Lambda Calculus

Akim Demaille `akim@lrde.epita.fr`

EPITA — École Pour l'Informatique et les Techniques Avancées

June 14, 2016

About these lecture notes

Many of these slides are largely inspired from Andrew D. Ker's lecture notes [Ker, 2005a, Ker, 2005b]. Some slides are even straightforward copies.

Lambda Calculus

1. λ-calculus
2. Reduction
3. λ-calculus as a Programming Language
4. Combinatory Logic

λ-calculus

1. λ-calculus
   - The Syntax of λ-calculus
   - Substitution, Conversions
2. Reduction
3. λ-calculus as a Programming Language
4. Combinatory Logic

Why the λ-calculus?

Church, Curry
  A theory of functions (1930s).

Turing
  A definition of effective computability (1930s).

Brouwer, Heyting, Kolmogorov
  A representation of formal proofs (1920–).

McCarthy, Scott, . . .
  A basis for functional programming languages (1960s–).

Montague, . . .
  Semantics for natural language (1960s–).

λ-calculus

Alonzo Church (1903–1995)

λ-calculus

Haskell Brooks Curry (1900–1982)

λ-calculus

Alan Mathison Turing (1912–1954)

λ-calculus

Richard Merritt Montague (1930–1971)

What is the λ-calculus?

- A mathematical theory of functions
- A (functional) programming language
- It allows reasoning on *operational* semantics
- Mathematicians are more inclined to *denotational* semantics

---

The Syntax of λ-calculus

1 λ-calculus
  - The Syntax of λ-calculus
  - Substitution, Conversions

2 Reduction

3 λ-calculus as a Programming Language

4 Combinatory Logic

The Pure Untyped λ-calculus

The simplest λ-calculus:

Variables $x, y, z\dots$

Functions $\lambda x \cdot M$

Application MN

No

- Booleans
- Numbers
- Types
- . . .

The λ-calculus Language

The λ-terms:

$$M \quad ::= \quad x \quad | \quad (\lambda x \cdot M) \quad | \quad (MM)$$

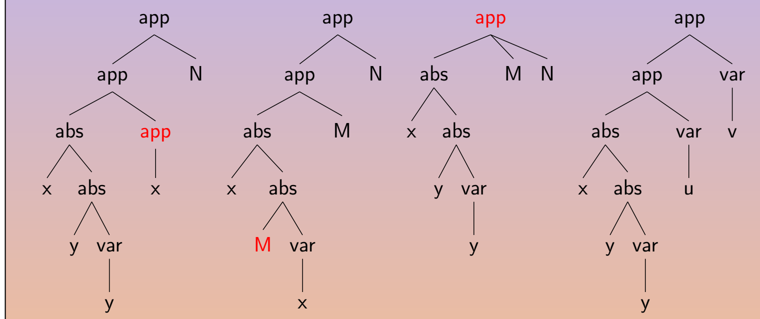
Conventions:

- Omit outer parentheses $MN = (MN)$
- Application associates to the left $MNL = (MN)L$
- Abstraction associates to the right $\lambda x \cdot MN = \lambda x \cdot (MN)$
- Multiple arguments as syntactic sugar $\lambda xy \cdot M = \lambda x \cdot \lambda y \cdot M$
  (Currying EN — Currification FR)

Notation

Usual	$x \mapsto 2x + 1$
λ-calculus	$\lambda x \cdot 2x + 1$
Originally	$\hat{}x \cdot 2x + 1$
Inpiration	$\hat{x} \cdot x = y$
Transition	$\Lambda x \cdot 2x + 1$

Which abstract-syntax trees are correct?



