ISR 2019

2019 07 06

λ-calculus

lecture 2

Femke van Raamsdonk

overview

- definability
- confluence
- simply typed lambda calculus
- strategies

overview

- definability
  - natural numbers
  - pairs
  - lists
  - recursive functions
- confluence
- simply typed lambda calculus
- strategies

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

natural numbers as lambda terms

we try to find:

inifinitely many

different

closed

normal forms

permitting to calculate

Church numerals

numerals

$$\begin{aligned}
c_0 &= \lambda s.\, \lambda z.\, z \\
c_1 &= \lambda s.\, \lambda z.\, s\, z \\
c_2 &= \lambda s.\, \lambda z.\, s\,(s\, z) \\
c_3 &= \lambda s.\, \lambda z.\, s\,(s\,(s\, z)) \\
c_4 &= \lambda s.\, \lambda z.\, s\,(s\,(s\,(s\, z))) \\
&\vdots \\
c_n &= \lambda s.\, \lambda z.\, s^n(z)
\end{aligned}$$

successor

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

indeed

$$\mathsf{S}\, c_0 = (\lambda x.\, \lambda s.\, \lambda z.\, s\,(x\, s\, z))\, c_0 =_\beta \lambda sz.\, (\lambda s'.\, \lambda z'.\, z')\, s\,(s\, z) =_\beta \lambda sz.\, s\, z$$

define other operations

$$\begin{array}{lcl}
c_0 & = & \lambda s.\, \lambda z.\, z \\
c_1 & = & \lambda s.\, \lambda z.\, s\, z \\
c_2 & = & \lambda s.\, \lambda z.\, s\,(s\, z) \\
c_3 & = & \lambda s.\, \lambda z.\, s\,(s\,(s\, z)) \\
c_4 & = & \lambda s.\, \lambda z.\, s\,(s\,(s\,(s\, z))) \\
 & \vdots & \\
c_n & = & \lambda s.\, \lambda z.\, s^n(z) \\
\mathsf{S} & = & \lambda x.\, \lambda s.\, \lambda z.\, s\,(x\, s\, z)
\end{array}$$

$\mathit{iszero} = \lambda n.\, n\,(\lambda y.\, \mathsf{false})\, \mathsf{true}$

$\mathsf{S}' = \lambda x.\, \lambda sz.\, x\, s\,(s\, z)$

arithmetic

addition:

plus $:= \lambda mn.\, \lambda sz.\, m\, s\, (n\, s\, z)$

multiplication:

mul $:= \lambda mn.\, \lambda sz.\, m\, (n\, s)\, z$

exponentiation:

exp $:= \lambda mn.\, n\, m$

definability

assume a natural number n is encoded in the λ-calculus by $[n]$

a function $f : \mathbb{N} \to \mathbb{N}$ is definable in the λ-calculus by a term F if

$F\,[n] =_\beta [f(n)]$ for every $n \in \mathbb{N}$

if we restrict attention to Church Numerals:

$F\ c_n =_\beta c_{f(n)}$

we have seen some definable functions, we even have:

$f : \mathbb{N} \to \mathbb{N}$ is computable iff f is λ-definable

pair: idea

we try to find:

a method to combine two terms in a pair

in such a way that a component can be extracted from the pair

pairs: definition

definition of pairing operator:

$$\pi := \lambda lrz.\, z\, l\, r$$

then:

$$\pi\, P\, Q =_\beta \lambda z.\, z\, P\, Q$$

projections::

$$\pi_1 := \lambda u.\, u\,(\lambda lr.\, l) = \lambda u.\, u\,\mathsf{true}$$

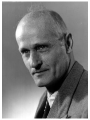
$$\pi_2 := \lambda u.\, u\,(\lambda lr.\, r) = \lambda u.\, u\,\mathsf{false}$$

then:

$$\pi_1\,(\pi\, P\, Q) =_\beta P$$

$$\pi_2\,(\pi\, P\, Q) =_\beta Q$$

predecessor



auxiliary definition:

$$\text{prefn} := \lambda f p.\, \pi\, \text{false}\, (\text{ite}\, (\pi_1\, p)\, (\pi_2\, p)\, (f\, (\pi_2\, p)))$$

then:

$$\text{prefn}\, f\, (\pi\, \text{true}\, x) =_\beta \pi\, \text{false}\, x$$

$$\text{prefn}\, f\, (\pi\, \text{false}\, x) =_\beta \pi\, \text{false}\, (f\, x)$$

definition predecessor:

$$\text{pred} := \lambda x.\, \lambda s z.\, \pi_2\, (x\, (\text{prefn}\, s)\, (\pi\, \text{true}\, z))$$

lists: idea

a list is obtained by repeatedly forming a pair

for example: $[1, 2, 3]$ is $(1, (2, (3, \mathsf{nil})))$

lists: definition

constructors:

