Type systems for programming languages

Didier Rémy

Academic year 2014-2015
Version of November 15, 2019

Contents

Chapter 8

Logical Relations

8.1 Introduction

Logical relations are relations between well-typed programs defined inductively on the structure of types.

There are two kinds of logical relations: unary and binary. Unary relations are predicate on expressions, while binary relations relates two expressions of the same type.

What they can be used for: unary relations can be used to prove type safety and strong normalisation; binary relations can be used to prove equivalence of programs and non-interferance properties.

Logical relations are a common proof method for programming language researchers that every one ought to know.

8.2 Normalization of simply-typed λ-calculus

In general, types also ensure termination of programs—as long as no form of recursion in types or terms has been added. Even if one wishes to add recursion explicitly later on, it is an important property of the design that non-termination is originating from the constructs for recursion only and could not occur without it.

The simply-typed λ-calculus is also lifted at the level of types in richer type systems such as System F^ω; then, the decidability of type-equality depends on the termination of the reduction at the type level.

Proving termination of reduction in fragments of the λ-calculus is often a difficult task because reduction may create new redexes or duplicate existing ones. However, the proof of termination for the simply-typed λ-calculus is simple enough and interesting to be presented here. Notice that our presentation of simply-typed λ-calculus is equipped with a call-by-value semantics, while proofs of termination are usually done with a strong evaluation strategy where reduction can occur in any context.

We follow the proof schema of Pierce (2002), which is a modern presentation in a call-by-value setting of an older proof by Hindley and Seldin (1986). The proof method, which is now a standard one, is due to Tait (1967). It consists in first building the set $\mathcal{T}_\tau$ of terminating closed terms of type τ, and then showing that any term of type τ is actually in $\mathcal{T}_\tau$, by induction on terms. Unfortunately, stated as such, this hypothesis is too weak. The difficulty in such cases is usually to find a strong enough induction hypothesis. The solution in this case is to require that terms in $\mathcal{T}_{\tau_1 \to \tau_2}$ not only terminate but also terminate when applied to any term in $\mathcal{T}_{\tau_1}$.

Definition 4 *Let $\mathcal{T}_\tau$ be defined inductively on τ as follows: let $\mathcal{T}_\alpha$ be the set of closed terms that terminates; let $\mathcal{T}_{\tau_2 \to \tau_1}$ be the set of (closed) terms M_1 of type $\tau_2 \to \tau_1$ that terminates and such that $M_1\ M_2$ is in $\mathcal{T}_{\tau_1}$ for any (closed) term M_2 in $\mathcal{T}_{\tau_2}$.*

The set $\mathcal{T}_\tau$ can be seen as a predicate, *i.e.* a unary relation. It is called a (unary) *logical relation* because it is defined inductively on the structure of types. The following proof is then schematic of the use of logical relations.

We state two obvious lemmas to prepare for the main proof. All terms in $\mathcal{T}_\tau$ terminate, by definition of $\mathcal{T}_\tau$:

Lemma 37 *For any type τ, the reduction of any term in $\mathcal{T}_\tau$ halts.*

Reduction of closed terms of type τ preserves membership in $\mathcal{T}_\tau$:

Lemma 38 *If $\varnothing \vdash M : \tau$ and $M \longrightarrow M'$, then $M \in \mathcal{T}_\tau$ iff $M' \in \mathcal{T}_\tau$.*

(Proof p. 173)

Therefore, it just remains to show that any term of type τ is in $\mathcal{T}_\tau$:

Lemma 39 *If $\varnothing \vdash M : \tau$, then $M \in \mathcal{T}_\tau$.*

The proof is by induction on (the typing derivation of) M. However, the case for abstraction requires some similar statement, but for open terms. We need to strengthen the lemma. Actually, to avoid considering open terms, we instead require the statement to hold for all closed instances of an open term:

