CS6180 Lecture 22 – Theory of Computability in CTT

Robert L. Constable

Abstract

This lecture will summarise core ideas in the theory of computing, including basic recursive function theory, using concepts from type theory. A fundamental concept is the type of *partial recursive functions* over the natural numbers and other types. The basic results include proofs that certain questions about partial functions are *unsolvable*, such as whether they converge on a specific input. This is the well known *halting problem*. Another question is whether two partial recursive functions are equal on all inputs. There are also famous theorems such as Kleene's *recursion theorem* and the *universal machine theorem*.

The next lecture will examine how these concepts and results are expressed in constructive type theory. We will draw on an axiomatic approach called *Basic Recursive Function Theory* (BRFT).

1 Introduction

The are excellent books on the theory of computing and unsolvability. Many results of this lecture were already presented in Kleene's classic 1952 book *Introduction to Metamathematics* [10]. Another classic book on this topic is *Theory of Recursive Functions and Effective Computability* [12], and this book freely appeals to Church's Thesis which we discuss below. Roger's says boldly on page 21:"Almost all the proofs in this book will use Church's Thesis to some extent." A popular book written for undergraduates is *Computability and Logic* by Boolos and Jeffrey [4]. A comprehensive textbook is *A Course in Mathematical Logic* by Bell and Machover [1].

Hopcroft and Ullman cover Turing Machines in their book *Formal Languages and Their Relation to Automata* [8] widely taught in computer science. Another early book on the topic is the 1,165 page tome *The Foundations of Mathematics* [2]. Another Cornell book on the topic is *Theory of Computation* by Kozen [11], these are his lecture notes from CS6820. An article that was very influential for this lecture is "A machine independent theory of computational complexity" by Blum [3]. Another direct influence was the article "Axiomatic recursive function theory" by Friedman [7].

Turing's definition of an idealized computer, now called a *Turing machine*, was presented in his 1937 article "On computable numbers, with an application to the Entscheidungs problem" [13]. He defines his machines in 38 pages. He proposed a definition of the computable real numbers using them. He also showed that it is not possible to use a Turing Machine to decide whether a another such machine working on a task will halt. He used a version of Cantor's diagonal method to show this. That is a fundamental technique in the study of computability just as it is in set theory. Turing also notes this:"The theorem that all effectively calculable (λ-definable sequences are computable

and its converse are proved below in outline." Here he connects his notion of computability with that being studied by Church and Kleene using the lambda calculus, the basis for modern functional programming languages.

In the same period, Alonzo Church was developing his model of the computable functions as essentially functional programs expressed using his lambda (λ) calculus. Church was Turing's thesis advisor, and he convinced Turing to show how to translate his lambda terms (functional programs) into Turing machine code – perhaps the first compiler.

1.1 Primitive recursive functions and beyond

Before the development of the lambda calculus and Turing machines, researchers were studying primitive recursive functions and their extension to more complex recursive definitions. Researchers were defining *hierarchies of recursive definitions* indexed by ordinal numbers as a shown in supplementary reading for this lecture. A motivating idea was to extend these hierarchies by allowing more expressive forms of recursion. This approach did not imagine a comprehensive class of computable functions as defined by Turing and Church, although later Gödel defined a class of recursive functions building on work of Herbrand. These were called Herbrand-Gödel recursive functions.

1.2 Properties of algorithms

Rogers starts his analysis of computability by discussing informally the characteristics of an algorithm and the notion that a function is computable by an algorithm. Here are the ten properties he lists.

1. An algorithm is given by a finite set of *instructions*.
2. There is an *agent* to carry out the instructions, call it L.
3. There are facilities for for storing the code and executing the steps.
4. Executing the steps is a *discrete process*.
5. L reacts to the code deterministically.
6. There is no fixed bound on the size of the inputs.
7. There is no fixed bound on the size of the instructions.
8. There is no fixed bound on the memory (storage) used to compute.
9. There is no fixed bound on the capacity of the computing agent.
10. There need not be an *a priori* bound on the length of the computation.

Rogers then goes no to illustrate these properties using the well studied class of *primitive recursive functions*. He claims that virtually all algorithmic functions in ordinary mathematics are primitive recursive, but mentions Ackermann's famous example of one that is not – obtained by majorizing all the primitive recursive functions.

Rogers also points out that a function given by an algorithm can be partial in that it is not defined on all inputs. He then takes Turing Machines as basic, already by page 21, and he proposes that they can be "indexed," that is, the can be enumerated. He calls this Theorem 1. He denotes the functions by their index, as ϕ_i.

