λ-calculus

Andrés Sicard-Ramírez

(Last modification: October 21, 2010)

References

- Textbook: J. R. Hindley and J. Seldin. *Lambda-calculus and Combinators. An Introduction.* Cambridge University Press, 2008.
- Henk Barendregt and Erik Barendsen. Introduction to Lambda Calculus. Revisited edition, Mar. 2000. Eprint: `www.cs.ru.nl/~henk/`, 2000.
- Henk Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. Elsevier, revised edition, 2004. Sixth impression.
- Lawrence C. Paulson. Foundations of functional programming. Eprint: `http://www.cl.cam.ac.uk/~lp15/`, 2000.

What is the λ-calculus?

Invented by Alonzo Church (around 1930s).

- The goal was to use it in the foundation of mathematics. Intended for studying functions and recursion.
- Computability model.
- Model of untyped functional programming languages.

Introduction

- λ-calculus is a collection of several formal systems
- λ-notation
  - Anonymous functions
  - Currying

Introduction

- λ-calculus is a collection of several formal systems
- λ-notation
  - Anonymous functions
  - Currying

λ-terms: (inductive definition)

Basis:

$$v \in V \Rightarrow v \in \lambda\text{-terms (atom)}$$
$$c \in C \Rightarrow c \in \lambda\text{-terms (atom)}$$

Inductive step:

$$M, N \in \lambda\text{-terms} \Rightarrow (MN) \in \lambda\text{-terms} \quad \text{(application)}$$
$$M \in \lambda\text{-terms}, x \in Vars \Rightarrow (\lambda x.M) \in \lambda\text{-terms (abstraction)}$$

where V/C: Set of variables/constants.

Introduction (cont.)

Conventions and syntactic sugar:

- $M \equiv N$ means the syntactic identity

Introduction (cont.)

Conventions and syntactic sugar:

- $M \equiv N$ means the syntactic identity
- Application associates to the left
  $M N_1 \ldots N_k$ means $(\ldots((M N_1) N_2) \ldots N_k)$

Introduction (cont.)

Conventions and syntactic sugar:

- $M \equiv N$ means the syntactic identity
- Application associates to the left
  $MN_1 \ldots N_k$ means $(...((MN_1)N_2)...N_k)$
- Application has higher precedence
  $\lambda x.PQ$ means $(\lambda x.(PQ))$

Introduction (cont.)

Conventions and syntactic sugar:

- $M \equiv N$ means the syntactic identity
- Application associates to the left
  $MN_1 \ldots N_k$ means $(...((MN_1)N_2)...N_k)$
- Application has higher precedence
  $\lambda x.PQ$ means $(\lambda x.(PQ))$
- $\lambda x_1 x_2 \ldots x_n.M$ means $(\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.M)\ldots)))$

Introduction (cont.)

Conventions and syntactic sugar:

- $M \equiv N$ means the syntactic identity
- Application associates to the left
  $MN_1 \ldots N_k$ means $(\ldots((MN_1)N_2)\ldots N_k)$
- Application has higher precedence
  $\lambda x.PQ$ means $(\lambda x.(PQ))$
- $\lambda x_1 x_2 \ldots x_n.M$ means $(\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.M)\ldots)))$

Example.

$(\lambda xyz.xz(yz))uvw \equiv ((((\lambda x.(\lambda y.(\lambda z.((xz)(yz)))))u)v)w)$.

Term-structure and substitution

Substitution ($[N/x]M$):

The result of substituting N for every free occurrence of x in M, and changing bound variables to avoid clashes.

Term-structure and substitution

Substitution ($[N/x]M$):

The result of substituting N for every free occurrence of x in M, and changing bound variables to avoid clashes.

$$
\begin{aligned}
[N/x]x &\equiv N && && (1)\\
[N/x]a &\equiv a && \text{for all atoms } a \not\equiv x && (2)\\
[N/x](PQ) &\equiv [N/x]P\ [N/x]Q
\end{aligned}
$$

Term-structure and substitution

Substitution ($[N/x]M$):

The result of substituting N for every free occurrence of x in M, and changing bound variables to avoid clashes.

$$
\begin{aligned}
[N/x]x &\equiv N && && (1)\\
[N/x]a &\equiv a && \text{for all atoms } a \not\equiv x && (2)\\
[N/x](PQ) &\equiv [N/x]P\ [N/x]Q && && (3)\\
[N/x](\lambda x.P) &\equiv \lambda x.P && && (4)\\
[N/x](\lambda y.P) &\equiv \lambda y.P && y \not\equiv x, x \notin FV(P) &&
\end{aligned}
$$

Term-structure and substitution

Substitution ($[N/x]M$):

The result of substituting N for every free occurrence of x in M, and changing bound variables to avoid clashes.

$$
\begin{aligned}
[N/x]x &\equiv N && && (1)\\
[N/x]a &\equiv a && \text{for all atoms } a \not\equiv x && (2)\\
[N/x](PQ) &\equiv [N/x]P\ [N/x]Q && && (3)\\
[N/x](\lambda x.P) &\equiv \lambda x.P && && (4)\\
[N/x](\lambda y.P) &\equiv \lambda y.P && y \not\equiv x, x \notin FV(P) && (5)\\
[N/x](\lambda y.P) &\equiv \lambda y.[N/x]P && y \not\equiv x, x \in FV(P), && (6)\\
& && y \notin FV(N) &&
\end{aligned}
$$

