \right) \\
 \Updownarrow \\
 \left( \forall \vec{z}_i . \text{“}z_i \text{ fresh, distinct, and of right amount”} \Rightarrow \begin{array}{c} \bullet \text{ ===== } \bullet \\ \nearrow \quad \alpha \quad \nwarrow \\ \vec{z}_i \quad \circ \quad \vec{z}_i \end{array} \right)
 \end{array}$$

**Proof** See Appendix D. □

The proof we give is interesting in its own right, not least because it is *zero-knowledge*, which means that it has repercussions for computability considerations about the employed fresh variable names [15, 17]. Having established the equivalence means that we repeatedly can change perspective in the proof of Lemma 4 and, thus, side-step the listed problems. We refer to the general form of this phenomenon and its consequences as *Some/Any proof theory*, cf. Section 4.

In conclusion, we remark that all aspects of this section have been constructive, which means that we actually have an algorithm for resolving  $\alpha$ -equivalence.

**Theorem 2.**  $\text{==}_\alpha$  is decidable by means of  $\overline{\text{--}}_{i\alpha_0^i}$ .

**Proof** Fairly straightforward from Lemmas 3 and (the lower conjunct of) 4. □

### 3 Semi-Standardisation

This section establishes so-called semi-standardisation, which says that “outermost” redexes can be contracted before “inner” redexes [11]. We pursue a slight adaptation of Takahashi’s adaptation [14] of Mitschke’s proof [11]. Instead of

<sup>4</sup> We note that the property thus would not help in deciding  $\alpha$ -equivalence, either.

$$\frac{\text{Capt}_x(e_1) \cap \text{FV}(e_2) = \emptyset}{(\lambda x.e_1)e_2 \dashrightarrow_{\beta^{\text{wh}}} e_1[x := e_2]} (\beta^{\text{wh}}) \frac{e_1 \dashrightarrow_{\beta^{\text{wh}}} e'_1}{e_1 e_2 \dashrightarrow_{\beta^{\text{wh}}} e'_1 e_2} (@^{\text{wh}})$$

**Fig. 5.** Weak-head  $\beta$ -reduction

$$\frac{\frac{e_1 \dashrightarrow_{\beta^{\text{I}}} e'_1}{e_1 e_2 \dashrightarrow_{\beta^{\text{I}}} e'_1 e_2} (@^{\text{I}}_1) \quad \frac{e_2 \dashrightarrow_{\beta} e'_2}{e_1 e_2 \dashrightarrow_{\beta^{\text{I}}} e_1 e'_2} (@^{\text{I}}_2) \quad \frac{e \dashrightarrow_{\beta} e'}{\lambda x.e \dashrightarrow_{\beta^{\text{I}}} \lambda x.e'} (\lambda^{\text{I}})}{\frac{}{x \dashrightarrow_{\beta^{\text{I}}} x} (\text{Var}^{\text{I}}) \quad \frac{e_1 \dashrightarrow_{\beta^{\text{I}}} e'_1 \quad e_2 \dashrightarrow_{\beta} e'_2}{e_1 e_2 \dashrightarrow_{\beta^{\text{I}}} e'_1 e'_2} (@^{\text{I}}_{\parallel}) \quad \frac{e \dashrightarrow_{\beta} e'}{\lambda x.e \dashrightarrow_{\beta^{\text{I}}} \lambda x.e'} (\lambda^{\text{I}}_{\parallel})}{} (\text{Var}^{\text{I}}_{\parallel})$$

**Fig. 6.** Inner and parallel inner  $\beta$ -reduction

head and a corresponding notion of inner reduction, we base the proof on weak-head reduction [10].<sup>5</sup> As stressed in [10], the crucial property of the inner relations is that they preserve the outermost syntax constructor of the term they reduce, thus enabling us to infer what rule has been applied to arrive at a given end-term and vice versa, cf. Lemma 16.

**Definition 7.** *Weak-head  $\beta$ -reduction,  $\dashrightarrow_{\beta^{\text{wh}}}$ , is defined in Figure 5. The corresponding (strong) inner,  $\dashrightarrow_{\beta^{\text{I}}}$ , and parallel inner,  $\dashrightarrow_{\beta^{\text{I}}}$ ,  $\beta$ -relations are defined in Figure 6. We also have real versions of these relations.*

$$\begin{aligned} [e] \longrightarrow_{\beta^{\text{wh}}} [e'] &\Leftrightarrow^{\text{def}} e ==_{\alpha}; \dashrightarrow_{\beta^{\text{wh}}}; ==_{\alpha} e' \\ [e] \longrightarrow_{\beta^{\text{I}}} [e'] &\Leftrightarrow^{\text{def}} e ==_{\alpha}; \dashrightarrow_{\beta^{\text{I}}}; ==_{\alpha} e' \\ [e] \dashrightarrow_{\beta^{\text{I}}} [e'] &\Leftrightarrow^{\text{def}} e ==_{\alpha}; \dashrightarrow_{\beta^{\text{I}}}; ==_{\alpha} e' \end{aligned}$$

**Proposition 3.**  $(\dashrightarrow_{\beta^{\text{I}}} \subseteq \dashrightarrow_{\beta^{\text{I}}} \subseteq \dashrightarrow_{\beta^{\text{I}}}) \wedge (\longrightarrow_{\beta^{\text{I}}} \subseteq \dashrightarrow_{\beta^{\text{I}}} \subseteq \longrightarrow_{\beta^{\text{I}}})$

While the ensuing lemma might seem unassuming, it, in fact, establishes the crucial proof principle that we can case-split on a  $\beta$ -reduction step and consider a weak-head and an inner case.

**Lemma 6.**  $(\dashrightarrow_{\beta} = \dashrightarrow_{\beta^{\text{I}} \beta^{\text{wh}}}) \wedge (\longrightarrow_{\beta} = \longrightarrow_{\beta^{\text{I}} \beta^{\text{wh}}})$

We point out that, although  $\dashrightarrow_{\beta^{\text{I}}}$  and  $\dashrightarrow_{\beta^{\text{wh}}}$  cannot contract the same redex, it is not the case that  $\dashrightarrow_{\beta^{\text{I}}} \cap \dashrightarrow_{\beta^{\text{wh}}} = \emptyset$ . The problem is, e.g., that some terms, like  $\Omega\Omega$ , both  $\beta^{\text{I}}$ - and  $\beta^{\text{wh}}$ -reduce to themselves. This is relevant because the first property does not easily lend itself to a formal proof while the second

<sup>5</sup> [10] is based on an ad hoc set up, which does not easily relate to simple syntax and structural proof principles and, thus, is not immediately relevant to this article.