Lemma 40 (strengthened) *If $(x_i : \tau_i)^{i \in I} \vdash M : \tau$, then for any closed values $(V_i)^{i \in I}$ in $(\mathcal{T}_{\tau_i})^{i \in I}$, the term $[(x_i \mapsto V_i)^{i \in I}]M$ is in $\mathcal{T}_\tau$.*

<u>Proof</u>: We write Γ for $(x_i : \tau_i)^{i \in I}$ and θ for $[(x_i \mapsto V_i)^{i \in I}]$. Assume $\Gamma \vdash M : \tau$ (**1**) and $(V_i)^{i \in I}$ in $(\mathcal{T}_{\tau_i})^{i \in I}$ (**2**). We show that θM is in $\mathcal{T}_\tau$ (**3**) by structural induction on M.

Case M is x_i: Immediate since the conclusion (3) is one of the hypothesese (2).

Case M is $M_1\ M_2$: By inversion of the typing judgment (1), we have $\Gamma \vdash M_1 : \tau_2 \to \tau$ (**4**) and $\Gamma \vdash M_2 : \tau_2$ (**5**) for some type τ_2. Therefore, by induction hypothesis applied to (4) and (5), we have $\theta M_1 \in \mathcal{T}_{\tau_2 \to \tau}$ and $\theta M_2 \in \mathcal{T}_{\tau_2}$. Thus, by definition of $\mathcal{T}_\tau$, we have$(\theta M_1)\ (\theta M_2) \in \mathcal{T}_\tau$; that is, $\theta M \in \mathcal{T}_\tau$.

Case M is $\lambda x{:}\tau_1 .\, M_2$: By inversion of the typing judgment (1), we have $\Gamma, x : \tau_1 \vdash M_2 : \tau_2$ (**6**) where $\tau_1 \to \tau_2$ is τ (**7**). Since M is a value, it is terminating. Hence, to ensure (3), it suffices to show that the application of θM to any M_1 in $\mathcal{T}_{\tau_1}$ is in $\mathcal{T}_{\tau_2}$ (**8**). Let $M_1 \in \mathcal{T}_{\tau_1}$. By definition of $\mathcal{T}_{\tau_1}$, the term M_1 reduces to some value V, which by subject reduction has type τ_1, and so is in $\mathcal{T}_{\tau_1}$ (**9**). We have:

$$\begin{array}{rcll}
(\theta M)\ M_1 & \triangleq & (\theta(\lambda x{:}\tau_1 .\, M_2))\ M_1 & \text{by definition of } M \\
 & = & (\lambda x{:}\tau_1 .\, \theta M_2)\ M_1 & \text{choose } x \# \vec{x} \\
 & \longrightarrow^* & (\lambda x{:}\tau_1 .\, \theta M_2)\ V & \text{by (9)} \\
 & \longrightarrow & [x \mapsto V](\theta M_2) & \text{by } (\beta) \\
 & = & ([x \mapsto V]\theta) M_2 & \\
 & \in & \mathcal{T}_{\tau_2} & \text{by I.H.}
\end{array}$$

In the last step, we may apply the induction hypothesis, since the first hypothesis is (6) and the second one follows from (2) and (9). In summary, $(\theta M)\ M_1$ reduces to a term in $\mathcal{T}_{\tau_2}$. Since $\mathcal{T}_{\tau_2}$ is closed by reduction, $(\theta M)\ M_1$ itself in in $\mathcal{T}_{\tau_2}$, which establishes (8), as expected.

8.3 Proofs and Solution to Exercises

Proof of Lemma 38

By induction on the structure of the type τ.

Case τ is α: Then $\mathcal{T}_\tau$ is the set of terms that terminates. If $M \longrightarrow M'$, the termination of M, *i.e.* $M \in \mathcal{T}_\alpha$, is equivalent to the termination of M', *i.e.* $M' \in \mathcal{T}_\alpha$.

