Lambda calculus

Madhavan Mukund, **S P Suresh**

Programming Language Concepts
Lecture 17, 14 March 2023

λ-calculus

- A notation for **computable functions**

λ-calculus

- A notation for **computable functions**
  - **Alonzo Church**

λ-calculus

- A notation for **computable functions**
  - **Alonzo Church**
- How do we describe a function?

λ-calculus

- A notation for **computable functions**
  - **Alonzo Church**
- How do we describe a function?
  - By its graph – a binary relation between **domain** and **codomain**

λ-calculus

- A notation for **computable functions**
  - **Alonzo Church**
- How do we describe a function?
  - By its graph – a binary relation between **domain** and **codomain**
  - Single-valued

λ-calculus

- A notation for **computable functions**
  - **Alonzo Church**
- How do we describe a function?
  - By its graph – a binary relation between **domain** and **codomain**
  - Single-valued
  - **Extensional** – graph completely defines the function

λ-calculus

- A notation for **computable functions**
  - **Alonzo Church**
- How do we describe a function?
  - By its graph – a binary relation between **domain** and **codomain**
  - Single-valued
  - **Extensional** – graph completely defines the function
- An extensional definition is not suitable for computation

λ-calculus

- A notation for **computable functions**
  - **Alonzo Church**
- How do we describe a function?
  - By its graph – a binary relation between **domain** and **codomain**
  - Single-valued
  - **Extensional** – graph completely defines the function
- An extensional definition is not suitable for computation
  - All sorting functions are the same!

λ-calculus

- A notation for **computable functions**
  - **Alonzo Church**
- How do we describe a function?
  - By its graph – a binary relation between **domain** and **codomain**
  - Single-valued
  - **Extensional** – graph completely defines the function
- An extensional definition is not suitable for computation
  - All sorting functions are the same!
- Need an **intensional** definition

λ-calculus

- A notation for **computable functions**
  - **Alonzo Church**
- How do we describe a function?
  - By its graph – a binary relation between **domain** and **codomain**
  - Single-valued
  - **Extensional** – graph completely defines the function
- An extensional definition is not suitable for computation
  - All sorting functions are the same!
- Need an **intensional** definition
  - How are outputs computed from inputs?

λ-calculus: syntax

- Assume a countably infinite set *Var* of variables

λ-calculus: syntax

- Assume a countably infinite set *Var* of variables
- The set Λ of lambda expressions is given by

$$\Lambda = x \mid \lambda x \cdot M \mid MN$$

where $x \in Var$ and $M, N \in \Lambda$.

λ-calculus: syntax

- Assume a countably infinite set Var of variables
- The set Λ of lambda expressions is given by

$$\Lambda = x \mid \lambda x \cdot M \mid MN$$

where $x \in Var$ and $M, N \in \Lambda$.

- $\lambda x \cdot M$: **Abstraction**

λ-calculus: syntax

- Assume a countably infinite set Var of variables
- The set Λ of lambda expressions is given by

$$\Lambda = x \mid \lambda x \cdot M \mid MN$$

  where $x \in Var$ and $M, N \in \Lambda$.
- $\lambda x \cdot M$: **Abstraction**
  - A function of x with computation rule M.

λ-calculus: syntax

- Assume a countably infinite set Var of variables
- The set Λ of lambda expressions is given by

$$\Lambda = x \mid \lambda x \cdot M \mid MN$$

  where $x \in Var$ and $M, N \in \Lambda$.
- $\lambda x \cdot M$: **Abstraction**
  - A function of x with computation rule M.
  - "Abstracts" the computation rule M over arbitrary input values x

λ-calculus: syntax

- Assume a countably infinite set Var of variables
- The set Λ of lambda expressions is given by

$$\Lambda = x \mid \lambda x \cdot M \mid MN$$

  where $x \in Var$ and $M, N \in \Lambda$.

- $\lambda x \cdot M$: **Abstraction**
  - A function of x with computation rule M.
  - "Abstracts" the computation rule M over arbitrary input values x
  - Like writing $f(x) = e$, but not assigning a name f

λ-calculus: syntax

- Assume a countably infinite set Var of variables
- The set Λ of lambda expressions is given by

$$\Lambda = x \mid \lambda x \cdot M \mid MN$$

  where $x \in Var$ and $M, N \in \Lambda$.
- $\lambda x \cdot M$: **Abstraction**
  - A function of x with computation rule M.
  - "Abstracts" the computation rule M over arbitrary input values x
  - Like writing $f(x) = e$, but not assigning a name f
- MN: **Application**

λ-calculus: syntax

- Assume a countably infinite set Var of variables
- The set Λ of lambda expressions is given by

$$\Lambda = x \mid \lambda x \cdot M \mid MN$$

  where $x \in Var$ and $M, N \in \Lambda$.
- $\lambda x \cdot M$: **Abstraction**
  - A function of x with computation rule M.
  - "Abstracts" the computation rule M over arbitrary input values x
  - Like writing $f(x) = e$, but not assigning a name f
- MN: **Application**
  - Apply the function M to the argument N

λ-calculus: syntax…

- Can write expressions such as xx — no types!

