An introduction to λ-calculus

24th of March, 2016

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not.

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not. For instance, in (classical) propositional logic:

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not. For instance, in (classical) propositional logic:

- $A \vee \neg A$ is

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not. For instance, in (classical) propositional logic:

- $A \vee \neg A$ is "always true" (if A is true or false, anyway $A \vee \neg A$ is true);

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not. For instance, in (classical) propositional logic:

- $A \vee \neg A$ is "always true" (if A is true or false, anyway $A \vee \neg A$ is true);
- $A \vee B$

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not. For instance, in (classical) propositional logic:

- $A \vee \neg A$ is "always true" (if A is true or false, anyway $A \vee \neg A$ is true);
- $A \vee B$ is not "always true" (if A and B are false, $A \vee B$ is false)

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not. For instance, in (classical) propositional logic:

- $A \vee \neg A$ is "always true" (if A is true or false, anyway $A \vee \neg A$ is true);
- $A \vee B$ is not "always true" (if A and B are false, $A \vee B$ is false)

In first-order logic (= predicate calculus):

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not. For instance, in (classical) propositional logic:

- $A \vee \neg A$ is "always true" (if A is true or false, anyway $A \vee \neg A$ is true);
- $A \vee B$ is not "always true" (if A and B are false, $A \vee B$ is false)

In first-order logic (= predicate calculus):

$(\forall y)((\forall x)P(x) \Rightarrow P(y))$ is...

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not. For instance, in (classical) propositional logic:

- $A \vee \neg A$ is "always true" (if A is true or false, anyway $A \vee \neg A$ is true);
- $A \vee B$ is not "always true" (if A and B are false, $A \vee B$ is false)

In first-order logic (= predicate calculus):

$(\forall y)((\forall x)P(x) \Rightarrow P(y))$ is... universally valid

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not. For instance, in (classical) propositional logic:

- $A \vee \neg A$ is "always true" (if A is true or false, anyway $A \vee \neg A$ is true);
- $A \vee B$ is not "always true" (if A and B are false, $A \vee B$ is false)

In first-order logic (= predicate calculus):

$(\forall y)((\forall x)P(x) \Rightarrow P(y))$ is... universally valid

$(\forall y)(\forall x)(P(x) \Rightarrow P(y))$ is...

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not. For instance, in (classical) propositional logic:

- $A \vee \neg A$ is "always true" (if A is true or false, anyway $A \vee \neg A$ is true);
- $A \vee B$ is not "always true" (if A and B are false, $A \vee B$ is false)

In first-order logic (= predicate calculus):

$(\forall y)((\forall x)P(x) \Rightarrow P(y))$ is... universally valid

$(\forall y)(\forall x)(P(x) \Rightarrow P(y))$ is... NOT universally valid

Universally valid formulas

A natural problem in logic is to know whether a given formula is "always true" (*universally valid*) or not. For instance, in (classical) propositional logic:

- $A \vee \neg A$ is "always true" (if A is true or false, anyway $A \vee \neg A$ is true);
- $A \vee B$ is not "always true" (if A and B are false, $A \vee B$ is false)

In first-order logic (= predicate calculus):

$(\forall y)((\forall x)P(x) \Rightarrow P(y))$ is... universally valid

$(\forall y)(\forall x)(P(x) \Rightarrow P(y))$ is... NOT universally valid

The decision problem

The decision problem consists in having a “procedure” that takes as an input a formula and that returns as an output

- “YES” if the formula is *universally valid* (i.e. true for any interpretation)
- “NO” if the formula is not *universally valid* (i.e. might be false for some interpretation)

The decision problem

The decision problem consists in having a "procedure" that takes as an input a formula and that returns as an output

- "YES" if the formula is *universally valid* (i.e. true for any interpretation)
- "NO" if the formula is not *universally valid* (i.e. might be false for some interpretation)

In 1915, Leopold Löwenheim described such a procedure for the monadic predicate calculus (i.e. every predicate is of arity 0 or 1, like in the two formulas above).

“The most important problem in mathematical logic”

In 1928, David Hilbert and Wilhelm Ackermann wrote a book in which they said that “the most important problem in mathematical logic” is to solve the decision problem for the predicate calculus, i.e. to find such a procedure for the predicate calculus (with predicates of any arity).

