# Lambda calculus – encoding arithmetic

Madhavan Mukund, **S P Suresh**

Programming Language Concepts

Lecture 18, 21 March 2024

# λ-calculus: syntax

- Assume a countably infinite set *Var* of variables

# λ-calculus: syntax

- Assume a countably infinite set $Var$ of variables
- The set $\Lambda$ 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

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

# λ-calculus: syntax

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

# λ-calculus: syntax

- Basic rule for computation (rewriting) is called **β-reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_\beta M[x := N]$
  - $M[x := N]$: substitute **free** occurrences of $x$ in $M$ by $N$

# λ-calculus: syntax

- Basic rule for computation (rewriting) is called **β-reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_\beta M[x := N]$
  - $M[x := N]$: substitute **free** occurrences of $x$ in $M$ by $N$
- We rename the bound variables in $M$ to avoid "capturing" free variables of $N$ in $M$

# λ-calculus: syntax

- Basic rule for computation (rewriting) is called **β-reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_\beta M[x := N]$
  - $M[x := N]$: substitute **free** occurrences of $x$ in $M$ by $N$
- We rename the bound variables in $M$ to avoid "capturing" free variables of $N$ in $M$
- β-reduction can be applied in any context, replacing a subterm of the form $(\lambda x \cdot M)N$ with $M[x := N]$

# λ-calculus: syntax

- Basic rule for computation (rewriting) is called **β-reduction** (or **contraction**)
  - $(\lambda x \cdot M)N \longrightarrow_\beta M[x := N]$
  - $M[x := N]$: substitute **free** occurrences of $x$ in $M$ by $N$
- We rename the bound variables in $M$ to avoid "capturing" free variables of $N$ in $M$
- β-reduction can be applied in any context, replacing a subterm of the form $(\lambda x \cdot M)N$ with $M[x := N]$
- Multi-step reduction is denoted $\xrightarrow{*}_\beta$

# 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
    - $\mathbf{0} = \varnothing$

# 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
    - $\mathbf{0} = \varnothing$
    - $\mathbf{1} = \{\varnothing\}$

# 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
    - $\mathbf{0} = \varnothing$
    - $\mathbf{1} = \{\varnothing\}$
    - $\mathbf{2} = \{\varnothing, \{\varnothing\}\}$

# 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
    - $\mathbf{0} = \varnothing$
    - $\mathbf{1} = \{\varnothing\}$
    - $\mathbf{2} = \{\varnothing, \{\varnothing\}\}$
    - $\mathbf{3} = \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}$

# Encoding arithmetic

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

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

# 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

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

# 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

# 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
  - $\mathbf{0} = \lambda f x \cdot x$

# 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
  - $\mathbf{0} = \lambda f x \cdot x$
  - $\mathbf{1} = \lambda f x \cdot fx$

# 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
  - $\mathbf{0} = \lambda f x \cdot x$
  - $\mathbf{1} = \lambda f x \cdot fx$
  - $\mathbf{2} = \lambda f x \cdot f(fx)$

# 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
  - $\mathbf{0} = \lambda f x \cdot x$
  - $\mathbf{1} = \lambda f x \cdot fx$
  - $\mathbf{2} = \lambda f x \cdot f(fx)$
  - $\mathbf{3} = \lambda f x \cdot f(f(fx))$

# 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
  - $\mathbf{0} = \lambda f x \cdot x$
  - $\mathbf{1} = \lambda f x \cdot fx$
  - $\mathbf{2} = \lambda f x \cdot f(fx)$
  - $\mathbf{3} = \lambda f x \cdot f(f(fx))$
  - ...

# 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
  - $\mathbf{0} = \lambda f x \cdot x$
  - $\mathbf{1} = \lambda f x \cdot fx$
  - $\mathbf{2} = \lambda f x \cdot f(fx)$
  - $\mathbf{3} = \lambda f x \cdot f(f(fx))$
  - $\ldots$
- $\mathbf{n}\, g\, y = (\lambda f x \cdot f(\cdots(fx)\cdots))\, g\, y \xrightarrow{*}_\beta g(\cdots(gy)\cdots) = g^n y$

# Encoding arithmetic functions

- **Successor function**: $succ(n) = n + 1$

# Encoding arithmetic functions

- **Successor function**: $succ(n) = n + 1$
- **succ** $= \lambda pfx \cdot f(pfx)$

# Encoding arithmetic functions

- **Successor function**: $succ(n) = n + 1$
- **succ** $= \lambda pfx \cdot f\,(pfx)$
- For all $n$ and $m$, if $m = n + 1$ then **succ n** $\xrightarrow{*}_{\beta}$ **m**