Fully qualified form for $\lambda nfx \cdot f(nfx)$

- ✓ $(\lambda n \cdot (\lambda f \cdot (\lambda x \cdot (f((nf)x)))))$
- ✗ $(\lambda x \cdot (\lambda f \cdot (\lambda n \cdot (f((nf)x)))))$
- ✗ $(\lambda n \cdot)(\lambda f \cdot)\lambda x \cdot (f((nf)x))$
- ✗ $(\lambda x \cdot (\lambda f \cdot (\lambda n \cdot f)))((nf)x))$

The λ-calculus Language: Alternative Presentation

The set Λ of λ-terms:

$$\frac{}{x \in \Lambda}\, x \in \mathcal{V} \qquad \frac{M \in \Lambda \quad N \in \Lambda}{(MN) \in \Lambda} \qquad \frac{M \in \Lambda}{(\lambda x \cdot M) \in \Lambda}\, x \in \mathcal{V}$$

For instance

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x \in \Lambda}}{(\lambda x \cdot x) \in \Lambda} \quad \dfrac{}{y \in \Lambda}}{((\lambda x \cdot x)y) \in \Lambda} \quad \dfrac{}{z \in \Lambda}}{(((\lambda x \cdot x)y)z) \in \Lambda}}{(\lambda z \cdot (((\lambda x \cdot x)y)z)) \in \Lambda} \quad \dfrac{}{x \in \Lambda}}{(\lambda z \cdot (((\lambda x \cdot x)y)z))x \in \Lambda}$$

Subterms

The set of subterms of M, $\mathrm{sub}(M)$:

$$\begin{aligned} \mathrm{sub}(x) &:= \{x\} \\ \mathrm{sub}(\lambda x \cdot M) &:= \{\lambda x \cdot M\} \cup \mathrm{sub}(M) \\ \mathrm{sub}(MN) &:= \{MN\} \cup \mathrm{sub}(M) \cup \mathrm{sub}(N) \end{aligned}$$

Variables

- The set of free variables of M, $\mathrm{FV}(M)$:

$$\begin{aligned} \mathrm{FV}(x) &:= \{x\} \\ \mathrm{FV}(\lambda x \cdot M) &:= \mathrm{FV}(M) \setminus \{x\} \\ \mathrm{FV}(MN) &:= \mathrm{FV}(M) \cup \mathrm{FV}(N) \end{aligned}$$

- A variable is free or bound.
- A variable may have bound and free occurrences: $x \lambda x \cdot x$.
- A term with no free variable is closed.
- A combinator is a closed term.

Substitution, Conversions

1. λ-calculus
   - The Syntax of λ-calculus
   - Substitution, Conversions
2. Reduction
3. λ-calculus as a Programming Language
4. Combinatory Logic

α-Conversion

α-conversion

M and N are α-convertible, $M \equiv N$, iff they differ only by renaming bound variables without introducing captures.

$$
\begin{gathered}
\lambda x \cdot x \equiv \lambda y \cdot y \\
x \lambda x \cdot x \equiv x \lambda y \cdot y \\
x \lambda x \cdot x \not\equiv y \lambda y \cdot y \\
\lambda x \cdot \lambda y \cdot xy \not\equiv \lambda x \cdot \lambda x \cdot xx
\end{gathered}
$$

From now on α-convertible terms are considered equal.

The Variable Convention

To avoid nasty capture issues, we will always silently α-convert terms so that no bound variable of a term is a variable (bound or free) of another.

Substitution

- The substitution of x by M in N is denoted $[M/x]N$.
- It is a notation, not an operation
- Intuitively, all the free occurrences of x are replaced by M.
- For instance $[\lambda z \cdot zz/x]\lambda y \cdot xy = \lambda y \cdot (\lambda z \cdot zz)y$.
- There are many notations for substitution:

$$
[M/x]N \qquad N[M/x] \qquad N[x := M] \qquad N[x \leftarrow M]
$$

and even

$$
N[x/M]
$$

Formal Definition of the Substitution

Substitution

$$
\begin{aligned}
{[M/x]x} &:= M \\
{[M/x]y} &:= y && \text{with } x \neq y \\
{[M/x](NL)} &:= ([M/x]N)([M/x]L) \\
{[M/x]\lambda y \cdot N} &:= \lambda y \cdot [M/x]N && \text{with } x \neq y \text{ and } y \notin \mathrm{FV}(M)
\end{aligned}
$$

