The core Caml system: status report and challenges

Xavier Leroy

INRIA Paris-Rocquencourt, projet Gallium

Caml User Group meeting, 2008-01-26

Outline

1. Caml development news
2. Some non-technical challenges
3. Some technical challenges

Outline

Latest major release: OCaml 3.10

Main novelties:

- Major overhaul of the Camlp4 preprocessor and pretty-printer.
- New tool: the `ocamlbuild` compilation manager.
- Introduction of virtual instance variables in classes.
  (Like methods, these instance variables can be defined and re-defined in sub-classes.)
- Printing of stack backtraces on uncaught exceptions extended to native-code programs.
- New ports: Windows 64; MacOS X / PowerPC 64-bit.

Released in May 2007. Bug-fix releases: 3.10.1 in January 2008, 3.10.2 soon.

Next major release: OCaml 3.11

Planned for Q3 2008.

- Dynamic loading of native code.
- Performance improvements in thread synchronization.
- Being discussed: small extensions of coercions
  $(e : \tau_1 >: \tau_2)$ and $(e >: \tau_2)$
- Being discussed: private type abbreviations
  $\texttt{type}\ t = \texttt{private}\ \tau$
- If time and availability permit, ARM/MacOS port (= iPhone).

Status of the Caml Consortium

In 2007, went from 4 to 7 members:

Dassault Aviation	Intel
Dassault Systèmes	Jane Street Capital
Lexifi	XenSource
Microsoft	

What the Consortium does:

- Sell permissive licensing conditions on the Caml code base.
- Enable lightweight corporate sponsoring.
- A place to discuss needs with some industrial users.

What the Consortium does not:

- Bring enough funds to pay for a full-time developer.
- Anything else.

Outline

Challenge 1: Manpower

"Why doesn't INRIA do XXX?"

Currently, the Caml development team is equivalent to **0.5** person full time.

We all have more cutting-edge research projects that we are supposed to work on full time.

What can this community do about it?

Refrain from unreasonable demands.

Help with various time-consuming tasks:

- Testing (esp. of release candidates).
- Contribute more to the Windows port.
- Take over the construction and distribution of binary releases for Windows and MacOS.
- Keep up the good work with bug reports.

Start and organize initiatives; don't wait for us to do it.

Apply for the positions of *ingénieur de recherche* that INRIA Paris-Rocquencourt is opening this year.

Challenge 2: The core system vs. the rest

"Why isn't XXX included in the `.tar.gz` from INRIA?"

By experience, anything we add to the INRIA distribution:

- ends up being maintained by us, forever;
- must be copyright INRIA (dual-licensing);
- slows down our already sluggish release cycles.

Ideally, the INRIA distribution should only contain a core Caml system:

- compilers and run-time system;
- libraries and tools intimately tied to them.

(Plus historical baggage.)

What can this community do about it?

Organize around a "packages and distribution" model, a la Linux, CPAN, etc.

Support existing efforts in this direction: GODI, the Debian Caml packages, etc.

Try to converge towards a single distribution effort in the style of CPAN, even if it is less sophisticated than the above.

Challenge 3: Backward compatibility

"This behavior is weird, why don't we do XXX instead?"

Incompatible changes to the language or its implementation just don't go down well with users.

There are very good reasons for backward compatibility, but it precludes incremental fixing of past design errors, or even any evolution other than feature accretion.

Outline

Challenge 4: More type system features

A great many features could be added to the Caml type system. Many are just not worth the increased complexity.

One that scores highest on practical interest: GADT (Generalized Algebraic Data Types).

```
type 'a ty =
  | Int                          constraint 'a = int
  | Pair of 'b ty * 'c ty        constraint 'a = 'b * 'c

let rec print : forall 'a. 'a ty -> 'a -> unit =
  fun t x ->
    match t with
    | Int -> print_int x
    | Pair(t1, t2) -> print t1 (fst x); print t2 (snd x)
```

The theory is done, but ...

Type system issue: The type inference engine

The current core language type-checker / type inference engine is reaching its limits:

- Design limitations.
- Incremental extensions piled on top of one another.
- Excessively imperative implementation.

Would need to be reimplemented from scratch based on more modern, constraint-based approaches.

Challenge 5: parallelism

- Today: small shared-memory multi-core processors.
- Tomorrow: more cores, distributed memory.
- As always: clusters.

Parallelism, issue 1: The programming model

What is a good programming model for parallelism?

From worst to best:

- Shared memory, locks, condition variables.
  (Java.)
- Same, plus static type-checking of locking.
  (Abadi & Flanagan.)
- Software transactional shared memory.
  (Glasgow Haskell.)
- Message passing.
  (Erlang, JoCaml.)

Parallelism, issue 2: The run-time system

From simplest to most complex:

- No shared heap, inter-process communications.
  (Zheng Li's coThreads library. No change required to the RTS.)
- No shared heap, intra-process communications when possible.
  (1/3 of the RTS to be rewritten.)
- Shared heap with concurrent GC.
  (2/3 of the RTS to be rewritten; algorithmic issues with weakly consistent memory.)

Parallelism, issue 3: Encapsulation of effects

To have clean semantics for parallel computations, need to statically control or eliminate entirely shared mutable data structures.

- Shared memory model: must guarantee synchronization (locking).
- Transactional memory: all globally-visible mutations must be within a transaction.
- Message passing: global mutable data or passing of mutable data in messages makes no sense.