“The most important problem in mathematical logic”

In 1928, David Hilbert and Wilhelm Ackermann wrote a book in which they said that “the most important problem in mathematical logic” is to solve the decision problem for the predicate calculus, i.e. to find such a procedure for the predicate calculus (with predicates of any arity).
Notice first that it is quite obvious that thet set of universally valid formulas is recursively enumerable: there exists a Turing machine that accepts the set of the universally valid formulas.

“The most important problem in mathematical logic”

In 1928, David Hilbert and Wilhelm Ackermann wrote a book in which they said that “the most important problem in mathematical logic” is to solve the decision problem for the predicate calculus, i.e. to find such a procedure for the predicate calculus (with predicates of any arity).
Notice first that it is quite obvious that thet set of universally valid formulas is recursively enumerable: there exists a Turing machine that accepts the set of the universally valid formulas. Equivalently there exists a Turing machine that writes all the universally valid formulas.

“The most important problem in mathematical logic”

In 1928, David Hilbert and Wilhelm Ackermann wrote a book in which they said that “the most important problem in mathematical logic” is to solve the decision problem for the predicate calculus, i.e. to find such a procedure for the predicate calculus (with predicates of any arity).
Notice first that it is quite obvious that thet set of universally valid formulas is recursively enumerable: there exists a Turing machine that accepts the set of the universally valid formulas. Equivalently there exists a Turing machine that writes all the universally valid formulas. So this set is RECURSIVELY ENUMERABLE.

“The most important problem in mathematical logic”

In 1928, David Hilbert and Wilhelm Ackermann wrote a book in which they said that “the most important problem in mathematical logic” is to solve the decision problem for the predicate calculus, i.e. to find such a procedure for the predicate calculus (with predicates of any arity).
Notice first that it is quite obvious that thet set of universally valid formulas is recursively enumerable: there exists a Turing machine that accepts the set of the universally valid formulas. Equivalently there exists a Turing machine that writes all the universally valid formulas. So this set is RECURSIVELY ENUMERABLE.
The question is: can one show that this set is RECURSIVE.

A formalization of the notion of procedure

In 1936, Alonzo Church proved that there is no such a procedure: the problem of decision for the predicate calculus is NOT decidable, i.e. the set of universally valid formulas is NOT RECURSIVE.

A formalization of the notion of procedure

In 1936, Alonzo Church proved that there is no such a procedure: the problem of decision for the predicate calculus is NOT decidable, i.e. the set of universally valid formulas is NOT RECURSIVE.
In order to prove it, he introduced the λ-calculus.
Indeed:

A formalization of the notion of procedure

In 1936, Alonzo Church proved that there is no such a procedure: the problem of decision for the predicate calculus is NOT decidable, i.e. the set of universally valid formulas is NOT RECURSIVE.
In order to prove it, he introduced the λ-calculus.
Indeed:
If you have in mind some procedure that solves some problem, you can describe it more or less informally even if you have no formal definition of the notion of "procedure".

A formalization of the notion of procedure

In 1936, Alonzo Church proved that there is no such a procedure: the problem of decision for the predicate calculus is NOT decidable, i.e. the set of universally valid formulas is NOT RECURSIVE.
In order to prove it, he introduced the λ-calculus.
Indeed:
If you have in mind some procedure that solves some problem, you can describe it more or less informally even if you have no formal definition of the notion of "procedure".
But if you want to prove that there exists no procedure that solves some problem, you need a formal definition of the notion of "procedure".

A formalization of the notion of procedure

In 1936, Alonzo Church proved that there is no such a procedure: the problem of decision for the predicate calculus is NOT decidable, i.e. the set of universally valid formulas is NOT RECURSIVE.

In order to prove it, he introduced the λ-calculus.

Indeed:

If you have in mind some procedure that solves some problem, you can describe it more or less informally even if you have no formal definition of the notion of "procedure".

But if you want to prove that there exists no procedure that solves some problem, you need a formal definition of the notion of "procedure".

The definition given by Church is: a procedure is a λ-term.

A formalization of the notion of procedure

In 1936, Alonzo Church proved that there is no such a procedure: the problem of decision for the predicate calculus is NOT decidable, i.e. the set of universally valid formulas is NOT RECURSIVE.
In order to prove it, he introduced the λ-calculus.
Indeed:
If you have in mind some procedure that solves some problem, you can describe it more or less informally even if you have no formal definition of the notion of "procedure".
But if you want to prove that there exists no procedure that solves some problem, you need a formal definition of the notion of "procedure".
The definition given by Church is: a procedure is a λ-term.

