<table>
<thead>
<tr>
<th>Title</th>
<th>Structural Induction and the $\lambda$-Calculus (Algebra, Logic and Geometry in Informatics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Vestergaard, Rene</td>
</tr>
<tr>
<td>Citation</td>
<td>数理解析研究所講究録 (2003), 1318: 30-45</td>
</tr>
<tr>
<td>Issue Date</td>
<td>2003-05</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/2433/43027">http://hdl.handle.net/2433/43027</a></td>
</tr>
<tr>
<td>Type</td>
<td>Departmental Bulletin Paper</td>
</tr>
<tr>
<td>Textversion</td>
<td>publisher</td>
</tr>
</tbody>
</table>

Kyoto University
Structural Induction and the λ-Calculus

René Vestergaard*

School of Information Science
Japan Advanced Institute of Science and Technology

Abstract. We consider formal provability with structural induction and related proof principles in the λ-calculus presented with first-order abstract syntax over one-sorted variable names. As well as summarising and elaborating on earlier, formally verified proofs (in Isabelle/HOL) of the relative renaming-freeness of β-residual theory and $\beta$-confluence, we also present proofs of $\eta$-confluence, $\delta_\beta$-confluence, the strong weakly-finite $\beta$-development (aka residual-completion) property, residual $\beta$-confluence, $\eta$-over-$\beta$-postponement, and notably $\beta$-standardisation. In the latter case, the known proofs fail in instructive ways. Interestingly, our uniform proof methodology, which has relevance beyond the λ-calculus, properly contains pen-and-paper proof practices in a precise sense. The proof methodology also makes precise what is the full algebraic proof burden of the considered results, which we, moreover, appear to be the first to resolve.

1 Introduction

The use of structural induction and related proof principles for simple syntax (i.e., first-order abstract syntax over one-sorted variable names) is a long-standing and widely-used practice in the programming-language theory community. Unfortunately, at a first, closer inspection it seems that the practice is not formally justifiable because of a need to avoid undue variable capture when performing substitution, thus breaking the syntactic equality underlying structural induction, etc. Even more worrying is the fact that, in spite of substantial efforts in the mechanised theorem-proving community, no formal proof developments (prior to what we report on here) have been able to overcome the problems that are encountered with substitution and go on to successfully employ the proof principles in question. Indeed, and starting with de Bruijn [6], it has become an active research area to define, formalise, and automate alternative syntactic frameworks that, on the one hand, preserve as much of the inherent naturality of simple syntax as possible. At the same time, they are customised to provide suitable induction and recursion principles for any considered language [6–10,12,17,21]. However, by changing the underlying syntactic framework, the algebraic meaning of, e.g., a diamond property also changes, which means that, e.g., confluence as proved and as defined no longer coincide, cf. Lemma 18 and [25].

In the recognition that the above is both unfortunate as far as the formal status of the existing informal literature is concerned and unsatisfactory from a mathematical perspective, we pursue the naive approach in this article (while incorporating the relevant aspects of [24,25]). In particular, we show that it is, indeed, possible to base formal proofs on first-order abstract syntax over one-sorted variable names and hope to convince the reader that, while the technical gap between pen-and-paper and formal proofs is rather large, the conceptual gap is somewhat smaller. Furthermore, we hope that the comprehensive range of applications of the proof methodology that we present here will establish its wider relevance.

1.1 Syntax of the λ-Calculus

The λ-calculus is intended to capture the concept of a function. It does so, first of all, by providing syntax that can be used to express function application and definition:

$$ e ::= x \mid e_1 e_2 \mid \lambda x.e $$

The above, informal syntax says that a λ-term, $e$, is defined inductively as either a variable name, as an application of one term to another, or as a $\lambda$-, or functional, abstraction of a variable name over a term. The variable names, $x$, are typically taken to be, or range over, words over the Latin alphabet. In Section 2, we will review the exact requirements to variable names in an abstract sense. Being based on a simple, inductive definition, λ-terms also come equipped with a range of primitive proof principles [1,3].

Syntactic Equality As a λ-term, $e$, is finite and consists of variable names, the obvious variable-name equality, $=_\forall \alpha$, which exists at least in the case of words over the Latin alphabet, canonically extends to all λ-terms:

$$ x =_\forall \forall \alpha y e_1 =_{\alpha^\forall} e_1' e_2 =_{\alpha^\forall} e_2' x =_\forall \forall \alpha y e =_{\alpha^\forall} e' $$

Structural Induction In order to prove properties about the set of λ-terms, we can proceed by means of structural induction, mimicking the inductive definition of the terms:

$$ \forall x.P(x) \forall e_1, e_2.P(e_1) \wedge P(e_2) \Rightarrow P(e_1 e_2) \forall x, e.P(e) \Rightarrow P(\lambda x.e) $$

$$ \forall e.P(e) $$

* vester@jaist.ac.jp, http://www.jaist.ac.jp/~veste
Structural Case-Splitting As each syntax constructor of the λ-calculus is unique, we see that it is possible to case-split on terms — with $E_1$ in some suitable metalanguage:

\[
\begin{align*}
\text{case } e \text{ of } & \ x \Rightarrow E_1(x) \\
& | e_1e_2 \Rightarrow E_2(e_1, e_2) \\
& | \lambda x.e_0 \Rightarrow E_3(x, e_0)
\end{align*}
\]

Structural Recursion Based on case-splitting and well-foundedness of terms, we can even define functions on λ-terms by means of structural recursion, i.e., by making recursive calls only on the sub-terms of a given constructor:

\[
\begin{align*}
f(x) &= E_1(x) \\
f(e_1e_2) &= E_2(f(e_1), f(e_2)) \\
f(\lambda x.e) &= E_3(x, f(e))
\end{align*}
\]

The above implies that $f$ is well-defined: it is computable by virtue of well-foundedness of terms and total because the definition case-splits exhaustively on λ-terms. As an example application, we define the function that computes the free variables in a term, i.e., the variable names that do not occur inside a λ-abstraction of themselves.

**Definition 1**

\[
\begin{align*}
\text{FV}(y) &= \{y\} \\
\text{FV}(e_1e_2) &= \text{FV}(e_1) \cup \text{FV}(e_2) \\
\text{FV}(\lambda y.e) &= \text{FV}(e) \setminus \{y\}
\end{align*}
\]

**Proposition 2** FV(–) is a total, computable function.

### 1.2 Reduction and Substitution

In order to have λ-abstractions act as functions and not to have too many, e.g., identity functions, amongst other things, we are typically interested in the following relations that can be applied anywhere in a term — their precise form is due to Curry [4].

1. $(\lambda x.e)e' \rightarrow_{\alpha} e[x := e']$
2. $\lambda y.e[x := y] \rightarrow_{\alpha} \lambda x.e$, if $y \not\in \text{FV}(e)$

Our interest in 2., above is the equivalence relation it induces. We denote it by $\equiv_\alpha$, cf. Appendix B, and we will eventually factor it out, as is standard.

