

Rewriting and termination in lambda calculus

Silvia Ghilezan

University of Novi Sad
Mathematical Institute SASA
Serbia

Lecture 1

Oregon Programming Language Summer School
Eugene, June 2022

About me...

PhD University of Novi Sad

advised by Henk Barendregt and Kosta Došen

Postdoc University of Nijmegen

Research and teaching

University of Turin

McGill University

École Normale Supérieure Lyon



Roadmap

- ▶ Rewriting basics
- ▶ Lambda calculus
- ▶ Confluence: Church-Rosser property, local confluence
- ▶ Normalisation: strong normalisation, normalisation
- ▶ Strategies: leftmost-outermost strategy, perpetual strategies
- ▶ Simple types in lambda calculus: strong normalization
- ▶ Intersection types in lambda calculus: complete characterisations of normalisations

Rewriting basics

Higher-order rewriting systems - Lambda calculus

Rewriting

Methods of **replacing**

- ▶ subexpressions of a formula (object)
- ▶ with other expressions

Examples: algebra, arithmetics, logic, geometry, linguistics, physics

- ▶ Term rewriting
- ▶ Higher-order term rewriting
- ▶ Graph rewriting
- ▶ String rewriting, semi-Thue systems
- ▶ Trace rewriting, concurrent computation and process calculi
- ▶ ...

Algebra - commutative group theory

$(a \star b) \star c \rightarrow a \star (b \star c)$	associativity
$a \star e \rightarrow a$	neutral element
$a \star a^{-1} \rightarrow e$	inverse element
$a \star b \rightarrow b \star a$	commutativity

Logic - propositional logic

$A \wedge B \rightarrow A$	$A \wedge B \rightarrow B$	conjunction
$A \rightarrow A \vee B$	$B \rightarrow A \vee B$	disjunction
$A \rightarrow \neg\neg A$	$\neg\neg A \rightarrow A$	negation

Abstract rewrite

$$\begin{aligned}f(x, x) &\rightarrow g(x) \\f(x, g(x)) &\rightarrow b \\h(a, x) &\rightarrow f(h(x, a), h(x, x))\end{aligned}$$

Abstract rewrite systems - ARS

$(A, \longrightarrow)$

- ▶ A is a set
- ▶ $\longrightarrow$ is a binary relation on A
 - ▶ $t \longrightarrow s$ is a step
- ▶ $\longrightarrow$ is a **transitive-reflexive** closure
 - ▶ $t \longrightarrow^* s$ is a sequence $t \longrightarrow s_1 \longrightarrow s_2 \dots \longrightarrow s_n \equiv s$
- ▶ $\longrightarrow_1 \cdot \longrightarrow_2$ is a **composition** $t \longrightarrow_1 u \longrightarrow_2 s$
- ▶ $\longrightarrow$ is **deterministic**
 - ▶ if for each $t \in A$ there is at most one $s \in A$ such that $t \longrightarrow s$

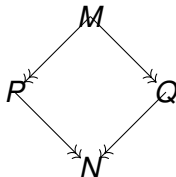
ARS main properties

We focus on:

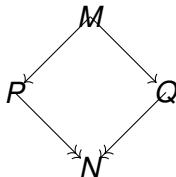
- ▶ confluence
- ▶ normalisation
- ▶ termination

Confluence

$(A, \longrightarrow)$ is **confluent** if $\longleftarrow \cdot \longrightarrow \subseteq \longrightarrow \cdot \longleftarrow$



$(A, \longrightarrow)$ is **locally confluent** if $\longleftarrow \cdot \longrightarrow \subseteq \longrightarrow \cdot \longleftarrow$



Normalisation and termination

- ▶ $n \in A$ is a **normal form** if there is no s such that $n \longrightarrow s$
- ▶ $t \in A$ **has a normal form** if $t \longrightarrow^* n$ and n is a normal form, we say that n is a NF of t
- ▶ $(A, \longrightarrow)$ is weakly normalising (**normalising**) if every $t \in A$ has a normal form
 - ▶ at least one (rewrite) sequence from t is finite
 - ▶ **How to find?**
- ▶ $(A, \longrightarrow)$ is **strongly normalising** (terminating) if for every $t \in A$
 - ▶ all (rewrite) sequences from t are finite
 - ▶ **How to prove?**
- ▶ $(A, \longrightarrow)$ is **not terminating** (terminating) if for some $t \in A$
 - ▶ at least one (rewrite) sequence from t is infinite
 - ▶ **How to find?**

ARS more properties

Theorem

Confluence implies uniqueness of normal form

Proof.

Suppose t has two normal forms n_1 and n_2 . Then $n_1 \leftarrow t \rightarrow n_2$. Then by confluence there is an s such that $n_1 \rightarrow s \leftarrow n_2$, which is not possible since n_1 and n_2 are normal forms.



Theorem (Newman's Lemma)

A terminating ARS is confluent if and only if it is locally confluent

Proof.