Case τ is $\tau_1 \to \tau_2$: Then $\mathcal{T}_\tau$ is the set of terms of type τ that terminate and also terminate when applied to any term M_1 of type τ_1. Assume $\varnothing \vdash M : \tau$ (**1**) and $M \longrightarrow M'$ (**2**). By subject reduction, we have $\varnothing \vdash M' : \tau$. Moreover, from (2), termination of M and termination of M' are equivalent. Therefore, it only remains to check that for any term M_1 of $\mathcal{T}_{\tau_1}$, $M\ M_1$ and $M'\ M_1$ are both in $\mathcal{T}_{\tau_2}$ or both outside of $\mathcal{T}_{\tau_2}$ (**3**). Let M_1 be in $\mathcal{T}_{\tau_1}$. We have $\varnothing \vdash M_1 : \tau$ and thus $\varnothing \vdash M\ M_1 : \tau_2$. We also have the call-by-value reduction $M\ M_1 \longrightarrow M'\ M_1$, Hence, (3) follows by induction hypothesis.

Bibliography

▷ A tour of scala: Implicit parameters. Part of scala documentation.

▷ Martín Abadi and Luca Cardelli. A theory of primitive objects: Untyped and first-order systems. *Information and Computation*, 125(2):78–102, March 1996.

▷ Martín Abadi and Luca Cardelli. A theory of primitive objects: Second-order systems. *Science of Computer Programming*, 25(2–3):81–116, December 1995.

▷ Amal Ahmed and Matthias Blume. Typed closure conversion preserves observational equivalence. In *ACM International Conference on Functional Programming (ICFP)*, pages 157–168, September 2008.

▷ Lennart Augustsson. Implementing Haskell overloading. In *FPCA '93: Proceedings of the conference on Functional programming languages and computer architecture*, pages 65–73, New York, NY, USA, 1993. ACM. ISBN 0-89791-595-X.

▷ Nick Benton and Andrew Kennedy. Exceptional syntax journal of functional programming. *J. Funct. Program.*, 11(4):395–410, 2001.

▷ Richard Bird and Lambert Meertens. Nested datatypes. In *International Conference on Mathematics of Program Construction (MPC)*, volume 1422 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 1998.

Nikolaj Skallerud Bjørner. Minimal typing derivations. In *In ACM SIGPLAN Workshop on ML and its Applications*, pages 120–126, 1994.

Daniel Bonniot. *Typage modulaire des multi-méthodes.* PhD thesis, École des Mines de Paris, November 2005.

▷ Daniel Bonniot. Type-checking multi-methods in ML (a modular approach). In *Workshop on Foundations of Object-Oriented Languages (FOOL)*, January 2002.

▷ Michael Brandt and Fritz Henglein. Coinductive axiomatization of recursive type equality and subtyping. *Fundamenta Informaticæ*, 33:309–338, 1998.

▷ Kim B. Bruce, Luca Cardelli, and Benjamin C. Pierce. Comparing object encodings. *Information and Computation*, 155(1/2):108–133, November 1999.

Luca Cardelli. An implementation of f¡:. Technical report, DEC Systems Research Center, 1993.

Giuseppe Castagna. *Object-Oriented Programming: A Unified Foundation.* Progress in Theoretical Computer Science Series. Birkäuser, Boston, 1997.

▷ Henry Cejtin, Matthew Fluet, Suresh Jagannathan, and Stephen Weeks. The MLton compiler, 2007.

▷ Arthur Charguéraud and François Pottier. Functional translation of a calculus of capabilities. In *ACM International Conference on Functional Programming (ICFP)*, pages 213–224, September 2008.

▷ Juan Chen and David Tarditi. A simple typed intermediate language for object-oriented languages. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 38–49, January 2005.

▷ Adam Chlipala. A certified type-preserving compiler from lambda calculus to assembly language. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, pages 54–65, June 2007.

▷ Karl Crary, Stephanie Weirich, and Greg Morrisett. Intensional polymorphism in type erasure semantics. *Journal of Functional Programming*, 12(6):567–600, November 2002.

Julien Crétin and Didier Rémy. Extending System F with Abstraction over Erasable Coercions. In *Proceedings of the 39th ACM Conference on Principles of Programming Languages*, January 2012.

