Working with Automated Reasoning Tools
– Typed Lambda Calculus –

Christoph Benzmüller and Geoff Sutcliffe

http://www.ags.uni-sb.de/~chris/

TPTPSYS'08

SS08, Block Course at Saarland University, Germany

UNIVERSITÄT
DES
SAARLANDES

λ-Calculus: Review

λ-Calculus: Review

λ-Calculus: Review

Consider the following arithmetical computations

$$
\begin{aligned}
(-1)^2 - 1 &= 0 \\
(1)^2 - 1 &= 0 \\
(2)^2 - 1 &= 3 \\
&\ldots
\end{aligned}
$$

λ-Calculus: Review

Consider the following arithmetical computations

$$
\begin{aligned}
(-1)^2 - 1 &= 0 \\
(1)^2 - 1 &= 0 \\
(2)^2 - 1 &= 3 \\
&\ldots
\end{aligned}
$$

A more general arithmetic expression for the LHS:

$$x^2 - 1$$

λ-Calculus: Motivation

Consider the 0's (Nullstellen) of this function; we can express the existence of two 0's in first-order logic as follows

$$\exists n, m. n^2 - 1 = 0 \wedge m^2 - 1 = 0 \wedge n \neq m$$

λ-Calculus: Motivation

Consider the 0's (Nullstellen) of this function; we can express the existence of two 0's in first-order logic as follows

$$\exists n, m. n^2 - 1 = 0 \wedge m^2 - 1 = 0 \wedge n \neq m$$

Now we may want to talk about the existence of a function f with two 0's:

$$(1) \quad \exists f. \exists n, m. f(n) = 0 \wedge f(m) = 0 \wedge n \neq m$$

λ-Calculus: Motivation

Consider the 0's (Nullstellen) of this function; we can express the existence of two 0's in first-order logic as follows

$$\exists n, m. n^2 - 1 = 0 \land m^2 - 1 = 0 \land n \neq m$$

Now we may want to talk about the existence of a function f with two 0's:

$$(1) \quad \exists f. \exists n, m. f(n) = 0 \land f(m) = 0 \land n \neq m$$

This expression is not a first-order statement; however we want to be able to express such statements. We also want to prove such statements and in a constructive proof we would like to provide witnesses for f and n, m. In first-order logic we can describe f by the following equation

$$f(x) = x^2 - 1$$

λ-Calculus: λ-terms

In λ-calculus the specified function f can be described (without giving it a name) by the witnessing λ-term

$$\mathsf{f} = (\lambda \mathsf{x}.\mathsf{x}^2 - 1)$$

and the witnesses for n and m are -1 and 1.

λ-Calculus: Set of λ-expressions

Given a countably infinite set of identifiers, say $a, b, c, ..., x, y, z, x1, x2, ...$ The set of all λ-expressions can then be described by the following context-free grammar in BNF:

1. <expr> ::= <identifier>

λ-Calculus: Set of λ-expressions

Given a countably infinite set of identifiers, say $a, b, c, ..., x, y, z, x1, x2, ...$ The set of all λ-expressions can then be described by the following context-free grammar in BNF:

1. <expr> ::= <identifier>
2. <expr> ::= (λ <identifier> . <expr>) abstraction

λ-Calculus: Set of λ-expressions

Given a countably infinite set of identifiers, say $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots, \mathsf{x}, \mathsf{y}, \mathsf{z}, \mathsf{x1}, \mathsf{x2}, \ldots$. The set of all λ-expressions can then be described by the following context-free grammar in BNF:

1. <expr> ::= <identifier>
2. <expr> ::= (λ <identifier> . <expr>) — abstraction
3. <expr> ::= (<expr> <expr>) — application

λ-Calculus: Conventions

We often omit brackets with the following conventions:

- $(F\,A\,B)$ means $((F\,A)\,B)$. (Application associates to the left.)

λ-Calculus: Conventions

We often omit brackets with the following conventions:

- $(F\,A\,B)$ means $((F\,A)\,B)$. (Application associates to the left.)
- $(\lambda x.\lambda y.\,B)$ means $(\lambda x.(\lambda y.\,B))$.

λ-Calculus: Conventions

We often omit brackets with the following conventions:

- $(F\,A\,B)$ means $((F\,A)\,B)$. (Application associates to the left.)
- $(\lambda x.\lambda y.\,B)$ means $(\lambda x.(\lambda y.\,B))$.
- A dot stands for a left bracket whose mate is as far to the right as possible without changing the existing bracketing.

λ-Calculus: β-reduction

Consider now the instantiation of (1) with these witness terms

$$\exists f. \exists n, m. f(n) = 0 \wedge f(m) = 0 \wedge n \neq m$$

λ-Calculus: β-reduction

Consider now the instantiation of (1) with these witness terms

$$\exists f.\exists n,m.f(n) = 0 \land f(m) = 0 \land n \neq m$$

$$\xrightarrow{f} \exists n,m.((\lambda x.x^2 - 1)\, n) = 0 \land ((\lambda x.x^2 - 1)\, m) = 0 \land n \neq m$$

λ-Calculus: β-reduction

Consider now the instantiation of (1) with these witness terms

$$
\begin{aligned}
& \exists f. \exists n, m. f(n) = 0 \land f(m) = 0 \land n \neq m \\
\stackrel{f}{\longrightarrow}\ & \exists n, m. ((\lambda x. x^2 - 1)\, n) = 0 \land ((\lambda x. x^2 - 1)\, m) = 0 \land n \neq m \\
\stackrel{n,m}{\longrightarrow}\ & ((\lambda x. x^2 - 1)\, (-1)) = 0 \land ((\lambda x. x^2 - 1)\, 1) = 0 \land -1 \neq 1
\end{aligned}
$$

λ-Calculus: β-reduction

Consider now the instantiation of (1) with these witness terms

$$\begin{aligned}
& \exists f.\exists n, m.f(n) = 0 \land f(m) = 0 \land n \neq m \\
\stackrel{f}{\longrightarrow}\ & \exists n, m.((\lambda x.x^2 - 1)\, n) = 0 \land ((\lambda x.x^2 - 1)\, m) = 0 \land n \neq m \\
\stackrel{n,m}{\longrightarrow}\ & ((\lambda x.x^2 - 1)\, (-1)) = 0 \land ((\lambda x.x^2 - 1)\, 1) = 0 \land -1 \neq 1
\end{aligned}$$

Finally we can ‘evaluate’ function applications by so called β-reduction

$$((-1)^2 - 1) = 0 \land (1^2 - 1) = 0 \land -1 \neq 1$$

λ-Calculus: β-reduction

The β-reduction rule expresses the idea of function application as motivated on the previous slide. Formally it states that

$$((\lambda x. A)\, B) \longrightarrow_{\beta} A[x/B]$$

if all free occurrences in B remain free in $A[x/B]$. Here, $A[x/B]$ means the expression E with every free occurrence of x in A replaced with B.

λ-Calculus: Currying

A function of two variables is expressed in lambda calculus as a function of one argument which returns a function of one argument. For instance, the function

$$f(x, y) = x^2 - y$$

is encoded as

$$(\lambda x.\lambda y.x^2 - y)$$

λ-Calculus: α-conversion

The names of the bound variables are unimportant:

$$\lambda x.x^2 - 1 \text{ and } \lambda y.y^2 - 1$$

denote the same function.

λ-Calculus: α-conversion

The names of the bound variables are unimportant:

$$\lambda x.x^2 - 1 \text{ and } \lambda y.y^2 - 1$$

denote the same function.
Formally, the α-conversion rule states that if x and y are variables and A is a λ-expression then

$$(\lambda x.A) \longleftrightarrow_\alpha (\lambda y.A[x/y])$$

if y does not appear freely in A and y is not bound by a λ in A whenever it replaces a x.