**Variable Capture** In his seminal formalist presentation of the λ-calculus [4], Curry defines the above substitution operator, $-[-]_{\alpha}$, essentially as in Figure 1. The last clause is the interesting one. It renames the considered $y$ into the first $z$ that has not been used already.

Consider, for example, the substitution of $z$ for $x$ in the two terms $\lambda z.x$ and $\lambda y.x$. Both terms-as-functions discard their argument. If we simply replace the $x$ in the terms with $x$, the latter would still discard its argument but the former would become the identity function and this discrepancy would lead to inconsistencies.

**Well-Definedness** Of formalist relevance, we remark that Curry-style substitution is not well-defined by construction as the definition does not employ structural.

---

1. While the notion “the first $z$” is trivially well-defined in the present case, the issue is a bit more subtle in a wider context, as we shall see in Section 2.
recursion. The offender is the last clause that applies $- [x := e]$ to a term, $e_0[y := z]$, which is not a subterm of $\lambda y. e_0$ in general. It can be observed that while $e_0[y := z]$ is not a sub-term of $\lambda y. e_0$, it will have the same size as $e_0$ and we can thus establish the well-formedness of $- [- := -]_e$ by external means. Alternatively, we can introduce a more advanced, parallel substitution operator [22]. However, as we eventually will distance ourselves from the use of renaming in substitution, we will do neither but instead refer to Section 2.3 for an alternative derivation of Curry-style substitution.

Variable-Name Indeterminacy Having initially committed ourselves to using renaming in substitution, a range of problems are brought down on us. Hindley [11] observed, for example, that it becomes impossible to predict the variable name used for a given abstraction after reducing, thus putting, e.g., confluence out of reach:

\[
(\lambda x. (\lambda y. \lambda z. x y)) y \rightarrow^\beta (\lambda y. \lambda z. x y) \quad \rightarrow^\beta (\lambda z. x y)
\]

In the lower branch, the innermost $z$-abstraction must be renamed to a $z$-abstraction, while the upper branch never encounters the variable-name clash. Hindley proceeded to define a $\beta$-relation on $\alpha$-equivalence classes that overcomes the above indeterminacy by factoring it out:

\[
[e] = \text{def} \{ e' \mid e \equiv_{\alpha} e' \}
\]

\[
[e_1] \rightarrow_{\beta \mu} [e_2] = \exists e_1' \in [e_1], e_2' \in [e_2]. e_1' \rightarrow_{\beta \mu} e_2'
\]

No relevant proof principles are introduced by this and the approach can not be used in a formal setting as it stands.

Broken Induction Steps Instead of factoring out $\alpha$-equivalence altogether, one could attempt to reason up to post-fixed name unification. Unfortunately, this would lead to a range of unusual situations as far as subsequent uses of abstract rewriting is concerned. An example is the following attempted adaptation of the well-known equivalence of the Church-Rosser property. Please refer to Appendix A for a precise definition of our diagram notation.

Non-Lemma 3

\[
\Rightarrow
\]

Proof (FAILS) By reflexive, transitive, symmetric induction in $=$. Base, Reflexive, Symmetric Cases: Simple.

Variable Monotonicity

Abstract Reasoning

Administrative Proof Layer

Commutativity Lemma

Substitution Lemma

Substitutivity Lemma

Variable Monotonicity

Substitution Sanity

Fig. 3. The proof-layer hierarchy for primitive equational reasoning about the $\lambda$-calculus as simple syntax.

Transitive Case: Breaks down.

\[
M_1 \equiv \alpha \quad \text{I.H.} \quad M_2 \equiv \alpha \quad \text{I.H.} \quad M_3
\]

\[
N_1 \equiv \alpha \quad \text{Assm.} \quad N_2 \equiv \alpha \quad N_3 \equiv \alpha \quad N_4 \equiv \alpha
\]

\[
\alpha \quad \text{N_5} \equiv \alpha \quad \text{N_6} \equiv \alpha
\]

\[
\boldsymbol{\alpha} \equiv \text{FAILS}
\]

Broken $\alpha$-Equality in Sub-Terms Having failed in our attempts to control limited use of $\alpha$-equivalence, one might think that the syntactic version of Hindley's approach, cf. Section 1.2, could work: that it is possible to state all properties about terms up to $\equiv_{\alpha}$ rather than the primitive $\equiv_{A^*}$.

Lemma 4 (Simplified Substitution modulo $\alpha$)

\[
e_1 \equiv_{\alpha} e_2 \land x \neq y_1 \land y_1 \neq y_2
\]

\[
e_1[x_1 := y_1]_e[x_2 := y_2] =_{\alpha} e_2[x_2 := y_2]_e[x_1 := y_1]_e
\]

Proof (FAILS) By structural induction in $e_1$.

Most Cases: Trivial.

Last Abstraction Case (simplified): Breaks down.

\[
(\lambda y^1. e)[x_1 := y_1]_e[x_2 := y_2] = \\
\lambda z. e'[x_1 := y_1]_e[x_2 := y_2]_e[x_1 := y_1]_e
\]

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 $\equiv_{\alpha}$-step can thus not be substantiated by the induction hypothesis. Consider, e.g., $e$ as $y_1$ and $e'$ as $z$. The above result is certainly correct but, unfortunately, not provable with the tools we have at our disposal at the moment.
1.3 This article

The results we are dealing with are mostly well-known and have been addressed in various contexts. Indeed, a number of truly beautiful and concise informal proofs exist; see, in particular, Takahashi [23], whom we owe a great debt. This article, therefore, spends little energy on those parts of the proofs and focuses instead on what it takes to formalise them. There are two key issues: (i) the syntactic properties that can actually be established up to $\equiv_{A^{\text{var}}}$ (as opposed to $\equiv_{\alpha}$, which we have seen to be highly problematic) and (ii) how to generalise these to the algebraic properties we are seeking. The full type-set proofs (roughly 100 pages for the proofs alone) are available from our homepage.

In general, our proofs follow the structure that we present in Figure 3. It is based on nested inductions. The full-coloured arrows mean "is the key lemma for", while the others mean "is used to substantiate side-conditions on lemma applications". The first issue above, (i), is expressed in the addition of the "Variable Monotonicity" proof layer in Figure 3. The second issue, (ii), is entirely accounted for in the "Administrative Proof Layer" in Figure 3.

The proofs underpinning Sections 3 and 4 have been verified in full in Isabelle/HOL (at least in the case of one of the alternatives they present) [24, 25]. By the nature of Figure 3, this means that substantial parts of the other proofs essentially have been verified as well.

Apart from the various technical sections in the body of this paper, the appendix section contains an explanation of our diagram notation (Appendix A) and our other notation (Appendix B) as well as some well-known rewriting results that we use (Appendix C).

