Lambda calculus

Functional models of computation

Ivan Trepakov

NSU Sys.Pro

History

- 1928 — Hilbert's *Entscheidungsproblem* [1]
  - Is there an *algorithm* for deciding whether a proposition in first-order logic is true or false?
- Replacement for set theory as foundation of mathematics
  - 1930 — Combinatory logic (*Curry, Schönfinkel*)
  - 1932 — λ-calculus (*Church*)
  - 1935 — Kleene–Rosser paradox
- Effective computability
  - 1935 — Untyped λ-calculus (*Church, Kleene, Rosser*)
  - 1936 — Turing machine
  - 1936 — Church–Turing thesis
- 1936 — Undecidability of first-order logic
  - Halting problem of Turing machine
  - Equivalence of λ-terms

[1] German for "decision problem"

Lambda calculus

History

- 1928 — Hilbert's *Entscheidungsproblem* [1]
  - Is there an *algorithm* for deciding whether a proposition in first-order logic is true or false?
- Replacement for set theory as foundation of mathematics
  - 1930 — Combinatory logic (*Curry, Schönfinkel*)
  - 1932 — λ-calculus (*Church*)
  - 1935 — Kleene–Rosser paradox
- Effective computability
  - 1935 — Untyped λ-calculus (*Church, Kleene, Rosser*)
  - 1936 — Turing machine
  - 1936 — Church–Turing thesis
- 1936 — Undecidability of first-order logic
  - Halting problem of Turing machine
  - Equivalence of λ-terms

David Hilbert

- Haskell Curry
- Wilhelm Ackermann
- John von Neumann
- Ernst Zermelo
- ...

Alonzo Church

- Stephen Cole Kleene
- J. Barkley Rosser
- Alan Turing
- Dana Scott
- Michael O. Rabin
- ...

[1] German for "decision problem"

Untyped lambda calculus

Syntax

$$term ::= \underbrace{var}_{\text{Variable}} \mid \underbrace{(term\ term)}_{\text{Application}} \mid \underbrace{(\lambda var.\, term)}_{\text{Abstraction}}$$

Examples

$$\lambda x.\, x \qquad (\lambda x.\, xx)(\lambda y.\, yy) \qquad \lambda f.\, \lambda x.\, f(f x)$$

Conventions

- Application is left associative
  $abc = (ab)c$
- Abstraction is right associative
  $\lambda x.\, \lambda y.\, x = \lambda x.\, (\lambda y.\, x)$
- Consecutive abstractions can be combined
  $\lambda x.\, \lambda y.\, x = \lambda xy.\, x$

Syntax

$$term ::= \underbrace{var}_{\text{Variable}} \mid \underbrace{(term\ term)}_{\text{Application}} \mid \underbrace{(\lambda var.\, term)}_{\text{Abstraction}}$$

Examples

$\lambda x.\, x$ $\quad (\lambda x.\, xx)(\lambda y.\, yy)$ $\quad \lambda f.\, \lambda x.\, f(f x)$

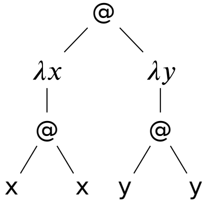
Conventions

- Application is left associative
  $abc = (ab)c$
- Abstraction is right associative
  $\lambda x.\, \lambda y.\, x = \lambda x.\, (\lambda y.\, x)$
- Consecutive abstractions can be combined
  $\lambda x.\, \lambda y.\, x = \lambda xy.\, x$

