Nurturing OCaml day by day, part two

Florian Angeletti
Inria
florian.angeletti@inria.fr

1 June 2022

Releasing often and predictably

Target release cycle

3 months feature development phase

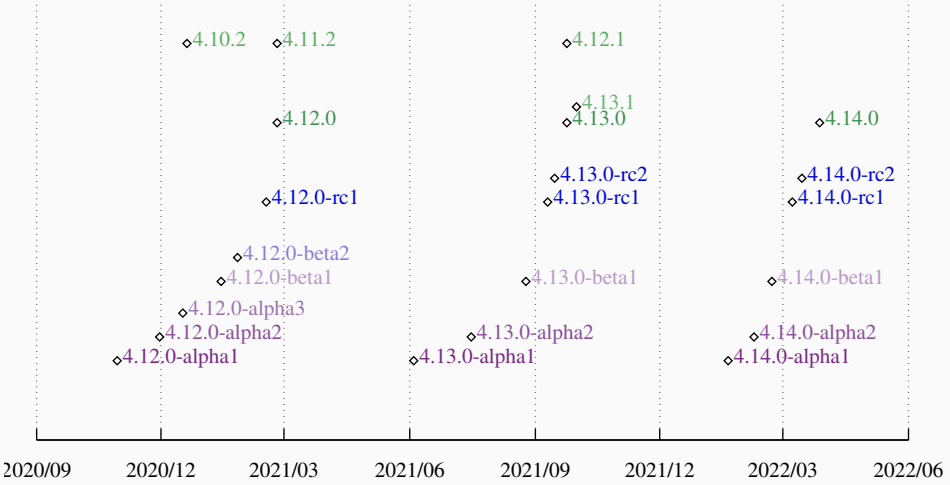
3 months Bug fixing

alpha releases stable features and IRs, no guarantee for users

beta releases stable internal API, ready for library writers

rc release opam ecosystem ready

Release rhythm



Opam ecosystem as an extended compiler test suite

Core idea
- Using opam ecosystem as CI as soon as a new version is branched
- require to patch core packages to explore more packages
- Collaboration with Kate Deplaix

Successes
- OCaml 4.14: computation time for new metadata exploded on functor heavy codebase like irmin, fixed before alpha
- Typechecker refactoring broke some recursive types: fixed in 4.14.0 rc2

Failures
- 4.13.0→4.13.1 : classes named “row” triggered an internal error
- 4.11.0→4.11.1: no more value restriction

Result
4.14.0 released with most opam packages ready

User experience

Improved error messages

- Error messages are the voice of the compiler during development
  - If an error message suggests a fix, it should be the right one
  - Error messages should be exhaustive

Higher-level error messages for functors and type definitions

```
Signature mismatch:
  ...
  Type declarations do not match:
    type t = A | B | D | C
  is not included in
    type t = A | B | C | D
 Constructors number 3 have different names, D and C.
```

(Work in collaboration with Gabriel Radanne)

Higher-level error messages for functors and type definitions

```
Signature mismatch:
  ...
  Type declarations do not match:
    type t = A | B | D | C
  is not included in
    type t = A | B | C | D
 Constructors C and D have been swapped.
```

(Work in collaboration with Gabriel Radanne)

Improved Merlin support

- Go-to definition is a complicated notion in OCaml
- Where is the Euler-Mascheroni constant defined in the code below?

```
module type S = sig val euler_mascheroni: float end
module F(X:S)(Y:S) = struct
  module A = Y module B = X
end
module G(X:sig module A:S end)=X.A
include G(F(Origin)(Origin))
```

- New metadata information for tracking the origin of definition across includes and functors applications. (Work by Thomas Refis, Ulysse Gérard; review by Gabriel Scherer, Nathanaëlle Courant and me)

Improved documentation

- Odoc-compatible generation of the documentation for the standard library
- New style for the online version of the manual by San Vũ Ngọc (reviewed by me)
- Many fixes in the latex code by Wiktor Kuchta (reviewed by me)
- Future work: integration with the new version of ocaml.org

Refactoring

Error traces extended

- Before OCaml 4.14.0, error traces at the module level did not explain type mismatches.

```
val f : y -> x
   is not included in
   val f : a -> b
```