▷ Derek Dreyer, Robert Harper, Manuel M. T. Chakravarty, and Gabriele Keller. Modular type classes. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 63–70, New York, NY, USA, 2007. ACM. ISBN 1-59593-575-4.

Joshua Dunfield. Greedy bidirectional polymorphism. In *ML '09: Proceedings of the 2009 ACM SIGPLAN workshop on ML*, pages 15–26, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-509-3. doi: http://doi.acm.org/10.1145/1596627.1596631.

▷ Ken-etsu Fujita and Aleksy Schubert. Existential type systems with no types in terms. In *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings*, pages 112–126, 2009. doi: 10.1007/978-3-642-02273-9_10.

Jun Furuse. Extensional polymorphism by flow graph dispatching. In Ohori (2003), pages 376–393. ISBN 3-540-20536-5.

▷ Jun Furuse. Extensional polymorphism by flow graph dispatching. In *Asian Symposium on Programming Languages and Systems (APLAS)*, volume 2895 of *Lecture Notes in Computer Science*. Springer, November 2003b.

▷ Jacques Garrigue. Relaxing the value restriction. In *Functional and Logic Programming*, volume 2998 of *Lecture Notes in Computer Science*, pages 196–213. Springer, April 2004.

Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse d'état, Université Paris 7, June 1972.

▷ Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1990.

▷ Dan Grossman. Quantified types in an imperative language. *ACM Transactions on Programming Languages and Systems*, 28(3):429–475, May 2006.

▷ Bob Harper and Mark Lillibridge. ML with callcc is unsound. Message to the TYPES mailing list, July 1991.

Robert Harper and Benjamin C. Pierce. Design considerations for ML-style module systems. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 8, pages 293–345. MIT Press, 2005.

▷ Fritz Henglein. Type inference with polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2):253–289, April 1993.

▷ J. Roger Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and Lambda-Calculus*. Cambridge University Press, 1986.

▷ Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler. A history of Haskell: being lazy with class. In *ACM SIGPLAN Conference on History of Programming Languages*, June 2007.

Gérard Huet. *Résolution d'équations dans des langages d'ordre* $1, 2, \ldots, \omega$. PhD thesis, Université Paris 7, September 1976.

▷ John Hughes. Why functional programming matters. *Computer Journal*, 32(2):98–107, 1989.

▷ Mark P. Jones. Simplifying and improving qualified types. In *FPCA '95: Proceedings of the seventh international conference on Functional programming languages and computer architecture*, pages 160–169, New York, NY, USA, 1995a. ACM. ISBN 0-89791-719-7.

Mark P. Jones. Typing Haskell in Haskell. In *In Haskell Workshop*, 1999a.

Mark P. Jones. *Qualified types: theory and practice*. Cambridge University Press, New York, NY, USA, 1995b. ISBN 0-521-47253-9.

▷ Mark P. Jones. Typing Haskell in Haskell. In *Haskell workshop*, October 1999b.

▷ Simon Peyton Jones, Mark Jones, and Erik Meijer. Type classes: an exploration of the design space. In *Haskell workshop*, 1997.

▷ Simon Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Mark Shields. Practical type inference for arbitrary-rank types. *Journal of Functional Programming*, 17(01):1, 2006.

Stefan Kaes. Type inference in the presence of overloading, subtyping and recursive types. In *LFP '92: Proceedings of the 1992 ACM conference on LISP and functional programming*, pages 193–204, New York, NY, USA, 1992. ACM. ISBN 0-89791-481-3. doi: http://doi.acm.org/10.1145/141471.141540.

▷ Assaf J. Kfoury, Jerzy Tiuryn, and Pawel Urzyczyn. ML typability is DEXPTIME-complete. In *Colloquium on Trees in Algebra and Programming*, volume 431 of *Lecture Notes in Computer Science*, pages 206–220. Springer, May 1990.