Term-structure and substitution

Substitution ($[N/x]M$):

The result of substituting N for every free occurrence of x in M, and changing bound variables to avoid clashes.

$$
\begin{aligned}
[N/x]x &\equiv N && && (1)\\
[N/x]a &\equiv a && \text{for all atoms } a \not\equiv x && (2)\\
[N/x](PQ) &\equiv [N/x]P\ [N/x]Q && && (3)\\
[N/x](\lambda x.P) &\equiv \lambda x.P && && (4)\\
[N/x](\lambda y.P) &\equiv \lambda y.P && y \not\equiv x, x \notin FV(P) && (5)\\
[N/x](\lambda y.P) &\equiv \lambda y.[N/x]P && y \not\equiv x, x \in FV(P), && (6)\\
& && y \notin FV(N) && \\
[N/x](\lambda y.P) &\equiv \lambda z.[N/x][z/y]P && y \not\equiv x, x \in FV(P), && (7)\\
& && y \in FV(N) &&
\end{aligned}
$$

where in the last equation, z is chosen to be a variable $\notin FV(NP)$.

Term-structure and substitution (cont.)

Example.

$[(\lambda y.vy)/x](y(\lambda v.xv)) \equiv y(\lambda z.(\lambda y.vy)z)$ (with $z \not\equiv v, y, x$).

Term-structure and substitution (cont.)

α-conversion or changed of bound variables: Replace $\lambda x.M$ by $\lambda y.[y/x]M$ ($y \notin FV(M)$).

α-congruence ($P \equiv_\alpha Q$):

P is changed to Q by a finite (perhaps empty) series of α-conversions.

Example. Whiteboard.

Term-structure and substitution (cont.)

α-conversion or changed of bound variables: Replace $\lambda x.M$ by $\lambda y.[y/x]M$ ($y \notin FV(M)$).

α-congruence ($P \equiv_\alpha Q$):

P is changed to Q by a finite (perhaps empty) series of α-conversions.

Example. Whiteboard.

Theorem. The relation $\equiv_\alpha$ is a equivalence relation.

β-reduction

β-contraction ($\cdot \rhd_{1\beta} \cdot$):

$(\lambda x.M)N$: β-redex $\qquad$ $[N/x]M$: contractum

$(\lambda x.M)N \rhd_{1\beta} [N/x]M$

$P \rhd_{1\beta} Q$: Replace an occurrence of $(\lambda x.M)N$ in P by $[N/x]M$.

Example. Whiteboard.

β-reduction

β-contraction $(\cdot \rhd_{1\beta} \cdot)$:

$(\lambda x.M)N$: β-redex $\qquad$ $[N/x]M$: contractum

$(\lambda x.M)N \rhd_{1\beta} [N/x]M$

$P \rhd_{1\beta} Q$: Replace an occurrence of $(\lambda x.M)N$ in P by $[N/x]M$.

Example. Whiteboard.

β-reduction $(P \rhd_\beta Q)$:

P is changed to Q by a finite (perhaps empty) series of β-contractions and α-conversions.

Example. $(\lambda x.(\lambda y.yx)z)v \rhd_\beta zv$.

β-reduction (cont.)

β-normal form: A term which contains no β-redex.

β-nf: The set of all β-normal forms.

Example. Whiteboard.

β-reduction (cont.)

Theorem (The Church-Rosser theorem for $\rhd_\beta$ (the diamond property)).

$$\frac{P \rhd_\beta M \quad P \rhd_\beta N}{\exists\, T. M \rhd_\beta T \wedge N \rhd_\beta T}$$

$$\begin{tikzcd}
 & P \arrow[dl] \arrow[dr] & \\
M \arrow[dr, dotted] & & N \arrow[dl, dotted] \\
 & \exists\, T &
\end{tikzcd}$$

β-reduction (cont.)

Theorem (The Church-Rosser theorem for $\triangleright_\beta$ (the diamond property)).

$$\frac{P \triangleright_\beta M \quad P \triangleright_\beta N}{\exists\, T. M \triangleright_\beta T \wedge N \triangleright_\beta T}$$

$$\begin{tikzcd}
 & P \arrow[dl] \arrow[dr] & \\
M \arrow[dr, dotted] & & N \arrow[dl, dotted] \\
 & \exists\, T &
\end{tikzcd}$$

Corollary. If P has a β-normal form, it is unique modulo $\equiv_\alpha$; that is, if P has β-normal forms M and N, then $M \equiv_\alpha N$.

Proof. Whiteboard. □

β-equality

β-equality or β-convertibility ($P =_\beta Q$):

Exist $P_0, \ldots, P_n$ such that

- $P_0 \equiv P$
- $P_n \equiv Q$
- $(\forall i \leq n-1)(P_i \rhd_{1\beta} P_{i+1} \quad \vee \quad P_{i+1} \rhd_{1\beta} P_i \quad \vee \quad P_i \equiv_\alpha P_{i+1})$

β-equality

β-equality or β-convertibility ($P =_\beta Q$):

Exist $P_0, \ldots, P_n$ such that

- $P_0 \equiv P$
- $P_n \equiv Q$
- $(\forall i \leq n-1)(P_i \triangleright_{1\beta} P_{i+1} \quad \vee \quad P_{i+1} \triangleright_{1\beta} P_i \quad \vee \quad P_i \equiv_\alpha P_{i+1})$

Theorem (Church-Rosser theorem for $=_\beta$).

$$\frac{P =_\beta Q}{\exists T.P \triangleright_\beta T \wedge Q \triangleright_\beta T}$$

Proof. Whiteboard. $\square$

β-equality (cont.)

