3. The Lambda Calculus

Lambda Calculus was developed by A. Church in 1941.
Goal was to describe which functions are "computable".

A. Turing: characterize computable functions with Turing machines

A. Church: characterize computable functions with Lambda calculus

The set of computable fcts with Turing machines =
— " ——————— Lambda calc. =
— " ——————— any usual prog. language

⇒ Church's Thesis: All prog. languages lead to the same set of "computable" functions.

In contrast to Turing machines, the Lambda Calculus can be used as the basis of prog. languages:
All functional prog. languages build upon the Lambda Calculus.
For implementation of fct. languages, one can automatically transform any program into Lambda Calculus.
⇒ One only needs an implementation for Lambda Calculus.

The Lambda Calculus can also be used to define an <u>operational semantics</u> of the prog. language:

semantics is defined by the reference implementation of our interpreter.

Complex Haskell $\xrightarrow{\text{Ch 2.}}$ Simple Haskell $\xrightarrow{\text{Ch 3}}$ Lambda Calculus $\dashrightarrow$ (Ch 3.4) Pure Lambda Calc.

3.1. Syntax of Lambda Calculus
3.2 (Operational) Semantics of Lambda Calculus
3.3 Implementing Haskell by the Lambda Calculus
3.4 Pure Lambda Calculus

3.1 Syntax of the Lambda Calculus

<u>Def 3.1.1</u> (Lambda Terms) (Slide 53)

Let $\mathcal{C}$ be a set of constants and $\mathcal{V}$ be an (infinite) set of variables. The set of lambda terms Λ is the smallest set such that

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

in Haskell: \ x -> t

Examples: x y Succ $+$ $\mathit{isa}_{\mathit{Succ}}$

$\in \mathcal{V}$ $\in \mathcal{C}$

$\mathit{Succ}\ (\mathit{Succ}\ x)$ $\lambda x.(\mathit{Succ}\ x)$

$(\mathit{Succ}\ \mathit{Succ})$ $\lambda x.x\,x$

Conventions for Notation:

- applications associate to the left:

  $(t_1\ t_2\ t_3)$ stands for $((t_1\ t_2)\ t_3)$

- the scope of a λ is as far to the right as possible:

  $\lambda x.\ x\ x$ stands for $\lambda x.(x\ x)$

- we write $\lambda x\,y.\ t$ for $\lambda x.\lambda y.\ t$

Similar to Haskell-expressions, we also define the free variables of a λ-term:

Def 3.1.2 (Free Variables of Lambda-Terms)

For every $t \in \Lambda$ we define $\mathit{free}(t) \subseteq \mathcal{V}$ as follows:

- $\mathit{free}(c) = \emptyset$ for all $c \in \mathcal{C}$
- $\mathit{free}(x) = \{x\}$ for all $x \in \mathcal{V}$
- $\mathit{free}(t_1\ t_2) = \mathit{free}(t_1) \cup \mathit{free}(t_2)$ for all $t_1, t_2 \in \Lambda$
- $\mathit{free}(\lambda x.\ t) = \mathit{free}(t) \setminus \{x\}$ for all $t \in \Lambda$, $x \in \mathcal{V}$

A Lambda term t is closed iff $\mathit{free}(t) = \emptyset$.

Ex: $\text{free}(\lambda x.\, x y) = \{y\}$

$\text{free}((\lambda x.x)(x\ y)) = \{x, y\}$

We now define <u>substitutions</u>:

$r, t \in \Lambda, \quad x \in \mathcal{V}$

$r[x/t]$ should be the term r, where all free occurrences of x are replaced by t.

$$(\lambda y.\, y\underline{x})[x/\lambda u.uv] =$$

$$\lambda y.\, y(\lambda u.\, uv)$$

$(\lambda y.\, yx)$ means: take a function and apply it to arg. x

$(\lambda v.\, v\ x)$ means: ———— " ————

$\Rightarrow$ The names of bound variables do not matter for the meaning of Lambda terms.

$\Rightarrow$ The names of bound variables shouldn't matter either when applying substitutions.

$$\left.\begin{aligned} (\lambda y.\, y\underline{x})[x/\lambda u.uv] &= \lambda y.\, y(\lambda u.uv) \\ (\lambda v.\, v x)[x/\lambda u.uv] &= \lambda v.\, v(\lambda u.uv) \end{aligned}\right\} ↯$$

These terms would have a different meaning! In the first term, v is free. The second term has no free variables.

Solution: if r has a bound variable that occurs in free(t), then we first rename this bound variable in r before applying $r\,[x/t]$.

Def 3.1.3 (Substitutions) (Slide 53)

For $r, t \in \Lambda$ and $x \in \mathcal{V}$, we define:

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

$$(\lambda v. v\, x)[x/\lambda u. u\, v] = \lambda v'. v'\,(\lambda u. u\, v)$$