Basic Reading on Computable Functions

Peter Smith
ps218@cam.ac.uk

January 15, 2012

This is an annotated reading list on the beginning elements of the theory of computable functions. It was structured so as to complement the first eight lectures of Thomas Forster's Part III course in Lent 2011, and should fit his Lent 2012 course equally well (see the first four chapters of his evolving handouts).

1 A rather basic general bibliography (in chronological order)

This list reflects what happens to be on my shelves, though I hope it is pretty representative. Corrections and suggestions for improvements/additions are welcome (particularly, perhaps, suggestions of notably good materials freely available online).

1. Rósza Péter, *Recursive Functions*, Academic Press 1967. This is the English translation of a book by one of those who was 'there at the beginning'; it has that old-school slow-and-steady un-flashy lucidity that makes it still a pleasure to read. Remains very worth looking at.

2. Hartley Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill 1967. A heavy-weight state-of-the-art-then classic, written at the end of the glory days. Speedily gets advanced. But the opening chapters are still excellent and are action-packed.

3. Herbert B. Enderton, 'Elements of Recursion Theory', in J. Barwise, ed., *Handbook of Mathematical Logic*, North Holland 1977. A helpful 40 page overview of the territory.

4. Nigel Cutland, *Computability: An Introduction to Recursive Function Theory*, CUP 1980. Rightly much-reprinted, this is a beautifully clear maths textbook, and notably more 'modern' in feel that the previous items. The first seven chapters develop and extend the material in TF's opening lectures.

5. Piergiorgio Odifreddi, *Classical Recursion Theory*, Vol. 1. North Holland 1989. Well-written, discursive: just the first two chapters amplifying TF's introductory lectures cover some 250 pages. You'll learn a lot!

6. Richard Epstein and Walter Carnielli, *Computability: Computable Functions, Logic, and the Foundations of Mathematics*, Wadsworth 2nd edn. 2000. A rather more introductory book, clearly done, and recommended because it is better than some on e.g. multiple recursion. Read Part II of the book.

7. A. Shen and N. K. Vereshchagin, *Computable Functions*, American Math. Soc. 2003. A lovely, elegant, little book in the AMA's 'Student Mathematical Library' – recommended for giving a differently-structured tour through the big ideas.

8. S. Barry Cooper, *Computability Theory*, Chapman & Hall/CRC 2004. A very nicely done modern textbook (a second edition is due in 2012). Read Part I of the book (about the same level of sophistication as the recommended chapters of Cutland, but with some extra topics).

9. George Boolos, John Burgess, Richard Jeffrey, *Computability and Logic*, CUP 5th edn. 2007. This is the latest edition of an absolute classic. The first version (just by Boolos and Jeffrey) was published in 1974; and there's much to be said for the 1990 third edition being the best. The later versions have been done by Burgess and have perhaps lost elegance and some of the individuality. But the first eight chapters (forming the first part of the book, on 'Computability Theory') are still terrific – more discursive and a bit less sophisticated than Cutland – and they introduce the right material.

10. Peter Smith, *An Introduction to Gödel's Theorems*, CUP 2009. This is the fourth, heavily corrected, reprint of my book first published in 2007 (make sure you get use this later version, of which I've given a copy to the Moore Library). Don't be put off by the series title 'Cambridge Introductions to Philosophy': putting it in that series was the price I paid for cheap paperback publication. The relevant chapters are 11 on primitive recursive functions, and 29, 31–35 on μ-recursiveness, Turing machines, etc.. These short chapters obviously can't get very far, but they could prove initially helpful, and there are some very neat proofs, although I say so myself! (There's some significance, by the way, to the fact that I can deal with primitive recursive functions and μ-recursive functions over 150 pages apart: for some purposes, e.g. if we are interested in Gödelian incompleteness phenomena, facts about primitive recursive functions are all that we initially need.)

If you are going to read just one of these, I'd probably suggest Cutland's book. But it is always good to criss-cross over new mathematical territory coming at things from slightly different angles, to get a better sense of how things hang together.