λ-calculus: syntax…

- Can write expressions such as xx — no types!
- What can we do without types?

λ-calculus: syntax...

- Can write expressions such as xx — no types!
- What can we do without types?
    - Set theory as a basis for mathematics

λ-calculus: syntax...

- Can write expressions such as xx — no types!
- What can we do without types?
  - Set theory as a basis for mathematics
  - Bit strings in memory

λ-calculus: syntax...

- Can write expressions such as xx — no types!
- What can we do without types?
  - Set theory as a basis for mathematics
  - Bit strings in memory
- In an untyped world, some data is **meaningful**

λ-calculus: syntax...

- Can write expressions such as xx — no types!
- What can we do without types?
  - Set theory as a basis for mathematics
  - Bit strings in memory
- In an untyped world, some data is **meaningful**
- Functions manipulate meaningful data to yield meaningful data

λ-calculus: syntax...

- Can write expressions such as xx — no types!
- What can we do without types?
  - Set theory as a basis for mathematics
  - Bit strings in memory
- In an untyped world, some data is **meaningful**
- Functions manipulate meaningful data to yield meaningful data
- Can also apply functions to non-meaningful data, but the result has no significance

λ-calculus: syntax...

- Application associates to the left

λ-calculus: syntax…

- Application associates to the left
  - $(MN)P$ is abbreviated MNP

λ-calculus: syntax...

- Application associates to the left
  - $(MN)P$ is abbreviated MNP
- Abstraction associates to the right

λ-calculus: syntax…

- Application associates to the left
  - $(MN)P$ is abbreviated MNP
- Abstraction associates to the right
  - $\lambda x \cdot (\lambda y \cdot M)$ is abbreviated $\lambda x \cdot \lambda y \cdot M$

λ-calculus: syntax...

- Application associates to the left
  - $(MN)P$ is abbreviated MNP
- Abstraction associates to the right
  - $\lambda x \cdot (\lambda y \cdot M)$ is abbreviated $\lambda x \cdot \lambda y \cdot M$
  - More drastically, $\lambda x_1 \cdot (\lambda x_2 \cdots (\lambda x_n \cdot M) \cdots)$ is abbreviated $\lambda x_1 x_2 \cdots x_n \cdot M$

λ-calculus: syntax…

- Application associates to the left
  - $(MN)P$ is abbreviated MNP
- Abstraction associates to the right
  - $\lambda x \cdot (\lambda y \cdot M)$ is abbreviated $\lambda x \cdot \lambda y \cdot M$
  - More drastically, $\lambda x_1 \cdot (\lambda x_2 \cdots (\lambda x_n \cdot M) \cdots)$ is abbreviated $\lambda x_1 x_2 \cdots x_n \cdot M$
  - $\lambda x \cdot MN$ means $(\lambda x \cdot (MN))$. Everything after the $\cdot$ is the body.

λ-calculus: syntax...

- Application associates to the left
  - $(MN)P$ is abbreviated MNP
- Abstraction associates to the right
  - $\lambda x \cdot (\lambda y \cdot M)$ is abbreviated $\lambda x \cdot \lambda y \cdot M$
  - More drastically, $\lambda x_1 \cdot (\lambda x_2 \cdots (\lambda x_n \cdot M) \cdots)$ is abbreviated $\lambda x_1 x_2 \cdots x_n \cdot M$
  - $\lambda x \cdot MN$ means $(\lambda x \cdot (MN))$. Everything after the $\cdot$ is the body.
  - Use $(\lambda x \cdot M)N$ for applying $\lambda x \cdot M$ to N

λ-calculus: syntax...

- Application associates to the left
  - $(MN)P$ is abbreviated MNP
- Abstraction associates to the right
  - $\lambda x \cdot (\lambda y \cdot M)$ is abbreviated $\lambda x \cdot \lambda y \cdot M$
  - More drastically, $\lambda x_1 \cdot (\lambda x_2 \cdots (\lambda x_n \cdot M) \cdots)$ is abbreviated $\lambda x_1 x_2 \cdots x_n \cdot M$
  - $\lambda x \cdot MN$ means $(\lambda x \cdot (MN))$. Everything after the $\cdot$ is the body.
  - Use $(\lambda x \cdot M)N$ for applying $\lambda x \cdot M$ to N
- Examples

λ-calculus: syntax...

