Tyre – Typed Regular Expressions

Gabriel RADANNE
Univ Paris Diderot, Sorbonne Paris Cité
gabriel.radanne@irif.fr

Tyre is a set of combinators to build type-safe regular expressions, allowing automatic extraction and modification of matched groups. Tyre is bi-directional: a typed regular expression can be used for parsing and unparsing. It also allows routing, by providing a list of regexes/routes and their handlers.

A common problem when considering inputs of programs is to interpret and validate such inputs. Parsing libraries, either based on parser generators such as menhir [Pottier and Régis-Gianas] or combinators such as Re [Vouillon] or Angstrom [Eliopoulos], only interpret (textual) data in term of OCaml data structures but do not validate them. Such validation is left as a burden to the programmer and is often prone to mistakes. Indeed, most of the objects manipulated are strings, which limit the usefulness of the type system. Furthermore, the code doing the parsing and the code doing the validation are distinct, which increases again the possibility of error.

Tyre [Radanne], for Typed Regular Expression, aims to solve this problem in the context of regular languages. It can be seen as a typed overlay for Re [Vouillon], which is an efficient Regular Expression library in pure OCaml. Regular languages provide a convenient setting for typed combinators: everything is decidable, regular expression lend themselves very well to an applicative-style DSL and efficiency can be retained by using a compilation phase from regular expressions to automatons.

Tyre leverages these properties to provide type-safe extraction, but also unparsing and routing (where the programmer provides a list of regular expressions and corresponding handlers). Basic usage of the library is demonstrated in Figure 1. We first define a regular expression `raw_dim` which parses text of the form `"dim:123x456"`. The combinators <&> and *> allow to compose primitive regular expressions such as `str"dim"` (which always parses `"dim"`) and `int` (which parses an arbitrary integer). Note the type of the regular expression, which properly reflects that we are only interested in the two integers and do not care about the constant strings. We then define a record type `dim` and a new regular expression, `nice_dim`, which wraps `raw_dim` with the `Tyre.conv` combinator, in order to only present the (nicer) record type. We then compile and use this regular expression, both for parsing and for printing.

During the workshop, we will present the Tyre library using a practical use case: parsing of URLs in `ocaml-uri`. We will also present some internal details, such as how we can use GADTs to build a witness that allow to reconstruct typed data from the untyped results provided by Re and the extensions that were needed in Re to make Tyre possible.

```
# let raw_dim = Tyre.( str"dim:" *> int <&> str"x" *> int ) ;;
val dim : (int * int) Tyre.t

# type dim = { x : int ; y : int }
# let nice_dim : dim Tyre.t =
    Tyre.conv (fun (x,y) -> {x;y}) (fun {x;y} -> (x,y)) raw_dim ;;
val nice_dim : dim Tyre.t

# let dim_re = Tyre.compile nice_dim ;;
val dim_re : dim Tyre.re

# Tyre.exec dim_re "dim:3x4" ;;
- : (dim, dim Tyre.error) result = Result.Ok {x = 3; y = 4}

# Tyre.eval nice_dim {x=2;y=5} ;;
- : string = "dim:2x5"
```

Figure 1: Small example using Tyre

References

S. Eliopoulos. Angstrom. URL https://github.com/inhabitedtype/angstrom.

F. Pottier and Y. Régis-Gianas. Menhir. URL http://gallium.inria.fr/~fpottier/menhir/.

G. Radanne. Tyre. URL https://github.com/Drup/tyre.

J. Vouillon. Re. URL https://github.com/ocaml/re.