

COMP3161/COMP9164 Supplementary Lecture Notes

Lambda Calculus

Thomas Sewell

October 2, 2025

This is a short note on the λ calculus. The λ calculus was created by Alonzo Church in the early part of the twentieth century. It was one of a number of attempts to develop an algebra in which to encode the other fields of mathematics. The goal was to build a system that was simple and unambiguous, and thus able to provide unambiguous meanings to everything that could be encoded in it. The λ calculus did not deliver on this grand ambition, and is now rarely used in mathematics, but it is considered an essential idea in the theory of computer science.

1 Syntax

The λ calculus syntax consists of symbols, applications, and lambda abstractions:

$$\begin{array}{l} t ::= \\ \quad x \quad \quad \quad (\textit{symbols}) \\ \quad | \quad t_1 t_2 \quad (\textit{applications}) \\ \quad | \quad \lambda x. t \quad (\textit{\lambda-abstractions}) \end{array}$$

2 Semantics

The semantics of the λ calculus is captured by an equivalence relation, which specifies which λ terms are equivalent. The equivalence relation is composed from three kinds of equivalence, named α , β and η . We assume that Alonzo Church had a preference for the Greek alphabet.

Informally, the three relations allow us to make three kinds of “moves”, α renaming, β contraction and η contraction, which look like:

$$\begin{array}{l} (\lambda x. x) \equiv_{\alpha} (\lambda y. y) \\ (\lambda x. fx) \equiv_{\eta} f \\ (\lambda x. e) y \mapsto_{\beta} e[x := y] \end{array}$$

The α and η rules capture ways in which λ calculus terms are “obviously” the same. We can change names that are local to a λ abstraction via an *alpha* renaming, and we can discard an unnecessary λ abstraction around a function via an η reduction (also called an η contraction).

The concept of α equivalence also applies to any other language with locally bound variable names. The note on “Syntax” already introduced an α equivalence notion for let expressions. The λ calculus is very influential, and it is common for other languages to borrow its names and its concepts, especially λ abstraction and α equivalence.

The key α , β and η steps can also be applied anywhere within a λ calculus term. For instance, the α equivalence relation can be presented in a natural deduction style:

$$\begin{array}{c}
\overline{x \equiv_{\alpha} x} \\
\frac{f_1 \equiv_{\alpha} f_2 \quad x_1 \equiv_{\alpha} x_2}{f_1 x_1 \equiv_{\alpha} f_2 x_2} \\
\frac{e_1 \equiv_{\alpha} e_2}{(\lambda x. e) \equiv_{\alpha} (\lambda x. e_2)} \\
\overline{(\lambda x. e) \equiv_{\alpha} (\lambda y. e[x := y])}
\end{array}$$

The rules here clarify that α equivalence is reflexive, and that it is contextual across applications and λ abstractions, and finally that a local α equivalence can be shown by substitution. The problem of performing a substitution of x without capturing a different x inside an inner $(\lambda x. e)$ is exactly the same as the one discussed in the “Syntax” note.

The η equivalence notion is the simplest. Like α equivalence, permits an η contraction to be applied anywhere in the term. Unlike the other two moves, it does not require any discussion of substitution or variable capture.

2.1 β Reduction

The most complex of the equivalence notions is built from β reduction. The β reduction process allows us to “evaluate” terms in the λ calculus by performing substitution.

$$(\lambda x. e) y \mapsto_{\beta} e[x := y]$$

Like the other two operations, a β -reduction can be applied anywhere in the syntax we find a λ abstraction applied to an argument. Such a site is called a reducible (sub-)expression, or redex. This is characterised formally in natural-deduction style by these rules:

$$\frac{(\lambda x. t) u \mapsto_{\beta} t[x := u]}{\frac{t \mapsto_{\beta} t' \quad s \mapsto_{\beta} s' \quad t \mapsto_{\beta} t'}{s t \mapsto_{\beta} s' t'} \quad \frac{s \mapsto_{\beta} s' \quad t \mapsto_{\beta} t'}{s t \mapsto_{\beta} s' t'} \quad \frac{t \mapsto_{\beta} t'}{\lambda x. t \mapsto_{\beta} \lambda x. t'}}$$

The β -reduction process is essentially a small-step relation. Small step relations are (or will be) discussed in detail in the “Semantics” note. A term is in normal form once it has no more reducible expressions. The β -reduction process may terminate, after a chain of reductions, in a term in normal form. Two terms t_1 and t_2 are defined to be β -equivalent if they can both be reduced to the same normal form, that is:

$$\frac{t_1 \mapsto_{\beta}^* n \quad t_2 \mapsto_{\beta}^* n \quad \text{redexes}(n) = \emptyset}{t_1 \equiv_{\beta} t_2}$$

Note that a term t may have multiple redexes (reducible subexpressions). There is no enforced evaluation order. Instead, any reduction may be performed, and thus the β -reduction process of a term may split into different paths. It can be proven that β -reduction for a term is *confluent* if any normal form exists. If it is possible to reach a normal form from t , then every β -reduction process from t eventually reaches the same normal form and terminates there. Since normal forms are unique, the β -equivalence relation is well-behaved.

The three kinds of moves can all be put together. Terms are $\alpha\beta\eta$ -equivalent if they have normal forms that are $\alpha\eta$ -equivalent, that is:

$$s \equiv_{\alpha\beta\eta} t \equiv \exists n_s n_t. s \mapsto_{\beta}^* n_s \wedge t \mapsto_{\beta}^* n_t \wedge n_s \equiv_{\alpha\eta} n_t$$

However, not all terms have a normal form. Consider this problematic expression:

$$(\lambda x. x x)(\lambda x. x x)$$

A β -reduction step returns to the same term (or an α -equivalent one, depending on how capture-avoiding substitution is done). The β -reduction process goes on forever with no progress. There is an infinite variety of terms that do not terminate, and the $\alpha\beta\eta$ -equivalence notion is not useful for them.

3 Church's Encodings

A future version of this document might say more about how Church encoded standard types and operations from mathematics into the λ calculus, including the encodings of Boolean terms and operations, and the encoding of natural numbers and their arithmetic.