- Application associates to the left
  - $(MN)P$ is abbreviated MNP
- Abstraction associates to the right
  - $\lambda x \cdot (\lambda y \cdot M)$ is abbreviated $\lambda x \cdot \lambda y \cdot M$
  - More drastically, $\lambda x_1 \cdot (\lambda x_2 \cdots (\lambda x_n \cdot M) \cdots)$ is abbreviated $\lambda x_1 x_2 \cdots x_n \cdot M$
  - $\lambda x \cdot MN$ means $(\lambda x \cdot (MN))$. Everything after the $\cdot$ is the body.
  - Use $(\lambda x \cdot M)N$ for applying $\lambda x \cdot M$ to N
- Examples
  - $(\lambda x \cdot x)(\lambda y \cdot y)(\lambda z \cdot z)$ is short for $((\lambda x \cdot x)(\lambda y \cdot y))(\lambda z \cdot z)$

λ-calculus: syntax...

- Application associates to the left
  - $(MN)P$ is abbreviated MNP
- Abstraction associates to the right
  - $\lambda x \cdot (\lambda y \cdot M)$ is abbreviated $\lambda x \cdot \lambda y \cdot M$
  - More drastically, $\lambda x_1 \cdot (\lambda x_2 \cdots (\lambda x_n \cdot M) \cdots)$ is abbreviated $\lambda x_1 x_2 \cdots x_n \cdot M$
  - $\lambda x \cdot MN$ means $(\lambda x \cdot (MN))$. Everything after the $\cdot$ is the body.
  - Use $(\lambda x \cdot M)N$ for applying $\lambda x \cdot M$ to N
- Examples
  - $(\lambda x \cdot x)(\lambda y \cdot y)(\lambda z \cdot z)$ is short for $((\lambda x \cdot x)(\lambda y \cdot y))(\lambda z \cdot z)$
  - $\lambda f \cdot (\lambda u \cdot f(uu))(\lambda u \cdot f(uu))$ is short for $(\lambda f \cdot ((\lambda u \cdot f(uu))(\lambda u \cdot f(uu))))$

The computation rule β

- Basic rule for computation (rewriting) is called β-**reduction** (or **contraction**)

The computation rule β

- Basic rule for computation (rewriting) is called β-**reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_{\beta} M[x := N]$

The computation rule β

- Basic rule for computation (rewriting) is called β-**reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_{\beta} M[x := N]$
  - A term of the form $(\lambda x \cdot M)N$ is a **redex**

The computation rule β

- Basic rule for computation (rewriting) is called β-**reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_{\beta} M[x := N]$
  - A term of the form $(\lambda x \cdot M)N$ is a **redex**
  - $M[x := N]$ is the **contractum**

The computation rule β

- Basic rule for computation (rewriting) is called β-**reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_{\beta} M[x := N]$
  - A term of the form $(\lambda x \cdot M)N$ is a **redex**
  - $M[x := N]$ is the **contractum**
- $M[x := N]$: substitute **free** occurrences of x in M by N

The computation rule β

- Basic rule for computation (rewriting) is called β-**reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_{\beta} M[x := N]$
  - A term of the form $(\lambda x \cdot M)N$ is a **redex**
  - $M[x := N]$ is the **contractum**
- $M[x := N]$: substitute **free** occurrences of x in M by N
- This is the normal rule we use for functions:

The computation rule β

- Basic rule for computation (rewriting) is called β-**reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_{\beta} M[x := N]$
  - A term of the form $(\lambda x \cdot M)N$ is a **redex**
  - $M[x := N]$ is the **contractum**
- $M[x := N]$: substitute **free** occurrences of x in M by N
- This is the normal rule we use for functions:
  - $f(x) = 2x^3 + 5x + 3$

The computation rule β

- Basic rule for computation (rewriting) is called β-**reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_\beta M[x := N]$
  - A term of the form $(\lambda x \cdot M)N$ is a **redex**
  - $M[x := N]$ is the **contractum**
- $M[x := N]$: substitute **free** occurrences of x in M by N
- This is the normal rule we use for functions:
  - $f(x) = 2x^3 + 5x + 3$
  - $f(7) = (2x^3 + 5x + 3)[x := 7] = 2 \cdot 7^3 + 5 \cdot 7 + 3 = 724$

The computation rule β

- Basic rule for computation (rewriting) is called β-**reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_{\beta} M[x := N]$
  - A term of the form $(\lambda x \cdot M)N$ is a **redex**
  - $M[x := N]$ is the **contractum**
- $M[x := N]$: substitute **free** occurrences of x in M by N
- This is the normal rule we use for functions:
  - $f(x) = 2x^3 + 5x + 3$
  - $f(7) = (2x^3 + 5x + 3)[x := 7] = 2 \cdot 7^3 + 5 \cdot 7 + 3 = 724$
- β is the **only** rule we need