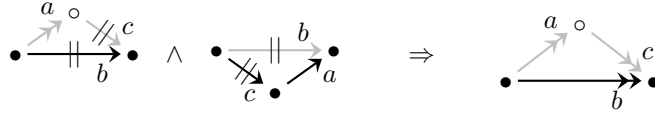
$$\begin{array}{c}
\frac{y \notin \text{Var}(e)}{\lambda x.e \xrightarrow{y} \lambda y.e[x := y]} (i\alpha_1) \quad \frac{e \xrightarrow{y} e' \quad y \neq x}{\lambda x.e \xrightarrow{y} \lambda x.e'} (L\alpha_1) \\
\\
\frac{e_1 \xrightarrow{y} e'_1 \quad y \notin \text{FV}(e_2)}{e_1 e_2 \xrightarrow{y} e'_1 e_2} (A1\alpha_1) \quad \frac{e_2 \xrightarrow{y} e'_2 \quad y \notin \text{FV}(e_1)}{e_1 e_2 \xrightarrow{y} e_1 e'_2} (Ar\alpha_1) \\
\\
\hline
\frac{}{\text{wBCF}(x)} \quad \frac{\text{wBCF}(e) \quad x \notin \text{BV}(e)}{\text{wBCF}(\lambda x.e)} \\
\\
\frac{\text{wBCF}(e_1) \quad \text{wBCF}(e_2) \quad \text{FV}(e_1) \cap \text{BV}(e_2) = \text{FV}(e_2) \cap \text{BV}(e_1) = \emptyset}{\text{wBCF}(e_1 e_2)}
\end{array}$$

**Fig. 7.** The weakly fresh-naming  $\alpha_1$ -relation and the wBCF-predicate

would, if true. The lack of formal disjointness is relevant to Theorem 3.

The parallel relations are used (i) to allow us to introduce key lemmas that are simpler than but imply the actual property we are after and (ii) to introduce designated proof principles for those key lemmas. To illustrate this, we present the following abstract rewriting result that is implicit in [14]. The  $a$ -relation will be weak-head reduction, the  $b$ -relation will be ordinary  $\beta$ -reduction, while the  $c$ -relation will be inner reduction.

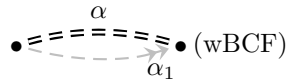
**Lemma 7.** Assume  $\xrightarrow{a} \subseteq \xrightarrow{b} \subseteq \xrightarrow{c}$  and  $\xrightarrow{c} \subseteq \xrightarrow{a} \subseteq \xrightarrow{b}$ .



Before proving the two assumed properties in the above result for the present case, we will introduce a weak notion of fresh-naming that will turn out to be needed. Identical binders are permitted in adjacent but not in nested positions in an abstract syntax tree and free and bound variables are disjoint.

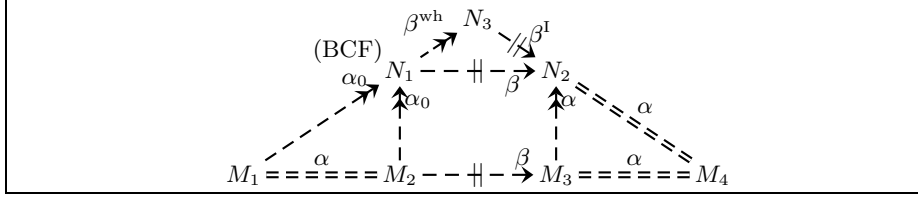
**Definition 8.** The weakly fresh-naming  $\xrightarrow{\alpha_1}$ -relation and the corresponding wBCF-predicate are defined in Figure 7.

**Lemma 8.**



Our main interest in the  $\alpha_1$ -relation and the wBCF-predicate is the following result from which the notions, in fact, are derived.

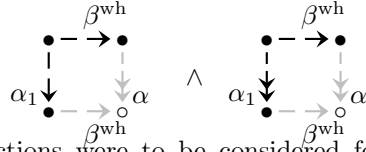
**Lemma 9.**  $e \xrightarrow{\beta^I} e' \wedge \text{BCF}(e) \Rightarrow \text{wBCF}(e')$



**Fig. 8.** The decomposition of parallel steps and  $\alpha$ -equivalence

In the next property, it is not crucial that the  $\beta$ -relation is restricted to weak-head redexes. The point is, rather, that only one  $\beta$ -contraction is performed.

**Lemma 10.**

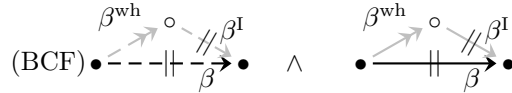


If several  $\beta$ -contractions were to be considered for  $\alpha_1$ -commutativity, we would immediately run into the problem that the end-term below is stuck.

$$(\lambda x. \lambda y. xy)(\lambda z. \lambda y. z) \dashrightarrow_{\beta} \lambda y. (\lambda z. \lambda y. z)y$$

We can now establish the relevant versions of the premises of Lemma 7.

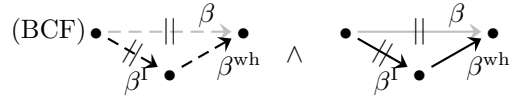
**Lemma 11.**



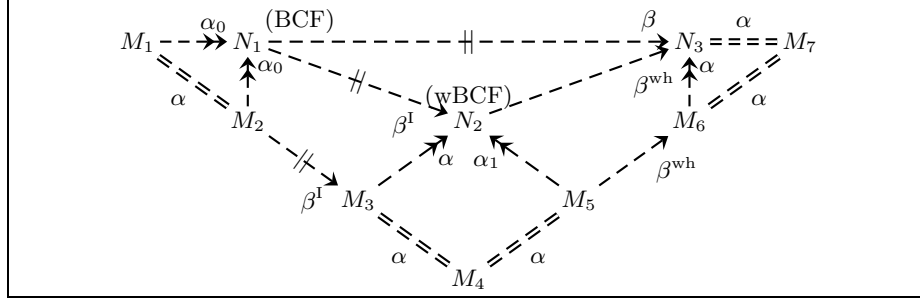
**Proof** The left-most conjunct is proved by rule induction in  $\dashrightarrow_{\beta}$ . The right-most conjunct is proved in Figure 8. The proof proceeds from the (arbitrary)  $M_i$ s and, first, has  $N_1$  introduced by Lemmas 3 and 4. Next, we introduce  $N_2$  by an unlisted commutation lemma for  $\dashrightarrow_{\alpha_0}$  and  $\dashrightarrow_{\beta}$ ; this lemma holds only for the  $\alpha_0$ -relation (and not  $\alpha$ ,  $\alpha_1$ ). The proof is completed by introducing  $N_3$  according to the left-most conjunct above.  $\square$

The use of BCF-initiality in the left-most conjunct above guarantees that weak-head redexes can be contracted first without waiting for the contraction of an inner redex to eliminate a variable clash.