The variable convention allows us to “require” that $y \notin \mathrm{FV}(M)$.
Without it:

$$
\begin{aligned}
{[M/x]\lambda y \cdot N} &:= \lambda y \cdot [M/x]N && \text{if } x \neq y \text{ and } y \notin \mathrm{FV}(M) \\
{[M/x]\lambda y \cdot N} &:= \lambda z \cdot [M/x][z/y]N && \text{if } x \neq y \text{ or } y \in \mathrm{FV}(M)
\end{aligned}
$$

Substitution

$$[yy/z](\lambda xy \cdot zy) \equiv \lambda xu \cdot (yy)u$$

β-Conversion

β-conversion

The β-convertibility between two terms is the relation β defined as:

$$(\lambda x \cdot M)N \quad \beta \quad [N/x]M$$

for any $M, N \in \Lambda$.

The $\lambda\beta$ Formal System

It is the “standard” theory of λ-calculus.

The $\lambda\beta$ Formal System

$$\frac{}{M = M} \qquad \frac{M = N}{N = M} \qquad \frac{M = N \quad N = L}{M = L}$$

$$\frac{M = M' \quad N = N'}{MN = M'N'} \qquad \frac{M = N}{\lambda x \cdot M = \lambda x \cdot N}$$

$$\frac{}{(\lambda x \cdot M)N = [N/x]M}$$

Reduction

1. λ-calculus
2. Reduction
    - β-Reduction
    - Church-Rosser
    - Reduction Strategies
3. λ-calculus as a Programming Language
4. Combinatory Logic

β-Reduction

1. λ-calculus
2. Reduction
   - β-Reduction
   - Church-Rosser
   - Reduction Strategies
3. λ-calculus as a Programming Language
4. Combinatory Logic

Reduction

One step R-Reduction from a relation R

The relation $\underset{R}{\rightarrow}$ is the smallest relation such that:

$$\frac{(M,N)\in R}{M \underset{R}{\rightarrow} N} \qquad \frac{M \underset{R}{\rightarrow} N}{ML \underset{R}{\rightarrow} NL} \qquad \frac{M \underset{R}{\rightarrow} N}{LM \underset{R}{\rightarrow} LN} \qquad \frac{M \underset{R}{\rightarrow} N}{\lambda x\cdot M \underset{R}{\rightarrow} \lambda x\cdot N}$$

R-Reduction: transitive, reflexive closure

The relation $\overset{*}{\underset{R}{\rightarrow}}$ is the smallest relation such that:

$$\frac{M \underset{R}{\rightarrow} N}{M \overset{*}{\underset{R}{\rightarrow}} N} \qquad \frac{}{M \overset{*}{\underset{R}{\rightarrow}} M} \qquad \frac{M \overset{*}{\underset{R}{\rightarrow}} N \quad N \overset{*}{\underset{R}{\rightarrow}} L}{M \overset{*}{\underset{R}{\rightarrow}} L}$$

β-Reduction

β-Redex

A β-redex is term under the form $(\lambda x\cdot M)N$.

One step β-Reduction

$$\frac{}{(\lambda x\cdot M)N \underset{\beta}{\rightarrow} [N/x]M} \qquad \cdots$$

β-Reduction

The relation $\overset{*}{\underset{\beta}{\rightarrow}}$ is the transitive, reflexive closure of $\underset{\beta}{\rightarrow}$.

β-Conversion

The relation $\underset{\beta}{\equiv}$ is the transitive, reflexive, symmetric closure of $\underset{\beta}{\rightarrow}$.

