3. The Lambda Calculus

Church defined the Lambda Calculus in 1941 in order to model "computability".

At a similar time, Turing defined Turing Machines to model "computability".

The set of computable functions is the same, no matter whether one uses Lambda Calculus, Turing Machines, Java, C, Python, ...

⇒ This is the set of computable functions (Church's Thesis).

In contrast to Turing Machines, the Lambda Calculus was used as the basis of functional programming languages.

FP languages are just Lambda Calculus + more constructs and syntax to improve readability.

But the Lambda Calculus is already Turing-complete.

To implement Haskell, one can compile it to the IR and then one only needs an implementation of...
10 implement LC, and then one only needs an implementation of the LC in order to implement Haskell.

3.1 Syntax of the Lambda Calculus

3.2 Semantics of the Lambda Calculus

Operational Semantics: Define a "reference interpreter" that determines how expressions are evaluated.

3.3 Translating Simple Haskell to the Lambda Calculus

3.4 Pure Lambda Calculus

3.1 Syntax of the Lambda Calculus

"Programs" of the LC are so-called Lambda Terms.

Def. 3.1.1 (Lambda Terms)

Let $C$ be a set of constants and let $V$ be an (infinite) set of variables. Then the set of lambda terms $\Lambda$ is the smallest set with:

- $C \subseteq \Lambda$
- $V \subseteq \Lambda$
- $(t_1, t_2) \in \Lambda$, if $t_1, t_2 \in \Lambda$ (application)
- $(\lambda x. t) \in \Lambda$, if $x \in V$ and $t \in \Lambda$ (lambda abstraction)

in Haskell: $\lambda x \rightarrow t$
If $Succ$, $Zero$, $s$, $t$, $is\_Succ$, $arg\_Succ$, $\ldots \in \mathcal{C}$
$x, y, t, \ldots \in \mathcal{V}$
then $\lambda x. \ Succ\ (Succ\ x) \in \Lambda$

Notation:
- Applications associate to the left:
  $(t_1, t_2, t_3)$ stands for $((t_1, t_2), t_3)$
- Scope of $\lambda$ goes as far to the right as possible:
  $\lambda x. x\ x$ stands for $\lambda x. (x\ x)$
  (not for $(\lambda x. x)\ x$)
- We write $\lambda x y. t$ instead of $\lambda x. \lambda y. t$

Def 3.1.2 (Free Variables of a Lambda Term)
For every $t \in \Lambda$, we define $\text{free}(t) \subseteq \mathcal{V}$:
- $\text{free}(c) = \emptyset$ for all $c \in \mathcal{C}$
- $\text{free}(x) = \{x\}$ for all $x \in \mathcal{V}$
- $\text{free}(t_1, t_2) = \text{free}(t_1) \cup \text{free}(t_2)$ for all $t_1, t_2 \in \Lambda$
- $\text{free}(\lambda x. t) = \text{free}(t) \setminus \{x\}$ for all $x \in \mathcal{V}$, $t \in \Lambda$

A $t \in \Lambda$ is closed if $\text{free}(t) = \emptyset$. 
\( \text{Ex: } \text{free}(\lambda y. y x) = \{x\} \)
\( \text{free}(\lambda x. x)(x y) = \{x, y\} \)

We now need the concept of substitutions.
For two lambda terms \( r, t \in \Lambda \) and \( x \in \mathbb{V} \),
\( r[x/t] \) should denote the term that results from \( r \) by instantiating
all free occurrences of \( x \) by \( t \).

\[
\lambda y. y x \ [x/\lambda u. u v] = \lambda y. y (\lambda u. u v)
\]

\( \lambda y. y x \) : returns the value of a function at argument \( x \)
\( \lambda y. y x \) : returns the value of a function at argument \( x \)

Names of bound variables are irrelevant.
Therefore, applying the same substitution to these
two terms should yield the same result.

\[
\lambda y. y x \ [x/\lambda u. u v] = \lambda y. y (\lambda u. u v)
\]
\[
\lambda v. v x \ [x/\lambda u. u v] = \lambda v. v (\lambda u. u v)
\]

Substitutions should take care of this.
If a substitution introduces a free variable (like \( v \)),

If a substitution introduces a free variable (like $v$), then the bound variables of the same name should first be renamed.

**Def 3.13 (Substitutions on Lambda Terms)**

For $x, t \in A$ and $x \in \mathcal{B}$, we define $x [x / t]$ as follows:

- $x [x / t] = t$
- $y [x / t] = y$ for all $y \in \mathcal{B}$ with $y \neq x$
- $c [x / t] = c$ for all $c \in \mathcal{C}$
- $(r_1, r_2) [x / t] = (r_1 [x / t], r_2 [x / t])$ for all $r_1, r_2 \in A$
- $(\lambda x. y) [x / t] = \lambda x. y$
- $(\lambda y. r) [x / t] = \lambda y. r [x / t]$ if $y \neq x$ and $y \notin \text{free} (t)$
- $(\lambda y. r) [x / t] = \lambda y'. (r [y'/y] [x / t])$

Ex: $(\lambda v. v \times) [\times / \lambda u. \mu \nu v] = \lambda v'. v' (\lambda u. \mu \nu)$

First rename $y$ to a fresh var. $y'$

Then apply the subst.

$[x / t]$