**Lemma 12.**

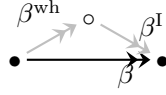


**Proof** The left-most conjunct follows by rule induction in  $\dashrightarrow_{\beta^I}$ . The right-most conjunct is proved in Figure 9 and proceeds from the arbitrary  $M_i$ s. First, we introduce  $N_1$  by Lemmas 3 and 4. Then we introduce  $N_2$  by an unlisted commutation lemma between  $\dashrightarrow_{\alpha_0}$  and  $\dashrightarrow_{\beta^I}$  and we note that  $N_2$  is a wBCF by Lemma 9. In turn, we therefore have  $M_5 \dashrightarrow_{\alpha_1} N_2$  by Lemma 8, which means that we can introduce  $N_3$  by Lemma 10 and, finally, we note that it is  $\dashrightarrow_{\beta}$ -reachable from  $N_1$  according to the left-most conjunct above.  $\square$



**Fig. 9.** The parallelisation of weak-head after inner reduction and  $\alpha$ -equivalence

**Lemma 13 (Semi-Standardisation).**



**Proof** By Lemma 7 applied to Lemmas 11, 12 and Propositions 1, 3.  $\square$

The reason for calling this result semi-standardisation is that a term can have at most one weak-head  $\beta$ -redex, sitting to the far left, if you wish, and that, if it is contracted, it will always be the first in a *standard* reduction sequence. This means that semi-standardisation does some but not all standardising.

## 4 Hereditary Semi-Standardisation

The idea of Mitschke's proof of (full) standardisation [11] is to hereditarily semi-standardise a considered  $\beta$ -reduction sequence. This means (i) contract any weak-head redexes that are to be contracted, (ii) consider the end-term of this and observe that any inner  $\beta$ -reductions it may have will amount to proper  $\beta$ -reductions of its sub-terms, and, so, (iii) repeat the method hereditarily.

The problems with this are (i) that we are not dealing with terms but rather with equivalence classes and they are not immediately associated with any notion of recursion and (ii) we are not actually wanting to recurse over the terms anyway but, rather, over the reflexive, transitive closure of a relation and this certainly is not immediately associated with any kind of recursion principle. In order to do the required recursion we must therefore establish that the relation in question,  $\longrightarrow_{\beta^I}$ , is suitably analytic in a well-founded manner. We shall restrict attention here to analyticity, which we derive from that of  $\Lambda^{\text{var}}$ , through one for  $\Lambda$ .

### $\Lambda$ -Analyticity

**Lemma 14.**

1.  $x ==_{\alpha} e \Rightarrow e = x$
2.  $e_a e_b ==_{\alpha} e \Rightarrow \exists e'_a, e'_b. e = e'_a e'_b \wedge e_a ==_{\alpha} e'_a \wedge e_b ==_{\alpha} e'_b$

3.  $\lambda x.e_a ==_\alpha e \Rightarrow \exists y, e_b. e = \lambda y.e_b$

**Proof** By straightforward reflexive, transitive, symmetric inductions in  $==_\alpha$ , observing that  $--\rightarrow_\alpha$  preserves the outer-most syntax constructor.  $\square$

We note that this first version of analyticity for  $\alpha$ -equivalence classes is rather weak seeing that it does not (and, in general, cannot) relate the bodies of  $\alpha$ -equivalent abstractions. To compensate, we present the following special form.

**Lemma 15.**  $\lambda x.e_a ==_\alpha \lambda x.e_b \Rightarrow e_a ==_\alpha e_b$

**Proof** Consider fresh  $\vec{z}_i$  and apply Lemma 4 to get  $\lambda x.e_v --\mathfrak{A}_{\alpha_0^{\text{io}}} \lambda z_1.e$ , for some  $e$ , with  $v \in \{a, b\}$ . By definition of  $--\mathfrak{A}_{\alpha_0^{\text{io}}}$ , it must be the case that  $e = e'[x := z_1]$ , for some  $e'$  such that  $e_v --\mathfrak{A}_{\alpha_0^{\text{io}}} e'$ . By Lemma 3, we therefore have  $e_a ==_\alpha e' ==_\alpha e_b$ .  $\square$

It is pertinent to stress that the proof uses Some/Any proof theory in the shape of Lemma 4 and that it seems to require it.

**$\longrightarrow_{\beta^I}$ -Analyticity** In order to establish  $\longrightarrow_{\beta^I}$ -analyticity, we use  $E$  to range over  $A$ , i.e., over  $\alpha$ -equivalence classes, making the below property the only second-order predicate in this paper. This works because we can rely on the specific definition of  $--\rightarrow_{\beta^I}$  to help us construct representatives of the  $E$ s.

**Lemma 16.**

1.  $[x] \longrightarrow_{\beta^I} E \Rightarrow E = [x] = \{x\}$
2.  $[e_1 e_2] \longrightarrow_{\beta^I} E \Rightarrow \exists e'_1, e'_2. E = [e'_1 e'_2] \wedge [e_1] \longrightarrow_{\beta^I} [e'_1] \wedge [e_2] \longrightarrow_{\beta} [e'_2]$
3.  $[\lambda x.e] \longrightarrow_{\beta^I} E \Rightarrow \exists e'. E = [\lambda x.e'] \wedge [e] \longrightarrow_{\beta} [e']$

**Proof** The proofs are simple transitive, reflexive inductions, using Lemma 14. We note, in particular, that the transitive case of the third property is virtually trivial as a result of the second-order quantification, cf. Appendix D.  $\square$

**Some-/Any-ness** Analysing Lemma 16, we see that it, in fact, harbours a Some/Any property that could have been used to directly prove a first-order variant of  $\longrightarrow_{\beta^I}$ -analyticity.<sup>6</sup>

**Lemma 17.**

$$\begin{aligned} \exists z. z \notin \text{Var}(e_1 e_2) \wedge [e_1[x := z]] \longrightarrow_{\beta} [e_2[y := z]] \\ \Downarrow \\ \forall z. z \notin \text{Var}(e_1 e_2) \Rightarrow [e_1[x := z]] \longrightarrow_{\beta} [e_2[y := z]] \end{aligned}$$

**Proof** Both properties are equivalent to  $[\lambda x.e_1] \longrightarrow_{\beta^I} [\lambda y.e_2]$ . To see that they are implied from it, pick some/any fresh  $z$  relative to  $\lambda x.e_1$  and  $\lambda y.e_2$ . By  $\alpha$ -reduction, we have  $[\lambda z.e_1[x := z]] \longrightarrow_{\beta^I} [\lambda y.e_2]$ . By Lemma 16, we have an  $e'_2$  such that  $[e_1[x := z]] \longrightarrow_{\beta} [e'_2]$  and  $\lambda z.e'_2 ==_\alpha \lambda y.e_2$ . By choice of  $z$ , we also have  $\lambda y.e_2 --\rightarrow_\alpha \lambda z.e_2[y := z]$ , and thus  $[e'_2] = [e_2[y := z]]$  by Lemma 15,