β-Reductions

$$\begin{aligned}
(\lambda x\cdot x)y &\rightarrow y\\
(\lambda x\cdot xx)y &\rightarrow yy\\
(\lambda x\cdot xx)(\lambda x\cdot xx) &\rightarrow (\lambda x\cdot xx)(\lambda x\cdot xx)\\
(\lambda x\cdot x(xx))(\lambda x\cdot x(xx)) &\rightarrow (\lambda x\cdot x(xx))((\lambda x\cdot x(xx))(\lambda x\cdot x(xx)))
\end{aligned}$$

Omega Combinators

$$\begin{aligned}
\omega &\equiv \lambda x\cdot xx\\
\Omega &\equiv \omega\omega\\
\widetilde{\Omega} &\equiv \lambda x\cdot x(xx)
\end{aligned}$$

More β-Reductions

$$\begin{aligned}
(\lambda x \cdot xyx)\lambda z \cdot z &\rightarrow (\lambda z \cdot z)y(\lambda z \cdot z)\\
(\lambda x \cdot x)((\lambda y \cdot y)x) &\rightarrow (\lambda x \cdot x)(x)\\
(\lambda x \cdot x)((\lambda y \cdot y)x) &\rightarrow ((\lambda y \cdot y)x)\\
(\lambda x \cdot x)((\lambda y \cdot y)x) &\overset{*}{\rightarrow} x\\
(\lambda x \cdot xx)((\lambda x \cdot xx)y) &\overset{*}{\rightarrow} yy(yy)\\
(\lambda x \cdot xx)((\lambda x \cdot x)y) &\overset{*}{\rightarrow} yy\\
(\lambda x \cdot x)((\lambda x \cdot xx)y) &\overset{*}{\rightarrow} yy
\end{aligned}$$

Therefore

$$\lambda\beta \vdash (\lambda x \cdot xx)((\lambda x \cdot x)y) = (\lambda x \cdot x)((\lambda x \cdot xx)y)$$

Other rules

η-reduction

$$\lambda x \cdot Mx \underset{\eta}{\rightarrow} M$$

η-expansion

$$M \underset{\eta_{exp}}{\rightarrow} \lambda x \cdot Mx$$

Church-Rosser

1. λ-calculus
2. Reduction
   - β-Reduction
   - Church-Rosser
   - Reduction Strategies
3. λ-calculus as a Programming Language
4. Combinatory Logic

Normal Forms

Given R, a relation on terms.

R-Normal Form (R-NF)

A term M is in R-Normal Form if there is no N such that $M \underset{R}{\rightarrow} N$.

R-Normalizable Term

A term M is R-Normalizable (or has an R-Normal Form) if there exists a term N in R-NF such that $M \underset{R}{\overset{*}{\rightarrow}} N$.

R-Strongly Normalization Term

A term M is R-Strongly Normalizable there is no infinite one-step reduction sequence starting from M. I.e., any one-step reduction sequence starting from M ends (on a R-NF term).

β-Normal Terms

- $I = \lambda x \cdot x$ is in β-NF
- II has a β-NF
  β-reduces to I
- II is β-strongly normalizing
- Ω is not (weakly) normalizable
  $\Omega = (\lambda x \cdot xx)(\lambda x \cdot xx) \to (\lambda x \cdot xx)(\lambda x \cdot xx) = \Omega$
- $KI\Omega$ is weakly normalizable ($K = \lambda x \cdot (\lambda y \cdot x)$)
  $KI\Omega \to I$
- $KI\Omega$ is not strongly normalizable
  $KI\Omega \to KI\Omega$

Normalizing Relation

Normalizing Relation

R is weakly normalizing if every term is R-normalizable.
R is strongly normalizing if every term is R-strongly normalizable.

β-Reduction

Ω is not weakly normalizable

β-reduction is not weakly normalizing!

Reduction Strategy

With a weakly normalizing relation that is not strongly normalizing:

- some terms are not weakly normalizable but not strongly
- i.e., some terms *can* be reduced *if* you reduce them “properly”

Reduction Strategy

A reduction strategy is a function specifying what is the next one-step reduction to perform.

Confluence

Given R, a relation on terms.

Diamond property

$\xrightarrow[R]{}$ satisfies the diamond property if $M \xrightarrow[R]{} N_1, M \xrightarrow[R]{} N_2$ implies the existence of L such that $N_1 \xrightarrow[R]{} L, N_2 \xrightarrow[R]{} L$.