The computation rule β

- Basic rule for computation (rewriting) is called β-**reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_\beta M[x := N]$
  - A term of the form $(\lambda x \cdot M)N$ is a **redex**
  - $M[x := N]$ is the **contractum**
- $M[x := N]$: substitute **free** occurrences of x in M by N
- This is the normal rule we use for functions:
  - $f(x) = 2x^3 + 5x + 3$
  - $f(7) = (2x^3 + 5x + 3)[x := 7] = 2 \cdot 7^3 + 5 \cdot 7 + 3 = 724$
- β is the **only** rule we need
- MN is meaningful only if M is of the form $\lambda x \cdot P$

The computation rule β

- Basic rule for computation (rewriting) is called β-**reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_{\beta} M[x := N]$
  - A term of the form $(\lambda x \cdot M)N$ is a **redex**
  - $M[x := N]$ is the **contractum**
- $M[x := N]$: substitute **free** occurrences of x in M by N
- This is the normal rule we use for functions:
  - $f(x) = 2x^3 + 5x + 3$
  - $f(7) = (2x^3 + 5x + 3)[x := 7] = 2 \cdot 7^3 + 5 \cdot 7 + 3 = 724$
- β is the **only** rule we need
- MN is meaningful only if M is of the form $\lambda x \cdot P$
  - Cannot do anything with terms like xx or $(y(\lambda x \cdot x))(\lambda y \cdot y)$

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M
- $\mathbf{fv}(M)$: set of all variables occurring free in M

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M
- $\mathbf{fv}(M)$: set of all variables occurring free in M
  - $\mathbf{fv}(x) = \{x\}$, for any $x \in Var$

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M
- $\mathbf{fv}(M)$: set of all variables occurring free in M
  - $\mathbf{fv}(x) = \{x\}$, for any $x \in Var$
  - $\mathbf{fv}(MN) = \mathbf{fv}(M) \cup \mathbf{fv}(N)$

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M
- $\mathbf{fv}(M)$: set of all variables occurring free in M
  - $\mathbf{fv}(x) = \{x\}$, for any $x \in Var$
  - $\mathbf{fv}(MN) = \mathbf{fv}(M) \cup \mathbf{fv}(N)$
  - $\mathbf{fv}(\lambda x \cdot M) = \mathbf{fv}(M) \setminus \{x\}$

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M
- $\mathbf{fv}(M)$: set of all variables occurring free in M
  - $\mathbf{fv}(x) = \{x\}$, for any $x \in Var$
  - $\mathbf{fv}(MN) = \mathbf{fv}(M) \cup \mathbf{fv}(N)$
  - $\mathbf{fv}(\lambda x \cdot M) = \mathbf{fv}(M) \setminus \{x\}$
- $\mathbf{bv}(M)$: set of all variables occurring bound in M

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M
- $\mathbf{fv}(M)$: set of all variables occurring free in M
  - $\mathbf{fv}(x) = \{x\}$, for any $x \in Var$
  - $\mathbf{fv}(MN) = \mathbf{fv}(M) \cup \mathbf{fv}(N)$
  - $\mathbf{fv}(\lambda x \cdot M) = \mathbf{fv}(M) \setminus \{x\}$
- $\mathbf{bv}(M)$: set of all variables occurring bound in M
  - $\mathbf{bv}(x) = \emptyset$, for any $x \in Var$

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M
- $\mathbf{fv}(M)$: set of all variables occurring free in M
  - $\mathbf{fv}(x) = \{x\}$, for any $x \in Var$
  - $\mathbf{fv}(MN) = \mathbf{fv}(M) \cup \mathbf{fv}(N)$
  - $\mathbf{fv}(\lambda x \cdot M) = \mathbf{fv}(M) \setminus \{x\}$
- $\mathbf{bv}(M)$: set of all variables occurring bound in M
  - $\mathbf{bv}(x) = \emptyset$, for any $x \in Var$
  - $\mathbf{bv}(MN) = \mathbf{bv}(M) \cup \mathbf{bv}(N)$

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M
- $\mathbf{fv}(M)$: set of all variables occurring free in M
  - $\mathbf{fv}(x) = \{x\}$, for any $x \in Var$
  - $\mathbf{fv}(MN) = \mathbf{fv}(M) \cup \mathbf{fv}(N)$
  - $\mathbf{fv}(\lambda x \cdot M) = \mathbf{fv}(M) \setminus \{x\}$
- $\mathbf{bv}(M)$: set of all variables occurring bound in M
  - $\mathbf{bv}(x) = \emptyset$, for any $x \in Var$
  - $\mathbf{bv}(MN) = \mathbf{bv}(M) \cup \mathbf{bv}(N)$
  - $\mathbf{bv}(\lambda x \cdot M) = \mathbf{bv}(M) \cup (\{x\} \cap \mathbf{fv}(M))$