Corollary. If $P, Q \in \beta\text{-}nf$ and $P =_\beta Q$, then $P \equiv_\alpha Q$.

Corollary. The relation $=_\beta$ is non-trivial (not all terms are β-convertible to each other).

Proof. Whiteboard. $\square$

What is the Combinatory Logic?

- Combinatory logic

 

Invented by Moses Schönfinkel (1920) and Haskell Curry (1927).

Intended for clarify the role of quantified variables.

- Idea: To do logic and mathematics without use bound variables.
- Combinators: Operators which manipulate expressions by cancellation, duplication, bracketing and permutation.

Introduction

Example (Informally). The commutative law for addition

$$\forall xy.x + y = y + x,$$

can be written as

$$A = \boldsymbol{C}A,$$

where Axy represents $x + y$ and $\boldsymbol{C}$ is a combinator with the property

$$\boldsymbol{C}fxy = fyx.$$

Introduction (cont.)

Example (Combinators).

$\boldsymbol{B}fgx = f(gx)$: A composition operator

$\boldsymbol{B}'fgx = g(fx)$: A reversed composition operator

$\boldsymbol{I}x = x$: The identity operator

$\boldsymbol{K}xy = x$: A projection operator

$\boldsymbol{S}fgx = fx(gx)$: A stronger composition operator

$\boldsymbol{W}fx = fxx$: A doubling operator

Introduction (cont.)

CL-terms: (inductive definition)

Basis:

$$v \in V \Rightarrow v \in CL\text{-terms}$$
$$c \in C \Rightarrow c \in CL\text{-terms}$$

Inductive step:

$$X, Y \in CL\text{-terms} \Rightarrow (XY) \in CL\text{-terms}$$

where

$$\begin{aligned} V &: \text{ Set of variables} \\ C = \{\boldsymbol{I}, \boldsymbol{K}, \boldsymbol{S}, \ldots\} &: \text{ Set of atomic constants} \end{aligned}$$

Introduction (cont.)

$FV(X)$: The set of variables occurring in X.

Atoms, basic combinators and combinator: A atom is a variable or atomic constant. The basic combinators are $\boldsymbol{I}$, $\boldsymbol{K}$ and $\boldsymbol{S}$. A combinator is a CL-term whose only atoms are basic combinators.

Introduction (cont.)

$FV(X)$: The set of variables occurring in X.

Atoms, basic combinators and combinator: A atom is a variable or atomic constant. The basic combinators are $\boldsymbol{I}$, $\boldsymbol{K}$ and $\boldsymbol{S}$. A combinator is a CL-term whose only atoms are basic combinators.

Substitution ($[U/x]Y$): The result of substituting U for every occurrence of x in Y:

$$\begin{aligned} [U/x]x &\equiv U \\ [U/x]a &\equiv a \qquad\qquad \text{for all atoms } a \not\equiv x \\ [U/x](VW) &\equiv ([U/x]V\ [U/x]W) \end{aligned}$$

Weak reduction

Weak redex: The CL-terms $\boldsymbol{I}X$, $\boldsymbol{K}XY$ and $\boldsymbol{S}XYZ$.

Weak reduction

Weak redex: The CL-terms $\boldsymbol{I}X$, $\boldsymbol{K}XY$ and $\boldsymbol{S}XYZ$.

Weak contraction ($U \rhd_{1w} V$):

Replace an occurrence of a weak redex in U using:

$$\begin{aligned} \boldsymbol{I}X &\text{ by } X, \\ \boldsymbol{K}XY &\text{ by } X, \\ \boldsymbol{S}XYZ &\text{ by } XZ(YZ). \end{aligned}$$

Weak reduction

Weak redex: The CL-terms $\boldsymbol{I}X$, $\boldsymbol{K}XY$ and $\boldsymbol{S}XYZ$.

Weak contraction ($U \rhd_{1w} V$):

Replace an occurrence of a weak redex in U using:

$$\begin{aligned} \boldsymbol{I}X &\text{ by } X, \\ \boldsymbol{K}XY &\text{ by } X, \\ \boldsymbol{S}XYZ &\text{ by } XZ(YZ). \end{aligned}$$

Weak reduction ($U \rhd_{w} V$):

U is changed to V by a finite (perhaps empty) series of weak contractions.

Weak reduction

Weak redex: The CL-terms $\boldsymbol{I}X$, $\boldsymbol{K}XY$ and $\boldsymbol{S}XYZ$.

Weak contraction ($U \rhd_{1w} V$):

Replace an occurrence of a weak redex in U using:

$$\begin{aligned} \boldsymbol{I}X &\text{ by } X, \\ \boldsymbol{K}XY &\text{ by } X, \\ \boldsymbol{S}XYZ &\text{ by } XZ(YZ). \end{aligned}$$

Weak reduction ($U \rhd_w V$):

U is changed to V by a finite (perhaps empty) series of weak contractions.

