Lambda Calculus

Akim Demaille akim@lrde.epita.fr

EPITA — École Pour l'Informatique et les Techniques Avancées

March 22, 2009

About these lecture notes

Many of these slides are largely inspired from Andrew D. Ker's lecture notes [2, 3]. Some slides are even straightforward copies from them.

Lambda Calculus

λ-calculus

Why the λ-calculus?

Church, Curry
A theory of functions (1920s).

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-).

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

The Pure Untyped λ-calculus

The simplest λ-calculus:

Variables $x, y, z...$

Functions $\lambda x \cdot M$

Application MN

No
- Booleans
- Numbers
- Types

The Pure Untyped λ-calculus

The simplest λ-calculus:

- Variables $x, y, z \ldots$
- 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 $MN = (MN)$

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 $MNL = (MN)L$
- Multiple arguments as syntactic sugar (Currification) $\lambda xy \cdot M = \lambda x \cdot \lambda y \cdot M$
- Abstraction associates to the right $\lambda x \cdot MN = \lambda x \cdot (MN)$

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 $MNL = (MN)L$
- Multiple arguments as syntactic sugar (Currification) $\lambda xy \cdot M = \lambda x \cdot \lambda y \cdot M$
- Abstraction associates to the right $\lambda x \cdot MN = \lambda x \cdot (MN)$

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 $MNL = (MN)L$
- Multiple arguments as syntactic sugar (Currification) $\lambda xy \cdot M = \lambda x \cdot \lambda y \cdot M$
- Abstraction associates to the right $\lambda x \cdot MN = \lambda x \cdot (MN)$

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
The Syntax of λ-calculus
Substitution, Conversions
Combinators

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 $MNL = (MN)L$
- Multiple arguments as syntactic sugar (Currification) $\lambda xy \cdot M = \lambda x \cdot \lambda y \cdot M$
- Abstraction associates to the right $\lambda x \cdot MN = \lambda x \cdot (MN)$

A. Demaille Lambda Calculus 9 / 62

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
The Syntax of λ-calculus
Substitution, Conversions
Combinators

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 $MNL = (MN)L$
- Multiple arguments as syntactic sugar (Currification) $\lambda xy \cdot M = \lambda x \cdot \lambda y \cdot M$
- Abstraction associates to the right $\lambda x \cdot MN = \lambda x \cdot (MN)$

A. Demaille Lambda Calculus 9 / 62

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

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

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. Sometimes called a combinator.

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. Sometimes called a combinator.

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. Sometimes called a combinator.

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
The Syntax of λ-calculus
Substitution, Conversions
Combinators

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. Sometimes called a combinator.

A. Demaille Lambda Calculus 12 / 62

Substitution, Conversions

α-Conversion

α-conversion

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

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

From now on α-convertible terms are considered equal.

α-Conversion

α-conversion

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

$$\begin{array}{rcl} \lambda x \cdot x & \equiv & \lambda y \cdot y \\ x\lambda x \cdot x & \equiv & x\lambda y \cdot y \\ x\lambda x \cdot x & \not\equiv & y\lambda y \cdot y \\ \lambda x \cdot \lambda y \cdot xy & \not\equiv & \lambda x \cdot \lambda x \cdot xx \end{array}$$

From now on α-convertible terms are considered equal.

α-Conversion

α-conversion

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

$$\begin{array}{rcl} \lambda x \cdot x & \equiv & \lambda y \cdot y \\ x\lambda x \cdot x & \equiv & x\lambda y \cdot y \\ x\lambda x \cdot x & \not\equiv & y\lambda y \cdot y \\ \lambda x \cdot \lambda y \cdot xy & \not\equiv & \lambda x \cdot \lambda x \cdot xx \end{array}$$

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 $[\lambda z \cdot zz/x]\lambda y \cdot xy = \lambda y \cdot \lambda z \cdot zzy$.
- There are many notations for substitution:

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

and even

$$N[x/M]$$

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 $[\lambda z \cdot zz/x]\lambda y \cdot xy = \lambda y \cdot \lambda z \cdot zzy$.
- There are many notations for substitution:

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

and even

$$N[x/M]$$

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
The Syntax of λ-calculus
Substitution, Conversions
Combinators

