Lambda calculus

Madhavan Mukund, **S P Suresh**

Programming Language Concepts
Lecture 16, 13 March 2025

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

- 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 $\overset{*}{\longrightarrow}_\beta$

Encoding arithmetic

- In set theory, use nesting to encode numbers

Encoding arithmetic

- In set theory, use nesting to encode numbers
  - Encoding of n: $\langle\!\langle n \rangle\!\rangle$

Encoding arithmetic

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

Encoding arithmetic

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

Encoding arithmetic

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

Encoding arithmetic

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

Encoding arithmetic

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

Encoding arithmetic

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

Encoding arithmetic

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

Church numerals

- $\ll n \gg = \lambda f x \cdot f^n x$

Church numerals

- $\ll n \gg = \lambda f x \cdot f^n x$
  - $f^0 x = x$

Church numerals

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

Church numerals

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

Church numerals

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

Church numerals

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

Church numerals

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

Church numerals

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

Church numerals

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

Church numerals

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

Church numerals

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

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, **succ** $\ll n \gg \xrightarrow{*}_{\beta} \ll n + 1 \gg$

Encoding arithmetic functions

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

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

Encoding arithmetic functions

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

Encoding arithmetic functions

- **Addition**: $plus(m, n) = m + n$
- **plus** $= \lambda pqfx \cdot pf\,(qfx)$

Encoding arithmetic functions

- **Addition**: $plus(m, n) = m + n$
- **plus** $= \lambda pqfx \cdot pf\,(qfx)$
- For all m, n, **plus** $\ll m \gg \ll n \gg \xrightarrow{*}_{\beta} \ll m + n \gg$

Encoding arithmetic functions

- **Addition**: $plus(m, n) = m + n$
- **plus** $= \lambda pqfx \cdot pf\,(qfx)$
- For all m, n, **plus** $\ll m \gg \ll n \gg \xrightarrow{*}_\beta \ll m + n \gg$
  - **plus** $\ll m \gg \ll n \gg$

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

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 $m \geqslant 0$, $(\!|\, n \,|\!)\, f\,)^m y \xrightarrow{*}_\beta f^{\,mn} y$

Encoding arithmetic functions

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

Encoding arithmetic functions

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

Encoding arithmetic functions

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

Encoding arithmetic functions

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

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 $m \geqslant 1, n \geqslant 0$, **exp** $\ll m \gg \ll n \gg \xrightarrow{*}_{\beta} \ll n^m \gg$

Encoding arithmetic functions

- **Exponentiation**: $exp(m, n) = n^m$
- **exp** $= \lambda pq.pq$
- For all $m \geqslant 1, n \geqslant 0$, **exp** $\ll m \gg \ll n \gg \xrightarrow{*}_\beta \ll n^m \gg$
  - **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 **f** be the encoding of f
- We want $\mathbf{f}\ «n_1» \cdots «n_k» \xrightarrow{*}_\beta «m»$ 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 **f** be the encoding of f
- We want $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle$ iff $f(n_1, \cdots, n_k) = m$
  1. If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle f(n_1, \cdots, n_k) \rangle\!\rangle$

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
- We want $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle$ iff $f(n_1, \cdots, n_k) = m$
  1. If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle f(n_1, \cdots, n_k) \rangle\!\rangle$
  2. If $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle$ and $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle p \rangle\!\rangle$, then $m = p$

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 $\mathbf{f} \ll n_1 \gg \cdots \ll n_k \gg \xrightarrow{*}_\beta \ll m \gg$ iff $f(n_1, \cdots, n_k) = m$
  1. If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f} \ll n_1 \gg \cdots \ll n_k \gg \xrightarrow{*}_\beta \ll f(n_1, \cdots, n_k) \gg$
  2. If $\mathbf{f} \ll n_1 \gg \cdots \ll n_k \gg \xrightarrow{*}_\beta \ll m \gg$ and $\mathbf{f} \ll n_1 \gg \cdots \ll n_k \gg \xrightarrow{*}_\beta \ll p \gg$, then $m = p$
  3. If $f(n_1, \ldots, n_k)$ is undefined, then $\neg \left(\mathbf{f} \ll n_1 \gg \cdots \ll n_k \gg \xrightarrow{*}_\beta \ll m \gg \right)$ for any m

Computability

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