# Encoding arithmetic functions

- **Successor function**: $succ(n) = n + 1$
- **succ** $= \lambda pfx \cdot f(pfx)$
- For all $n$ and $m$, if $m = n + 1$ then **succ n** $\xrightarrow{*}_\beta$ **m**
  - **succ n**

$$
\begin{array}{lll}
(\lambda pfx \cdot f(pfx))\ \mathbf{n} & \longrightarrow_\beta & \lambda fx \cdot f(\mathbf{n}\, fx) \\
 & \xrightarrow{*}_\beta & \lambda fx \cdot f(\ f^n x) \\
 & = & \lambda fx \cdot f^{n+1} x \\
 & = & \lambda fx \cdot f^m x \\
 & = & \mathbf{m}
\end{array}
$$

# Encoding arithmetic functions

- **Addition**: $plus(m, n) = m + n$

# Encoding arithmetic functions

- **Addition**: $plus(m, n) = m + n$
- **plus** $= \lambda pq\, f\, x \cdot p\, f\, (q\, f\, x)$

# Encoding arithmetic functions

- **Addition**: $plus(m, n) = m + n$
- **plus** $= \lambda pq\, fx \cdot p\, f\, (q\, f\, x)$
- For all $m$, $n$ and $o$, if $m + n = o$ then **plus m n** $\xrightarrow{*}_{\beta}$ **o**

# Encoding arithmetic functions

- **Addition**: $plus(m, n) = m + n$
- **plus** $= \lambda pq\, fx \cdot p\, f\, (q\, f\, x)$
- For all $m$, $n$ and $o$, if $m + n = o$ then **plus m n** $\xrightarrow{*}_\beta$ **o**
  - **plus m n**

$$
\begin{aligned}
(\lambda pq\, fx \cdot p\, f\, (q\, f\, x))\, \mathbf{m}\, \mathbf{n} &\longrightarrow_\beta (\lambda q\, fx \cdot \mathbf{m}\, f\, (q\, f\, x))\mathbf{n} \\
&\longrightarrow_\beta \lambda\, fx \cdot \mathbf{m}\, f\, (\mathbf{n}\, f\, x) \\
&\xrightarrow{*}_\beta \lambda\, fx \cdot f^m\, (\mathbf{n}\, f\, x) \\
&\xrightarrow{*}_\beta \lambda\, fx \cdot f^m\, (\, f^n\, x) \\
&= \lambda\, fx \cdot f^{m+n}\, x \\
&= \lambda\, fx \cdot f^o\, x \\
&= \mathbf{o}
\end{aligned}
$$

# Encoding arithmetic functions

- **Multiplication**: $mult(m, n) = mn$

# Encoding arithmetic functions

- **Multiplication**: $mult(m, n) = mn$
- **mult** $= \lambda pq\, f \cdot p(q\, f)$

# Encoding arithmetic functions

- **Multiplication**: $mult(m, n) = mn$
- **mult** $= \lambda pq\, f \cdot p(q\, f)$
- For all $m \geq 0$, $(\mathbf{n}\, f)^m y \xrightarrow{*}_\beta f^{mn} y$

# Encoding arithmetic functions

- **Multiplication**: $mult(m, n) = mn$
- **mult** $= \lambda pq\, f \cdot p(q\, f)$
- For all $m \geq 0$, $(\mathbf{n}\, f)^m y \xrightarrow{*}_\beta f^{mn} y$
  - $(\mathbf{n}\, f)\, {}^0 y = y = f^{0 \cdot n} y$

# Encoding arithmetic functions

- **Multiplication**: $mult(m, n) = mn$
- **mult** $= \lambda pq\, f \cdot p(q\, f)$
- For all $m \geq 0$, $(\mathbf{n}\, f)^m y \xrightarrow{*}_\beta f^{mn} y$
  - $(\mathbf{n}\, f)^0 y = y = f^{0 \cdot n} y$
  - $$\begin{aligned} (\mathbf{n}\, f)^{m+1} y &= (\mathbf{n}\, f)((\mathbf{n}\, f)^m y) \\ &\xrightarrow{*}_\beta \mathbf{n}\, f(f^{mn} y) \\ &\xrightarrow{*}_\beta f^n(f^{mn} y) = f^{mn+n} y = f^{(m+1)n} y \end{aligned}$$