It has to be said that these recommendations won't strike Part III mathmos as conceptually very complicated or mathematically very sophisticated or tricky (though reading on in Rogers, Odifreddi or Cooper will get you to serious stuff). But that's the nature of the beast: elementary computability theory is indeed conceptually very neat and natural, and the early Big Results are proved in quite remarkably straightforward ways (just get the hang of the basic diagonalization construction, the idea of Gödel-style coding, and one or two other tricks and off you go . . .).

The rest of this reading list makes some topic-by-topic suggestions, with varying degrees of detail.

2 History

Textbook mathematics is often presented in a take-it-or-leave-it kind of way, with few hints as to the historical evolution of the central concepts. But a smidgin of historical

knowledge about the evolution of work on X can be helpful in giving some initial orientation and a sense of why the contemporary theory of X has the shape it has. I think that this is particularly so in the case where X = computability. So some historical reading:

1. Cooper's short first chapter on 'Hilbert and the Origins of Computability Theory' gives some of the headlines.

2. Richard Epstein offers a *very* helpful 28 page timeline on 'Computability and Undecidability' at the end of the 2nd edn. of Epstein/Carnielli.

3. Robert I. Soare, 'The History and Concept of Computability', in E. Griffor, ed., *Handbook of Computability Theory*, Elsevier 1999. A more detailed and very illuminating 30 page essay.

4. Robin Gandy, 'The Confluence of Ideas in 1936' in R. Herken, ed., *The Universal Turing Machine: A Half-century Survey*, OUP 1988. Seeks to explain why so many of the absolutely key notions all got formed in the mid-thirties.

3 Primitive recursion, and a computable function which isn't p.r.

3.1 Primitive recursive functions – TF §3.1, 3.2.1

Some texts start by characterizing the full class of computable functions and only later isolate the primitive recursive (p.r.) functions as a subclass. But historically, the p.r. functions were described first and only later did we get a more general account.

1. I'd recommend – well, I would, wouldn't I? – starting with Smith, *IGT* Chap 11. This clearly defines the class of p.r. functions and (a signficant omission from TF's 2011 notes) gives the standard *diagonalization argument* to prove that there are intuitively computable functions which *aren't* p.r. (my Theorem 11.1). Also defines what it is for properties or relations to be p.r., and says a bit about how to build new p.r. functions out of old ones using bounded minimization, bounded quantification etc.

2. Epstein and Carnielli, Chap 11, is another clear basic treatment.

3.2 Why defining functions by recursion is kosher – TF §3.1.2

[Extracted from one of my handouts, as people sometimes don't quite get the point of why we might *want* a proof.]

Consider how we standardly characterize the factorial function $fact\colon N \to N$ using the recursion equations

i. $fact(0) = 1$,
ii. $fact(m+1) = (m+1) \cdot fact(m)$.

Tritely, clause (i) gives us the value of $fact(0)$ and then we can use clause (ii) to calculate $fact(1)$ in terms of $fact(0)$, and $fact(2)$ in terms of $fact(1)$, etc., etc. So, intuitively, the clauses uniquely fix the value of the function for every natural number argument.

So far, so obvious. Now we *could* just stop there: we could take it as a bedrock truth about the natural numbers that the intuitive argument here is sound – i.e. we could accept it as evident that recursive equations such those characterizing *fact* do indeed well-define functions, and not worry about further analysis. That plainly would not lead us astray in informal arithmetic.

However, as Dedekind was the first to see, we don't *have* to take the success of recursive definitions in pinning down functions as a *new* fact about the numbers: rather we can use the induction principle for natural numbers to prove the acceptability of any properly constructed such definition. And as Dedekind also saw (and Peano famously missed), if we have purported to give a full axiomatic characterization of the numbers in terms of their satisfying the Dedekind/Peano axioms, then we *need* such a proof to warrant going on to use recursive definitions to introduce new functions inside our axiomatized theory.

So let's use induction to prove, as a sample case, that *there exists a unique function, fact*$: N \to N$*, which satisfies the recursion equations (i) and (ii).* Here's the strategy. The first stage is to show that for any finite n, the recursion equations do properly characterize a function defined over the numbers up to n. In symbols, if $N_{\leq n}$ are the numbers less than or equal to n, then for any n, the equations *do* define a unique function $f_n\colon N_{\leq n} \to N$. Then, at the second stage, we put $fact(n) = f_n(n)$. *This* function is now defined for every number, and we can show that this it does indeed satisfy the recursion equations for all n. That gives us the existence claim in our theorem; the uniqueness claim quickly follows.[1]

The argument then generalizes in the obvious way to show that other functions specified by recursion invoking already-known well-defined functions are themselves well-defined.