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M
- $\mathbf{fv}(M)$: set of all variables occurring free in M
  - $\mathbf{fv}(x) = \{x\}$, for any $x \in Var$
  - $\mathbf{fv}(MN) = \mathbf{fv}(M) \cup \mathbf{fv}(N)$
  - $\mathbf{fv}(\lambda x \cdot M) = \mathbf{fv}(M) \setminus \{x\}$
- $\mathbf{bv}(M)$: set of all variables occurring bound in M
  - $\mathbf{bv}(x) = \emptyset$, for any $x \in Var$
  - $\mathbf{bv}(MN) = \mathbf{bv}(M) \cup \mathbf{bv}(N)$
  - $\mathbf{bv}(\lambda x \cdot M) = \mathbf{bv}(M) \cup (\{x\} \cap \mathbf{fv}(M))$
- Example: $M = xy(\lambda x \cdot z)(\lambda y \cdot y)$

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M
- $\mathbf{fv}(M)$: set of all variables occurring free in M
  - $\mathbf{fv}(x) = \{x\}$, for any $x \in Var$
  - $\mathbf{fv}(MN) = \mathbf{fv}(M) \cup \mathbf{fv}(N)$
  - $\mathbf{fv}(\lambda x \cdot M) = \mathbf{fv}(M) \setminus \{x\}$
- $\mathbf{bv}(M)$: set of all variables occurring bound in M
  - $\mathbf{bv}(x) = \emptyset$, for any $x \in Var$
  - $\mathbf{bv}(MN) = \mathbf{bv}(M) \cup \mathbf{bv}(N)$
  - $\mathbf{bv}(\lambda x \cdot M) = \mathbf{bv}(M) \cup (\{x\} \cap \mathbf{fv}(M))$
- Example: $M = xy(\lambda x \cdot z)(\lambda y \cdot y)$
  - $\mathbf{fv}(M) = \{x, y, z\}$ $\quad$ $\mathbf{bv}(M) = \{y\}$

Free and bound variables

- An occurrence of a variable x in M is free if it does not occur in the scope of a λx inside M
- $\mathbf{fv}(M)$: set of all variables occurring free in M
  - $\mathbf{fv}(x) = \{x\}$, for any $x \in Var$
  - $\mathbf{fv}(MN) = \mathbf{fv}(M) \cup \mathbf{fv}(N)$
  - $\mathbf{fv}(\lambda x \cdot M) = \mathbf{fv}(M) \setminus \{x\}$
- $\mathbf{bv}(M)$: set of all variables occurring bound in M
  - $\mathbf{bv}(x) = \emptyset$, for any $x \in Var$
  - $\mathbf{bv}(MN) = \mathbf{bv}(M) \cup \mathbf{bv}(N)$
  - $\mathbf{bv}(\lambda x \cdot M) = \mathbf{bv}(M) \cup (\{x\} \cap \mathbf{fv}(M))$
- Example: $M = xy(\lambda x \cdot z)(\lambda y \cdot y)$
  - $\mathbf{fv}(M) = \{x, y, z\}$ $\quad$ $\mathbf{bv}(M) = \{y\}$
  - **Warning**: Possible for a variable to be both in $\mathbf{fv}(M)$ and $\mathbf{bv}(M)$

Variable capture

- Consider $N = \lambda x \cdot (\lambda y \cdot xy)$ and $M = Ny$

Variable capture

- Consider $N = \lambda x \cdot (\lambda y \cdot xy)$ and $M = Ny$
  - N takes two arguments and applies the first argument to the second

Variable capture

- Consider $N = \lambda x \cdot (\lambda y \cdot xy)$ and $M = Ny$
  - N takes two arguments and applies the first argument to the second
  - M fixes the first argument of N

Variable capture

- Consider $N = \lambda x \cdot (\lambda y \cdot xy)$ and $M = Ny$
  - N takes two arguments and applies the first argument to the second
  - M fixes the first argument of N
  - Meaning of M: Take an argument and apply y to it!

Variable capture

- Consider $N = \lambda x \cdot (\lambda y \cdot xy)$ and $M = Ny$
  - N takes two arguments and applies the first argument to the second
  - M fixes the first argument of N
  - Meaning of M: Take an argument and apply y to it!
- β-reduction on M yields $\lambda y \cdot yy$

Variable capture

- Consider $N = \lambda x \cdot (\lambda y \cdot xy)$ and $M = Ny$
  - N takes two arguments and applies the first argument to the second
  - M fixes the first argument of N
  - Meaning of M: Take an argument and apply y to it!
- β-reduction on M yields $\lambda y \cdot yy$
  - Meaning: Take an argument and apply it to itself!