2 The $\lambda^{\text{var}}$-Calculus

Having seen that the standard presentations of the $\lambda$-calculus lead to formalist problems, we will now give an alternative presentation that overcomes them. The different presentations differ only in how they lend themselves to provability. Their equational properties are equivalent.

2.1 Formal Syntax

We use $e$'s to range over the inductively built-up set of $\lambda$-terms. The variable names, $\mathcal{V}N$, are generic but must meet certain minimal requirements.

Definition 5 $A^{\text{var}} := \mathcal{V}N \mid A^{\text{var}}A^{\text{var}} \mid \lambda\mathcal{V}N.A^{\text{var}}$

Assertion 6 $\mathcal{V}N$ is a single-sorted set of objects, aka variable names.

Assertion 7 $\mathcal{V}N$-equality, $=_{\mathcal{V}N}$, is decidable.

\[ \text{Definition 9 (The $\lambda^{\text{var}}$-Calculus) The terms of the } \lambda^{\text{var}} \text{-calculus are } A^{\text{var}}, \text{ cf. Definition 5. The (indexed) } \alpha-, \beta-, \text{ and } \eta\text{-reduction relations of } \lambda^{\text{var}}: \neg\rightarrow_{\alpha}, \neg\rightarrow_{\beta}, \text{ and } \neg\rightarrow_{\eta} \text{ are given inductively in Figure 5. The plain } \alpha\text{-relation is:} \]

\[ e \rightarrow_{\alpha} e' \iff \exists y.e \rightarrow_{\eta}^{y} e' \]

Unlike the situation with Curry-style substitution, we see that our notion of substitution is defined by structural recursion and, hence, is well-defined by construction.

Proposition 10 $-[x := e]$ is a total, computable function.

\[ \text{Assertion 8 There exists a total, computable function, } \text{Fresh}(-) : A^{\text{var}} \rightarrow \mathcal{V}N, \text{ such that:} \]

\[ \text{Fresh}(e) \notin \text{FV}(e) \cup \text{BV}(e) \]

The last assertion trivially implies that $\mathcal{V}N$ is infinite. 3

We shall use $x$'s, $y$'s, and $z$'s as meta-variables of $\mathcal{V}N$ and, by a slight abuse of notation, also as actual variable names in terms. We will suppress the $\mathcal{V}N$ suffix on variable-name equality and merely write, e.g., $x = y$.

2.2 Orthonormal Reduction

The key technicality to prevent implicit renaming is our use of a predicate, $\text{Capt}_{x}(e_{1}) \cap \text{FV}(e_{2}) = \emptyset$, cf. Figure 4, which guarantees that no capture takes place in the substitution: $e_{1}[x := e_{2}]$. It coincides with the notion of not free for.

The full-coloured arrows mean "is the key lemma for", while the others mean "is used to substantiate side-conditions on lemma applications". The first issue above, (i), is expressed in the addition of the "Variable Monotonicity" proof layer in Figure 3. The second issue, (ii), is entirely accounted for in the "Administrative Proof Layer" in Figure 3.
The $\beta$- and $\eta$-relations we have presented above do not incur any renaming that could have been performed in a stand-alone fashion by the $\alpha$-relation, thus making them orthogonal. The normality part of our informal orthonormality principle is established by the following property, symmetry of $\rightarrow_{\alpha}$, which implies that the $\alpha$-relation itself is renaming-free.

**Lemma 11**

\[ \eta \rightarrow_{\alpha} \eta \]

**2.3 Curry's $\lambda$-Calculus Decomposed**

In order to assure ourselves that the $\lambda^{\var}$-calculus is indeed the right calculus and partly to test the usefulness of the associated primitive proof principles, we now show how to derive Curry's presentation from ours. First, we show that as far as our use of substitution is concerned, $\lceil x := e \rceil_{\text{Cu}}$ coincides with $\lceil \cdot := \cdot \rceil_{\text{Cu}}$.

**Proposition 12**

\[ \text{Capt}_{\alpha}(e_{a}) \cap \text{FV}(e) = \emptyset \]

\[ \downarrow \]

\[ e_{a}[x := e] = e_{a}[x := e]_{\text{Cu}} \]

**Proof** A straightforward structural induction in $e_{a}$. $\blacksquare$

What might not be obvious is that Curry-style substitution can be shown to decompose into the $\lambda^{\var}$-calculus. In contrast to the structurally flawed Figure 1, Figure 2 introduces a primtively-defined, 4-ary relation that is Curry-style substitution, albeit with no claim of well-definedness.

**Lemma 13**

\[ e_{a}[x := e]_{\text{Cu}} = e'_{a} \]

\[ \downarrow \]

\[ \exists e_{b}. e_{a} \rightarrow_{\alpha} e_{b} \land e_{b}[x := e] = e'_{a} \]

**Proof** By rule induction in Curry-style substitution-as-a-relation, cf. Figure 2. Uniqueness of $e_{b}$ is guaranteed by the functionality of $\text{Fresh}(\cdot)$.

We stress that the above property is not provable by structural induction in $e_{a}$ and that it ensures that Curry-style substitution is, indeed, well-defined and functional.

**Lemma 14** For any $x$ and $e$, $\lceil x := e \rceil_{\text{Cu}} = \lceil \cdot := \cdot \rceil_{\text{Cu}}$ is a total, computable function of the first, open argument onto the second, open argument.

Lemma 13 also establishes the decomposition of Curry's calculus as a whole into the $\lambda^{\var}$-calculus.

**Lemma 15** $\rightarrow_{\alpha} \subseteq (\rightarrow_{\alpha}^{\text{Cu}})^{-1} \subseteq \rightarrow_{\alpha}$

**Lemma 16** $\rightarrow_{\beta} \subseteq \rightarrow_{\text{Cu}}^{\text{Cu}} \subseteq \rightarrow_{\alpha}^{\text{Cu}} \rightarrow_{\beta}$

**2.4 The Real $\lambda$-Calculus**

As suggested previously, the actual calculus we are interested in is the $\alpha$-collapse of $\lambda^{\var}$. Algebraically speaking, this means that we want to consider the following structure, cf. Hindley's presentation, Section 1.2.

**Definition 17** (The Real $\lambda$-Calculus)

$\Lambdatext{def} \lambda^{\var} / \eta_{\alpha}$
To address any inherent requirements for renaming in the \(\lambda\)-calculus, we introduce a formal notion called
Barendregt Conventional Form (BCF),\(^5\) which, as it turns out, provides a rational reconstruction of the usual
(informal) Barendregt Variable Convention [2], cf. [25].
BCFs are terms where all variable names are different.

**Definition 20** \(\text{Cf. Figures 4 and 6:}\)

\[
\text{BCF}(e) = \text{UB}(e) \land (\text{BV}(e) \cap \text{FV}(e) = \emptyset)
\]

As a first approximation to renaming-freeness, we note that it is a straightforward proof that BCFs residually
completes, i.e., that all marked redexes in a BCF can be contracted from within without causing variable
clashes.