▷ Peter J. Landin. Correspondence between ALGOL 60 and Church's lambda-notation: part I. *Communications of the ACM*, 8(2):89–101, 1965.

▷ Konstantin Läufer and Martin Odersky. Polymorphic type inference and abstract data types. *ACM Transactions on Programming Languages and Systems*, 16(5):1411–1430, September 1994.

▷ Didier Le Botlan and Didier Rémy. Recasting MLF. *Information and Computation*, 207(6): 726–785, 2009. ISSN 0890-5401. doi: 10.1016/j.ic.2008.12.006.

▷ Xavier Leroy. *Typage polymorphe d'un langage algorithmique*. PhD thesis, Université Paris 7, June 1992.

▷ Xavier Leroy. Formal certification of a compiler back-end or: programming a compiler with a proof assistant. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 42–54, January 2006.

▷ Xavier Leroy and François Pessaux. Type-based analysis of uncaught exceptions. *ACM Trans. Program. Lang. Syst.*, 22(2):340–377, 2000. ISSN 0164-0925. doi: http://doi.acm.org/10.1145/349214.349230.

▷ John M. Lucassen and David K. Gifford. Polymorphic effect systems. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 47–57, January 1988.

▷ Harry G. Mairson. Deciding ML typability is complete for deterministic exponential time. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 382–401, 1990.

▷ David McAllester. A logical algorithm for ML type inference. In *Rewriting Techniques and Applications (RTA)*, volume 2706 of *Lecture Notes in Computer Science*, pages 436–451. Springer, June 2003.

Todd D. Millstein and Craig Chambers. Modular statically typed multimethods. In *ECOOP '99: Proceedings of the 13th European Conference on Object-Oriented Programming*, pages 279–303, London, UK, 1999. Springer-Verlag. ISBN 3-540-66156-5.

▷ Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, December 1978.

▷ Yasuhiko Minamide, Greg Morrisett, and Robert Harper. Typed closure conversion. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 271–283, January 1996.

▷ John C. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 76(2–3):211–249, 1988.

▷ John C. Mitchell and Gordon D. Plotkin. Abstract types have existential type. *ACM Transactions on Programming Languages and Systems*, 10(3):470–502, 1988.

▷ Benoît Montagu and Didier Rémy. Modeling abstract types in modules with open existential types. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 63–74, January 2009.

J. Garrett Morris and Mark P. Jones. Instance chains: type class programming without overlapping instances. In *ICFP '10: Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, pages 375–386, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-794-3. doi: http://doi.acm.org/10.1145/1863543.1863596.

▷ Greg Morrisett and Robert Harper. Typed closure conversion for recursively-defined functions (extended abstract). In *International Workshop on Higher Order Operational Techniques in Semantics (HOOTS)*, volume 10 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 1998.

▷ Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From system F to typed assembly language. *ACM Transactions on Programming Languages and Systems*, 21(3):528–569, May 1999.

▷ Alan Mycroft. Polymorphic type schemes and recursive definitions. In *International Symposium on Programming*, volume 167 of *Lecture Notes in Computer Science*, pages 217–228. Springer, April 1984.

▷ Matthias Neubauer, Peter Thiemann, Martin Gasbichler, and Michael Sperber. Functional logic overloading. pages 233–244, 2002. doi: http://doi.acm.org/10.1145/565816.503294.

▷ Martin Odersky, Philip Wadler, and Martin Wehr. A second look at overloading. In *FPCA '95: Proceedings of the seventh international conference on Functional programming languages and computer architecture*, pages 135–146, New York, NY, USA, 1995. ACM. ISBN 0-89791-719-7.

▷ Martin Odersky, Martin Sulzmann, and Martin Wehr. Type inference with constrained types. *Theory and Practice of Object Systems*, 5(1):35–55, 1999.

▷ Martin Odersky, Matthias Zenger, and Christoph Zenger. Colored local type inference. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 41–53, 2001.

