**Lambda calculus**

**Leibniz** had as ideal the following:

1. Create a ‘universal language’ in which all possible problems can be stated

-Set theory + predicate calculus

1. Find a decision methos to solve all the problems stated in the universal language

-philosophical problem – Entscheidungsproblem

**Entscheidungsproblem** – negative outcome

-Alonzo Church – proposes LC as extension of logic

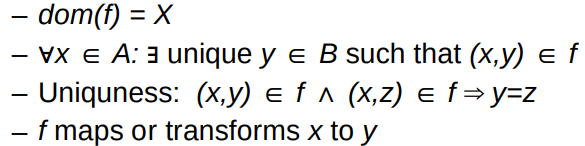
-shows the existence of undecidable problem

-functional programming languages

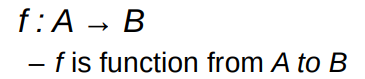
-Alan Turing – proposes TM

-proved that both models define the same class of computable functions

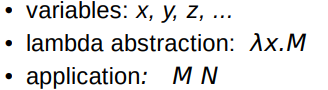
-Imperative programming languages

**Functions** – basic concept of classical and modern mathematics

A, B – sets and f-relation



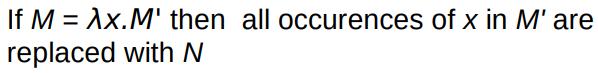
**Lambda notation**

 -Lambda expression

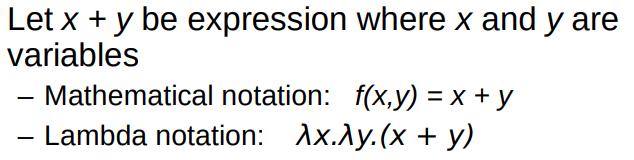
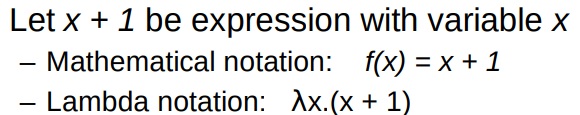
 -Lambda abstraction represents function

- x is function argument

-Mis function expression (receipt that specifies how function is >>computed<<)

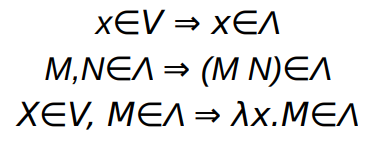
 -Application M N

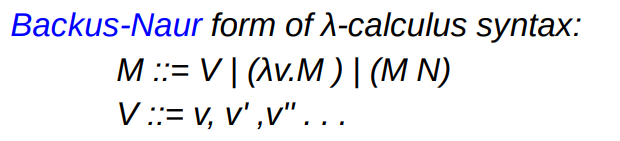
Mechanical definition of parameter passing.



-Obvious difference: λ-notation does not name function

**LC syntax**

 -Definition: The set of λ-expressions Λ is constructed form infinite set of variables {v, v', v'', v''', ...} by using application and λ-abstraction:

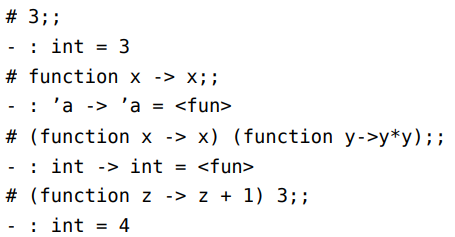


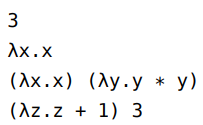
**Syntax rules**

-Application is left-associative

 -Λ-abstraction is right-associative

 -We often use the following abbreviation





**Free and bound variables**

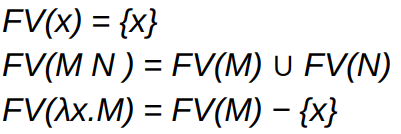
-Abstraction λx.M binds variable x in expression M

-In similar manner the function arguments are bound to the function body.

-M is scope of variable x in expression λx.M

-Variable x is free in some expression M if there exists no λ-abstraction that binds it

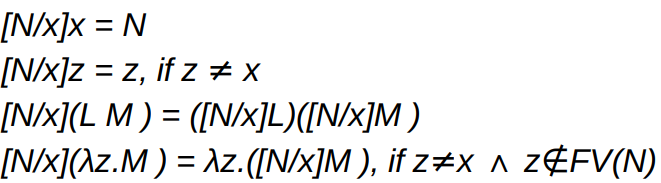
-Name of free variable is important while the name of bound variable is not

**Computing free variables** – the set of free variables of λ-expression M, denoted FV(M), is defined with the following rules: Example:

-λ-expression M is closed if FV(M) = {}.