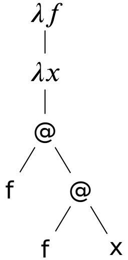
Tree representation

$\lambda x.\, x$

$(\lambda x.\, xx)(\lambda y.\, yy)$



$\lambda f.\, \lambda x.\, f(f x)$



Free and bound variables

Free variables $FV(t)$

Variable: $FV(x) = \{x\}$

Application: $FV(M\,N) = FV(M) \cup FV(N)$

Abstraction: $FV(\lambda x.\, M) = FV(M) \setminus \{x\}$

Bound variables $BV(t)$

Variable: $BV(x) = \varnothing$

Application: $BV(M\,N) = BV(M) \cup BV(N)$

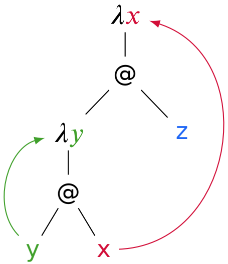
Abstraction: $BV(\lambda x.\, M) = BV(M) \cup \{x\}$

Closed terms

Term t is called closed or combinator if $FV(t) = \varnothing$

Example

$\lambda x.\,(\lambda y.\, y x) z$



Substitution

Substitution $t_{[v:=S]}$

$$x_{[v:=S]} = \begin{cases} S & v = x \\ x & v \neq x \end{cases}$$

$$(M\,N)_{[v:=S]} = (M_{[v:=S]}\ N_{[v:=S]})$$

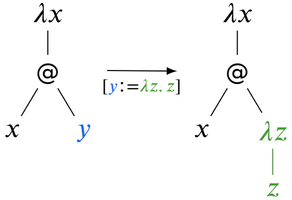
$$(\lambda x.\, M)_{[v:=S]} = \begin{cases} \lambda x.\, M & v = x \\ \lambda x.\, M_{[v:=S]} & v \neq x \end{cases}$$

Safe substitution

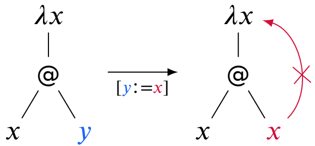
Substitution $t_{[v:=S]}$ is safe if $BV(t) \cap FV(S) = \varnothing$

Example

$$(\lambda x.\, x\,y)_{[y:=\lambda z.\, z]} = \lambda x.\, x(\lambda z.\, z)$$



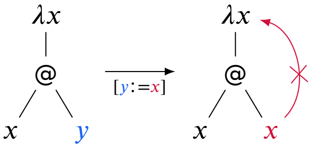
$$(\lambda x.\, x\,y)_{[y:=x]} = \lambda x.\, x\,x$$



Renaming

Problem

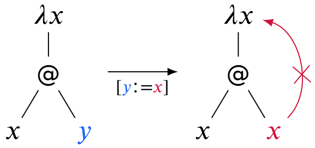
$$\lambda x.\, x y_{[y := x]} = \lambda x.\, x x$$



Renaming

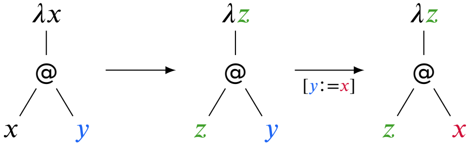
Problem

$$\lambda x.\, x y_{[y:=x]} = \lambda x.\, x x$$



Solution

$$(\lambda x.\, x y)_{[y:=x]} \equiv (\lambda z.\, z y)_{[y:=x]} = \lambda z.\, z x$$



α-equivalence

$$\lambda x.\, M \underset{\alpha}{\equiv} \lambda y.\, M_{[x:=y]} \quad \text{if } x \notin FV(M)$$

$$\lambda x.\, M \underset{\alpha}{\equiv} \lambda x.\, N \quad \text{if } M \underset{\alpha}{\equiv} N$$

$$M\,P \underset{\alpha}{\equiv} N\,P \quad \text{if } M \underset{\alpha}{\equiv} N$$

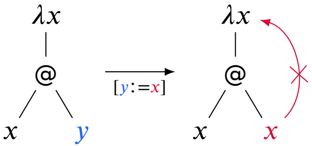
$$P\,M \underset{\alpha}{\equiv} P\,N \quad \text{if } M \underset{\alpha}{\equiv} N$$

Conventions

- λ-terms are considered identical up to α-equivalence
- Appropriate renaming happens implicitly if required during substitution

