The state of OCaml, 2012

Xavier Leroy

INRIA Paris-Rocquencourt

OCaml Users and Developers Workshop, 2012-09-14
Outline

1. OCaml development news

2. OCaml community news
The new major release: OCaml 4.00

- Objects and classes
- Classes, take 2
- Polymorphic variants
- Labeled arguments
- Polymorphic record fields
- Recursive modules
- Lazy patterns
- First-class modules
- GADTs

X. Leroy (INRIA)

The state of OCaml, 2012
What’s new

Language features:
- Generalized Algebraic Data Types
- More lightweight support for modules packed as first-class values.

Implementation features:
- Exposing rich typed ASTs and compiler internals (for IDEs and more)
- Lots of new warnings
- Revamped ARM code generator
- Improvements in marshaling and generic hashing.

Development process:
- More external contributions
- 160 issues fixed, 50 feature wishes granted
- More rigorous (but slow) release process.
A (seemingly minor) extension to the declaration of variant data types that enables programmers to

1 express properties of data structures via type equalities;
2 have the type-checker enforce these properties.
Without GADTs: tagged interpreters

type expr =
  | Lit of string
  | Pair of expr * expr
  | Fst of expr
  | Snd of expr

and value = (* results of evaluation *)
  | VString of string (* “tagged” with their types *)
  | VPair of value * value

let rec eval : expr -> value = function (* produces a tagged value *)
  | Lit s -> VString s
  | Pair(e1, e2) -> VPair(eval e1, eval e2)
  | Fst e1 ->
    (match eval e1 with VPair(v1, v2) -> v1 | _ -> raise Error)
  | Snd e1 ->
    (match eval e1 with VPair(v1, v2) -> v2 | _ -> raise Error)

(* dynamic typing during evaluation *)
With GADTs: tagless interpreters

Can define $\tau$ expr as the type of symbolic expressions that safely evaluate to a Caml value of type $\tau$.

```ocaml
type _ expr =
  | Lit: string -> string expr
  | Pair: 'a expr * 'b expr -> ('a * 'b) expr
  | Fst: ('a * 'b) expr -> 'a expr
  | Snd: ('a * 'b) expr -> 'b expr
```

The evaluator, then, needs not tag result values, and cannot fail.

```ocaml
let rec eval : type v. v expr -> v = function
  | Lit s -> s (* v = string here *)
  | Pair(e1, e2) -> (eval e1, eval e2) (* v = v1 * v2 here *)
  | Fst e -> fst (eval e) (* statically safe *)
  | Snd e -> snd (eval e)
```
A new compiler option, `--bin-annot`, causing the production of a `.cmt` file containing a rich Abstract Syntax Tree annotated with:

- Source file locations
- Scoping and binding information for identifiers
- Types inferred by the typechecker.

(Generalizes the `--annot` option, which generated only a subset of this information, in an Emacs-specific format.)
The OCaml compilation chain, before 4.00

**Parsetree:** (produced by the parser)
- Very close to source text
- Annotated by source locations (file name, line #, column #)
- No types, no scoping information

**Typedtree:** (produced by the typechecker)
- Annotated by (inferred) types
- Explicit scoping and binding of idents
- Some source constructs eliminated (open, include, type constraints)
- No source locations
- All source constructs represented
- Same location info as in Parsetree
The OCaml compilation chain, in 4.00

Source
  Parsetree
    Typedtree
      .cmt file
      Lambda
        Bytecode
          ULambda
            C--
              Mach
                  Linear
                      Assembly

**Parsetree:** (produced by the parser)
- Very close to source text
- Annotated by source locations
  (file name, line #, column #)
- No types, no scoping information

**Typedtree:** (produced by the typechecker)
- Annotated by (inferred) types
- Explicit scoping and binding of idents
- Some source constructs eliminated
  (open, include, type constraints)
- No source locations
- All source constructs represented
- Same location info as in Parsetree
Using typed ASTs

**Current use: for IDEs** (e.g. TypeRex, OCamlSpotter)
show inferred types; jump to definition; scoping-aware identifier renaming; type-aware completion; etc.

**Possible future use: for code generation**
Camlp4-style preprocessing that has access to type & scope info.

**Caveat:** currently, no stable API to work on typed ASTs; must use compiler internal modules.
Using typed ASTs

**Current use: for IDEs** (e.g. TypeRex, OCamlSpotter)
show inferred types; jump to definition; scoping-aware identifier renaming; type-aware completion; etc.

**Possible future use: for code generation**
Camlp4-style preprocessing that has access to type & scope info.

![Diagram showing the relationship between Parsetree, Typedtree, Camlp*, and ???]

**Caveat:** currently, no stable API to work on typed ASTs; must use compiler internal modules.
This release brought to you by . . .

(More help is welcome.)
Future directions

A bug-fix release 4.00.1 this Fall.

Work in progress on:
- Name space management
- Run-time representations of types.
- Performance improvements (native compiler, run-time system).

Shedding more weight off the core system.
(By splitting off some libraries and tools as independent projects.)
News from the community
(not exhaustive)

Some new or recently open-sourced projects:

- TypeRex (OCamlPro’s IDE)
- OPAM (OCamlPro’s package manager)
- Opa (MLstate’s Web programming framework)
- Mirage (OCaml as a Xen guest OS)
- JS-of-OCaml (OCaml running in any browser)
- Functory and Parmap (parallel computation)
- ZArith (arbitrary-precision integers)
- Async (Jane Street’s lightweight cooperative threads)
News from the community
(not exhaustive)

New releases of major libraries and frameworks, such as:

- Batteries and Core (comprehensive standard libraries)
- Frama-C (static analysis framework)
- Ocsigen (Web programming framework)
- OCaml “companion tools”
- ODT (Eclipse plug-in)
- OUnit (unit testing framework)
- Plasma (distributed file system and map-reduce)
News from the community
(not exhaustive)

Cool factor:

- OCaml iPhone/iPad apps (psellos.com, M. Hayden, J. Kimmit)
- TryOCaml (toplevel in browser)

Textbooks:

- *Real-World OCaml* (J. Hickey, A. Madhavepeddy, Y. Minsky) (soon?)
- *Think OCaml: How to Think Like a Computer Scientist* (N. Monje and A. Downey)
- *Programmation de droite à gauche (et vice-versa)* (P. Manoury)
In closing...  

A lively language; a lively implementation; a very lively community.

Some growing pains.

Many individual contributions, deserve better integration & accessibility.

High hopes in a future OCaml Platform.