**Lemma 21** \(\text{(BCF)} \bullet \cdots 

\begin{array}{c}
\begin{array}{c}
\beta^o
\end{array}
\end{array}
\square
\]

We also show that the residual-completion relation is functional on the full \(\beta\)-residual theory of a term, i.e.,
that residual completion always catches up with itself.

**Lemma 22**

\[
\begin{array}{c}
\begin{array}{c}
\beta^o
\end{array}
\end{array}
\square
\]

**Proof** The right-most conjunct follows from the left-most by a simple reflexive, transitive induction in
which the latter constitutes the base case. The left-most conjunct follows by a rule induction in
\(\leftarrow\beta^o\) for which it is paramount that redexes are enabled if \(\text{Capt}_x(-) \cap \text{FV}(-) = \emptyset\) rather than only if
\(\text{BV}(-) \cap \text{FV}(-) = \emptyset\). Other than that, the proof is mostly straightforward, albeit big.

The above property asserts that when residual completion exists, the considered divergence can be resolved
as shown. The property allows us to prove that \(\beta\)-residual theory is renaming-free up to BCF-initiality,
namely, that no redexes are blocked by their side-condition.

**Theorem 23** \(\text{(BCF)} \bullet \cdots 

\begin{array}{c}
\begin{array}{c}
\beta^o
\end{array}
\end{array}
\square
\]

**Proof** Consider a BCF and a \(\leftarrow\beta^o\)-reduction of it.
By Lemma 21, the considered BCF also residually completes and, by Lemma 22, the thus-created divergence
can be resolved by a trailing residual completion.

A subtle point of interest is that the above proof, in fact, shows that the \(\beta\)-residual theory of any term that
residually-completes, i.e., is renaming-free if contracted from within, is renaming-free in general.

\(^5\) The term was suggested to us by Randy Pollack.
4 Confluence

The previous section establishes a rather large fragment of the $\lambda_{\text{var}}$-calculus as susceptible to primitive equational reasoning. This section summarises and elaborates on our formally verified efforts to bring this to bearing on $\beta$-confluence [25]. We also present proofs that apply the methodology to prove $\eta$- and $\beta\eta$-confluence.

4.1 $\beta$-Confluence

The $\rightarrow_{\beta}$-relation does not enjoy the diamond property because a redex that is contracted in one direction of a divergence can be duplicated (and erased) in the other direction by the substitution operator. As shown by Tait and Martin-Löf, the potential divergence "blow-up" does not materialise because it can be controlled by parallel reduction. Please refer to Figure 7 for the $\lambda_{\text{var}}$-version of this relation.

Lemma 24

$$
\text{(BCF)} \quad \vdash \vdash_{\beta} \equiv
$$

Proof Rather than prove this property by an exhaustive case-splitting, thus resulting in a minimally resolving end-term, Takahashi observed that the considered diamond can be diagonalised by the relation that contracts all redexes in one step, i.e., by a maximally resolving end-term [23]. As we saw in Section 3 this is within reach of the structural proof principles of $\lambda_{\text{var}}$. □

The Full Proof Burden A real version of the parallel $\beta$-relation on syntax can be defined along the lines of Definition 17 (which, further to Lemma 21, turns out to be the real parallel $\beta$-relation).

Definition 25

$$
[c_1] \rightarrow_{\beta} [c_2] \iff [c_1] \equiv_{\eta} [c_2] \equiv_{\eta} [c_2]
$$

In order to prove the diamond property for $\rightarrow_{\beta}$, we need some measure of commutativity between $\alpha$- and $\eta$-steps. This measure is provided by the definition of $\alpha$-commutativity in Figure 7. We write $\alpha \equiv_{\beta} e'$ to mean that $e$ and $e'$ are related by $\alpha$-commutativity. Note that $\alpha \equiv_{\beta} e'$ is defined as the least type such that $\alpha \rightarrow_{\beta} e'$.

Fresh-Naming As the general $\alpha$-commutativity result is not provable, we introduce the following restricted $\alpha$-relation, which only fresh-names.

Definition 26

$$
e \rightarrow_{\alpha} e' \iff \exists x. \exists e \quad e \equiv_{\beta} e' \quad e \in \text{FV}(e) \cup \text{BV}(e)
$$

The fresh-naming $\alpha$-relation can straightforwardly be proven to commute with the parallel (actually, any) $\beta$-relation with the proviso that the resolving $\alpha$-steps are not necessarily fresh-naming (because of $\beta$-incurred term duplication).

Lemma 27

$$
\vdash_{\beta} \vdash_{0} \vdash_{\alpha}
$$

Similarly, the fresh-naming $\alpha$-relation can be shown to resolve $\alpha$-equivalence to a BCF (although the formal proof of this is surprisingly involved, cf. [25]).

Lemma 28

$$
\vdash_{0} \vdash_{\lambda} \vdash_{\alpha} \vdash_{0}
$$

Applying Administration With these results in place, we can lift Lemma 24 to the real $\lambda$-calculus.

Lemma 29

$$
\alpha(\rightarrow_{\beta}) \land \alpha(\rightarrow_{0}) \land \alpha(\rightarrow_{\alpha})
$$

Proof As for the left-most conjunct, see Figure 8 for the step by step resolution of the definitionally-given syntactic divergence. We trust the steps are self-evident and that it can be seen that a slight adaptation of the figure also proves the right-most conjunct. □

We are now in a position to establish $\beta$-confluence.

Theorem 30

$$
\text{Confl}(-_{\beta}) \land \text{Confl}(-_{0}) \land \text{Confl}(-_{\alpha})
$$

Proof The two top-most conjuncts are equivalent by Lemma 18. They can also be proved independently by applying the Diamond Tiling Lemma of Appendix C to the corresponding conjunct in Lemma 29. The third conjunct follows by Lemmas 15 and 16. The final conjunct follows in an analogous manner. □
4.2 $\eta$-Confluence

Unlike the $\beta$-relation, $\eta$-reduction is natively renaming-free:

**Lemma 31 (\alpha/\eta Commutativity)**

\[ \alpha \downarrow \quad \eta \downarrow \quad \eta \quad \frac{1}{\eta} \quad \frac{1}{\eta} \quad \frac{1}{\eta} \]

**Lemma 32 (\eta Commutativity)**

\[ \eta \downarrow \quad \eta \quad \eta \quad \frac{1}{\eta} \quad \frac{1}{\eta} \quad \frac{1}{\eta} \]

**Proof** The left-most conjunct is straightforwardly provable by structural means. The proof of the right-most property follows from the left-most as displayed in Figure 9. The top part of the figure is a proof by the general method; the lower part is an optimised version that takes advantage of $\eta$ commuting with $\alpha$, not just with $\alpha_0$.