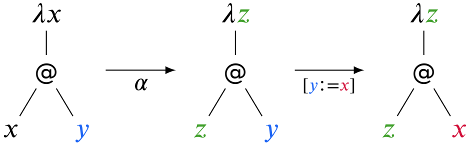
Problem

$$\lambda x.\, x y_{[y:=x]} = \lambda x.\, x x$$



Solution

$$(\lambda x.\, x y)_{[y:=x]} \underset{\alpha}{\equiv} (\lambda z.\, z y)_{[y:=x]} = \lambda z.\, z x$$



Evaluation

Definitions

- Subterm of form $(\lambda x.\, M)N$ is called β-redex
- Redex $(\lambda x.\, M)N$ can be reduced to $M_{[x:=N]}$
- $M \to_\beta M'$ denotes single β-reduction
- $M \twoheadrightarrow_\beta M'$ denotes several β-reductions
- $M \leftrightarrow_\beta M'$ denotes β-conversion as smallest equivalence relation containing $\to_\beta$

β-reduction

$$(\lambda x.\, M)N \to_\beta M_{[x:=N]}$$

$$\lambda x.\, M \to_\beta \lambda x.\, N \qquad \text{if } M \to_\beta N$$

$$M\,P \to_\beta N\,P \qquad \text{if } M \to_\beta N$$

$$P\,M \to_\beta P\,N \qquad \text{if } M \to_\beta N$$

Definitions

- Subterm of form $(\lambda x.\, M)N$ is called β-redex
- Redex $(\lambda x.\, M)N$ can be reduced to $M_{[x:=N]}$
- $M \to_\beta M'$ denotes single β-reduction
- $M \twoheadrightarrow_\beta M'$ denotes several β-reductions
- $M \leftrightarrow_\beta M'$ denotes β-conversion as smallest equivalence relation containing $\to_\beta$

β-reduction

$$(\lambda x.\, M)N \to_\beta M_{[x:=N]}$$

$$\lambda x.\, M \to_\beta \lambda x.\, N \qquad \text{if } M \to_\beta N$$

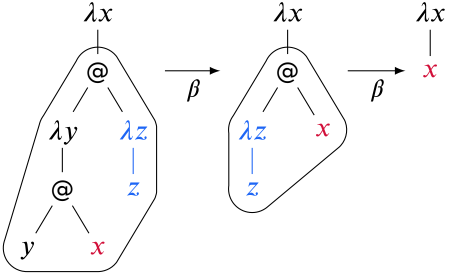
$$M\,P \to_\beta N\,P \qquad \text{if } M \to_\beta N$$

$$P\,M \to_\beta P\,N \qquad \text{if } M \to_\beta N$$

Example

$$(\lambda x.\, \underline{(\lambda y.\, y x)(\lambda z.\, z)}) \to_\beta$$

$$(\lambda x.\, \underline{(\lambda z.\, z) x}) \to_\beta \lambda x.\, x$$

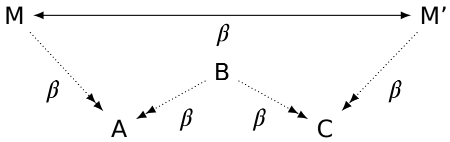


Evaluation

Definitions

- Subterm of form $(\lambda x.\, M)N$ is called β-redex
- Redex $(\lambda x.\, M)N$ can be reduced to $M_{[x:=N]}$
- $M \to_\beta M'$ denotes single β-reduction
- $M \twoheadrightarrow_\beta M'$ denotes several β-reductions
- $M \leftrightarrow_\beta M'$ denotes β-conversion as smallest equivalence relation containing $\to_\beta$

