What's new in OCaml 4.02

Xavier Leroy

INRIA Paris-Rocquencourt

OCaml Workshop, 2014-09-05

Recent releases

Major release 4.01.0: (Sept. 2013)

- More lenient typing of ambiguous record labels and data constructors.
- Usability features:
  - `-short-path` option to print inferred types more legibly.
  - A lot of new warnings.
  - Suggested corrections for misspelt identifiers.
- 135 bug fixes, 25 feature wishes.

Major release 4.02.0: (Sept. 2014)

- A great many new features!
- 66 bug fixes, 18 feature wishes.

Unified matching on values and exceptions

A classic programming problem: in `let` $x = a$ `in` b, catch exceptions raised by a but not those raised by b.

Wrong solution: put a `try...with` around `let`.

```
let rec readfile ic accu =
  try let l = input_line ic in readfile ic (l :: accu)
  with End_of_file -> List.rev accu
```

The `try` covers the recursive call to `readfile` and prevents it from being a tail call.

Option-based encoding: correct but heavy.

```
let rec readfile ic accu =
  match
    try Some(input_line ic) with End_of_file -> None
  with
  | Some l -> readfile ic (l :: accu)
  | None   -> List.rev accu
```

Unified matching on values and exceptions

(Jeremy Yallop, Alain Frisch)

In OCaml 4.02: the `match` *a* `with` construct is extended to analyse both the value of *a* and exceptions raised during *a*'s evaluation.

```
let rec readfile ic accu =
  match input_line ic with
  | "" -> readfile ic accu
  | l  -> readfile ic (l :: accu)
  | exception End_of_file -> List.rev accu
```

Compilation preserves tail-call optimization for the recursive call. No allocation of intermediate Some results.

The usual `try...with` construct is a special case:

```
try ... with Not_found -> 0
                ≡
match ... with x -> x | exception Not_found -> 0
```

Module aliases

Common practice: bind an existing module to another name.

```
module A = struct
  module M = AnotherModule
  ...
end
```

- To have short qualified names `M.x` inside the body of `A`.
- To re-export a module or compilation unit as a submodule of another.

Module aliases

```
module A = struct
  module M = AnotherModule
  ...
end
```

In classic OCaml, this binding is treated like any other definition `module M =` *module-expr*

- Typing: the equality `A.M` = `AnotherModule` is not recorded, can cause type errors with applicative functors.
- Compilation: accesses `A.M.x` sometimes go through more indirections than `AnotherModule.x`.
- Linking: a reference to `A.M.x` links in the whole of module `A`, not just `AnotherModule`.

Module aliases in OCaml 4.02

(Jacques Garrigue, Leo White)

In 4.02, `module M = AnotherModule` is treated specially:

- During type-checking: the equality is recorded in the signature of the enclosing module

```
A : sig module M = AnotherModule ... end
```

- During compilation: direct accesses to `AnotherModule` are generated.
- During linking: an access `A.M.x` no longer forces `A` to be linked in full, but only `AnotherModule`.

Application to libraries

Goal: expose (to users) a library as a single unit `L` with submodules `L.A`, `L.B`, etc.

Approach 1: single source

Write and compile `L` as a single source file `L.ml`.

Problem: does not scale.

Approach 2: packing

Put `A`, `B` in distinct source files `A.ml`, `B.ml`, separately compiled.

Use `ocamlopt -pack -o L.cmx` to produce the single unit `L`.

Problem: `L` is always linked in full.

Application to libraries

New approach: use module aliases

Put `A`, `B` in distinct source files `L_A.ml`, `L_B.ml`, separately compiled.

(Note: the names `L_A`, `L_B` must be globally unique.)

Write and compile a wrapper `L.ml` that re-exports them with short names:

```
L.ml:   module A = L_A
        module B = L_B
```

Build and install an archive `L.cmxa` containing `L_A`, `L_B`, and `L`.

Mutability of strings

For historical reasons, the `string` type of OCaml is mutable in place and has two broad uses:

- To represent text:

```
let string_of_pair (x,y) = "(" ^ x ^ ", " ^ y ^")"
```

- As I/O buffers:

```
let read_data ic n =
  let buf = String.create n in
  let len = Unix.read ic buf 0 n in
  String.sub buf 0 len
```

OCaml programmers are very good at keeping the two uses distinct.

But mistakes happen, and user code can be malicious...

