λ-calculus and simple types

Sabine Broda
Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto

MAP-i, Braga 2007

λ-calculus

λ-calculus

- conceived (ca. 1930) as part of a general (later shown inconsistent) theory of functions and logic, intended as a foundation for mathematics;

λ-calculus

- conceived (ca. 1930) as part of a general (later shown inconsistent) theory of functions and logic, intended as a foundation for mathematics;
- all recursive functions can be represented in the (pure) λ-calculus;

λ-calculus

- conceived (ca. 1930) as part of a general (later shown inconsistent) theory of functions and logic, intended as a foundation for mathematics;
- all recursive functions can be represented in the (pure) λ-calculus;
- theory modelling functions and their applicative behaviour;

λ-calculus

- conceived (ca. 1930) as part of a general (later shown inconsistent) theory of functions and logic, intended as a foundation for mathematics;
- all recursive functions can be represented in the (pure) λ-calculus;
- theory modelling functions and their applicative behaviour;
- concept of function seen as a rule, i.e. process of passing an argument to a value (contrary to the notion of seeing a function as a graph);

λ-calculus

- conceived (ca. 1930) as part of a general (later shown inconsistent) theory of functions and logic, intended as a foundation for mathematics;
- all recursive functions can be represented in the (pure) λ-calculus;
- theory modelling functions and their applicative behaviour;
- concept of function seen as a rule, i.e. process of passing an argument to a value (contrary to the notion of seeing a function as a graph);
- this is important for the study of computability and for theory of computation in general, since it emphasizes the computational aspect associated to the notion of function.

λ-terms

λ-terms

- infinite set of *term-variables* x, y, z, $\ldots$;

λ-terms

- infinite set of *term-variables* $x, y, z, \ldots$;
  - each variable x is a λ-term;

λ-terms

- infinite set of *term-variables* $x, y, z, \ldots$;

- each variable x is a λ-term;
- if M and N are λ-terms, then (MN) is a λ-term, (*application*);

λ-terms

- infinite set of *term-variables* $x, y, z, \ldots$;

- each variable x is a λ-term;
- if M and N are λ-terms, then (MN) is a λ-term, (*application*);
- if M is a λ-term and x a variable, then $(\lambda x.M)$ is a λ-term, (*abstraction*).

λ-terms

- infinite set of *term-variables* x, y, z, $\ldots$;
  - each variable x is a λ-term;
  - if M and N are λ-terms, then (MN) is a λ-term, (*application*);
  - if M is a λ-term and x a variable, then $(\lambda x.M)$ is a λ-term, (*abstraction*).

Examples: $(\lambda x.x)$, $(x(\lambda y.(xy)))$,$\ldots$

Conventions

Conventions

- application associates to the left;

$$MNO \text{ stands for } ((MN)O)$$

Conventions

- application associates to the left;

$$MNO \text{ stands for } ((MN)O)$$

- bodies of lambdas extend as far as possible;

$$\lambda x.\lambda y.M \text{ stands for } \lambda x.(\lambda y.M)$$

Conventions

- application associates to the left;

  MNO stands for $((MN)O)$

- bodies of lambdas extend as far as possible;

  $\lambda x.\lambda y.M$ stands for $\lambda x.(\lambda y.M)$

- nested lambdas may be collapsed together;

  $\lambda xy.M$ stands for $\lambda x.(\lambda y.M)$

Bound occurrences of variables - α-conversion

Bound occurrences of variables - α-conversion

- all occurrences of a variable x that occur in an expression of the form $\lambda x.M$ are *bound*;

Bound occurrences of variables - α-conversion

- all occurrences of a variable x that occur in an expression of the form $\lambda x.M$ are *bound*;
- an occurrence of a variable that is not bound is called *free*;

Bound occurrences of variables - α-conversion

- all occurrences of a variable x that occur in an expression of the form $\lambda x.M$ are *bound*;
- an occurrence of a variable that is not bound is called *free*;
- $FV(M)$ is the set of variables with free occurrences in M;

Bound occurrences of variables - α-conversion

- all occurrences of a variable x that occur in an expression of the form $\lambda x.M$ are *bound*;
- an occurrence of a variable that is not bound is called *free*;
- $FV(M)$ is the set of variables with free occurrences in M;
- if $FV(M) = \emptyset$ we say that M is closed;

Bound occurrences of variables - α-conversion

- all occurrences of a variable x that occur in an expression of the form $\lambda x.M$ are *bound*;
- an occurrence of a variable that is not bound is called *free*;
- $FV(M)$ is the set of variables with free occurrences in M;
- if $FV(M) = \emptyset$ we say that M is closed;
- we will consider λ-terms equivalent up to bound variable renaming, (α-*conversion*).

Bound occurrences of variables - α-conversion

- all occurrences of a variable x that occur in an expression of the form $\lambda x.M$ are *bound*;
- an occurrence of a variable that is not bound is called *free*;
- $FV(M)$ is the set of variables with free occurrences in M;
- if $FV(M) = \emptyset$ we say that M is closed;
- we will consider λ-terms equivalent up to bound variable renaming, (α-*conversion*).

