ISR 2019

2019 07 05

λ-calculus

lecture 1

Femke van Raamsdonk

overview

- introduction
- terms
- reduction
- fixed point combinators
- Curry's paradox
- definability

overview

- introduction
- terms
- reduction
- fixed point combinators
- Curry's paradox
- definability

λ-calculus

inventor: Alonzo Church (1936)

a language expressing functions or algorithms

concept of computability and basis of functional programming

a language expressing proofs

untyped and typed

historical note: notation for functions

Frege defined the graph of a function (1893)

Russell and Whitehead and Russell (1910)

Schönfinkel defined function calculus (1920)

Curry defined combinary logic (1920)

Combinatory Logic (CL)

inventor Moses Schönfinkel (1924)

rewrite rules

$$\begin{aligned} \mathsf{I}\,X &\to X \\ (\mathsf{K}\,X)\,Y &\to X \\ ((\mathsf{S}\,X)\,Y)\,Z &\to (X\,Z)\,(Y\,Z) \end{aligned}$$

rewriting

I can be defined

$\mathsf{S}\,\mathsf{K}\,K\,x \to (\mathsf{K}\,x)\,(\mathsf{K}\,x) \to x$

rewriting may be infinite

$(\mathsf{S}\,\mathsf{I}\,\mathsf{I})\,(\mathsf{S}\,\mathsf{I}\,\mathsf{I}) \to \mathsf{I}\,(\mathsf{S}\,\mathsf{I}\,\mathsf{I})\,(\mathsf{I}\,(\mathsf{S}\,\mathsf{I}\,\mathsf{I})) \to$
$(\mathsf{S}\,\mathsf{I}\,\mathsf{I})\,(\mathsf{I}\,(\mathsf{S}\,\mathsf{I}\,\mathsf{I})) \to (\mathsf{S}\,\mathsf{I}\,\mathsf{I})\,(\mathsf{S}\,\mathsf{I}\,\mathsf{I})$

play with combinators

define $D = \mathsf{S}\,\mathsf{I}\,\mathsf{I}$

then $D\,x =_{CL} x\,x$

define $B = \mathsf{S}\,(\mathsf{K}\,\mathsf{S})\,\mathsf{K}$

then $B\,f\,g\,x =_{CL} f\,(g\,x)$

define $L = D\,(B\,D\,D)$

then $L =_{CL} L\,L$

extending and restricting

extending CL leads to first-order rewriting

restricting CL leads to studying the rule for S

extending λ leads to higher-order rewriting

slogan-like: λ : HRS = CL : TRS

overview

- introduction
- terms
- reduction
- fixed point combinators
- Curry's paradox
- definability

notation for (anonymous) functions

mathematical notation:

$$f : \mathsf{nat} \to \mathsf{nat}$$
$$f(x) = \mathsf{square}(x)$$

or also:

$$f : \mathsf{nat} \to \mathsf{nat}$$
$$f : x \mapsto \mathsf{square}(x)$$

lambda notation:

$$\lambda x.\, \mathsf{square}\, x$$

we start with the untyped λ-calculus

lambda terms: intuition

abstraction:

$\lambda x.\, M$ is the function mapping x to M

$\lambda x.\, x$ is the function mapping x to x

$\lambda x.\, \mathsf{square}\, x$ is the function mapping x to $\mathsf{square}\, x$

application:

$F\ M$ is the application of the function F to its argument M

(not the result of applying)

lambda terms: inductive definition

we assume a countably infinite set of variables $(x, y, z \ldots)$

sometimes we in addition assume a set of contstants

the set of λ-terms is defined inductively by the following clauses:

a variable x is a λ-term

a constant c is a λ-term

if M is a λ-term, then $(\lambda x.\, M)$ is a λ-term, called an abstraction

if F and M are λ-terms, then $(F\, M)$ is a λ-term, called an application

famous terms

$\mathsf{I} = (\lambda x.\, x) = \lambda x.\, x$

$\mathsf{K} = \lambda x.\, (\lambda y.\, x) = \lambda x.\, \lambda y.\, x$

$\mathsf{S} = \lambda x.\, \lambda y.\, \lambda z.\, (x\, z)\, (y\, z) = \lambda x.\, \lambda y.\, \lambda z.\, x\, z\, (y\, z)$