λ-Calculus: η-reduction

η-reduction expresses the idea of (functional) extensionality, which in this context is that two functions are the same iff they give the same result for all arguments:

$$(\lambda x.Fx) \longrightarrow_\eta F$$

whenever x does not appear free in F.

λ-Calculus: $\beta\eta$-equivalence

- We define $\longleftrightarrow^*_{\alpha\beta\eta}$ as the smallest equivalence relation closed under the reduction rules $\longrightarrow_\beta$ and $\longrightarrow_\eta$ and α-conversion. (Similarly we may define $\longleftrightarrow^*_{\mathsf{M}}$ for M $\subset \{\alpha, \beta, \eta\}$)

λ-Calculus: $\beta\eta$-equivalence

- We define $\longleftrightarrow^*_{\alpha\beta\eta}$ as the smallest equivalence relation closed under the reduction rules $\longrightarrow_\beta$ and $\longrightarrow_\eta$ and α-conversion. (Similarly we may define $\longleftrightarrow^*_{\mathsf{M}}$ for M $\subset \{\alpha, \beta, \eta\}$)

- We call two λ-terms E and T $\alpha\beta\eta$-equivalent (or short equivalent) if

$$\mathsf{E} \longleftrightarrow^*_{\alpha\beta\eta} \mathsf{T}$$

λ-Calculus: $\beta\eta$-equivalence

- We define $\longleftrightarrow^*_{\alpha\beta\eta}$ as the smallest equivalence relation closed under the reduction rules $\longrightarrow_\beta$ and $\longrightarrow_\eta$ and α-conversion. (Similarly we may define $\longleftrightarrow^*_{\mathsf{M}}$ for M $\subset \{\alpha, \beta, \eta\}$)

- We call two λ-terms E and T $\alpha\beta\eta$-equivalent (or short equivalent) if

$$\mathsf{E} \longleftrightarrow^*_{\alpha\beta\eta} \mathsf{T}$$

(Similarly we may define M-equivalence for M $\subset \{\alpha, \beta, \eta\}$)

λ-Calculus: Normalforms

- A λ-expression is called a β-normal form if it does not allow any β-reduction, i.e., has no subexpression of the form

$$((\lambda \mathsf{x} \,.\, \mathsf{A})\, \mathsf{B})$$

λ-Calculus: Normalforms

- A λ-expression is called a β-normal form if it does not allow any β-reduction, i.e., has no subexpression of the form

$$((\lambda \mathsf{x}\ .\ \mathsf{A})\ \mathsf{B})$$

- A λ-expression is called a η-normal form if it does not allow any η-reduction, i.e., has no subexpression of the form (where x does not occur free in E)

$$(\lambda \mathsf{x}.\mathsf{E}\ \mathsf{x})$$

λ-Calculus: Normalforms

- A λ-expression is called a β-normal form if it does not allow any β-reduction, i.e., has no subexpression of the form

$$((\lambda x\ .\ A)\ B)$$

- A λ-expression is called a η-normal form if it does not allow any η-reduction, i.e., has no subexpression of the form (where x does not occur free in E)

$$(\lambda x.E\ x)$$

- A λ-expression is called a $\beta\eta$-normal form if it satisfies both conditions.

λ-Calculus: Normalforms

- Not every λ-expression is equivalent to a ?-normal form (where $? \in \{\beta, \beta\eta\}$)

λ-Calculus: Normalforms

- Not every λ-expression is equivalent to a ?-normal form (where $? \in \{\beta, \beta\eta\}$)
- The Church-Rosser theorem(s) state that if $A \longrightarrow^{?*} B$ and $A \longrightarrow^{?*} C$, then there is some D such that $B \longrightarrow^{?*} D$ and $C \longrightarrow^{?*} D$.

λ-Calculus: Normalforms

- Not every λ-expression is equivalent to a ?-normal form (where $? \in \{\beta, \beta\eta\}$)
- The Church-Rosser theorem(s) state that if A $\longrightarrow^{?*}$ B and A $\longrightarrow^{?*}$ C, then there is some D such that B $\longrightarrow^{?*}$ D and C $\longrightarrow^{?*}$ D.
- From Church-Rosser it follows that every term has at most one $?$-normal form (up to α-conversion).

λ-Calculus: Iteration

Consider twofold iteration of function $\mathsf{f} := (\lambda \mathsf{x}.\mathsf{x}^2 - 1)$

$$\mathsf{f}(\mathsf{f}(\mathsf{x})) = (\mathsf{x}^2 - 1)^2 - 1 = \mathsf{x}^4 - 2\mathsf{x}^2$$

λ-Calculus: Iteration

Consider twofold iteration of function $\mathsf{f} := (\lambda \mathsf{x}.\mathsf{x}^2 - 1)$

$$\mathsf{f}(\mathsf{f}(\mathsf{x})) = (\mathsf{x}^2 - 1)^2 - 1 = \mathsf{x}^4 - 2\mathsf{x}^2$$

The following λ-term expresses twofold iteration of a function

$$(\lambda g.\lambda y.g\ (g\ y))$$

λ-Calculus: Iteration

Consider twofold iteration of function $\mathsf{f} := (\lambda \mathsf{x}.\mathsf{x}^2 - 1)$

$$\mathsf{f}(\mathsf{f}(\mathsf{x})) = (\mathsf{x}^2 - 1)^2 - 1 = \mathsf{x}^4 - 2\mathsf{x}^2$$

The following λ-term expresses twofold iteration of a function

$$(\lambda \mathsf{g}.\lambda \mathsf{y}.\mathsf{g}\ (\mathsf{g}\ \mathsf{y}))$$

Let us apply this λ-term now to our function f

$$((\lambda \mathsf{g}.\lambda \mathsf{y}.\mathsf{g}\ (\mathsf{g}\ \mathsf{y}))\ (\lambda \mathsf{x}.\mathsf{x}^2 - 1))$$

λ-Calculus: Iteration

Consider twofold iteration of function $\mathsf{f} := (\lambda \mathsf{x}.\mathsf{x}^2 - 1)$

$$\mathsf{f}(\mathsf{f}(\mathsf{x})) = (\mathsf{x}^2 - 1)^2 - 1 = \mathsf{x}^4 - 2\mathsf{x}^2$$

The following λ-term expresses twofold iteration of a function

$$(\lambda \mathsf{g}.\lambda \mathsf{y}.\mathsf{g}\ (\mathsf{g}\ \mathsf{y}))$$

Let us apply this λ-term now to our function f

$$\begin{aligned} &((\lambda \mathsf{g}.\lambda \mathsf{y}.\mathsf{g}\ (\mathsf{g}\ \mathsf{y}))\ (\lambda \mathsf{x}.\mathsf{x}^2 - 1)) \\ \longrightarrow_\beta\ &(\lambda \mathsf{y}.(\lambda \mathsf{x}.\mathsf{x}^2 - 1)((\lambda \mathsf{x}.\mathsf{x}^2 - 1)\mathsf{y})) \end{aligned}$$

λ-Calculus: Iteration

Consider twofold iteration of function $\mathsf{f} := (\lambda x.x^2 - 1)$

$$f(f(x)) = (x^2 - 1)^2 - 1 = x^4 - 2x^2$$

The following λ-term expresses twofold iteration of a function

$$(\lambda g.\lambda y.g\ (g\ y))$$

Let us apply this λ-term now to our function f

$$\begin{aligned}
&((\lambda g.\lambda y.g\ (g\ y))\ (\lambda x.x^2 - 1)) \\
\longrightarrow_\beta\ &(\lambda y.(\lambda x.x^2 - 1)((\lambda x.x^2 - 1)y)) \\
\longrightarrow_\beta\ &\lambda y.(\lambda x.x^2 - 1)\ (y^2 - 1)
\end{aligned}$$

