CS 4110

Programming Languages & Logics

Lecture 15
Encodings

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.

Booleans

We need to define functions TRUE, FALSE, AND, NOT, IF, and other operators that behave as follows:

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

Booleans

We need to define functions TRUE, FALSE, AND, NOT, IF, and other operators that behave as follows:

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

Let's start by defining TRUE and FALSE:

$$\begin{aligned} \text{TRUE} &\triangleq \\ \text{FALSE} &\triangleq \end{aligned}$$

Booleans

We need to define functions TRUE, FALSE, AND, NOT, IF, and other operators that behave as follows:

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

Let's start by defining TRUE and FALSE:

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

Booleans

We want the function IF to behave like

$$\lambda b.\ \lambda t.\ \lambda f.\ \text{if } b \text{ is our term TRUE then } t\text{, otherwise } f$$

Booleans

We want the function IF to behave like

$$\lambda b.\ \lambda t.\ \lambda f.\ \text{if } b \text{ is our term TRUE then } t\text{, otherwise } f$$

We can rely on the way we defined TRUE and FALSE:

$$\text{IF} \triangleq \lambda b.\ \lambda t.\ \lambda f.\ b\ t\ f$$

Booleans

We want the function IF to behave like

$$\lambda b.\ \lambda t.\ \lambda f.\ \text{if } b \text{ is our term TRUE then } t\text{, otherwise } f$$

We can rely on the way we defined TRUE and FALSE:

$$\text{IF} \triangleq \lambda b.\ \lambda t.\ \lambda f.\ b\ t\ f$$

We can also write the standard Boolean operators.

$$\text{NOT} \triangleq$$

$$\text{AND} \triangleq$$

$$\text{OR} \triangleq$$

Booleans

We want the function IF to behave like

$$\lambda b.\ \lambda t.\ \lambda f.\ \text{if } b \text{ is our term TRUE then } t\text{, otherwise } f$$

We can rely on the way we defined TRUE and FALSE:

$$\text{IF} \triangleq \lambda b.\ \lambda t.\ \lambda f.\ b\ t\ f$$

We can also write the standard Boolean operators.

$$\text{NOT} \triangleq \lambda b.\ b\ \text{FALSE TRUE}$$

$$\text{AND} \triangleq \lambda b_1.\ \lambda b_2.\ b_1\ b_2\ \text{FALSE}$$

$$\text{OR} \triangleq \lambda b_1.\ \lambda b_2.\ b_1\ \text{TRUE}\ b_2$$

Church Numerals

Let’s encode the natural numbers!

We’ll write $\overline{n}$ for the encoding of the number n. The central function we’ll need is a *successor* operation:

$$\text{SUCC}\ \overline{n} = \overline{n+1}$$

Church Numerals

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

$$\begin{aligned} \overline{0} &\triangleq \lambda f.\, \lambda x.\, x \\ \overline{1} &\triangleq \lambda f.\, \lambda x.\, f\, x \\ \overline{2} &\triangleq \lambda f.\, \lambda x.\, f\,(f\, x) \end{aligned}$$

Church Numerals

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

$$\begin{aligned}\overline{0} &\triangleq \lambda f.\, \lambda x.\, x \\ \overline{1} &\triangleq \lambda f.\, \lambda x.\, f\, x \\ \overline{2} &\triangleq \lambda f.\, \lambda x.\, f\,(f\, x)\end{aligned}$$

We can write a successor function that “inserts” another application of *f*:

$$\text{SUCC} \triangleq \lambda n.\, \lambda f.\, \lambda x.\, f\,(n\, f\, x)$$

Addition

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

$$\text{PLUS} \triangleq$$

Addition

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

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

Church Numerals

We can define more functions on integers:

$$
\begin{aligned}
\text{SUCC} &\triangleq \lambda n.\, \lambda f.\, \lambda x.\, f\,(n\, f\, x) \\
\text{PLUS} &\triangleq \lambda n_1.\, \lambda n_2.\, n_1\ \text{SUCC}\ n_2
\end{aligned}
$$

Church Numerals

We can define more functions on integers:

$$\begin{aligned} \text{SUCC} &\triangleq \lambda n.\, \lambda f.\, \lambda x.\, f\,(n\,f\,x) \\ \text{PLUS} &\triangleq \lambda n_1.\, \lambda n_2.\, n_1\ \text{SUCC}\ n_2 \end{aligned}$$

Church Numerals

We can define more functions on integers:

$$\begin{aligned}
\text{SUCC} &\triangleq \lambda n.\, \lambda f.\, \lambda x.\, f\,(n\,f\,x) \\
\text{PLUS} &\triangleq \lambda n_1.\, \lambda n_2.\, n_1\ \text{SUCC}\ n_2 \\
\text{TIMES} &\triangleq \lambda n_1.\, \lambda n_2.\, n_1\ (\text{PLUS}\ \mathrm{n}_2)\ \overline{0}
\end{aligned}$$

Church Numerals

We can define more functions on integers:

$$
\begin{array}{rcl}
\text{SUCC} & \triangleq & \lambda n.\, \lambda f.\, \lambda x.\, f\,(n\, f\, x) \\
\text{PLUS} & \triangleq & \lambda n_1.\, \lambda n_2.\, n_1\, \text{SUCC}\, n_2 \\
\text{TIMES} & \triangleq & \lambda n_1.\, \lambda n_2.\, n_1\, (\text{PLUS n}_2)\, \overline{0} \\
\text{ISZERO} & \triangleq & \lambda n.\, n\, (\lambda z.\, \text{FALSE})\, \text{TRUE}
\end{array}
$$