Variable capture

- Consider $N = \lambda x \cdot (\lambda y \cdot xy)$ and $M = Ny$
  - N takes two arguments and applies the first argument to the second
  - M fixes the first argument of N
  - Meaning of M: Take an argument and apply y to it!
- β-reduction on M yields $\lambda y \cdot yy$
  - Meaning: Take an argument and apply it to itself!
- The y substituted for inner x has been “confused” with the y bound by λy

Variable capture

- Consider $N = \lambda x \cdot (\lambda y \cdot xy)$ and $M = Ny$
  - N takes two arguments and applies the first argument to the second
  - M fixes the first argument of N
  - Meaning of M: Take an argument and apply y to it!
- β-reduction on M yields $\lambda y \cdot yy$
  - Meaning: Take an argument and apply it to itself!
- The y substituted for inner x has been "confused" with the y bound by λy
- Rename bound variables to avoid capture

Variable capture

- Consider $N = \lambda x \cdot (\lambda y \cdot xy)$ and $M = Ny$
  - N takes two arguments and applies the first argument to the second
  - M fixes the first argument of N
  - Meaning of M: Take an argument and apply y to it!
- β-reduction on M yields $\lambda y \cdot yy$
  - Meaning: Take an argument and apply it to itself!
- The y substituted for inner x has been "confused" with the y bound by λy
- Rename bound variables to avoid capture
  - $(\lambda x \cdot (\lambda y \cdot xy))y = (\lambda x \cdot (\lambda z \cdot xz))y \longrightarrow_\beta \lambda z \cdot yz$

Variable capture

- Consider $N = \lambda x \cdot (\lambda y \cdot xy)$ and $M = Ny$
  - N takes two arguments and applies the first argument to the second
  - M fixes the first argument of N
  - Meaning of M: Take an argument and apply y to it!
- β-reduction on M yields $\lambda y \cdot yy$
  - Meaning: Take an argument and apply it to itself!
- The y substituted for inner x has been "confused" with the y bound by λy
- Rename bound variables to avoid capture
  - $(\lambda x \cdot (\lambda y \cdot xy))y = (\lambda x \cdot (\lambda z \cdot xz))y \longrightarrow_{\beta} \lambda z \cdot yz$
- Renaming bound variables does not change the funciton

Variable capture

- Consider $N = \lambda x \cdot (\lambda y \cdot xy)$ and $M = Ny$
  - N takes two arguments and applies the first argument to the second
  - M fixes the first argument of N
  - Meaning of M: Take an argument and apply y to it!
- β-reduction on M yields $\lambda y \cdot yy$
  - Meaning: Take an argument and apply it to itself!
- The y substituted for inner x has been "confused" with the y bound by λy
- Rename bound variables to avoid capture
  - $(\lambda x \cdot (\lambda y \cdot xy))y = (\lambda x \cdot (\lambda z \cdot xz))y \longrightarrow_\beta \lambda z \cdot yz$
- Renaming bound variables does not change the funciton
  - $f(x) = 2x + 7$ vs $f(z) = 2z + 7$

$M[x \coloneqq N]$

- $x[x \coloneqq N] = N$

$M[x := N]$

- $x[x := N] = N$
- $y[x := N] = y$, where $y \in Var$ and $y \neq x$

$M[x := N]$

- $x[x := N] = N$
- $y[x := N] = y$, where $y \in Var$ and $y \neq x$
- $(PQ)[x := N] = (P[x := N])(Q[x := N])$

$M[x := N]$

- $x[x := N] = N$
- $y[x := N] = y$, where $y \in Var$ and $y \neq x$
- $(PQ)[x := N] = (P[x := N])(Q[x := N])$
- $(\lambda x \cdot P)[x := N] = \lambda x \cdot P$

$M[x := N]$

- $x[x := N] = N$
- $y[x := N] = y$, where $y \in Var$ and $y \neq x$
- $(PQ)[x := N] = (P[x := N])(Q[x := N])$
- $(\lambda x \cdot P)[x := N] = \lambda x \cdot P$
- $(\lambda y \cdot P)[x := N] = \lambda y \cdot (P[x := N])$, where $y \neq x$ and $y \notin \mathbf{fv}(N)$

$M[x := N]$

- $x[x := N] = N$
- $y[x := N] = y$, where $y \in Var$ and $y \neq x$
- $(PQ)[x := N] = (P[x := N])(Q[x := N])$
- $(\lambda x \cdot P)[x := N] = \lambda x \cdot P$
- $(\lambda y \cdot P)[x := N] = \lambda y \cdot (P[x := N])$, where $y \neq x$ and $y \notin \mathbf{fv}(N)$
- $(\lambda y \cdot P)[x := N] = \lambda z \cdot ((P[y := z])[x := N])$, where $y \neq x$, $y \in \mathbf{fv}(N)$, and z does not occur in P or N

$M[x := N]$

- $x[x := N] = N$
- $y[x := N] = y$, where $y \in Var$ and $y \neq x$
- $(PQ)[x := N] = (P[x := N])(Q[x := N])$
- $(\lambda x \cdot P)[x := N] = \lambda x \cdot P$
- $(\lambda y \cdot P)[x := N] = \lambda y \cdot (P[x := N])$, where $y \neq x$ and $y \notin \mathbf{fv}(N)$
- $(\lambda y \cdot P)[x := N] = \lambda z \cdot ((P[y := z])[x := N])$, where $y \neq x$, $y \in \mathbf{fv}(N)$, and z does not occur in P or N
  - We fix a global ordering on Var and choose z to be the **first** variable not occurring in either P or N