λ-Calculus: Iteration

Consider twofold iteration of function $f := (\lambda x.x^2 - 1)$

$$f(f(x)) = (x^2 - 1)^2 - 1 = x^4 - 2x^2$$

The following λ-term expresses twofold iteration of a function

$$(\lambda g.\lambda y.g\ (g\ y))$$

Let us apply this λ-term now to our function f

$$\begin{aligned}
&((\lambda g.\lambda y.g\ (g\ y))\ (\lambda x.x^2 - 1)) \\
\longrightarrow_\beta\ &(\lambda y.(\lambda x.x^2 - 1)((\lambda x.x^2 - 1)y) \\
\longrightarrow_\beta\ &\lambda y.(\lambda x.x^2 - 1)\ (y^2 - 1) \\
\longrightarrow_\beta\ &(\lambda y.(y^2 - 1)^2 - 1) = \lambda y.y^4 - 2y^2
\end{aligned}$$

λ-Calculus: Church Numerals

We employ iteration to define natural numbers as Church numerals:

$$\overline{0} = (\lambda f.\lambda x.x), \qquad \overline{1} = (\lambda f.\lambda x.fx), \qquad \overline{2} = (\lambda f.\lambda x.f(fx)), \qquad \ldots$$

λ-Calculus: Church Numerals

We employ iteration to define natural numbers as Church numerals:

$$\overline{0} = (\lambda f.\lambda x.x), \qquad \overline{1} = (\lambda f.\lambda x.fx), \qquad \overline{2} = (\lambda f.\lambda x.f(fx)), \qquad \ldots$$

Generally a natural number n is encoded as the Church numeral

$$\overline{n} = (\lambda f.\lambda y.f^n\, y)$$

where f^n is an abbreviation for $\underbrace{(f\ (f\ (f \ldots (f}_{n-\text{times}}\ y)))$.

λ-Calculus: Church Numerals

We employ iteration to define natural numbers as Church numerals:

$$\overline{0} = (\lambda f.\lambda x.x), \qquad \overline{1} = (\lambda f.\lambda x.fx), \qquad \overline{2} = (\lambda f.\lambda x.f(fx)), \qquad \ldots$$

Generally a natural number n is encoded as the Church numeral

$$\overline{n} = (\lambda f.\lambda y.f^n\, y)$$

where f^n is an abbreviation for $\underbrace{(f\ (f\ (f \ldots (f}_{n-\text{times}}\ y)))$.

Intuitively, the number n in lambda calculus is a function that takes a function f as argument and returns the n-th iterate of f.

λ-Calculus: Church Numerals

We can now define a successor function $\overline{\mathsf{SUCC}}$, which takes a number $\overline{\mathsf{n}}$ and returns $\overline{\mathsf{n}+1}$:

$$\overline{\mathsf{SUCC}} = (\lambda \mathsf{n}.\lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{f}(\mathsf{nfx}))$$

λ-Calculus: Church Numerals

We can now define a successor function $\overline{\mathsf{SUCC}}$, which takes a number $\overline{\mathsf{n}}$ and returns $\overline{\mathsf{n}+1}$:

$$\overline{\mathsf{SUCC}} = (\lambda \mathsf{n}.\lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{f}(\mathsf{nfx}))$$

Addition is the defined as follows:

$$\overline{\mathsf{PLUS}} = (\lambda \mathsf{m}.\lambda \mathsf{n}.\lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{mf}(\mathsf{nfx}))$$

λ-Calculus: Church Numerals

We can now define a successor function $\overline{\mathsf{SUCC}}$, which takes a number $\overline{\mathsf{n}}$ and returns $\overline{\mathsf{n}+1}$:

$$\overline{\mathsf{SUCC}} = (\lambda \mathsf{n}.\lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{f}(\mathsf{nfx}))$$

Addition is the defined as follows:

$$\overline{\mathsf{PLUS}} = (\lambda \mathsf{m}.\lambda \mathsf{n}.\lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{mf}(\mathsf{nfx}))$$

Multiplication can then be defined as

$$\overline{\mathsf{MULT}} = \lambda \mathsf{m}.\lambda \mathsf{n}.\mathsf{m}(\overline{\mathsf{PLUS}}\ \mathsf{n})\overline{0},$$

the idea being that multiplying m and n is the same as adding n to 0 m times.

λ-Calculus: Church Numerals

The predecessesor function is more difficult:

$$\overline{\mathsf{PRED}} = \lambda \mathsf{n}.\lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{n}(\lambda \mathsf{g}.\lambda \mathsf{h}.\mathsf{h}\ (\mathsf{g}\ \mathsf{f}))\ (\lambda \mathsf{u}.\mathsf{x})\ (\lambda \mathsf{u}.\mathsf{u})$$

λ-Calculus: Church Numerals

The predecessesor function is more difficult:

$$\overline{\mathsf{PRED}} = \lambda \mathsf{n}.\lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{n}(\lambda \mathsf{g}.\lambda \mathsf{h}.\mathsf{h}\ (\mathsf{g}\ \mathsf{f}))\ (\lambda \mathsf{u}.\mathsf{x})\ (\lambda \mathsf{u}.\mathsf{u})$$

or alternatively

$$\overline{\mathsf{PRED}} = \lambda \mathsf{n}.\mathsf{n}(\lambda \mathsf{g}.\lambda \mathsf{k}.(\mathsf{g}\ \overline{1})\ (\lambda \mathsf{u}.\overline{\mathsf{PLUS}}\ (\mathsf{g}\ \mathsf{k})\ \overline{1})\ \mathsf{k})\ (\lambda \mathsf{l}.\ \overline{0})\ \overline{0}$$

λ-Calculus: Church Numerals

The predecessesor function is more difficult:

$$\overline{\mathsf{PRED}} = \lambda n.\lambda f.\lambda x.n(\lambda g.\lambda h.h\ (g\ f))\ (\lambda u.x)\ (\lambda u.u)$$

or alternatively

$$\overline{\mathsf{PRED}} = \lambda n.n(\lambda g.\lambda k.(g\ \overline{1})\ (\lambda u.\overline{\mathsf{PLUS}}\ (g\ k)\ \overline{1})\ k)\ (\lambda l.\ \overline{0})\ \overline{0}$$

Note the trick $(g\overline{1})(\lambda u.\overline{\mathsf{PLUS}}(g\ k)\ \overline{1})k$ which evaluates to k if $(g\ \overline{1})$ is $\overline{0}$ and to $(g\ k) + \overline{1}$ otherwise.

λ-Calculus: Sets

$$\{\mathsf{x}|\mathsf{x}^2 - 1 = 0\}$$

$$(\{-1, 1\})$$

λ-Calculus: Sets

$$\{x | x^2 - 1 = 0\}$$

$$(\{-1, 1\})$$

The set A has two elements:

$$\exists A. \exists m, n. m \in A \wedge n \in A \wedge m \neq n$$

λ-Calculus: Sets

$$\{x|x^2 - 1 = 0\}$$

$$(\{-1, 1\})$$

The set A has two elements:

$$\exists A.\exists m, n.m \in A \land n \in A \land m \neq n$$

In first-order, A can be 'defined' by:

$$(x \in A) \equiv (x^2 - 1 = 0)$$

λ-Calculus: Sets

$$\{x | x^2 - 1 = 0\}$$

$$(\{-1, 1\})$$

The set A has two elements:

$$\exists A. \exists m, n. m \in A \wedge n \in A \wedge m \neq n$$

In first-order, A can be 'defined' by:

$$(x \in A) \equiv (x^2 - 1 = 0)$$

In this expression we talk about 'membership'

λ-Calculus: Sets

$$\{x|x^2 - 1 = 0\}$$

$$(\{-1, 1\})$$

The set A has two elements:

$$\exists A.\exists m, n.m \in A \wedge n \in A \wedge m \neq n$$

In first-order, A can be ’defined’ by:

$$(x \in A) \equiv (x^2 - 1 = 0)$$

In this expression we talk about ’membership’
Alternatively, we can express the characteristic function of A by the λ-term

$$(\lambda x.(x^2 - 1 = 0))$$

λ-Calculus: Sets

$$(\lambda x.x^2 - 1 = 0)$$

λ-Calculus: Sets

$$(\lambda x. x^2 - 1 = 0)$$

The idea is as follows

$((\lambda x. x^2 - 1 = 0)\, a)$ evaluates to $a^2 - 1 = 0$

λ-Calculus: Sets

$$(\lambda x.x^2 - 1 = 0)$$

The idea is as follows

$((\lambda x.x^2 - 1 = 0)\, a)$ evaluates to $a^2 - 1 = 0$

The expression $a^2 - 1 = 0$ is $\top$ ($\top$ denotes Truth) if a is -1 or 1.