<sup>6</sup> E.g.,  $[\lambda x.e_1] \longrightarrow_{\beta^I} [\lambda y.e_2] \Rightarrow \exists z. [e_1[x := z]] \longrightarrow_{\beta} [e_2[y := z]]$ , cf. Lemmas 4, 5.

and we are done. To prove that the existential property implies the existence of the considered  $\longrightarrow_{\beta^1}$ -step, it essentially suffices to show the simple fact that  $[e_1] \longrightarrow_{\beta} [e_2] \Rightarrow [\lambda z.e_1] \longrightarrow_{\beta^1} [\lambda z.e_2]$ . To prove that the universal property implies the existential one, use  $\text{Fresh}(e_1 e_2)$ .  $\square$

It is unclear whether  $\longrightarrow_{\beta^1}$ -analyticity can be established by first-order means, only. If it cannot, one might want to consider the expressive power of first-order logic plus Lemma 17 (or other lemmas like it) and this would then be justification for a stand-alone notion of *Some/Any proof theory*. We are currently pursuing the details.

**(Informal) Standardisation** As for standardisation, the end-result of the above is a not-quite-formal (formulation- and substantiation-wise) theorem whose key lemmas, however, are fully formalised. The problem is in defining what “standardisation” means in this case. It is difficult because we do not have formal notions of location of redexes, contraction order, and disjointness of weak-head and inner reduction, cf. discussion after Lemma 6.

**Theorem 3.** *Given  $[e_1] \longrightarrow_{\beta} [e_2]$ , we can repeatedly apply Lemmas 13 and 16 so as to obtain a standard  $\beta$ -reduction sequence (and this process terminates).*

## 5 Absorptive Weak-Head Standardisation

Plotkin [12] defines standardisation as the least contextually-closed relation on terms that enjoys left-absorptivity over weak-head reduction [10].

$$\frac{e \dashrightarrow_{\beta^{\text{wh}}} e' \quad e' \succ_{\dashrightarrow_{\text{P}}} e''}{e \succ_{\dashrightarrow_{\text{P}}} e''} \quad \frac{e_1 \succ_{\dashrightarrow_{\text{P}}} e'_1 \quad e_2 \succ_{\dashrightarrow_{\text{P}}} e'_2}{e_1 e_2 \succ_{\dashrightarrow_{\text{P}}} e'_1 e'_2} \quad \frac{e \succ_{\dashrightarrow_{\text{P}}} e'}{\lambda x.e \succ_{\dashrightarrow_{\text{P}}} \lambda x.e'}$$

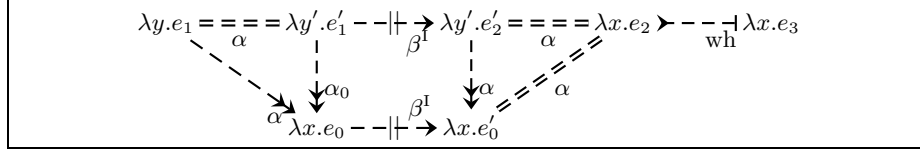
Unfortunately,  $\succ_{\dashrightarrow_{\text{P}}}$  does not standardise  $\beta$ -reduction in the formal sense. To see this, observe first that we can prove what appears to be the key lemma:



The involved relations are limited by being defined on syntax. This means that they must be renaming-free in order to allow for formal provability. In the case of standardisation, this limitation is too restrictive. If, for example, we could generalise the above key lemma to  $\alpha$ -equivalence classes, we could easily establish standardisation by a subsequent reflexive, transitive induction and by observing that  $\succ_{\dashrightarrow_{\text{P}}}$  is reflexive.



But, the generalised version is incorrect as the following counter-example shows.



**Fig. 10.** Name unification for Lemma 19, abstraction case under  $\alpha$ -equivalence

$$(\lambda s.ss)(\lambda x.\lambda y.xy) \dashrightarrow_{\beta} (\lambda x.\lambda y.xy)(\lambda x.\lambda y.xy)$$

The considered end-term can be made into a BCF that standardises:

$$(\lambda x_1.\lambda y_1.x_1y_1)(\lambda x_2.\lambda y_2.x_2y_2) \succ_{\dashv\text{P}} \lambda y_1.\lambda y_2.y_1y_2$$

As the end-term of this step uses the two  $y$  copies nested within each other, we see that the original start term does not  $\succ_{\dashv\text{P}}$ -reduce directly to it.

In order to avoid these problems, we adapt Plotkin's original definition slightly.

**Definition 9.**

$$\frac{[e] \xrightarrow{\beta^{\text{wh}}} [e'] \quad e' \succ_{\dashv\text{wh}} e''}{e \succ_{\dashv\text{wh}} e''} \quad \frac{e_1 \succ_{\dashv\text{wh}} e'_1 \quad e_2 \succ_{\dashv\text{wh}} e'_2}{e_1 e_2 \succ_{\dashv\text{wh}} e'_1 e'_2} \quad \frac{e \succ_{\dashv\text{wh}} e'}{\lambda x.e \succ_{\dashv\text{wh}} \lambda x.e'}$$

The above definition mixes the advantages of being able to define relations inductively over terms with the use of reduction in the real  $\lambda$ -calculus to avoid issues of renaming. Note, however, that it by no means is obvious that this mixture will lend itself to formal reasoning. The proof-technical issue surfaces in the abstraction case of the proof of Lemma 19.

**Lemma 18.**

$$\begin{array}{ccc} \bullet & \xrightarrow{\beta^{\text{wh}}} & \circ \\ \beta^{\text{I}} \downarrow & & \downarrow \beta^{\text{I}} \\ \bullet & \xrightarrow{\beta^{\text{wh}}} & \bullet \end{array}$$

**Proof** The property is established as part of the proof of (Takahashi's) Lemma 7, given that we have Lemmas 11, 12 and Propositions 1 and 3.  $\square$

The key technical lemma in the present standardisation proof development is the following *absorption* property.

