Journal of Applied Logic 18 (2016) 42–70

Contents lists available at ScienceDirect

Journal of Applied Logic

www.elsevier.com/locate/jal

Bridging Curry and Church's typing style

Fairouz Kamareddine [a,*], Jonathan P. Seldin [b,*], J.B. Wells [a]

[a] *School of Maths and Computer Sc., Heriot-Watt Univ., Edinburgh, UK*
[b] *Maths and Computer Sc., Univ. of Lethbridge, Canada*

ARTICLE INFO

Article history:
Received 24 March 2015
Accepted 26 May 2016
Available online 16 June 2016

Keywords:
Church-style typing
Curry-style typing
Domain-full typing
Domain-free typing

ABSTRACT

There are two versions of type assignment in the λ-calculus: Church-style, in which the type of each variable is fixed, and Curry-style (also called "domain free"), in which it is not. As an example, in Church-style typing, $\lambda_{x:A}.x$ is the identity function on type A, and it has type $A \to A$ but not $B \to B$ for a type B different from A. In Curry-style typing, $\lambda_x.x$ is a general identity function with type $C \to C$ for *every* type C. In this paper, we will show how to interpret in a Curry-style system every Pure Type System (PTS) in the Church-style without losing any typing information. We will also prove a kind of conservative extension result for this interpretation, a result which implies that for most consistent PTSs of the Church-style, the corresponding Curry-style system is consistent. We will then show how to interpret in a system of the Church-style (a modified PTS, stronger than a PTS) every PTS-like system in the Curry style.

1. Introduction

There are two main styles of type theory in λ-calculus: the Church-style, in which each abstraction indicates the type of the variable, as in

$$\lambda_{x:A}.M,$$

and the Curry-style, in which no such type is given:

$$\lambda_x.M.$$

These two styles of typing are often called the *domain-full* and the *domain-free* styles respectively. These styles are compared and discussed in [3].

* Corresponding author.
E-mail addresses: fairouz@macs.hw.ac.uk (F. Kamareddine), seldin@cs.uleth.ca (J.P. Seldin).

http://dx.doi.org/10.1016/j.jal.2016.05.008
1570-8683/

Remark 1. Barthe and Sørensen [3] distinguish between domain-free systems, which they regard as Church-style systems with the types of the bound variables omitted, and what they think of as the Curry view, in which typing rules assign types to terms that already exist in the pure λ-calculus. In this they are following Barendregt [2, Definition 4.1.7], who identifies as the Curry-version of $\lambda 2$ a system in which the rules for $\forall$ are given as follows:

$$\text{($\forall$-elimination)} \quad \frac{\Gamma \vdash M : (\forall \alpha.\sigma)}{\Gamma \vdash M : [\tau/\alpha]\sigma}$$

$$\text{($\forall$-introduction)} \quad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : (\forall \alpha.\sigma)} \qquad \alpha \notin \mathsf{FV}(\Gamma)$$

But these two rules seem to be closer to the ideas of the intersection type systems than to any of the systems that interested Curry. As should be clear from [7, Chapter 14], Curry was basically interested in the system usually called $\lambda \rightarrow$, and the basic characteristic of his version is that $\lambda_x.x$ in λ-calculus and I in a system of combinators can have any type of the form $\alpha \rightarrow \alpha$, which Curry wrote $\mathsf{F}\alpha\alpha$. This is what Curry called *functionality*. He also suggested what he called *generalized functionality*, in which the constant F was replaced by G, where $\mathsf{G}\alpha\beta$ is the type we now write $(\Pi_{x:\alpha}.\beta x)$. Seldin treated a basic form of generalized functionality in Curry's style in his paper [20]. On the other hand, Church's typing is probably best exemplified by his simple type theory of [5], which is characterized by the presence of the type of each bound variable, as in $\lambda_{x^\alpha}.M$. Hence, to be historically accurate, it is better to identify the Curry-style with domain-free type systems.

In a Curry-style system, there are terms like $\lambda_y.yy$ that have no types. But there is a sense in which this is also true in a Church-style system: $\lambda_{y:A}.yy$ is a perfectly good pseudoterm of a Church-style system, but it will not have a type in many of the usual systems. Perhaps the vocabulary used may disguise the similarity here: a pseudoterm in a Church-style system corresponds to a term in a Curry-style system.

There is one standard interpretation of a Church-style system in a corresponding Curry-style system: the function Erase, which simply deletes the domains from a formula, so that

$$\text{Erase}(\lambda_{x:A}.M) \equiv \lambda_x.M.$$

Erase has been used extensively to relate Church-style systems and Curry-style systems. For example, Erase and modifications of Erase are used by Steffen van Bakel et al. [22] to compare Church-style PTSs (which they call typed systems) and Curry-style PTSs (which they call type assignment systems). But using Erase to interpret a Church-style PTS in a Curry-style PTS causes some type information to be lost. One might think that this information can be restored from the type $(\Pi_{x:A}.B)$ of an abstraction term $(\lambda_x.M)$ by mapping this to $(\lambda_{x:A}.M)$. But as is shown in [22, Example 3.5, Theorem 3.6], this is too simple and may not work properly for some systems.

For these reasons, we think there is a need for a method of relating the Church-style typing and the Curry-style typing that does not lose this type information.

In this paper, we propose to show how to interpret a system of each style in an appropriate system of the other without this kind of loss of typing information. In one direction, the direction from Church-style to Curry-style, the interpretation is defined by allowing an abstraction of the form $(\lambda_{x:A}.M)$ to be an abbreviation for a term of the Curry-style system, so the information about the type of the bound variable in the λ-abstraction is not lost. This interpretation extends previous work with Garrel Pottinger [18],[1] which carried through the interpretation for three systems from the Barendregt cube: $\lambda \rightarrow$ [5], $\lambda 2$ [8,19], λC [6], and its extension, the system ECC [14,15].

[1] But the reader will not need knowledge of [18] to understand the present paper.