3.3 The Ackermann-Péter function and simultaneous recursion – TF §§3.3.1, 3.3.2

The diagonalization argument shows that not all intuitively computable functions are p.r. (a thought that should in any case look plausible when we note that p.r. functions can be computed using only 'for' loops, but we allow procedures using open-ended 'do until'

[1]For the record, at the first stage of the proof define $F \subseteq N$ by putting $n \in F$ just if, for every $j \leq n$, there is a function $f_j\colon N_{\leq j} \to N$ which (1) satisfies equation (i), (2) satisfies equation (ii) for any $m < j$, and further (3) if $j < n$, $f_{j+1}(j+1) = (j+1) \cdot f_j(j)$. Then

(a) Trivially, $0 \in F$ – we just define $f_0(0) = 1$.

(b) Suppose that $n \in F$. Then define f_{n+1} over $N_{\leq n+1}$ by putting $f_{n+1}(k) = f_n(k)$ for $k \leq n$, and $f_{n+1}(n+1) = (n+1) \cdot f_n(n)$. This evidently ensures that (1), (2) and (3) still hold for $n+1$. Hence $n+1 \in F$

Hence by the induction principle, for all n, $n \in F$: so we can legitimately talk of f_n for any n.

Second stage. Now put $fact(n) = f_n(n)$. The evidently, $fact(0) = f_0(0) = 1$, and $fact(n+1) = f_{n+1}(n+1) = (n+1) \cdot f_n(n) = (n+1) \cdot fact(n)$. So *fact* obeys the recursion equations (i) and (ii).

Moreover, this function is unique. For suppose *fact* and *fact′* are two different functions that both satisfy the recursion equations. Then – by induction, in the form of LNP – there is a least number for which they differ in value. That least number can't be zero since, by clause (i), $fact(0) = 1 = fact'(0)$. So it must be some $n+1$, where $fact(n) = fact'(n)$. But then, by clause (ii), $fact(n+1) = (n+1) \cdot fact(n) = (n+1) \cdot fact'(n) = fact'(n+1)$, contradicting the definition of $n+1$.

So, by induction, we have somewhat laboriously shown that there indeed exists one and only one function satisfying the recursion equations (i) and (ii)!

loops to count as computations). What, though, is a 'nice' example of a computable-but-not-p.r. function?

The standard example is the Ackermann-Péter function, which 'grows too fast to be primitive recursive'.

1. Smith, *IGT* §§29.3–29.4 gives motivation, proves the definition specifies a terminating computation (fn. 4 in §29.3), shows the function is indeed μ-recursive (using a standard coding argument), but only gives a proof-sketch that the function is not p.r.

2. If you want to fill out that sketch, look at TF's notes, or why not look at the classic treatment by Rósza Péter herself, in §9 of her book, 'Example of a number-theoretic function which is not primitive recursive'?

3. Chs 12 and 13 of Epstein/Carnielli treat the Ackermann-Péter function nicely, and then press on to say more about multiple recursion (which is also discussed by Péter in her §§10, 11).

3.4 Showing the (primitive) recursive functions are arithmetic, using the β-function trick – TF §3.1.2

The standard language of arithmetic L_A (the language of first-order Peano Arithmetic) has as primitive non-logical vocabulary just the constant term 0, a one-place function S (for successor), together with $+$ and $\times$.[2] However we can, surprisingly, show that every p.r. function can be expressed in this spartan language. That is to say, we can show that for any one-place p.r. function f we can define an open L_A-wff $\varphi(x, y)$ such that $\varphi(\overline{m}, \overline{n})$ is true iff $f(m) = n$ (here $\overline{m}$ is L_A's standard numeral for m, i.e. $SSS\ldots S0$ with m occurrences of S). Similarly for many-place functions. For a neat proof see Smith *IGT*, §§13.3–13.5.

Note, by the way, that the proof readily extends to show that every μ-recursive function can be expressed in L_A. That's shown in Smith *IGT*, §30.1.

