Intership: Good error messages and documentation for OCaml modules via diffing

Gabriel Radanne, Inria CASH/LIP
Florian Angeletti, Inria Cambium

2021

1 Context

OCaml is well known for providing powerful abstraction and modularity. It allows programmers to create and compose programs like Lego : by creating small self-contained bricks, assembling them together to in-turn present them as bigger building units. While this is convenient and powerful for programmers, the resulting module language is rich and can sometimes feel overwhelming to users. Notably, it makes it difficult to create good error messages and easy-to-read documentation.

For instance, whenever an OCaml module implementation (the actual code) and its interface (the exposed types) mismatch, it can be hard for the compiler to identify and report the exact source of the error. Consequently, type-checkers often resort to printing the whole module types, and hope that the human user will navigate the catalogue of definitions. In documentation, it can be similarly delicate to identify what changed from one version to another. In practice, users and developers have to manually look at the APIs to identify which functions are new and which types have changed.

2 Objective of the Internship

Despite its overall complexity, typical uses of module are quite simple, as seen in Figure 1a.

Consequently, most module type errors have simple cause. For instance many errors appear during code refactoring, where a handful of changes are common such as adding a new function, removing an old function, renaming

```
module Car : sig
  type t
  val color : t -> color
  val date : t -> date
end
module Catalogue :
  Set.S with type elt = Car.t

module Car = struct
  type t = { brand : ... ; model : ... }
  let compare car1 car2 = ....
  let color car = ...
  let date car = ...
end
module Catalogue =
  Map.Make(Car)
```

(a) A module Car and its (wrong) interface

```
Error: Signature mismatch:
  The type `elt' is required but not provided
  Expected declaration:
   `inter' is required but not provided
   `disjoint' is required but not provided
   `diff' is required but not provided
   ...
   <more than 20 lines after>
   ...
```

(b) The error message, elided

```
Error: Signature mismatch:
  This module has type
    Map.S
  but
    Set.S
  was expected
```

(c) The desired error message

FIGURE 1 – Modules in practice

LIP – UMR CNRS / ENS Lyon / UCB Lyon 1 / Inria – 69007 Lyon // Inria Paris
E-mail : gabriel.radanne@inria.fr // florian.angeletti@inria.fr

or changing the type of a function. Nevertheless, error messages even for such simple errors can be very large (often printing the whole module, which can have hundreds of definitions). For instance, in the Car module above, the Map was used instead of Set, resulting in a very large error message partially shown in Figure 1b. Tracking the actual mismatched definition is then very difficult for the programmer. Ideally, we would obtain the simpler message shown in Figure 1c.

This combination of theoretically complex module type errors, simple common use cases of modules, and a small class of common high-level errors makes a good argument for trying to give users a higher-level view in module-level type error messages. During this internship, we will try to use the tree-like shape of module types to leverage *diffing algorithm* on trees and lists.

Diffing compares two structures (lists, strings, ...) and summarizes the differences between them. For instance, on a string, it could determine that a new word was inserted, and where. The concept of diffing is known to most programmers as the concept powering code versioning systems such as git [1], but it is also used in varied domains such as spellchecking and bioinformatics [3]. Diffing can also be used on richer structures. Notably, diffing for trees [2] has found a large application in Web programming for UIs [4]. Tree diffing, in particular, is very appropriate to identify and track declarations in modules.

The goal of this internship will be to investigate how to improve error messages and documentation for modules. We will start by using diffing techniques for error messages, and then investigate their use on versioning documentation. In all cases, we will develop algorithms based on diffing, coupled with type-system considerations to ensure that the error messages are correct.

This internship aims to both develop a solid theoretical understanding of the problem and its solution, but also to apply and implement the techniques in the OCaml compiler and tools.

3 Profile

The candidate should be familiar with formal approach to programming languages, and have a taste for developing and implement new algorithmic solutions.

On the practical side, an experience in software development is welcome, along with a minimal knowledge of collaborative tools such as `git`.

Detailed knowledge of the OCaml module system is not required, but it is strictly necessary to know how to program in OCaml for the internship.

4 Contact and Localisation

The internship will take place in Lyon. Contact Gabriel Radanne (gabriel.radanne@inria.fr) and Florian Angeletti (florian.angeletti@inria.fr) if you are interested by this internship.

Références

[1] Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In Pablo de la Fuente, editor, *Seventh International Symposium on String Processing and Information Retrieval, SPIRE 2000, A Coruña, Spain, September 27-29, 2000*, pages 39–48. IEEE Computer Society, 2000. doi : 10.1109/SPIRE.2000.878178. URL https://doi.org/10.1109/SPIRE.2000.878178.

[2] Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3) :217–239, 2005. doi : 10.1016/j.tcs.2004.12.030. URL https://doi.org/10.1016/j.tcs.2004.12.030.

[3] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1) :31–88, 2001. doi : 10.1145/375360.375365. URL https://doi.org/10.1145/375360.375365.

[4] The ReactJS team. Reconciliation in reactjs, 2020. URL https://reactjs.org/docs/reconciliation.html.