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 $[\lambda z \cdot zz/x]\lambda y \cdot xy = \lambda y \cdot \lambda z \cdot zzy$.
- There are many notations for substitution:

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

and even

$$N[x/M]$$

A. Demaille Lambda Calculus 16 / 62

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
The Syntax of λ-calculus
Substitution, Conversions
Combinators

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 $[\lambda z \cdot zz/x]\lambda y \cdot xy = \lambda y \cdot \lambda z \cdot zzy$.
- There are many notations for substitution:

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

and even

$$N[x/M]$$

A. Demaille Lambda Calculus 16 / 62

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
The Syntax of λ-calculus
Substitution, Conversions
Combinators

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 $[\lambda z \cdot zz/x]\lambda y \cdot xy = \lambda y \cdot \lambda z \cdot zzy$.
- There are many notations for substitution:

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

and even

$$N[x/M]$$

A. Demaille Lambda Calculus 16 / 62

Formal Definition of the Substitution

Substitution

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

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

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

Formal Definition of the Substitution

Substitution

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

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

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

Formal Definition of the Substitution

Substitution

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

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

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

Substitution

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

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

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

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

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

Combinators

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

Combinators

Classic Combinators

$$\begin{array}{rcl} \mathsf{S} & \equiv & (\lambda x \cdot (\lambda y \cdot (\lambda z \cdot ((xz)(yz))))) \\ \mathsf{K} & \equiv & (\lambda x \cdot (\lambda y \cdot x)) \\ \mathsf{I} & \equiv & (\lambda x \cdot x) \end{array}$$

We no longer need λ!

$$\begin{array}{rcl} SXYZ & \rightarrow & XZ(YZ) \\ KXY & \rightarrow & X \\ IX & \rightarrow & X \end{array}$$

Combinators

The Combinator I

$$\mathsf{I} \quad \equiv \quad (\lambda x \cdot x)$$

$$IX \quad \to \quad X$$

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

Combinators

The Combinator I

$$\mathsf{I} \quad \equiv \quad (\lambda x \cdot x)$$

$$IX \quad \to \quad X$$

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

Combinators

The Combinator I

$$\mathsf{I} \quad \equiv \quad (\lambda x \cdot x)$$

$$IX \quad \to \quad X$$

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

Combinators

The Combinator I

$$\mathsf{I} \quad \equiv \quad (\lambda x \cdot x)$$

$$IX \quad \to \quad X$$

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

$$I \quad = \quad SKK$$

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?

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?

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?

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?

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?

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?

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?

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

$$\begin{aligned} \mathrm{T} &\equiv \lambda xy \cdot x \\ \mathrm{F} &\equiv \lambda xy \cdot y \end{aligned}$$

$$\begin{aligned} \mathrm{TXY} &\rightarrow X \\ \mathrm{FXY} &\rightarrow Y \end{aligned}$$

$$\begin{aligned} \mathrm{T} &= K \\ \mathrm{F} &= KI \end{aligned}$$

$$\mathrm{KIXY} = (((\mathrm{KI})\mathrm{X})\mathrm{Y}) \rightarrow \mathrm{IY} \rightarrow \mathrm{Y}$$

Reduction

β-Reduction

Reduction

One step R-Reduction from a relation R

The relation $\rightarrow_R$ is the smallest relation such that:

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

R-Reduction: transitive, reflexive closure

The relation $\rightarrow_R^*$ is the smallest relation such that:

$$\frac{M \rightarrow_R N}{M \rightarrow_R^* N} \qquad \frac{}{M \rightarrow_R^* M} \qquad \frac{M \rightarrow_R^* N \quad N \rightarrow_R^* L}{M \rightarrow_R^* L}$$

Reduction

One step R-Reduction from a relation R

The relation $\to_R$ is the smallest relation such that:

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

R-Reduction: transitive, reflexive closure

The relation $\to_R^*$ is the smallest relation such that:

$$\frac{M \to_R N}{M \to_R^* N} \qquad \frac{}{M \to_R^* M} \qquad \frac{M \to_R^* N \quad N \to_R^* L}{M \to_R^* L}$$

β-Reduction

β-Redex

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

One step β-Reduction

It is the relation $\to_\beta$:

$$\frac{}{(\lambda x \cdot M)N \to_\beta [N/x]M} \quad \cdots$$

β-Reduction

The relation $\to_\beta^*$ is transitive, reflexive closure of $\to_\beta$.

