Lambda Calculus

Ken Q Pu, Faculty of Science, Ontario Tech University

λ-Calculus, aka Lambda Calculus (LC)

- Lambda Calculus is the *smallest universal programming language* (Rojas).
- LC is an alternative to TM.
- LC consists of a **syntax** that describes *valid* LC expressions as strings.
- LC also has a set of transformation rules

Atomic building block: functions

Every value in LC is a function with a single input and single output.

Functions

What can be the input of this function?

- The input must be a value in LC, and thus a function.

What can be the output of this function?

- The output value must be a function as well.

LC is a language describing functions

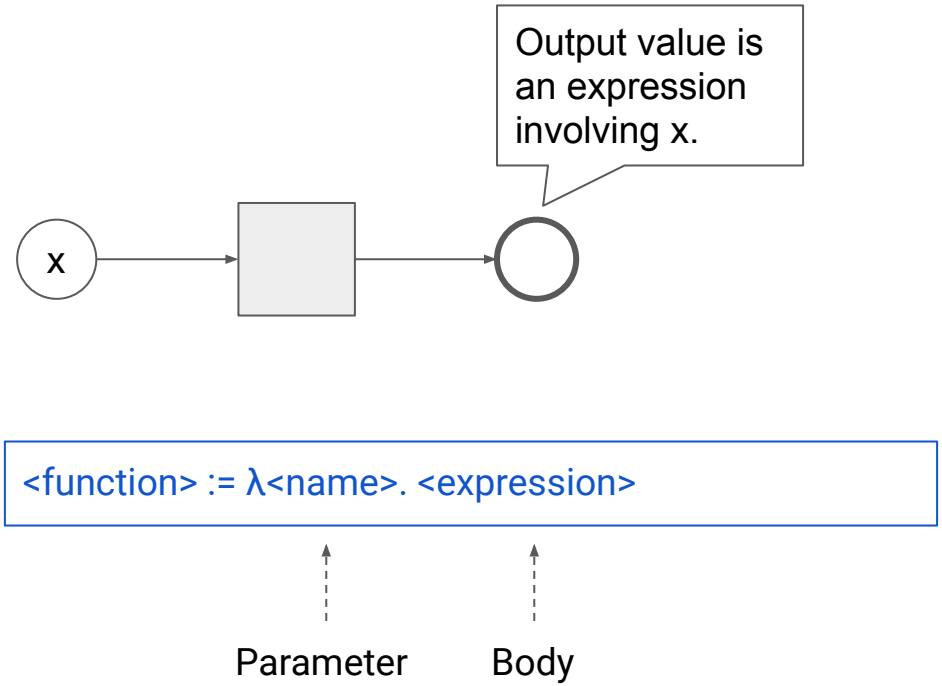
Function

- Definition of a function:
  Describe the output value in terms of the input value

Application

- Derive an output value by applying a function to an input value.

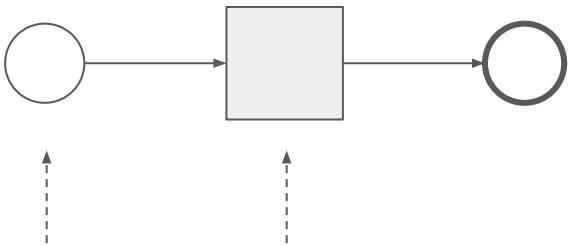
Names and function definitions



We need to give the input value by a name, which we call the **parameter** of the function.

Then, we can define the output value in terms of the input parameter name using an expression which we call the **body** of the function.

Application



<application> := (<expression> <expression>)

Input

Application allows us to derive *new values* from an existing value as the input to a function.

Very important to remember:

- Input value is a function.
- Function is a value.
- Output value is a function.

Because everything is a function.

The formal grammar of LC

```
<function>      := λ<name>. <expression>
<application>   := ( <expression> <expression> )
<expression>    := <name> | <function> | <application>
```

Ken Q Pu, Faculty of Science, Ontario Tech University

Some syntactic considerations

Parentheses are optional:

$$f\ x \equiv (f\ x)$$

If we have multiple applications, we use left association rule.

$$f\ g\ x\ y \equiv (((f\ g)\ x)\ y)$$

It's difficult to typeset λ, so we may use the backslash "\" as a substitution.

$$\backslash x.e \equiv \lambda x.e$$

For nested function definitions, we use right association rule.

$$\backslash x.\backslash y.\backslash z.e \equiv (\backslash x.(\backslash y.(\backslash z.e))))$$

Semantics: what does it all mean?