Weak normal form: A CL-term which contains no weak redex.

Weak reduction (cont.)

Example. (whiteboard)

- $\boldsymbol{B} \equiv \boldsymbol{S}(\boldsymbol{KS})\boldsymbol{K}$. Then $\boldsymbol{B}XYZ \rhd_w X(YZ)$.

Weak reduction (cont.)

Example. (whiteboard)

- $\boldsymbol{B} \equiv \boldsymbol{S}(\boldsymbol{K}\boldsymbol{S})\boldsymbol{K}$. Then $\boldsymbol{B}XYZ \triangleright_w X(YZ)$.
- $\boldsymbol{W} \equiv \boldsymbol{S}\boldsymbol{S}(\boldsymbol{K}\boldsymbol{I})$. Then $\boldsymbol{W}XY \triangleright_w XYY$.

Weak reduction (cont.)

Example. (whiteboard)

- $\boldsymbol{B} \equiv \boldsymbol{S}(\boldsymbol{KS})\boldsymbol{K}$. Then $\boldsymbol{B}XYZ \rhd_w X(YZ)$.
- $\boldsymbol{W} \equiv \boldsymbol{SS}(\boldsymbol{KI})$. Then $\boldsymbol{W}XY \rhd_w XYY$.
- $\boldsymbol{WWW} \rhd_w \boldsymbol{WWW} \rhd_w \cdots$

Weak reduction (cont.)

Example. (whiteboard)

- $\boldsymbol{B} \equiv \boldsymbol{S}(\boldsymbol{KS})\boldsymbol{K}$. Then $\boldsymbol{B}XYZ \triangleright_w X(YZ)$.
- $\boldsymbol{W} \equiv \boldsymbol{SS}(\boldsymbol{KI})$. Then $\boldsymbol{W}XY \triangleright_w XYY$.
- $\boldsymbol{WWW} \triangleright_w \boldsymbol{WWW} \triangleright_w \cdots$

Theorem (Church-Rosser theorem for $\triangleright_w$).

$$\frac{P \triangleright_w M \quad P \triangleright_w N}{\exists T. M \triangleright_w T \wedge N \triangleright_w T}$$

Weak reduction (cont.)

Example. (whiteboard)

- $\boldsymbol{B} \equiv \boldsymbol{S}(\boldsymbol{KS})\boldsymbol{K}$. Then $\boldsymbol{B}XYZ \triangleright_w X(YZ)$.
- $\boldsymbol{W} \equiv \boldsymbol{SS}(\boldsymbol{KI})$. Then $\boldsymbol{W}XY \triangleright_w XYY$.
- $\boldsymbol{WWW} \triangleright_w \boldsymbol{WWW} \triangleright_w \cdots$

Theorem (Church-Rosser theorem for $\triangleright_w$).

$$\frac{P \triangleright_w M \quad P \triangleright_w N}{\exists T. M \triangleright_w T \wedge N \triangleright_w T}$$

Corollary (Uniqueness of nf). A CL-term can have at most one weak normal form.

Weak equality

Weak equality or weak convertibility $(X =_w Y)$:

Exist $X_0, \ldots, X_n$ such that

- $X_0 \equiv X$
- $X_n \equiv Y$
- $(\forall i \leq n-1)(X_i \rhd_{1w} X_{i+1} \quad \vee \quad X_{i+1} \rhd_{1w} X_i)$

Weak equality

Weak equality or weak convertibility ($X =_w Y$):

Exist $X_0, \ldots, X_n$ such that

- $X_0 \equiv X$
- $X_n \equiv Y$
- $(\forall i \leq n-1)(X_i \rhd_{1w} X_{i+1} \quad \vee \quad X_{i+1} \rhd_{1w} X_i)$

Theorem (Church-Rosser theorem for $=_w$).

$$\frac{X =_w Y}{\exists\, T.X \rhd_w T \wedge Y \rhd_w T}$$

Weak equality

Weak equality or weak convertibility ($X =_w Y$):

Exist $X_0, \ldots, X_n$ such that

- $X_0 \equiv X$
- $X_n \equiv Y$
- $(\forall i \leq n-1)(X_i \rhd_{1w} X_{i+1} \quad \vee \quad X_{i+1} \rhd_{1w} X_i)$

Theorem (Church-Rosser theorem for $=_w$).

$$\frac{X =_w Y}{\exists T. X \rhd_w T \wedge Y \rhd_w T}$$

Corollary. If X and Y are distinct weak normal forms, them $X \neq_w Y$; in particular $\boldsymbol{S} \neq_w \boldsymbol{K}$. Hence $=_w$ is non-trivial in the sense that not all terms are weakly equal.

Fixed-point combinators

Idea: For every term F there is a term X such

$$FX =_\beta X.$$

The term X is called a fixed-point of F.

Fixed-point combinators

Idea: For every term F there is a term X such

$$FX =_\beta X.$$

The term X is called a fixed-point of F.

Theorem. $\forall F \exists X. FX =_\beta X$.

Proof. Let $W \equiv \lambda x.F(xx)$, and let $X \equiv WW$. Then

$$\begin{aligned} X &\equiv (\lambda x.F(xx))W \\ &=_\beta F(WW) \\ &\equiv FX \end{aligned}$$

$\square$

Fixed-point combinators (cont).