λ-terms

Intuitively: any λ-term represents a function.

λ-terms

Intuitively: any λ-term represents a function.
Formally: The grammar of the terms is defined as follows. We are given a denumerable set $\mathcal{V}$ of variables and:

- any variable x is a λ-term;

λ-terms

Intuitively: any λ-term represents a function.
Formally: The grammar of the terms is defined as follows. We are given a denumerable set $\mathcal{V}$ of variables and:

- any variable x is a λ-term;
- if v and u are λ-terms, then $(v)u$ is a λ-term;

λ-terms

Intuitively: any λ-term represents a function.
Formally: The grammar of the terms is defined as follows. We are given a denumerable set $\mathcal{V}$ of variables and:

- any variable x is a λ-term;
- if v and u are λ-terms, then $(v)u$ is a λ-term;
- if v is a λ-term and x is a variable, then $\lambda x.v$ is a λ-term.

λ-terms

Intuitively: any λ-term represents a function.
Formally: The grammar of the terms is defined as follows. We are given a denumerable set $\mathcal{V}$ of variables and:

- any variable x is a λ-term;
- if v and u are λ-terms, then $(v)u$ is a λ-term;
- if v is a λ-term and x is a variable, then $\lambda x.v$ is a λ-term.

Example: $(\lambda x.x)x$ is a λ-term

λ-terms

Intuitively: any λ-term represents a function.
Formally: The grammar of the terms is defined as follows. We are given a denumerable set $\mathcal{V}$ of variables and:

- any variable x is a λ-term;
- if v and u are λ-terms, then $(v)u$ is a λ-term;
- if v is a λ-term and x is a variable, then $\lambda x.v$ is a λ-term.

Example: $(\lambda x.x)x$ is a λ-term
We will write also $(t)u_1 \ldots u_{k+1}$ or even $tu_1 \ldots u_{k+1}$ instead of $(\ldots(((t)u_1)u_2)\ldots)u_{k+1}$.

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

Example: In $(\lambda y.\lambda x.x)zx$, the bound variables are...

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

Example: In $(\lambda y.\lambda x.x)zx$, the bound variables are... y and x.

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

Example: In $(\lambda y.\lambda x.x)zx$, the bound variables are... y and x.

We will identify λ-terms up to the names of the bound variables.

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

Example: In $(\lambda y.\lambda x.x)zx$, the bound variables are... y and x.

We will identify λ-terms up to the names of the bound variables.

Like in: $\int_a^b f(x)\,\mathrm{d}x = \int_a^b f(y)\,\mathrm{d}y$

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

Example: In $(\lambda y.\lambda x.x)zx$, the bound variables are... y and x.

We will identify λ-terms up to the names of the bound variables.

Like in: $\int_a^b f(x)\,\mathrm{d}x = \int_a^b f(y)\,\mathrm{d}y$

or in $(\forall y)((\forall x)P(x) \Rightarrow P(y)) = (\forall x)((\forall y)P(y) \Rightarrow P(x))$

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

Example: In $(\lambda y.\lambda x.x)zx$, the bound variables are... y and x.

We will identify λ-terms up to the names of the bound variables.

Like in: $\displaystyle\int_a^b f(x)\,\mathrm{d}x = \int_a^b f(y)\,\mathrm{d}y$

or in $(\forall y)((\forall x)P(x) \Rightarrow P(y)) = (\forall x)((\forall y)P(y) \Rightarrow P(x))$

This identification is called α-equivalence and is defined as follows: two terms t and t' are α-equivalent if one can obtain t' from t by a series of *changes of bound variables*.

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

Example: In $(\lambda y.\lambda x.x)zx$, the bound variables are... y and x.

We will identify λ-terms up to the names of the bound variables.

Like in: $\displaystyle\int_a^b f(x)\,\mathrm{d}x = \int_a^b f(y)\,\mathrm{d}y$

or in $(\forall y)((\forall x)P(x) \Rightarrow P(y)) = (\forall x)((\forall y)P(y) \Rightarrow P(x))$

This identification is called α-equivalence and is defined as follows: two terms t and t' are α-equivalent if one can obtain t' from t by a series of *changes of bound variables*.

A term v' is obtained from a term v by a change of a bound variable if it is obtained by replacing some part $\lambda x.u$ of v by $\lambda y.u'$, where:

- y is a variable that does not occur in u;
- and u' is obtained by replacing every occurrence of x by an occurrence of y.