λ-Calculus: Sets

$$(\lambda x.x^2 - 1 = 0)$$

The idea is as follows

$$((\lambda x.x^2 - 1 = 0)\, a) \text{ evaluates to } a^2 - 1 = 0$$

The expression $a^2 - 1 = 0$ is $\top$ ($\top$ denotes Truth) if a is -1 or 1. Otherwise, $a^2 - 1 = 0$ is $\bot$ ($\bot$ denotes Falsehood)

λ-Calculus: Sets

$$(\lambda x.x^2 - 1 = 0)$$

The idea is as follows

$$((\lambda x.x^2 - 1 = 0)\, a) \text{ evaluates to } a^2 - 1 = 0$$

The expression $a^2 - 1 = 0$ is $\top$ ($\top$ denotes Truth) if a is -1 or 1.
Otherwise, $a^2 - 1 = 0$ is $\bot$ ($\bot$ denotes Falsehood)
The characteristic function $(\lambda x.x^2 - 1 = 0)$ provides a witness for

$$\exists P.\exists m, n.\, (P\, m) \land (P\, n) \land m \neq n$$

λ-Calculus: Sets

For each natural number n there is a Church numeral:

$$\overline{n} = \lambda f.\lambda y.(f^n\, y)$$

λ-Calculus: Sets

For each natural number n there is a Church numeral:

$$\overline{\mathsf{n}} = \lambda \mathsf{f}.\lambda \mathsf{y}.(\mathsf{f}^{\mathsf{n}}\,\mathsf{y})$$

We can also define the *set* $\overline{\mathsf{N}}$ of all Church numerals

λ-Calculus: Sets

For each natural number n there is a Church numeral:

$$\overline{\mathsf{n}} = \lambda \mathsf{f}.\lambda \mathsf{y}.(\mathsf{f}^{\mathsf{n}}\,\mathsf{y})$$

We can also define the *set* $\overline{\mathsf{N}}$ of all Church numerals

$\overline{\mathsf{N}}$ must satisfy three properties:

λ-Calculus: Sets

For each natural number n there is a Church numeral:

$$\overline{n} = \lambda f.\lambda y.(f^n\, y)$$

We can also define the *set* $\overline{N}$ of all Church numerals

$\overline{N}$ must satisfy three properties:

1. $(\overline{N}\,\overline{0})$ “$\overline{0}$ is a Church numeral”

λ-Calculus: Sets

For each natural number n there is a Church numeral:

$$\overline{n} = \lambda f.\lambda y.(f^n\, y)$$

We can also define the *set* $\overline{N}$ of all Church numerals

$\overline{N}$ must satisfy three properties:

1. $(\overline{N}\,\overline{0})$ "$\overline{0}$ is a Church numeral"
2. $\forall x.(\overline{N}\, x) \supset (\overline{N}(\overline{SUCC}\, x))$ "$\overline{N}$ is closed under successor"

λ-Calculus: Sets

For each natural number n there is a Church numeral:

$$\overline{n} = \lambda f.\lambda y.(f^n\, y)$$

We can also define the *set* $\overline{N}$ of all Church numerals
$\overline{N}$ must satisfy three properties:

1. $(\overline{N}\,\overline{0})$ "$\overline{0}$ is a Church numeral"
2. $\forall x.(\overline{N}\,x) \supset (\overline{N}(\overline{SUCC}\,x))$ "$\overline{N}$ is closed under successor"
3. $\forall P.(P\,\overline{0}) \wedge (\forall x.(P\,x) \supset (P\,(\overline{SUCC}\,x))) \supset (\overline{N} \subseteq P)$
   "$\overline{N}$ is the least such set"

λ-Calculus: Sets

For each natural number n there is a Church numeral:

$$\overline{n} = \lambda f.\lambda y.(f^n\, y)$$

We can also define the *set* $\overline{N}$ of all Church numerals

$\overline{N}$ must satisfy three properties:

1. $(\overline{N}\,\overline{0})$ "$\overline{0}$ is a Church numeral"
2. $\forall x.(\overline{N}\,x) \supset (\overline{N}(\overline{SUCC}\,x))$ "$\overline{N}$ is closed under successor"
3. $\forall P.(P\,\overline{0}) \wedge (\forall x.(P\,x) \supset (P\,(\overline{SUCC}\,x))) \supset (\overline{N} \subseteq P)$
   "$\overline{N}$ is the least such set"

Define $\overline{N}$ to be:

$$\lambda z.\forall P.((P\,\overline{0}) \wedge (\forall x.\,(P\,x) \supset (P\,.\,\overline{SUCC}\,x))) \supset (P\,z)$$

λ-Calculus: Sets

Define $\overline{\mathsf{N}}$ to be:

$$\lambda z.\forall P.((P\,\overline{0}) \wedge (\forall x.\,(P\,x) \supset (P\,.\,\overline{SUCC}\,x))) \supset (P\,z)$$

λ-Calculus: Sets

Define $\overline{\mathsf{N}}$ to be:

$$\lambda z.\forall P.((P\,\overline{0}) \wedge (\forall x.\,(P\,x) \supset (P\,.\,\overline{SUCC}\,x))) \supset (P\,z)$$

This satisfies the three requirements.

λ-Calculus: Sets

Define $\overline{\mathsf{N}}$ to be:

$$\lambda z.\forall P.((P\,\overline{0}) \wedge (\forall x.\,(P\,x) \supset (P\,.\,\overline{SUCC}\,x))) \supset (P\,z)$$

This satisfies the three requirements.

- $(\overline{\mathsf{N}}\,\overline{0})$ since $(P\,\overline{0})$ implies $(P\,\overline{0})$

λ-Calculus: Sets

Define $\overline{\mathsf{N}}$ to be:

$$\lambda z.\forall P.((P\,\overline{0}) \wedge (\forall x.\,(P\,x) \supset (P\,.\,\overline{\mathsf{SUCC}}\,x))) \supset (P\,z)$$

This satisfies the three requirements.

- $(\overline{\mathsf{N}}\,\overline{0})$ since $(P\,\overline{0})$ implies $(P\,\overline{0})$
- $\forall x.(\overline{\mathsf{N}}\,x) \supset (\overline{\mathsf{N}}(\overline{\mathsf{SUCC}}\,x))$ since if $P\,x$ and P is closed under successor, then $P\,(\overline{\mathsf{SUCC}}p))$

λ-Calculus: Sets

Define $\overline{N}$ to be:

$$\lambda z.\forall P.((P\,\overline{0}) \wedge (\forall x.\,(P\,x) \supset (P\,.\,\overline{SUCC}\,x))) \supset (P\,z)$$

This satisfies the three requirements.

- $(\overline{N}\,\overline{0})$ since $(P\,\overline{0})$ implies $(P\,\overline{0})$
- $\forall x.(\overline{N}\,x) \supset (\overline{N}(\overline{SUCC}\,x))$ since if $P\,x$ and P is closed under successor, then $P\,(\overline{SUCC}p))$
- $\forall P.(P\,\overline{0}) \wedge (\forall x.(P\,x) \supset (P\,(\overline{SUCC}\,x))) \supset (\overline{N} \subseteq P)$
  $\overline{N}$ is the least such set as the intersection of all such sets P

λ-Calculus: Sets

Define $\overline{\mathsf{N}}$ to be:

$$\lambda z.\forall P.((P\,\overline{0}) \land (\forall x.\,(P\,x) \supset (P\,.\,\overline{SUCC}\,x))) \supset (P\,z)$$

This satisfies the three requirements.