$\Omega = (\lambda x.\, x\, x)\, (\lambda x.\, x\, x)$

omit outermost parentheses

application is associative to the left

abstraction is associative to the right

lambda extends to the right as far as possible

terms as trees

a subterm corresponds to a subtree

subterms of $\lambda x.\, y$ are $\lambda x.\, y$ and y

bound variables: definition

x is bound by the first λx above it in the term tree

examples: the underlined x is bound in

$\lambda x. \underline{x}$

$\lambda x. \underline{x}\, \underline{x}$

$(\lambda x. \underline{x})\, x$

$\lambda x. y\, \underline{x}$

$\lambda x. \lambda x. \underline{x}$

free variables: definition

a variable that is not bound is free

alternatively: define recursively the set $\mathsf{FV}(M)$ of free variables of M:

$$\begin{aligned}\mathsf{FV}(x) &= \{x\}\\ \mathsf{FV}(c) &= \emptyset\\ \mathsf{FV}(\lambda x.\, M) &= \mathsf{FV}(M)\backslash\{x\}\\ \mathsf{FV}(F\, P) &= \mathsf{FV}(F)\cup\mathsf{FV}(P)\end{aligned}$$

a term is closed if it has no free variables

currying

reduce a function with several arguments to functions with single arguments

example:

$f : x \mapsto x + x$ becomes $\lambda x.\, x + x$

$g : (x, y) \mapsto x + y$ becomes $\lambda x.\, \lambda y.\, x + y$, not $\lambda(x, y).\, \mathsf{plus}\, x\, y$

partial application:

$(\lambda x.\, \lambda y.\, x + y)\, 3$

history:

due to Frege, Schönfinkel, and Curry

related to the isomorphism between $A \times B \to C$ and $A \to (B \to C)$

towards computation

we will use terms to compute, as for example in

$$(\lambda x.\, f\ x)\, 5 \rightarrow_\beta (f\ x)[x := 5] = f\ 5$$

the definition of substitution requires more preparation

intuitive meaning of $M[x := N]$:

the result of replacing in M all free occurrences of x by N

substititition: recursive definition

substitution in a variable or a constant:

$x[x := N] = N$

$a[x := N] = a$ with $a \neq x$ a variable or a constant

substitution in an application:

$(P\,Q)[x := N] = (P[x := N])\,(Q[x := N])$

substitution in an abstraction:

$(\lambda x.\,P)[x := N] = \lambda x.\,P$

$(\lambda y.\,P)[x := N] = \lambda y.\,(P[x := N])$ if $x \neq y$ and $y \notin \mathrm{FV}(N)$

$(\lambda y.\,P)[x := N] = \lambda z.\,(P[y := z][x := N])$
if $x \neq y$ and $z \notin \mathrm{FV}(N) \cup \mathrm{FV}(P)$ and $y \in \mathrm{FV}(N))$

substitution: examples

$(\lambda x.\, x)[x := \mathsf{c}] = \lambda x.\, x$

$(\lambda x.\, y)[y := \mathsf{c}] = \lambda x.\, \mathsf{c}$

$(\lambda x.\, y)[y := x] = \lambda z.\, x$

$(\lambda y.\, x\,(\lambda w.\, v\, w\, x))[x := u\, v] = \lambda y.\, u\, v(\lambda w.\, v\, w\,(u\, v))$

$(\lambda y.\, x\,(\lambda x.\, x))[x := \lambda y.\, x\, y] = \lambda y.(\lambda y.\, x\, y)\,(\lambda x.\, x)$

alpha conversion

intuition:

bound variables may be renamed

example:

$\lambda x.\, x =_\alpha \lambda y.\, y$

definition α-conversion axiom:

$\lambda x.\, M =_\alpha \lambda y.\, M[x := y]$ with $y \notin FV(M)$

definition α-equivalence relation $=_\alpha$: on terms

$P =_\alpha Q$ if Q can be obtained from P

by finitely many 'uses' of the α-conversion axiom

that is: by finitely many renamings of bound variables in context

alpha equivalence classes

we identify α-equivalent λ-terms

just as we identify $f : x \mapsto x^2$ and $f : y \mapsto y^2$

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

we work with equivalence classes modulo α

examples

which of the following pairs of terms are α-equivalent?

