Reducibility method: an overview

Silvia Ghilezan
University of Novi Sad, Serbia

LAP 2013, Dubrovnik, September 16-21, 2013

In collaboration with (alphabetic order):

, , ,

- Henk Barendregt,
- Mariangiola Dezani-Ciancaglini
- Daniel Dougherty
- Jelena Ivetić
- Pierre Lescanne
- Silvia Likavec
- Viktor Kunčak

Working group at University of Novi Sad

University of Novi Sad, Faculty of Technical Sciences, Serbia
Computation, Concurrency, Logic and Reasoning - Co^2LoR

- Silvia Ghilezan, Professor,
- Jovanka Pantović, Professor
- Jelena Ivetić, Teaching Assistant, PhD candidate
- Svetlana Jaksić, Teaching Assistant, PhD student
- Jelena Čolić, Teaching Assistant, PhD student
- Jovana Obradović, Teaching Assistant, PhD student
- Jovana Radenović, Teaching Assistant, PhD student
- Andrijana Stamenković, PhD student

Group members abroad (Italy, Switzerland, France, USA)

- Silvia Likavec, U. Torino, (PhD, U. Torino, ENS-Lyon)
- Viktor Kunčak, EPFL Lausanne, (PhD MIT)
- Dragiša Zunić, FIMEK, (PhD, ENS-Lyon)

Key notions

λ-calculus vs Intuitionistic logic

- Curry–Howard correspondence
- computational content of logic

Reducibility method

- simple and intersection types
- polymorphic types

Logical relations

- λ-calculus
- programming languages

1. λ-calculus vs Intuitionistic logic
   - Intuitionistic logic
   - λ-calculus
2. Types in λ-calculus
   - Simple types
   - Curry–Howard paradigm
   - Intersection types
3. Reducibility
   - Simple and Intersection types
   - Polymorphic types
   - Classical logic
4. Logical relations
   - Theory
   - Application

Logical systems: Axiomatic, Natural Deduction, Sequent

Three most well-known kinds of proof calculi are:

- Axiomatic (Hilbert) system:
  - axioms (e.g. $A \to A$)
  - Modus Ponens

$$\frac{A \to B \qquad A}{B}$$

- Natural Deduction:
  - axioms
  - elimination rules (MP)
  - introduction rules

- Sequent Calculus:
  - axioms
  - left introduction rules correspond to elimination rules
  - right introduction rules correspond to introduction rules
  - cut rule

Paradise of logical systems

From theory

- classical logic: propositional and predicate
- intuitionistic logic: propositional and predicate
- intermediate logic
- substructural logics: relevance, affine, linear logic
- modal logics: possibility and necessity
- probabilistic logics
- temporal logic:
- fuzzy logic
- paraconsistent logic

to practice and application

- programming languages
- model checking
- interactive proof assistance, theorem proving
- hardware and software verification

Axiomatic (Hilbert style) system

Intuitionistic Logic

Brouwer, Heyting, Kolomgorov - algorithmic interpretation of connectives

- Axioms (implicational fragment)
  $(Ax1)\ A \to A$
  $(Ax2)\ A \to (B \to A)$
  $(Ax3)\ (A \to (B \to C)) \to ((A \to B) \to (A \to C))$
- Rule

$$(MP) \qquad \frac{A \to B \quad A}{B}$$

Not provable

- $A \vee \neg A$, law of excluded middle (tertium non datur)
- $((A \to B) \to A) \to A$, Peirce's law
- $\neg\neg A \to A$, double negation elimination

Natural Deduction
Intuitionistic Logic

Gentzen, Prawitz 1960s

- Axiom

$$(Ax) \qquad \frac{}{\Gamma, A \vdash A}$$

- Rules

$$(\rightarrow_{elim}) \qquad \frac{\Gamma \vdash A \rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

$$(\rightarrow_{intr}) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}$$

The syntax

λ-terms are expressions divided into three categories:

1. variables: $x, y, z, z_1, ...$
   - free or bound
2. application: MN
   - function application, "apply M to N"
3. abstraction: $\lambda x.M$
   - function generation by binding a variable thus creating the parameter of the function, like integrals, universal quantification.