We have used quantification over sets (characteristic functions – the variable P) to define $\overline{\mathsf{N}}$.

λ-Calculus: Russell's Paradox

Our representation framework is very powerful.

λ-Calculus: Russell's Paradox

Our representation framework is very powerful.

Actually it is so powerful that it is inconsistent!

λ-Calculus: Russell's Paradox

Our representation framework is very powerful.

Actually it is so powerful that it is inconsistent!

Russell's paradox:

Consider the term R:

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

λ-Calculus: Russell's Paradox

Our representation framework is very powerful.
Actually it is so powerful that it is inconsistent!
Russell's paradox:
Consider the term R:

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

As a characteristic function, R represents the set of all sets which do not contain themselves:

$$\{x | x \notin x\}$$

λ-Calculus: Russell's Paradox

Consider the term R:

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

λ-Calculus: Russell's Paradox

Consider the term R:

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

Now we evaluate the expression $E := (R\,R)$

$$((\lambda x.\neg.x\ x)\ R)$$

λ-Calculus: Russell's Paradox

Consider the term R:

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

Now we evaluate the expression $\mathsf{E} := (\mathsf{R}\,\mathsf{R})$

$((\lambda x.\neg.x\ x)\ \mathsf{R})$ evaluates to

λ-Calculus: Russell's Paradox

Consider the term R:

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

Now we evaluate the expression $E := (R\,R)$

$$((\lambda x.\neg.x\ x)\ R) \qquad \text{evaluates to} \qquad \neg(R\,R)$$

λ-Calculus: Russell's Paradox

Consider the term R:

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

Now we evaluate the expression $E := (R\,R)$

$((\lambda x.\neg.x\ x)\ R)$ evaluates to $\neg(R\,R)$

And we evaluate $\neg(R\,R)$

$$\neg((\lambda x.\neg.x\,x)\,R)$$

λ-Calculus: Russell's Paradox

Consider the term R:

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

Now we evaluate the expression $E := (R\,R)$

$((\lambda x.\neg.x\ x)\ R)$ evaluates to $\neg(R\,R)$

And we evaluate $\neg(R\,R)$

$\neg((\lambda x.\neg.x\,x)\,R)$ evaluates to

λ-Calculus: Russell's Paradox

Consider the term R:

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

Now we evaluate the expression $E := (R\,R)$

$((\lambda x.\neg.x\ x)\ R)$ evaluates to $\neg(R\,R)$

And we evaluate $\neg(R\,R)$

$\neg((\lambda x.\neg.x\,x)\,R)$ evaluates to $\neg\neg(R\,R)$

λ-Calculus: Russell's Paradox

Consider the term R:

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

Now we evaluate the expression $E := (R\,R)$

$$((\lambda x.\neg.x\ x)\ R) \qquad \text{evaluates to} \qquad \neg(R\,R)$$

And we evaluate $\neg(R\,R)$

$$\neg((\lambda x.\neg.x\,x)\,R) \qquad \text{evaluates to} \qquad \neg\neg(R\,R)$$

which is equivalent to $(R\,R)$

λ-Calculus: Russell's Paradox

Consider the term R:

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

Now we evaluate the expression $E := (R\,R)$

$$((\lambda x.\neg.x\ x)\ R) \qquad \text{evaluates to} \qquad \neg(R\,R)$$

And we evaluate $\neg(R\,R)$

$$\neg((\lambda x.\neg.x\,x)\,R) \qquad \text{evaluates to} \qquad \neg\neg(R\,R)$$

which is equivalent to $(R\,R)$

Thus if E holds we can infer $\neg E$ and vice versa. This is Russell's paradox.

λ-Calculus: Nontermination

Note that the term $(\lambda x.\neg.x\,x)$ (just as the standard example $(\lambda x.x\,x)$) does not terminate with respect to β-reduction:

$$(R\,R) \longrightarrow_\beta \neg(R\,R) \longrightarrow_\beta \neg\neg(R\,R) \longrightarrow_\beta \ldots$$

Typed λ-Calculus

We can avoid Russell's paradox using simple types.

Typed λ-Calculus

We can avoid Russell's paradox using simple types.

Simple Types:

- o Base type of propositions

Typed λ-Calculus

We can avoid Russell's paradox using simple types.

Simple Types:

- o Base type of propositions
- ι Base type of individuals

Typed λ-Calculus

We can avoid Russell's paradox using simple types.
Simple Types:

- o Base type of propositions
- ι Base type of individuals
- $(\alpha\beta)$ (or $(\beta \to \alpha)$) Type of functions from β to α

Typed λ-Calculus

We can avoid Russell's paradox using simple types.
Simple Types:

- o Base type of propositions
- ι Base type of individuals
- $(\alpha\beta)$ (or $(\beta \to \alpha)$) Type of functions from β to α

One may include arbitrarily many base types $\iota^1, \ldots, \iota^n, \ldots$.

Typed λ-Calculus

We can avoid Russell's paradox using simple types.
Simple Types:

- o Base type of propositions
- ι Base type of individuals
- $(\alpha\beta)$ (or $(\beta \rightarrow \alpha)$) Type of functions from β to α

We often omit parenthesis in types. $(\alpha\beta\gamma)$ means $((\alpha\beta)\gamma)$

Typed λ-Calculus

We can avoid Russell's paradox using simple types.
Simple Types:

- o Base type of propositions
- ι Base type of individuals
- $(\alpha\beta)$ (or $(\beta \rightarrow \alpha)$) Type of functions from β to α

We often omit parenthesis in types. $(\alpha\beta\gamma)$ means $((\alpha\beta)\gamma)$
Likewise $(\gamma \rightarrow \beta \rightarrow \alpha)$ means $(\gamma \rightarrow (\beta \rightarrow \alpha))$

Typed λ-Calculus

We can avoid Russell's paradox using simple types.
Simple Types:

- o Base type of propositions
- ι Base type of individuals
- $(\alpha\beta)$ (or $(\beta \to \alpha)$) Type of functions from β to α

We often omit parenthesis in types. $(\alpha\beta\gamma)$ means $((\alpha\beta)\gamma)$
Likewise $(\gamma \to \beta \to \alpha)$ means $(\gamma \to (\beta \to \alpha))$
Note that the type $(\alpha\beta\gamma)$ (or $(\gamma \to \beta \to \alpha)$) is the type of a (Curried) function of two arguments which returns a value of type α.