**Theorem 33** $\text{Conf}(\rightarrow \eta) \land \text{Conf}(\rightarrow \eta) \land \text{Conf}(\rightarrow \alpha \eta)$

**Proof** The two left-most conjuncts can be established from the corresponding conjuncts in Lemma 32 by the Hindley-Rosen Lemma of Appendix C. The right-most conjunct can be established either by the Commuting Confluence Lemma of Appendix C applied to the left-most conjunct and generalisations of Lemmas 11 and 31 or, alternatively, it can be observed that the two right-most conjuncts are equivalent by Lemma 18.

4.3 $\beta\eta$-Confluence

Since the $\eta$-relation is natively renaming-free and the $\beta$-relation relies on the $\alpha$-relation, we must show that $\eta$ commutes with combined $\alpha\beta$-reduction in order to apply the Commuting Confluence Lemma of Appendix C.

**Lemma 34**

\[ \eta \downarrow \quad \frac{1}{\eta} \alpha \beta \quad \frac{1}{\eta} \alpha \beta \quad \frac{1}{\eta} \alpha \beta \]

**Proof** The proof of the left-most conjunct is straightforward. The $\alpha$-step in the resolution on the right is needed for the obvious divergence on $\lambda x. \lambda y. e x$, with $x \neq y$. The middle conjunct combines the left-most conjunct and Lemma 31. The right-most conjunct follows from the middle by the Hindley-Rosen Lemma of Appendix C.
Lemma 35
\[ \beta \Rightarrow \eta \quad \text{and} \quad \text{Confl}(\rightarrow_{\beta \eta}) \]

The left-most conjunct follows from the left-most conjunct of Lemma 34 as shown in Figure 10. The top part of the figure is by the general method; the lower part is an optimisation based on (full) on-commutativity, Lemma 31. The right-most conjunct follows by the Hindley-Rosen Lemma of Appendix C. \(\square\)

Theorem 36 Confl(\(\rightarrow_{\beta \eta}\)) \& Confl(\(\rightarrow_{\alpha \beta \eta}\))

Proof We first observe that the two conjuncts are equivalent by Lemma 18. They can also be proved independently by the Commuting Confluence Lemma of Appendix C applied to Theorems 30 and 33 as well as Lemma 35 and Lemma 34, respectively. \(\square\)

5 Residual \(\beta\)-Confluence

We say that the reflexive, transitive closure of a residual relation is the associated development relation, a step of which is said to be complete if the target term does not contain a mark, unMarked(\(-\)). With this terminology in place, we define a weakened version of the strong finite development property.\(^6\)

\(^6\) The strong finite development property also requires that the residual relation is strongly normalising. It is typically used to prove (residual) confluence.

Definition 37 Let \(\rightarrow_{\eta}\) be the residual relation of \(\rightarrow\). We say that \(\rightarrow\) enjoys the strong weakly-finite development property, SWFDP(\(\rightarrow\)), if

1. \(t \rightarrow:\eta t' \Rightarrow \exists t'', t'' \rightarrow_{\eta} \Leftrightarrow \text{unMarked}(t'')\)

--- developments can be completed

2. \(t \rightarrow_{\eta} t_i \land \text{unMarked}(t) \land i \in \{1, 2\} \Rightarrow t_1 = t_2\)

--- completions are unique

To motivate the name of the property, we see that, indeed:

Proposition 38 SWFDP(\(\rightarrow\)) \(\Rightarrow\) WN(\(\rightarrow_{\eta}\))\(^7\)

Proof By Definition 37, 1. and reflexivity of \(\rightarrow_{\eta}\). \(\square\)

Surprisingly, perhaps, we have that already the SWFDP implies residual confluence.

Lemma 39 SWFDP(\(\rightarrow\)) \(\Rightarrow\) Confl(\(\rightarrow_{\eta}\))

Proof Consider the following divergence:

\[ M_{1} \rightarrow M_{2} \]

\(^7\) The predicate WN(\(-\)) stands for Weak Normalisation and means that all terms reduce to a normal form.
By Definition 37, 1., there exist $N_1$, $N_2$, such that unMarked($N_1$), unMarked($N_2$) and:

$$
\begin{array}{c}
\alpha \downarrow M \quad \alpha \downarrow M \\
N_1 \quad N_2
\end{array}
$$

By transitivity of $\rightarrow_\alpha$ and Definition 37, 2., we see that, in fact, $N_1 = N_2$ and we are done.

With direct reference to Section 3, we define the following property, which is fairly easily proven to be equivalent to the SWFDP.

**Definition 40** A relation, $\rightarrow$, enjoys the residual-completion property, RCP($\rightarrow$), if there exists a residual-completion relation, $\rightarrow_\alpha$, such that:

1. $\rightarrow_\alpha \subseteq \rightarrow$
   - residual-completion is a development

2. $\rightarrow$ (NF$_\alpha$)
   - residual-completion totally completes

3. $\rightarrow_\alpha$
   - residual-completion is residually co-final

**Lemma 41** RCP($\rightarrow$) $\iff$ SWFDP($\rightarrow$)

Our interest in the RCP is its constructive nature, in particular when the residual-completion relation is defined as a computable function the way we did in Section 3.

**Lemma 42** RCP($\rightarrow_\beta$) $\land$ SWFDP($\rightarrow_\beta$)

**Proof** We prove the left-most conjunct. Clause 1. follows from the easily established fact that $\rightarrow_\beta \subseteq \rightarrow_\alpha$. Clause 2 follows from Lemmas 21 and 28. Finally, Clause 3 is proved as shown in Figure 11.

**Theorem 43** Confl($\rightarrow_\alpha$) $\land$ Confl($\rightarrow_\beta$)

We see that SN($\rightarrow_\alpha$) (i.e., the difference between the SWFDP and the strong finite development property) is not needed for concluding confluence from a residual analysis of the $\beta$-relation, something which is in stark contrast to established opinion [2, p.283]. Strong finite development essentially implies confluence through Newman’s Lemma, thus relying crucially on the (non-equational) SN-property for the residual relation. We think it a nice “purification” of the equational import of residual theory that an externally justified termination property is not needed for concluding confluence.

6 $\eta$-over-$\beta$-Postponement

As well as condensing Tait and Martin-Löf’s use of parallel $\beta$-reduction for proving $\beta$-confluence, Takahashi [23] also shows how to adapt the parallel-reduction technology to other typical situations in the equational theory of the $\lambda$-calculus. One such situation is for proving $\eta$-over-$\beta$-postponement, cf. Figure 12. The proof presented by Takahashi [23] essentially goes through up to BCF-initiaility as it stands, albeit not completely. Rather than focusing on the low-level technical details, this section merely shows the Administrative and Abstract proof layers of our formalisation of Takahashi’s proof.

The notion of commutativity that we have considered so far is orthogonal in nature to that employed in the $\eta$-over-$\beta$-Postponement Theorem. Whereas the former can be described as divergence commutativity, this section focuses on composition commutativity.