$\mathsf{nil} := \lambda xy.\, y$

$\mathsf{cons} := \lambda ht.\, \lambda z.\, z\, h\, t = \pi$

definition:

$\mathsf{head} := \lambda l.\, l\, (\lambda ht.\, h) = \pi_1$

$\mathsf{tail} := \lambda l.\, l\, (\lambda ht.\, t) = \pi_2$

then:

$\mathsf{head}\, (\mathsf{cons}\, H\ T) =_\beta H$

$\mathsf{tail}\, (\mathsf{cons}\, H\ T) =_\beta T$

empty

how do we define a predicate empty on lists?

$\mathsf{cons}\, H\ T =_\beta \lambda z.\, z\, H\ T$

$\mathsf{nil} := \lambda xy.\, y$

$\mathsf{isempty} := \lambda l.\, l\, (\lambda x.\lambda y.\lambda z.\, \mathsf{false})\, \mathsf{true}$

alternative:

$\mathsf{nil} = \lambda z.\, \mathsf{true}$

$\mathsf{isempty} = \lambda l.\, l\, (\lambda x.\, \lambda y.\, \lambda z.\, \mathsf{false})$

recursive functions: examples in Haskell

```
factorial n = if (n==0)
  then 1
  else (n * factorial (n-1))
```

```
som [] = 0
som (h:t) = h + (som t)
```

```
length [] = 0
length (h:t) = (length t) + 1
```

how do we define length in lambda-calculus?

first idea:

length $= \lambda l.\,$if l is empty then zero, else length of tail of l plus 1

lists represented as nil $:= \lambda xy.\, y$ and cons $:= \lambda ht.\, \lambda z.\, z\, h\, t$

conditional represented as ite $= \lambda b.\, \lambda x. \lambda y.\, b\, x\, y$

check on empty represented as isempty $:= \lambda l.\, l\, (\lambda xyz.\, \mathsf{false})\, \mathsf{true}$

Church numerals with 0 represented as $c_0 = \lambda s.\, \lambda z.\, z$

tail represented as tail $= \lambda l.\, l\, (\lambda h.\, \lambda t.\, t)$

plus one represented as $S = \lambda x.\, \lambda sz.\, s\, (x\, s\, z)$

use fixed point combinator

So far we have:

$$\text{length} := \lambda l.\, \text{ite}\,(\text{isempty}\; l)\; c_0\, (\text{S}\, (\text{length}\, (\text{tail}\; l)))$$

which still contains length. Now using

$$M := \lambda a.\, \lambda l.\, \text{ite}\,(\text{isempty}\; l)\; c_0\, (\text{S}\, (a\, (\text{tail}\; l)))$$

we have

$$\text{length} =_\beta M\, \text{length}$$

So we actually look for a fixed point of M! So we take:

$$\text{length} := \text{Y}\; M$$

with M defined as above, and Y Curry's fixed point combinator

recursive functions: method

we try to define:

G with $G = \ldots G \ldots$

hence we look for:

G with $G = (\lambda g.\ \ldots g \ldots)\, G$

hence we take:

a fixed point of $\lambda g.\ \ldots g \ldots$

that is, using a fixed point combinator Y we define:

$G = \mathsf{Y}\,(\lambda g.\ \ldots g \ldots)$

from Haskell to lambda

Haskell is translated to core Haskell which can be seen as $\lambda+$

```
length [] = 0
length (h:t) = (length t) + 1
```

becomes

```
length l = case l of
                     [] -> 0
                     (h:t) -> 1 + length t
```

becomes (...) becomes roughly

$$\mathsf{Y}\,(\lambda a.\,\lambda l.\,\mathsf{if}\,(l == [])\,\mathsf{then}\,0\,\mathsf{else}\,(\lambda(h:t).\,(1+(a\,t)))\,l)$$

remark: lambda calculi with patterns

computation by reduction and pattern matching:

first projection:
$(\lambda\langle x, y\rangle.\, x)\, \langle 3, 5\rangle \to x[x, y := 3, 5] = 3$

length of a non-empty list:
$(\lambda(h : t).\, 1 + (\mathsf{length}\ t))\, (1 : \mathsf{nil}) \to (1 + (\mathsf{length}\ t))[t := \mathsf{nil}] = 1 + \mathsf{length}\, \mathsf{nil}$

further reading

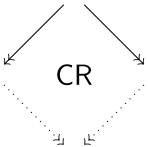
linear numeral systems
Ian Mackie
JAR 2018

overview

- definability
- confluence
- simply typed lambda calculus
- strategies

confluence: definition

every two coinitial rewrite sequences can be joined



confluence yields uniqueness of normal forms and consistency

how to prove confluence?