4 μ-Recursive functions – TF §3.3

By this stage, we know that not all intuitively computable functions are primitive recursive. Question: what function-building constructor can we add to function-composition and definition-by-recursion to get more intuitively computable functions (hopefully, all of them)? Answer: minimization – a open-ended search operation which returns the first number which satisfies some condition.

But now there is an important choice to be made. Do we (1) require minimization to be invoked only when the condition in question *will* eventually be satisfied, so the search for the first satisfying number will return a value; or do we (2) allow more free-wheeling use of minimization, so the condition may never be satisfied? If we take option (1), then the application of minimization still keeps us in the class of total computable functions; if we take option (2), then we are broadening out to consider the class of

[2] It would plainly be as good to start with what TF calls the ring language with non-logical vocabulary 0, 1, + and ×.

partial computable functions.[3] Historically, the first approach was the initial one; but Kleene c. 1938 came to see the mathematical advantages of the liberalized approach.

1. Smith *IGT*, Ch. 29, defines μ-recursive functions the first way. (And this is in fact just fine if our main interest is in major limitative results about incompleteness, undecidability, etc.)

2. But TF takes the more standard second line. Important: do read the short chapters 14 and 15 of Epstein/Carnielli for some crucial explanations and crucial warnings. Nomenclature hereabouts can be varied: I think Epstein/Carnielli exemplify best practice in how they distinguish partial recursive and general recursive (and so for them, the equivalence of general recursive and total partial recursive is a theorem, not a definition).

OK, having added minimization to our stock of constructors for computable functions, is there more to do? Or can we now capture all computable numerical functions? Church's Thesis says yes, intuitively computable functions are μ-recursive.

Reality check: make you sure you understand why we can 'diagonalize out' of the class of p.r. functions but not out of the μ-recursive functions (*IGT*, §29.6).

5 Machines

5.1 Turing machines

As TF says, when he moves to talking about abstract computing machines, 'it does not matter what [reasonable] kind of architecture our machines have as long as they have unbounded memory and can run indefinitely'. He goes on, then, to discuss register machines, about which more in just a moment. But you should certainly know what a Turing machine is!

1. http://plato.stanford.edu/entries/turing-machine/ is a good introductory survey encyclopedia article (which also outlines proofs of some basic results).

2. Smith *IGT*, Ch. 31, defines Turing machines, and then Ch. 32 elegantly proves that every μ-recursive function is (total) Turing computable and vice versa (the proof strategies generalize to deal with partial functions, if you prefer).

3. Also read Ch. 3 of Boolos, Burgess and Jeffrey.

4. Why not look at Turing's original 1936 paper, 'On Computable Numbers, with an Application to the *Entscheidungsproblem*'? Reprinted in e.g. B. Jack Copeland, ed, *The Essential Turing* (and available online, *Proc. London Math. Soc.* (1937) s2-42(1): 230-265).

5.2 Register machines – TF §§4.2, 4.3

TF's preferred machine formalism:

1. Cutland, Chs 1 and 2, discusses register machines nicely.

[3] Warning! Some authors use 'p.r.' for 'primitive recursive', some for 'partial recursive'.

2. Boolos, Burgess and Jeffrey, Ch. 5, discuss similar machines under the label 'abacus machines', and show, inter alia, that all recursive functions are abacus computable, and all abacus computable functions are Turing computable.

5.3 Other approaches ...

For other general approaches to computation, see e.g. Cutland §3.5 (Post/Markov) and Odifreddi, §I.6 (prefiguring TF's lecture 9 on λ calculus).

5.4 The Church-Turing Thesis – TF §1

Church conjectured that all intuitively computable functions are recursive; Turing argued that all intuitively computable functions are computable by a Turing machine. But we know that the recursive functions are just the Turing computable functions. Hence we can (and should!) speak of the Church-Turing Thesis. It gains further support from the fact that other attempts to characterize the computable functions locate the same class of recursive/Turing computable functions. For discussion see

1. Smith *IGT* §§29.5, 29.7 and Ch. 34, gives a basic discussion. Do note the important point made in 29.7, distinguishing the labour-saving and the interpretative uses of the Thesis.