Church-Rosser

$\xrightarrow[R]{}$ is Church-Rosser if $\xrightarrow[R]{*}$ satisfies the diamond property.

$\xrightarrow[R]{}$ is Church-Rosser if $M \xrightarrow[R]{*} N_1, M \xrightarrow[R]{*} N_2$ implies the existence of L such that $N_1 \xrightarrow[R]{*} L, N_2 \xrightarrow[R]{*} L$.

Confluence

Given R, a relation on terms.

Unique Normal Form Property

$\xrightarrow[R]{}$ has the unique normal form property if $M \xrightarrow[R]{*} N_1, M \xrightarrow[R]{*} N_2$ with N_1, N_2 in normal form, implies $N_1 \equiv N_2$.

Properties

- The diamond property implies Church-Rosser.
- If R is Church-Rosser then $M \underset{R}{=} N$ iff there exists L such that $M \xrightarrow[R]{*} L$ and $N \xrightarrow[R]{*} L$.
- If R is Church-Rosser then it has the unique normal form property.

λ-calculus has the Church-Rosser Property

β-reduction is Church-Rosser.

Any term has (at most) a unique NF.

Reduction Strategies

1. λ-calculus
2. Reduction
    - β-Reduction
    - Church-Rosser
    - Reduction Strategies
3. λ-calculus as a Programming Language
4. Combinatory Logic

Reduction Strategy

Reduction Strategy

A reduction strategy is a (partial) function from term to term.

If $\rightarrow$ is a reduction strategy, then any term has a unique maximal reduction sequence.

Head Reduction

Head Reduction

The head reduction $\xrightarrow{h}$ on terms is defined by:

$$\lambda\vec{x}\cdot(\lambda y\cdot M)N\vec{L} \xrightarrow{h} \lambda\vec{x}\cdot[N/y]M\vec{L}$$

$$\lambda x_1\dots x_n\cdot(\lambda y\cdot M)NL_1\dots L_m \xrightarrow{h} \lambda x_1\dots x_n\cdot[N/y]ML_1\dots L_m \quad n,m\geq 0$$

Note that any term has one of the following forms:

$$\lambda\vec{x}\cdot(\lambda y\cdot M)\vec{L} \qquad \lambda\vec{x}\cdot y\vec{L}$$

Head Reduction

$$\begin{aligned} KI\Omega &\xrightarrow{h} I \\ K\Omega I &\xrightarrow{h} \Omega I \\ &\xrightarrow{h} II \\ &\xrightarrow{h} I \\ xIx &\not\xrightarrow{h} xx \end{aligned}$$

Normal terms have the form:

$$\lambda\vec{x}\cdot y\vec{L}$$

Leftmost Reduction

Leftmost Reduction

The leftmost reduction $\xrightarrow{l}$ performs a single step of β-conversion on the leftmost $\lambda x \cdot M$.

Any head reduction is a leftmost reduction (but not conversly).

Leftmost reduction is normalizing.

λ-calculus as a Programming Language

1. λ-calculus
2. Reduction
3. λ-calculus as a Programming Language
   - Booleans
   - Natural Numbers
   - Pairs
   - Recursion
4. Combinatory Logic

Booleans

1. λ-calculus
2. Reduction
3. λ-calculus as a Programming Language
   - Booleans
   - Natural Numbers
   - Pairs
   - Recursion
4. Combinatory Logic

Booleans

- How would you code Booleans in λ-calculus?
- How would you translate `if M then N else L`?
- `if` MNL
- Do we *need* `if`?
- What if Booleans *were* the `if`?
- MNL
- What is true?
- What is false?