**Lemma 19.**  $[e_1] \dashrightarrow_{\beta^{\text{I}}} [e_2] \wedge e_2 \succ_{\dashv\text{wh}} e_3 \Rightarrow e_1 \succ_{\dashv\text{wh}} e_3$

**Proof** By rule induction in  $\succ_{\dashv\text{wh}}$ ; the only interesting case is for abstraction.

**Abstraction:** We are considering the following situation.

$$\lambda y.e_1 \equiv_{\alpha} \lambda y'.e'_1 \dashrightarrow_{\beta^{\text{I}}} \lambda y'.e'_2 \equiv_{\alpha} \lambda x.e_2 \succ_{\dashv\text{wh}} \lambda x.e_3$$

By Definition 7 and the case, we have  $e'_1 \dashrightarrow_{\beta} e'_2$  and  $e_2 \succ_{\dashrightarrow_{\text{wh}}} e_3$ . In case  $y' = x$ , we have from Lemma 15 that  $e'_2 =_{\alpha} e_2$ , which means that we are considering  $[e'_1] \dashrightarrow_{\beta} [e'_2] \succ_{\dashrightarrow_{\text{wh}}} e_3$ , so to speak. From Lemma 11, we thus have:  $[e'_1] \dashrightarrow_{\beta^{\text{wh}}} [e''_1] \dashrightarrow_{\beta^i} [e'_2] \succ_{\dashrightarrow_{\text{wh}}} e_3$ . An application of the I.H. and an invocation of the wh-prefix rule will then give us that  $e'_1 \succ_{\dashrightarrow_{\text{wh}}} e_3$  and we have  $\lambda x.e'_1 \succ_{\dashrightarrow_{\text{wh}}} \lambda x.e_3$  by the abstraction rule. A final (reflexive) application of the wh-prefix rule thus finishes the case:  $\lambda y.e_1 \succ_{\dashrightarrow_{\text{wh}}} \lambda x.e_3$ . Unfortunately, we are not guaranteed that  $y' = x$ . Instead, Figure 10 shows how to overcome this. Based on the upper line, we rewrite  $\lambda y'.e'_1$  to (the BCF)  $\lambda x.e_0$  by, e.g., applying the lower conjunct of Lemma 4 twice. The commuting square involving  $\lambda x.e'_0$  exists by the unlisted  $\dashrightarrow_{\beta^i} / \dashrightarrow_{\alpha_0}$ -commutativity lemma that we also invoked previously and we are done as shown above.  $\square$

**Theorem 4.**  $[e_1] \dashrightarrow_{\beta} [e_2] \Rightarrow e_1 \succ_{\dashrightarrow_{\text{wh}}} e_2$

**Proof** By reflexive, left-transitive induction in  $\dashrightarrow_{\beta}$ . The reflexive case is a straightforward structural induction. The left-transitive case follows by an I.H.-application followed by a case-split on the considered  $\dashrightarrow_{\beta}$ -step, cf. Lemma 6. In case of  $\dashrightarrow_{\beta^{\text{wh}}}$ , we are done by definition of  $\succ_{\dashrightarrow_{\text{wh}}}$ . In case of  $\dashrightarrow_{\beta^i}$ , we are done by Lemma 19, cf. Proposition 3.  $\square$

We see that this theorem, unlike Theorem 3, is fully formal by way of the inductive definition of  $\succ_{\dashrightarrow_{\text{wh}}}$ , which formalises what “standard” means.

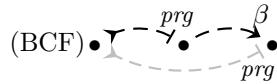
## 6 (Failing) Progression Standardisation

An alternative proof development for standardisation was proposed by David [4] and further considered in [7–9]. Informally, the key technical point is to define a standardising relation that contracts terms as follows, cf. [7, 9]:

$$\frac{(..(e[x := e_0]e_1)..)e_k \succ_{\dashrightarrow_{\text{prg}}} e'}{(..((\lambda x.e)e_0)e_1..)e_k \succ_{\dashrightarrow_{\text{prg}}} e'}$$

This ensures that contraction *progresses* from left-to-right while at the same time allowing newly created redexes to be contracted. Other rules allow redexes not to be contracted and provide for contextual closure. The method fails on the equivalent result to Lemma 19, i.e., left-absorptivity will hold on syntax up-to BCF-initiality but will not generalise to  $\alpha$ -equivalence classes (with the same counter-example) . As an alternative, consider instead right-absorptivity.

**Non-Lemma 1**



However, this property is not even true with BCF-initiality. For example:

$$(\lambda s.ss)(\lambda x.(\lambda y.xy)z) \succ_{\dashrightarrow_{\text{prg}}} (\lambda y.(\lambda x.xz)y)z \dashrightarrow_{\beta} (\lambda y.yz)z$$

The problem here is the (implicitly) last standardisation step, which amounts to the contraction of the redex involving the inner  $y$ -abstraction below.

$$(\lambda y.(\lambda x.(\lambda y.xy)z)y)z$$

This is the point where the considered  $\dashrightarrow_{\beta}$ -step must be inserted but that is not possible because of a clash with the inner  $y$ -abstraction. The difference between right- and left-absorptivity is that the universal quantification over  $\succ\text{-}\dashrightarrow_{\text{prg}}$  covers far fewer steps in the latter case than in the former.

In the previous section, we overcame these problems by mixing real reduction with inductive definability over syntax. That is not possible in the present case because here contraction is built in to the inductive rules, whereas in Plotkin's proof, contraction was handled by a stand-alone relation. David's and Plotkin's approaches are essentially the same at the outermost level in that they only differ in how closely the standardising relation integrates contraction.<sup>7</sup>

## 7 Conclusion

We have accounted comprehensively for what it takes to formalise Mitschke's and Plotkin's proofs of standardisation and shown that David's proof fails from a formal perspective. The key technical difficulty is the interaction between syntactically presented binding and structural proof principles. In order to get the proofs to work, we carefully choose when to work with syntax and when to map the obtained results over to  $\alpha$ -equivalence classes and finish the proof there. The cross-over point is basically dictated by the point where we invoke abstract-rewriting results. Our main contribution is to show-case the various standard forms this can take. We have also initiated a more foundational study of (the first-order principles of) structural induction, recursion, etc.. We identified the addition of first-order some/any properties that are not first-order provable as a way of bridging the gap between first and higher-order theories.

## A Primitive Proof Principles for $\Lambda^{\text{var}}$

**Structural Induction** In order to prove properties about  $\Lambda^{\text{var}}$ -terms, we can proceed by means of *structural induction*, mimicking the structure of the terms:

$$\frac{\forall x.P(x) \quad \forall e_1, e_2.P(e_1) \wedge P(e_2) \Rightarrow P(e_1 e_2) \quad \forall x, e.P(e) \Rightarrow P(\lambda x.e)}{\forall e.P(e)}$$

Informally, structural induction says that, to prove a  $\lambda$ -term property, it suffices to establish that all syntax constructors preserve the property. The rationale behind this is that  $\Lambda^{\text{var}}$  is the least fix-point of the equation in Definition 1, which means that only terms that are constructed exclusively by the use of the given clauses exist.

