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"

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"

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
- ...

Syntax

Grammar

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

Examples

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

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

Grammar

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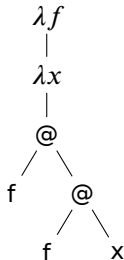
Examples

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

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
  $\lambda x.\, \lambda y.\, x = \lambda xy.\, x$

Tree representation

$\lambda x.\, x$

$(\lambda x.\, xx)(\lambda y.\, yy)$ $\lambda f.\, \lambda x.\, f(f x)$
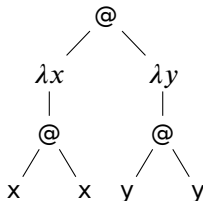
Free variables $FV(t)$

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

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

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

Bound variables $BV(t)$

Variable: $BV(x) = \varnothing$
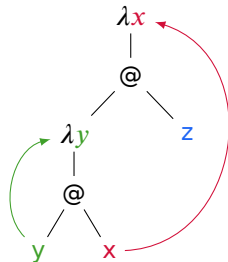
Application: $BV(MN) = BV(M) \cup BV(N)$

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

Closed terms

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

Example

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

Substitution

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

$$x_{[v:=S]} = \begin{cases} S & v = x \\ x & v \neq x \end{cases}$$
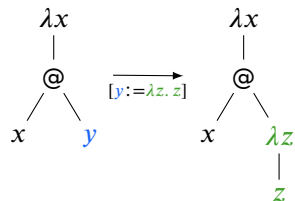
$$(MN)_{[v:=S]} = (M_{[v:=S]} \; N_{[v:=S]})$$

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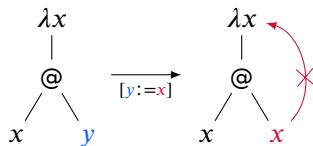
Safe substitution

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

Example

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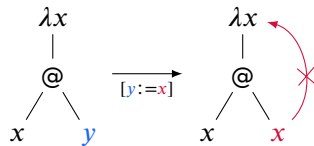

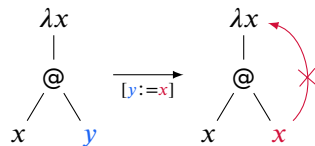
$$(\lambda x.\, xy)_{[y:=x]} = \lambda x.\, xx$$

Problem

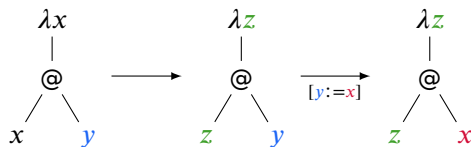$$\lambda x.\, xy_{[y:=x]} = \lambda x.\, xx$$

Problem

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

Solution

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

Renaming

α-equivalence

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

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

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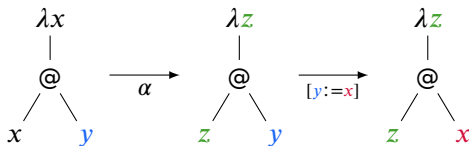
$$P\,M \underset{\alpha}{\equiv} P\,N \qquad\quad \text{if } M \underset{\alpha}{\equiv} N$$

Conventions

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

Problem

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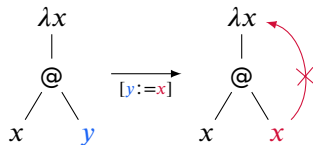
Solution

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

Definitions

- Subterm of form $(\lambda x.\, M)N$ is called β-redex
- Redex $(\lambda x.\, M)N$ can be reduced to $M_{[x:=N]}$
- Reduction of single redex in term M is called
  β-reduction and denoted as $M \to_\beta M'$
- β-reduction in multiple steps is denoted as
  $M \twoheadrightarrow_\beta M'$

β-reduction

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

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

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

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

Evaluation

Definitions

- Subterm of form $(\lambda x.\, M)N$ is called β-redex
- Redex $(\lambda x.\, M)N$ can be reduced to $M_{[x:=N]}$
- Reduction of single redex in term M is called β-reduction and denoted as $M \rightarrow_\beta M'$
- β-reduction in multiple steps is denoted as $M \twoheadrightarrow_\beta M'$

β-reduction

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

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

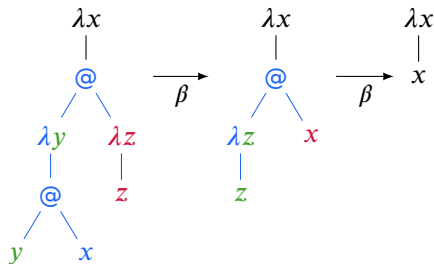$$M\, P \rightarrow_\beta N\, P \qquad \text{if } M \rightarrow_\beta N$$

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

Example

$$(\lambda x.\, (\lambda y.\, yx)(\lambda z.\, z)) \;\rightarrow_\beta$$

$$(\lambda x.\, (\lambda z.\, z)x) \;\rightarrow_\beta$$

$$\lambda x.\, x$$

η-conversion

$$(\lambda x.\, M\, x) \underset{\eta}{\longleftrightarrow} M \text{ if } x \notin FV(M)$$

First Church-Rosser theorem

Second Church-Rosser theorem

Normal order reduction

Fixed-point combinator

Curry's Y-combinator

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

Turing's Θ-combinator

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

Undecidability

Church numerals

Relation to folds

Algebraic data types

Predecessor

Q&A