Examples: $\lambda xy.xyz \equiv_\alpha \lambda yu.yuz$, but $(\lambda x.x)z \not\equiv_\alpha (\lambda x.y)z$

Substitution

Substitution

The expression $M[N/x]$ denotes the result of substituting in M each free occurrence of x by N and making any changes of bound variables needed to prevent variables free in N from becoming bound in $M[N/x]$.

Substitution

The expression $M[N/x]$ denotes the result of substituting in M each free occurrence of x by N and making any changes of bound variables needed to prevent variables free in N from becoming bound in $M[N/x]$.

Example:

$$(\lambda xy.xyz)[(\lambda u.y)/z] \not\equiv \lambda xy.xy(\lambda u.y)$$

Substitution

The expression $M[N/x]$ denotes the result of substituting in M each free occurrence of x by N and making any changes of bound variables needed to prevent variables free in N from becoming bound in $M[N/x]$.

Example:

$$(\lambda xy.xyz)[(\lambda u.y)/z] \not\equiv \lambda xy.xy(\lambda u.y)$$

but

$$(\lambda xy.xyz)[(\lambda u.y)/z] \equiv \lambda xv.xv(\lambda u.y)$$

β-reduction

β-reduction

- a term of the form $(\lambda x.M)N$ is called a β-*redex*;

β-reduction

- a term of the form $(\lambda x.M)N$ is called a β-*redex*;
- its *contractum* is the term $M[N/x]$;

β-reduction

- a term of the form $(\lambda x.M)N$ is called a *β-redex*;
- its *contractum* is the term $M[N/x]$;
- we write $M \to_{1\beta} N$, and say that M reduces in one step of β-reduction to N, iff N can be obtained from M by replacing one β-redex in M by its contractum;

β-reduction

- a term of the form $(\lambda x.M)N$ is called a β-*redex*;
- its *contractum* is the term $M[N/x]$;
- we write $M \rightarrow_{1\beta} N$, and say that M reduces in one step of β-reduction to N, iff N can be obtained from M by replacing one β-redex in M by its contractum;
- $\rightarrow_\beta$ is the reflexive and transitive closure of $\rightarrow_{1\beta}$;

β-reduction

- a term of the form $(\lambda x.M)N$ is called a β-*redex*;
- its *contractum* is the term $M[N/x]$;
- we write $M \rightarrow_{1\beta} N$, and say that M reduces in one step of β-reduction to N, iff N can be obtained from M by replacing one β-redex in M by its contractum;
- $\rightarrow_{\beta}$ is the reflexive and transitive closure of $\rightarrow_{1\beta}$;
- $\equiv_{\beta}$ is the reflexive, simetric and transitive closure of $\rightarrow_{1\beta}$.

β-normal forms

β-normal forms

- A term M is said to be in β-*normal form* (or β-nf) if it contains no β-redex;

β-normal forms

- A term M is said to be in *β-normal form* (or β-nf) if it contains no β-redex;
- we say that M *has a β-nf* if there is some β-nf N such that $M \twoheadrightarrow_\beta N$.

β-normal forms

- A term M is said to be in *β-normal form* (or β-nf) if it contains no β-redex;
- we say that M *has a β-nf* if there is some β-nf N such that $M \twoheadrightarrow_\beta N$.

Exercise: Reduce the following terms to their β-normal form.

- $(\lambda x.xx)(\lambda x.xx)$
- $(\lambda xy.x)(\lambda x.x)((\lambda x.xx)(\lambda x.xx))$
- $(\lambda x.xx)(\lambda yz.yz)$.

β-normal forms

- A term M is said to be in *β-normal form* (or β-nf) if it contains no β-redex;
- we say that *M has a β-nf* if there is some β-nf N such that $M \twoheadrightarrow_\beta N$.

Exercise: Reduce the following terms to their β-normal form.

- $(\lambda x.xx)(\lambda x.xx)$
- $(\lambda xy.x)(\lambda x.x)((\lambda x.xx)(\lambda x.xx))$
- $(\lambda x.xx)(\lambda yz.yz)$.

Conclusions:

- The term $(\lambda x.xx)(\lambda x.xx)$ has no β-nf since

$$\begin{aligned} (\lambda x.xx)(\lambda x.xx) &\rightarrow_{1\beta} (\lambda x.xx)(\lambda x.xx) \\ &\rightarrow_{1\beta} (\lambda x.xx)(\lambda x.xx) \\ &\rightarrow_{1\beta} \cdots \end{aligned}$$

β-normal forms

- A term M is said to be in *β-normal form* (or β-nf) if it contains no β-redex;
- we say that *M has a β-nf* if there is some β-nf N such that $M \twoheadrightarrow_\beta N$.

Exercise: Reduce the following terms to their β-normal form.

- $(\lambda x.xx)(\lambda x.xx)$
- $(\lambda xy.x)(\lambda x.x)((\lambda x.xx)(\lambda x.xx))$
- $(\lambda x.xx)(\lambda yz.yz)$.