# Encoding arithmetic functions

- **Multiplication**: $mult(m, n) = mn$
- $\mathbf{mult} = \lambda pq\, f \cdot p(q\, f)$
- For all $m \geq 0$, $(\mathbf{n}\, f)^m y \xrightarrow{*}_\beta f^{mn} y$
  - $(\mathbf{n}\, f)^0 y = y = f^{0 \cdot n} y$
  - $$\begin{aligned}(\mathbf{n}\, f)^{m+1} y &= (\mathbf{n}\, f)((\mathbf{n}\, f)^m y) \\ &\xrightarrow{*}_\beta \mathbf{n}\, f(f^{mn} y) \\ &\xrightarrow{*}_\beta f^n(f^{mn} y) = f^{mn+n} y = f^{(m+1)n} y\end{aligned}$$
- For all $m$, $n$ and $o$, if $mn = o$ then $\mathbf{mult\, m\, n} \xrightarrow{*}_\beta \mathbf{o}$

# Encoding arithmetic functions

- **Multiplication**: $mult(m, n) = mn$
- $\mathbf{mult} = \lambda pq\, f \cdot p(q\, f)$
- For all $m \geq 0$, $(\mathbf{n}\, f)^m y \xrightarrow{*}_\beta f^{mn} y$
  - $(\mathbf{n}\, f)^0 y = y = f^{0 \cdot n} y$
  - $$\begin{aligned}(\mathbf{n}\, f)^{m+1} y &= (\mathbf{n}\, f)((\mathbf{n}\, f)^m y) \\ &\xrightarrow{*}_\beta \mathbf{n}\, f(f^{mn} y) \\ &\xrightarrow{*}_\beta f^n(f^{mn} y) = f^{mn+n} y = f^{(m+1)n} y\end{aligned}$$
- For all $m$, $n$ and $o$, if $mn = o$ then $\mathbf{mult\, m\, n} \xrightarrow{*}_\beta \mathbf{o}$
  - $$\begin{aligned}(\lambda pq\, f \cdot p(q\, f))\, \mathbf{m\, n} &\xrightarrow{*}_\beta \lambda f \cdot \mathbf{m}(\mathbf{n}\, f) \\ &= \lambda f \cdot (\lambda gy \cdot g^m y)(\mathbf{n}\, f) \\ &\xrightarrow{*}_\beta \lambda f \cdot (\lambda y \cdot (\mathbf{n}\, f)^m y) \\ &\xrightarrow{*}_\beta \lambda fy \cdot f^{mn} y = \lambda fy \cdot f^o y = \mathbf{o}\end{aligned}$$

# Encoding arithmetic functions

- **Exponentiation**: $exp(m, n) = n^m$

# Encoding arithmetic functions

- **Exponentiation**: $exp(m, n) = n^m$
- **exp** $= \lambda pq.pq$

# Encoding arithmetic functions

- **Exponentiation**: $exp(m, n) = n^m$
- **exp** $= \lambda pq.pq$
- For all $m \geq 1$, $n \geq 0$ and $o$, if $n^m = o$ then **exp m n** $\xrightarrow{*}_\beta$ **o**

# Encoding arithmetic functions

- **Exponentiation**: $exp(m, n) = n^m$
- **exp** $= \lambda pq.pq$
- For all $m \geq 1$, $n \geq 0$ and $o$, if $n^m = o$ then **exp m n** $\xrightarrow{*}_\beta$ **o**
  - **Proof**: Exercise!

# Computability

- Church numerals encode $n \in \mathbb{N}$

# Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \to \mathbb{N}$?

# Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \to \mathbb{N}$?
  - Let **f** be the encoding of $f$

# Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \to \mathbb{N}$?
  - Let $\mathbf{f}$ be the encoding of $f$
  - We want $\mathbf{f}\,\mathbf{n}_1 \cdots \mathbf{n}_k \xrightarrow{*}_\beta \mathbf{m}$ iff $f(n_1, \cdots, n_k) = m$

# Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \to \mathbb{N}$?
  - Let $\mathbf{f}$ be the encoding of $f$
  - We want $\mathbf{f}\,\mathbf{n}_1 \cdots \mathbf{n}_k \xrightarrow{*}_\beta \mathbf{m}$ iff $f(n_1, \cdots, n_k) = m$
  - If $f(n_1, \ldots, n_k)$ is undefined, then $\neg\left(\mathbf{f}\,\mathbf{n}_1 \cdots \mathbf{n}_k \xrightarrow{*}_\beta \mathbf{m}\right)$ for any $m$

# Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \to \mathbb{N}$?
  - Let $\mathbf{f}$ be the encoding of $f$
  - We want $\mathbf{f}\,\mathbf{n}_1 \cdots \mathbf{n}_k \xrightarrow{*}_\beta \mathbf{m}$ iff $f(n_1, \cdots, n_k) = m$
  - If $f(n_1, \ldots, n_k)$ is undefined, then $\neg\left(\mathbf{f}\,\mathbf{n}_1 \cdots \mathbf{n}_k \xrightarrow{*}_\beta \mathbf{m}\right)$ for any $m$
- We need a syntax for computable functions – **Next class!**