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

Example: In $(\lambda y.\lambda x.x)zx$, the bound variables are... y and x.

We will identify λ-terms up to the names of the bound variables.

Like in: $\displaystyle\int_a^b f(x)\,\mathrm{d}x = \int_a^b f(y)\,\mathrm{d}y$

or in $(\forall y)((\forall x)P(x) \Rightarrow P(y)) = (\forall x)((\forall y)P(y) \Rightarrow P(x))$

This identification is called α-equivalence and is defined as follows: two terms t and t' are α-equivalent if one can obtain t' from t by a series of *changes of bound variables*.

A term v' is obtained from a term v by a change of a bound variable if it is obtained by replacing some part $\lambda x.u$ of v by $\lambda y.u'$, where:

- y is a variable that does not occur in u;
- and u' is obtained by replacing every occurrence of x by an occurrence of y.

Are $\lambda x.(x)y$ and $\lambda z.(z)y$ α-equivalent?

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

Example: In $(\lambda y.\lambda x.x)zx$, the bound variables are... y and x.

We will identify λ-terms up to the names of the bound variables.

Like in: $\int_a^b f(x)\,\mathrm{d}x = \int_a^b f(y)\,\mathrm{d}y$

or in $(\forall y)((\forall x)P(x) \Rightarrow P(y)) = (\forall x)((\forall y)P(y) \Rightarrow P(x))$

This identification is called α-equivalence and is defined as follows: two terms t and t' are α-equivalent if one can obtain t' from t by a series of *changes of bound variables*.

A term v' is obtained from a term v by a change of a bound variable if it is obtained by replacing some part $\lambda x.u$ of v by $\lambda y.u'$, where:

- y is a variable that does not occur in u;
- and u' is obtained by replacing every occurrence of x by an occurrence of y.

Are $\lambda x.(x)y$ and $\lambda z.(z)y$ α-equivalent? YES

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

Example: In $(\lambda y.\lambda x.x)zx$, the bound variables are... y and x.

We will identify λ-terms up to the names of the bound variables.

Like in: $\displaystyle\int_a^b f(x)\,\mathrm{d}x = \int_a^b f(y)\,\mathrm{d}y$

or in $(\forall y)((\forall x)P(x) \Rightarrow P(y)) = (\forall x)((\forall y)P(y) \Rightarrow P(x))$

This identification is called α-equivalence and is defined as follows: two terms t and t' are α-equivalent if one can obtain t' from t by a series of *changes of bound variables*.

A term v' is obtained from a term v by a change of a bound variable if it is obtained by replacing some part $\lambda x.u$ of v by $\lambda y.u'$, where:

- y is a variable that does not occur in u;
- and u' is obtained by replacing every occurrence of x by an occurrence of y.

Are $\lambda x.(x)y$ and $\lambda z.(z)y$ α-equivalent? YES

Are $\lambda z.(z)y$ and $\lambda y.(y)y$ α-equivalent?

α-equivalence

The *bound variables* of a term t are the variables that appear in t immediatly after some λ.

Example: In $(\lambda y.\lambda x.x)zx$, the bound variables are... y and x.

We will identify λ-terms up to the names of the bound variables.

Like in: $\displaystyle\int_a^b f(x)\,\mathrm{d}x = \int_a^b f(y)\,\mathrm{d}y$

or in $(\forall y)((\forall x)P(x) \Rightarrow P(y)) = (\forall x)((\forall y)P(y) \Rightarrow P(x))$

This identification is called α-equivalence and is defined as follows: two terms t and t' are α-equivalent if one can obtain t' from t by a series of *changes of bound variables*.

A term v' is obtained from a term v by a change of a bound variable if it is obtained by replacing some part $\lambda x.u$ of v by $\lambda y.u'$, where:

- y is a variable that does not occur in u;
- and u' is obtained by replacing every occurrence of x by an occurrence of y.