Conclusions:

- The term $(\lambda x.xx)(\lambda x.xx)$ has no β-nf since
$$\begin{aligned}(\lambda x.xx)(\lambda x.xx) &\to_{1\beta} (\lambda x.xx)(\lambda x.xx)\\ &\to_{1\beta} (\lambda x.xx)(\lambda x.xx)\\ &\to_{1\beta} \cdots\end{aligned}$$
- the term $(\lambda xy.x)(\lambda x.x)((\lambda x.xx)(\lambda x.xx))$ has normal form $\lambda x.x$, but not every reduction sequence leads to this normal form.

Confluence

Confluence

Theorem: (Church-Rosser) If $M \twoheadrightarrow_\beta N_1$ and $M \twoheadrightarrow_\beta N_2$, then there is a term P such that $N_1 \twoheadrightarrow_\beta P$ and $N_2 \twoheadrightarrow_\beta P$.

Confluence

Theorem: (Church-Rosser) If $M \twoheadrightarrow_\beta N_1$ and $M \twoheadrightarrow_\beta N_2$, then there is a term P such that $N_1 \twoheadrightarrow_\beta P$ and $N_2 \twoheadrightarrow_\beta P$.

Corollary: Every term M has at most one β-nf.

Confluence

Theorem: (Church-Rosser) If $M \twoheadrightarrow_\beta N_1$ and $M \twoheadrightarrow_\beta N_2$, then there is a term P such that $N_1 \twoheadrightarrow_\beta P$ and $N_2 \twoheadrightarrow_\beta P$.

Corollary: Every term M has at most one β-nf.

Normal order reduction: Deterministic strategy which chooses the leftmost, outermost redex, until there are no more redexes.

Confluence

Theorem: (Church-Rosser) If $M \twoheadrightarrow_\beta N_1$ and $M \twoheadrightarrow_\beta N_2$, then there is a term P such that $N_1 \twoheadrightarrow_\beta P$ and $N_2 \twoheadrightarrow_\beta P$.

Corollary: Every term M has at most one β-nf.

Normal order reduction: Deterministic strategy which chooses the leftmost, outermost redex, until there are no more redexes.

Theorem: A term M has a β-nf N iff the normal order reduction of M is finite and ends at N (this is an undecidable problem!).

Confluence

Theorem: (Church-Rosser) If $M \twoheadrightarrow_\beta N_1$ and $M \twoheadrightarrow_\beta N_2$, then there is a term P such that $N_1 \twoheadrightarrow_\beta P$ and $N_2 \twoheadrightarrow_\beta P$.

Corollary: Every term M has at most one β-nf.

Normal order reduction: Deterministic strategy which chooses the leftmost, outermost redex, until there are no more redexes.

Theorem: A term M has a β-nf N iff the normal order reduction of M is finite and ends at N (this is an undecidable problem!).

Structure of β-nfs: Every β-normal form M is of the form

$$\lambda x_1 \ldots x_n . y N_1 \ldots N_m$$

with $n, m \geq 0$ and such that $N_1, \ldots, N_m$ are terms in β-normal form.

η-reduction

η-reduction

- a term of the form $\lambda x.Mx$, such that $x \notin FV(M)$, is called an *η-redex*;

η-reduction

- a term of the form $\lambda x.Mx$, such that $x \notin FV(M)$, is called an η-*redex*;
- its *contractum* is the term M;

η-reduction

- a term of the form $\lambda x.Mx$, such that $x \notin FV(M)$, is called an *η-redex*;
- its *contractum* is the term M;
- $\rightarrow_{1\eta}$, $\rightarrow_{\eta}$ and $\equiv_{\eta}$;

η-reduction

- a term of the form $\lambda x.Mx$, such that $x \notin FV(M)$, is called an *η-redex*;
- its *contractum* is the term M;
- $\rightarrow_{1\eta}$, $\rightarrow_{\eta}$ and $\equiv_{\eta}$;
- all η-reductions are finite;

η-reduction

- a term of the form $\lambda x.Mx$, such that $x \notin FV(M)$, is called an *η-redex*;
- its *contractum* is the term M;
- $\rightarrow_{1\eta}$, $\rightarrow_{\eta}$ and $\equiv_{\eta}$;
- all η-reductions are finite;
- Church-Rosser;

η-reduction

- a term of the form $\lambda x.Mx$, such that $x \notin FV(M)$, is called an *η-redex*;
- its *contractum* is the term M;
- $\rightarrow_{1\eta}$, $\rightarrow_{\eta}$ and $\equiv_{\eta}$;
- all η-reductions are finite;
- Church-Rosser;
- every term has exactly one η-nf;

η-reduction

- a term of the form $\lambda x.Mx$, such that $x \notin FV(M)$, is called an η-*redex*;
- its *contractum* is the term M;
- $\rightarrow_{1\eta}$, $\rightarrow_{\eta}$ and $\equiv_{\eta}$;
- all η-reductions are finite;
- Church-Rosser;
- every term has exactly one η-nf;
- the η-*family* of a term M is the (finite) set of all terms N such that $M \rightarrow_{\eta} N$.

