Generic Programming in OCaml[‡]

Florent Balestrieri Michel Mauny

U2IS, ENSTA-ParisTech, Université Paris-Saclay,
828 boulevard des Maréchaux, 91762 Palaiseau Cedex

We present a library for generic programming in OCaml, adapting some techniques borrowed from other functional languages. The library makes use of three recent additions to OCaml: generalised abstract datatypes are essential to reflect types, extensible variants allow this reflection to be open for new additions, and extension points (PPX) provide syntactic sugar and generate boiler plate code that simplify the use of the library. The building blocks of the library can be used to support many approaches to generic programming through the concept of view. We present an application of the library to provide type-safe alternatives to OCaml's unsafe deserialization and unsafe type-cast.

Why Generic Programming?

Although the strong type system of OCaml is an asset when considering the safety properties that it ensures, it can also be perceived as a frustrating barrier that prevents us to generalise over some programming patterns, leading to a lot of boilerplate code and duplicated logic.

For instance, there is no convenient way to define a generic equality in OCaml. This operation being so common, a built-in generic equality operator is shipped as standard with OCaml. A fixed set of other generic operations are available in OCaml, yet they raise some issues:

- They break type abstraction;
- They sometimes even break type safety;
- We cannot easily define additional generic operations.

Our library

To address the above issues, we designed a library for generic programming in OCaml. Unlike the built-in generic operations, our library is strongly typed and therefore preserves type abstraction and type safety.

[*]This work was partially supported by the ANRT through the SecurOCaml project.
[†]Submitted for Presentation at OCaml 2016

Type Reflection. Like parametric polymorphic functions, generic functions work on all types. Unlike parametric polymorphic functions, generic functions may inspect the values that they process. For this to be possible, generic functions are structurally defined on the types. In OCaml, this is now possible using generalised abstract datatypes (GADTs). We define a type α ty whose values reflect the type parameter α:

```
type α ty =
  | Int : int ty
  | String : string ty
  | Pair : α ty × β ty → (α,β) ty
  | List : α ty → α list ty
```

The value reflecting the type (int, string) list is List (Pair (Int, String)) and it has type (int, string) list ty. We say the values of type α ty are type codes.

Generic functions are polymorphic functions parameterised by a type code, we call them type-indexed functions. For instance the deseralization function and generic equality may be typed as follows:

```
val deserialize : α ty → string → α
val equal : α ty → α → α → bool
```

As new types may be added by the user, type reflection needs to be extensible, and so does type-indexed functions. Whereas extensible types have been available since OCaml version 4.02, there is still no language support for extensible functions. We therefore provide a library implementation of extensible functions using PPX for syntax sugar.

Generic Views. Whereas type codes are the value level reflection of type terms, generic views reflect the structure of types. For instance, a common generic view is the sum of product view which can describe all variant types. Generic functions will usually be defined by structural recursion over a generic view.

Syntactic Support. The library requires specific work when new types are introduced, if we wish that the generic operations work on them. Thanks to extension points (PPX), we may free the programmer from this systematic task. We also provide cleaner syntax for writing extensible type-indexed functions. Finally, PPX may also be used to fill in automatically the type code arguments of generic functions.

Concrete applications. The library has been applied to deserialization, as part of the project SecurOCaml. The generic function is particular in that it works with the underlying memory representation of OCaml values, just like the standard (de-)serialization functions from the module `Marshal`. However, unlike them, the algorithm is recursively defined on the type representation, which not only provides type safety, but also correct support of abstract datatype.

Availability The library is licensed as open source software. A pre-release is expected at the end of July 2016, at `https://github.com/OCamlPro/SecurOCaml`.