Conclusion

Although computer science has become an industrial activity, in many respects the success of a programming language is a subjective affair. If “the heart has its reasons of which reason knows nothing,” then Objective Caml is a reasonable choice for a lover of heart.

It is based on solid theoretical foundations, all while providing a wide spectrum of programming paradigms. If one adds the simplicity of interaction with the language which the toplevel supports, that makes it a language perfectly adapted for teaching.

- Structured types and abstract types support approaching algorithmic problems and their complex data structures, all while abstracting away from problems of memory representation and allocation.
- The functional theoretical model underlying the language supplies a precise introduction to the notions of evaluation and typing which, as a “true programmer”, one owes it to oneself to be taught.
- The various programming models can be approached independently of one another: from modular or object-oriented program structure to low-level systems programming, there are few areas where Objective Caml is not useful.
- Its suitability for symbolic programming makes it an excellent support for theoretical courses such as compiling or artifical intelligence.

For these qualities, Objective Caml is often used as the basis of the introductory computer science curriculum as well as for advanced programming courses which make explicit the link between the language’s high level of abstraction and its execution. Many teachers have been and remain seduced by the pedagogical advantages of Objective Caml and, by way of consequence, many computer scientists have been schooled in it.

One of the first causes for satisfaction in Objective Caml development is how comfortable it is to use. The compiler loads rapidly and its static type inference lets nothing escape. Other static analyses of the code give the programmer precious indices of anomalies if not errors: incomplete pattern-matching is signaled, partial application of a function in a sequence is detected, etc. To this first cause of satisfaction is added a second: the compiler very rapidly generates efficient code.

Compiler performance, conciseness of expression of functional programming, quality and diversity of libraries make Objective Caml a language perfectly adapted to the needs of “disposable software”. But it would be diminishing it to restrict it to this single application domain. For these same reasons, Objective Caml is a precious tool for experimentation and application prototyping. Moreover, when the structuring mechanisms of modules and objects come to be added to the features already mentioned, the language opens the way to the conception and development of finished applications.

Finally, Objective Caml and its developer community form a milieu which reacts quickly to innovation in the area of programming. The free availability and the distribution of the source code of the language offer emerging concepts a terrain for experimentation.

Learning Objective Caml requires a certain effort from the programmer familiar with other languages. And this, as well as the object of study is in constant evolution. We hope that without masking the complexity of certain concepts, this book will facilitate this phase of learning and can thus accelerate the return on investment for the Objective Caml application developer.