$\beta\eta$-reduction

$\beta\eta$-reduction

- a $\beta\eta$-redex is any β- or η-redex;

$\beta\eta$-reduction

- a $\beta\eta$-redex is any β- or η-redex;
- $\longrightarrow_{1\beta\eta}$, $\longrightarrow_{\beta\eta}$ and $\equiv_{\beta\eta}$;

$\beta\eta$-reduction

- a $\beta\eta$-redex is any β- or η-redex;
- $\longrightarrow_{1\beta\eta}$, $\longrightarrow_{\beta\eta}$ and $\equiv_{\beta\eta}$;
- Church-Rosser;

$\beta\eta$-reduction

- a $\beta\eta$-redex is any β- or η-redex;
- $\longrightarrow_{1\beta\eta}$, $\longrightarrow_{\beta\eta}$ and $\equiv_{\beta\eta}$;
- Church-Rosser;
- every term has at most one $\beta\eta$-nf;

$\beta\eta$-reduction

- a $\beta\eta$-redex is any β- or η-redex;
- $\longrightarrow_{1\beta\eta}$, $\longrightarrow_{\beta\eta}$ and $\equiv_{\beta\eta}$;
- Church-Rosser;
- every term has at most one $\beta\eta$-nf;
- if M is a β-nf, then all members of its η-family are β-nfs and exactly one of them is a $\beta\eta$-nf.

λ-definability

λ-definability

Notation: $F^n X = \underbrace{F(F(\ldots(F}_{n} X)\ldots))$

- **Church numerals:** $c_n = \lambda fx.f^n x$, for $n \geq 0$;

λ-definability

Notation: $F^n X = \underbrace{F(F(\ldots(F}_{n} X)\ldots))$

- **Church numerals:** $c_n = \lambda fx.f^n x$, for $n \geq 0$;
- $A_+ = \lambda mnfx.mf(nfx)$;

λ-definability

Notation: $F^n X = \underbrace{F(F(\ldots(F}_{n} X)\ldots))$

- **Church numerals:** $c_n = \lambda fx.f^n x$, for $n \geq 0$;
- $A_+ = \lambda mnfx.mf(nfx)$;

(show that $A_+ c_n c_m \equiv c_{n+m}$)

λ-definability

Notation: $F^n X = \underbrace{F(F(\ldots(F}_{n} X)\ldots))$

- **Church numerals:** $c_n = \lambda fx.f^n x$, for $n \geq 0$;
- $A_+ = \lambda mnfx.mf(nfx)$;

(show that $A_+ c_n c_m \equiv c_{n+m}$)

- $A_* = \lambda mnfx.m(nf)x$;

λ-definability

Notation: $F^n X = \underbrace{F(F(\ldots(F}_{n} X)\ldots))$

- **Church numerals:** $c_n = \lambda fx.f^n x$, for $n \geq 0$;
- $A_+ = \lambda mnfx.mf(nfx)$;

(show that $A_+ c_n c_m \equiv c_{n+m}$)

- $A_* = \lambda mnfx.m(nf)x$;

(show that $A_* c_n c_m \equiv c_{n*m}$)

λ-definability

Notation: $F^n X = \underbrace{F(F(\ldots(F}_{n} X)\ldots))$

- **Church numerals:** $c_n = \lambda fx.f^n x$, for $n \geq 0$;
- $A_+ = \lambda mnfx.mf(nfx)$;

  (show that $A_+ c_n c_m \equiv c_{n+m}$)
- $A_* = \lambda mnfx.m(nf)x$;

  (show that $A_* c_n c_m \equiv c_{n*m}$)
- $A_{exp} = \lambda mnfx.nmfx$;

λ-definability

Notation: $F^n X = \underbrace{F(F(\dots(F}_{n} X)\dots))$

- **Church numerals:** $c_n = \lambda fx.f^n x$, for $n \geq 0$;
- $A_+ = \lambda mnfx.mf(nfx)$;

  (show that $A_+ c_n c_m \equiv c_{n+m}$)
- $A_* = \lambda mnfx.m(nf)x$;

  (show that $A_* c_n c_m \equiv c_{n*m}$)
- $A_{exp} = \lambda mnfx.nmfx$;

  (show that $A_{exp} c_n c_m \equiv c_{n^m}$)

λ-definability (cont.)

λ-definability (cont.)

Booleans

λ-definability (cont.)

Booleans

- $\texttt{true} = \lambda xy.x$;

λ-definability (cont.)

Booleans

- $\texttt{true} = \lambda xy.x$;
- $\texttt{false} = \lambda xy.y$;

λ-definability (cont.)

Booleans

- `true` $= \lambda xy.x$;
- `false` $= \lambda xy.y$;
- `if` $= \lambda bxy.bxy$;

λ-definability (cont.)

Booleans

- $\texttt{true} = \lambda xy.x$;
- $\texttt{false} = \lambda xy.y$;
- $\texttt{if} = \lambda bxy.bxy$;

(show that $\texttt{if}\,\texttt{true}\,M\,N \equiv M$ and $\texttt{if}\,\texttt{false}\,M\,N \equiv N$)

λ-definability (cont.)

Booleans

- $\mathtt{true} = \lambda xy.x$;
- $\mathtt{false} = \lambda xy.y$;
- $\mathtt{if} = \lambda bxy.bxy$;

(show that $\mathtt{if}\,\mathtt{true}\,M\,N \equiv M$ and $\mathtt{if}\,\mathtt{false}\,M\,N \equiv N$)

Ordered pairs

λ-definability (cont.)

