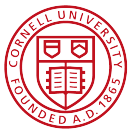
CS 4110 – Programming Languages and Logics
Lecture #15: Encodings



1 λ-calculus encodings

The pure λ-calculus contains only functions as values. It is not exactly easy to write large or interesting programs in the pure λ-calculus. We can however encode objects, such as booleans, and integers.

1.1 Booleans

Let us start by encoding constants and operators for booleans. That is, we want to define functions TRUE, FALSE, AND, NOT, IF, and other operators that behave as expected. For example:

$$\begin{aligned} \mathsf{AND\ TRUE\ FALSE} &= \mathsf{FALSE} \\ \mathsf{NOT\ FALSE} &= \mathsf{TRUE} \\ \mathsf{IF\ TRUE}\ e_1\ e_2 &= e_1 \\ \mathsf{IF\ FALSE}\ e_1\ e_2 &= e_2 \end{aligned}$$

Let's start by defining TRUE and FALSE:

$$\begin{aligned} \mathsf{TRUE} &\triangleq \lambda x.\, \lambda y.\, x \\ \mathsf{FALSE} &\triangleq \lambda x.\, \lambda y.\, y \end{aligned}$$

Thus, both TRUE and FALSE are functions that take two arguments; TRUE returns the first, and FALSE returns the second. We want the function IF to behave like

$$\lambda b.\, \lambda t.\, \lambda f.\, \text{if } b = \mathsf{TRUE} \text{ then } t \text{ else } f.$$

The definitions for TRUE and FALSE make this very easy.

$$\mathsf{IF} \triangleq \lambda b.\, \lambda t.\, \lambda f.\, b\ t\ f$$

Definitions of other operators are also straightforward.

$$\begin{aligned} \mathsf{NOT} &\triangleq \lambda b.\, b\ \mathsf{FALSE}\ \mathsf{TRUE} \\ \mathsf{AND} &\triangleq \lambda b_1.\, \lambda b_2.\, b_1\ b_2\ \mathsf{FALSE} \\ \mathsf{OR} &\triangleq \lambda b_1.\, \lambda b_2.\, b_1\ \mathsf{TRUE}\ b_2 \end{aligned}$$

1.2 Church numerals

Church numerals encode a number n as a function that takes f and x, and applies f to x n times.

$$\overline{0} \triangleq \lambda f.\, \lambda x.\, x$$
$$\overline{1} = \lambda f.\, \lambda x.\, f\ x$$
$$\overline{2} = \lambda f.\, \lambda x.\, f\ (f\ x)$$
$$\mathsf{SUCC} \triangleq \lambda n.\, \lambda f.\, \lambda x.\, f\ (n\ f\ x)$$

In the definition for SUCC, the expression $n\ f\ x$ applies f to x n times (assuming that variable n is the Church encoding of the natural number n). We then apply f to the result, meaning that we apply f to x $n + 1$ times.

Given the definition of SUCC, we can easily define addition. Intuitively, the natural number $n_1 + n_2$ is the result of apply the successor function n_1 times to n_2.

$$\mathsf{PLUS} \triangleq \lambda n_1.\, \lambda n_2.\, n_1\ \mathsf{SUCC}\ n_2$$