using Newman's Lemma:

SN and weak confluence $\Rightarrow$ confluence

but λ-calculus is not SN; see Ω

using a method due to Tait and Martin-Löf:

show the diamond property for a relation $\longmapsto$ with $\to_\beta \subseteq \longmapsto \subseteq \to_\beta^*$

what can we use for $\longmapsto$?

parallel beta-reduction

definition:

$x \Rightarrow_\beta x$

if $M \Rightarrow_\beta M'$ then $\lambda x.\, M \Rightarrow_\beta \lambda x.\, M'$

if $M \Rightarrow_\beta M'$ and $N \Rightarrow_\beta N'$ then $M\, N \Rightarrow_\beta M'\, N'$

$(\lambda x.\, M)\, N \Rightarrow_\beta M[x := N]$

example: $(\mathsf{I}\,\mathsf{I})\,(\mathsf{I}\,\mathsf{I}) \Rightarrow_\beta \mathsf{I}\,\mathsf{I}$

parallel reduction is a congruence

it is not the compatible closure of a reduction rule

parallel β-reduction does not have the diamond property

we have the divergence

$(\lambda x.\,(\lambda y.\,x)\,\mathsf{I})\,(\mathsf{I}\,\mathsf{I}) \Rightarrow_\beta (\lambda y.\,\mathsf{I}\,\mathsf{I})\,\mathsf{I}$

and

$(\lambda x.\,(\lambda y.\,x)\,\mathsf{I})\,(\mathsf{I}\,\mathsf{I}) \Rightarrow_\beta (\lambda x.\,x)\,\mathsf{I}$

the intended common reduct I cannot be reached with $\Rightarrow$ from $(\lambda y.\,\mathsf{I}\,\mathsf{I})\,\mathsf{I}$

the residual of $\mathsf{I}\,\mathsf{I}$ is nested in the residual of $(\lambda y.\,x)\,\mathsf{I}$

simlarly, parallel reduction for HRSs does not have the diamond property

multi-step beta-reduction

instead of parallel reduction we use multi-step reduction

which corresponds to a complete development

$x ⇴ x$

if $M ⇴ M'$ then $\lambda x.\, M ⇴ \lambda x.\, M'$

if $M ⇴ M'$ and $N ⇴ N'$ then $M\, N ⇴ M'\, N'$

if $M ⇴ M'$ and $N ⇴ N'$ then $(\lambda x.\, M)\, N ⇴ M'[x := N']$

examples

$$\frac{\mathsf{I}\,\mathsf{I} \mathrel{\circ\!\!\!\!\rightarrow} \mathsf{I} \qquad \mathsf{I} \mathrel{\circ\!\!\!\!\rightarrow} \mathsf{I}}{(\lambda y.\,\mathsf{I}\,\mathsf{I})\,\mathsf{I} \mathrel{\circ\!\!\!\!\rightarrow} \mathsf{I}}$$

$$\frac{x\,(\mathsf{I}\,a) \mathrel{\circ\!\!\!\!\rightarrow} x\,a \qquad \mathsf{I}\,\mathsf{I} \mathrel{\circ\!\!\!\!\rightarrow} \mathsf{I}}{(\lambda x.\,x\,(\mathsf{I}\,a))\,(\mathsf{I}\,\mathsf{I}) \mathrel{\circ\!\!\!\!\rightarrow} \mathsf{I}\,a}$$

$$\frac{\mathsf{I}\,\mathsf{I} \mathrel{\circ\!\!\!\!\rightarrow} \mathsf{I} \qquad \mathsf{I}\,a \mathrel{\circ\!\!\!\!\rightarrow} a}{(\mathsf{I}\,\mathsf{I})\,(I\,a) \mathrel{\circ\!\!\!\!\rightarrow} \mathsf{I}\,a}$$