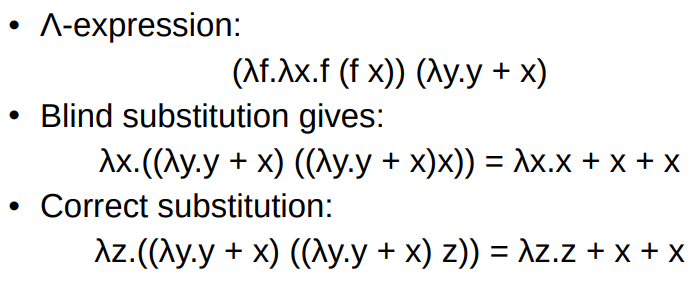
**Substitution** – basic of LC evaluation

 -Substitute all instances of a variable x in λ-expression M with N: [N/x]M

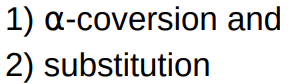


**Alpha conversion** – renaming λ-expression yields equivalent λ-expression

 -Alpha conversion rule:



**Evaluation**

 -Λ-calculus is very expressive language equivalent to Turing machine

-Evaluation of λ-expressions is based on:

-Evaluation is often called reduction

-Λ-expressions are reduced to value – normal forms of λ-expressions i.e. λ-expressions that can not be further reduced

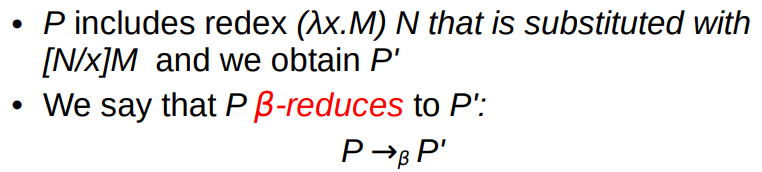
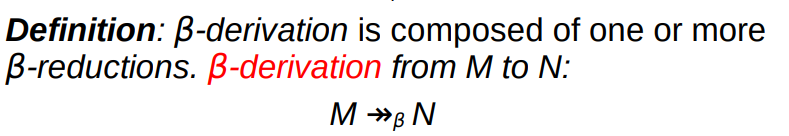
**Β-reduction** – only rule used for evaluation of pure λ-calculus (aside from renaming)

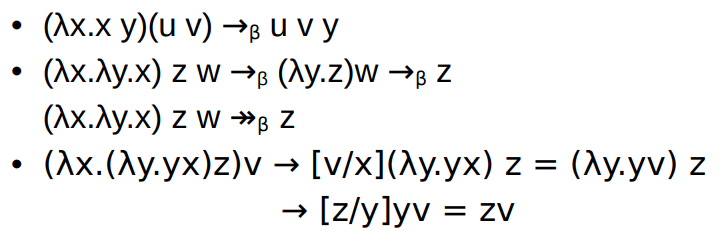
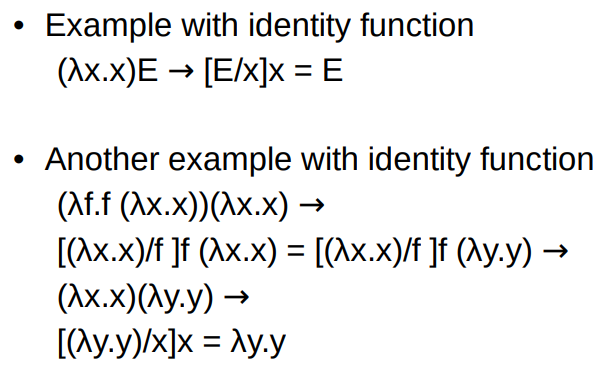
-Expression (λx.M) N stands for operator (λx.M) applied to parameter N

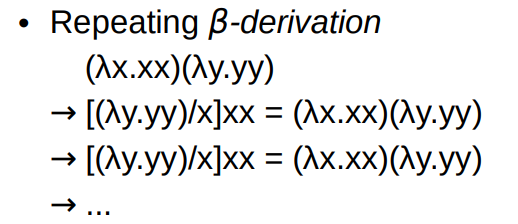
-Intuitive interpretation of (λx.M) N is substitution of x in M for N

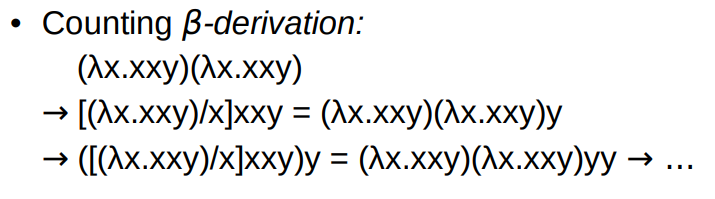
Definition: Let λx.M be λ-expression. Application of (λx.M) on parameter N is implemented with β-reduction:

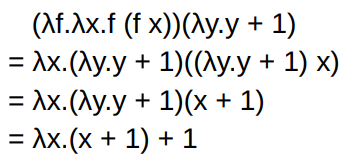
-Expression (λx.M) N is called redex (reducable expression), expression [N/x]M is called contractum.