He next notes that it is easy to create slightly different versions of each ϕ_i by saving some values in a table. So there are an unbounded number of indexes for each ϕ_i. He also notes that it is easy to show that at least classically there are many more non-computable functions since Cantor showed there are uncountably many number theoretic functions of one variable.

In his Theorem IV he shows that there is a universal function for all n-argument partial recursive functions. For one argument, he defines $\phi_z(x, y) = \phi_x(y)$. This is called the *enumeration theorem* in some literature. There is a version for each arity (number of inputs).

It is now easy to state the theorem that the *halting problem is undecidable*. This is Thm VII of Rogers. We give a simpler proof. We need the notation that $\phi_x(y) \downarrow$ means that the function ϕ_x halts on its input y. We diagonalize over the list of one argument partial recursive functions. For each ϕ_i, we ask whether $\phi_i(i)$ halts. If it does then we define the recursive function d such that $d(i)$ diverges. If it does not halt, then we define $d(i)$ to have value 0. This gives us a computable function which is not equal to any ϕ_i. But that list is assumed to enumerate every partial recursive function of one argument. So there can't be a partial recursive function that decides halting.

Theorem: There is no recursive function h such that $\forall x, y.h(x, y) = if\ \phi_x(y) \downarrow\ then\ 1\ else\ 0$.

1.3 An argument against Church's Thesis

Not everyone accepts the Church/Turing Thesis, which we call here Church's Thesis. In particular, Kalmar [9] wrote a short article in 1959 entitled "An Argument Against the Plausibility of Church's Thesis." We will look at this argument and at other opinions about Church's Thesis. Recent results about the intuitionistic type theory implemented by Nuprl actually refute CT, which makes me uneasy since Prof. Church was my undergraduate thesis advisor and started me on the path to Cornell and to type theory. Here is the argument that Kalmar gives. He says to consider a general recursive function g and consider a computable but non-Church/Turing computable function defined as follows.

$$\phi(x) = \mu y(g(x, y) = 0) = least\ y\ such\ that\ g(x, y) = 0\ else\ 0\ if\ no\ such\ y.$$

He allows us to decide the case that there is no y such that $g(x, y) = 0$ *by proof.*

In intuitionistic type theory as implemented in Nuprl, we now allow Brouwer's free choice sequences as computable functions, and these are not general recursive. We also allow distributed processes, which we will examine in this course. We argue that these are also not general recursive yet necessary in defining distributed computing tasks, as we will see.

In the next lecture we will look at a way to formalize the standard results from *basic recursive function theory* in type theory. We built on work of Harvey Friedman [7], calling our paper Computational foundations of basic recursive function theory (BRFT) [6, 5].

References

[1] John Bell and Moshe Machover. *A Course in Mathematical Logic*. North-Holland, New York, 1977.

[2] Evert W. Beth. *The Foundations of Mathematics*. North-Holland, Amsterdam, 1959.

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

[4] G. S. Boolos and R. C. Jeffrey. *Computability and Logic, Third Edition*. Cambridge University Press, 1988.

[5] Robert L. Constable and Scott F. Smith. Computational foundations of basic recursive function theory. *Journal of Theoretical Computer Science*, 121(1&2):89–112, December 1993.

[6] Robert L. Constable and Scott Fraser Smith. Partial objects in constructive type theory. In D. Gries, editor, *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science*, pages 183–193. IEEE Computer Society Press, June 1987.

[7] Harvey M. Friedman. Axiomatic recursive function theory. In R.O. Gandy and C.M.E. Yates, editors, *LOGIC COLLOQUIUM '69*, volume 61 of *Studies in Logic and the Foundations of Mathematics*, pages 113 – 137. Elsevier, 1971.

[8] John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley, Reading, Massachusetts, 1969.

[9] Laszlo Kalmar. An argument against the plausibility of Church's thesis. In A. Heyting, editor, *Constructivity in Mathematics. Amsterdam: North-Holland.*, pages 72 – 76. North-Holland, 1959.

[10] S. C. Kleene. *Introduction to Metamathematics*. D. Van Nostrand, Princeton, 1952.

[11] Dexter Kozen. *Theory of Computation*. Springer, New York, 2006.

[12] Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill Book Company, New York, 1967.

[13] A. M. Turing. On computable numbers, with an application to the Entscheidungs problem. In *Proceedings London Math Society*, pages 116–154, 1937.