

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:

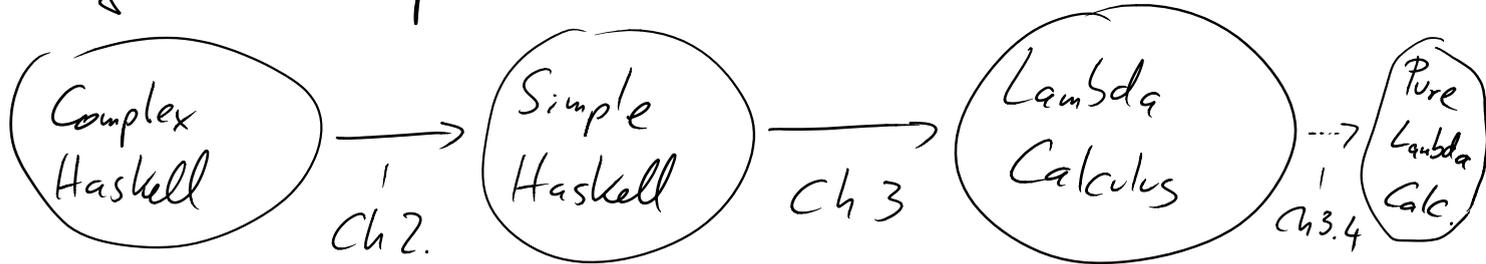
All functional prog. languages build upon the Lambda
Calculus.

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 operational semantics of the prog. language:

Semantics is defined by the reference implementation of our interpreter.



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

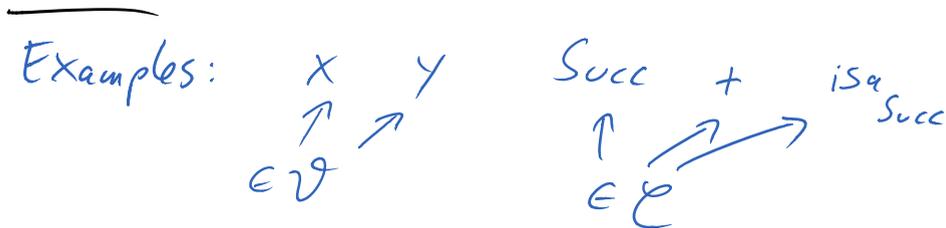
3.1 Syntax of the Lambda Calculus

Def 3.1.1 (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: $\lambda x \rightarrow t$



$Succ (Succ x)$

$\lambda x. (Succ x)$

$(Succ Succ) \lambda x. xx$

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 $t \in \mathcal{T}$ we define $free(t) \subseteq \mathcal{V}$ as follows:

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

A lambda term t is closed iff $free(t) = \emptyset$.

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

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

We now define substitutions:

$$r, t \in \Delta, x \in \mathcal{V}$$

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

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

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

$(\lambda y. y x)$ 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.

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

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



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 $\text{free}(t)$, 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 $c \in \mathcal{C}$
- $(r_1 r_2)[x/t] = (r_1[x/t] r_2[x/t])$ 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 $y \neq x$, $y \in \text{free}(t)$
 $y' \notin \text{free}(r) \cup \text{free}(t)$

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