# Towards a formalization of Modular Implicits

MPRI research internship, 2023

|  |  |  |
|---|---|---|
| Subject: | Towards a formalization of Modular Implicits | |
| Supervisors: | Didier Rémy | Didier.Remy@inria.fr |
| | Gabriel Scherer | Gabriel.Scherer@inria.fr |
| Location: | Centre Inria de Paris | |

## Context

This research is motivated by the introduction of modular implicits in the OCaml language. A proof of concept proposal [White, Bour, and Yallop, 2014], has been implemented as an experimental branch of (version 4.02 of) OCaml. This was quite promising, as it achieved a form of dynamic overloading similar to type classes in Haskell, but reasoning at the level of modules, hence in a way that is modular by construction and well-integrated in a language such as OCaml. However, many aspects remained to be further explored, redesigned, and formalized, in order to obtain a predictive, robust, and efficient implementation of modular implicits.

The idea of using the module system to mimic type classes was first introduced by [Dreyer, Harper, Chakravarty, and Keller, 2007], but without any attempt to *elaborate* module arguments. That is, the user had to provide himself the “instances”. In parallel, emulation of type classes with implicit arguments was first proposed by [Lewis, Launchbury, Meijer, and Shields, 2000], but became more popular after their introduction in the Scala language [Oliveira, Moors, and Odersky, 2010]. Modular implicits combine both approaches, using the module system instead of the core language to emulate type classes. They use implicit *module* arguments to automate the construction of dictionaries, and search for the right applications of functors to module arguments among those that have been explicitly put in scope for that purpose.

Modular implicits build on two long lines of works on (1) the ML module system and (2) implicit arguments. Still, further investigation is required in both directions.

On the one hand, ML modules need to be extended, as explained in [White, Bour, and Yallop, 2014], to increase the interaction with the core language and, more importantly, to better trace sharing of applicative functors so as to avoid false ambiguities during the elaboration process. There is already work in this direction with an ongoing PhD by Clément Blaudeau on the formalization and improvement of OCaml modules based on an elaboration into $F^\omega$ and largely inspired by [Rossberg, Russo, and Dreyer, 2014] and [Rossberg, 2018].

On the other hand, the many works that explored the inference of implicit parameters do not directly apply to modular implicits for both practical and theoretical reasons.

While modules have originally been designed to program in the large, modular implicits will be used to program in the small—and used quite extensively. This will induce both a heavy use of modules and a stronger interaction between the core and module levels, which in turn may require more support from the programming language to improve the users' experience and from the compiler to avoid (or just reduce) the overhead when programming at a more abstract level. This is also an opportunity to revisit and reduce the stratification between the core and module levels, taking inspiration either from 1ML [Rossberg and Dreyer, 2013] or from the dot calculus [Amin et al., 2014] introduced to model the core of Scala.

Altogether, modular implicits will be a significant change to the language, which will encourage the use of the most advanced features of the module language, even in simple programs. This requires a robust design from the start, which in turn calls for a formal, eventually mechanized definition of (a large subset of) the language.

## Internship description

The goal of the internship is to focus on the resolution of implicits and explore it in a simplified setting. This should be a first key step towards the formalization of modular implicits.

The resolution of implicit modules amounts to inferring implicit module expressions from their types where implicit module expressions are limited to applications of modules to other modules taken in a database of pre-existing modules, that is, given a typing context mapping module names to their signatures (and program variables to first-order types).

To simplify, we will restrict to the case of generative functors and see modules and their signatures as their encoding in $\mathsf{F}^\omega$, thus as $\mathsf{F}^\omega$ terms. Therefore, the resolution amounts to proving formulas in higher-order logic, but with some particularities:

- We are not only interested in finding proofs, but checking that all $\mathsf{F}^\omega$ proof terms solutions to the inference problem are *observationally* equivalent, so as to ensure a deterministic semantics.
- We shall restrict the types of modules to those that encode signatures, which strongly constraints the position of universal and existential quantifiers.
- Given that implicit modules are inherently first-class modules, we have to deal with and arbitrate the interaction between the inference of modules from their types and the inference of the types of core language expressions.

Since the inference problem is undecidable in the general case, we will need to design restrictions of the problems that are decidable and tractable—and still sufficiently expressive.

Besides the design and formalization, the work will also require a prototype implementation and experimentation on small representative examples.

### Going further

The goal of the internship is just a proof of concept, i.e. having a precise formalization, even if over-restricted and limited to a subset of OCaml modules. However, the long term goal is to build on this internship to reimplement modular implicits in OCaml on solid bases. This will require to extend the work to applicative modules and may also necessitate to lift some of the restrictions or explore alternative compromises. In order to scale up, there may also be interesting algorithmic issues so as to efficiently deal with a large base of implicit modules.

## References

Nada Amin, Tiark Rompf, and Martin Odersky. Foundations of path-dependent types. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, pages 233–249, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2585-1. doi: 10.1145/2660193.2660216. URL http://doi.acm.org/10.1145/2660193.2660216.

Derek Dreyer, Robert Harper, Manuel M. T. Chakravarty, and Gabriele Keller. Modular type classes. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 63–70. ACM, 2007. ISBN 1-59593-575-4. doi: 10.1145/1190216.1190229. URL https://doi.org/10.1145/1190216.1190229.

Jeffrey R. Lewis, John Launchbury, Erik Meijer, and Mark B. Shields. Implicit Parameters: Dynamic Scoping with Static Types. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '00, pages 108–118, New York, NY, USA, 2000. ACM. ISBN 1-58113-125-9. doi: 10.1145/325694.325708. URL http://doi.acm.org/10.1145/325694.325708. event-place: Boston, MA, USA.

Bruno C. d S. Oliveira, Adriaan Moors, and Martin Odersky. Type classes as objects and implicits. In *Proceedings of the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA*, pages 341–360, 2010. doi: 10.1145/1869459.1869489. URL https://doi.org/10.1145/1869459.1869489.

Andreas Rossberg. 1ml - Core and modules united. *J. Funct. Program.*, 28:e22, 2018. doi: 10.1017/S0956796818000205. URL https://doi.org/10.1017/S0956796818000205.

Andreas Rossberg and Derek Dreyer. Mixin'Up the ML Module System. *ACM Trans. Program. Lang. Syst.*, 35(1):2:1–2:84, April 2013. ISSN 0164-0925. doi: 10.1145/2450136.2450137. URL http://doi.acm.org/10.1145/2450136.2450137.

Andreas Rossberg, Claudio Russo, and Derek Dreyer. F-ing modules. *Journal of Functional Programming*, 24(5):529–607, 2014. doi: 10.1017/S0956796814000264.

Leo White, Frédéric Bour, and Jeremy Yallop. Modular implicits. In Oleg Kiselyov and Jacques Garrigue, editors, *Proceedings ML Family/OCaml Users and Developers workshops, ML/OCaml 2014, Gothenburg, Sweden, September 4-5, 2014.*, volume 198 of *EPTCS*, pages 22–63, 2014. doi: 10.4204/EPTCS.198.2. URL https://doi.org/10.4204/EPTCS.198.2.