Are $\lambda x.(x)y$ and $\lambda z.(z)y$ α-equivalent? YES

Are $\lambda z.(z)y$ and $\lambda y.(y)y$ α-equivalent? NO

Free occurrences of variables

The *free occurrences* of a variable x in a term t are defined, by induction, as follows:

Free occurrences of variables

The *free occurrences* of a variable x in a term t are defined, by induction, as follows:

- if t is the variable x, then the occurrence of x in t is free;

Free occurrences of variables

The *free occurrences* of a variable x in a term t are defined, by induction, as follows:

- if t is the variable x, then the occurrence of x in t is free;
- if $t = (v)u$, then the free occurrences of x in t are those off x in u and v;

Free occurrences of variables

The *free occurrences* of a variable x in a term t are defined, by induction, as follows:

- if t is the variable x, then the occurrence of x in t is free;
- if $t = (v)u$, then the free occurrences of x in t are those off x in u and v;
- if $t = \lambda y.u$, then the free occurrences of x in t are those of x in u, except if $x = y$; in that case, no occurrencee of x in t is free.

Free occurrences of variables

The *free occurrences* of a variable x in a term t are defined, by induction, as follows:

- if t is the variable x, then the occurrence of x in t is free;
- if $t = (v)u$, then the free occurrences of x in t are those off x in u and v;
- if $t = \lambda y.u$, then the free occurrences of x in t are those of x in u, except if $x = y$; in that case, no occurrencee of x in t is free.

Example: What are the free occurrences of x in $(\lambda x.x)x$?

Free occurrences of variables

The *free occurrences* of a variable x in a term t are defined, by induction, as follows:

- if t is the variable x, then the occurrence of x in t is free;
- if $t = (v)u$, then the free occurrences of x in t are those off x in u and v;
- if $t = \lambda y.u$, then the free occurrences of x in t are those of x in u, except if $x = y$; in that case, no occurrencee of x in t is free.

Example: What are the free occurrences of x in $(\lambda x.x)x$?
There is exactly one free occurrence of x in $(\lambda x.x)x$: it is the third one.

β-reduction

β-reduction is the way how λ-terms are executed.

β-reduction

β-reduction is the way how λ-terms are executed.

A *redex* is a term of the shape $(\lambda x.u)v$.

β-reduction

β-reduction is the way how λ-terms are executed.
A *redex* is a term of the shape $(\lambda x.u)v$.
We have $t \beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).

β-reduction

β-reduction is the way how λ-terms are executed.
A *redex* is a term of the shape $(\lambda x.u)v$.
We have $t \beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).
Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.

β-reduction

β-reduction is the way how λ-terms are executed.

A *redex* is a term of the shape $(\lambda x.u)v$.

We have $t\beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).

Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.

Example: $(\lambda x.\lambda y.(x)yy)vu\beta$

β-reduction

β-reduction is the way how λ-terms are executed.

A *redex* is a term of the shape $(\lambda x.u)v$.

We have $t \beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).

Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.

Example: $(\lambda x.\lambda y.(x)yy)vu\beta\ (\lambda y.(v)yy)u$

β-reduction

β-reduction is the way how λ-terms are executed.
A *redex* is a term of the shape $(\lambda x.u)v$.
We have $t\beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).
Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.
Example: $(\lambda x.\lambda y.(x)yy)vu\beta$ $(\lambda y.(v)yy)u$
and $(\lambda y.(v)yy)u\beta$

β-reduction

β-reduction is the way how λ-terms are executed.

A *redex* is a term of the shape $(\lambda x.u)v$.

We have $t\beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).

Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.

Example: $(\lambda x.\lambda y.(x)yy)vu\beta\ (\lambda y.(v)yy)u$ and $(\lambda y.(v)yy)u\beta\ (v)uu$.

β-reduction

β-reduction is the way how λ-terms are executed.
A *redex* is a term of the shape $(\lambda x.u)v$.
We have $t \beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).
Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.
Example: $(\lambda x.\lambda y.(x)yy)vu\beta\ (\lambda y.(v)yy)u$
and $(\lambda y.(v)yy)u\beta\ (v)uu$.
One can have several occurrences of redexes.

β-reduction

β-reduction is the way how λ-terms are executed.

A *redex* is a term of the shape $(\lambda x.u)v$.

We have $t\beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).

Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.

Example: $(\lambda x.\lambda y.(x)yy)vu\beta\ (\lambda y.(v)yy)u$
and $(\lambda y.(v)yy)u\beta\ (v)uu$.