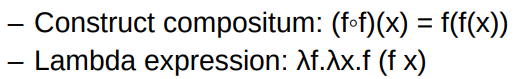
Examples of the evaluation



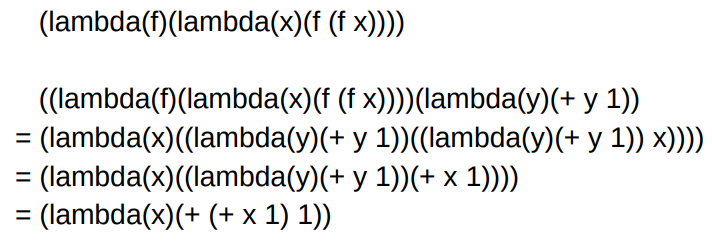
****

**Higher-order functions** – function that can either:

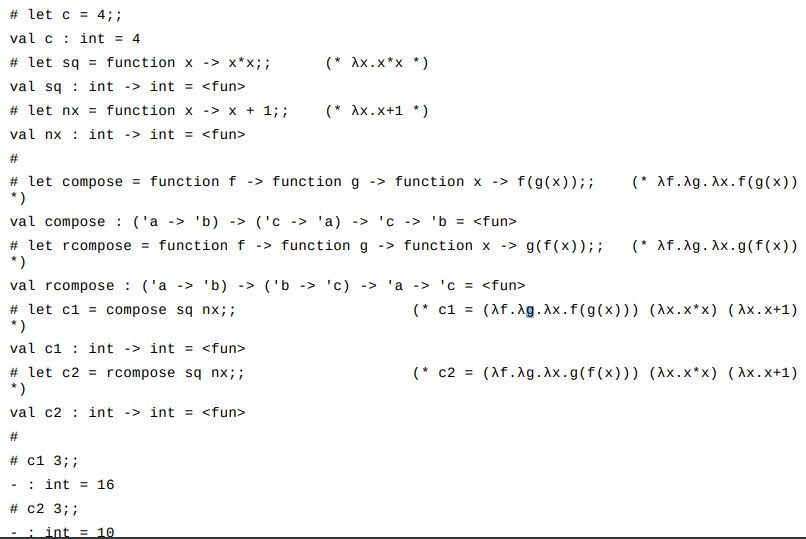
-take another function as an argument, or,

 -return function as the result of function application

-The same function in Lisp



Examples in Ocaml



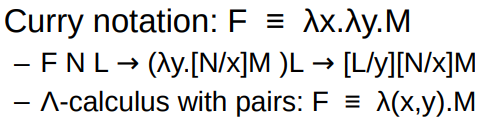
**Programming in LC**

**Curry functions** – can have single parameter in λ-calculus

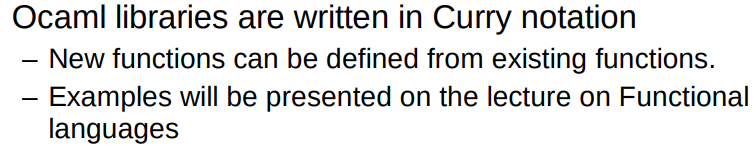
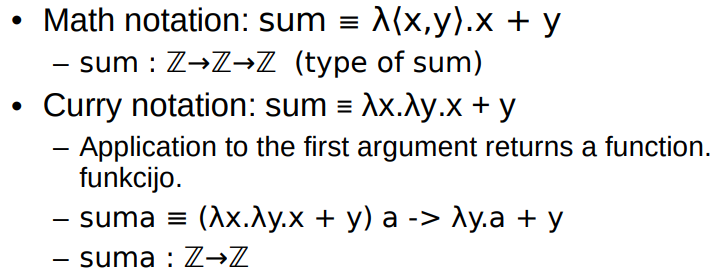
-multiple parameters can be implemented by using higher-order functions