β-Conversion

The relation $\equiv_\beta$ is transitive, reflexive, symmetric closure of $\to_\beta$.

β-Reduction

β-Redex

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

One step β-Reduction

It is the relation $\to_\beta$:

$$\frac{}{(\lambda x \cdot M)N \to_\beta [N/x]M} \quad \cdots$$

β-Reduction

The relation $\to_\beta^*$ is transitive, reflexive closure of $\to_\beta$.

β-Conversion

The relation $\equiv_\beta$ is transitive, reflexive, symmetric closure of $\to_\beta$.

β-Reduction

β-Redex

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

One step β-Reduction

It is the relation $\to_\beta$:

$$\frac{}{(\lambda x \cdot M)N \to_\beta [N/x]M} \quad \cdots$$

β-Reduction

The relation $\to_\beta^*$ is transitive, reflexive closure of $\to_\beta$.

β-Conversion

The relation $\equiv_\beta$ is transitive, reflexive, symmetric closure of $\to_\beta$.

β-Reduction

β-Redex

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

One step β-Reduction

It is the relation $\rightarrow_\beta$:

$$\frac{}{(\lambda x \cdot M)N \rightarrow_\beta [N/x]M} \quad \ldots$$

β-Reduction

The relation $\rightarrow_\beta^*$ is transitive, reflexive closure of $\rightarrow_\beta$.

β-Conversion

The relation $\equiv_\beta$ is transitive, reflexive, symmetric closure of $\rightarrow_\beta$.

β-Reductions

$$(\lambda x \cdot x)y \rightarrow$$

β-Reductions

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

β-Reductions

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

β-Reductions

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

Omega Combinators

$$\begin{array}{rcl} \omega & \equiv & \lambda x \cdot xx \\ \Omega & \equiv & \omega\omega \end{array}$$

More β-Reductions

$$(\lambda x \cdot xyx)\lambda z \cdot z \quad \rightarrow$$

More β-Reductions

$$\begin{array}{rcl} (\lambda x \cdot xyx)\lambda z \cdot z & \rightarrow & (\lambda z \cdot z)y(\lambda z \cdot z) \\ (\lambda x \cdot x)((\lambda y \cdot y)x) & \rightarrow & \end{array}$$

More β-Reductions

$$\begin{array}{rcl} (\lambda x \cdot xyx)\lambda z \cdot z & \rightarrow & (\lambda z \cdot z)y(\lambda z \cdot z) \\ (\lambda x \cdot x)((\lambda y \cdot y)x) & \rightarrow & (\lambda x \cdot x)(x) \\ (\lambda x \cdot x)((\lambda y \cdot y)x) & \rightarrow & \end{array}$$

More β-Reductions

$$\begin{array}{rcl}
(\lambda x \cdot xyx)\lambda z \cdot z & \to & (\lambda z \cdot z)y(\lambda z \cdot z) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to & (\lambda x \cdot x)(x) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to & ((\lambda y \cdot y)x) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to^* &
\end{array}$$

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
β-Reduction
Church-Rosser
Reduction Strategies

More β-Reductions

$$\begin{array}{rcl}
(\lambda x \cdot xyx)\lambda z \cdot z & \to & (\lambda z \cdot z)y(\lambda z \cdot z) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to & (\lambda x \cdot x)(x) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to & ((\lambda y \cdot y)x) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to^* & x \\
(\lambda x \cdot xx)((\lambda x \cdot xx)y) & \to^* &
\end{array}$$

A. Demaille Lambda Calculus 31 / 62

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
β-Reduction
Church-Rosser
Reduction Strategies

More β-Reductions

$$\begin{array}{rcl}
(\lambda x \cdot xyx)\lambda z \cdot z & \to & (\lambda z \cdot z)y(\lambda z \cdot z) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to & (\lambda x \cdot x)(x) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to & ((\lambda y \cdot y)x) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to^* & x \\
(\lambda x \cdot xx)((\lambda x \cdot xx)y) & \to^* & yy(yy) \\
(\lambda x \cdot xx)((\lambda x \cdot x)y) & \to^* &
\end{array}$$

A. Demaille Lambda Calculus 31 / 62

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
β-Reduction
Church-Rosser
Reduction Strategies

