Deriving Higher-Order Unification in Haskell

Anonymous Author(s)

Abstract

When implementing type inference for a dependent type system, one of the cornerstones is higher-order unification. Even though many such algorithms exist, state-of-the-art is implementing it directly for a particular proof assistant. We propose an approach inspired by second-order abstract syntax of Fiore, data types là carte of Swierstra, and instrinsic scoping of Bird and Patterson. Our approach allows to handle scopes in a language-agnostic way and provide generic higher-order unification algorithms, which then can serve as foundation for the implementation of dependent type inference.

Keywords: second-order abstract syntax, equational unification, algebraic data type

To reduce explicit types in proofs involving dependent types, a proof assistant requires type inference, which in turn often relies on higher-order unification. Many unification algorithms exist [4, 8, 10], but implementing them requires extra effort and is often error-prone. For these reasons, in prototype implementations type inference is often omitted or reduced, as developers opt out for a more straightforward implementation while limiting the capabilities of a prototype. At the same time, in a sufficiently complex dependently typed language, even small examples can be challenging to comprehend without some type inference.

In his 2001 pearl [9], Sheard described an efficient and modularized implementation of single-sorted first-order unification. Wren Romano has implemented this approach in the Haskell programming language as the `unification-fd` library. Romano's implementation also mixes well with Swierstra's data types à la carte [11]: terms with metavariables are constructed using free monads.

Free monad construction is a common technique for modeling side effects (e.g., input/output) in embedded domain-specific languages [12, 13]. However, they are also used to generate abstract syntax trees for terms [11], where the monadic binding operation corresponds to a substitution of variables. Such representation allows for more flexibility in terms of modular extensions to the language of terms and annotations (such as source code location or types).

Unfortunately, free monads cannot be used directly for higher-order unification, as we have to take extra care of the bound variables. For expressions with scopes (such as let-expressions or λ-abstractions), substitution (implemented manually or via free monads) is not safe by default since a name capture might happen. To avoid this, de Bruijn indices [1] are commonly used in practice. However, recently generalized de Bruijn indices[1] have also been used (e.g. in Epigram [7]) to keep track of scoping in types and also to allow lifting entire subexpressions to optimize substitutions further.

Second-order abstract syntax [2, 3] has proven useful when working with languages with arbitrary binding constructions. Moreover, higher-order unification can be expressed as a special case of equational unification for second-order abstract syntax [6].

We present a work-in-progress on an approach to specification of abstract syntax in Haskell that combines features of second-order abstract syntax of Fiore, data types à la carte of Swierstra, and intrinsic scoping of Bird and Patterson, allowing us to work comfortably with the syntax tree while also being able to derive a higher-order unification algorithm for such syntax, given reduction rules or a set of equations.

Earlier version of the approach [5] with heuristics based on Huet's pre-unification algorithm [4] has been implemented in the first version of a prototype proof assistant Rzk (tag `v0.1.0`[2]).

References

[1] N.G de Bruijn. 1972. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)* 75, 5 (1972), 381–392. https://doi.org/10.1016/1385-7258(72)90034-0

[2] Marcelo Fiore and Chung-Kil Hur. 2010. Second-Order Equational Logic (Extended Abstract). In *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6247)*, Anuj Dawar and Helmut Veith (Eds.). Springer, 320–335. https://doi.org/10.1007/978-3-642-15205-4_26

[3] Marcelo Fiore and Dmitrij Szamozvancev. 2022. Formal Metatheory of Second-Order Abstract Syntax. *Proc. ACM Program. Lang.* 6, POPL, Article 53 (jan 2022), 29 pages. https://doi.org/10.1145/3498715

[4] G.P. Huet. 1975. A unification algorithm for typed λ-calculus. *Theoretical Computer Science* 1, 1 (1975), 27–57. https://doi.org/10.1016/0304-3975(75)90011-0

[1] such as implemented in the bound package, available at http://hackage.haskell.org/package/bound

[2] see https://github.com/rzk-lang/rzk/tree/v0.1.0#readme

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

[5] Nikolai Kudasov. 2022. Functional Pearl: Dependent type inference via free higher-order unification. arXiv:2204.05653 [cs.LO]

[6] Nikolai Kudasov. 2023. E-Unification for Second-Order Abstract Syntax. In *8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 260)*, Marco Gaboardi and Femke van Raamsdonk (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 10:1–10:22. https://doi.org/10.4230/LIPIcs.FSCD.2023.10

[7] Conor McBride. 2004. Epigram: Practical Programming with Dependent Types. In *Proceedings of the 5th International Conference on Advanced Functional Programming* (Tartu, Estonia) *(AFP'04)*. Springer-Verlag, Berlin, Heidelberg, 130–170. https://doi.org/10.1007/11546382_3

[8] Dale Miller. 1991. A logic programming language with lambda-abstraction, function variables, and simple unification. In *Extensions of Logic Programming*, Peter Schroeder-Heister (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 253–281.

[9] Tim Sheard. 2001. Generic Unification via Two-Level Types and Parameterized Modules - Functional Pearl. *Sigplan Notices - SIGPLAN* 36 (10 2001), 86–97. https://doi.org/10.1145/507546.507648

[10] Wayne Snyder. 1990. Higher order E-unification. In *10th International Conference on Automated Deduction*, Mark E. Stickel (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 573–587.

[11] Wouter Swierstra. 2008. Data types à la carte. *Journal of Functional Programming* 18, 4 (2008), 423–436. https://doi.org/10.1017/S0956796808006758

[12] Janis Voigtländer. 2008. Asymptotic Improvement of Computations over Free Monads. 388–403. https://doi.org/10.1007/978-3-540-70594-9_20

[13] Nicolas Wu and Tom Schrijvers. 2015. Fusion for Free. In *Mathematics of Program Construction*, Ralf Hinze and Janis Voigtländer (Eds.). Springer International Publishing, Cham, 302–322.