Fixed-point combinator: A fixed-point combinator is any combinator $\boldsymbol{Y}$ such $\boldsymbol{Y}F =_\beta F(\boldsymbol{Y}F)$, for all terms F.

Fixed-point combinators (cont).

Fixed-point combinator: A fixed-point combinator is any combinator $\boldsymbol{Y}$ such $\boldsymbol{Y}F =_\beta F(\boldsymbol{Y}F)$, for all terms F.

Theorem. (Turing) $\boldsymbol{Y} \equiv UU$, where $U \equiv \lambda ux.x(uux)$ is a fixed-point combinator. (Whiteboard)

Theorem. (Curry and Rosenbloom) $\boldsymbol{Y} \equiv \lambda f.VV$, where $V \equiv \lambda x.f(xx)$ is a fixed-point combinator. (Whiteboard)

Fixed-point combinators (cont).

Fixed-point combinator: A fixed-point combinator is any combinator $\boldsymbol{Y}$ such $\boldsymbol{Y}F =_\beta F(\boldsymbol{Y}F)$, for all terms F.

Theorem. (Turing) $\boldsymbol{Y} \equiv UU$, where $U \equiv \lambda ux.x(uux)$ is a fixed-point combinator. (Whiteboard)

Theorem. (Curry and Rosenbloom) $\boldsymbol{Y} \equiv \lambda f.VV$, where $V \equiv \lambda x.f(xx)$ is a fixed-point combinator. (Whiteboard)

Corollary. For every term Z and $n \geq 0$, the equation

$$xy_1 \ldots y_n = Z$$

can be solved for x. That is, there is a term X such that

$$Xy_1 \ldots y_n =_\beta [X/x]Z.$$

Proof. $X \equiv \boldsymbol{Y}(\lambda xy_1 \ldots y_n.Z)$ (whiteboard). $\square$

Reduction strategies

Idea: Proving that a given term has no normal form.

Contraction $(X \triangleright_R Y)$:

$X \triangleright_R Y$: R is an redex in X and Y is the result of contracting R in X.

Example. $(\lambda x.(\lambda y.yx)z)v \triangleright_{(\lambda y.yx)z} (\lambda x.zx)v$.

Reduction strategies

Idea: Proving that a given term has no normal form.

Contraction ($X \triangleright_R Y$):

$X \triangleright_R Y$: R is an redex in X and Y is the result of contracting R in X.

Example. $(\lambda x.(\lambda y.yx)z)v \triangleright_{(\lambda y.yx)z} (\lambda x.zx)v$.

Reduction: A reduction ρ is a finite or infinite sequence of contractions separated by α-conversions

$$X_1 \triangleright_{R_1} Y_1 \equiv_\alpha X_2 \triangleright_{R_2} \ldots$$

Reduction strategies

Idea: Proving that a given term has no normal form.

Contraction $(X \rhd_R Y)$:

$X \rhd_R Y$: R is an redex in X and Y is the result of contracting R in X.

Example. $(\lambda x.(\lambda y.yx)z)v \rhd_{(\lambda y.yx)z} (\lambda x.zx)v$.

Reduction: A reduction ρ is a finite or infinite sequence of contractions separated by α-conversions

$$X_1 \rhd_{R_1} Y_1 \equiv_\alpha X_2 \rhd_{R_2} \ldots$$

Question: Given an initial term X, there is some way of choosing a reduction that will terminate if X has a normal form?

Reduction strategies (cont.)

Outermost (maximal) redex: A redex is called outermost iff it is not contained in any other redex.

Normal-order evaluation (left-most reduction) (call-by-name): In every contraction, the contracted redex is the leftmost outermost.

Eager evaluation (call-by-value): A redex is reduced only when its right hand side has reduced to a canonical form (variable, constant or lambda abstraction).

Reduction strategies (cont.)

Outermost (maximal) redex: A redex is called outermost iff it is not contained in any other redex.

Normal-order evaluation (left-most reduction) (call-by-name): In every contraction, the contracted redex is the leftmost outermost.

Eager evaluation (call-by-value): A redex is reduced only when its right hand side has reduced to a canonical form (variable, constant or lambda abstraction).

Example. (Whiteboard).

1. $(\lambda y.a)\Omega$, where $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$
2. $(\lambda x.xx)((\lambda y.yy)(\lambda z.zz))$

Reduction strategies (cont.)

Theorem (Standardization theorem (left-most reduction theorem)). If a term X has a normal form X^*, then the normal-order evaluation of X is finite and ends at X^*.

λ-calculus and inconsistencies

- λ-calculus $+$ logic: Curry paradox[1]
- λ-calculus $+$ set theory: Rusell paradox[2]

[1] J. Barkley Rosser. Highlights of the history of lambda-calculus. *Annals of the History of Computing*, 6(4):337–349, 1984.

[2] Paulson, 2000, sec. 4.6.