Atsushi Ohori, editor. *Programming Languages and Systems, First Asian Symposium, APLAS 2003, Beijing, China, November 27-29, 2003, Proceedings*, volume 2895 of *Lecture Notes in Computer Science*, 2003. Springer. ISBN 3-540-20536-5.

▷ Chris Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1999.

▷ Bruno C.d.S. Oliveira, Tom Schrijvers, Wontae Choi, Wonchan Lee, and Kwangkeun Yi. The implicit calculus: a new foundation for generic programming. In *Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation*, PLDI '12, pages 35–44, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1205-9. doi: 10.1145/2254064.2254070.

▷ Simon Peyton Jones. Tackling the awkward squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell. Online lecture notes, January 2009.

▷ Simon Peyton Jones and Mark Shields. Lexically-scoped type variables. Manuscript, April 2004.

▷ Simon Peyton Jones and Philip Wadler. Imperative functional programming. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 71–84, January 1993.

Frank Pfenning. Partial polymorphic type inference and higher-order unification. In *LFP '88: Proceedings of the 1988 ACM conference on LISP and functional programming*, pages 153–163, New York, NY, USA, 1988. ACM. ISBN 0-89791-273-X. doi: http://doi.acm.org/10.1145/62678.62697.

▷ Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

▷ Benjamin C. Pierce and David N. Turner. Local type inference. *ACM Transactions on Programming Languages and Systems*, 22(1):1–44, January 2000.

▷ Andrew M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10:321–359, 2000.

▷ François Pottier. Notes du cours de DEA "Typage et Programmation", December 2002.

François Pottier. A typed store-passing translation for general references. In *Proceedings of the 38th ACM Symposium on Principles of Programming Languages (POPL'11)*, Austin, Texas, January 2011. Supplementary material.

François Pottier. Syntactic soundness proof of a type-and-capability system with hidden state. *Journal of Functional Programming*, 23(1):38–144, January 2013.

François Pottier. Hindley-Milner elaboration in applicative style. In *Proceedings of the 2014 ACM SIGPLAN International Conference on Functional Programming (ICFP'14)*, September 2014.

▷ François Pottier and Nadji Gauthier. Polymorphic typed defunctionalization and concretization. *Higher-Order and Symbolic Computation*, 19:125–162, March 2006.

François Pottier and Jonathan Protzenko. Programming with permissions in Mezzo. Submitted for publication, October 2012.

François Pottier and Jonathan Protzenko. Programming with permissions in Mezzo. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Functional Programming (ICFP'13)*, pages 173–184, September 2013.

▷ François Pottier and Didier Rémy. The essence of ML type inference. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 10, pages 389–489. MIT Press, 2005.

▷ François Pottier and Didier Rémy. The essence of ML type inference. Draft of an extended version. Unpublished, September 2003.

▷ Didier Rémy. Simple, partial type-inference for System F based on type-containment. In *Proceedings of the tenth International Conference on Functional Programming*, September 2005.

▷ Didier Rémy. Programming objects with ML-ART: An extension to ML with abstract and record types. In *International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 321–346. Springer, April 1994a.

▷ Didier Rémy. Type inference for records in a natural extension of ML. In Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects Of Object-Oriented Programming: Types, Semantics and Language Design*. MIT Press, 1994b.

▷ Didier Rémy and Jérôme Vouillon. Objective ML: An effective object-oriented extension to ML. *Theory and Practice of Object Systems*, 4(1):27–50, 1998.

Didier Rémy and Boris Yakobowski. Efficient Type Inference for the MLF language: a graphical and constraints-based approach. In *The 13th ACM SIGPLAN International Conference on Functional Programming (ICFP'08)*, pages 63–74, Victoria, BC, Canada, September 2008. doi: http://doi.acm.org/10.1145/1411203.1411216.

▷ John C. Reynolds. Towards a theory of type structure. In *Colloque sur la Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer, April 1974.

▷ John C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing 83*, pages 513–523. Elsevier Science, 1983.