2. See http://plato.stanford.edu/entries/church-turing/ for an excellent survey article by B. Jack Copeland.

3. Smith *IGT* Ch. 35 pursues the vexed question of how far the Thesis can be 'proved'.

6 Coding, and universal machines – TF §4.3.1

Given some nice machine formalism, we can number off possible machines, e.g. by coding their programs using a single Gödel-number e. This is an absolutely key idea. As TF noted in lectures, it is the resulting double-role of numbers – as (codes for) machines and the input/output data for machines – that enables the proofs of key results.

A first consequence: there will be a 'universal' machine that in effect simulates any machine by taking input e, decoding, applying the resulting program to input $\vec{n}$. For more on this, see

1. Cooper, Ch.4, 'Coding, self-reference and the Universal Turing Machine'.

2. Cultand, Chs. 4 and 5. (Incidentally, Cutland Ch. 4 is a nice source for the S-m-n Theorem, that TF mentions in his §4.6.1.)

7 The undecidable and the uncomputable

7.1 Decidable and semi-decidable sets – TF §4.4

For definitions and some elementary results (e.g. a set of numbers is a domain of a partial recursive function if and only if it is the range of a total recursive function), see

1. Shen and Vereshchagin, Ch. 1: brief but nicely done.

For more detailed treatments, see

2. Odifreddi, §II.1.

3. Boolos, Burgess and Jeffrey, Ch. 7.

4. Rogers, Ch. 5 (covers the projection theorem TF mentions).

7.2 Aside on Craig's Theorem – TF §4.5.1

Suppose a theory T is recursively axiomatized; then its theorems form a recursively enumerable set (generate 'in alpabetical order' the strings of T-symbols, filter out those which are T-proofs – which can be mechanically decided if T is recursively axiomatized – and write down the last wffs in the remaining proof arrays). Conversely, any recursively enumerable set of sentences has a recursive axiomatization (indeed a primitive recursive one), which uses only the non-logical vocabulary that occurs in those sentences. That converse is Craig's Theorem. For a proof, see Smith, *IGT* §19.1.

This has some interest in the philosophy of science. Suppose T is a nicely axiomatized scientific theory, and suppose O is selected out from T's vocabulary as being the 'observational' vocabulary. Then T's observational implications – the theorems couched just in O – will evidently form an r.e. set; so these implications can be re-axiomatized in a new theory which uses only vocabulary O. This seems to imply that T's non-observational vocabulary (and the theoretical entities it is about) is strictly speaking dispensable. [Philosophy tripos question: is that right?]

7.3 The unsolvability of the halting problem – TF §4.6

This is a key result about what *can't* be done algorithmically: there is no computational procedure that will decide, given e, n, whether machine e will halt when given input n or churn on for ever.

1. Smith, *IGT*, §§33.1, 33.2. In §33.6 I also prove a version of Kleene's Normal Form Theorem.

2. Boolos, Burgess and Jeffrey, Ch. 4. Also shows that the so-called Busy Beaver function is uncomputable.

3. Cooper, Ch. 5. Also proves Kleene's Normal Form Theorem and the uncomputability of the Busy Beaver function.

7.4 Rice's theorem – TF §4.6.1

Cooper §7.1 and Cutland §6.1 both cover this, but you'll have to backtrack in the books to pick up on their (pretty standard) notation.

7.5 The *Entscheidungsproblem*

The *Entscheidungsproblem* asks whether there is a mechanical procedure for deciding what is a theorem of first-order logic. The problem was posed by Hilbert in the 1920s and was shown to be unsolvable by Turing and Church. There are two different proofs of this result in Smith, *IGT*, §30.5 and §33.3; the second one uses the unsolvability of the halting

problem but also imports assumptions that go beyond the material covered or gestured to in in TF's initial lectures. There's a more stand-alone derivation of the unsolvability of the decision problem for logic from the unsolvability of the halting problem for Turing machines in Boolos, Burgess and Jeffrey, §11.1.

7.6 Gödelian incompleteness

All this, of course, relates intimately to Gödelian incompleteness results; for which see *IGT*, in particular – at least in connection with this course – §33.4 and §33.7. See also Ch. 8 of Cutland, etc. Lectures that I gave on Gödel as a supplement to TF's in 2011 are available at http://www.logicmatters.net/resources/pdfs/PartIII.pdf