Computational Models

Tutorial

Dr S. S. Chandra

Cellular Machines

Understanding how the modern-day computer is designed and structured is important for creating and using it for computer science. Especially when related to what a computer might look like into the future. John von Neumann designed a practical implementation of a Turing machine we use today, which he later extended to a broader theory of computation involving self-replicating machines. I have written a chapter on this in my book (see Chapter 4, v0.8 or later), which I recommend for answering the questions below.

Section I – Von Neumann Architecture

Using the formal definition of Turing machines, describe how its key components are found in the modern-day computer and what changes to Turing machines were introduced by John von Neumann in his EDVAC report to create the computer architecture that we use today.

Section II – Self-replicating Machines

Describe how John von Neumann took the concept of universal Turing machines in order to develop machines that can create copies of itself. Ensure that your description includes brief details on the type of implementation that he designed and what evidence there is of its successful implementation.

Lambda Calculus

Lambda calculus is a model of computation that is equivalent to Turing machines (as per the Church-Turing thesis). Lambda calculus, with which the λ (lambda) symbol in computer science is synonymous, reduces all computation to the basic operations involving only functions and its application.

Quoting *Types and Programming Languages* (Benjamin C. Pierce, 2002), it can be seen ``simultaneously as a simple programming language in which computations can be described and as a mathematical object about which rigorous statements can be proved.''. It has wide applications in computer science, namely in programming language theory (which is also a popular research area at UQ and in Australia!). Lambda calculus has inspired many old and modern programming languages, and is notably the basis of functional programming, a programming paradigm in which entire programs are made using functions; functional programming languages that you might know are Lisp, Haskell and OCaml. All other modern programming languages have now borrowed concepts from functional programming, through the use of a lambda keyword in one form or another including C++ (in the 2011 standard), Javascript and even Python. You can find more details in Chapter 7, section 4 of (Moore and Mertens, 2011).

Section III – Lambda Calculus

For the following exercises, you must first come up with a Lambda calculus statement for the corresponding function by hand, and then try it out using Mikrokosmos; you can use inbuilt LaTeX or Markdown interpreter in Jupyter notebooks to annotate your work with Mikrokosmos.

You may refer to Mikrokosmos' standard library and tutorials to do some of these trivially, but if you've done the tutorials on Mikrokosmos and learnt its syntax, then you can simply use the representations on the Wikipedia page as Mikrokosmos has a one-to-one correspondence (for this exercise).

[See the relevant sections of the Mikrokosmos Tutorial for hints]

1. Define the numbers 0 through 4 with Church encoding using the identity function and the successor function.

2. Define separate functions that doubles its argument, adds two numbers, and multiplies two numbers.

3. Define the logical operators true and false in lambda calculus.

4. Define the AND, OR and NOT operators in lambda calculus.

5. Define the Y-combinator in Lambda calculus, and give a brief explanation on how it works.

References

Benjamin C. Pierce, 2002. Types and Programming Languages, The MIT Press. The MIT Press, Cambridge, Mass.

Moore, C., Mertens, S., 2011. The Nature of Computation, 1 edition. ed. Oxford University Press, Oxford New York Auckland Caoe Town Dar es Salaam.