Contents

Models of Computation

First Lesson

Contact Information

Prof email: piperno@di.uniroma1.it

Actual lecture times:

- Wednsday 13:30 - 15:00
- Thursday 16:00 - 17:30, or 16:15 - 17:55

Course Contents

The course content will focus on **functional programming** and λ-calculus.

The main application of these languages is that of *function application*, these languages are devoid of *assignment* semantics, and are therefore called *pure*.

The main ingredients of our programs will be:

- Variables, which are usually denoted by lowercase letters
- Starting from these, you obtain the set of all programs, which are called λ-terms, this set is denoted as Λ.

Set Definition

We can provide an inductive definition of a set of λ-terms, Λ:

$$\frac{x \in V}{x \in \Lambda} \quad \text{(var)}$$

$$\frac{M \in \Lambda \quad N \in \Lambda}{(MN) \in \Lambda} \quad \text{(app)}$$

$$\frac{M \in \Lambda \quad x \in V}{\lambda x.M \in \Lambda} \quad \text{(abs)}$$

Applying these inductive rules (variables, application, abstraction).

Bactus Normal Form

Another way of describing a set of lambda terms is by using Bactus Normal Form (BNF).

$$\Lambda :: Var|\Lambda\Lambda|\lambda Var.\Lambda$$

Which is essentially a grammar for the set of lambda terms.

Lambda calculus is left-associative:

$$((xy)z) = xyz \neq x(yz)$$

All functions are unary (we assume currying). For example, the function $f(x, y)$ is represented as:

$$\lambda x.\lambda y.fxy$$

.

Functions in lambda calculus can be applied to other functions or themselves, they can also return functions.

Beta Reduction

The main operation in lambda calculus is *beta reduction*, which is the application of a function to an argument.

$$\underbrace{(\lambda x.M)N}_{redex} \rightarrow_\beta M[N/x]$$

Where $M[N/x]$ is the result of substituting all occurrences of x in M with N.

Some other examples:

$$(\lambda x.x)y \rightarrow_\beta y$$

$$(\lambda x.xx)y \rightarrow_\beta yy$$

$$(\lambda xy.yx)(\lambda u.u) \rightarrow_\beta \lambda y.y(\lambda u.u)$$

$$(\lambda xy.yx)(\lambda t.y) \rightarrow_\beta \lambda y.y(\lambda t.y)$$

This rule can be applied in any context in which it appears.

Types of Variables

We distinguish two kinds of variables:

- Free variables: variables that are not bound by an abstraction
- Bound variables: variables that are bound by an abstraction

For example, in the term $\lambda x.xy$, y is a free variable, while x is a bound variable.

Bound variables can be renamed, whereas for free variables the naming is relevant.

$$\begin{cases} FV(x) = \{x\} \\ FV(MN) = FV(M) \cup FV(N) \\ FV(\lambda x.M) = FV(M) - \{x\} \end{cases}$$

A set of lambda terms where $LM(\Lambda) =$ is called *closed*.

Extra Rules

$$(\mu) \quad \frac{M \to_\beta M'}{NM \to_\beta NM'}$$

$$(\nu) \quad \frac{M \to_\beta M'}{MN \to_\beta M'N}$$

$$(\xi) \quad \frac{M \to M'}{\lambda x.M \to \lambda x.M'}$$

These rules allow us to select redexes in a context-free manner in the middle of our lambda term. We can then choose the order of evaluation of our redexes, while still taking care of the left-associative order of precedence. Our calculus is therefore not *determinate* but is still *deterministic*, meaning that there may be multiple reduction strategies but they all lead to the same result.

This corollary is called the *Church Rosser Theorem*, discovered in 1936.

In general, a call-by-value-like semantic is preferrable when choosing evaluation paths, as it clears the most amount of terms as early as possible.

- Call By Value is *efficient*
- Call By Name is *complete*, **if** the lambda term is normalizable