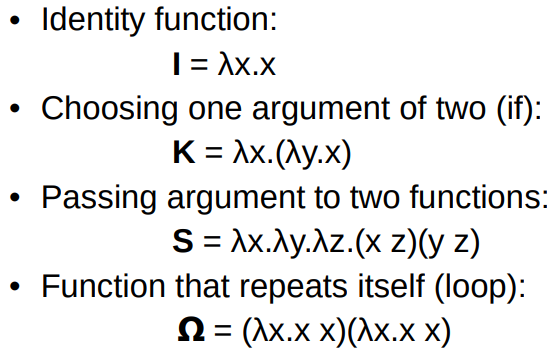
-F is a function with parameters (N, L) and body M

-M be expression with free variables x and y

 -We wish to replace x with N and y with L

-Transforming from λ(x, y).M to λx. λy.M is called Currying

Example:

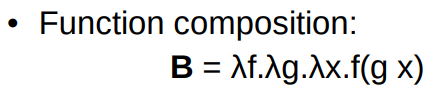


**Combinators** – primitive functions

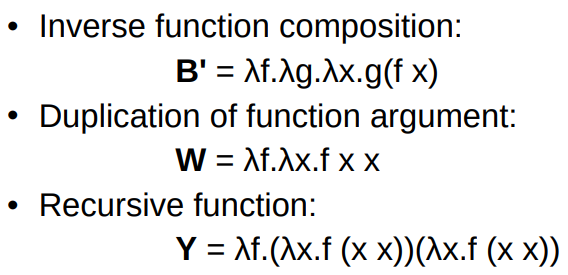
-expressing basic operations of computation

-functions – identity, composition, choice, etc.

-Combinatory logic CL (Curry, Feys)

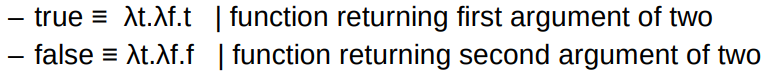
 -Combinators – building blocks of CL

-Cl uses combinators **I**, **K**, and **S**

 -Combinators are often used in programming languages

-Higher-order functions: apply, map, fold, filter, etc.

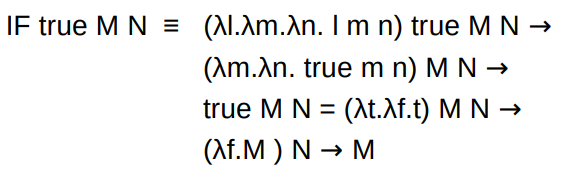
**Logical values**

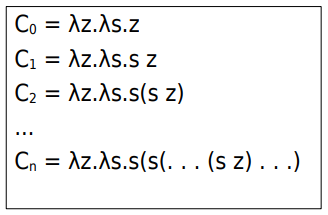


-IF statement is simple application of truth value

-λl.λm.λn. l m n

-Truth value determines first or second choice

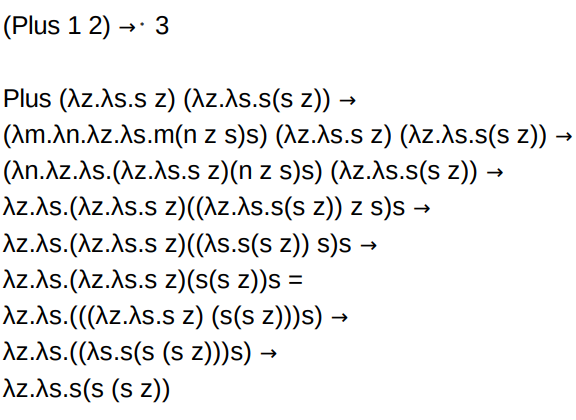
 -Evaluation of IF statement

**Church numbers**

-Number n is represented with Cn

-n = 0+1+ … +1 | n times successor of 0

-z stands for zero and s represents successor function

 -Arithmetic operations

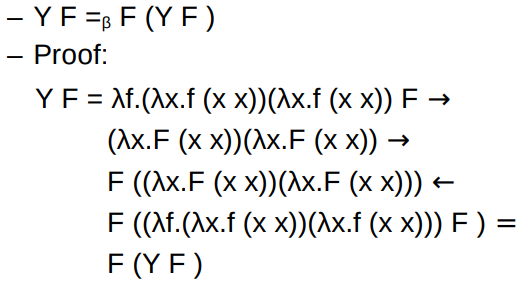
-Plus = λm.λn.λz.λs.m (n z s) s

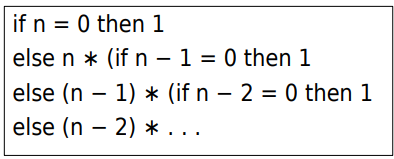
-Times = λm.λn.m C0 (Plus n)

