CS6180 Lecture 23 – Basic Recursive Function Theory in CTT

Robert L. Constable

Abstract

In this lecture we look at the notion of *partial types* in constructive type theory and how we can express the basic ideas of computability theory using such types.

Many ideas presented in this lecture have been implemented in Nuprl. There is much more research to be done in this area which is not a topic that Agda and Coq can yet (if ever) express since they do not support *partial types*.

Most of the basic ideas are from one long article *Computational Foundations of Basic Recursive Function Theory* that Scott Smith and I wrote in 1993 [2]. Here it is abbreviated as CBRFT for *Constructive Basic Recursive Function Theory*. Scott Smith wrote an entire PhD thesis on this subject, *Partial Objects in Type Theory* [7]. This lecture will only discuss highlights to show the style and potential of such result in constructive type theory.

1 Introduction

Two articles that were very influential for the results presented in this lecture are (a) "A machine independent theory of computational complexity" by Blum [1] and (b) "Axiomatic recursive function theory" by Friedman [6]. The key concept needed is the ability to define general recursive functions using a fixed point operator. We call this operator **fix**. This corresponds to an operation in Curry's *combinatory calculus* called the **Y combinator**. The combinatory calculus was developed by Haskell Curry [5, 3, 4, 8]. The book by Stenlund, *Combinators, λ–Terms, and Proof Theory* [8] is excellent background reading for some of the key concepts used in this work.

The main results of the lecture are: (a) the small theory of computability using partial types, and (b) a proof that the halting problem expressed in this theory is unsolvable. The unsolvability theorem is just this: no term in CBRFT solves its halting problem. It is presented here in this form.

Theorem: No term in CBRFT solves the halting problem for functions $f \in \mathbb{N} \rightarrow \bar{\mathbb{N}}$.

Specific computing models such as Turing machines (TMs) or Random Access Machines (RAMs), or functional programs are good models of general computability. However, the more abstract approaches allow us to isolate the key ideas. Before looking at this abstract approach, let's recall how the results are presented using the lambda calculus, a well defined sub-language of constructive type theory.

First we defined the Y-combinator, the name coming from Curry's *combinatory logic* [3] developed in the period that Church was creating the lambda calculus. In lambda notation the Y-combinator is this function.

Definition: *Y-combinator* is the lambda term $\lambda(y.(\lambda(x.y(x(x))))(\lambda(x.y(x(x)))))$.

This is the *fixed point combinator* Y with the property that for any function f, $Y(f) = f(Y(f))$.

Theorem: There is no λ-definable function to decide the halting problem in the untyped λ-calculus.

Proof: Suppose $h(x) = 1$ if x halts and equals 0 otherwise. Define $d = Y(\lambda(x.if\ h(x) = 1 then\ \perp\ else 0))$. Notice that $Y(d) = d(Y(d))$ where $d = if\ h(d) = 1\ then\ \perp\ else\ 0$.

Consider how d executes. If $h(d) = 1$, then d will diverge, but this contradicts the definition of h. If $h(d) = 0$, then $d = 0$, contradicting the definition of h. Therefore no such h can exist.

Qed

We will not work in exactly this way because self application is not typeable. Kleene used indexings, Gödel's approach. We will add a *fix operator*.

2 Constructive Basic Recursive Function Theory (CBRFT)

Here is the essence of a constructive type theoretic account of a constructive theory of computability as presented in [2].

Syntax:

- 0,1,2, ... numerical constants
- $\lambda x.t$ computable functions
- $f(t)$ function application
- $s;t$ sequentialization
- $succ(r)$ successor operation
- $pred(r)$ predecessor operation
- $zero(r;s;t)$ decide on value of r
- $fix(f)$ fixed point operator defined above.
- $\perp$ is $fix(\lambda(x.x))$,
- the *types* are $S, T, S \rightarrow T, \bar{(}S)$

The logical relations are $s = t \in T$, $t \in T$ means $t = t \in T$. To say that a term t converges we write $t\downarrow$, and to say that it diverges we write $t\uparrow$.

Here are the axioms for this small theory.

Ax1: *Function introduction* If $t = t'$ *in* A implies $b[t/x] = b[t'/x]$ *in* B, then $(\lambda x.b = \lambda x.b') = in\ A \to B$.

Ax2: *Function elimination* If $f = f'$ *in* $A \to B$ and $a = a'$ *in* A, then $f(a) = f'(a')$ *in* B.

Ax3: *Bar introduction* $If\ (a'\downarrow\ iff\ a\downarrow)\&(a\downarrow \Rightarrow (a = a'\ in\ A)\ then\ (a = a'\ in\ \bar{A}$.

Ax4: *Bar elimination* $If\ (a = a'\ in\ \bar{A})\&a\downarrow\ then\ (a = a'\ in\ A)$.

Ax5: *Fixed point* $If\ f = f'\ in\ (\bar{A} \to \bar{A})\ then\ fix(f) = fix(f')\ in\ \bar{A}$.

Ax6: Equality is a partial equivalence relation.

Ax7: $\mathbb{N}$ is a type, $\mathbf{1} = 0$ and $\mathbf{2} = 0, 1$.

Theorem: No term in CBRFT solves the Halting Problem for f *in* $(\mathbb{N} \to \bar{\mathbb{N}})$.

Proof: Assume there is a function $h : \bar{\mathbb{N}} \to \mathbf{2}$ such that $h(t) = 1$ iff $t\downarrow$. Show how to derive a contradiction from these hypotheses. *Qed*.

This argument shows that the notion of a partial function in constructive type theory differs in a fundamental way from the classical notion.

References

[1] M. Blum. A machine independent theory of computational complexity. *Journal of the Association for Computing Machinery*, 14:322–336, 1967.

[2] Robert L. Constable and Scott F. Smith. Computational foundations of basic recursive function theory. In *Proceedings of the 3rd IEEE Symposium on Logic in Computer Science*, pages 360–371, Edinburgh, UK, 1988. IEEE Computer Society Press. (Cornell TR 88-904).

[3] H. B. Curry, R. Feys, and W. Craig. *Combinatory Logic, Volume I.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1958.

[4] H. B. Curry, J. Roger Hindley, and Jonathan P. Seldin. *Combinatory Logic, Volume II.* Studies in Logic and the Foundations of Mathematics, Vol. 65. North-Holland, Amsterdam, 1972.

[5] Haskell B. Curry. Functionality in combinatory logic. *Proc National Acad of Science*, 20:584 – 590, 1934.

[6] Harvey Friedman. Axiomatic recursive function theory. In *Logic Colloquium '69*, pages 385 – 404. North-Holland, Amsterdam, 1974.

[7] S.F. Smith. *Partial Objects in Type Theory.* PhD thesis, Cornell University, Ithaca, NY, 1989.

[8] S. Stenlund. *Combinators, λ–Terms, and Proof Theory.* D. Reidel, Dordrechte, 1972.