**Lemma 44** (BCF)

$$
\begin{array}{c}
\bullet \downarrow \beta \\
\eta \\
\eta \downarrow \eta
\end{array}
$$
Theorem 46

Proof By reflexive, transitive induction in \( \rightarrow_{\beta\eta\eta} \). The only interesting case is the transitive case, which follows in a manner akin to the Hindley-Rosen Lemma of Appendix C using Lemma 45.

7 \( \beta \)-Standardisation

Standardisation is also a composition-commutativity result like postponement. It is a very powerful result that, informally speaking, says that any reduction sequence can be performed left-to-right. Standardisation implies results such as the left-most reduction lemma, etc., [2], and guarantees the existence of evaluation-order independent semantics [20].

This section addresses three different approaches to proving standardisation due to Mitschke [18], Plotkin [20], and David [5], respectively. The three approaches are fairly closely related, with Plotkin’s proof bridging the other two, so to speak. Mitschke’s and Plotkin’s proofs both use semi-standardisation while David’s and Plotkin’s both can be described as absorption standardisation. In spite (actually because) of this, only Plotkin’s approach is formally provable by the proof principles we are considering. We shall examine the failures of the other two proofs closely.

7.1 Semi-Standardisation with Hereditary Recursion

In this section, we shall pursue a slight adaptation of Takahashi’s adaptation [23] of Mitschke’s proof [18]. Instead of head and a corresponding notion of inner reduction, we base the proof on weak-head reduction. This does not affect the formal status of the proof technique but does allow us to reuse the results of this section when pursuing Plotkin’s approach. The main proof burden is to show that (weak-)head redexes can contracted before any inner redexes, so-called semi-standardisation.

Definition 47 Weak-head \( \beta \)-redaction, \( \rightarrow_{\beta^w} \), is defined in Figure 14. The corresponding (strong) inner, \( \rightarrow_{\beta^i} \), and parallel inner, \( \rightarrow_{\beta^i\beta^i} \), \( \beta \)-relations are defined in Figure 15.
Standardisation

Weak-Head

Lemma 48

\[ \beta^{\text{wh}} \cdot \circ \quad \beta \quad \beta^{\text{wh}} \cdot \circ \quad \beta \]

\[ \text{(BCF)} \quad \alpha \quad N_1 \quad \beta \quad N_2 \quad \alpha \quad M_1 \quad \beta^{\text{wh}} \quad N_3 \quad \beta \quad N_4 \quad \alpha \]

\[ \alpha \quad M_1 \quad \beta^{\text{wh}} \quad M_2 \quad \beta \quad M_3 \quad \beta^{\text{wh}} \quad M_4 \quad \alpha \]

\[ \text{Fig. 16. The administrative proof layer for weak-head before inner parallel decomposition} \]

Lemma 49

\[ \text{(BCF)} \quad \alpha \quad N_1 \quad \beta \quad N_2 \quad \alpha \quad M_1 \quad \beta^{\text{wh}} \quad N_3 \quad \beta \quad N_4 \quad \alpha \]

\[ \alpha \quad M_1 \quad \beta^{\text{wh}} \quad M_2 \quad \beta \quad M_3 \quad \beta^{\text{wh}} \quad M_4 \quad \alpha \]

\[ \text{Fig. 17. The administrative proof layer for parallelization of weak-head after inner} \]

\[ \text{Lemma 48} \]

\[ \beta^{\text{wh}} \cdot \circ \quad \beta \quad \beta^{\text{wh}} \cdot \circ \quad \beta \]

\[ \text{(BCF)} \quad \alpha \quad N_1 \quad \beta \quad N_2 \quad \alpha \quad M_1 \quad \beta^{\text{wh}} \quad N_3 \quad \beta \quad N_4 \quad \alpha \]

\[ \alpha \quad M_1 \quad \beta^{\text{wh}} \quad M_2 \quad \beta \quad M_3 \quad \beta^{\text{wh}} \quad M_4 \quad \alpha \]

\[ \text{Fig. 16. The administrative proof layer for weak-head before inner parallel decomposition} \]

\[ \text{Lemma 49} \]

\[ \text{(BCF)} \quad \alpha \quad N_1 \quad \beta \quad N_2 \quad \alpha \quad M_1 \quad \beta^{\text{wh}} \quad N_3 \quad \beta \quad N_4 \quad \alpha \]

\[ \alpha \quad M_1 \quad \beta^{\text{wh}} \quad M_2 \quad \beta \quad M_3 \quad \beta^{\text{wh}} \quad M_4 \quad \alpha \]

\[ \text{Fig. 17. The administrative proof layer for parallelization of weak-head after inner} \]

\[ \text{Proof} \quad \text{Please refer to Figure 16 for the proof of the right-most conjunct based on the left-most conjunct, which, in turn, is proved by rule induction in } - \ast \beta. \]

\[ \text{Lemma 50 (Semi-Standardisation)} \]

\[ \beta^{\text{wh}} \ast \circ \quad \beta \]

\[ \text{Proof} \quad \text{From Lemmas 48 and 49, cf. the obvious reflexive, transitive generalisation of Lemma 53 in the transitive case.} \]

At this point, the idea is to recursive over the \( \circ \) in Lemma 50 and show that the sub-terms in which the outgoing \( - \ast \beta \)-step are ordinary \( \beta \)-steps, themselves can be semi-standardised and so on. Unfortunately, the \( \circ \) is quantified over \( \alpha \)-equivalence classes, for which no recursion is possible and we are stuck.

\[ \text{7.2 Hereditary Weak-Head Standardisation} \]

Plotkin [20] defines standardisation as the least contextually-closed relation on terms that enjoys left-absorptivity over weak-head reduction. The following presentation of the proof methodology owes a great deal to McKinna and Pollack [17]. The difference between their and our presentation is that we focus on provability with structural induction, etc., while they work with an alternative syntactic framework that is derived from first-order abstract syntax with two-sorted variable names. The proof requirements in their setting and in ours are substantially different as a result.

\[ \text{A First (Failing) Approach} \]

A first approach, which immediately fails, is to define Plotkin's relation directly on terms.

\[ \frac{e \longrightarrow \beta^{\text{wh}} e' e' \longrightarrow_{-\beta} e''}{e \longrightarrow_{-\beta} e''} \quad \frac{x \longrightarrow_{-\beta} x}{e_1 \longrightarrow_{-\beta} e'_1 e_2 \longrightarrow_{-\beta} e'_2 e' e} \]

As standardisation pertains to all \( \beta \)-reductions (i.e., \( -\ast \beta \), not just \( -\ast \beta \)), the naive approach needs the full \( \lambda \)-calculus to be renaming-free, which it is not. The problem manifests itself in the required administrative proof layer for the standardisation property and its exact nature is of independent interest. The point is that, even if it is possible to prove the following key property (which, in fact, seems to be the case\(^8\)), we cannot prove

\[ \text{\footnotesize{\text{\textsuperscript{8} Coincidentally, it is interesting to note that the proof of the property can only be conducted by rule induction in } \longrightarrow_{-\beta} \text{ and not in } -\ast \beta.}} \]
full standardisation but at most standardisation of the renaming-free fragment of the $\lambda^{\omega\alpha}-$calculus.

$$(BCF) \xrightarrow{\beta} P \xrightarrow{\beta} P$$

Please refer to Figure 18 for the only two sensible approaches to the administrative proof layer for the following property, which is derived from the one above.