**Recursion**

-Can be expressed using combinator **Y**

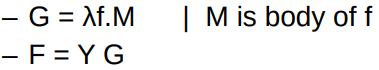
-Y = λf.(λx.f (x x))(λx.f (x x))

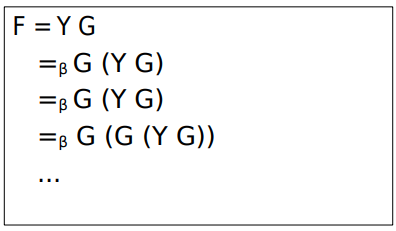
**** -Important property of **Y**

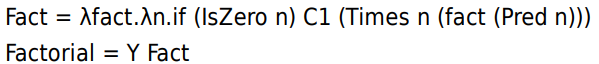
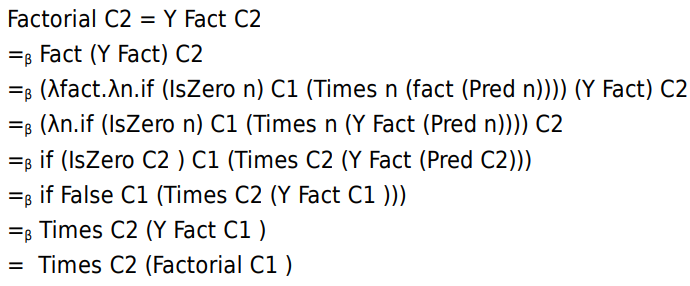


-Operation factorial: n!

-Intuitive definition

 -Definition of recursive function F

 -Definition of F

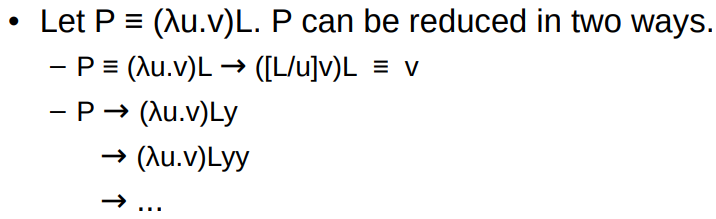
**Factorial**

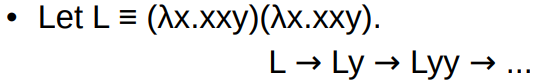
**Β-normal form**

-Definition: 1) λ-expression Q that does not include β-redexes is in β-normal form

2) The class of all β-normal forms is called β-nf.

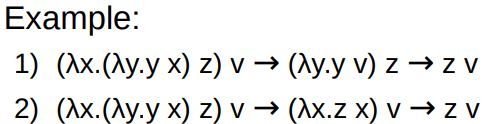
3) If P β-reduces to Q, which is β-nf, then Q is βnormal form of P

Is every λ-expression normalizable? – **No**



-P has β-nf but also infinite derivation!

**On evaluation order**

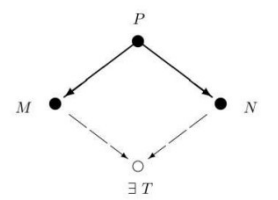
 -Some λ-expressions can be reduced in more than one way.

-Evaluation strategies:

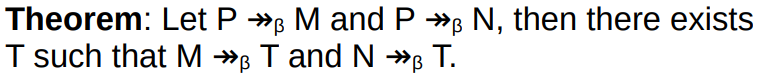
-Normal form strategie (left-outer redex first)

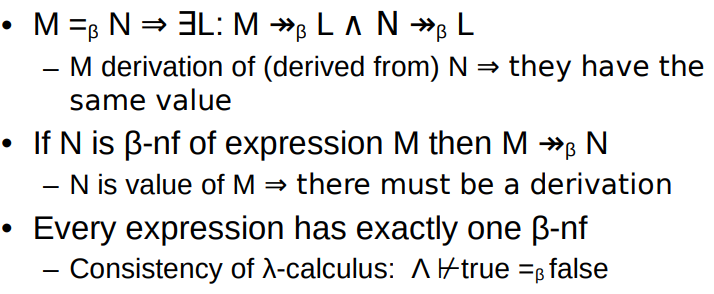
-Call by name (no reduction in λ-abstractions + nf)

-Call by value (outer redex but after right-hand side reduced)



**Church-Rosser theorem** – a central theorem in lambda calculus





Consequences of CR

**Properties of LC**

-LC is consistent

-LC is equivalent to TM (Turing machine)

-LC is r.e. language

-LC is partially computable (not total !)

-LC with types is total function

-Very limited class of languages

-The characterization of total TM is not known