$\lambda x.\, x\, y$ and $\lambda y.\, y\, y$

$\lambda x.\, x\, y$ and $\lambda u.\, u\, y$

$\lambda x.\, x\, y$ and $\lambda x.\, x\, u$

$x\,(\lambda x.\, x)$ and $y\,(\lambda y.\, y)$

alpha-conversion and substitution: intuitive approach

we defined first substitution $[x := P]$ and then α using substitution $[x := y]$

an alternative intuitive approach:

define α as renaming of bound variables

work modulo α

define substitution $M[x := N]$ using renaming of bound variables:

replace all free occurrences of x in M by N,

rename bound variables if necessary

example: $(\lambda x.y)[y := x] =_\alpha (\lambda x'.y)[y := x] = \lambda x'.x$

now we know the statics of the lambda-calculus

we consider λ-terms modulo α-conversion

application and abstraction

bound and free variables

currying

substitution

we continue with the dynamics: β-reduction

overview

- introduction
- terms
- reduction
- fixed point combinators
- Curry's paradox
- definability

definition beta reduction

the β-reduction rule: $(\lambda x.\, M)\, N \to_\beta M[x := N]$

here we have the following:

x is a variable

M and N are terms

$[x := N]$ is the substitution of N for x

definition beta reduction

the β-reduction rule: $(\lambda x.\, M)\, N \to_\beta M[x := N]$

the beta-reduction relation is obtained using

$$\frac{M \to_\beta M'}{\lambda x.\, M \to_\beta \lambda x.\, M'}$$

$$\frac{M \to_\beta M'}{M\, N \to_\beta M'\, N}$$

$$\frac{N \to_\beta N'}{M\, N \to_\beta M\, N'}$$

beta reduction: examples

$(\lambda x.\, x)\, y \to_\beta x[x := y] = y$

$(\lambda x.\, x\, x)\, y \to_\beta (x\, x)[x := y] = y\, y$

$(\lambda x.\, x\, z)\, y \to_\beta (x\, z)[x := y] = y\, z$

$(\lambda x.\, z)\, y \to_\beta z[x := y] = z$

$\Omega = (\lambda x.\, x\, x)\, (\lambda x.\, x\, x) \to_\beta \Omega$

$\mathsf{K}\, \mathsf{I}\, \Omega \to_\beta \mathsf{K}\, \mathsf{I}\, \Omega$ and also $\mathsf{K}\, \mathsf{I}\, \Omega \to_\beta (\lambda y.\mathsf{I})\, \Omega \to_\beta \mathsf{I}$

terminology and notation as for TRSs

β-redex

β-reduction step $\rightarrow_\beta$

β-reduction sequence or β-rewrite sequence $\rightarrow_\beta^*$

β-conversion $=_\beta$

β-normal form (NF)

strongly normalizing (SN) or terminating

weakly normalizing (WN)

beta reduction

is a model of computation

is non-deterministic

however: gives unique normal forms

see: confluence

is non-terminating

however: there are normalizing strategies

see: strategies

we really need renaming

α is a source of problems but we cannot do without:

$$(\lambda x.\, x\, x)\,(\lambda s.\, \lambda z.\, s\, z) \quad \to_\beta$$

$$(\lambda s.\, \lambda z.\, s\, z)\,(\lambda s.\, \lambda z.\, s\, z) \quad \to_\beta$$

$$\lambda z.\, (\lambda s.\, \lambda z.\, s\, z)\, z \quad \to_\beta$$

$$\lambda z.\, \lambda z'.\, z z'$$

De Bruijn notation

instead of names use a reference to the binding λ

$\lambda x.\, x$ is $\lambda 1$

$\lambda x. \lambda y. x$ is $\lambda\,\lambda\,2$

another rule: eta

$$\lambda x.\, M\, x \to_\eta M \qquad \text{if } x \text{ not free in } M$$

we do *not* have the step:

$$\lambda x.\, x\, x \to_\eta x$$

overview

- introduction
- terms
- reduction
- fixed point combinators
- Curry's paradox
- definability

fixed point

definition:

$x \in A$ is a fixed point of $f : A \to B$ if $f(x) = x$

examples:

0 and 1 are fixed points of $f : \mathbb{R} \to \mathbb{R}$ with $x \mapsto x^2$

for λ-calculus:

M is a fixed point of F if $F\, M =_\beta M$

example:

every term M is a fixed point of I because $\mathsf{I}\, M =_\beta M$

fixed point combinator

definition:

Y is a fixed point combinator if

$F\,(Y\,F) =_\beta Y\,F$ for every λ-term F

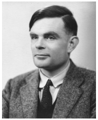
informally:

we can use Y to construct a fixed point for a given term F

fixed point combinators

Curry's fixed point combinator:

$$\mathsf{Y} = \lambda f.\,(\lambda x.\,f\,(x\,x))\,(\lambda x.\,f\,(x\,x))$$



Turing's fixed point combinator:

$$\mathsf{T} = (\lambda x.\,\lambda y.\,y\,(x\,x\,y))\,(\lambda x.\,\lambda y.\,y\,(x\,x\,y))$$

consider Curry's fixed point combinator

for an arbirary F we have:

$$\begin{array}{rcl} \mathrm{Y}\,F & = & (\lambda f.\,(\lambda x.\,f\,(x\,x))\,(\lambda x.\,f\,(x\,x)))\,F \\ & \to_\beta & (\lambda x.\,F\,(x\,x))\,(\lambda x.\,F\,(x\,x)) \\ & \to_\beta & F\,((\lambda x.\,F\,(x\,x))\,(\lambda x.\,F\,(x\,x))) \\ & \leftarrow & F\,((\lambda f.\,(\lambda x.\,f\,(x\,x))\,(\lambda x.\,f\,(x\,x)))\,F) \\ & = & F\,(\mathrm{Y}\,F) \end{array}$$

and also:

$$F\,((\lambda x.\,F\,(x\,x))\,(\lambda x.\,F\,(x\,x))) \to_\beta F\,(F\,((\lambda x.\,F\,(x\,x))\,(\lambda x.\,F\,(x\,x))))$$

consider Turing's fixed point combinator

for an arbitrary F we have:

$$\begin{array}{rcl} \mathsf{T}\,F & = & (\lambda x.\,\lambda y.\,y\,(x\,x\,y))\,(\lambda x.\,\lambda y.\,y\,(x\,x\,y))\,F \\ & \to_\beta & (\lambda y.\,y\,(\mathsf{t}\,\mathsf{t}\,y))\,F \\ & \to_\beta & F\,(\mathsf{tt}\,F) \\ & = & F\,(\mathsf{T}F) \end{array}$$

with $\mathsf{t} = \lambda x.\,\lambda y.\,y\,(x\,x\,y)$

example (Hindley)

question: define X such that $X\,y =_\beta X$ (a garbage dosposer)

$X\,y =_\beta X$

follows from $X =_\beta \lambda y.\,X$

follows from $X =_\beta (\lambda x.\,\lambda y.\,x)\,X$

follows from $X =_\beta \mathsf{Y}\,(\lambda x.\,\lambda y.\,x)$

so define $X = \mathsf{Y}\,(\lambda x.\,\lambda y.\,x)$

example (Hindley)

question: define X such that $X\,y\,z =_\beta X\,z\,y$ (bureaucrat)