Booleans

- $\texttt{true} = \lambda xy.x$;
- $\texttt{false} = \lambda xy.y$;
- $\texttt{if} = \lambda bxy.bxy$;

(show that $\texttt{if}\,\texttt{true}\,M\,N \equiv M$ and $\texttt{if}\,\texttt{false}\,M\,N \equiv N$)

Ordered pairs

- $\texttt{pair} = \lambda xyf.fxy$;

λ-definability (cont.)

Booleans

- $\texttt{true} = \lambda xy.x$;
- $\texttt{false} = \lambda xy.y$;
- $\texttt{if} = \lambda bxy.bxy$;

(show that $\texttt{if}\,\texttt{true}\,M\,N \equiv M$ and $\texttt{if}\,\texttt{false}\,M\,N \equiv N$)

Ordered pairs

- $\texttt{pair} = \lambda xyf.fxy$;
- $\texttt{fst} = \lambda p.p\,\texttt{true}$;

λ-definability (cont.)

Booleans

- $\texttt{true} = \lambda xy.x$;
- $\texttt{false} = \lambda xy.y$;
- $\texttt{if} = \lambda bxy.bxy$;

(show that $\texttt{if}\,\texttt{true}\,M\,N \equiv M$ and $\texttt{if}\,\texttt{false}\,M\,N \equiv N$)

Ordered pairs

- $\texttt{pair} = \lambda xyf.fxy$;
- $\texttt{fst} = \lambda p.p\,\texttt{true}$;
- $\texttt{snd} = \lambda p.p\,\texttt{false}$;

λ-definability (cont.)

Booleans

- $\texttt{true} = \lambda xy.x$;
- $\texttt{false} = \lambda xy.y$;
- $\texttt{if} = \lambda bxy.bxy$;

(show that $\texttt{if}\,\texttt{true}\,M\,N \equiv M$ and $\texttt{if}\,\texttt{false}\,M\,N \equiv N$)

Ordered pairs

- $\texttt{pair} = \lambda xyf.fxy$;
- $\texttt{fst} = \lambda p.p\,\texttt{true}$;
- $\texttt{snd} = \lambda p.p\,\texttt{false}$;

(show that $\texttt{fst}\,(\texttt{pair}\,M\,N) \equiv M$ and ...)

λ-definability (cont.)

λ-definability (cont.)

- $\texttt{iszero} = \lambda n.n(\lambda x.\texttt{false})\texttt{true}$;

λ-definability (cont.)

- $\texttt{iszero} = \lambda n.n(\lambda x.\texttt{false})\texttt{true}$;
- $\texttt{suc} = \lambda nfx.f(nfx)$;

λ-definability (cont.)

- $\mathtt{iszero} = \lambda n.n(\lambda x.\mathtt{false})\mathtt{true}$;
- $\mathtt{suc} = \lambda nfx.f(nfx)$;
- $\mathtt{prefn} = \lambda fp.\mathtt{pair}(f(\mathtt{fst}\, p))(\mathtt{fst}\, p)$;

λ-definability (cont.)

- $\texttt{iszero} = \lambda n.n(\lambda x.\texttt{false})\texttt{true}$;
- $\texttt{suc} = \lambda nfx.f(nfx)$;
- $\texttt{prefn} = \lambda fp.\texttt{pair}(f(\texttt{fst}\,p))(\texttt{fst}\,p)$;
- $\texttt{pre} = \lambda nfx.\texttt{snd}(n(\texttt{prefn}\,f)(\texttt{pair}\,xx))$;

λ-definability (cont.)

- $\texttt{iszero} = \lambda n.n(\lambda x.\texttt{false})\texttt{true}$;
- $\texttt{suc} = \lambda nfx.f(nfx)$;
- $\texttt{prefn} = \lambda fp.\texttt{pair}(f(\texttt{fst}\,p))(\texttt{fst}\,p)$;
- $\texttt{pre} = \lambda nfx.\texttt{snd}(n(\texttt{prefn}\,f)(\texttt{pair}\,xx))$;
- $\texttt{sub} = \lambda mn.n\,\texttt{pre}\,m$;

λ-definability (cont.)

- $\mathtt{iszero} = \lambda n.n(\lambda x.\mathtt{false})\mathtt{true}$;
- $\mathtt{suc} = \lambda nfx.f(nfx)$;
- $\mathtt{prefn} = \lambda fp.\mathtt{pair}(f(\mathtt{fst}\,p))(\mathtt{fst}\,p)$;
- $\mathtt{pre} = \lambda nfx.\mathtt{snd}(n(\mathtt{prefn}\,f)(\mathtt{pair}\,xx))$;
- $\mathtt{sub} = \lambda mn.n\,\mathtt{pre}\,m$;

Lists

λ-definability (cont.)