Typed λ-Calculus: Typed Terms

- Typed Variables x_α

Typed λ-Calculus: Typed Terms

- Typed Variables x_α
- Typed Constants and Parameters P_α

Typed λ-Calculus: Typed Terms

- Typed Variables x_α
- Typed Constants and Parameters P_α
- Application $(F_{\alpha\beta}B_\beta)_\alpha$ – or $(F_{\beta\rightarrow\alpha}B_\beta)_\alpha$

Typed λ-Calculus: Typed Terms

- Typed Variables x_α
- Typed Constants and Parameters P_α
- Application $(F_{\alpha\beta} B_\beta)_\alpha$ – or $(F_{\beta\rightarrow\alpha} B_\beta)_\alpha$
- λ-abstraction $(\lambda y_\beta . A_\alpha)_{\alpha\beta}$ – or $(\lambda y_\beta . A_\alpha)_{\beta\rightarrow\alpha}$

Typed λ-Calculus: Typed Terms

- Typed Variables x_α
- Typed Constants and Parameters P_α
- Application $(F_{\alpha\beta} B_\beta)_\alpha$ – or $(F_{\beta\rightarrow\alpha} B_\beta)_\alpha$
- λ-abstraction $(\lambda y_\beta . A_\alpha)_{\alpha\beta}$ – or $(\lambda y_\beta . A_\alpha)_{\beta\rightarrow\alpha}$

Examples:

- $(\lambda x_\alpha . x_\alpha)$ term of type $(\alpha\alpha)$ – identity on type α

Typed λ-Calculus: Typed Terms

- Typed Variables x_α
- Typed Constants and Parameters P_α
- Application $(\mathsf{F}_{\alpha\beta}\mathsf{B}_\beta)_\alpha$ – or $(\mathsf{F}_{\beta\to\alpha}\mathsf{B}_\beta)_\alpha$
- λ-abstraction $(\lambda \mathsf{y}_\beta.\,\mathsf{A}_\alpha)_{\alpha\beta}$ – or $(\lambda \mathsf{y}_\beta.\,\mathsf{A}_\alpha)_{\beta\to\alpha}$

Examples:

- $(\lambda \mathsf{x}_\alpha.\,\mathsf{x}_\alpha)$ term of type $(\alpha\alpha)$ – identity on type α
- $(\lambda \mathsf{y}_\beta.\,\mathsf{x}_\alpha)$ term of type $(\alpha\beta)$ – constant x-valued function

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda \mathsf{x}.\mathsf{x}^2 - 1)$$

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.x^2 - 1)$$

This is shorthand for

$$(\lambda x.\ (\mathsf{MINUS}\ (\mathsf{SQUARE}\ x)\ 1))$$

where MINUS, SQUARE and 1 are constants.

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.x^2 - 1)$$

This is shorthand for

$$(\lambda x.\, (\mathsf{MINUS}\, (\mathsf{SQUARE}\, x)\, 1))$$

where MINUS, SQUARE and 1 are constants.

Is there a corresponding typed term?

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.x^2 - 1)$$

This is shorthand for

$$(\lambda x.\,(\mathsf{MINUS}\,(\mathsf{SQUARE}\,x)\,1))$$

where MINUS, SQUARE and 1 are constants.

Is there a corresponding typed term?

Assume the type of individuals ι corresponds to real numbers.

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.x^2 - 1)$$

This is shorthand for

$$(\lambda x.\,(\mathsf{MINUS}\,(\mathsf{SQUARE}\,x)\,1))$$

where MINUS, SQUARE and 1 are constants.
Is there a corresponding typed term?
Assume the type of individuals ι corresponds to real numbers.

- x and 1 should be real numbers (type ι)

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.x^2 - 1)$$

This is shorthand for

$$(\lambda x.\,(\mathsf{MINUS}\,(\mathsf{SQUARE}\,x)\,1))$$

where MINUS, SQUARE and 1 are constants.
Is there a corresponding typed term?
Assume the type of individuals ι corresponds to real numbers.

- x and 1 should be real numbers (type ι)
- SQUARE should take a real number to a real number (type $(\iota\iota)$)

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.x^2 - 1)$$

This is shorthand for

$$(\lambda x.\,(\mathsf{MINUS}\,(\mathsf{SQUARE}\,x)\,1))$$

where MINUS, SQUARE and 1 are constants.
Is there a corresponding typed term?
Assume the type of individuals ι corresponds to real numbers.

- x and 1 should be real numbers (type ι)
- SQUARE should take a real number to a real number (type $(\iota\iota)$)
- MINUS should take two real numbers to a real number (type $(\iota\iota\iota)$)

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.x^2 - 1)$$

This is shorthand for

$$(\lambda x.\,(\mathsf{MINUS}\,(\mathsf{SQUARE}\,x)\,1))$$

where MINUS, SQUARE and 1 are constants.

Is there a corresponding typed term?

Assume the type of individuals ι corresponds to real numbers.

Typed Term:

$$(\lambda x_\iota.\,(\mathsf{MINUS}_{\iota\iota\iota}\,(\mathsf{SQUARE}_{\iota\iota}\,x_\iota)\,1_\iota))$$

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.x^2 - 1)$$

This is shorthand for

$$(\lambda x. (\mathsf{MINUS}\, (\mathsf{SQUARE}\, x)\, 1))$$

where MINUS, SQUARE and 1 are constants.
Is there a corresponding typed term?
Assume the type of individuals ι corresponds to real numbers.
Typed Term:

$$(\lambda x_\iota. (\mathsf{MINUS}_{\iota\iota\iota}\, (\mathsf{SQUARE}_{\iota\iota}\, x_\iota)\, 1_\iota))$$

This term has type $(\iota\iota)$.

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x. (x^2 - 1 = 0))$$

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.\,(x^2 - 1 = 0))$$

This is shorthand for

$$(\lambda x\,.(=\ (\mathsf{MINUS}\,(\mathsf{SQUARE}\,x)\,1)\,0))$$

where $=$, MINUS, SQUARE, 0 and 1 are constants.

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.\,(x^2 - 1 = 0))$$

This is shorthand for

$$(\lambda x\,.(=\ (\mathsf{MINUS}\,(\mathsf{SQUARE}\,x)\,1)\,0))$$

where $=$, MINUS, SQUARE, 0 and 1 are constants.

- Already know types of MINUS, SQUARE and 1.

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.\,(x^2 - 1 = 0))$$

This is shorthand for

$$(\lambda x\,.(=\ (\mathsf{MINUS}\,(\mathsf{SQUARE}\,x)\,1)\,0))$$

where $=$, MINUS, SQUARE, 0 and 1 are constants.

- Already know types of MINUS, SQUARE and 1.
- 0 should be a real number (type ι)

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.\,(x^2 - 1 = 0))$$

This is shorthand for

$$(\lambda x\,.(=\;(\mathsf{MINUS}\,(\mathsf{SQUARE}\,x)\,1)\,0))$$

where $=$, MINUS, SQUARE, 0 and 1 are constants.

- Already know types of MINUS, SQUARE and 1.
- 0 should be a real number (type ι)
- $=$ takes two real numbers and returns a truth value (type $(o\iota\iota)$)

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x.\,(x^2 - 1 = 0))$$

This is shorthand for

$$(\lambda x\,.(=\ (\mathsf{MINUS}\,(\mathsf{SQUARE}\,x)\,1)\,0))$$

where $=$, MINUS, SQUARE, 0 and 1 are constants.
Typed Term:

$$(\lambda x_\iota.\,(=_{o\iota\iota}\ (\mathsf{MINUS}_{\iota\iota\iota}\,(\mathsf{SQUARE}_{\iota\iota}\,x_\iota)\,1_\iota)\,0_\iota)$$

Typed λ-Calculus: Typed Terms

Consider the untyped term

$$(\lambda x. (x^2 - 1 = 0))$$

This is shorthand for

$$(\lambda x.(= \ (MINUS\,(SQUARE\,x)\,1)\,0))$$

where $=$, MINUS, SQUARE, 0 and 1 are constants.
Typed Term:

$$(\lambda x_\iota. (=_{o\iota\iota} \ (MINUS_{\iota\iota\iota}\,(SQUARE_{\iota\iota}\,x_\iota)\,1_\iota)\,0_\iota)$$

This term has type $(o\iota)$.

Typed λ-Calculus: Assigning Types

General algorithm for assigning types to terms (when this is possible) – see Hindley97.