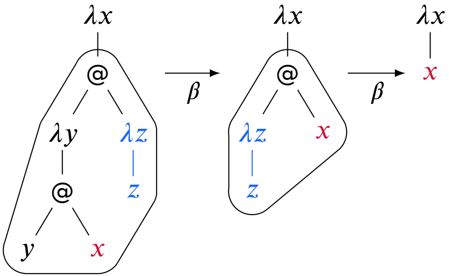
β-conversion



Example

$$(\lambda x.\, \underline{(\lambda y.\, yx)(\lambda z.\, z)}) \to_\beta$$

$$(\lambda x.\, \underline{(\lambda z.\, z)x}) \to_\beta \lambda x.\, x$$



Normal order

- Term without any redex is in β-normal form
- Reduction sequence that always reduces leftmost outermost redex is called normal order reduction

Example

$$\underline{(\lambda z.\, \underline{((\lambda x.\, xz)f)}\ \underline{((\lambda y.\, yz)g)})a} \to_\beta$$

$$\underline{((\lambda x.\, xa)f)}\ \underline{((\lambda y.\, ya)g)} \to_\beta$$

$$(fa)\ \underline{((\lambda y.\, ya)g)} \to_\beta (fa)\ (ga)$$

Reduction order

Normal order

- Term without any redex is in β-normal form
- Reduction sequence that always reduces leftmost outermost redex is called normal order reduction

Example

$$\underline{(\lambda z.\, \underline{((\lambda x.\, xz) f)}\ \underline{((\lambda y.\, yz) g)}) a} \to_\beta$$

$$\underline{((\lambda x.\, xa) f)}\ \underline{((\lambda y.\, ya) g)} \to_\beta$$

$$(fa)\ \underline{((\lambda y.\, ya) g)} \to_\beta (fa)\ (ga)$$

Totality

Reduction order

Normal order

- Term without any redex is in β-normal form
- Reduction sequence that always reduces leftmost outermost redex is called normal order reduction

Example

$$\underline{(\lambda z.\, \underline{((\lambda x.\, xz)f)}\ \underline{((\lambda y.\, yz)g})) a} \to_\beta$$

$$\underline{((\lambda x.\, xa)f)}\ \underline{((\lambda y.\, ya)g)} \to_\beta$$

$$(fa)\ \underline{((\lambda y.\, ya)g)} \to_\beta (fa)\ (ga)$$

Totality

$$\Omega = \underline{(\lambda x.\, xx)(\lambda x.\, xx)} \to_\beta$$

$$(xx)_{[x:=(\lambda x.\, xx)]} \to_\beta$$

$$\underline{(\lambda x.\, xx)(\lambda x.\, xx)} \to_\beta \dots$$

Reduction order

Normal order

- Term without any redex is in β-normal form
- Reduction sequence that always reduces leftmost outermost redex is called normal order reduction

Example

$$(\lambda z.\, \underline{((\lambda x.\, xz) f)}\ \underline{((\lambda y.\, yz) g)})a \to_\beta$$

$$\underline{((\lambda x.\, xa) f)}\ \underline{((\lambda y.\, ya) g)} \to_\beta$$

$$(fa)\ \underline{((\lambda y.\, ya) g)} \to_\beta (fa)\ (ga)$$

Totality

$$\Omega = \underline{(\lambda x.\, xx)(\lambda x.\, xx)} \to_\beta$$

$$(xx)_{[x := (\lambda x.\, xx)]} \to_\beta$$

$$\underline{(\lambda x.\, xx)(\lambda x.\, xx)} \to_\beta \ldots$$

Ω has no β-normal form

Reduction order

Normal order

- Term without any redex is in β-normal form
- Reduction sequence that always reduces leftmost outermost redex is called normal order reduction

