A Formalised First-Order Confluence Proof for the $\lambda$-Calculus Using One-Sorted Variable Names

(Barendregt Was Right after all ... almost)

René Vestergaard$^1$ and James Brotherston$^2$

$^1$ CNRS-IML, Marseille, France, vester@iml.univ-mrs.fr*

$^2$ University of Edinburgh, Scotland, jjb@dcs.ed.ac.uk**

Abstract. We present the titular proof development which has been implemented in Isabelle/HOL. As a first, the proof is conducted exclusively by the primitive induction principles of the standard syntax and the considered reduction relations: the naive way, so to speak. Curiously, the Barendregt Variable Convention takes on a central technical role in the proof. We also show (i) that our presentation coincides with Curry’s and Hindley’s when terms are considered equal up-to $\alpha$ and (ii) that the confluence properties of all considered calculi are equivalent.

1 Introduction

The $\lambda$-calculus is a higher-order language: terms can be abstracted over terms. It is intended to formalise the concept of a function. The terms of the $\lambda$-calculus are typically generated inductively thus: $\Lambda^{\text{var}} ::= x \mid \Lambda^{\text{var}} \Lambda^{\text{var}} \mid \lambda x.\Lambda^{\text{var}}$

A $\lambda$-term, $e \in \Lambda^{\text{var}}$, is hence finite and is either a variable, an application of one term to another, or the functional abstraction (aka binding) of a variable over a term, respectively. On top of the terms, we define reduction relations, as we shall see shortly. Intuitively, we will also want to consider terms that only differ in the particular names used to express abstraction to be equal. However, this is a slightly tricky construction as far as the algebra of the syntax goes and we will only undertake it after mature consideration.

It is common, informal practice to take the variables to belong to a single infinite set of names, $\mathcal{V}N$, with a decidable equality relation, $=$, and that is indeed what we will do. Recent research [8,17] has shown that there can be formalist advantages to employing a certain amount of ingenuity on the issue of variable names. Still, we make a point of following the naive approach. In fact, the main contribution of this paper is to show that it is not only possible but also feasible and even instructive to use this, the naive set-up, for formal purposes. This is relevant both from a foundational and a practical perspective.

* Supported under EU TMR grant # ERBFMRXCT-980170: LINEAR. Work done in part while visiting LFCS, University of Edinburgh from Heriot-Watt University.

** Supported by a grant from LFCS, University of Edinburgh.
The latter more-so as we, as a first, give a rational reconSTRUCTION of the widely used and very helpful Barendregt Variable Convention (BVC) [1].

We stress that $\lambda^\text{var}$ is first-order abstract syntax (FOAS) and therefore comes equipped with a primitive (first-order) principle of \textit{structural induction} [2]:

$$
\forall x.P(x) \quad \forall e_1, e_2. P(e_1) \land P(e_2) \rightarrow P(e_1 e_2) \quad \forall x, e. P(e) \rightarrow P(\lambda x.e)
$$

Similarly, the syntax also comes equipped with a primitive recursion principle so we can define auxiliary notions (e.g., free variables) by case-splitting.

\textbf{The Issues.} In the set-up of FOAS defined over one-sorted variable names (FOAS$_{VN}$), name-overlaps seem inevitable when computing. Traditionally, one therefore renames offending binders when appropriate. This has a two-fold negative impact: (i) the notion ‘sub-term of’ on which structural induction depends is typically broken and (ii) as a term can reduce in different directions, the resulting name for a given abstraction cannot be pre-determined. Consider, e.g., the following example taken from [11] — for precise definitions see Section 1.2:

$$(\lambda x.(\lambda y.\lambda x.xy)x)y \xrightarrow{\beta^C} (\lambda y.\lambda x.xyy) \xrightarrow{\beta^C} \lambda x.xy$$

Equational reasoning about FOAS$_{VN}$ can thus seemingly only be conducted up-to post-fixed “name-unification”. Aside from any technical problems this might pose, the formal properties we establish require some interpretation.

The basic problems with FOAS$_{VN}$ has directly resulted in the inception of syntax formalisms (several of them recent) which overcome the issues by native means [4,5,6,7,8,12]. In general, they mark a conceptual and formal departure from the naive qualities of FOAS$_{VN}$. This is in part unfortunate because FOAS$_{VN}$ is the de facto standard in programming language theory where, as a result of the problems, it is customary to reason while “assuming the BVC” [1]

\begin{itemize}
  \item \textbf{2.1.12.} Terms that are $\alpha$-[equivalent] are identified."
  \item \textbf{2.1.13.} If $M_1, \ldots, M_n$ occur in a certain mathematical context, [their] bound variables are chosen to be different from the free variables.”
  \item \textbf{2.1.14.} Using 2.1.12/13 one can work with $\lambda$-terms the naive way.”
\end{itemize}

\textbf{Our Contribution.} We

\begin{itemize}
  \item show that it is possible and feasible to conduct formal equational proofs about higher-order languages by simple, first-order means
  \item show that this can be done over FOAS$_{VN}$, as done by hand
\end{itemize}

\footnote{1 Thanks to Regnier for observing that this need not happen with parallel substitution.}

\footnote{2 We make reference to Barendregt because it is common practice to do so. Many other people have imposed hygiene conditions on variables.}