Typed λ-Calculus: Assigning Types

The basis for such an algorithm is the following deduction system:

Typed λ-Calculus: Assigning Types

The basis for such an algorithm is the following deduction system:

$$\frac{\mathsf{C} : \alpha \in \Gamma \quad \mathsf{C}\ \text{variable, parameter or constant}}{\Gamma \vdash_{\mathsf{TA}} \mathsf{C} : \alpha}\ \mathsf{Hyp}$$

Typed λ-Calculus: Assigning Types

The basis for such an algorithm is the following deduction system:

$$\frac{\mathsf{C} : \alpha \in \Gamma \quad \mathsf{C}\ \text{variable, parameter or constant}}{\Gamma \vdash_{\mathsf{TA}} \mathsf{C} : \alpha}\ \mathsf{Hyp}$$

$$\frac{\Gamma, \mathsf{y} : \beta \vdash_{\mathsf{TA}} \mathsf{A} : \alpha}{\Gamma \vdash_{\mathsf{TA}} (\lambda \mathsf{y}.\, \mathsf{A}) : \alpha\beta}\ \mathsf{Lam}$$

Typed λ-Calculus: Assigning Types

The basis for such an algorithm is the following deduction system:

$$\frac{\mathsf{C} : \alpha \in \Gamma \quad \mathsf{C}\text{ variable, parameter or constant}}{\Gamma \vdash_{\mathsf{TA}} \mathsf{C} : \alpha}\ \mathsf{Hyp}$$

$$\frac{\Gamma, \mathsf{y} : \beta \vdash_{\mathsf{TA}} \mathsf{A} : \alpha}{\Gamma \vdash_{\mathsf{TA}} (\lambda \mathsf{y}.\, \mathsf{A}) : \alpha\beta}\ \mathsf{Lam} \qquad \frac{\Gamma \vdash_{\mathsf{TA}} \mathsf{F} : \alpha\beta \quad \Gamma \vdash_{\mathsf{TA}} \mathsf{B} : \beta}{\Gamma \vdash_{\mathsf{TA}} (\mathsf{F}\,\mathsf{B}) : \alpha}\ \mathsf{App}$$

Typed λ-Calculus: Assigning Types

The basis for such an algorithm is the following deduction system:

$$\frac{\mathsf{C} : \alpha \in \Gamma \quad \text{C variable, parameter or constant}}{\Gamma \vdash_{\mathsf{TA}} \mathsf{C} : \alpha}\ \mathsf{Hyp}$$

$$\frac{\Gamma, \mathsf{y} : \beta \vdash_{\mathsf{TA}} \mathsf{A} : \alpha}{\Gamma \vdash_{\mathsf{TA}} (\lambda \mathsf{y}.\, \mathsf{A}) : \alpha\beta}\ \mathsf{Lam} \qquad \frac{\Gamma \vdash_{\mathsf{TA}} \mathsf{F} : \alpha\beta \quad \Gamma \vdash_{\mathsf{TA}} \mathsf{B} : \beta}{\Gamma \vdash_{\mathsf{TA}} (\mathsf{F}\,\mathsf{B}) : \alpha}\ \mathsf{App}$$

We can assign the type α to a term A in context Γ whenever we can derive

$$\Gamma \vdash_{\mathsf{TA}} \mathsf{A} : \alpha$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda \mathsf{x}.\, (\mathsf{SQUARE}\, \mathsf{x}))$

Goal: Find a type α such that

$\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}.\, (\mathsf{SQUARE}\, \mathsf{x})) : \alpha$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda \mathsf{x}.\,(\mathsf{SQUARE}\,\mathsf{x}))$

Goal: Find a type α such that

$\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}.\,(\mathsf{SQUARE}\,\mathsf{x})) : \alpha$

$$\begin{array}{c} \vdots \\ \mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}.\,(\mathsf{SQUARE}\,\mathsf{x})) : \alpha \end{array}$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda \mathsf{x}. (\mathsf{SQUARE}\, \mathsf{x}))$

Goal: Find a type α such that

$\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}. (\mathsf{SQUARE}\, \mathsf{x})) : \alpha$

α is $(\gamma\beta)$

$$\dfrac{\begin{array}{c}\vdots \\ \mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \beta \vdash_{\mathsf{TA}} (\mathsf{SQUARE}\, \mathsf{x}) : \gamma\end{array}}{\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}. (\mathsf{SQUARE}\, \mathsf{x})) : \gamma\beta}\ \mathsf{Lam}$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda \mathsf{x}.\, (\mathsf{SQUARE}\, \mathsf{x}))$

Goal: Find a type α such that

$\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}.\, (\mathsf{SQUARE}\, \mathsf{x})) : \alpha$

$$\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \beta \vdash_{\mathsf{TA}} \mathsf{SQUARE} : (\gamma\delta)} \qquad \dfrac{\vdots}{\mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \beta \vdash_{\mathsf{TA}} \mathsf{x} : \delta}}{\mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \beta \vdash_{\mathsf{TA}} (\mathsf{SQUARE}\, \mathsf{x}) : \gamma}\,\mathsf{App}}{\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}.\, (\mathsf{SQUARE}\, \mathsf{x})) : \gamma\beta}\,\mathsf{Lam}$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda \mathsf{x}.\, (\mathsf{SQUARE}\, \mathsf{x}))$

Goal: Find a type α such that

$\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}.\, (\mathsf{SQUARE}\, \mathsf{x})) : \alpha$

γ and δ are both ι

$$\dfrac{\dfrac{\dfrac{}{\mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \beta \vdash_{\mathsf{TA}} \mathsf{SQUARE} : (\iota\iota)}\,\mathsf{Hyp} \qquad \begin{array}{c}\vdots \\ \mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \beta \vdash_{\mathsf{TA}} \mathsf{x} : \iota\end{array}}{\mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \beta \vdash_{\mathsf{TA}} (\mathsf{SQUARE}\, \mathsf{x}) : \iota}\,\mathsf{App}}{\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}.\, (\mathsf{SQUARE}\, \mathsf{x})) : \iota\beta}\,\mathsf{Lam}$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda \mathsf{x}.\, (\mathsf{SQUARE}\, \mathsf{x}))$

Goal: Find a type α such that

$\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}.\, (\mathsf{SQUARE}\, \mathsf{x})) : \alpha$

β is ι

$$\dfrac{\dfrac{}{\mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \iota \vdash_{\mathsf{TA}} \mathsf{SQUARE} : (\iota\iota)}\,\mathsf{Hyp} \qquad \dfrac{}{\mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \iota \vdash_{\mathsf{TA}} \mathsf{x} : \iota}\,\mathsf{Hyp}}{\dfrac{\mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \iota \vdash_{\mathsf{TA}} (\mathsf{SQUARE}\, \mathsf{x}) : \iota}{\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}.\, (\mathsf{SQUARE}\, \mathsf{x})) : \iota\iota}\,\mathsf{Lam}}\,\mathsf{App}$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda x. (SQUARE\, x))$

Goal: Find a type α such that

$SQUARE : (\iota\iota) \vdash_{TA} (\lambda x. (SQUARE\, x)) : \alpha$

β is ι

$$\dfrac{\dfrac{\dfrac{}{SQUARE : (\iota\iota), x : \iota \vdash_{TA} SQUARE : (\iota\iota)}\,Hyp \qquad \dfrac{}{SQUARE : (\iota\iota), x : \iota \vdash_{TA} x : \iota}\,Hyp}{SQUARE : (\iota\iota), x : \iota \vdash_{TA} (SQUARE\, x) : \iota}\,App}{SQUARE : (\iota\iota) \vdash_{TA} (\lambda x. (SQUARE\, x)) : \iota\iota}\,Lam$$

So $(\lambda x. (SQUARE\, x))$ can be assigned the type $(\iota\iota)$ in context $SQUARE : (\iota\iota)$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda \mathsf{x}.\,(\mathsf{SQUARE}\,\mathsf{x}))$

Goal: Find a type α such that

$\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}.\,(\mathsf{SQUARE}\,\mathsf{x})) : \alpha$

β is ι

$$\dfrac{\dfrac{\dfrac{}{\mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \iota \vdash_{\mathsf{TA}} \mathsf{SQUARE} : (\iota\iota)}\,\mathsf{Hyp} \quad \dfrac{}{\mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \iota \vdash_{\mathsf{TA}} \mathsf{x} : \iota}\,\mathsf{Hyp}}{\mathsf{SQUARE} : (\iota\iota), \mathsf{x} : \iota \vdash_{\mathsf{TA}} (\mathsf{SQUARE}\,\mathsf{x}) : \iota}\,\mathsf{App}}{\mathsf{SQUARE} : (\iota\iota) \vdash_{\mathsf{TA}} (\lambda \mathsf{x}.\,(\mathsf{SQUARE}\,\mathsf{x})) : \iota\iota}\,\mathsf{Lam}$$