- $\mathtt{iszero} = \lambda n.n(\lambda x.\mathtt{false})\mathtt{true}$;
- $\mathtt{suc} = \lambda nfx.f(nfx)$;
- $\mathtt{prefn} = \lambda fp.\mathtt{pair}(f(\mathtt{fst}\,p))(\mathtt{fst}\,p)$;
- $\mathtt{pre} = \lambda nfx.\mathtt{snd}(n(\mathtt{prefn}\,f)(\mathtt{pair}\,xx))$;
- $\mathtt{sub} = \lambda mn.n\,\mathtt{pre}\,m$;

Lists

- $\mathtt{nil} = \lambda z.z$;

λ-definability (cont.)

- $\mathtt{iszero} = \lambda n.n(\lambda x.\mathtt{false})\mathtt{true}$;
- $\mathtt{suc} = \lambda nfx.f(nfx)$;
- $\mathtt{prefn} = \lambda fp.\mathtt{pair}(f(\mathtt{fst}\,p))(\mathtt{fst}\,p)$;
- $\mathtt{pre} = \lambda nfx.\mathtt{snd}(n(\mathtt{prefn}\,f)(\mathtt{pair}\,xx))$;
- $\mathtt{sub} = \lambda mn.n\,\mathtt{pre}\,m$;

Lists

- $\mathtt{nil} = \lambda z.z$;
- $\mathtt{cons} = \lambda xy.\mathtt{pair}\,\mathtt{false}\,(\mathtt{pair}\,xy)$;

λ-definability (cont.)

- $\texttt{iszero} = \lambda n.n(\lambda x.\texttt{false})\texttt{true}$;
- $\texttt{suc} = \lambda nfx.f(nfx)$;
- $\texttt{prefn} = \lambda fp.\texttt{pair}(f(\texttt{fst}\,p))(\texttt{fst}\,p)$;
- $\texttt{pre} = \lambda nfx.\texttt{snd}(n(\texttt{prefn}\,f)(\texttt{pair}\,xx))$;
- $\texttt{sub} = \lambda mn.n\,\texttt{pre}\,m$;

Lists

- $\texttt{nil} = \lambda z.z$;
- $\texttt{cons} = \lambda xy.\texttt{pair}\,\texttt{false}\,(\texttt{pair}\,xy)$;
- $\texttt{null} = \texttt{fst}$;

λ-definability (cont.)

- $\texttt{iszero} = \lambda n.n(\lambda x.\texttt{false})\texttt{true}$;
- $\texttt{suc} = \lambda nfx.f(nfx)$;
- $\texttt{prefn} = \lambda fp.\texttt{pair}(f(\texttt{fst}\,p))(\texttt{fst}\,p)$;
- $\texttt{pre} = \lambda nfx.\texttt{snd}(n(\texttt{prefn}\,f)(\texttt{pair}\,xx))$;
- $\texttt{sub} = \lambda mn.n\,\texttt{pre}\,m$;

Lists

- $\texttt{nil} = \lambda z.z$;
- $\texttt{cons} = \lambda xy.\texttt{pair}\,\texttt{false}\,(\texttt{pair}\,xy)$;
- $\texttt{null} = \texttt{fst}$;
- $\texttt{hd} = \lambda z.\texttt{fst}\,(\texttt{snd}\,z)$;

λ-definability (cont.)

- $\texttt{iszero} = \lambda n.n(\lambda x.\texttt{false})\texttt{true}$;
- $\texttt{suc} = \lambda nfx.f(nfx)$;
- $\texttt{prefn} = \lambda fp.\texttt{pair}(f(\texttt{fst}\, p))(\texttt{fst}\, p)$;
- $\texttt{pre} = \lambda nfx.\texttt{snd}(n(\texttt{prefn}\, f)(\texttt{pair}\, xx))$;
- $\texttt{sub} = \lambda mn.n\, \texttt{pre}\, m$;

Lists

- $\texttt{nil} = \lambda z.z$;
- $\texttt{cons} = \lambda xy.\texttt{pair}\, \texttt{false}\, (\texttt{pair}\, xy)$;
- $\texttt{null} = \texttt{fst}$;
- $\texttt{hd} = \lambda z.\texttt{fst}\, (\texttt{snd}\, z)$;
- $\texttt{tl} = \lambda z.\texttt{snd}\, (\texttt{snd}\, z)$.

λ-definability (cont.)

λ-definability (cont.)

Recursive Functions

- $\mathbf{Y}$ is a fixed point operator iff $\mathbf{Y}F \equiv F(\mathbf{Y}F)$ for all terms F;

λ-definability (cont.)

Recursive Functions

- $\mathbf{Y}$ is a fixed point operator iff $\mathbf{Y}F \equiv F(\mathbf{Y}F)$ for all terms F;
- show that $\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ is a fixed point operator (there are many others!);

λ-definability (cont.)

Recursive Functions

- $\mathbf{Y}$ is a fixed point operator iff $\mathbf{Y}F \equiv F(\mathbf{Y}F)$ for all terms F;
- show that $\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ is a fixed point operator (there are many others!);
- show that $Mx_1 \ldots x_n \equiv PM$ is satisfied by defining $M = \mathbf{Y}(\lambda g x_1 \ldots x_n.Pg)$, whenever $\mathbf{Y}$ is a fixed point operator;

λ-definability (cont.)

