# Type indexing in OCaml: search and find functions in a large ecosystem

Gabriel RADANNE, Inria CASH/LIP
Laure Gonnord – Grenoble INP/LCIS & LIP/CASH

2021-2022

## 1 Context

Sometimes, we need a function so deeply that we have to go out and search for it. How do we find it ? Sometimes, we have a precise idea of the desired type : "this function has at least 2 parameters, a *bike* and a *date* and returns a boolean value". We will then search at some precise places (the directory containing the `Bike` module, for instance). This activity becomes more tedious when the ecosystem of the language becomes huge : for instance, the OCaml ecosystem contains 3259 opam packages, each with hundreds or thousands of functions.

We can thus imagine search tools that, given a type, find appropriate functions. Such a tool would avoid unnecessary boring details such that the order or the number of arguments and focus on the type form. The results of such search algorithms should be correct (the function indeed respects the signature) and complete (all such functions are found). Naturally, this procedure should also be fast in practice.

Concretely, in a language such that OCaml, we would like to search for a function of the following type : `int * bike list -> bike`. This search could find in the ecosystem functions like `int -> bike list -> bike`, or `List.nth`, and even `'a list -> int -> 'a`. The aim is thus also to find more general, or more specialized functions, along with functions with re-ordered arguments.

This idea was initiated 25 years ago under the name of "search modulo type isomorphisms" [5, 4]. Unfortunately, algorithms for unification modulo type isomorphisms are extremely costly, notably because they were tailored for statically-typed languages of that time, whose ecosystems were tiny (the OCaml standard library had 294 functions !). Nowadays, these algorithms do not scale anymore, and more modern approaches like Hoogle [2] use techniques which are neither correct nor complete.

## 2 Internship objective

Recently, we have developped `Dowsing`[1], a type-search tool on a set of OCaml packages [3, 1]. Our tool scales well on an ecosystem of medium-size (for instance, a local installation). The objective of this internship is to develop novel algorithmic techniques for searching in much larger ecosystems (the whole OCaml ecosystem, or even Typescript). In particular, we will take inspiration from indexing techniques in databases on the one hand, and term rewriting procedures on the other hand, to enable further scaling of our existing tool.

## 3 Internship

The internship will take place in the CASH Team, LIP lab, in Lyon France.

1. https ://github.com/Drup/dowsing



LIP – UMR CNRS / ENS Lyon / UCB Lyon 1 / Inria – 69007 Lyon
E-mail : gabriel.radanne@inria.fr and Laure.Gonnord@ens-lyon.fr

**Candidate profile** The candidate should ideally be familiar with formal approaches in programming language design, notably type systems and logic. They should also have taste for algorithmic design.

From the practical point of view, a basic experience in software programming and usage of collaborative tools such that `git`. Knowledge of the Ocaml programming language is mandatory.

This internship strongly relies on the fact that practical implementation should have strong theoritical functions and that further raffinements of the theory should get inspiration from the practical side. We expect the candidate to agree with this philosophy.

## Références


[1] Github dowsing. URL https://www.youtube.com/watch?v=rxUb-1eKEJM.

[2] Hoogle. URL https://hoogle.haskell.org/.

[3] Clément Allain, Gabriel Radanne, and Laure Gonnord. Isomorphisms are back ! In *ML 2021 - ML Workshop*, pages 1–3, Virtual, France, August 2021. URL https://hal.archives-ouvertes.fr/hal-03355381.

[4] Roberto Di Cosmo. A short survey of isomorphisms of types. *Mathematical Structures in Computer Science*, 15(5) :825–838, 2005. doi : $10.1017/\mathrm{S}0960129505004871$. URL https://doi.org/10.1017/S0960129505004871.

[5] Mikael Rittri. Using types as search keys in function libraries. *J. Funct. Program.*, 1(1) :71–89, 1991. doi : $10.1017/\mathrm{S}095679680000006\mathrm{X}$. URL https://doi.org/10.1017/S095679680000006X.