limitations of multi-step reduction

we have $(\lambda x.\, \lambda y.\, x\, y)\, a\, b \rightarrow_\beta (\lambda y.\, a\, y)\, b \rightarrow_\beta a\, b$
but not $(\lambda x.\, \lambda y.\, x\, y)\, a\, b \mathrel{\circ\mkern-11mu\rightarrow} a\, b$

we have $\mathsf{I}\, (\lambda x.\, x)\, a \rightarrow_\beta (\lambda x.\, x)\, a \rightarrow_\beta a$
but not $\mathsf{I}\, (\lambda x.\, x)\, a \mathrel{\circ\mkern-11mu\rightarrow} a$

we have $(\lambda x.\, x\, a)\, (\lambda y.\, x) \rightarrow_\beta (\lambda y.\, y)\, a \rightarrow_\beta a$
but not $(\lambda x.\, x\, a)\, (\lambda y.\, y) \mathrel{\circ\mkern-11mu\rightarrow} a$

multi-step reduction corresponds to complete developments
where only residuals of initially present redexes are contracted

(Jean-Jacques Lévy, 1974)

uniform common reduct (Takahashi)

corresponds to complete development of the set of all redexes

$x^* = x$

$\lambda x.\, P^* = \lambda x.\, P^*$

$P\,Q^* = P^*\,Q^*$ if $P\,Q$ not a redex

$(\lambda x.\, P)\,Q^* = P^*[x := Q^*]$

for example:

$(\lambda x.\, x\,(\mathsf{I} a))\,(\mathsf{I}\, b)^* = b\, a$

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

confluence and hence consistency

$\mathrel{\ominus\!\!\!\!\rightarrow}$ has the triangle property: if $M \mathrel{\ominus\!\!\!\!\rightarrow} N$ then $N \mathrel{\ominus\!\!\!\!\rightarrow} M^*$

hence it has the diamond property

hence $\rightarrow_\beta$ is confluent

hence we have consistency:

$\lambda xy.\,x \neq_\beta \lambda xy.\,y$

adding eta yields critical pairs

$$M\,N \;{}_{\eta}\!\!\leftarrow (\lambda x.\, M\,x)\,N \rightarrow_{\beta} M\,N$$

$$\lambda x.\, M \;{}_{\beta}\!\!\leftarrow \lambda x.\,(\lambda u.\, M)\,x \rightarrow_{\eta} \lambda u.\, M$$

the critical pairs are trivial, so beta-eta is weakly orthogoanal
we can adapt the proof !

Z

Z-property:

if there is a map $*$ such that if $a \to b$ then $b \to^* a^*$ and $a^* \to b^*$

if $\to$ has the Z-property then $\to$ is confluent

we can use the uniform common reduct by Takahashi

(van Oostrom, Dehornoy)

further reading

Parallel Reductions in λ-calculus
Masako Takahashi
IC 118(1), pp. 120-127, 1995

More Church-Rosser proofs
Tobias Nipkow
JAR 26(1), pp. 51-66, 2001

A short machanized proof of the Church-Rosser Theorem by the Z-property for the $\lambda\beta$-calculus in Nomina Isabelle
Julian Nagele, Vincent van Oostrom, Christian Sternagel
5th IWC, pp. 55–59, 2016

further reading

Higher-order rewrite systems and their confluence
Richard Mayr and Tobias Nipkow
TCS 192, –. 3–29, 1998

Developing developments
Vincent van Oostrom
TCS 175, pp. 159–181, 1997

Modularity of Confluence – Constructed
Vincent van Oostrom
Proc. 4th IJCAR, LNAI 5195, pp. 348 – 363, 2008

Higher-Order (Non-)Modularity
Claus Appel, Vincent van Oostrom, Jakob Grue Simonsen
Proc. 21th RTA, LIPIcs 284, pp. 17 – 32, 2010

CoCo 2015 participant: CSI-ho 0.1
Julian Nagele
Proceedings of IWC 2015, p. 41

overview

- definability
- confluence
- simply typed lambda calculus
- strategies

simple types: definition

we assume a base type o

- a base type is a simple type
- if A and B are simple types then $A \to B$ is a simple type