Boolean Combinators

Boolean Combinators (Church Booleans)

$$T := \lambda xy \cdot x$$
$$F := \lambda xy \cdot y$$

Natural Numbers

1. λ-calculus
2. Reduction
3. λ-calculus as a Programming Language
    - Booleans
    - Natural Numbers
    - Pairs
    - Recursion
4. Combinatory Logic

Church's Integers

Integers

$$\underline{n} := \lambda f \cdot \lambda x \cdot f^n x = \lambda f \cdot \lambda x \cdot \underbrace{(f \cdots (f}_{n \text{ times}}\, x\, \underbrace{) \cdots)}_{n \text{ times}}$$

$$\underline{2} = \lambda f \cdot \lambda x \cdot f(fx)$$
$$\underline{3} = \lambda f \cdot \lambda x \cdot f(f(fx))$$

Church's Integers

Operations

succ

$$\text{succ} := \lambda n \cdot \lambda f \cdot \lambda x \cdot f(nfx)$$

plus

$$\text{plus} := \lambda m \cdot \lambda n \cdot \lambda f \cdot \lambda x \cdot mf(nfx)$$
$$\text{plus} := \lambda m \cdot \lambda n \cdot n \text{ succ } m$$
$$\text{plus} := \lambda n \cdot n \text{ succ}$$

Pairs

1. λ-calculus
2. Reduction
3. λ-calculus as a Programming Language
   - Booleans
   - Natural Numbers
   - Pairs
   - Recursion
4. Combinatory Logic

Church's pairs

Pairs

$$\begin{aligned} pair &:= \lambda xy \cdot \lambda f \cdot fxy \\ first &:= \lambda p \cdot pT \\ second &:= \lambda p \cdot pF \end{aligned}$$

Recursion

1. λ-calculus
2. Reduction
3. λ-calculus as a Programming Language
   - Booleans
   - Natural Numbers
   - Pairs
   - Recursion
4. Combinatory Logic

Fixed point Combinators

Curry's Y Combinator

$$Y := \lambda f \cdot (\lambda x \cdot f(xx))(\lambda x \cdot f(xx))$$

Turing's Θ Combinator

$$\Theta := (\lambda xy \cdot y(xxy))(\lambda xy \cdot y(xxy))$$

There are infinitely many fixed-point combinators.

Fixed point Combinators

Curry's Y Combinator

$$Y := \lambda f \cdot (\lambda x \cdot f(xx))(\lambda x \cdot f(xx))$$

$$\begin{aligned}
Y\ g &= (\lambda f \cdot (\lambda x \cdot f(xx))(\lambda x \cdot f(xx)))\ g \\
&\to_\beta (\lambda x \cdot g(xx))(\lambda x \cdot g(xx)) \\
&\to_\beta g((\lambda x \cdot g(xx))(\lambda x \cdot g(xx))) \\
g(Y\ g) &\to_\beta g(\lambda f \cdot ((\lambda x \cdot f(xx))(\lambda x \cdot f(xx)))g) \\
&\to_\beta g(\lambda f \cdot ((\lambda x \cdot f(xx))(\lambda x \cdot f(xx)))g)
\end{aligned}$$

Reduction strategies in Programming Languages

Full beta reductions

Reduce any redex.

Applicative order

The leftmost, innermost redex is always reduced first. Intuitively reduce function "arguments" before the function itself. Applicative order always attempts to apply functions to normal forms, even when this is not possible.

Normal order

The leftmost, outermost redex is reduced first.

Reduction strategies in Programming Languages

Call by name

As normal order, but no reductions are performed inside abstractions. $\lambda x \cdot (\lambda x \cdot x)x$ is in NF.

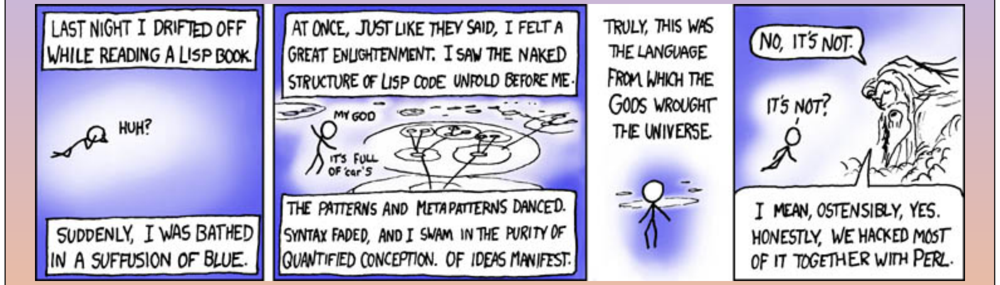
Call by value

Only the outermost redexes are reduced: a redex is reduced only when its right hand side has reduced to a value (variable or lambda abstraction).

