On the $\lambda$-Definable Tree Operations

Marek Zaionc$^1$

ABSTRACT A $\lambda$-language over simple type structures is considered. The type $T = (0 \rightarrow (0 \rightarrow 0)) \rightarrow (0 \rightarrow 0)$ is called a binary tree type because of the isomorphism between binary trees and closed terms of this type. Therefore any closed term of type $T \rightarrow (T \rightarrow \cdots \rightarrow (T \rightarrow) \cdots)$ represents an $n$-ary tree function. The problem is to characterize tree operations represented by the closed terms of the examined type. It is proved that the set of $\lambda$ definable tree operations is the minimal set containing constant functions, projections and closed under composition and the limited version of recursion. This result should be contrasted with the results of Schwichtenberg and Statman (cf. [Sch75], [Sta79]) which characterize the $\lambda$ definable functions over the natural number type $(0 \rightarrow 0) \rightarrow (0 \rightarrow 0)$ by composition only, as well as with the result of Zaionc (cf [Zai87]) for word $\lambda$ definable functions over type $(0 \rightarrow 0) \rightarrow ((0 \rightarrow 0) \rightarrow (0 \rightarrow 0))$ which are also characterized by means of composition.

Introduction

The $\lambda$ calculus introduced by Church is a calculus of expressions that naturally describes the notion of function. Functionals are considered dynamically like rules rather than set-theoretic graphs. The $\lambda$ calculus mimics the procedure of computation of the program by the process called beta reduction. There is a natural way of expressing objects like numbers, words, trees and other syntactic entities in the $\lambda$ calculus. Therefore dynamic operations on objects of this kind can be described by terms of $\lambda$ calculus. All those objects are of considerable value for computer scientists. For example natural number "n" (Church's numeral) is represented by a functional that returns the $n$-fold composition of its argument. Therefore $\lambda$ terms may be considered as algorithms (programs) on Church's numerals. This is a well known result (proved by Church and Kleene) relating all partial computable functions with $\lambda$ terms. Of course, the notion of partial recursive function can be naturally extended to other structures like words, trees etc. It is natural that the Church-Kleene Theorem might be extended to these structure.

The typed version of $\lambda$ calculus is obtained by imposing the simple types on the $\lambda$ calculus. The problem of representing structures is basically the same in the typed $\lambda$ calculus, however the rigid type structure in the syntax of $\lambda$ calculus dramatically reduces expressiveness of functions on these structures. The first result concerning representability was proved by

$^1$Instytut Informatyki Uniwersytet Jagielloński Kopernika 27, 31-501 Kraków, Poland
Schwichtenberg in 1975 (see [Sch75]). The numerical functions represented in typed \( \lambda \) calculus are exactly the functions generated by the operation of composition from the constants 0 and 1 and functions \textit{addition}, \textit{multiplication}, and \textit{conditional} (extended polynomials). A similar result for word operations is the following: the word functions represented in typed \( \lambda \) calculus are exactly the functions generated by composition from the empty word and operations \textit{append}, \textit{substitution} and \textit{cut}. (Consult [Zai87] for details). Since any homogeneous free algebra may be represented in typed \( \lambda \) calculus in the same fashion as numbers or words, it seems interesting to investigate lambda definability on such algebras.

In this paper we investigate lambda definable operations on the algebra of binary trees. We prove the following characterization of tree functions represented in typed \( \lambda \) calculus: the \( \lambda \) definable tree functions are exactly the functions generated by composition and limited primitive recursion from the empty tree and operation \textit{join}. It is not known to the author if the set of all \( \lambda \) definable tree operations can be generated by composition alone from some finite set of basic tree functions or if it is possible to reduce limited primitive recursion to composition only.

1 Binary Trees and Tree Operations

The set of binary trees \( T \) is defined recursively as follows: \( \varepsilon \) is a tree, and if \( t_1, t_2 \) are trees then \( t_1 \land t_2 \) is a tree. \( \varepsilon \) is called the empty tree and \( t_1, t_2 \) are left and right subtrees respectively of the tree \( t_1 \land t_2 \). We investigate tree operations, \( \text{i.e.}, \) functions \( f : T^n \to T \). We begin by defining several tree operations.

**Definition 1.1.** The tree constructor \( \land \) can be viewed as a function \( \land : T^2 \to T \). \( \varepsilon^n \) is the \( n \)-ary function which maps onto the empty tree \( \varepsilon \) constantly. \( p_i^n \) is the \( n \)-ary projection which extracts the \( i \)-th argument.

\[
\land (x_1, x_2) = x_1 \land x_2 \\
\varepsilon^n(x_1, \ldots, x_n) = \varepsilon \\
p_i^n(x_1, \ldots, x_n) = x_i.
\]

We are going to investigate some classes of tree operations.

**Definition 1.2.** A class \( X \) of tree functions is closed under primitive recursion if the \( n + 1 \)-ary function \( h \) defined by

\[
h(\varepsilon, x_1, \ldots, x_n) = g(x_1, \ldots, x_n) \\
h(s \land s', x_1, \ldots, x_n) = f(h(s, x_1, \ldots, x_n), h(s', x_1, \ldots, x_n), x_1, \ldots, x_n)
\]

belongs to \( X \) whenever functions \( g, f \) are in \( X \).

Let us distinguish four classes of tree operations \( F, F_0, F_1, F_\lambda \).