Introduction to higher-order computation

Nordic Logic School, Stockholm, 2017

Martín Hötzel Escardó
Theory Group, School of Computer Science
University of Birmingham, UK

Version of Thursday 10th August, 2017

Abstract

These notes complement the lectures. They will be under construction until the course ends.

1 Introduction

First-order computability discusses which functions $\mathbb{N} \to \mathbb{N}$ are computable. Or which functions mapping strings to strings are computable. Or which functions mapping finite objects to finite objects are computable.

Higher-order computability discusses which functions involving infinite objects, such as infinite strings, real numbers, and even functions themselves, etc., are computable. And, more importantly, how to compute them. In practice, computation with infinite objects often takes place in languages such as ML, Haskell, Agda etc. In theory, some canonical systems are Gödel's system T, Platek-Scott-Plotkin PCF, Martin-Löf's dependent type theory, among many others. We will begin to study the theory, but also use practical languages for illustration.

But how can we (or a computer) compute with infinite objects, given that we have a finite amount of time and a finite amount of memory and a finite amount of any resource? Topology comes to the rescue, and much of what we will see in the course revolves around the dichotomy finite vs. infinite, mediated by topology. We can say that topology is precisely about the relation between finiteness and infiniteness that is relevant to computation. We will see surprising examples of computations with infinite objects, which at first sight are not possible, until we see their topological explanations.

2 Pre-requisites

1. Simultaneously attend the course on categorical logic. The course on proof theory will be useful for some aspects of our discussions, particularly regarding ordinals.

2. Some mathematical and logical maturity.

3. Basic computability theory (Turing machines, (primitive) recursive functions, Halting Problem, etc.).

3 Course contents

This will be adjusted as the course proceeds (removing or adding topics, according to the pace of the lecturer and the students). This is a list of topics I intend to cover, rather than a temporal organization of the course.

1. Some basic examples of infinite objects we would like to compute with, including ((infinite) sequences of natural numbers, real numbers represented as infinite sequences of digits, functions on real numbers (e.g. we may wish to integrate them), etc.).

2. Continuous and uniformly continuous functions $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ and $2^{\mathbb{N}} \to \mathbb{N}$.

   Here $\mathbb{N}$ is the set of natural numbers, 2 is the set $\{0, 1\}$ of binary numbers, $\mathbb{N}^{\mathbb{N}}$ is the set of sequences of natural numbers (equivalently the set of functions $\mathbb{N} \to \mathbb{N}$), and $2^{\mathbb{N}}$ is the set of binary sequences (or functions $\mathbb{N} \to 2$).

3. Some languages for computation with infinite objects:

   (a) Gödel's system T.

      i. Examples of computations with infinite objects.
      ii. Set-theoretic model of system T.
      iii. Proof that definable functions $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ are continuous, and definable functions $2^{\mathbb{N}} \to \mathbb{N}$ are uniformly continuous.
      iv. Topological model of system T.

   (b) Platek–Scott-Plotkin PCF.

      i. Now the language becomes much more powerful. What can we do in PCF that we can't do in system T?
      ii. Domain-theoretic model of PCF.
      iii. "Parallel functions", and two Turing-completeness results for PCF.

   (c) Martin Löf Type Theory (MLTT). Only briefly. We revisit some examples and models, and explain what we gain.

4. Some practical counter-parts (Haskell, Agda) for the sake or writing and running some examples discussed above. This will be developed in parallel with the above theoretical languages.

5. Infinite sets that can be exhaustibly searched in finite time, and their topological compactness.

6. The close connection between topology and computation.

7. Discussion of further directions.

4 Supporting external material

Including material that goes much beyond the lectures, for the interested students.

1. John Longley and Dag Normann's Higher-Order Computability book.

   This is a fairly comprehensive book.

2. A normalization proof for Gödel's system T by Thierry Coquand.

   This is a short note with a proof.

3. Gordon Plotkin's PCF paper.

   This defines the language PCF, gives computation rules for it, gives a domain model for it, relates the model to the computation rules, and proves that all computable functionals, of any order, are computable, if we extend PCF with "parallel or" and "exists".

4. Domain-theoretic foundations of functional programming by Thomas Streicher.

   This is a relatively short textbook covering the above about PCF, and more.

5. Domain theory by Achim Jung

6. Computability via PCF by Peter Dybjer

7. Lazy functional algorithms for exact real functionals by Alex Simpson

   This shows how to use exaustive search over the Cantor space $2^{\mathbb{N}}$ to compute Riemann integrals of continuous functions on the reals in PCF.

8. Computability on the partial continuous functionals by Dag Normann.

   This shows that parallel or and exists are not needed to show that all computable (hereditarily) total functions are PCF definable. This is a rather hard paper. In the course I will only formulate precisely what is proved, but there won't be enough time to cover the proof.

9. Fifty years of continuous functionals (slides) by Dag Normann.

   This gives a survey of the historical development of the subject.

10. Learn you a Haskell for Great Good! by Miran Lipovača

    If you want to write and run in practice some of the examples we will do. Actually I will write some of the examples in Haskell. If you have time, it is probably a good idea to try to read some of this entertaining book. It may also help to understand the theoretical programming languages that we are going to study.

11. Per Martin-Löf's intuitionistic type theory (Bibliopolis 1984).

    We will cover only a very small fraction of this, as a taster.

12. Introduction to Agda ("Dependent Types at Work") by Ana Bove and Peter Dybjer (and many more references).

    And this can be considered as a more practical counter-part of Martin-Löf type theory, which I may also use for the sake of illustration in the course.

13. A constructive manifestation of the Kleene–Kreisel continuous functionals by Chuangjie Xu and myself

    There are many equivalent definitions of the Kleene–Kreisel continuous functionals. This is yet another one. It is both simple and constructive.

14. Abstract data types for real numbers in type theory, by Alex Simpson and myself.

    This shows how to compute with real numbers in Gödel's system T.

15. Topology via logic by Steve Vickers.

16. Some papers, notes and other material by myself:

    (a) Continuity of Godel's system T functionals via effectful forcing paper.

    (b) Topology for functional programming material.
        This is material for a short course. We will use a bit of that.

    (c) Exhaustible sets in higher-type computation paper.
        This shows how to write exhastive search programs for certain infinite sets, and that the sets for which such programs can be written must be topologically compact.

    (d) Infinite sets that satisfy the principle of omniscience in any variety of constructive mathematics paper.
        Among other things, the above shows that in system T we can exhaustively search sets of Cantor-Bendixon rank $< \epsilon_0$, conjecturing that this is the best we can do in system T.

    (e) Synthetic topology of data types and classical spaces "Barbados notes".

(f) Computing with real numbers represented as infinite sequences of digits.

This is both a Haskell program and lecture notes together.

17. On the Cantor-Bendixon rank of a set that is searchable in G̈dels T by Dag Normann.

This shows that any exhastibly searchable set in System T must have Cantor-Bendixon rank $< \epsilon_0$, given a converse of a result mentioned above.