Example

$$\underline{(\lambda z.\, \underline{((\lambda x.\, xz)f)}\ \underline{((\lambda y.\, yz)g)})a} \rightarrow_\beta$$

$$\underline{((\lambda x.\, xa)f)}\ \underline{((\lambda y.\, ya)g)} \rightarrow_\beta$$

$$(fa)\ \underline{((\lambda y.\, ya)g)} \rightarrow_\beta (fa)\ (ga)$$

Totality

$$\Omega = \underline{(\lambda x.\, xx)(\lambda x.\, xx)} \rightarrow_\beta$$

$$(xx)_{[x := (\lambda x.\, xx)]} \rightarrow_\beta$$

$$\underline{(\lambda x.\, xx)(\lambda x.\, xx)} \rightarrow_\beta \ldots$$

Ω has no β-normal form

Strictness

Reduction order

Normal order

- Term without any redex is in β-normal form
- Reduction sequence that always reduces leftmost outermost redex is called normal order reduction

Example

$$\underline{(\lambda z.\, \underline{((\lambda x.\, xz)f)}\ \underline{((\lambda y.\, yz)g)})a} \to_\beta$$

$$\underline{((\lambda x.\, xa)f)}\ \underline{((\lambda y.\, ya)g)} \to_\beta$$

$$(fa)\ \underline{((\lambda y.\, ya)g)} \to_\beta (fa)\ (ga)$$

Totality

$$\Omega = \underline{(\lambda x.\, xx)(\lambda x.\, xx)} \to_\beta$$

$$(xx)_{[x:=(\lambda x.\, xx)]} \to_\beta$$

$$\underline{(\lambda x.\, xx)(\lambda x.\, xx)} \to_\beta \ldots$$

Ω has no β-normal form

Strictness

$$\underline{(\lambda x.\, y)\Omega} \to_\beta y$$

Reduction order

Normal order

- Term without any redex is in β-normal form
- Reduction sequence that always reduces leftmost outermost redex is called normal order reduction

Example

$$\underline{(\lambda z.\, \underline{((\lambda x.\, xz) f)}\ \underline{((\lambda y.\, yz) g)}) a} \to_\beta$$

$$\underline{((\lambda x.\, xa) f)}\ \underline{((\lambda y.\, ya) g)} \to_\beta$$

$$(fa)\ \underline{((\lambda y.\, ya) g)} \to_\beta (fa)\ (ga)$$

Totality

$$\Omega = \underline{(\lambda x.\, xx)(\lambda x.\, xx)} \to_\beta$$

$$(xx)_{[x := (\lambda x.\, xx)]} \to_\beta$$

$$\underline{(\lambda x.\, xx)(\lambda x.\, xx)} \to_\beta \ldots$$

Ω has no β-normal form

Strictness

$$\underline{(\lambda x.\, y)\Omega} \to_\beta y$$

Normal order reduction models non-strict or lazy evaluation

Reduction order

Reduction order

Church–Rosser theorem

If $M \leftrightarrow_\beta M'$ then there exists N such that $M \twoheadrightarrow_\beta N$ and $M' \twoheadrightarrow_\beta N$

Church–Rosser

$$\begin{tikzcd}
M \arrow[rr, leftrightarrow, "\beta"'] \arrow[dr, dotted, two heads, "\beta"'] & & M' \arrow[dl, dotted, two heads, "\beta"] \\
& N &
\end{tikzcd}$$

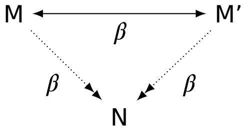
Church–Rosser theorem

If $M \leftrightarrow_\beta M'$ then there exists N such that $M \twoheadrightarrow_\beta N$ and $M' \twoheadrightarrow_\beta N$

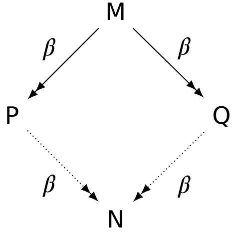
Confluence property

If $M \twoheadrightarrow_\beta P$ and $M \twoheadrightarrow_\beta Q$ then there exists N such that $P \twoheadrightarrow_\beta N$ and $Q \twoheadrightarrow_\beta N$

Church–Rosser



Confluence



Church–Rosser theorem

If $M \leftrightarrow_\beta M'$ then there exists N such that $M \twoheadrightarrow_\beta N$ and $M' \twoheadrightarrow_\beta N$