$\to$ is assumed to be right-associative

outermost parentheses are omitted

every simple type is of the form $A_1 \to \ldots \to A_n \to \mathsf{o}$

simply typed lambda-terms: defintion

we assume a priori typed variables $x, y, z, \ldots$

- $x : A$ is a simply typed λ-term of type A
- $\lambda x.\, M : B \to C$ is a simply typed λ-term of type $B \to C$

  if $M : C$ and $x : B$
- $M\, N : A$ is a simply typed λ-term of type A

  if $M : B \to A$ and $N : B$

examples

$\lambda x.\, x : A \to A$ if $x : A$

$\lambda x.\, \lambda y. x : A \to B \to A$ if $x : A$ and $y : B$

$\lambda s.\, \lambda z. s\, z : (A \to A) \to A \to A$ if $s : A \to A$ and $z : A$

$\Omega = (\lambda x.\, x\, x)\,(\lambda x.\, x\, x)$ is not typable

$\mathsf{Y} = \lambda f.\,(\lambda x.\, f\,(x\, x))\,(\lambda x.\, f\,(x\, x))$ is not typable

$\mathsf{T} = (\lambda x.\, \lambda y.\, y\,(x\, x\, y))\,(\lambda x.\, \lambda y.\, y\,(x\, x\, y))$ is not typable

termination of beta-reduction

unlike untyped λ-calculus,

simply typed λ-calculus is terminating

termination follows using the computability proof method

proof method is crucial for HO-termination proofs

method due to Tait and Girard

computability: definition

Definition

- M : o is computable
  if M is terminating
- $M : A \to B$ is computable
  if $M\,N : B$ is computable for every computable $N : A$

Example

- $x : \mathrm{o}$ is computable
- $(\lambda x.\,x)\,y : \mathrm{o}$ is computable
- $x : A \to B$ computable?

first attempt of the termination proof

- step 1 A: computability implies termination
  (for type o this holds by definition)
  try induction on the definition of computability
  for $M : A \to B$ we need to take a computable term of type A
  do we have such a term?

- step 1B: show that variables are computable
  try induction on the definition of computability
  for $x : A \to B$ we need to show that $x\,P$ is computable

- prove 1A and 1B simultaneously

- step 2: all simply typed terms are computable

Lemma (gives steps 1A and 1B)

1. if $M : C$ computable then $M : C$ terminating
2. for all $n \geq 0$:
   if $x\,P_1 \ldots P_n : C$ terminating then $x\,P_1 \ldots P_n : C$ computable

simultaneous induction on the definition of types.

- for type o: both follow by definition of computabiity
- for type $A \to B$:
  1: take $M : A \to B$ computable
  take $x : A$; it is computable by IH2
  by definition, $M\,x : B$ is computable, and hence by IH1 terminating
  hence $M : A \to B$ is terminating

  2: take $x\,P_1 \;\ldots\; P_n : A \to B$ terminating
  take $Q : A$ computable (exists by IH2); it is terminating by IH1
  hence $x\,P_1\,P_n\,Q : B$ is terminating; it is computable by IH2
  so $x\,P_1 \ldots P_n : A \to B$ computable

■

all terms are computable: proof attempt

induction on the definition of terms

1. $M = x$
2. $M = P\,Q$
3. $M = \lambda x.\, P$ here it does not work immediately
   we need to show: $(\lambda x.\, P)\, Q$ is computable for computable Q
   we will use:
   $P[x := Q]$ computable implies $(\lambda x.\, P)\, Q$ computable
   and strengthening of the current statement

all terms are computable (gives step 2)

Theorem

σ computable $\Rightarrow$ M^σ computable

Proof.

Induction on the definition of terms.

1. $M = x$

   variables are computable and σ is computable

∎

all terms are computable (gives step 2)

Theorem

σ computable $\Rightarrow$ M^σ computable

Proof.

Induction on the definition of terms.

1. $M = x$
2. $M = \lambda x.\, P$
   $\sigma[x := N]$ is a computable substitution for N computable
   by IH $P^{\sigma[x:=N]} = P^\sigma[x := N]$ is computable
   by (exercise) lemma below, $(\lambda x.\, P^\sigma)\, N$ is computable

■

Lemma

for all $n \geq 0$: if $M[x := N]\, P_1 \ldots P_n$ computable and N computable

then $(\lambda x.\, M)\, N\, P_1 \ldots\, P_n$ computable

all terms are computable (gives step 2)

Theorem

σ computable $\Rightarrow M^\sigma$ computable

Proof.

Induction on the definition of terms.