Recursive Functions

- $\mathbf{Y}$ is a fixed point operator iff $\mathbf{Y}F \equiv F(\mathbf{Y}F)$ for all terms F;
- show that $\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ is a fixed point operator (there are many others!);
- show that $Mx_1 \ldots x_n \equiv PM$ is satisfied by defining $M = \mathbf{Y}(\lambda g x_1 \ldots x_n.Pg)$, whenever $\mathbf{Y}$ is a fixed point operator;
- define the functions `fact` and `tail`.

Restricted classes of λ-terms

Restricted classes of λ-terms

- M is a λI*-term* iff for every subterm of the form $\lambda x.N$ of M, x occurs *at least once* free in N;

Restricted classes of λ-terms

- M is a *λI-term* iff for every subterm of the form $\lambda x.N$ of M, x occurs *at least once* free in N;
- M is a *BCK-term* iff for every subterm of the form $\lambda x.N$ of M, x occurs *at most once* free in N;

Restricted classes of λ-terms

- M is a *λI-term* iff for every subterm of the form $\lambda x.N$ of M, x occurs *at least once* free in N;
- M is a *BCK-term* iff for every subterm of the form $\lambda x.N$ of M, x occurs *at most once* free in N;
- M is a *BCI-term* iff for every subterm of the form $\lambda x.N$ of M, x occurs *exactly once* free in N.

Simple Types

Simple Types

- infinite set of *type-variables* a, b, c, $\ldots$;

Simple Types

- infinite set of *type-variables* a, b, c, . . . ;
  - each type-variable a is a type (atomic);

Simple Types

- infinite set of *type-variables* a, b, c, $\ldots$;
  - each type-variable a is a type (atomic);
  - if α and β are type, then $(\alpha \to \beta)$ is a type.

Simple Types

- infinite set of *type-variables* a, b, c, $\ldots$;
  - each type-variable a is a type (atomic);
  - if α and β are type, then $(\alpha \to \beta)$ is a type.

Convention: $\to$ associates to the right

Simple Types

- infinite set of *type-variables* a, b, c, $\ldots$;
  - each type-variable a is a type (atomic);
  - if α and β are type, then $(\alpha \to \beta)$ is a type.

Convention: $\to$ associates to the right

$$a \to b \to c \to d$$

stands for

$$(a \to (b \to (c \to d)))$$

Simple Types

- infinite set of *type-variables* a, b, c, $\ldots$;
  - each type-variable a is a type (atomic);
  - if α and β are type, then $(\alpha \rightarrow \beta)$ is a type.

Convention: $\rightarrow$ associates to the right

$$a \rightarrow b \rightarrow c \rightarrow d$$

stands for

$$(a \rightarrow (b \rightarrow (c \rightarrow d)))$$

Examples: a, $a \rightarrow a$, $((a \rightarrow b) \rightarrow a) \rightarrow a$

Type-assignment

Type-assignment

- an expression $M : \alpha$ is a *type-assignment* (M is called its subject);

Type-assignment

- an expression $M : \alpha$ is a *type-assignment* (M is called its subject);
- a *type-context* is a finite, perhaps empty, set of type-assignments

$$\Gamma = \{x_1 : \alpha_1, \ldots, x_n : \alpha_n\},$$

such that $x_1, \ldots, x_n$ are <u>distinct</u> term-variables.

The system $\lambda{\rightarrow}$-Curry

The system $\lambda\!\rightarrow$-Curry

We say that the type-assignment $M : \tau$ is *derivable from a context* Γ, and write

$$\Gamma \vdash M : \tau,$$

iff the formula $\Gamma \vdash M : \tau$ can be produced by the following rules.

The system $\lambda\!\rightarrow$-Curry

We say that the type-assignment $M : \tau$ is *derivable from a context* Γ, and write

$$\Gamma \vdash M : \tau,$$

iff the formula $\Gamma \vdash M : \tau$ can be produced by the following rules.

(axiom) $\dfrac{}{\Gamma \;\vdash\; x : \alpha}$ (if $x : \alpha \in \Gamma$)

The system $\lambda\!\rightarrow$-Curry

We say that the type-assignment $M : \tau$ is *derivable from a context* Γ, and write

$$\Gamma \vdash M : \tau,$$

iff the formula $\Gamma \vdash M : \tau$ can be produced by the following rules.

$$\text{(axiom)} \quad \frac{}{\Gamma \;\vdash\; x : \alpha} \qquad (\text{if } x : \alpha \in \Gamma)$$

$$\text{(app)} \quad \frac{\Gamma \vdash M : \alpha \to \beta \qquad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta}$$

The system $\lambda\!\rightarrow$-Curry

We say that the type-assignment $M : \tau$ is *derivable from a context* Γ, and write

$$\Gamma \vdash M : \tau,$$

iff the formula $\Gamma \vdash M : \tau$ can be produced by the following rules.

$$\text{(axiom)} \quad \frac{}{\Gamma \;\vdash\; x : \alpha} \qquad (\text{if } x : \alpha \in \Gamma)$$

$$\text{(app)} \quad \frac{\Gamma \vdash M : \alpha \to \beta \qquad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta}$$

$$\text{(abs)} \quad \frac{\Gamma, x : \alpha \;\vdash\; M : \beta}{\Gamma \;\vdash\; \lambda x.M : \alpha \to \beta}$$