▷ John C. Reynolds. Three approaches to type structure. In *International Joint Conference on Theory and Practice of Software Development (TAPSOFT)*, volume 185 of *Lecture Notes in Computer Science*, pages 97–138. Springer, March 1985.

François Rouaix. Safe run-time overloading. In *Proceedings of the 17th ACM Conference on Principles of Programming Languages*, pages 355–366, 1990. doi: http://doi.acm.org/10.1145/96709.96746.

▷ Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. Cochis: Deterministic and coherent implicits. Technical report, KU Leuven, May 2017.

▷ Christian Skalka and François Pottier. Syntactic type soundness for HM(X). In *Workshop on Types in Programming (TIP)*, volume 75 of *Electronic Notes in Theoretical Computer Science*, July 2002.

Geoffrey S. Smith. Principal type schemes for functional programs with overloading and subtyping. In *Science of Computer Programming*, 1994.

Morten Heine Sørensen and Pawel Urzyczyn. *Studies in Logic and the Foundations of Mathematics*, chapter Lectures on the Curry-Howard Isomorphism. Elselvir Science Inc, 2006.

▷ Matthieu Sozeau and Nicolas Oury. First-Class Type Classes. In Sofiène Tahar, Otmame Ait-Mohamed, and César Muñoz, editors, *TPHOLs 2008: Theorem Proving in Higher Order Logics, 21th International Conference*, Lecture Notes in Computer Science. Springer, August 2008.

▷ Paul A. Steckler and Mitchell Wand. Lightweight closure conversion. *ACM Transactions on Programming Languages and Systems*, 19(1):48–86, 1997.

▷ Christopher Strachey. Fundamental concepts in programming languages. *Higher-Order and Symbolic Computation*, 13(1–2):11–49, April 2000.

▷ Peter J. Stuckey and Martin Sulzmann. A theory of overloading. In *ICFP '02: Proceedings of the seventh ACM SIGPLAN international conference on Functional programming*, pages 167–178, New York, NY, USA, 2002. ACM. ISBN 1-58113-487-8.

▷ W. W. Tait. Intensional interpretations of functionals of finite type i. *The Journal of Symbolic Logic*, 32(2):pp. 198–212, 1967. ISSN 00224812.

▷ Jean-Pierre Talpin and Pierre Jouvelot. The type and effect discipline. *Information and Computation*, 11(2):245–296, 1994.

Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, April 1975.

▷ Jerzy Tiuryn and Pawel Urzyczyn. The subtyping problem for second-order types is undecidable. *Information and Computation*, 179(1):1–18, 2002.

▷ Mads Tofte, Lars Birkedal, Martin Elsman, and Niels Hallenberg. A retrospective on region-based memory management. *Higher-Order and Symbolic Computation*, 17(3):245–265, September 2004.

▷ Andrew Tolmach and Dino P. Oliva. From ML to Ada: Strongly-typed language interoperability via source translation. *Journal of Functional Programming*, 8(4):367–412, July 1998.

▷ Philip Wadler. Theorems for free! In *Conference on Functional Programming Languages and Computer Architecture (FPCA)*, pages 347–359, September 1989.

▷ Philip Wadler. The Girard-Reynolds isomorphism (second edition). *Theoretical Computer Science*, 375(1–3):201–226, May 2007.

▷ Philip Wadler and Stephen Blott. How to make ad-hoc polymorphism less ad-hoc. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 60–76, January 1989.

Mitchell Wand. Corrigendum: Complete type inference for simple objects. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, 1988.

▷ J. B. Wells. The essence of principal typings. In *International Colloquium on Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 913–925. Springer, 2002.

▷ J. B. Wells. The undecidability of Mitchell's subtyping relation. Technical Report 95-019, Computer Science Department, Boston University, December 1995.

▷ J. B. Wells. Typability and type checking in system F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1–3):111–156, 1999.

▷ Andrew K. Wright. Simple imperative polymorphism. *Lisp and Symbolic Computation*, 8 (4):343–356, December 1995.

▷ Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, November 1994.