1. $M = x$
2. $M = \lambda x.\, P$
3. $M = P\, Q$
   by IH P^σ and Q^σ are computable
   by definition of computability, $P^\sigma\, Q^\sigma$ is computable

∎

finally, the result

we have: computability implies termination

we have: M^σ computable for every M and for every computable σ

we have: identity substitution is computable

Corollary

simply typed λ-calculus with β-reduction is terminating

Curry-Howard-De Bruijn isomorphism

the formulas and proofs of first-order propositional logic

correspond to

the types and terms of simply typed λ-calculus

$$\dfrac{\dfrac{[A^x]}{B \to A}\, I[y]\!\to}{A \to B \to A}\, I[x]\!\to \quad : \quad A \to B \to A$$

$$\sim$$

$$\lambda x : A.\, \lambda y : B.\, x \quad : \quad A \to B \to A$$

overview

- definability
- confluence
- simply typed lambda calculus
- strategies

strategy: informally

there may be different ways to reduce a term

a strategy tells us how to reduce a term

a term may be weakly normalizing (WN) but not terminating (SN)

a normalizing strategy yields a reduction to normal form if possible

a perpetual strategy yields an infinite reduction if possible

in general: a strategy gives us a reduction with a desired property

reduction graph of a λ-term

terms are the vertices and the reduction steps are the edges

a reduction graph may be finite and cycle-free; example: $\mathsf{I}\,x$

a reduction graph may be finite with cycles; example: Ω

a reduction graph may be infinite; example: $(\lambda x.\, x\,x\,x)\,(\lambda x.\, x\,x\,x)$

a reduction graph is not necessarily simple; example: $\mathsf{I}\,(\mathsf{I}\,\mathsf{I})$

a reduction graph may be nice to draw; example: $(\lambda x.\, \mathsf{I}\,x\,x)\,(\lambda x.\, \mathsf{I}\,x\,x)$

the leftmost-innermost reduction strategy

is not normalizing:

$(\lambda x.\, y)\, \Omega \rightarrow_\beta (\lambda x.\, y)\, \Omega \rightarrow_\beta (\lambda x.\, y)\, \Omega \rightarrow_\beta \ldots$

does not copy redexes (example):

$(\lambda x.\, f\, x\, x)\, (((\lambda x.\, x)\, a)) \rightarrow_\beta (\lambda x.\, f\, x\, x)\, a \rightarrow_\beta f\, a\, a$

may contract redexes that are not needed:

$(\lambda x.\, y)\, (\mathsf{I}\, z) \rightarrow_\beta (\lambda x.\, y)\, z \rightarrow_\beta y$

innermost reduction

for first-order orthogonal TRSs, any innermost strategy is perpetual

for λ-calculus this is not true:

the term $(\lambda x.\,(\lambda y.\,z)\,(x\,x))\,(\lambda x.\,x\,x)$ is WIN:

$(\lambda x.\,(\lambda y.\,z)\,(x\,x))\,(\lambda x.\,x\,x) \to_\beta (\lambda x.\,z)\,(\lambda x.\,x\,x) \to_\beta z$

but not SN:

$(\lambda x.\,(\lambda y.\,z)\,(x\,x))\,(\lambda x.\,x\,x) \to_\beta (\lambda y.\,z)\,\Omega \to_\beta (\lambda y.\,z)\,\Omega \to_\beta \ldots$

so innermost reduction is not perpetual for λ-calculus

we do not have: strongly innermost normalizing implies strongly normalizing

the leftmost-outermost strategy

is normalizing for left-normal TRSs

λ-calculus is left-normal (but not a TRS)

lefmost-outermost strategy is normalizing

first proof by Curry 1958,

recent proofa by Hirokawa, Middeldorp, and Moser,

and by Toyama and Van Oostrom

example: $(\lambda x.\, y)\, \Omega \to_{\beta} y$

the rightmost-outermost strategy

is not normalizing:

$((\lambda x.\,\lambda y.\,x)\,\mathsf{I})\,\Omega \to ((\lambda x.\,\lambda y.\,x)\,\mathsf{I})\,\Omega \to \ldots$

λ-calculus is not right-normal

further reading

Leftmost outermost revisited
Nao Hirokawa, Aart Middeldorp and Georg Moser
LIPIcs 36 (2015)

Normalisation by Random Descent
Vincent van Oostrom and Yoshihito Toyama
LIPIcs 52 (2016)

conclusion

for λ-calculus and for higher-order rewriting

we often need multi-step reduction instead of parallel reduction

for λ-calculus and for higher-order rewriting

we often need a variation of the computability proof method

more importantly

many thanks to the organizers

Adeline, Claire, Frédéric, Olivier !!