$M[x := N]$

- $x[x := N] = N$
- $y[x := N] = y$, where $y \in Var$ and $y \neq x$
- $(PQ)[x := N] = (P[x := N])(Q[x := N])$
- $(\lambda x \cdot P)[x := N] = \lambda x \cdot P$
- $(\lambda y \cdot P)[x := N] = \lambda y \cdot (P[x := N])$, where $y \neq x$ and $y \notin \mathbf{fv}(N)$
- $(\lambda y \cdot P)[x := N] = \lambda z \cdot ((P[y := z])[x := N])$, where $y \neq x$, $y \in \mathbf{fv}(N)$, and z does not occur in P or N
  - We fix a global ordering on Var and choose z to be the **first** variable not occurring in either P or N
  - Makes the definition deterministic

Applying β in context

- We can contract a redex appearing anywhere inside an expression

Applying β in context

- We can contract a redex appearing anywhere inside an expression
- Captured by the following rules

$$(\lambda x \cdot M)N \longrightarrow_\beta M[x := N] \qquad \frac{M \longrightarrow_\beta M'}{MN \longrightarrow_\beta M'N} \qquad \frac{N \longrightarrow_\beta N'}{MN \longrightarrow_\beta MN'} \qquad \frac{M \longrightarrow_\beta M'}{\lambda x \cdot M \longrightarrow_\beta \lambda x \cdot M'}$$

Applying β in context

- We can contract a redex appearing anywhere inside an expression
- Captured by the following rules

$$(\lambda x \cdot M)N \longrightarrow_\beta M[x := N] \qquad \frac{M \longrightarrow_\beta M'}{MN \longrightarrow_\beta M'N} \qquad \frac{N \longrightarrow_\beta N'}{MN \longrightarrow_\beta MN'} \qquad \frac{M \longrightarrow_\beta M'}{\lambda x \cdot M \longrightarrow_\beta \lambda x \cdot M'}$$

- $M \xrightarrow{*}_\beta N$: repeatedly apply β-reduction to get N

Applying β in context

- We can contract a redex appearing anywhere inside an expression
- Captured by the following rules

$$(\lambda x \cdot M)N \longrightarrow_\beta M[x := N] \qquad \frac{M \longrightarrow_\beta M'}{MN \longrightarrow_\beta M'N} \qquad \frac{N \longrightarrow_\beta N'}{MN \longrightarrow_\beta MN'} \qquad \frac{M \longrightarrow_\beta M'}{\lambda x \cdot M \longrightarrow_\beta \lambda x \cdot M'}$$

- $M \xrightarrow{*}_\beta N$: repeatedly apply β-reduction to get N
  - There is a sequence $M_0, M_1, \ldots, M_k$ such that

$$M = M_0 \longrightarrow_\beta M_1 \longrightarrow_\beta \cdots \longrightarrow_\beta M_k = N$$

Encoding arithmetic

- In set theory, use nesting to encode numbers

Encoding arithmetic

- In set theory, use nesting to encode numbers
  - Encoding of n: $\mathbf{n}$

Encoding arithmetic

- In set theory, use nesting to encode numbers
  - Encoding of n: $\mathbf{n}$
  - $\mathbf{n} = \{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{n\text{-}1}\}$

Encoding arithmetic

- In set theory, use nesting to encode numbers
  - Encoding of n: $\mathbf{n}$
  - $\mathbf{n} = \{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{n\text{-}1}\}$
  - Thus

Encoding arithmetic

- In set theory, use nesting to encode numbers
  - Encoding of n: $\mathbf{n}$
  - $\mathbf{n} = \{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{n\text{-}1}\}$
  - Thus
    - $\mathbf{0} = \emptyset$

Encoding arithmetic

- In set theory, use nesting to encode numbers
  - Encoding of n: $\mathbf{n}$
  - $\mathbf{n} = \{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{n\text{-}1}\}$
  - Thus
    - $\mathbf{0} = \emptyset$
    - $\mathbf{1} = \{\emptyset\}$

Encoding arithmetic

- In set theory, use nesting to encode numbers
  - Encoding of n: $\mathbf{n}$
  - $\mathbf{n} = \{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{n\text{-}1}\}$
  - Thus
    - $\mathbf{0} = \emptyset$
    - $\mathbf{1} = \{\emptyset\}$
    - $\mathbf{2} = \{\emptyset, \{\emptyset\}\}$