More β-Reductions

$$\begin{array}{rcl}
(\lambda x \cdot xyx)\lambda z \cdot z & \to & (\lambda z \cdot z)y(\lambda z \cdot z) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to & (\lambda x \cdot x)(x) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to & ((\lambda y \cdot y)x) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to^* & x \\
(\lambda x \cdot xx)((\lambda x \cdot xx)y) & \to^* & yy(yy) \\
(\lambda x \cdot xx)((\lambda x \cdot x)y) & \to^* & yy \\
(\lambda x \cdot x)((\lambda x \cdot xx)y) & \to^* &
\end{array}$$

A. Demaille Lambda Calculus 31 / 62

More β-Reductions

$$
\begin{array}{rcl}
(\lambda x \cdot xyx)\lambda z \cdot z & \to & (\lambda z \cdot z)y(\lambda z \cdot z) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to & (\lambda x \cdot x)(x) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to & ((\lambda y \cdot y)x) \\
(\lambda x \cdot x)((\lambda y \cdot y)x) & \to^* & x \\
(\lambda x \cdot xx)((\lambda x \cdot xx)y) & \to^* & yy(yy) \\
(\lambda x \cdot xx)((\lambda x \cdot x)y) & \to^* & yy \\
(\lambda x \cdot x)((\lambda x \cdot xx)y) & \to^* & yy
\end{array}
$$

Therefore

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

Normal Forms

Given R, a relation on terms.

R-Normal Form

A term M is in R-Normal Form (R-NF) if there is no N such that $M \to_R N$.

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 $M \to_R^* N$.

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).

Normal Forms

Given R, a relation on terms.

R-Normal Form

A term M is in R-Normal Form (R-NF) if there is no N such that $M \to_R N$.

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 $M \to_R^* N$.

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).

Normal Forms

Given R, a relation on terms.

R-Normal Form

A term M is in R-Normal Form (R-NF) if there is no N such that $M \to_R N$.

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 $M \to_R^* N$.

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

- $\lambda x \cdot x$ is in β-NF
- $(\lambda x \cdot x)(\lambda x \cdot x)$ has a β-NF
  β-reduces to $\lambda x \cdot x$
- $(\lambda x \cdot x)(\lambda x \cdot x)$ is β-strongly normalizing
- Ω is not (weakly) normalizable
  $\Omega = (\lambda x \cdot xx)(\lambda x \cdot xx) \rightarrow (\lambda x \cdot xx)(\lambda x \cdot xx) = \Omega$
- $\mathsf{K}\mathsf{I}\Omega$ is weakly normalizable
  $\mathsf{K}\mathsf{I}\Omega \rightarrow \mathsf{I}$
- $\mathsf{K}\mathsf{I}\Omega$ is not strongly normalizable
  $\mathsf{K}\mathsf{I}\Omega \rightarrow \mathsf{K}\mathsf{I}\Omega$

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
β-Reduction
Church-Rosser
Reduction Strategies

β-Normal Terms

- $\lambda x \cdot x$ is in β-NF
- $(\lambda x \cdot x)(\lambda x \cdot x)$ has a β-NF
  β-reduces to $\lambda x \cdot x$
- $(\lambda x \cdot x)(\lambda x \cdot x)$ is β-strongly normalizing
- Ω is not (weakly) normalizable
  $\Omega = (\lambda x \cdot xx)(\lambda x \cdot xx) \rightarrow (\lambda x \cdot xx)(\lambda x \cdot xx) = \Omega$
- $\mathsf{K}\mathsf{I}\Omega$ is weakly normalizable
  $\mathsf{K}\mathsf{I}\Omega \rightarrow \mathsf{I}$
- $\mathsf{K}\mathsf{I}\Omega$ is not strongly normalizable
  $\mathsf{K}\mathsf{I}\Omega \rightarrow \mathsf{K}\mathsf{I}\Omega$

A. Demaille Lambda Calculus 33 / 62

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
β-Reduction
Church-Rosser
Reduction Strategies

