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

[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

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

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

Grammar

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

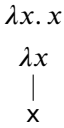
Examples

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

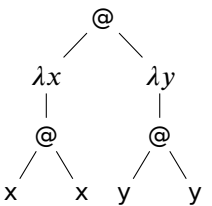
Conventions

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

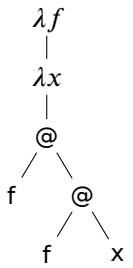
Tree representation



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



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



Free and bound variables

Free variables $FV(t)$

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

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

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

Bound variables $BV(t)$

Variable: $BV(x) = \varnothing$

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

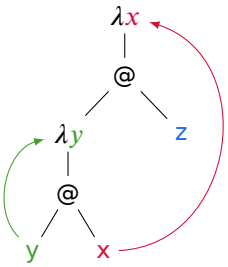
Abstraction: $BV(\lambda x.\, M) = BV(M) \cup \{x\}$

Closed terms

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

Example

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



Substitution

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

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

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

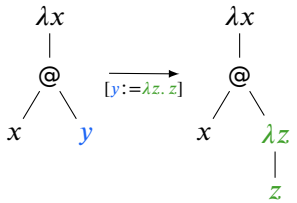
$$(\lambda x.\, M)_{[v:=S]} = \begin{cases} \lambda x.\, M & v = x \\ \lambda x.\, M_{[v:=S]} & v \neq x \end{cases}$$

Safe substitution

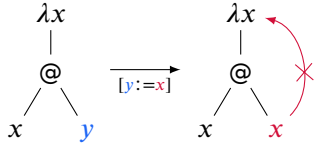
Substitution $t_{[v:=S]}$ is safe if $BV(t) \cap FV(S) = \emptyset$

Example

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



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



α-conversion

Substitution

α-equivalence

β-conversion

β-reduction

β-abstraction

η-conversion

Convertibility

Normal order reduction

First Church-Rosser theorem

Second Church-Rosser theorem

Normal order reduction

Recursion

Fixed-point combinator

Curry's Y-combinator

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

Turing's Θ-combinator

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

Church-Turing thesis

Undecidability

Church numerals

Relation to folds

Algebraic data types

Predecessor

Q&A