Confluence property

If $M \twoheadrightarrow_\beta P$ and $M \twoheadrightarrow_\beta Q$ then there exists N such that $P \twoheadrightarrow_\beta N$ and $Q \twoheadrightarrow_\beta N$

Corollary

β-normal form is unique up to α-equivalence

Church–Rosser

$$\begin{tikzcd}
M \arrow[rr, leftrightarrow, "\beta"'] \arrow[dr, dotted, "\beta"'] & & M' \arrow[dl, dotted, "\beta"] \\
& N &
\end{tikzcd}$$

Confluence

$$\begin{tikzcd}
& M \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
P \arrow[dr, dotted, "\beta"'] & & Q \arrow[dl, dotted, "\beta"] \\
& N &
\end{tikzcd}$$

Reduction order

Church–Rosser theorem

If $M \leftrightarrow_\beta M'$ then there exists N such that $M \twoheadrightarrow_\beta N$ and $M' \twoheadrightarrow_\beta N$

Confluence property

If $M \twoheadrightarrow_\beta P$ and $M \twoheadrightarrow_\beta Q$ then there exists N such that $P \twoheadrightarrow_\beta N$ and $Q \twoheadrightarrow_\beta N$

Corollary

β-normal form is unique up to α-equivalence

Normal order theorem

If $M \twoheadrightarrow_\beta N$ and N is in β-normal form then there exists normal order reduction sequence from M to N

Church–Rosser

$$\begin{tikzcd}
M \arrow[rr, leftrightarrow, "\beta"'] \arrow[dr, dotted, "\beta"'] & & M' \arrow[dl, dotted, "\beta"] \\
& N &
\end{tikzcd}$$

Confluence

$$\begin{tikzcd}
& M \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
P \arrow[dr, dotted, "\beta"'] & & Q \arrow[dl, dotted, "\beta"] \\
& N &
\end{tikzcd}$$

Recursion

Fixed-point combinator

$$Y\,f \leftrightarrow_\beta f\,(Y\,f)$$

Fixed-point combinator

$$Y f \leftrightarrow_\beta f(Y f)$$

Curry's Y-combinator

$$Y = \lambda f.\,(\lambda x.\, f(xx))\,(\lambda x.\, f(xx))$$

Fixed-point combinator

$$Y\,f \leftrightarrow_\beta f\,(Y\,f)$$

Curry's Y-combinator

$$Y = \lambda f.\,(\lambda x.\, f\,(x x))\ (\lambda x.\, f\,(x x))$$

Note: neither $Y\,f \rightarrow_\beta f\,(Y\,f)$ nor $f\,(Y\,f) \rightarrow_\beta Y\,f$

Fixed-point combinator

$$Y\,f \leftrightarrow_\beta f\,(Y\,f)$$

Curry's Y-combinator

$$Y = \lambda f.\,(\lambda x.\,f\,(xx))\,(\lambda x.\,f\,(xx))$$

Note: neither $Y\,f \rightarrow_\beta f\,(Y\,f)$ nor $f\,(Y\,f) \rightarrow_\beta Y\,f$

Turing's Θ-combinator

$$\Theta = (\lambda xy.\,x(xxy))\,(\lambda xy.\,x(xxy))$$

Fixed-point combinator

$$Y\,f \leftrightarrow_\beta f\,(Y\,f)$$

Curry's Y-combinator

$$Y = \lambda f.\,(\lambda x.\, f\,(xx))\;(\lambda x.\, f\,(xx))$$

Note: neither $Y\,f \to_\beta f\,(Y\,f)$ nor $f\,(Y\,f) \to_\beta Y\,f$

Turing's Θ-combinator

$$\Theta = (\lambda xy.\, x(xxy))\;(\lambda xy.\, x(xxy))$$

Note: $\Theta f \to_\beta f\,(\Theta f)$

Q&A