β-Normal Terms

- $\lambda x \cdot x$ is in β-NF
- $(\lambda x \cdot x)(\lambda x \cdot x)$ has a β-NF
  β-reduces to $\lambda x \cdot x$
- $(\lambda x \cdot x)(\lambda x \cdot x)$ is β-strongly normalizing
- Ω is not (weakly) normalizable
  $\Omega = (\lambda x \cdot xx)(\lambda x \cdot xx) \rightarrow (\lambda x \cdot xx)(\lambda x \cdot xx) = \Omega$
- $\mathsf{K}\mathsf{I}\Omega$ is weakly normalizable
  $\mathsf{K}\mathsf{I}\Omega \rightarrow \mathsf{I}$
- $\mathsf{K}\mathsf{I}\Omega$ is not strongly normalizable
  $\mathsf{K}\mathsf{I}\Omega \rightarrow \mathsf{K}\mathsf{I}\Omega$

A. Demaille Lambda Calculus 33 / 62

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
β-Reduction
Church-Rosser
Reduction Strategies

β-Normal Terms

- $\lambda x \cdot x$ is in β-NF
- $(\lambda x \cdot x)(\lambda x \cdot x)$ has a β-NF
  β-reduces to $\lambda x \cdot x$
- $(\lambda x \cdot x)(\lambda x \cdot x)$ is β-strongly normalizing
- Ω is not (weakly) normalizable
  $\Omega = (\lambda x \cdot xx)(\lambda x \cdot xx) \rightarrow (\lambda x \cdot xx)(\lambda x \cdot xx) = \Omega$
- $\mathsf{K}\mathsf{I}\Omega$ is weakly normalizable
  $\mathsf{K}\mathsf{I}\Omega \rightarrow \mathsf{I}$
- $\mathsf{K}\mathsf{I}\Omega$ is not strongly normalizable
  $\mathsf{K}\mathsf{I}\Omega \rightarrow \mathsf{K}\mathsf{I}\Omega$

A. Demaille Lambda Calculus 33 / 62

β-Normal Terms

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

β-Normal Terms

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

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.

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.

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.

Church-Rosser

Confluence

Given R, a relation on terms.

Diamond property

$\longrightarrow_R$ satisfies the diamond property if $M \longrightarrow_R N_1, M \longrightarrow_R N_2$ implies the existence of L such that $N_1 \longrightarrow_R L, N_2 \longrightarrow_R L$.

Church-Rosser

$\longrightarrow_R$ is Church-Rosser if $\longrightarrow_R^*$ satisfies the diamond property.

$\longrightarrow_R$ is Church-Rosser if $M \longrightarrow_R^* N_1, M \longrightarrow_R^* N_2$ implies the existence of L such that $N_1 \longrightarrow_R^* L, N_2 \longrightarrow_R^* L$.

Confluence

Given R, a relation on terms.

Diamond property

$\longrightarrow_R$ satisfies the diamond property if $M \longrightarrow_R N_1, M \longrightarrow_R N_2$ implies the existence of L such that $N_1 \longrightarrow_R L, N_2 \longrightarrow_R L$.

Church-Rosser

$\longrightarrow_R$ is Church-Rosser if $\longrightarrow_R^*$ satisfies the diamond property.

$\longrightarrow_R$ is Church-Rosser if $M \longrightarrow_R^* N_1, M \longrightarrow_R^* N_2$ implies the existence of L such that $N_1 \longrightarrow_R^* L, N_2 \longrightarrow_R^* L$.

Confluence

Given R, a relation on terms.

Diamond property

$\longrightarrow_R$ satisfies the diamond property if $M \longrightarrow_R N_1, M \longrightarrow_R N_2$ implies the existence of L such that $N_1 \longrightarrow_R L, N_2 \longrightarrow_R L$.

Church-Rosser

$\longrightarrow_R$ is Church-Rosser if $\longrightarrow_R^*$ satisfies the diamond property.

$\longrightarrow_R$ is Church-Rosser if $M \longrightarrow_R^* N_1, M \longrightarrow_R^* N_2$ implies the existence of L such that $N_1 \longrightarrow_R^* L, N_2 \longrightarrow_R^* L$.

Confluence

Given R, a relation on terms.

Unique Normal Form Property

$\longrightarrow_R$ has the unique normal form property if $M \longrightarrow_R^* N_1, M \longrightarrow_R^* N_2$ 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 $M =_R N$ iff there exists L such that $M \to_R^* L$ and $N \to_R^* L$.
- If R is Church-Rosser then it has the unique normal form property.