<sup>7</sup> They differ internally, e.g., by David's proof not using semi-standardisation.

**Structural Case-Splitting** The used syntax constructors are (implicitly) unique and, so, it is possible to *case-split* on terms; with the  $T_i$  in some meta-language:

$$\begin{array}{l} \text{case } e \text{ of} \quad x \Rightarrow T_1(x) \\ \quad \quad \quad | \quad e_1 e_2 \Rightarrow T_2(e_1, e_2) \\ \quad \quad \quad | \quad \lambda x. e_0 \Rightarrow T_3(x, e_0) \end{array}$$

Case-splitting amounts to *analyticity* of the terms and, more specifically, allows us to refer unambiguously to the *sub-term(s)* of a term.

**Structural Recursion** *Structural recursion*, in turn, allows us to define  $\Lambda^{\text{var}}$ -functions by making recursive calls on sub-terms, only:

$$\begin{aligned} f(x) &= T_1(x) \\ f(e_1 e_2) &= T_2(f(e_1), f(e_2)) \\ f(\lambda x. e) &= T_3(x, f(e)) \end{aligned}$$

A function that is defined by structural recursion is well-defined by construction (if the  $T_i$  are): it is functional because the clauses are disjoint, computable by virtue of term well-foundedness, and total because the case-splitting is exhaustive. We can, e.g., define the functions that compute the *free* and *bound variables* in a term, i.e., the variable names that do not occur inside a  $\lambda$ -abstraction of themselves and those that do.

**Definition 10.**

$$\begin{aligned} \text{FV}(y) &= \{y\} & \text{BV}(y) &= \emptyset \\ \text{FV}(e_1 e_2) &= \text{FV}(e_1) \cup \text{FV}(e_2) & \text{BV}(e_1 e_2) &= \text{BV}(e_1) \cup \text{BV}(e_2) \\ \text{FV}(\lambda y. e) &= \text{FV}(e) \setminus \{y\} & \text{BV}(\lambda y. e) &= \{y\} \cup \text{BV}(e) \\ \text{Var}(e) &= \text{FV}(e) \cup \text{BV}(e) \end{aligned}$$

**Proposition 4.**  $\text{FV}(-)$ ,  $\text{BV}(-)$ , and  $\text{Var}(-)$  are total, computable functions.

**Definition by Induction** The inductive structure of  $\Lambda^{\text{var}}$  can also be used to axiomatically define relations on terms, such as, for example, equality.

$$\frac{x =_{\mathcal{V}\mathcal{N}} y}{x =_{\Lambda^{\text{var}}} y} \quad \frac{e_1 =_{\Lambda^{\text{var}}} e'_1 \quad e_2 =_{\Lambda^{\text{var}}} e'_2}{e_1 e_2 =_{\Lambda^{\text{var}}} e'_1 e'_2} \quad \frac{x =_{\mathcal{V}\mathcal{N}} y \quad e =_{\Lambda^{\text{var}}} e'}{\lambda x. e =_{\Lambda^{\text{var}}} \lambda y. e'}$$

As presented, equality is a relation that comes associated with a notion of *rule induction*. Rule induction allows us to prove properties about the relation by considering the given rules as induction cases but only comes with a recursion principle in special cases as no notion of case-splitting need exist and well-foundedness is not guaranteed.

**Functional vs. Relational Definitions** In the case of equality, we note (without proof) that it can also be defined as a truth function on  $\Lambda^{\text{var}} \times \Lambda^{\text{var}}$ , e.g.:

$$x =_{\Lambda^{\text{var}}} e = \text{case } e \text{ of } y \Rightarrow x =_{\mathcal{V}\mathcal{N}} y \mid e_1 e_2 \Rightarrow \text{False} \mid \lambda y. e_0 \Rightarrow \text{False}$$

Without providing details, we contrast these two presentations as follows.

**Proposition 5.**

1.  $=_{\Lambda^{\text{var}}}$  (as a function to truth values) is total and computable.
2.  $=_{\Lambda^{\text{var}}}$  (as a relation) is decidable.

**Proof** The first property holds by construction and Assumption 2, with the second following from it — it cannot easily be established in any other way.  $\square$

## B Problems with Naive Reduction and Substitution

The standard presentation of  $\lambda$ -calculus reduction is due to Curry [3] and goes as follows, with  $-[- := -]_{\text{Cu}}$  standing for meta-level, capture-avoiding substitution.

$$\begin{aligned} (\lambda x.e)e' &\dashrightarrow_{\beta\text{Cu}} e[x := e']_{\text{Cu}} \\ \lambda y.e[x := y]_{\text{Cu}} &\dashrightarrow_{\alpha\text{Cu}} \lambda x.e \quad \text{if } y \notin \text{FV}(e) \end{aligned}$$

**Variable Capture** Curry introduced a formalist notion of substitution.

$$\begin{aligned} y[x := e]_{\text{Cu}} &= \begin{cases} e & \text{if } x = y \\ y & \text{otherwise} \end{cases} \\ (e_1e_2)[x := e]_{\text{Cu}} &= e_1[x := e]_{\text{Cu}}e_2[x := e]_{\text{Cu}} \\ (\lambda y.e_0)[x := e]_{\text{Cu}} &= \begin{cases} \lambda y.e_0 & \text{if } x = y \\ \lambda y.e_0[x := e]_{\text{Cu}} & \text{if } \begin{cases} x \neq y \wedge \\ (y \notin \text{FV}(e) \vee \\ x \notin \text{FV}(e_0)) \end{cases} \\ \lambda z.e_0[y := z]_{\text{Cu}}[x := e]_{\text{Cu}} & \text{o/w; } z = \text{Fresh}((e_0e)x) \end{cases} \end{aligned}$$

The last clause is the interesting one. It renames the considered  $y$  into a fresh  $z$  to avoid undesirable variable capture.

**Lacking Well-Definedness** We remark that Curry-style substitution is not well-defined by construction as it is not structural-recursive. The offender is the last clause that applies  $-[x := e]_{\text{Cu}}$  to a term,  $e_0[y := z]_{\text{Cu}}$ , that is not a subterm of  $\lambda y.e_0$  in general. However, while  $e_0[y := z]_{\text{Cu}}$  is not a sub-term of  $\lambda y.e_0$ , it will have the same size as  $e_0$  and we can establish the well-formedness of  $-[- := -]_{\text{Cu}}$  by external means. Alternatively, we can introduce *parallel substitution* [13]. As we wish to distance ourselves from the use of implicit renaming, we shall not pursue the details.

**Variable-Name Indeterminacy** If we commit ourselves to using renaming in substitution, a range of problems follows. Hindley [6] observed, for example, that it becomes impossible to predetermine the variable name that will be used for abstractions.