**Non-Lemma 51**

$$\xrightarrow{\beta} P \xrightarrow{\beta} P$$

The left-most diagram in the figure attempts to align itself with Figure 13, which fails because $\succ \tau_{wh}$ only commutes with $\succ \alpha_{0}$. The right-most diagram adheres to this and fails because of the inserted $\succ \alpha_{0}$, which we cannot incorporate into the syntactic version of the property. It is even straightforward to come up with a counter-example.

$$(\lambda x.x)(\lambda x.\lambda y.x y) \xrightarrow{\beta} (\lambda x.\lambda y.x y)(\lambda x.\lambda y.x y)$$

We can turn the end-term into an $\alpha$-equivalent BCF, as it happens, which standardises:

$$(\lambda x_{1}.\lambda y_{1}.x_{1} y_{1}),(\lambda x_{2}.\lambda y_{2}.x_{2} y_{2}) \xrightarrow{\succ \alpha} \lambda x_{1}.\lambda y_{1}.x_{1} y_{1}$$

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 standardise to it.

**Lemma 54**

$$\frac{(\lambda y_{1} e_{1} \equiv \equiv \lambda y_{1}'.e_{1}' \equiv \equiv \lambda z_{1}.\lambda e_{2} \equiv \equiv \lambda z_{1}.e_{3}}{\lambda z_{1}.e_{0} \equiv \equiv \lambda z_{1}.e_{0}}$$

Fig. 18. Failed administrative proof layer for left-absorptivity of progression standardisation

Fig. 19. The administrative proof layer for the $(\lambda_{\succ \alpha})$-case of Lemma 54

**Combining Term Structure and $\alpha$-Collapsed Reduction**

In order to avoid these problems, we adapt the above definition slightly.

**Definition 52**

$$\frac{e \xrightarrow{\succ \tau_{wh}} e'}{(\tau_{wh})}$$

The 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, further to the failed proof of Lemma 4, it is by no means obvious whether this mixture will lend itself to primitive structural reasoning. The proof-technical issue surfaces in the $(\lambda_{\succ \alpha})$-case of the proof of Lemma 54.

**Lemma 53**

$$\frac{\beta \xrightarrow{\alpha} \beta'}{\beta \xrightarrow{\tau_{wh}} \beta'}$$

**Proof** The property can be derived from Lemmas 48 and 49 based on a suitable adaptation of the Hindley-Rosen Lemma, cf. Appendix C.

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

**Lemma 54**

$$[(\lambda e_{1}) \equiv \equiv \lambda e_{2} \wedge e_{2} \equiv \equiv \lambda e_{3}] \Rightarrow e_{1} \equiv \equiv \lambda e_{3}$$

**Proof** The proof is by rule induction in $\succ \tau_{wh}$ and uses Lemma 53 before applying the I.H. and the definitional left-absorptivity over weak-head reduction when needed. As far as administration is concerned, the only interesting case is for abstraction.

**Case $(\lambda_{\succ \alpha})$:** We are considering the following situation (although this takes some effort to substantiate).

$$\lambda y_{1}.e_{1} \equiv \equiv \lambda y_{1}'.e_{1}' \equiv \equiv \lambda z_{1}.e_{2} \equiv \equiv \lambda z_{1}.e_{3} \equiv \equiv \lambda z_{1}.e_{0}$$
By Definition 47 and the case, we have \( e'_1 \equiv_{\beta} e_2' \) and \( e_2 \succ_{\tau_{\text{prg}}} e_3 \). If \( y' = x \), we can prove \( e'_2 \equiv_{\alpha} e_2 \), which means that we are considering \( [e'_1] \rightarrow_{\beta} [e'_2] \succ_{\tau_{\text{wh}}} e_3 \), so to speak. From Lemma 48, we thus have: 
\[
[e'_1] \rightarrow_{\text{prg}} [e'_2] \rightarrow_{\beta} [e'_2] \succ_{\tau_{\text{wh}}} e_3.
\]
An application of the I.H. and an invocation of the \((\text{wh}_{\text{pre}})\)-rule will then give us that \( e'_1 \succ_{\tau_{\text{wh}}} e_3 \) and we have \( \lambda x. e'_1 \succ_{\tau_{\text{wh}}} \lambda x. e_3 \) by the \((\lambda y_{\text{wh}})\)-rule. A final (reflexive) application of the \((\text{wh}_{\text{pre}})\)-rule thus finishes the case: \( \lambda y. e_1 \succ_{\tau_{\text{wh}}} \lambda x. e_3 \). Unfortunately, we can not guarantee \( y' = x \). Instead, Figure 19 shows how to overcome this using our general administrative proof-layer technology, cf. Figure 3. Based on the upper line, we first rewrite \( \lambda y'. e'_1 \) to \( \text{(the BCF)} \lambda x. e_0 \) (although it takes some effort to substantiate that this is possible). The commuting square involving \( \lambda x. e'_0 \) can then be constructed by the obvious adaptation of Lemma 27 and the diagram can finally be closed based on Lemma 11. To show that \( \lambda y. e_1 \succ_{\tau_{\text{wh}}}, \lambda x. e_3 \), first apply the reasoning above to show that \( \lambda x. e_0 \) does and, then, use the \((\text{wh}_{\text{pre}})\)-rule reflexively to show the result we are after.

Other Cases: Fairly straightforward. \( \square \)

**Theorem 55** 
\[ [e_1] \rightarrow_{\beta} [e_2] \Rightarrow e_1 \succ_{\tau_{\text{wh}}} e_2 \]

**Proof** By reflexive, left-transitive induction in \( \rightarrow_{\beta} \).

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 \( \rightarrow_{\beta} \)-step into \( \rightarrow_{\text{pre}} \) and \( \rightarrow_{\beta} \) (seeing that we can show that the union of the latter two is the former). In case of \( \rightarrow_{\text{pre}} \), we are done by definition of \( \succ_{\tau_{\text{wh}}} \). In case of \( \rightarrow_{\beta} \), we are done by Lemma 54. \( \square \)

### 7.3 (Failing) Progression Standardisation

An alternative proof development for standardisation was proposed by David [5] and pursued, more or less independently, in [13–15]. The idea is to define a standardisation relation directly by induction over terms (although this is only done implicitly in [5]): \( \succ_{\text{prg}} \), and to show that this relation right-absorbs the ordinary \( \beta \)-relation. In that sense, the proof development is the dual approach to what we considered in the previous section. Informally, the key technical point is to contract terms as follows, cf. [13, 15]:

\[
(\ldots(e[x := e_0]e_1)\ldots)e_k \succ_{\text{prg}} e'
\]

\[
(\ldots(\lambda x.e_0) e_1)\ldots)e_k \succ_{\text{prg}} e'
\]

In order for the relation to make sense in the current setting, it is necessary to supply it with a finite axiomatisation, which can be done.