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
β-Reduction
Church-Rosser
Reduction Strategies

Properties

- The diamond property implies Church-Rosser.
- If R is Church-Rosser then $M =_R N$ iff there exists L such that $M \to_R^* L$ and $N \to_R^* L$.
- If R is Church-Rosser then it has the unique normal form property.

A. Demaille Lambda Calculus 40 / 62

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
β-Reduction
Church-Rosser
Reduction Strategies

Properties

- The diamond property implies Church-Rosser.
- If R is Church-Rosser then $M =_R N$ iff there exists L such that $M \to_R^* L$ and $N \to_R^* L$.
- If R is Church-Rosser then it has the unique normal form property.

A. Demaille Lambda Calculus 40 / 62

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
β-Reduction
Church-Rosser
Reduction Strategies

λ-calculus has the Church-Rosser Property

β-reduction is Church-Rosser.

Any term has (at most) a unique NF.

λ-calculus has the Church-Rosser Property

β-reduction is Church-Rosser.

Any term has (at most) a unique NF.

Reduction Strategies

Reduction Strategy

Reduction Strategy

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

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

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
β-Reduction
Church-Rosser
Reduction Strategies

Reduction Strategy

Reduction Strategy

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

If $\longrightarrow$ 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:

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

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
β-Reduction
Church-Rosser
Reduction Strategies

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:

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

A. Demaille Lambda Calculus 44 / 62

Head Reduction

$$\begin{array}{rcl}
\mathsf{K}\mathsf{I}\Omega & \xrightarrow{h} & \mathsf{I} \\
\mathsf{K}\Omega\mathsf{I} & \xrightarrow{h} & \Omega\mathsf{I} \\
 & \xrightarrow{h} & \mathsf{I}\mathsf{I} \\
 & \xrightarrow{h} & \mathsf{I} \\
x\mathsf{I}x & \not\xrightarrow{h} & xx
\end{array}$$

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.

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.

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

Booleans

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

Integers

Church's Integers

Integers

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

$$\begin{aligned} \underline{2} &= \lambda f \cdot \lambda x \cdot f(fx) \\ \underline{3} &= \lambda f \cdot \lambda x \cdot f(f(fx)) \end{aligned}$$

Church's Integers

Operations

succ

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

plus

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

Pairs

Recursion

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

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

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

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

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:
  $$Yk = (LLLLLLLLLLLLLLLLLLLLLLLLLLLL)$$
  where:
  $$L = \lambda abcdefghijklmnopqstuvwxyzr(r(thisisafixedpointcombinator))$$

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:
  $$L = \lambda abcdefghijklmnopqstuvwxyzr(r(thisisafixedpointcombinator))$$

λ-calculus
Reduction
λ-calculus as a Programming Language
Combinatory Logic
Booleans
Integers
Pairs
Recursion

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:
$$L = \lambda abcdefghijklmnopqstuvwxyzr(r(thisisafixedpointcombinator))$$

A. Demaille Lambda Calculus 55 / 62

Reduction strategies in Programming Languages

Full beta reductions Reduce any redex.

Applicative order The rightmost, 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".

Combinatory Logic

Combinatory Logic

- λ-reduction is complex
- its implementation is full of subtle pitfalls

A simpler alternative: *Combinatory Logic*, invented by Shoenfinkel and developed by Curry and others in the 1920's.

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

Combination is left-associative:
$SKKX = (((SK)K)X) \to KX(KX) \to X$. I.e., $I = SKK$: two symbols and two rule suffice. Same expressive power as λ-calculus.

Bibliography Notes

[2] Complete and readable lecture notes on λ-calculus. Uses conventions different from ours.

[3] Additional information, including slides.

[1] A classical introduction to λ-calculus.

Bibliography I

- Henk Barendregt and Erik Barendsen.
  Introduction to lambda calculus, March 2000.
  http://www.cs.ru.nl/~erikb/onderwijs/T3/materiaal/lambda.pdf.
- Andrew D. Ker.
  Lambda calculus and types, May 2005.
  http://web.comlab.ox.ac.uk/oucl/work/andrew.ker/lambda-calculus-notes-full-v3.pdf.
- Andrew D. Ker.
  Lambda calculus notes, May 2005.
  http://web.comlab.ox.ac.uk/oucl/work/andrew.ker/.