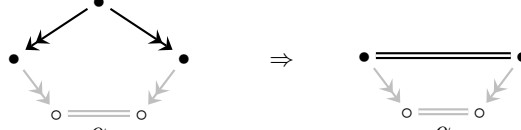
$$\begin{array}{ccc} (\lambda x.(\lambda y.\lambda x.xy)x)y & \xrightarrow{\beta\text{Cu}} & (\lambda y.\lambda x.xy)y \xrightarrow{\beta\text{Cu}} \lambda x.xy \\ & \xrightarrow{\beta\text{Cu}} & (\lambda x.\lambda z.zx)y \xrightarrow{\beta\text{Cu}} \lambda z.zy \end{array}$$

In the lower branch, the innermost  $x$ -abstraction must be renamed to a  $z$ -abstraction, while the upper branch never encounters the variable-name clash and, e.g., confluence escapes us. Hindley instead defined  $\beta$ -reduction on  $\alpha$ -equivalence classes:

$$\begin{aligned} [e] &=^{\text{def}} \{e' \mid e ==_{\alpha} e'\} \\ [e_1] \rightarrow_{\beta\text{Hi}} [e_2] &=^{\text{def}} \exists e'_1 \in [e_1], e'_2 \in [e_2]. e'_1 \dashrightarrow_{\beta\text{Cu}} e'_2 \end{aligned}$$

**Broken Transitivity** Instead of factoring out  $\alpha$ -equivalence, one might reason up to post-fixed name unification. Unfortunately, this leads to a range of problems as far as subsequent uses of abstract rewriting goes, for example, in an attempted adaptation of the well-known equivalence between confluence and the Church-Rosser property.

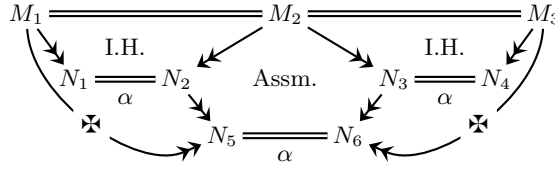
**Non-Lemma 2**



**Proof** [Broken] By reflexive, transitive, symmetric induction in  $=$ .

**Base, Reflexive, Symmetric Cases:** Simple.

**Transitive Case:** Breaks down.



The general problem we encounter is broken transitive induction.

**Non-Analytical  $\alpha$ -Equality** Failing to control limited use of  $\alpha$ -equivalence, one might think that the syntactic version of Hindley's approach could work: that it is possible to state properties about terms up to  $=_{\alpha}$  rather than  $=_{\lambda\text{var}}$ .

**Lemma 20 (Simplified Substitution modulo  $\alpha$ ).**

$$\begin{aligned} e_1 &=_{\alpha} e_2 \wedge x \neq y_i \wedge y_1 \neq y_2 \\ \Downarrow \\ e_1[x_1 := y_1]_{\text{Cu}}[x_2 := y_2]_{\text{Cu}} &=_{\alpha} e_2[x_2 := y_2]_{\text{Cu}}[x_1 := y_1]_{\text{Cu}} \end{aligned}$$

**Proof** [Broken] By structural induction in  $e_1$ .

**Most Cases:** Trivial.

**Last Abstraction Case (simplified):** Breaks down.

$$\begin{aligned} &(\lambda y_1. e)[x_1 := y_1]_{\text{Cu}}[x_2 := y_2] \\ &= \lambda z. e'[x_1 := y_1]_{\text{Cu}}[x_2 := y_2]_{\text{Cu}} \\ &=_{\alpha}^{\times} \lambda z. e'[x_2 := y_2]_{\text{Cu}}[x_1 := y_1]_{\text{Cu}} \\ &= (\lambda z. e')[x_2 := y_2]_{\text{Cu}}[x_1 := y_1]_{\text{Cu}} \end{aligned}$$

The problem above is that  $e$  and  $e'$  are *not* actually  $\alpha$ -equivalent, even if  $\lambda y_1. e$  and  $\lambda z. e'$  are, and the  $=_{\alpha}$ -step is *not* substantiated by the induction hypothesis. The result is certainly correct but unfortunately not directly provable.

## C Commutative Diagrams

We use commutative diagrams in three ways, discernable in the vertex notation.

**Vertices as Terms** With terms, commutative diagrams are reduction scenarios.

**Vertices as  $\bullet$ 's,  $\circ$ 's** Formally, a commutative diagram of this nature is a set of vertices and a set of directed edges between pairs of vertices. Informally, the colour of a vertex ( $\bullet$  vs  $\circ$ ) denotes quantification modes over terms, universal and existential, respectively. A vertex may be guarded by a predicate. Edges are written as the relational symbol they pertain to and are either dark-coloured (black) or light-coloured (gray). Informally, the colour indicates assumed and concluded relations, respectively. An edge connected to a  $\circ$  must be light-coloured. A diagram must be type-correct on domains. A property is read off of a diagram thus:

1. write universal quantifications for all  $\bullet$ 's
2. assume the dark-coloured relations and the guarding predicate for a  $\bullet$
3. conclude the guarded existence of all  $\circ$ s and their relations

The following diagram and property are thus equivalent.

$$\begin{array}{ccc}
 (P) \bullet \rightarrow \bullet & e_1 \rightarrow e_2 \wedge e_1 \rightarrow e_3 \wedge P(e_1) & \\
 \downarrow & \downarrow & \\
 \bullet \rightarrow \circ(Q) & \exists e_4. e_2 \rightarrow e_4 \wedge e_3 \rightarrow e_4 \wedge Q(e_4) & 
 \end{array}$$

**Vertices as  $M$ 's,  $N$ 's** Commutative diagrams express rewriting predicates, e.g.:

“For all terms, such that, . . . , there exist terms, such that, . . . .”

In order to prove these results, we start by writing  $M$ 's for the universally quantified terms and then introduce  $N$ 's from supporting lemmas to eventually substantiate the existence claims. We use  $\boxtimes$  to signify “claimed” existences that are impossible.

## D Proofs

**Lemma 16, 3.**  $[\lambda x.e] \xrightarrow{\beta^I} E \Rightarrow \exists e'. E = [\lambda x.e'] \wedge [e] \xrightarrow{\beta} [e']$

**Proof**

**Base:** We are considering  $\lambda x.e \equiv_{\alpha} \lambda y.e'_1 \dashrightarrow_{\beta^I} \lambda y.e'_2$  by Lemma 14 and definition of  $\dashrightarrow_{\beta^I}$ . A fresh  $\vec{z}_i$ , allows us to can construct the following commuting diagram from upper-left to lower-right, with the introduced terms given in the process:

$$\begin{array}{ccc}
 \lambda x.e_1 \equiv_{\alpha} \lambda y.e'_1 & \dashrightarrow_{\beta^I} & \lambda y.e'_2 \\
 \downarrow \alpha_0 & & \downarrow \alpha \\
 \lambda z_1.e''_1 & \dashrightarrow_{\beta^I} & \lambda z_1.e''_2 \\
 \downarrow \alpha_0 & & \downarrow \alpha \\
 \lambda x.e'''_1 & \dashrightarrow_{\beta^I} & \lambda x.e_2
 \end{array}$$

By Lemma 15 and definition of  $\dashrightarrow_{\beta^I}$ , we thus have  $[e_1] \xrightarrow{\beta^I} [e_2]$ .