- x
  This is just a value. It can be anything.
- \x.x
  This is the identity function. The output is always just the input value.
- \x.(f x)
  This is the same as the function f, as the output is the output of f.
- \x.\y.y
  Challenge.

Currying

Currying is a method that simulates functions with multiple parameters with nested functions of single parameters.

Single parameters are called unary functions.

Currying is to use unary functions to build functions with higher arity.

\(x, y, z) <body>

The body is an expression involving x, y and z. ***This is not allowed by LC****.*

\x. (\y. (\z. <body>))

= \x.\y.\z. <body>

But we can achieve the same by nested function abstractions.

Currying

Currying is a method that simulates functions with multiple parameters with nested functions of single parameters.

Single parameters are called unary functions.

Currying is to use unary functions to build functions with higher arity.

Currying is a fundamental technique in building numbers and performing arithmetics in Lambda Calculus.

Ken Q Pu, Faculty of Science, Ontario Tech University

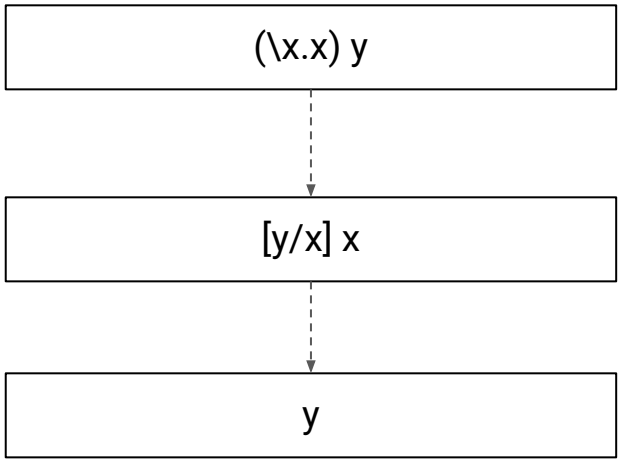
Application as substitution

Consider the identity function: \x.x

Let's apply it to some value y.

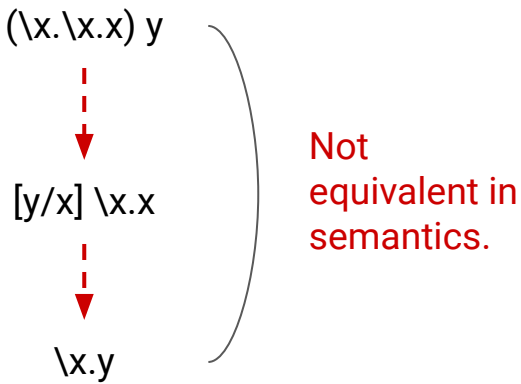
LC has a rewriting rule that allows us to compute the output value:

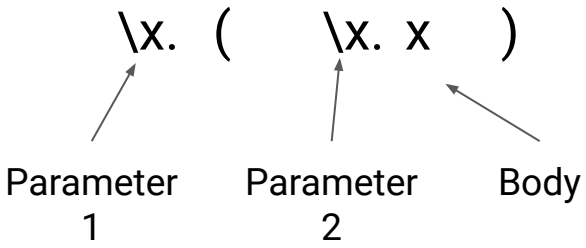
- Substitute the parameters by the input value in the body of the function expression.
- [input / parameter] body



Substitutions are constrained

Careless substitutions will cause serious semantic problems. LC explicitly forbidden such bad substitutions.





The body is referring to parameter 2. This is called name shadowing.

The argument y is only applied to parameter 1.

What's to come?

- Formalize parameter name collision and name shadowing
  - Free and bound variables
- Constrain substitutions during application to ensure semantic correctness
  - Parameter renaming
- Formalize expression rewrite rules
  - α-conversion: parameter renaming
  - β-reduction: function application using (legal) substitution
  - η-reduction: simply function definitions