

A Short History of Ocaml

Brian Hurt

A long time ago, in a computer room far, far away...

Back in the 1940's, the mathematician Alonzo Church (possibly inspired by ideas in Gödel's Incompleteness Theorem) proposed the theory of Lambda Calculus, as an alternative basis for computational theory to Turing's Machine. Now, Lambda Calculus has about as much to do with learning Ocaml as Turing Machines have to do with learning Java (i.e. nothing at all), but Lambda Calculus and Fortran- which originally stood for FORMula TRANslator, and was intended to allow scientists and engineers to simply write formulas, and not code- combined in the mind of John McCarthy to form Lisp, in about 1958.

Lisp was originally intended to be just a theoretical exercise, and McCarthy was as surprised as anyone when one of his grad students, Steve Russell, actually implemented the language. The language escaped into the wild, and real programmers started using this theoretical language for real problems. To this day, a large, active, thriving community of Lisp developers exists the world around.

But the invention of Lisp raised an interesting question. The core idea of Lisp was that you wrote functions- abstract equations with no side effects, rather like mathematics. It turned out that this style of code (which got termed "functional", as opposed to what Fortran, Pascal, C, Java, etc. all were, which was "imperative") was incredibly easy to reason about- so easy you could write programs to reason about the code for you. Could you actually *prove* the code was correct, or at least that it didn't contain any instances of this wide class of bugs? What were the limits?

In 1971, Robin Milner started the LCF project- the Logic of Computable Functions. This was an investigation into automatic theorem proving about code. The theoretical language LCF worked on he developed in 1973, and dubbed "Meta-Language" aka ML. ML had the same easy to analyze functional structure of Lisp, but with a more natural, mathematics-like notation heavily influenced by Pascal. Once again, this theoretical language escaped into the wild, and once again real programmers proceeded to write real code in ML. It's decedents are still in active use and development, under the name Standard ML or SML. Are you noticing a trend here?

One of the places where ML was being used for "real" work was in the Formel Project in the Institut National de Recherche en Informatique et en Automatique (National Institute for Research into Information and Automation- or INRIA) in France. Now, INRIA was doing computer science research into language design, but they needed a pragmatic, workaday language to actually build stuff in. Up until about 1987, the Formel project had been using ML, but ML had a number of short comings and mistakes. You never get things perfect the first time around, and ML was no exception. So Guy Cousineau wrote a cleaned-up version of ML, fixing most of the obvious flaws, and built it on top of the Categorical Abstract Machine (basically Lisp), or CAM. As a pun, CAM-ML got shortened to CAML, when it was released in 1987.

In 1991, Xavier Leroy and Damien Doligez wrote Caml-light, CAML compiling directly to and running on a virtual machine, instead of compiling to and running on the CAM. This was faster and turned out to be more portable. In 1994, development of CAML officially passed from the Formel project to Project Cristal. Xavier Leroy, Jérôme Vouillon, and Didier Rémy added Objects and the ability to compile to native code in 1996. The rest, as they say, is history.