One can have several occurrences of redexes. Example:
$(\lambda z.z)\underbrace{(\lambda y.y)x}$

β-reduction

β-reduction is the way how λ-terms are executed.
A *redex* is a term of the shape $(\lambda x.u)v$.
We have $t\beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).
Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.
Example: $(\lambda x.\lambda y.(x)yy)vu\beta\ (\lambda y.(v)yy)u$
and $(\lambda y.(v)yy)u\beta\ (v)uu$.
One can have several occurrences of redexes. Example:
$(\lambda z.z)\underbrace{(\lambda y.y)x}\ \beta(\lambda z.z)x$

β-reduction

β-reduction is the way how λ-terms are executed.
A *redex* is a term of the shape $(\lambda x.u)v$.
We have $t \beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).
Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.
Example: $(\lambda x.\lambda y.(x)yy)vu\beta\ (\lambda y.(v)yy)u$
and $(\lambda y.(v)yy)u\beta\ (v)uu$.
One can have several occurrences of redexes. Example:
$(\lambda z.z)\underbrace{(\lambda y.y)x}\ \beta(\lambda z.z)x$ and $\underbrace{(\lambda z.z)(\lambda y.y)x}\beta(\lambda y.y)x$
One can have no redex at all. Example:

β-reduction

β-reduction is the way how λ-terms are executed.

A *redex* is a term of the shape $(\lambda x.u)v$.

We have $t \beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).

Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.

Example: $(\lambda x.\lambda y.(x)yy)vu\beta\ (\lambda y.(v)yy)u$
and $(\lambda y.(v)yy)u\beta\ (v)uu$.

One can have several occurrences of redexes. Example:
$(\lambda z.z)\underbrace{(\lambda y.y)x}\ \beta(\lambda z.z)x$ and $\underbrace{(\lambda z.z)(\lambda y.y)x}\beta(\lambda y.y)x$

One can have no redex at all. Example: $\lambda x.x$

β-reduction

β-reduction is the way how λ-terms are executed.
A *redex* is a term of the shape $(\lambda x.u)v$.
We have $t\beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).
Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.
Example: $(\lambda x.\lambda y.(x)yy)vu\beta\ (\lambda y.(v)yy)u$
and $(\lambda y.(v)yy)u\beta\ (v)uu$.
One can have several occurrences of redexes. Example:
$(\lambda z.z)\underbrace{(\lambda y.y)x}\beta(\lambda z.z)x$ and $\underbrace{(\lambda z.z)(\lambda y.y)x}\beta(\lambda y.y)x$
One can have no redex at all. Example: $\lambda x.x$
In this case one says that the term is *normal*.

β-reduction

β-reduction is the way how λ-terms are executed.

A *redex* is a term of the shape $(\lambda x.u)v$.

We have $t \beta t'$ if, and only if, t' is obtained from t by replacing some occurrence of some redex $(\lambda x.u)v$ by $v[u/x]$, which is the term v in which every free occurrence of x in v is substituted by v (we *contracted the redex* $(\lambda x.u)v$).

Intuition: $\lambda x.u$ is like a function f that associates with x the value u (like in $f(x) = u$). Now $(\lambda x.u)v$ is the value of $f(v)$: if $f(x) = x^2 + x + 1$, then $f(3)$ is $f(x)$ in which every occurrence of x is substituted by 3, which gives $3^2 + 3 + 1$.

Example: $(\lambda x.\lambda y.(x)yy)vu\beta\ (\lambda y.(v)yy)u$

and $(\lambda y.(v)yy)u\beta\ (v)uu$.

One can have several occurrences of redexes. Example:

$(\lambda z.z)\underbrace{(\lambda y.y)x}\ \beta(\lambda z.z)x$ and $\underbrace{(\lambda z.z)(\lambda y.y)x}\beta(\lambda y.y)x$

One can have no redex at all. Example: $\lambda x.x$

In this case one says that the term is *normal*. And one says that a term t is *normalizable* if there exists a normal λ-term t' such that $t\beta^* t'$.

Reduction graphs

The *reduction graph* of a term t is the directed graph such that:

Reduction graphs

The *reduction graph* of a term t is the directed graph such that:

- its vertices are all the terms t' such that $t\beta^* t'$ (β^* is the reflexive transitive closure of β);

Reduction graphs

The *reduction graph* of a term t is the directed graph such that:

- its vertices are all the terms t' such that $t\beta^* t'$ (β^* is the reflexive transitive closure of β);
- there is a directed arrow from v to v' iff $v\beta v'$; moreover this arrow is labelled by the redex one contracted to go from v to v'.