Definition

Λ is built up from an infinite set of variables $V = \{x, y, z, x_1...\}$:

- $x \in V \Rightarrow x \in \Lambda$
- $M, N \in \Lambda \Rightarrow (MN) \in \Lambda$
- $M \in \Lambda, x \in V \Rightarrow (\lambda x.M) \in \Lambda$

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

Reduction rules

- Reduction rules are used for term simplifying (or *executing*):

$$(\lambda xy.\mathbf{add}xy)\underline{3}\,\underline{4} \longrightarrow \underline{7}$$

- There are three reductions (*redex* and *contractum*):
  1. α-reduction (renaming)

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

  2. β-reduction (computation)

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

     provided that $M, N \in \Lambda$ and $M[x := N]$ is valid substitution.
  3. η-reduction (extensionality)

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

     This rule identifies two functions that always produce equal results if taking equal arguments.
- Generalized reductions enable application of rules on the sub-terms.

Important properties - normalisation

- Normal form, NF, is a completely evaluated λ-term
  - $\lambda x.x \equiv I$, $\lambda xy.x \equiv K$, $\lambda xyz.xz(yz) \equiv S$
  - $\lambda x.xx \equiv \Delta$
- Terms without NF:
  - $\Omega \equiv (\lambda x.xx)(\lambda x.xx) \rightarrow \Omega \rightarrow \ldots$
  - $Y \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$
- A term is strongly normalising, SN, if all reduction sequences starting from it are **finite**
- The characterisation of the SN set is an important problem for programming languages (the way to avoid Halting problem), and its solution has been found in the typed λ-calculus.

Important properties - expressiveness

Theorem

(Kleene, 1936.) *All computable (recursive) functions can be represented by λ-calculus.*

- Representation of significant number of objects and data structures
- inefficiency in the representation of all objects and data structures.

Simplicity, expressive power, universality!

Types

- Disadvantages of the untyped λ-calculus:
  - there exist λ-terms without a normal form;
  - it is allowed to create meaningless applications, like **sin log**
- Types are syntactical objects that can be assigned to λ-terms.
- Intuitively, a type denotes the domain of a function
- Two typing approaches:
  - *à* la Curry - implicit type assignment (*lambda calculus with type assignment*)
  - *à* la Church - explicit type assignment (*typed lambda calculus*)

Simple types

$$\mathbf{T}: \quad A \quad ::= \quad p \mid A \to A \mid A \cap A$$

Type assignment $\Lambda : \mathbf{T}$

- $t : A$ is a **type assignment**, where $t \in \Lambda$ and $A \in \mathbf{T}$.
- $x : A$ is a **basic type assignment** (declaration)
- $\Gamma = \{x_1 : A_1, , x_n : A_n\}$, a **basis** is a set of basic type assignments with different term variables.
- $\Gamma \vdash t : A$

λ-calculus with simple types

$$(axiom) \quad \frac{}{\Gamma, x : A \vdash x : A}$$

$$(\rightarrow_{elim})\ (app) \quad \frac{\Gamma \vdash t : A \rightarrow B \qquad \Gamma \vdash s : A}{\Gamma \vdash ts : B}$$

$$(\rightarrow_{intr})\ (abs) \quad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B}$$

Theorem

If M is typeable in $\lambda \rightarrow$, then $M \in SN$.

$\lambda x.x : A \rightarrow A$

$\lambda xy.x : A \rightarrow B \rightarrow A$

Not typeable NF: $\lambda x.xx$!!!

Natural Deduction NJ - λ-calculus

$$(axiom) \quad \frac{}{\Gamma, \quad A \vdash \quad A}$$

$$(\rightarrow_{elim}) \quad \frac{\Gamma \vdash \quad A \rightarrow B \qquad \Gamma \vdash \quad A}{\Gamma \vdash \quad B}$$

$$(\rightarrow_{intr}) \quad \frac{\Gamma, \quad A \vdash \quad B}{\Gamma \vdash \qquad A \rightarrow B}$$

