Conflicting Accounts of λ-Definability

Barry Jay and Jose Vergara
University of Technology, Sydney
{Barry.Jay,Jose.Vergara}@uts.edu.au

Abstract

A function on some domain is λ-definable if the corresponding function of λ-terms is so definable. However, the correspondence is parametrized by a representation of the domain. Often there is a natural choice of representation, but when the domain consists of λ-terms then they can be represented by either themselves or by the Church numeral of their Gödel number. This choice determines whether or not all computable functions are λ-definable.

For over fifteen years, the lead author has been developing calculi that are more expressive than λ-calculus, beginning with the *constructor calculus* [8], then *pattern calculus* [2, 7, 3], SF-calculus [6] and now λSF-calculus [5]. In brief, pattern calculus supports generic versions of the usual data base queries, that select and update. The SF-calculus is a combinatory logic that can also query programs, i.e. closed normal forms. Now λSF-calculus is able to query programs expressed as λ-abstractions, as well as combinators, something that is beyond pure λ-calculus. In particular, we have proved (and verified in Coq [4]) that equality of closed normal forms is definable within λSF-calculus. So, equality of closed normal forms is λSF-definable. Further, since these forms include the closed normal forms of pure λ-calculus, it follows that the equality function of the latter, also known as Church's δ, is definable. Hence

Corollary 1 *Church's δ is λSF-definable.*

and expressive power has been increased.

Unfortunately, the most common reaction to these results has been skepticism, tending towards hostility and derision. After all, the story goes, your claims contradict the Church-Turing Thesis; nothing is more expressive than λ-calculus. We can source this counter-argument back to a statement in Kleene's book [10, p. 320] "The equivalence of the computable to the λ-definable functions ... was proved by Turing 1937." Since equality of closed normal forms in λ-calculus is computable, this implies

Corollary 2 *Church's δ is λ-definable.*

which renders Corollary 1 impotent. However, in Barendregt's book [1, p. 520, Corollary 20.3.3] we have its negation

Corollary 3 *Church's δ is not λ-definable.*

So the source of the conflict lies within the standard account of the expressive power of λ-calculus, as expressed in Corollaries 2 and 3. This note will resolve

the conflict between their accounts of λ-definability, so that all three corollaries can be interpreted as true statements, but then λ-calculus is found wanting.

Although the following critique is quite straightforward, it may trouble those who were content with the standard account. The peaceful approach is thus to assess the truth of each sentence of our critique in turn, postponing any reflection upon its implications until the end.

Kleene's paper *λ-definability and recursiveness* [9] shows that all recursive numerical functions are λ-definable, with respect to a numerical definition of λ-definability. However, he intends his results to apply to computation with respect to domains other than natural numbers, such as real numbers or strings, when he gives, on page 343, a more general definition

> ... and we shall understand, by the lambda-definition of a function of which the arguments or the values are other mathematical entities, the lambda-definition of a function which corresponds under the representation of the mathematical entities by well-formed formulas (in case a representation has been specified).

Let us consider four examples, on three domains. The canonical example in his paper is the *Church representation*, that maps a natural number to its Church numeral. Another example, familiar to Kleene, concerns the domain of words written in some alphabet (e.g. of a Turing machine), whose representation is given by Gödel numbering. That is, any domain supporting a Gödel numbering has a *Gödel representation*. Now consider the domain consisting of λ-terms in closed normal form. These have both a Gödel representation and also an *inclusion representation*, in which terms are represented by themselves. Let us say that a function that is λ-definable with respect to an inclusion representation is *definable as a λ-term*. Similarly, a function that is λSF-definable with respect to an inclusion representation is *definable as a λSF-term*.

Now we can resolve the conflict by observing that Corollary 2 is true for the Gödel representation of λ-terms, while Corollary 3 is true for the inclusion representation. Similarly, Corollary 1 is true for the inclusion representation. All conflicts between the corollaries have been resolved.

The status of λ-calculus can now be summarized as follows. There are effectively calculable functions of λ-terms in closed normal form, including equality and the Gödel representation, that are *not* definable as λ-terms but are definable as λSF-terms. Hence there are calculi that are provably more expressive than λ-calculus, and this is fully consistent with the existing foundations of computing, properly understood.

Postscript

Reviewers for the journal made some suggestions, which have been used to improve the paper. They also raised some issues which we thought had already been addressed in the text above. Rather than trying to add emphasis to it, or footnotes, let us summarize and address their concerns directly.

All agree that the technical claims in the note are true. The only technical concern was whether the term equality being used is syntactic or semantic (i.e. being equivalent with respect to reduction). However, since we are only concerned with closed normal forms, semantically equal terms are syntactically equal after suitably renaming bound variables by α-conversion. So the results

hold true for either notion. If renaming is a concern, then one can replace the usual syntax of λ-calculus with de Bruijn indices, for which semantic equality is identical with syntactic equality.

The remaining issues concern the nature of the non-technical claims, and their significance. The strongest objection is that there is nothing new here, since Barendregt has already shown how to add Church's δ to λ-calculus. Certainly, this holds if one considers the works of Barendregt and Jay only. However, the conflict is between Barendregt and Kleene, who disagree about whether δ is λ-definable; Jay is merely an observer.

The other objection concerns the significance of this note. Perhaps the conflict we have identified is merely a potential source of error, a trap for novices that experts easily avoid. It is hard to see how such a claim might be justified. Certainly, our personal experience, including private correspondence and anonymous referees reports, all suggest the opposite. However, let us discount this, and focus on formal publications. Much could be said here, but perhaps one example will suffice. According to the index in Barendregt's book [1], the *definition* of λ-definability is limited to numerical functions, but its *use* in characterizing Church's δ considers functions of λ-terms. The more general definition is not to be found in Kleene's book [10] either. Rather, one must track down the original paper from 1936. Whether this error is potential or potent, the issues deserve clarification.

References

[1] Henk P. Barendregt. *The Lambda Calculus — Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, revised edition, 1984.

[2] Barry Jay. The pattern calculus. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 26(6):911–937, November 2004.

[3] Barry Jay. *Pattern Calculus: Computing with Functions and Structures*. Springer, 2009.

[4] Barry Jay. LamSF repository of proofs in Coq, February 2016. `https://github.com/Barry-Jay/lambdaSF`.

[5] Barry Jay. Programs as data structures in λSF-calculus. *Electronic Notes in Theoretical Computer Science*, 325:221 – 236, 2016. The Thirty-second Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXII).

[6] Barry Jay and Thomas Given-Wilson. A combinatory account of internal structure. *Journal of Symbolic Logic*, 76(3):807–826, 2011.

[7] Barry Jay and Delia Kesner. First-class patterns. *Journal of Functional Programming*, 19(2):191–225, 2009.

[8] C.B. Jay. Distinguishing data structures and functions: the constructor calculus and functorial types. In S. Abramsky, editor, *Typed Lambda Calculi and Applications: 5th International Conference TLCA 2001, Kraków,*

Poland, May 2001 Proceedings, volume 2044 of *Lecture Notes in Computer Science*, pages 217–239. Springer, 2001.

[9] Stephen C. Kleene. λ-definability and recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.

[10] Steven C. Kleene. *Introduction to Metamathematics*. Bibliotheca Mathematica. North-Holland Publishing Company, Amsterdam, 1952. Co-publisher: Wolters–Noordhoff; 8th revised ed.1980.