Reduction graphs

The *reduction graph* of a term t is the directed graph such that:

- its vertices are all the terms t' such that $t\beta^* t'$ (β^* is the reflexive transitive closure of β);
- there is a directed arrow from v to v' iff $v\beta v'$; moreover this arrow is labelled by the redex one contracted to go from v to v'.

Example: What is the reduction graph of $(\lambda z.z)(\lambda y.y)x$?

Reduction graphs

The *reduction graph* of a term t is the directed graph such that:

- its vertices are all the terms t' such that $t\beta^* t'$ (β^* is the reflexive transitive closure of β);
- there is a directed arrow from v to v' iff $v\beta v'$; moreover this arrow is labelled by the redex one contracted to go from v to v'.

Example: What is the reduction graph of $(\lambda z.z)(\lambda y.y)x$?

$$
\begin{tikzcd}
(I)(I)x \arrow[d, bend right, "(I)(I)x"'] \arrow[d, bend left, "(I)x"] \\
(I)x \arrow[d, "(I)x"] \\
x
\end{tikzcd}
$$

where $I = \lambda z.z$

Questions

1) Does there exist any non-normalizable λ-term?

Questions

1) Does there exist any non-normalizable λ-term?

Questions

1) Does there exist any non-normalizable λ-term?

2) Does there exist a term with a non-finite reduction graph?

Questions

1) Does there exist any non-normalizable λ-term?

2) Does there exist a term with a non-finite reduction graph?

3) If yes, does it imply that this term is non-normalizable?

Questions

1) Does there exist any non-normalizable λ-term?

2) Does there exist a term with a non-finite reduction graph?

3) If yes, does it imply that this term is non-normalizable?

4) Is a λ-term with a finite reduction graph necessarily normalizable?

Questions

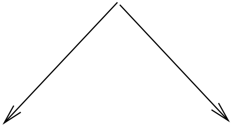
1) Does there exist any non-normalizable λ-term?

2) Does there exist a term with a non-finite reduction graph?

3) If yes, does it imply that this term is non-normalizable?

4) Is a λ-term with a finite reduction graph necessarily normalizable?

5) (More difficult) Does there exist a term whose reduction graph has the following shape?



Church numerals

For any integer n, the Church numeral $\ulcorner n \urcorner$ is the λ-term

$$\lambda f.\lambda x.\underbrace{(f)\ldots(f)}_{n\ times}x$$

Church numerals

For any integer n, the Church numeral $\ulcorner n \urcorner$ is the λ-term

$$\lambda f.\lambda x.\underbrace{(f)\dots(f)}_{n\ times}x$$

Examples:

Church numerals

For any integer n, the Church numeral $\ulcorner n \urcorner$ is the λ-term

$$\lambda f.\lambda x.\underbrace{(f)\ldots(f)}_{n\ times} x$$

Examples:

- $\ulcorner 0 \urcorner = \lambda f.\lambda x.x$

Church numerals

For any integer n, the Church numeral $\ulcorner n \urcorner$ is the λ-term

$$\lambda f.\lambda x.\underbrace{(f)\dots(f)}_{n \text{ times}}x$$

Examples:

- $\ulcorner 0 \urcorner = \lambda f.\lambda x.x$
- $\ulcorner 1 \urcorner = \lambda f.\lambda x.(f)x$

Church numerals

For any integer n, the Church numeral $\ulcorner n \urcorner$ is the λ-term

$$\lambda f.\lambda x.\underbrace{(f)\ldots(f)}_{n\ times}x$$

Examples:

- $\ulcorner 0 \urcorner = \lambda f.\lambda x.x$
- $\ulcorner 1 \urcorner = \lambda f.\lambda x.(f)x$
- $\ulcorner 2 \urcorner = \lambda f.\lambda x.(f)(f)x$

Church numerals

For any integer n, the Church numeral $\ulcorner n \urcorner$ is the λ-term

$$\lambda f.\lambda x.\underbrace{(f)\dots(f)}_{n\ times}x$$

Examples:

- $\ulcorner 0 \urcorner = \lambda f.\lambda x.x$
- $\ulcorner 1 \urcorner = \lambda f.\lambda x.(f)x$
- $\ulcorner 2 \urcorner = \lambda f.\lambda x.(f)(f)x$

Can you program the successor, the addition, the substraction, the multiplication...?