Natural Deduction NJ - λ-calculus

$$(axiom) \quad \frac{}{\Gamma, x : A \vdash x : A}$$

$$(\rightarrow_{elim})\ (app) \quad \frac{\Gamma \vdash t : A \rightarrow B \qquad \Gamma \vdash s : A}{\Gamma \vdash ts : B}$$

$$(\rightarrow_{intr})\ (abs) \quad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B}$$

$$\vdash A \Leftrightarrow \vdash t : A$$

Core of computation

- computation over terms
- normalisation over proofs

Computational interpretations of intuitionistic logic

- 1950s Curry
- 1968 (1980) Howard formulae-as-types
- 1970s Lambek - CCC Cartesian Closed Categories
- 1970s de Bruijn AUTOMATH

logic vs term calculus

$$\vdash A \quad \Leftrightarrow \quad \vdash t : A$$

Curry - Howard - Lambek - de Bruijn correspondence

Curry-Howard paradigm:

formulae – as – types

proofs – as – terms

proofs – as – programs

λ-calculus with intersection types

Coppo and Dezani, Pottinger, Sallé in the 1970s

$$(\cap_{intr}) \quad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash t : B}{\Gamma \vdash t : A \cap B}$$

$$(\cap_{elim}) \quad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : A}$$

Theorem

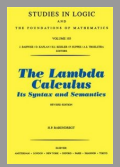
M is typeable in $\lambda\cap$ iff $M \in SN$.

$\lambda x.xx : (A \to B) \cap A \to B$

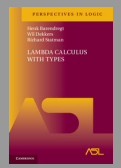
More types

- Barendregt's cube consists of 8 calculi:
  - $\lambda\rightarrow$, $\lambda 2$, $\lambda\omega$, $\lambda\underline{\omega}$,
  - λP, $\lambda P2$, $\lambda P\omega$, $\lambda P\underline{\omega}$.
- recursive types
- intersection types
- variants of λ-calculus: λLJ, $\overline{\lambda}$, λ^{Gtz}, $\lambda_{\circledR}$...

References:



H.P. Barendregt.
Lambda Calculus: Its syntax and Semantics.
North Holland 1984.



H.P. Barendregt, W. Dekkers, R. Statman.
Lambda Calculus with Types.
contributors F. Alessi, H, Barendregt, M. Bezem, F. Cardone, M. Coppo, W. Dekkers, M. Dezani-Ciancaglini, G Dowek, S. Ghilezan, F. Honsell, M. Moortgat, P. Severi, R. Statman, P. Urzyczyn.
Cambridge University Press 2013.

Reducibility method - History

Reducibility: a technique for proving SN of systems with simple types, intersection types...

Properties:

- Terms typeable in simply typed λ-calculus ($\lambda \rightarrow$) are SN

  1967 Tait

- Terms typeable in polymorphic λ-calculus (system F or $\lambda 2$) are SN

  1971 J.-Y. Girard - candidats de réductibilité

  1971 J. Reynolds - Forsythe

- Terms typeable in λ-calculus with intersection types ($\lambda\cap$ or system D) are SN

  1990 Krivine

  1990s S. van Bakel, S.G., Amadio and Curien, among others

Reducibility method for simple and intersection types

Theorem

Terms typeable in $\lambda\to$ and $\lambda\cap$ are SN

Reducibility based on:

type interpretation

$$[[p]] = SN \subset \Lambda,$$

$$[[A \to B]] = [[A]] \longrightarrow [[B]] = \{t \mid s \in [[A]] \text{ implies } ts \in [[B]]\}$$

$$[[A \cap B]] = [[A]] \cap [[B]]$$

term valuationS

$$\rho(x) \in \Lambda,$$

$$[[t]]_\rho = t[\rho(x_1)/x_1, \ldots, \rho(x_n)/x_n]$$

- $[[A]] \subseteq SN$
- (Soundness) $\vdash t : A$ then $[[t]]_\rho \in [[A]]$
- (SN) By taking $\rho(x) = x$, we get $\vdash t : A$ implies $t \in [[A]] \subseteq SN$

Reducibility method for simple and intersection types

Suitable modifications of the reducibility method lead to uniform proofs of other fundamental reduction properties

- Normalisation, head normalisation, weak normalisation,...
- Standardisation theorem
- Confluence (Church–Rosser)

