Type Inference Techniques: Implementation and Formalization, Better Together

Ningning Xie
ningningxie@cs.toronto.edu
University of Toronto

Andong Fan
andong@cs.toronto.edu
University of Toronto

Type inference plays a key role in the practical implementations of type systems. In the implementations of type inference, such as those in GHC and the OCaml compiler, various *implementation techniques* are often incorporated. While these implementation techniques can improve algorithmic efficiency and lead to a more uniform handling of multiple type features, they often exist only as *folklore* within the compiler source code, understandable only to a select group of implementers. Moreover, these techniques are not always reflected in the formal specifications of type inference, compromising the theoretical guarantees established by the formalism.

In this talk, we argue that formalization of type inference implementation techniques is crucial in its own right. It exposes folklore to a wider community, and places implementation techniques on a more secured theoretical foundation, uncovering subtle bugs in the implementation. A precise and formal understanding also allows for exploration of a wider design space with different tradeoffs of the algorithms. We present our ongoing efforts in bringing various type inference implementation techniques and their formalization together, under different settings of type system features, including bidirectional typing for higher-rank polymorphism [1, 7], constraint-based type inference for generalized algebraic data types (GADTs) [10], and its generalization to type classes and type families [11], among other interesting directions.

Higher-Rank Polymorphism with Levels

Levels, an implementation technique originally proposed by Rémy [8], have been widely adopted in the modern type inference engines. Moving beyond its original application for sound and efficient implementation of *let generalization* [4], levels have become a crucial concept that governs *scope* at the type level in practical type checkers, including checking if *skolem* (or *rigid*) type variables escape their scope, in particular with the presence of higher-rank polymorphic types and their subtyping [1, 6, 7]. We present the first formalism of levels for higher-rank polymorphic type inference and local data types (aka type regions) with bidirectional typing [2]. We establish novel level-based properties and invariants of the algorithm. The soundness and completeness results between specifications of the algorithm with and without levels are mechanized in Rocq. Furthermore, we provide an implementation of levels for the type inference engine in Koka [5] with empirical evidence showing that level-based type inference is indeed more efficient in practice.

GADTs with Levels

Although GADTs bring extra expressiveness to types, type inference for GADTs [12] is notoriously difficult, as *local type equality assumptions* introduced by pattern matching of GADTs bring ambiguity that risks the existence of principal types. *OutsideIn* [10] is the approach implemented in GHC for principal type inference with GADTs. By forbidding unification to happen on a set of *untouchable* type variables, i.e. those with lower levels from the "outside", it effectively prevents local assumptions from leaking out of their scope. We present an ongoing work on a formalism of the constraint-based OutsideIn algorithm with levels and establish its meta-theoretical properties, including principality of type inference, in the new level-based setting. This level-based formalism further serves as a foundation for exploration of the design space of the algorithm, and uncovers a subtle bug in the implementation of GHC, which caused loss of principality of the inferred types.

Future Directions

We discuss a few directions we are currently exploring. The implementation of OutsideIn in GHC maintains a crucial data structure called *inert set*, as the backbone of its constraint solver based on *interactions* between constraints, which handles an enriched set of constraints including type class constraints and type family axioms [11], and stores constraints currently unsolvable until further information comes in light. We are interested in a rigorous formal understanding of it and its properties, such as termination of the solver, and its interaction with levels. Moreover, OCaml employs the *ambivalent type* [3] approach for principal type inference with GADTs, implemented by keeping track of an additional *scope* number on types. We are interested in its combination with the constraint-based OutsideIn approach, as hinted by Rémy [9], in a level-based framework. Furthermore, we are interested in mechanizing the level-based algorithm for higher-rank type inference, as levels shed new light on mechanization of type inference algorithms, where it is crucial to ensure well-scoped unification.

WITS '26, Rennes, France
2026.

References

[1] Jana Dunfield and Neelakantan R Krishnaswami. 2013. Complete and easy bidirectional typechecking for higher-rank polymorphism. In *Proceedings of the 18th ACM SIGPLAN international conference on Functional programming*. 429–442.

[2] Andong Fan, Han Xu, and Ningning Xie. 2025. Practical Type Inference with Levels. In *Proceedings of the 46th ACM SIGPLAN Conference on Programming Language Design and Implementation*.

[3] Jacques Garrigue and Didier Rémy. 2013. Ambivalent types for principal type inference with GADTs. In *Programming Languages and Systems: 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11, 2013. Proceedings 11*. Springer, 257–272.

[4] Oleg Kiselyov. 2022. How OCaml type checker works – or what polymorphism and garbage collection have in common. (2022). https://okmij.org/ftp/ML/generalization.html

[5] Daan Leijen. 2013. *Koka: Programming with Row-Polymorphic Effect Types*. Technical Report MSR-TR-2013-79. Microsoft. https://www.microsoft.com/en-us/research/publication/koka-programming-with-row-polymorphic-effect-types/

[6] Martin Odersky and Konstantin Läufer. 1996. Putting type annotations to work. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (St. Petersburg Beach, Florida, USA) *(POPL '96)*. Association for Computing Machinery, New York, NY, USA, 54–67. doi:10.1145/237721.237729

[7] Simon Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Mark Shields. 2007. Practical type inference for arbitrary-rank types. *Journal of functional programming* 17, 1 (2007), 1–82.

[8] Didier Rémy. 1992. *Extension of ML type system with a sorted equation theory on types*. Ph. D. Dissertation. INRIA.

[9] Didier Rémy. 2013. Ambivalent Types for Principal Type Inference with GADTs. Slides, IFIP WG 2.8—Functional Programming meeting. https://gallium.inria.fr/~remy/gadts/ifip-wg2.8-2013.pdf

[10] Tom Schrijvers, Simon Peyton Jones, Martin Sulzmann, and Dimitrios Vytiniotis. 2009. Complete and decidable type inference for GADTs. In *Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming* (Edinburgh, Scotland) *(ICFP '09)*. Association for Computing Machinery, New York, NY, USA, 341–352. doi:10.1145/1596550.1596599

[11] Dimitrios Vytiniotis, Simon Peyton Jones, Tom Schrijvers, and Martin Sulzmann. 2011. OutsideIn (X) Modular type inference with local assumptions. *Journal of functional programming* 21, 4-5 (2011), 333–412.

[12] Hongwei Xi, Chiyan Chen, and Gang Chen. 2003. Guarded recursive datatype constructors. In *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (New Orleans, Louisiana, USA) *(POPL '03)*. Association for Computing Machinery, New York, NY, USA, 224–235. doi:10.1145/604131.604150