Church numerals

For any integer n, the Church numeral $\ulcorner n \urcorner$ is the λ-term

$$\lambda f.\lambda x.\underbrace{(f)\dots(f)}_{n\ times}x$$

Examples:

- $\ulcorner 0 \urcorner = \lambda f.\lambda x.x$
- $\ulcorner 1 \urcorner = \lambda f.\lambda x.(f)x$
- $\ulcorner 2 \urcorner = \lambda f.\lambda x.(f)(f)x$

Can you program the successor, the addition, the substraction, the multiplication...?
Let us try to program the successor:

Church numerals

For any integer n, the Church numeral $\ulcorner n \urcorner$ is the λ-term

$$\lambda f.\lambda x.\underbrace{(f)\dots(f)}_{n \ times} x$$

Examples:

- $\ulcorner 0 \urcorner = \lambda f.\lambda x.x$
- $\ulcorner 1 \urcorner = \lambda f.\lambda x.(f)x$
- $\ulcorner 2 \urcorner = \lambda f.\lambda x.(f)(f)x$

Can you program the successor, the addition, the substraction, the multiplication...?

Let us try to program the successor: we look for a λ-term t such that $(t)\lambda f.\lambda x.\underbrace{(f)\dots(f)}_{n \ times} x$

Church numerals

For any integer n, the Church numeral $\ulcorner n \urcorner$ is the λ-term

$$\lambda f.\lambda x.\underbrace{(f)\ldots(f)}_{n\ times}x$$

Examples:

- $\ulcorner 0 \urcorner = \lambda f.\lambda x.x$
- $\ulcorner 1 \urcorner = \lambda f.\lambda x.(f)x$
- $\ulcorner 2 \urcorner = \lambda f.\lambda x.(f)(f)x$

Can you program the successor, the addition, the substraction, the multiplication...?

Let us try to program the successor: we look for a λ-term t such that $(t)\lambda f.\lambda x.\underbrace{(f)\ldots(f)}_{n\ times}x \ \beta^* \lambda f.\lambda x.\underbrace{(f)\ldots(f)}_{n+1\ times}x$.

Church numerals

For any integer n, the Church numeral $\ulcorner n \urcorner$ is the λ-term

$$\lambda f.\lambda x.\underbrace{(f)\ldots(f)}_{n \textit{ times}}x$$

Examples:

- $\ulcorner 0 \urcorner = \lambda f.\lambda x.x$
- $\ulcorner 1 \urcorner = \lambda f.\lambda x.(f)x$
- $\ulcorner 2 \urcorner = \lambda f.\lambda x.(f)(f)x$

Can you program the successor, the addition, the substraction, the multiplication...?

Let us try to program the successor: we look for a λ-term t such that $(t)\lambda f.\lambda x.\underbrace{(f)\ldots(f)}_{n \textit{ times}}x \ \beta^* \lambda f.\lambda x.\underbrace{(f)\ldots(f)}_{n+1 \textit{ times}}x$.

$$t = \lambda n.\lambda f.\lambda x.(f)(n)fx$$

Strong normalizability

Consider a λ-term with the following reduction graph G:

Strong normalizability

Consider a λ-term with the following reduction graph G:

Is it normalizable?

Strong normalizability

Consider a λ-term with the following reduction graph G:

Is it normalizable? YES

Strong normalizability

Consider a λ-term with the following reduction graph G: 

Is it normalizable? YES

We say that a term t is *strongly normalizable* if there exists no infinite sequence $(t_i)_{i\in\mathbb{N}}$ of λ-terms such that

- $t_0 = t$
- and $(\forall i \in \mathbb{N})t_i \beta t_{i+1}$

Strong normalizability

Consider a λ-term with the following reduction graph G:
Is it normalizable? YES
We say that a term t is *strongly normalizable* if there exists no infinite sequence $(t_i)_{i\in\mathbb{N}}$ of λ-terms such that

- $t_0 = t$
- and $(\forall i \in \mathbb{N}) t_i \beta t_{i+1}$

Consider again a λ-term with the reduction graph G: is it strongly normalizable?

Strong normalizability

Consider a λ-term with the following reduction graph G:
Is it normalizable? YES
We say that a term t is *strongly normalizable* if there exists no infinite sequence $(t_i)_{i \in \mathbb{N}}$ of λ-terms such that

- $t_0 = t$
- and $(\forall i \in \mathbb{N}) t_i \beta t_{i+1}$

Consider again a λ-term with the reduction graph G: is it strongly normalizable? NO