1985 R. Statman

1990s G. Koletsos, Y. Stavrinos

2000s S.G., S. Likavec, V.Kunčak

2000s M. Dezani, S.G., S. Likavec

2010s F. Kamareddine, V. Rahli, J. B. Wells

Reducibility method for polymorphic types

Theorem

Terms typeable in polymorphic λ calculus ($\lambda 2$, system F) are SN

Reducibility based on
type interpretationS

$$[\![p]\!]_\xi = \Xi \in \mathsf{SAT} \ \ (\Xi \subseteq SN),$$

$$[\![A \to B]\!]_\xi = [\![A]\!]_\xi \longrightarrow [\![B]\!]_\xi == \{t | s \in [\![A]\!]_\xi \ \text{ implies } \ ts \in [\![B]\!]_\xi\}$$

term valuationS

$$\rho(x) \in \Lambda,$$

$$[\![t]\!]_\rho = t[\rho(x_1)/x_1, \ldots, \rho(x_n)/x_n]$$

- (Soundness) $\vdash t : A$ then $[\![t]\!]_\rho \in [\![A]\!]_\xi$
- (SN) By taking $\rho(x) = x$ and $\xi(p) = SN$ we get $\vdash t : A$ implies $t \in SN$

Reducibility method for polymorphic types

Importance

- the statement can be formulated in first order Peano arithmetic
- the proof cannot be formulated in first order Peano arithmetic

Meaningful Gödel sentence, 1970s J.-Y. Girard, J. Reynolds

Importance in computer science

- Tait's method is a powerful proof technique frequently used for showing foundational properties of languages based on typed λ-calculi
- these proofs have been extremely difficult to formalize in proof assistants with weak meta-logics

Reducibility method and classical logic

The reducibility method is not well suited to prove strong normalization for $\lambda\mu$-calculus, the simply typed classical term calculus
R. David, K. Nour (2005)

- The algorithmic interpretation of classical logic changes the focus of computation
- The "symmetric candidates" technique used to prove strong normalisation employs a fixed-point technique

Logical relations - theory

Reducibility relates terms typeable in a certain type system (simple, intersection, polymorphic...) and terms satisfying certain reduction properties (strong normalisation, head normalisation ...).

- It can be represented by a unary predicate $P_\alpha(t)$, which means that a term t typeable by α satisfies the property P.
- Types are interpreted as suitable sets of terms called saturated or stable sets.
- Soundness of the type assignment is obtained with respect to these interpretations.
- A consequence of soundness: every term typeable in the type system belongs to the interpretation of its type.
- This is an intermediate step between the terms typeable in the given type system and terms satisfying the considered property P.

Logical relations - theory

Logical relations

- Method based on binary relations $R_\alpha(t, t')$, which relate terms t and t' typeable by the type α that satisfy the relation R.
- Types are then interpreted as admissible relations.

1985 R. Statman
1985 G. Koletsos
confluence (the Church-Rosser property) of $\beta\eta$-reduction of the simply typed λ-calculus.

Logical relations - application

Powerful tool in programming languages.
1996 J. Mitchell
2002 B. Pierce
Important applications:

- Observational equivalence: logical relations prove that terms obtained by optimisation are equivalent.
- Compiler correctness: logical relations are employed to relate the source and target language.
- Security information flow: logical relations prove that the system prevents high security data to leak in low security output.

Publications


S. Ghilezan, J. Ivetić, P. Lescanne, and S. Likavec.
Intersection types for the resource control lambda calculi.
ICTAC 2011, LNCS 6916: 116-134 (2011)


D. Dougherty, S. Ghilezan and P. Lescanne.
Characterizing strong normalization in the Curien-Herbelin symmetric lambda calculus: extending the Coppo-Dezani heritage.
Theoretical Computer Science 398: 114-128 (2008)


J. Espírito Santo, S. Ghilezan, J. Ivetić.
Characterizing strongly normalising intuitionistic sequent terms.
TYPES 2007, Lecture Notes in Computer Science 4941: 85-99 (2007)


M. Dezani-Ciancaglini, S. Ghilezan, and S. Likavec.
Behavioural inverse limit lambda models.
Theoretical Computer Science 316:49-74, (2004)


S. Ghilezan, S. Likavec.
Reducibility: a ubiquitous method in lambda calculus with intersection types.
ITRS 2002, ENTCS 70 (2003)


S. Ghilezan, V. Kunčak.
Confluence of Untyped Lambda Calculus via Simple Types.
ICTCS 2000, LNCS 2202: 38-49 (2001)


H. P. Barendregt, S. Ghilezan.
Lambda terms for natural deduction, sequent calculus and cut-elimination.
Journal of Functional Programming 10(1): 121-134 (2000)


S. Ghilezan.
Strong normalization and typability with intersection types.
Notre Dame Journal of Formal Logic 37: 44-53 (1996).