$X\,y\,z =_\beta X\,z\,y$

follows from $X = \lambda y.\,\lambda z.\,X\,z\,y$

follows from $X = (\lambda x.\,\lambda y.\,\lambda z.\,x\,z\,y)\,X$

follows from $X = \mathsf{Y}\,(\lambda x.\,\lambda y.\,\lambda z.\,x\,z\,y)$

so defined $X = \mathsf{Y}\,(\lambda x.\,\lambda y.\,\lambda z.\,x\,z\,y)$

overview

- introduction
- terms
- reduction
- fixed point combinators
- Curry's paradox
- definability

inconsistency

Kleene and Rosser discovered in 1934 that

Church's system and Curry's combinatory logic are inconsistent

they encoded Richard's paradox

Curry presented a new exposition of the paradox

then Curry showed inconsistency via Curry's paradox

Church's orginal system: terms

terms formed by application, so $M\,N$

terms formed by abstraction, so $\lambda x.\,M$

a rule for changing the names of bound variables, so α-conversion

a rule for calculating the values of a function, so β-reduction

Church's original system: logic

atomic constants for representing logical connectives and quantifiers

we write implication with $\to$ in infix notation

a notion of provability

modus ponens (MP): if $A \to B$ and A provable then B provable

$(A \to (A \to B)) \to (A \to B)$ provable

if A is provable and $A =_\beta A'$ then A' is provable

notation

on the following three slides:

the logic part is in black

the part in Church's sytem is in blue

tautology of prop1

$(A \to (A \to B)) \to (A \to B)$ is a tautology of first-order propositional logic:

$$\dfrac{\dfrac{\dfrac{\dfrac{A \to (A \to B) \qquad A}{A \to B}\, E\to \qquad A}{B}}{A \to B}}{(A \to (A \to B)) \to (A \to B)}$$

we assume $(A \to (A \to B)) \to (A \to B)$ provable in Church's system

next step

if $A = A \to B$ then we have:

$$\dfrac{(A \to (A \to B)) \to (A \to B) \qquad \dfrac{\dfrac{A}{A \to A}}{A \to (A \to B)} =}{A \to B}$$

we define $A = \mathrm{Y}\,(\lambda x.\, x \to (x \to B))$ for an arbitrary B

then $A =_\beta (\lambda x.\, x \to (x \to B))\, A =_\beta A \to (A \to B)$

we have $(A \to (A \to B)) \to (A \to B)$ provable (system)

using β we have $A \to (A \to B)$ provable

using MP we have $A \to B$ provable

next step

if $A \to B$ provable and using $A = A \to B$ then we have:

$$\dfrac{A \to B \qquad \dfrac{A \to B}{A}=}{B}$$

we have $A \to (A \to B)$ provable as shown on the previous slide

using β we have A provable

we have $A \to B$ provable as shown on the previous slide

using MP we have B provable, and B was arbitrary

what now?

Church restricted attention to the part dealing with functions:

the λ-calculus

Curry had already shown

the corresponding part of his system to be consistent (1930)

Church and Rosser proved consistency of the λ-calculus in 1936

via what is known as the Church-Rosser theorem

overview

- introduction
- terms
- reduction
- fixed point combinators
- Curry's paradox
- definability
  - booleans

expressive power

the λ-calculus is Turing-complete

Church's thesis: everything that is computable

is definable in the pure untyped lambda calculus

we illustrate the expressive power

by considering the encoding of several data-types

booleans as lambda-terms: idea

we try to find:

two

different

closed

normal forms

permitting to calculate

booleans and negations as lambda terms: definition

definition of term for true

$$\mathsf{true} = \lambda xy.\, x$$

definition of term for false

$$\mathsf{false} = \lambda xy.\, y$$

negation

$$\mathsf{not} = \lambda x.\, x\, \mathsf{false}\, \mathsf{true}$$

indeed

$$\mathsf{not}\, \mathsf{true} =_\beta (\lambda x.\, x\, \mathsf{false}\, \mathsf{true})\, \mathsf{true} =_\beta \mathsf{true}\, \mathsf{false}\, \mathsf{true} =_\beta \mathsf{false}$$

define other operations on booleans

$\mathsf{true} = \lambda xy.\, x$

$\mathsf{false} = \lambda xy.\, y$

$\mathsf{not} = \lambda x.\, x\, \mathsf{false}\, \mathsf{true}$

$\mathsf{ite} = \lambda bxy.\, b\, x\, y$

$\mathsf{and} = \lambda xy.\, x\, y\, \mathsf{false}$

$\mathsf{or} = \lambda xy.\, x\, \mathsf{true}\, y$

$\mathsf{xor} = \lambda xy.\, x\, (\mathsf{not}\, y)\, y$