Encoding data in the λ-calculus

(From: Paulson, 2000, ch. 3)

Booleans:

$$
\begin{aligned}
\boldsymbol{true} &\equiv \lambda xy.x \\
\boldsymbol{false} &\equiv \lambda xy.y \\
\boldsymbol{if} &= \lambda pxy.pxy
\end{aligned}
$$

where

$$
\begin{aligned}
\boldsymbol{if}\ \boldsymbol{true}\ MN &=_\beta M \\
\boldsymbol{if}\ \boldsymbol{false}\ MN &=_\beta N
\end{aligned}
$$

Encoding data in the λ-calculus (cont.)

Ordered pairs:

$$
\begin{aligned}
\boldsymbol{pair} &\equiv \lambda xyf.fxy \\
\boldsymbol{fst} &\equiv \lambda p.p\ \boldsymbol{true} \\
\boldsymbol{snd} &= \lambda p.p\ \boldsymbol{false}
\end{aligned}
$$

where

$$
\begin{aligned}
\boldsymbol{fst}(\boldsymbol{pair}\,MN) &=_\beta M \\
\boldsymbol{snd}(\boldsymbol{pair}\,MN) &=_\beta N
\end{aligned}
$$

Encoding data in the λ-calculus (cont.)

Natural numbers:

Notation:

$$X^n Y \equiv \underbrace{X(X(\ldots(X}_{n\ \text{`}X\text{'s}} Y)\ldots)) \quad \text{if } n \geq 1 \ ,$$
$$X^0 Y \equiv Y.$$

The Church numerals:

$$\overline{n} \equiv \lambda f x. f^n x$$

Encoding data in the λ-calculus (cont.)

Some operations:

$$\begin{aligned}
\boldsymbol{add} &\equiv \lambda mnfx.mf(nfx) \\
\boldsymbol{mult} &\equiv \lambda mnfx.m(nf)x \\
\boldsymbol{isZero} &\equiv \lambda n.(\lambda x.\boldsymbol{false})\boldsymbol{true}
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{add}\ \overline{m}\ \overline{n} &=_\beta \overline{m+n} \\
\boldsymbol{mult}\ \overline{m}\ \overline{n} &=_\beta \overline{m \times n}
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{isZero}\ \overline{0} &=_\beta \boldsymbol{true} \\
\boldsymbol{isZero}\ \overline{n+1} &=_\beta \boldsymbol{false}
\end{aligned}$$

Recursion using fixed-points

Example. (Informally[3])

$$fac \equiv \lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } n * fac\ (n-1)$$

[3]Simon Peyton Jones. *The Implementation of Functional Programming Languages*. New York: Prentice-Hall International, 1987.

Recursion using fixed-points

Example. (Informally[3])

$$fac \equiv \lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } n * fac\ (n-1)$$
$$fac \equiv \lambda n.(\ldots fac \ldots)$$

[3] Simon Peyton Jones. *The Implementation of Functional Programming Languages*. New York: Prentice-Hall International, 1987.

Recursion using fixed-points

Example. (Informally[3])

$$fac \equiv \lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } n * fac\ (n-1)$$
$$fac \equiv \lambda n.(\ldots fac \ldots)$$
$$fac \equiv (\lambda f n.(\ldots f \ldots))fac$$

[3] Simon Peyton Jones. *The Implementation of Functional Programming Languages*. New York: Prentice-Hall International, 1987.

Recursion using fixed-points

Example. (Informally[3])

$$fac \equiv \lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } n * fac\ (n-1)$$
$$fac \equiv \lambda n.(\ldots fac \ldots)$$
$$fac \equiv (\lambda f n.(\ldots f \ldots))fac$$

$h \equiv \lambda f n.(\ldots f \ldots)$ -- not recursive!

$fac \equiv h\ fac$ -- fac is a fixed-point of h!

[3] Simon Peyton Jones. *The Implementation of Functional Programming Languages*. New York: Prentice-Hall International, 1987.

Recursion using fixed-points

Example. (Informally[3])

$$fac \equiv \lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } n * fac\ (n-1)$$
$$fac \equiv \lambda n.(\ldots fac \ldots)$$
$$fac \equiv (\lambda f n.(\ldots f \ldots))fac$$

$$h \equiv \lambda f n.(\ldots f \ldots) \qquad \text{-- not recursive!}$$
$$fac \equiv h\ fac \qquad \text{-- } fac \text{ is a fixed-point of } h!$$

$$fac \equiv \boldsymbol{Y} h$$

[3] Simon Peyton Jones. *The Implementation of Functional Programming Languages*. New York: Prentice-Hall International, 1987.

Recursion using fixed-points (cont.)

Example (cont.).

$$
\begin{aligned}
fac\,1 \equiv\; & \boldsymbol{Y}h\ 1 \\
=_\beta\; & h(\boldsymbol{Y}h)\ 1 \\
\equiv\; & (\lambda f n.(\dots f \dots))(\boldsymbol{Y}h)\ 1 \\
\rhd_\beta\; & \text{if } 1 = 0 \text{ then } 1 \text{ else } 1 * (\boldsymbol{Y}h\ 0) \\
\rhd_\beta\; & 1 * (\boldsymbol{Y}h\ 0) \\
=_\beta\; & 1 * (h(\boldsymbol{Y}h)\ 0) \\
\equiv\; & 1 * ((\lambda f n.(\dots f \dots))(\boldsymbol{Y}h)0) \\
\rhd_\beta\; & 1 * (\text{if } 0 = 0 \text{ then } 1 \text{ else } 1 * (\boldsymbol{Y}h\ (-1))) \\
\rhd_\beta\; & 1 * 1 \\
\rhd_\beta\; & 1
\end{aligned}
$$