Based on well-founded induction



Example

Rewrite system

$$f(x, x) \rightarrow g(x)$$

$$f(x, g(x)) \rightarrow b$$

$$h(a, x) \rightarrow f(h(x, a), h(x, x))$$

- ▶ Normalizing: $h(a, a)$ has two NF b and $g(b)$
- ▶ Not terminating: $h(a, a)$ has an infinite derivation
- ▶ Not confluent: $h(a, a)$ has two NF, then it is not confluent

Reduction strategies

Reduction strategy: A restriction to a subreduction

$$\longrightarrow_e \subseteq \longrightarrow$$

is a way to control that in a term there are different possible choices of reduction

Example. CBN, CBV, perpetual, leftmost (*later*)

Normalisation strategy: A reduction strategy which reaches the normal form if it exists

Example. Leftmost (*later*)

Term rewriting

A **term rewriting system** (TRS) is a rewriting system (ARS) whose objects are terms, which are expressions with nested sub-expressions.

Example Logic - propositional, above, is a term rewriting system

- ▶ the terms are composed of binary operators ($\vee$) and ($\wedge$) and the unary operator ($\neg$)
- ▶ the rules contain variables, which represent any possible term (though a single variable always represents the same term throughout a single rule)

Higher-order term rewriting systems (HOR) are a generalization of first-order term rewriting systems to systems allowing

- ▶ higher order functions
- ▶ bound variables

Higher-order rewriting systems - models of computation 1930s

- ▶ Alonzo Church: lambda calculus
 - ▶ theory of functions - formalisation of mathematics
 - ▶ successful model for computable functions
- ▶ Moses Schonfinkel (1921): combinators
- ▶ Haskell Curry: combinatory logic

References



 F. Baader, T. Nipkow

Term rewriting and all that
Cambridge University Press, 1999



 Term Rewriting Systems - Terese

eds M. Bezem, J. W. Klop, R. de Vrijer
Cambridge University Press, 2003

 F. van Raamsdonk

Higher-Order Rewriting
RTA 1999: 220-239

 Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, P. Wadler

The Call-by-Need Lambda Calculus
POPL 1995: 233-246

Rewriting basics

Higher-order rewriting systems - Lambda calculus

Lambda calculus: background

In mathematics,

1. expressions with free variables
2. functions

Example.

$$1. x^2 + 1 < 10 \quad x^2 < 9 \quad x \in (-3, 3) \quad \text{free}$$

$$2. f(x) = x^2 + 1 \quad f(5) = 5^2 + 1 \quad f(5) = 26 \quad \text{bound}$$

Church: $\lambda x.(x^2 + 1)$ denotes a function of x

HOR feature - bound variables

Function behaviours treated formally in lambda calculus

- ▶ The name of the argument is not important

$$f(x) = x^2 + 1 \text{ and } f(y) = y^2 + 1$$

$\lambda x.(x^2 + 1)$ and $\lambda y.(y^2 + 1)$ must be considered as equal

α -conversion

- ▶ Function application evaluation

$(\lambda x.(x^2 + 1))(5)$ and $5^2 + 1$ must be considered as equal

β -conversion

- ▶ Substitution and bindings

$(\lambda x.(\lambda y.(x^2 + 1 + y)))(y)$ should be $\lambda w.(y^2 + 1 + w)$
but not $\lambda y.(y^2 + 1 + y)$

Barendregt variable convention: no variable is both free and bound

HOR feature - functions of multiple arguments

In lambda calculus, functions of multiple arguments can be treated as iterated applications of single argument functions

$$f(x, y, z) = x + y + z$$

$$f(1, 2, 3) = 6$$

f becomes $(\lambda x.(\lambda y.(\lambda z.x + y + z)))$

$(\lambda x.(\lambda y.(\lambda z.x + y + z)))(1)(2)(3)$ evaluates to

$(\lambda y.(\lambda z.1 + y + z))(2)(3)$ evaluates to

$(\lambda z.1 + 2 + z))(3)$ evaluates to

6

currying

Syntax

$V = \{x, y, z, x_1, \dots\}$ a countable set of variables

$C = \{a, b, c, a_1, \dots\}$ a countable set of constants

Definition

The set Λ of (lambda) terms is inductively defined

- ▶ $V \subseteq \Lambda$ and $C \subseteq \Lambda$ **atomic terms**
- ▶ $M, N \in \Lambda$ then $(MN) \in \Lambda$ **application**
- ▶ $M \in \Lambda, x \in V$ then $(\lambda x.M) \in \Lambda$ **abstraction**

Conventions for minimizing the number of the parentheses:

- ▶ $M_1 M_2 M_3$ stands for $((M_1 M_2) M_3)$ application associates to left
- ▶ $\lambda x.y.M$ stands for $(\lambda x.(\lambda y.(M)))$ abstraction associates to right
- ▶ $\lambda x.M_1 M_2 \equiv \lambda x.(M_1 M_2)$; application has priority over abstraction

Pure lambda calculus, if $C = \emptyset$

$$M ::= x \mid MM \mid \lambda x.M$$

Running example

$xyzx$

$\lambda x.zx$

I = $\lambda x.x$

combinator **I**

K = $\lambda xy.x$

combinator **K**

S = $\lambda xyz.xz(yz)$

combinator **S**

Δ = $\lambda x.xx$

selfapplication

Y = $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$

fixed point combinator

Ω = $\Delta\Delta = (\lambda x.xx)(\lambda x.xx)$

higher-order function

Free and bound variables

Definition

(i) The set $FV(M)$ of **free** variables of M is defined inductively:

- ▶ $FV(x) = \{x\}$
- ▶ $FV(MN) = FV(M) \cup FV(N)$
- ▶ $FV(\lambda x.M) = FV(M) \setminus \{x\}$

(ii) A variable in M is **bound** if it is not free

- ▶ x is bound in M if it appears in a subterm of the form $\lambda x.N$

(ii) M is a **closed** λ -term (or *combinator*) if $FV(M) = \emptyset$

Λ^0 denotes the set of closed λ -terms.

