Scope

- A term with no free variable is said to be closed
- Closed terms are also called combinators
- The simplest combinator is called the identity function:
  id = λx. x

Operational Semantics

- $(\lambda x.\ t_{12})t_2$ -> $(x \mapsto t_2)t_{12}$
  - Evaluate the term t_{12} by replacing every occurrence of x with t_2
  - What is the reduction result of (λx. x) y ?
  - What is the evaluation result of the term (λx. x (λx. x))(u r) ?
  - All terms of the form $(\lambda x.\ t_{12})t_2$ is called redex (reducible expression)
  - The operation of rewriting a redex according to the above rule is called beta-reduction

An Example of Reduction

- (λx. x) ((λx. x)(λz. (λx. x) z))

-> (λx. x)(λz. (λx. x) z)

-> λz. (λx. x) z

Programming in the Lambda-Calculus

- Multiple arguments
  - Lambda-calculus provides no built-in support for multi-argument functions
  - But we can use higher-order functions to achieve the same effect

Multiple Arguments

- Suppose
  - s is a term involving two free variables x and y
  - We want to write a function f, such that for each pair of arguments (v, w), f yields the result of substituting v for x, and w for y
  - f = λx. λy. s
  - Applying f to (v, w): f v w

Multiple Arguments

- The transformation of multi-argument functions into higher-order functions is called ***currying***

Church Booleans

- Define the terms tru and fls as follows:

  tru = λt. λf. t

  fls = λt. λf. f
- The terms **tru** and **fls** can be viewed as representing the boolean values "true" and "false"
- They can be used to test the truth of a boolean expression

`test` function

- test = λl. λm. λn. l m n
- test is a combinator

  test tru v w

= (λl. λm. λn. l m n) tru v w

-> (λm. λn. tru m n) v w

-> (λn. tru v n) w

-> tru v w

= (λt. λf. t) v w

-> (λf. v) w

-> v

`test` function

- What is the semantic meaning of `test b v w` function?
- test b v w = b v w

`and` function

- and = λb. λc. b c fls
- What is the reduction of and tru tru?
- What is the reduction of and fls tru?
- Can you define logical or function?

`pair` function

- pair = λf. λs. λb. b f s
- fst = λp. p tru
- snd = λp. p fls
- That is, pair v w is a function that, when applied to a boolean value b, this application yields v if b is tru, and w if b is fls
- Therefore, fst and snd can be implemented by supplying the appropriate boolean

`pair` function

- What is the reduction of fst (pair v w)?

fst (pair v w)
= fst ((λf. λs. λb. b f s) v w)
-> fst ((λs. λb. b v s) w)
-> fst (λb. b v w)
= (λp. p tru) (λb. b v w)
-> (λb. b v w) tru
-> tru v w
->* v

`pair` function

- What is the reduction of snd (pair v w)?