Representing the computable functions

Representability:

Let ϕ a partial function $\phi : \mathbb{N}^n \to \mathbb{N}$. A term X represents ϕ iff

$$\begin{array}{rcl} \phi(m_1, \ldots, m_n) = p & \Rightarrow & X\overline{m_1} \ldots \overline{m_n} =_\beta \overline{p}, \\ \phi(m_1, \ldots, m_n) \text{ does not exits } & \Rightarrow & X\overline{m_1} \ldots \overline{m_n} \text{ has no nf }. \end{array}$$

Representing the computable functions

Representability:

Let ϕ a partial function $\phi : \mathbb{N}^n \to \mathbb{N}$. A term X represents ϕ iff

$$\begin{array}{rl} \phi(m_1, \ldots, m_n) = p \Rightarrow & X\overline{m_1} \ldots \overline{m_n} =_\beta \overline{p}, \\ \phi(m_1, \ldots, m_n) \text{ does not exits } \Rightarrow & X\overline{m_1} \ldots \overline{m_n} \text{ has no nf }. \end{array}$$

Example. The successor function $suc(n) = n + 1$ is represented by

$$\boldsymbol{suc} \equiv \lambda nfx.f(nfx)$$

Representing the computable functions

Representability:

Let ϕ a partial function $\phi : \mathbb{N}^n \to \mathbb{N}$. A term X represents ϕ iff

$$\phi(m_1, \ldots, m_n) = p \Rightarrow X\overline{m_1} \ldots \overline{m_n} =_\beta \overline{p},$$
$$\phi(m_1, \ldots, m_n) \text{ does not exits } \Rightarrow X\overline{m_1} \ldots \overline{m_n} \text{ has no nf }.$$

Example. The successor function $suc(n) = n + 1$ is represented by

$$\boldsymbol{suc} \equiv \lambda nfx.f(nfx)$$

Theorem (Representation of Turing-computable functions). In λ-calculus every Turing-computable function can be represented by a combinator.

Undecidability

Gödel number $\#M$:

$$\begin{aligned}\#x_i &= 2^i\\ \#(\lambda x_i.M) &= 3^i 5^{\#M}\\ \#(MN) &= 7^{\#M} 11^{\#N}\end{aligned}$$

Notation: $\ulcorner M \urcorner = \overline{\#M}$

Theorem. (Double fixed-point theorem) $\forall F \exists X. F \ulcorner X \urcorner =_\beta X$

Proof. (Whiteboard) $\square$

Undecidability (cont.)

Theorem. (Rice's theorem for the λ-calculus) Let $A \subset \lambda$-terms such A is non-trivial (i.e. $A \neq \emptyset$, $A \neq \lambda$-terms). Suppose that A is closed under $=_\beta$ (i.e. $M \in A, M =_\beta N \Rightarrow N \in A$). Then A is no recursive, that is $\#A = \{\#M \mid M \in A\}$ is not recursive.

Proof. (Whiteboard)[4] □

Theorem. The set $NF = \{M \mid M \text{ has a normal form }\}$ is not recursive.

Proof. NF is not trivial and it is closed under $=_\beta$. □

[4] Henk Barendregt. Functional programming and lambda calculus. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 321–363. Elsevier, 1990.

ISWIM: λ-calculus as a programming language

(From: Paulson, 2000, ch. 3)

- ISWIM: If you See What I Mean
- P. J. Landin. The next 700 programming languages. *Commun. ACM*, 9(3):157–166, 1966

ISWIM features

Simple declaration:

$\boldsymbol{let}\ x = M\ \boldsymbol{in}\ N \quad \equiv \quad (\lambda x.N)M$

Example.

- $\boldsymbol{let}\ n = \overline{0}\ \boldsymbol{in}\ \boldsymbol{suc}\ n$
- $\boldsymbol{let}\ m = \overline{0}\ \boldsymbol{in}\ (\boldsymbol{let}\ n = \overline{1}\ \boldsymbol{in}\ \boldsymbol{add}\ m\ n)$

Function declaration:

$\boldsymbol{let}\ f x_1 \ldots x_k = M\ \boldsymbol{in}\ N \quad \equiv \quad (\lambda f.N)(\lambda x_1 \ldots x_k.M)$

Example. $\boldsymbol{let}\ \boldsymbol{suc}\ n = \lambda f x.f(n f x)\ \boldsymbol{in}\ \boldsymbol{suc}\ \overline{0}$

ISWIM features (cont.)

Recursive declaration:

$$\textbf{\textit{letrec}}\ f x_1 \ldots x_k = M\ \textbf{\textit{in}}\ N \quad \equiv \quad (\lambda f.N)(\boldsymbol{Y}(\lambda f x_1 \ldots x_k.M))$$

Example.

$$\textbf{\textit{letrec}}\ fact\ n = \textbf{\textit{if}}\ (n == 0)\ 1\ (n * fact(n-1))\ \textbf{\textit{in}}\ fact\ 0$$

Pairs:

(M, N) : pair constructor

$\textbf{\textit{fst}}, \textbf{\textit{snd}}$: projections

$$\textbf{\textit{let}}\ \lambda(x, y).E \quad \equiv \quad \lambda z.(\lambda x y.E)(\textbf{\textit{fst}} z)(\textbf{\textit{snd}} z)$$

Example.

$$\textbf{\textit{let}}\ (x, y) = (\overline{2}, \overline{3})\ \textbf{\textit{in}}\ \textbf{\textit{add}}\ x\ y$$

The formal theory $\lambda\beta$ of β-equality

Formulas: $M = N$, where $M, N \in \lambda$-terms.