Example

- ▶ In $\lambda x.zx$, variable z is free.
- ▶ Term $\lambda xy.xxy$ is closed.

Running example: 1. free variables

M	$Fv(M)$
$xyzx$	
$\lambda x.zx$	
$\mathbf{I} = \lambda x.x$	
$\mathbf{K} = \lambda xy.x$	
$\mathbf{S} = \lambda xyz.xz(yz)$	
$\Delta = \lambda x.xx$	
$\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	
$\Omega = \Delta\Delta = (\lambda x.xx)(\lambda x.xx)$	

Running example: 1. free variables

M	$Fv(M)$
$xyzx$	$\{x, y, z\}$
$\lambda x.zx$	
$\mathbf{I} = \lambda x.x$	
$\mathbf{K} = \lambda xy.x$	
$\mathbf{S} = \lambda xyz.xz(yz)$	
$\Delta = \lambda x.xx$	
$\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	
$\Omega = \Delta\Delta = (\lambda x.xx)(\lambda x.xx)$	

Running example: 1. free variables

M	$Fv(M)$
$xyzx$	$\{x, y, z\}$
$\lambda x.zx$	$\{z\}$
$\mathbf{I} = \lambda x.x$	
$\mathbf{K} = \lambda xy.x$	
$\mathbf{S} = \lambda xyz.xz(yz)$	
$\Delta = \lambda x.xx$	
$\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	
$\Omega = \Delta\Delta = (\lambda x.xx)(\lambda x.xx)$	

Running example: 1. free variables

M	$Fv(M)$
$xyzx$	$\{x, y, z\}$
$\lambda x.zx$	$\{z\}$
$\mathbf{I} = \lambda x.x$	$\emptyset$
$\mathbf{K} = \lambda xy.x$	
$\mathbf{S} = \lambda xyz.xz(yz)$	
$\Delta = \lambda x.xx$	
$\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	
$\Omega = \Delta\Delta = (\lambda x.xx)(\lambda x.xx)$	

Running example: 1. free variables

M	$Fv(M)$
$xyzx$	$\{x, y, z\}$
$\lambda x.zx$	$\{z\}$
$\mathbf{I} = \lambda x.x$	$\emptyset$
$\mathbf{K} = \lambda xy.x$	$\emptyset$
$\mathbf{S} = \lambda xyz.xz(yz)$	$\emptyset$
$\Delta = \lambda x.xx$	$\emptyset$
$\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	$\emptyset$
$\Omega = \Delta\Delta = (\lambda x.xx)(\lambda x.xx)$	$\emptyset$

Substitution

Implicit substitution, meta notion (not in the language)

Definition

$N[x := M]$ is defined by induction on the structure of N

$$\begin{array}{lll} N = x & x[x := M] & = M \\ N = y & y[x := M] & = y \\ N = N_1 N_2 & (N_1 N_2)[x := M] & = N_1[x := M] N_2[x := M] \\ N = \lambda x. N & (\lambda x. N)[x := M] & = \lambda x. N \\ N = \lambda y. N & (\lambda y. N)[x := M] & = \lambda y. (N[x := M]), \quad y \notin FV(M) \\ N = \lambda y. N & (\lambda y. N)[x := M] & = (\lambda z. N[y := z])[x := M], \\ & & y \in FV(M) \wedge z \notin FV(M) \end{array}$$

Explicit substitution, calculi where substitution is in the language

Rewrite rules

α -reduction:

$$\lambda x.M \longrightarrow_{\alpha} \lambda y.M[x := y], \quad y \notin FV(M)$$

β -reduction:

$$(\lambda x.M)N \longrightarrow_{\beta} M[x := N]$$

Rewrite rules

α -reduction:

$$\lambda x.M \longrightarrow_{\alpha} \lambda y.M[x := y], \quad y \notin FV(M)$$

β -reduction:

$$(\lambda x.M)N \longrightarrow_{\beta} M[x := N]$$

η -reduction:

$$\lambda x.(Mx) \longrightarrow_{\eta} M, \quad x \notin FV(M)$$

Rewrite rules

α -reduction:

$$\lambda x.M \longrightarrow_{\alpha} \lambda y.M[x := y], \quad y \notin FV(M)$$

β -reduction:

$$(\lambda x.M)N \longrightarrow_{\beta} M[x := N]$$

Properties of rewrite systems: confluence, normal forms, normalisation, strong normalisation, strategies