Pitfalls of mutable strings

Mutating someone else's string literals: (Lafosec study, ANSSI)

```
# (Char.escaped '\t').[1] <- 'n';;
- : unit = ()
# Char.escaped '\t';;
- : string = "\\n"

# String.blit "true " 0 (string_of_bool false) 0 5;;
- : unit = ()
# string_of_bool false;;
- : string = "true "
```

The disappearing key:

```
let s = "foo" in
Hashtbl.add tbl s 10;
s.[1] <- 'x';
Hashtbl.find tbl "foo"
```

Pitfalls of mutable strings

LCF-style theorem proving:

```
module ProofSystem : sig
  type thm
  val modusponens: thm -> thm -> thm
  val axiom_eqrefl: term -> thm
  val axiom_and: formula -> formula -> formula
  ...
end
```

> *By subverting the OCaml type system or using mutability of strings, it is possible to derive false results even in our LCF prover, but we restrict ourselves to 'normal' functional programming.*
>
> *J. Harrison*, Handbook of Practical Logic and Automated Reasoning

Towards immutable strings (Damien Doligez)

In 4.02, two base types `string` (for text) and `bytes` ($\approx$ byte arrays):

- By default, `string` and `bytes` are synonymous.
- They are incompatible if option `-safe-string` is given.

In standard library `String`, functions that mutate strings are given types involving `bytes` and marked as deprecated:

```
string.mli:    val set : bytes -> int -> char -> unit
                  [@@ocaml.deprecated]
```

A new standard library module `Bytes` is provided with a full set of imperative functions operating over `bytes`, including conversion functions (with copy):

```
bytes.mli:     val set : bytes -> int -> char -> unit
               val of_string: string -> bytes
               val to_string: bytes -> string
```

What changes?

In default mode:

- All existing code compiles.
- Warnings when using `String` functions that mutate in place.

In `-safe-string` mode:

- Values of type `string` are immutable.
  (Unless unsafe coercions are used.)
- I/O code and imperative constructions of strings need to be rewritten to use type `bytes` and library functions `Bytes`.

A plea for developers of libraries: try to port your code to `-safe-string` mode and let us know how it worked.

Other novelties in OCaml 4.02

Language features:

- Explicitly generative functors. (Jacques Garrigue)
- A language of annotations over OCaml terms.
  Annotations are used by the compiler (e.g. to mark deprecated functions) and by preprocessors. (Alain Frisch)
- External preprocessors that rewrite the typed abstract syntax tree (`-ppx` option). (Alain Frisch)
- Open datatypes, extensible a posteriori with additional constructors. (Leo White)

Other novelties in OCaml 4.02

Performance improvements:

- More aggressive constant propagation; dead code elimination; common subexpression elimination. (Xavier Leroy)
- More efficient pattern-matching over strings.
  (Benoît Vaugon, Luc Maranget)
- More efficient and safer implementation of `printf` based on GADTs.
  (Benoît Vaugon, Gabriel Scherer)
- More efficient representation of exceptions without arguments.
  (Alain Frisch)

Other novelties in OCaml 4.02

Usability:

- New toplevel directives to query the environment, e.g. `#show_type list` or `#show_module String`.
  (Jacques Garrigue, Alain Frisch)

New port:

- AArch64 (ARM in 64-bit mode).
  (Xavier Leroy, debugging by Richard Jones)

Source reorganization:

- Camlp4 and Labltk were split off the core distribution and now live as independent projects — one `opam install` away.

This release brought to you by...

Damien Doligez,
release manager and general wizard.

The core Caml development team: Jacques Garrigue, Alain Frisch, Fabrice Le Fessant, Xavier Leroy, Gabriel Scherer, Mark Shinwell, Leo White, Jeremy Yallop







With much appreciated contributions from: Anil Madhavapeddy, Benoît Vaugon, Daniel Bünzli, David Allsopp, Hongbo Zhang, Jacques-Henri Jourdan, Jacques-Pascal Deplaix, John Whitington, Jonathan Protzenko, Jun Furuse, Jérémie Dimino, Luc Maranget, Markus Mottl, Maxence Guesdon, Phil Denys, Pierre Chambart, Richard Jones, Romain Bardou, Stephen Dolan, Stéphane Glondu, Thomas Refis, Vladimir Brankov, ygrek.

What's next?

Bug-fix release 4.02.1.

Language features:

- Various experiments in progress; nothing decided yet.

Implementation:

- More optimizations, esp. improved inlining (Pierre Chambart).
- Ephemerons (François Bobot, Damien Doligez)
- GDB support (Mark Shinwell)
- Tweaks to support OPAM better.

In closing...

Enjoy OCaml 4.02!

Try to port your code to "safe string" mode: one day it might become the default.

OPAM-package your code and support the OCaml Platform!

Keep those contributions and this feedback flowing!