Call by need

As normal order, but function applications that would duplicate terms instead name the argument, which is then reduced only "when it is needed". Called in practical contexts "lazy evaluation".

λ-calculus as a Programming Language



Lisp (xkcd 224)

Combinatory Logic

1. λ-calculus
2. Reduction
3. λ-calculus as a Programming Language
4. Combinatory Logic

Moses Ilyich Schönfinkel (1889–1942)

Russian logician and mathematician.
Member of David Hilbert's group at the University of Göttingen.
Mentally ill and in a sanatorium in 1927.
His papers were burned by his neighbors for heating.

Combinatory Logic

λ-reduction

- is complex
- its implementation is full of subtle pitfalls
- invented in 1936 by Alonzo Church

Combinatory Logic

- a simpler alternative
- invented by Moses Schönfinkel in 1920's
- developed by Haskell Curry in 1925

Combinators

Classic Combinators

$$\begin{aligned} S &:= (\lambda x \cdot (\lambda y \cdot (\lambda z \cdot ((xz)(yz))))) \\ K &:= (\lambda x \cdot (\lambda y \cdot x)) \\ I &:= (\lambda x \cdot x) \end{aligned}$$

We no longer need λ!

$$\begin{aligned} SXYZ &\to XZ(YZ) \\ KXY &\to X \\ IX &\to X \end{aligned}$$

Combinators

The Combinator I

$$I := (\lambda x \cdot x)$$

$$IX \to X$$

$$SKKX \to KX(KX) \to X$$

$$I = SKK$$

Combinatory Logic

S $SXYZ \to XZ(YZ)$
$(\lambda x \cdot (\lambda y \cdot (\lambda z \cdot ((xz)(yz)))))$

K $KXY \to X$
$(\lambda x \cdot (\lambda y \cdot x))$

I $IX \to X$
$(\lambda x \cdot x)$

Combinatory Logic

- Combination is left-associative:

$$SKKX = (((SK)K)X) \to KX(KX) \to X$$

- I.e., $I = SKK$: two symbols and two rules suffice.
- Same expressive power as λ-calculus.

Boolean Combinators

Boolean Combinators

$$T = K$$
$$F = KI$$

$$TXY \to X$$
$$FXY \to Y$$

$$KIXY = (((KI)X)Y) \to IY \to Y$$

The Y Combinator in SKI

- $$Y = S(K(SII))(S(S(KS)K)(K(SII)))$$
- The simplest fixed point combinator in SK
$$Y = SSK(S(K(SS(S(SSK))))K$$
- by Jan Willem Klop:
$$Yk = (LLLLLLLLLLLLLLLLLLLLLLLLLL)$$
where:
$$L = \lambda abcdefghijklmnopqstuvwxyzr(r(thisisafixedpointcombinator))$$

Bibliography Notes

[Ker, 2005a]	Complete and readable lecture notes on λ-calculus. Uses conventions different from ours.
[Ker, 2005b]	Additional information, including slides.
[Barendregt and Barendsen, 2000]	A classical introduction to λ-calculus.

Bibliography I

Barendregt, H. and Barendsen, E. (2000).
Introduction to lambda calculus.
http://www.cs.ru.nl/~erikb/onderwijs/T3/materiaal/lambda.pdf.

Ker, A. D. (2005a).
Lambda calculus and types.
http://web.comlab.ox.ac.uk/oucl/work/andrew.ker/lambda-calculus-notes-full-v3.pdf.

Ker, A. D. (2005b).
Lambda calculus notes.
http://web.comlab.ox.ac.uk/oucl/work/andrew.ker/.