So $(\lambda \mathsf{x}.\,(\mathsf{SQUARE}\,\mathsf{x}))$ can be assigned the type $(\iota\iota)$ in context $\mathsf{SQUARE} : (\iota\iota)$

Corresponding Typed Term: $(\lambda \mathsf{x}_\iota.\,(\mathsf{SQUARE}_{\iota\iota}\,\mathsf{x}_\iota))$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda \mathsf{x} \,.\, \neg\,(\mathsf{x}\,\mathsf{x}))$

Goal: Find a type α such that $\neg : (\mathsf{oo}) \vdash_{\mathsf{TA}} (\lambda \mathsf{x} \,.\, \neg\,(\mathsf{x}\,\mathsf{x})) : \alpha$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda x . \neg (x x))$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} (\lambda x . \neg (x x)) : \alpha$

$$\vdots$$
$$\neg : (oo) \vdash_{TA} (\lambda x . \neg (x x)) : \alpha$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda x\,.\,\neg\,(x\,x))$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} (\lambda x\,.\,\neg\,(x\,x)) : \alpha$

α is $(\gamma\beta)$

$$\dfrac{\begin{array}{c}\vdots\\ \neg : (oo), x : \beta \vdash_{TA} (\neg\,(x\,x)) : \gamma\end{array}}{\neg : (oo) \vdash_{TA} (\lambda x\,.\,\neg\,(x\,x)) : \gamma\beta}\ \mathrm{Lam}$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda x . \neg (x x))$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} (\lambda x . \neg (x x)) : \alpha$

$$\dfrac{\dfrac{\dfrac{\vdots}{\neg : (oo), x : \beta \vdash_{TA} \neg : (\gamma\delta)} \qquad \dfrac{\vdots}{\neg : (oo), x : \beta \vdash_{TA} (x x) : \delta}}{\neg : (oo), x : \beta \vdash_{TA} (\neg (x x)) : \gamma}\,\mathrm{App}}{\neg : (oo) \vdash_{TA} (\lambda x. (\neg (x x)) : \gamma\beta}\,\mathrm{Lam}$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda x . \neg (x x))$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} (\lambda x . \neg (x x)) : \alpha$

γ and δ are both o

$$\dfrac{\dfrac{}{\neg : (oo), x : \beta \vdash_{TA} \neg : (oo)}\text{Hyp} \qquad \begin{matrix} \vdots \\ \neg : (oo), x : \beta \vdash_{TA} (x x) : o \end{matrix}}{\dfrac{\neg : (oo), x : \beta \vdash_{TA} (\neg (x x)) : o}{\neg : (oo) \vdash_{TA} (\lambda x. (\neg (x x)) : o\beta}\text{Lam}}\text{App}$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda x . \neg (x x))$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} (\lambda x . \neg (x x)) : \alpha$

$$\vdots$$

$$\neg : (oo), x : \beta \vdash_{TA} (x x) : o$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda x . \neg (x\,x))$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} (\lambda x . \neg (x\,x)) : \alpha$

$$\dfrac{\dfrac{\vdots}{\neg : (oo), x : \beta \vdash_{TA} x : (o\epsilon)} \qquad \dfrac{\vdots}{\neg : (oo), x : \beta \vdash_{TA} x : \epsilon}}{\neg : (oo), x : \beta \vdash_{TA} (x\,x) : o}\,\mathrm{App}$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda x . \neg (x\, x))$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} (\lambda x . \neg (x\, x)) : \alpha$

β is $(o\epsilon)$

$$\dfrac{\dfrac{}{\neg : (oo), x : (o\epsilon) \vdash_{TA} x : (o\epsilon)}\,\text{Hyp} \qquad \begin{array}{c}\vdots \\ \neg : (oo), x : (o\epsilon) \vdash_{TA} x : \epsilon\end{array}}{\neg : (oo), x : (o\epsilon) \vdash_{TA} (x\, x) : o}\,\text{App}$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda x . \neg (x x))$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} (\lambda x . \neg (x x)) : \alpha$

Only remaining subgoal:

$$\neg : (oo), x : (o\epsilon) \vdash_{TA} x : \epsilon$$

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda \mathsf{x} . \neg (\mathsf{x}\,\mathsf{x}))$
Goal: Find a type α such that $\neg : (\mathsf{oo}) \vdash_{\mathsf{TA}} (\lambda \mathsf{x} . \neg (\mathsf{x}\,\mathsf{x})) : \alpha$
Only remaining subgoal:

$$\neg : (\mathsf{oo}), \mathsf{x} : (\mathsf{o}\epsilon) \vdash_{\mathsf{TA}} \mathsf{x} : \epsilon$$

This goal cannot be solved since $(\mathsf{o}\epsilon)$ cannot equal ϵ.

Typed λ-Calculus: Assigning Types

Untyped Term: $(\lambda x . \neg (x x))$
Goal: Find a type α such that $\neg : (oo) \vdash_{TA} (\lambda x . \neg (x x)) : \alpha$
Only remaining subgoal:

$$\neg : (oo), x : (o\epsilon) \vdash_{TA} x : \epsilon$$

This goal cannot be solved since $(o\epsilon)$ cannot equal ϵ.

Hence $(\lambda x. (\neg (x x)))$ cannot be typed – avoiding Russell's Paradox.

Typed λ-Calculus: $\beta\eta$

β-reduction:

$$((\lambda \mathsf{y}_\beta \,.\, \mathsf{A}_\alpha)\ \mathsf{B}_\beta) \longrightarrow_\beta \mathsf{A}_\alpha[\mathsf{y}_\beta/\mathsf{B}_\beta]$$

Typed λ-Calculus: $\beta\eta$

β-reduction:

$$((\lambda \mathsf{y}_\beta \, . \, \mathsf{A}_\alpha) \; \mathsf{B}_\beta) \longrightarrow_\beta \mathsf{A}_\alpha[\mathsf{y}_\beta / \mathsf{B}_\beta]$$

η-reduction:

$$(\lambda \mathsf{y}_\beta . \mathsf{F}_{\alpha\beta} \; \mathsf{y}_\beta) \longrightarrow_\eta \mathsf{F}_{\alpha\beta}$$

Typed λ-Calculus: $\beta\eta$

β-reduction:

$$((\lambda \mathsf{y}_\beta \,.\, \mathsf{A}_\alpha)\ \mathsf{B}_\beta) \longrightarrow_\beta \mathsf{A}_\alpha[\mathsf{y}_\beta/\mathsf{B}_\beta]$$

η-reduction:

$$(\lambda \mathsf{y}_\beta.\mathsf{F}_{\alpha\beta}\ \mathsf{y}_\beta) \longrightarrow_\eta \mathsf{F}_{\alpha\beta}$$

Facts:

- $\beta\eta$-normalization terminates for typed terms.

Typed λ-Calculus: $\beta\eta$

β-reduction:

$$((\lambda \mathsf{y}_\beta \,.\, \mathsf{A}_\alpha)\ \mathsf{B}_\beta) \longrightarrow_\beta \mathsf{A}_\alpha[\mathsf{y}_\beta/\mathsf{B}_\beta]$$

η-reduction:

$$(\lambda \mathsf{y}_\beta.\mathsf{F}_{\alpha\beta}\ \mathsf{y}_\beta) \longrightarrow_\eta \mathsf{F}_{\alpha\beta}$$

Facts:

- $\beta\eta$-normalization terminates for typed terms.
- Every typed term has a unique $\beta\eta$-normal form.