The formal theory $\lambda\beta$ of β-equality

Formulas: $M = N$, where $M, N \in \lambda$-terms.

Axiom-schemes:

$$
\begin{array}{lll}
(\alpha) & \lambda x.M = \lambda y.[y/x]M & \text{if } y \in FV(M), \\
(\beta) & (\lambda x.M)N = [N/x]M, & \\
(\rho) & M = M. &
\end{array}
$$

The formal theory $\lambda\beta$ of β-equality

Formulas: $M = N$, where $M, N \in \lambda$-terms.

Axiom-schemes:

$$
\begin{array}{lll}
(\alpha) & \lambda x.M = \lambda y.[y/x]M & \text{if } y \in FV(M), \\
(\beta) & (\lambda x.M)N = [N/x]M, & \\
(\rho) & M = M. &
\end{array}
$$

Rules of inference:

$$
(\mu)\quad \frac{M = M'}{NM = NM'} \qquad (\nu)\quad \frac{M = M'}{MN = M'N} \qquad (\tau)\quad \frac{M = N \quad N = P}{M = P}
$$

$$
(\xi)\quad \frac{M = M'}{\lambda x.M = \lambda x.M'} \qquad (\sigma)\quad \frac{M = N}{N = M}
$$

The formal theory $\lambda\beta$ of β-equality (cont.)

Deductions: $\lambda\beta, A_1, \ldots, A_n \vdash B$ (There is a deduction of B from the assumptions $A_1, \ldots, A_n$ in $\lambda\beta$).

Theorems: $\lambda\beta \vdash B$ (The formula B is probable in $\lambda\beta$).

The formal theory $\lambda\beta$ of β-equality (cont.)

Deductions: $\lambda\beta, A_1, \dots, A_n \vdash B$ (There is a deduction of B from the assumptions $A_1, \dots, A_n$ in $\lambda\beta$).

Theorems: $\lambda\beta \vdash B$ (The formula B is probable in $\lambda\beta$).

Remark: $\lambda\beta$ is a equational theory and it is a logic-free theory (there are not logical connectives or quantifiers in its formulae).

The formal theory $\lambda\beta$ of β-equality (cont.)

Example. Let M and N two closed terms

$$\dfrac{\dfrac{(\lambda x.(\lambda y.x))M = [M/x]\lambda y.x \equiv \lambda y.M}{(\lambda x.(\lambda y.x))MN = (\lambda y.M)N}(\nu) \qquad (\lambda y.M)N = [N/y]M \equiv M}{(\lambda x.(\lambda y.x))MN = M}(\tau)$$

That is to say, $\lambda\beta \vdash (\lambda xy.x)MN = M$.

The formal theory $\lambda\beta$ of β-equality (cont.)

Example. Let M and N two closed terms

$$\dfrac{\dfrac{(\lambda x.(\lambda y.x))M = [M/x]\lambda y.x \equiv \lambda y.M}{(\lambda x.(\lambda y.x))MN = (\lambda y.M)N}(\nu) \qquad (\lambda y.M)N = [N/y]M \equiv M}{(\lambda x.(\lambda y.x))MN = M}(\tau)$$

That is to say, $\lambda\beta \vdash (\lambda xy.x)MN = M$.

Theorem.

$$M =_\beta N \Longleftrightarrow \lambda\beta \vdash M = N.$$

The formal theory $\lambda\beta$ of β-reduction

Similar to the formal theory of β-equality, but:

1. Formulas: $M \rhd_\beta N$.
2. To change '$=$' by '$\rhd_\beta$'.
3. Remove the rule (σ).

The formal theory $\lambda\beta$ of β-reduction

Similar to the formal theory of β-equality, but:

1. Formulas: $M \triangleright_\beta N$.
2. To change '$=$' by '$\triangleright_\beta$'.
3. Remove the rule (σ).

Theorem.

$$M \triangleright_\beta N \iff \lambda\beta \vdash M \triangleright_\beta N.$$

The formal theory $\lambda\beta$ of β-reduction

Similar to the formal theory of β-equality, but:

1. Formulas: $M \rhd_\beta N$.
2. To change '$=$' by '$\rhd_\beta$'.
3. Remove the rule (σ).

Theorem.

$$M \rhd_\beta N \iff \lambda\beta \vdash M \rhd_\beta N.$$

Remark: Formal theories for combinatory logic.

The formal theory $\lambda\beta$ of β-reduction

Similar to the formal theory of β-equality, but:

1. Formulas: $M \rhd_\beta N$.
2. To change '$=$' by '$\rhd_\beta$'.
3. Remove the rule (σ).

Theorem.

$$M \rhd_\beta N \iff \lambda\beta \vdash M \rhd_\beta N.$$

Remark: Formal theories for combinatory logic.

Remark: $\lambda\beta$ is not a first-order theory.