**Reflexive:** We are trivially done as  $E = [\lambda x.e]$ .

**Transitive:** We are done by two simple, consecutive applications of the I.H..  $\square$

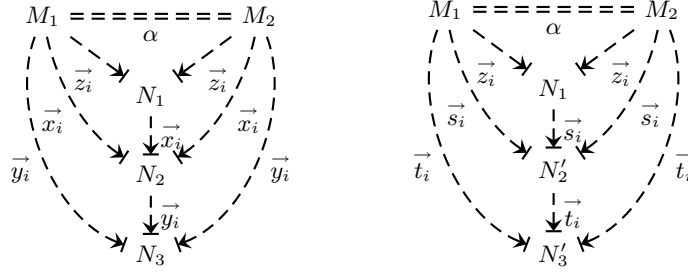
**Lemma 5.**

$$\left( \exists \vec{z}_i . \text{“}z_i \text{ fresh, distinct, and of right amount”} \wedge \begin{array}{c} \bullet \text{=====} \bullet \\ \searrow \quad \swarrow \\ \vec{z}_i \quad \circ \quad \vec{z}_i \end{array} \right) \\ \Downarrow \\ \left( \forall \vec{z}_i . \text{“}z_i \text{ fresh, distinct, and of right amount”} \Rightarrow \begin{array}{c} \bullet \text{=====} \bullet \\ \searrow \quad \swarrow \\ \vec{z}_i \quad \circ \quad \vec{z}_i \end{array} \right)$$

**Proof**

**Case “ $\Uparrow$ ”:** Given  $\alpha$ -equal  $e_1$  and  $e_2$ , identify enough fresh variable names by applying  $\text{Fresh}(-)$  repeatedly and, subsequently, invoke the assumed property (the right-hand side of the equivalence).

**Case “ $\Downarrow$ ”:** Construct these two diagrams by the obvious lemmas and the assumption.



By introducing the  $N_2$  (on the left), we see that we strengthen the “for some”  $\vec{z}_i$  to “for any”  $\vec{x}_i$  that are fresh with respect to  $N_1$ ,  $M_1$ , and  $M_2$ . Unfortunately, this does not suffice as the variables in  $N_1$  are still excluded from consideration. Constructing the  $N'_2$  on the right allows us to use variables,  $\vec{s}_i$ , that are fresh with respect to any specific  $\vec{x}_i$  as well as  $N_1$ ,  $M_1$ , and  $M_2$ . By subsequently adding the layers of  $N_3$  and  $N'_3$ , we can on the one hand use any  $\vec{y}_i$  that are fresh with respect to  $M_1$ ,  $M_2$ , and any specific  $\vec{x}_i$ . On the other hand, we can also use any  $\vec{t}_i$  that are fresh with respect to  $M_1$ ,  $M_2$ , and any specific  $\vec{s}_i$ . As  $\vec{s}_i$  are fresh with respect to any specific  $\vec{x}_i$  by construction, we are thus able to use any variable names,  $\vec{y}_i$  or  $\vec{t}_i$ , that are fresh with respect to just  $M_1$  and  $M_2$ .  $\square$

## References

1. Peter Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, chapter C.7, pages 739–782. North-Holland, Amsterdam, 1977.
2. Rod Burstall. Proving properties of programs by structural induction. *The Computer Journal*, 12, 1967.
3. H. B. Curry and R. Feys. *Combinatory Logic*. North-Holland, Amsterdam, 1958.
4. René David. Une preuve simple de résultats classiques en  $\lambda$  calcul. *Comptes Rendus de l'Académie des Sciences*, 320(11):1401–1406, 1995. Série I.

5. Murdoch Jamie Gabbay and Andrew Pitts. A new approach to abstract syntax involving binders. In Giuseppe Longo, editor, *Proceedings of LICS-14*, pages 214–224. IEEE CS Press, 1999.
6. J. Roger Hindley. *The Church-Rosser Property and a Result in Combinatory Logic*. PhD thesis, University of Newcastle upon Tyne, 1964.
7. Felix Joachimski and Ralph Matthes. Standardization and confluence for a lambda calculus with generalized applications. In Leo Bachmair, editor, *Proceedings of RTA-11*, volume 1833 of *LNCS*. Springer Verlag, 2000.
8. Ryo Kashima. On the standardization theorem for lambda-beta-eta-calculus. In *Proceedings of RPC'01*, 2001. Technical report, the Research Institute of Electrical Communication, Tohoku University, Japan.
9. Ralph Loader. Notes on simply typed lambda calculus. Technical report no. ECS-LFCS-98-381 of LFCS, University of Edinburgh, 1998.
10. James McKinna and Randy Pollack. Some lambda calculus and type theory formalized. *Journal of Automated Reasoning*, 23(3–4), November 1999.
11. Gerd Mitschke. The standardization theorem for  $\lambda$ -calculus. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 25:29–31, 1979.
12. Gordon D. Plotkin. Call-by-name, call-by-value and the  $\lambda$ -calculus. *Theoretical Computer Science*, 1:125–159, 1975.
13. Allen Stoughton. Substitution revisited. *Theoretical Computer Science*, 59(3):317–325, August 1988.
14. Masako Takahashi. Parallel reductions in  $\lambda$ -calculus. *Information and Computation*, 118:120–127, 1995.
15. René Vestergaard. *The Primitive Proof Theory of the  $\lambda$ -Calculus*. PhD thesis, School of Mathematical and Computer Sciences, Heriot-Watt University, 2003.
16. René Vestergaard and James Brotherston. A formalised first-order confluence proof for the  $\lambda$ -calculus using one-sorted variable names (*Barendregt was right after all ... almost*). In Aart Middeldorp, editor, *Proceedings of RTA-12*, volume 2051 of *LNCS*. Springer-Verlag, 2001. A full version is [17].
17. René Vestergaard and James Brotherston. A formalised first-order confluence proof for the  $\lambda$ -calculus using one-sorted variable names. *Information and Computation*, 183(2):212 – 244, 2003. Special edition with selected papers from RTA01.