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 as the rule otherwise would be leftmost reduction.

**Right-Absorptivity** As mentioned, the key technical lemma is purported to show right-absorptivity of \( \succ_{\sigma} \), \( \rightarrow_{\beta} \), which appears to be straightforward, at least in the case of the above contraction rule [5, 13–15].

**Non-Lemma 56**

\[
(\text{BCF}) \succ_{\sigma} \succ_{\beta} \succ_{\text{prg}}
\]

Unfortunately, not even the BCF-initial version of the property is true. The following is a counter-example.

\[
(\lambda s ss)(\lambda x. (\lambda y. xy) s z) \succ_{\sigma} \succ_{\beta} (\lambda y. (\lambda x. xs) y z)\] (\lambda y. (\lambda x. xy) z)

The problem in the counter-example is the last step of the standardisation, which amounts to the contraction of the redex involving the inner \( y \)-abstraction below.

\[
(\lambda y. (\lambda x. (\lambda y. xy) z)) y z
\]

As it happens, this is the point where the considered \( \rightarrow_{\beta} \)-step (i.e., the contraction of the redex involving the \( x \)-abstraction) must be inserted but that is not possible because of a clash with the inner \( y \)-abstraction.

**Left-Absorptivity** In sharp contrast with the above (and surprisingly, at first), it turns out that it is possible to prove left-absorptivity, as also seen at the beginning of Section 7.2.

**Non-Lemma 57**

\[
(\text{BCF}) \succ_{\beta} \succ_{\text{prg}}
\]

The difference between right- and left-absorptivity is that the universal quantification over \( \succ_{\text{prg}} \) covers far fewer steps in the latter case than in the former. As we saw, this manifests itself when trying to prove standardisation for the real \( \lambda \)-calculus.
8 Conclusion

Standard, informal practice in the programming language theory community when using structural induction and related proof principles is to assume that variables clashes are not an issue (aka Barendregt’s Variable Convention). We have shown this to be formally correct for a range of standard properties, possibly up to BCF-initiability, cf. Lemmas 21, 22, 24, 32, 34, 44, 48, 49, and 54. For the most part, we have been able to show that the undertaken proof burden resolution is formally incomplete in the sense that the formal proof burden can be met by the addition of a fairly simple administrative proof layer, cf. Figures 8, 9, 10, 11, 13, 16, and 17. The administrative proof layers mostly rely on the same additional lemmas, thus presenting a blow-up of proof obligations. We studied standardisation in some detail and found that only one out of three proof techniques appears to be amenable to the use of structural induction, etc..

A Commutative Diagrams

We use commutative diagrams in three different ways, which are distinguished in the way they write vertices.

A.1 Vertices as Terms

When written with terms as vertices, commutative diagrams simply describe reduction scenarios.

A.2 Vertices as M’s, N’s

We shall see next that commutative diagrams are used to express rewriting predicates such as:

“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 gradually introduce N’s from supporting lemmas to eventually substantiate the existence claims. Please note that we use * to signify “claimed” existence that are impossible.

A.3 Vertices as o’s, o’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 (o vs o) 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 full-coloured (black) or half-coloured (gray). Informally, the colour indicates assumed and concluded relations, respectively. An edge connected to a o must be half-coloured. A diagram must be type-correct on domains. A property is read off of a diagram thus:

1. write universal quantifications for all o’s
2. assume the full-coloured relations and the validation of any guard for a o
3. conclude the guarded existence of all os and their relations

The following diagram and property are thus equivalent.

\[(P) \overset{\cdot}{\rightarrow} \overset{\cdot}{\rightarrow} \; e_1 \rightarrow e_2 \; \wedge \; e_1 \rightarrow e_3 \; \wedge \; P(e_1)\]
\[\Downarrow \; \Downarrow \; \Downarrow\]
\[
\overset{\cdot}{\rightarrow} \; \overset{\cdot}{\rightarrow} \; \exists e_4 \; e_2 \rightarrow e_4 \; \wedge \; e_3 \rightarrow e_4 \; \wedge \; Q(e_4)
\]

B Notation and Terminology

We say that a term reduces to another if the two are related by a reduction relation and we denote the relationship by an infix arrow between the two terms. The “direction” of the reduction should be thought of as being from-left-to-right. The sub term of the left-hand side that a reduction step “acts upon” is called the redex of the reduction and it is said to be contracted.

- The converse of a relation, →, is written (→)*.
- Composition is:

\[a \rightarrow_{1}; \rightarrow_{2} c \iff \exists b \; a \rightarrow_{1} b \; \land \; b \rightarrow_{2} c\]

- Given two reduction relations →, and →*, we have: \[\rightarrow_{1, 2} \iff \rightarrow_{1} \cup \rightarrow_{2}.\] If no confusion is possible, we omit the comma.

- The reflexive closure of a relation is:  

\[e_1 \rightarrow e_2 \; \rightarrow_{1, 2} \; e \rightarrow_{1, 2} e\]

- The reflexive, transitive closure is:

\[e_1 \rightarrow e_2 \; e \rightarrow e \; e_2 \rightarrow e_3 \; e \rightarrow e_3\]

We will also denote → by (→)*.

- The reflexive, transitive, and symmetric closure is:

\[e_1 \rightarrow e_2 \; e = e \; e_2 \rightarrow e_3 \; e_1 \rightarrow e_3\]

- The situation of a term reducing to two terms is called a divergence.

\[10\] This and the next two items are immediately associated with primitive induction principles. Equality, however, is only point-wise (or extensional), and no recursion principle is possible.
- Two diverging reductions, as defined above, are said to be co-initial.
- Dually, two reductions that share their end-term are said to be co-final.
- Co-initial reductions are resolvable if they compose with co-final reductions.
- A relation has the diamond property, if any divergence can be resolved.
- A relation, →, is confluent, Confl, if ω(→).

C Known Abstract Results

Diamond Tiling Lemma

(3→2,→1 ⊆→2 ⊆→1 ∧ ω(→2)) =⇒ ω(→1)

Hindley-Rosen Lemma

\[ 2 \rightarrow 1 \rightarrow \downarrow 2 \Rightarrow 2 \rightarrow 1 \rightarrow \downarrow 2 \]

Commuting Confluence Lemma

\[ 2 \rightarrow \downarrow 1 \rightarrow \uparrow \text{Confl}(→1) \land \text{Confl}(→2) =⇒ \text{Confl}(→1,→2) \]

References


