Modelling Computation of Functions II
The λ-Calculus

The ***λ-calculus*** is sometimes described as "the world's smallest programming language". The simplest version of this has no numbers, strings, Boolean constants, or any non-function data type — but it still can be used to define the same class of "computable" functions as Turing machines. This model of computation was developed by Alonzo Church in the 1930's — independently of Turing's work and, very possibly, slightly before it. While it was (arguably) not enthusiastically received when introduced, its popularity increased over time, and it provides a basis for the development and study of ***functional programming languages*** today.

This will not be studied in any detail in this course. The purpose of this document is to provide a very brief introduction to it. Students who are interested in this should consider the online tutorial of Rojas [2], the text of Robic [1] (which also includes a brief, readable, introduction to this — along with historical information about the period when the λ-calculus) — or contact Professor Robin Cockett, another faculty member in the Department of Computer Science, who frequently uses the λ-calculus in his teaching and research.

There are several versions, or "variants", of the λ-calculus. The version discussed here is called the "untyped" λ-calculus in the literature.

λ-Terms

Like other models of computation, the λ-calculus includes ***variables*** which represent "parameters" for function. Functions are represented by ***λ-terms***, which are inductively defined as follows.

(a) A variable is a λ-term; this (simplest) kind of λ-term is also called an ***atom***.

(b) If M is a λ-term and x is a variable, then

$$(\lambda x.M)$$

is a λ-term, which is said to be obtained from M by ***abstraction***.

(c) If M and N are λ-terms, then

$$(MN)$$

is a λ-term, which is called the ***application*** of M to N.

Free and Bound Variables

A variable, that occurs in a λ-term, is either ***free*** or ***bound*** — or both! This can be decided using the following rules.

- The variable x is ***free*** (and not ***bound***) in the atom (that is, special kind of λ-term) x. No other variables, besides x, are either free or bound in x.
- Let x be a variable and let $(\lambda y.M)$ be a λ term such that x occurs in $(\lambda y.M)$.
  - If x and y are the same variable, here, then x is ***bound*** in $(\lambda y.M)$.
  - On the other hand, if x and y are different variables, then x is *free* in $(\lambda y.M)$ if and only if x is free in M.
- Let x be a variable and let M and N be λ-terms
  - x is ***free*** in (MN) if and only if either x is free in M or x is free in N (or both).
  - x is ***bound*** in (MN) if and only if either x is bound in M or x is bound in N (or both).

To see that a variable really *can* be both free and bound in a λ-term at the same time, let x and y be variables and note that the following is a λ-term (it can be obtained using the inductive definition, given above):

$$((\lambda x.(xy))(\lambda y.y)) \tag{1}$$

The variable y is ***free*** in this λ-term, because it is free in $(\lambda x.(xy))$, but it is also ***bound*** in this λ-term, because it is bound in $(\lambda y.y)$.

Descriptions of the λ-calculus also refer to ***free occurrences*** and ***bound occurrences*** of variables. In the λ-term shown in line (1), we would say that the copy of y in the subexpression (xy) is a ***free occurrence*** of y, while the last copy of y, appearing in the subexpression $(\lambda y.y)$, is a ***bound occurrence*** of y.

β-Normal Form

Any λ-term of the form

$$((\lambda x.M)N)$$

for a variable x, and λ-terms M and N, is called a ***β-redex***. A λ-term may contain zero or more β-redexes as subexpressions.

A λ-term that does not have any β-redexes as subexpressions is said to be in ***β-normal form***.

Transformations of λ-Terms

λ-terms can be used to produce other λ-terms using transformations called ***β-reductions***. Two kinds of β-reductions are used.

- An ***α-conversion*** changes the name of a bound variable in a λ-term. For example, an α-conversion could be applied to the λ-term at line (1) to obtain the λ-term

  $$((\lambda x.(xy))(\lambda z.z))$$

  (changing the name of the second bound variable from y to z).

  The names of bound variables are similar to names for local variables in programs — if you change their names then this has, essentially, no effect on computations that are carried out. α-conversions should be used, before other transformations, to eliminate "name clashes" — circumstances where a variable is being introduced and used in more than one way (as y is, in the λ-term at line (1)).

- A ***β-reduction*** modifies a β-redex

  $$((\lambda x.M)N)$$

  by replacing occurrences of the bound variable x, in M, with the expression N.

  For example, a β-reduction could be applied to the above redex

  $$((\lambda x.(xy))(\lambda z.z))$$

  to obtain the λ-term

  $$((\lambda z.z)y)$$

  This is also a redex — and another β-reduction could be applied to obtain the λ-term

  $$y$$

What is Going On, Here?

While it might not be clear, a λ-term, representing a function or value, effectively provides a "recipe" for calculating it. Thus λ-terms represent ***algorithms*** (and the functions computed by them). A ***computation*** is represented by a sequence of transformations, of the types described above, to convert an initial λ-term into a "final" one.

The λ-Calculus and the Church-Turing Thesis

In the 1930's Church tried to persuade other mathematicians that the λ-calculus, then being proposed really could be used to model computation. *At the time*, his argument was not as widely accepted as he would have wished. Reasons for this might have included the following:

- The model seems to be quite abstract, with no apparent connection to the way that people solve problems. In contrast, Turing's original description [3], of what is now called a "Turing machine", described how the model was inspired by what people do when performing a calculation by hand, using a pencil and paper. While Turing simplified this — replacing a two-dimensional piece of paper with a one-dimensional storage tape — a connection to how people solve problems was still apparent.
- It was too early to make this kind of claim: We now have many more models of computations along with proofs of their equivalence, so that it is reasonably easy to accept that Church's model is a "universal" model of computation, once it has been explained that (for example) this is a consequence of the fact that the Turing machine is this kind of model of computation too. These other models — and proofs of equivalence — were not available at the time, so that the evidence in support of this "thesis" would have seemed much weaker than it is today.

References

[1] Borut Robič. *The Foundations of Computability Theory*. Springer-Verlag, second edition, 2020.

[2] Raul Rojas. A tutorial introduction to the Lambda calculus. Published online, on arXiv, at `https://arxiv.org/pdf/1503.09060`, 2015. Version 2.0.

[3] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 43:230–265, 1937.