Merlin, an assistant for editing Ocaml code

Frédéric Bour, Thomas Refis & Simon Castellan

June 16, 2013

1 Introduction

Merlin is a tool designed to provide assistance to programmers editing Ocaml code.

By offering features such as completion and interactive error reporting, it aims at improving the coding experience, narrowing the gap between editors and “modern IDEs”.

1.1 Comparing with existing solutions

Other tools are developped with similar goals, most notably *Typerex* and *Camlspotter*, though these don’t directly overlap with *Merlin*.

Whereas those tools work on files produced by the compiler (“cmt” files), *Merlin* address the particular problem of working directly on the editor buffer. It should even be possible to make those collaborate: by relying on *Merlin* to analyse and extract information from the buffer then invoking other tools.

In this respect, *Merlin* can be seen as a collection of heuristics allowing to incrementally parse and typecheck incomplete and/or partially incorrect codes.

2 General design

Starting with a demonstration, the presentation will focus on Merlin’s architecture, describing the steps needed to get from the content in the editor to showing back the result of the analysis.

Largely reusing the OCaml compiler, it follows the traditional pipeline of a front-end, lexer / parser / type checker, though somewhat tweaked to work in an incremental context.

2.1 Mimicking Proof-Assistants

The ocaml file is split in chunks that can be processed separately. These checkpoints are located at each definition, when entering and leaving a structure (ignoring first class modules) and more generally between all toplevel constructions found in ML modules.

This workflow comes from *Proof-Assistants* which have to deal with very complex contexts. Immediate feedbacks are of great help to the developer that can query the prover at some determined points.

Here, the parsing is done in two passes: a first one to extract the potential checkpoints and isolate syntactically incorrect parts, then a classic one is invoked to generate an AST from extracted parts.

2.2 Cornercases

As the syntax of Ocaml has not been designed with such use in mind, particular precautions have to be taken. Notably, the Ocaml parser has been heavily modified to better handle error recovery.

Also, a few parts of the typer had to be patched to control some of its side effects and relax some rules to ease interactive use.

2.3 Integration with editors

The merlin process communicates via a stream of JSON objects. Though still subject to modifications, the protocol has been designed to make merlin easy to integrate in “dumb editors” without leaking implementation details.

The goal of this part would be to show potential users that *Merlin* could be ported to different editors (e.g. *Eclipse*) or tailored to suit particular needs with a reasonable amount of work.

3 Current limitations

Most limitations come from the parsing step.

- Syntactic errors are particularly hard to deal with with current parser generators. The original grammar had to be tweaked to handle more gracefully common errors, at the cost of hampering support for future versions.
- CamlP4 syntax extensions are not supported, though some have been reimplemented.
- However, the future looks brighter with ppx extensions. With a few restrictions on the extension behavior, it should be possible to transparently integrate this step in the pipeline.

4 Roadmap

Points to be adressed in the future:

- on the parsing side, improve support for recursive definitions, classes and objects and first-class modules.
- on the typing side, make use of contextual information like “is the cursor inside a pattern, a type or an expression?” to direct analysis.
- try to implement more exotic extensions, like *Eliom.*