The system $\lambda\!\rightarrow$-Curry

We say that the type-assignment $M : \tau$ is *derivable from a context* Γ, and write

$$\Gamma \vdash M : \tau,$$

iff the formula $\Gamma \vdash M : \tau$ can be produced by the following rules.

$$\text{(axiom)} \quad \frac{}{\Gamma \vdash x : \alpha} \qquad (\text{if } x : \alpha \in \Gamma)$$

$$\text{(app)} \quad \frac{\Gamma \vdash M : \alpha \to \beta \qquad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta}$$

$$\text{(abs)} \quad \frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta}$$

A deduction Δ of $\Gamma \vdash M : \tau$ is a tree of formulae, those at the tops of branches being axioms and those below being deduced from those immediately above them by a rule ((app) or (abs)) and with bottom formula $\Gamma \vdash M : \tau$.

Related problems:

Related problems:

- *Type-checking:* Given Γ, M and τ, is it true that $\Gamma \vdash M : \tau$?

Related problems:

- *Type-checking:* Given Γ, M and τ, is it true that $\Gamma \vdash M : \tau$?
- *Typability:* Given M, are there Γ and τ such that $\Gamma \vdash M : \tau$? (M is said to be typable)

Related problems:

- *Type-checking:* Given Γ, M and τ, is it true that $\Gamma \vdash M : \tau$?
- *Typability:* Given M, are there Γ and τ such that $\Gamma \vdash M : \tau$? (M is said to be typable)
- *Inhabitation:* Given Γ and τ, is there M such that $\Gamma \vdash M : \tau$? (If $\Gamma = \emptyset$, we say that τ is inhabited; also M is called an inhabitant of τ)

Related problems:

- *Type-checking:* Given Γ, M and τ, is it true that $\Gamma \vdash M : \tau$?
- *Typability:* Given M, are there Γ and τ such that $\Gamma \vdash M : \tau$? (M is said to be typable)
- *Inhabitation:* Given Γ and τ, is there M such that $\Gamma \vdash M : \tau$? (If $\Gamma = \emptyset$, we say that τ is inhabited; also M is called an inhabitant of τ)

All these problems are decidable!

Exercises

1. Show that $\vdash \lambda x.x : a \rightarrow a$.
2. Show that $\vdash \lambda x.x : (a \rightarrow b) \rightarrow a \rightarrow b$.
3. Find Γ and α such that $\Gamma \vdash (\lambda xy.xy)z : \alpha$.
4. Find M such that $\vdash M : a \rightarrow b \rightarrow a$.
5. Find M such that $\vdash M : ((a \rightarrow b) \rightarrow a) \rightarrow a$.

Properties

Properties

- Confluence (Church-Rosser);

Properties

- Confluence (Church-Rosser);
- strong normalization;

Properties

- Confluence (Church-Rosser);
- strong normalization;
- existence of unique normal forms;

Properties

- Confluence (Church-Rosser);
- strong normalization;
- existence of unique normal forms;
- subject-reduction;

Properties

- Confluence (Church-Rosser);
- strong normalization;
- existence of unique normal forms;
- subject-reduction;
- principal types;

The system $\lambda\!\rightarrow$-Church

- term-variables annotated with types: x^{α}, x^{β}, $\ldots y^{\alpha}$, $\ldots$;

The system $\lambda{\rightarrow}$-Church

- term-variables annotated with types: x^{α}, x^{β}, $\ldots y^{\alpha}$, $\ldots$;
  - each annotated variable x^{α} is a λ-term of type α;

The system $\lambda\!\rightarrow$-Church

- term-variables annotated with types: x^{α}, x^{β}, $\ldots y^{\alpha}$, $\ldots$;
  - each annotated variable x^{α} is a λ-term of type α;
  - if M and N are λ-terms, respectively of type $\alpha \to \beta$ and α, then (MN) is a λ-term of type β, (*application*);

The system $\lambda\!\rightarrow$-Church

- term-variables annotated with types: x^α, x^β, $\dots y^\alpha$, $\dots$;
  - each annotated variable x^α is a λ-term of type α;
  - if M and N are λ-terms, respectively of type $\alpha \to \beta$ and α, then (MN) is a λ-term of type β, (*application*);
  - if M is a λ-term of type β and x^α an annotated variable, then $(\lambda x^\alpha.M)$ is a λ-term of type $\alpha \to \beta$, (*abstraction*).

The system $\lambda\!\rightarrow$-Church

- term-variables annotated with types: x^{α}, x^{β}, $\ldots y^{\alpha}$, $\ldots$;
  - each annotated variable x^{α} is a λ-term of type α;
  - if M and N are λ-terms, respectively of type $\alpha \rightarrow \beta$ and α, then (MN) is a λ-term of type β, (*application*);
  - if M is a λ-term of type β and x^{α} an annotated variable, then $(\lambda x^{\alpha}.M)$ is a λ-term of type $\alpha \rightarrow \beta$, (*abstraction*).

Church vs. Curry Differences and similarities...