Encoding arithmetic

- In set theory, use nesting to encode numbers
  - Encoding of n: $\mathbf{n}$
  - $\mathbf{n} = \{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{n\text{-}1}\}$
  - Thus
    - $\mathbf{0} = \emptyset$
    - $\mathbf{1} = \{\emptyset\}$
    - $\mathbf{2} = \{\emptyset, \{\emptyset\}\}$
    - $\mathbf{3} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$

Encoding arithmetic

- In set theory, use nesting to encode numbers
  - Encoding of n: $\mathbf{n}$
  - $\mathbf{n} = \{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{n\text{-}1}\}$
  - Thus
    - $\mathbf{0} = \emptyset$
    - $\mathbf{1} = \{\emptyset\}$
    - $\mathbf{2} = \{\emptyset, \{\emptyset\}\}$
    - $\mathbf{3} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$
- In λ-calculus, we encode n by the number of times we apply a function (**successor**) to an element (**zero**)

Church numerals

- $\mathbf{n} = \lambda f x \cdot f^n x$

Church numerals

- $\mathbf{n} = \lambda fx \cdot f^n x$
  - $f^0 x = x$

Church numerals

- $\mathbf{n} = \lambda f x \cdot f^n x$
  - $f^0 x = x$
  - $f^{n+1} x = f(f^n x)$

Church numerals

- $\mathbf{n} = \lambda fx \cdot f^n x$
  - $f^0 x = x$
  - $f^{n+1} x = f(f^n x)$
  - Thus $f^n x = f(f(\cdots(fx)\cdots))$, where f is applied repeatedly n times

Church numerals

- $\mathbf{n} = \lambda fx \cdot f^n x$
    - $f^0 x = x$
    - $f^{n+1} x = f(f^n x)$
    - Thus $f^n x = f(f(\cdots(fx)\cdots))$, where f is applied repeatedly n times
- For instance

Church numerals

- $\mathbf{n} = \lambda fx \cdot f^n x$
  - $f^0 x = x$
  - $f^{n+1} x = f(f^n x)$
  - Thus $f^n x = f(f(\cdots(fx)\cdots))$, where f is applied repeatedly n times
- For instance
  - $\mathbf{0} = \lambda fx \cdot x$

Church numerals

- $\mathbf{n} = \lambda fx \cdot f^n x$
  - $f^0 x = x$
  - $f^{n+1} x = f(f^n x)$
  - Thus $f^n x = f(f(\cdots(fx)\cdots))$, where f is applied repeatedly n times
- For instance
  - $\mathbf{0} = \lambda fx \cdot x$
  - $\mathbf{1} = \lambda fx \cdot fx$

Church numerals

- $\mathbf{n} = \lambda f x \cdot f^n x$
  - $f^0 x = x$
  - $f^{n+1} x = f(f^n x)$
  - Thus $f^n x = f(f(\cdots(fx)\cdots))$, where f is applied repeatedly n times
- For instance
  - $\mathbf{0} = \lambda f x \cdot x$
  - $\mathbf{1} = \lambda f x \cdot f x$
  - $\mathbf{2} = \lambda f x \cdot f(f x)$

Church numerals

- $\mathbf{n} = \lambda fx \cdot f^n x$
  - $f^0 x = x$
  - $f^{n+1} x = f(f^n x)$
  - Thus $f^n x = f(f(\cdots(fx)\cdots))$, where f is applied repeatedly n times
- For instance
  - $\mathbf{0} = \lambda fx \cdot x$
  - $\mathbf{1} = \lambda fx \cdot fx$
  - $\mathbf{2} = \lambda fx \cdot f(fx)$
  - $\mathbf{3} = \lambda fx \cdot f(f(fx))$

Church numerals

- $\mathbf{n} = \lambda fx \cdot f^n x$
  - $f^0 x = x$
  - $f^{n+1} x = f(f^n x)$
  - Thus $f^n x = f(f(\cdots(fx)\cdots))$, where f is applied repeatedly n times
- For instance
  - $\mathbf{0} = \lambda fx \cdot x$
  - $\mathbf{1} = \lambda fx \cdot fx$
  - $\mathbf{2} = \lambda fx \cdot f(fx)$
  - $\mathbf{3} = \lambda fx \cdot f(f(fx))$
  - …

Church numerals

- $\mathbf{n} = \lambda fx \cdot f^n x$
  - $f^0 x = x$
  - $f^{n+1} x = f(f^n x)$
  - Thus $f^n x = f(f(\cdots(fx)\cdots))$, where f is applied repeatedly n times
- For instance
  - $\mathbf{0} = \lambda fx \cdot x$
  - $\mathbf{1} = \lambda fx \cdot fx$
  - $\mathbf{2} = \lambda fx \cdot f(fx)$
  - $\mathbf{3} = \lambda fx \cdot f(f(fx))$
  - …
- $\mathbf{n}\, gy = (\lambda fx \cdot f(\cdots(fx)\cdots))gy \xrightarrow{*}_{\beta} g(\cdots(gy)\cdots) = g^n y$