- Refactoring work by Antal Spector-Zabusky (Janestreet) (reviewed by me)

Error traces extended

- Before OCaml 4.14.0, error traces at the module level did not explain type mismatches.

```
val f : y -> x
   is not included in
   val f : a -> b
   The type y -> x is not compatible with
     the type a -> b
   Type x = [ `C | `D ] is not compatible with
     type b = [ `C | `D | `E ]
   The first variant type does not allow tag(s) `E
```

- Refactoring work by Antal Spector-Zabusky (Janestreet) (reviewed by me)

Making the typechecker more approachable

Classes and class types

- Class and class types typechecking was full of implicit invariants
- Leo White refactored the code to be more explicit and more maintainable

Type expressions

- Types expressions are represented as mutable digraphs in the OCaml typechecker

```
type 'a t = < x: 'a > as 'a
```

- Manipulating those digraphs without breaking abstractions was an art form
- Jacques Garrigue and Takafumi Saikawa refactored the typechecker to add abstraction barriers to make it easier for non-expert to manipulate those digraphs (reviewed by Gabriel Radanne, Leo White, and me)

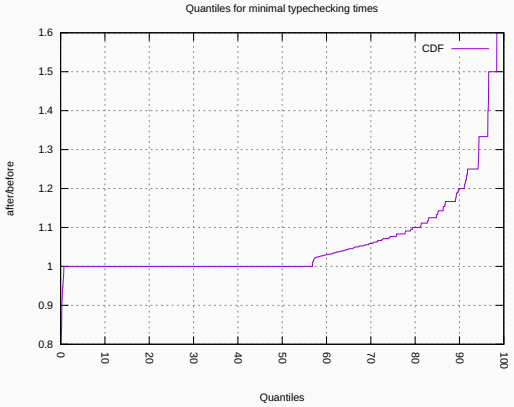
Monitoring compiler performance

Abstraction versus performance

- More abstraction for type expressions $\implies$ loss of optimization opportunity
- Does that matter? How do we know?
- Few tests on few programs $\implies$ statistical bias

Statistical monitoring

- Benchmarking compilation time with enough opam packages to avoid selection bias, with enough runs to collect statistical information



Long-term monitoring

- Compiler performance: detecting regressions, guiding design decisions
- Collaboration with Tarides (Arthur Wendling, Shakthi Kannan, Jon Ludlam, Jan Midtgaard, Puneeth Chaganti, Zineb Jambin): long-term monitoring with more data points both in term of monitored packages and historical data

OCaml 5 on the horizon

OCaml multicore

- Long-term effort since 2014 with major contribution from KC Sivaramakrishnan, Stephen Dolan, Enguerrand Decorne, Sadiq Jaffer, Sudha Parimala, David Allsopp, Leo White, and the rest of the Tarides multicore team
- Convergence between the OCaml experimental runtime and the main runtime started with OCaml 4.10
- Acceleration in OCaml 4.12 and OCaml 4.13
- OCaml multicore has been merged in the main OCaml branch on January 10
- OCaml core team has been working on understanding, stabilising, and documenting the new code.
- OCaml 5.0: experimental version

OCaml 5 is coming
... this summer

Parallelism

- OCaml 4 runtime was protected by a global lock: only one thread was ever doing OCaml-side work
- OCaml 5 supports parallelism using domains
- The standard library expose low-level constructs. Users are expected to use higher-level libraries.

```
module Task = Domainslib.Task
let rec fib_par pool n =
  if n > 20 then begin
    let a = Task.async pool (fun _ -> fib_par pool (n-1)) in
    let b = Task.async pool (fun _ -> fib_par pool (n-2)) in
    Task.await pool a + Task.await pool b
  end else fib n
```

Effect handlers

Effect handlers allow the programmers to describe *computations* that *perform* effectful *operations*, whose meaning is described by *handlers* that enclose the computations.

- OCaml 5 runtime supports algebraic effects and effect handlers
- Experimental support in the language exposed through low-level primitives
- Syntax and type systems will be designed in future versions

```
open Effect
open Effect.Deep
type _ Effect.t += Xchg: int -> int t
let comp1 () = perform (Xchg 0) + perform (Xchg 1)


try_with comp1 ()
{ effc = fun (type a) (eff: a t) ->
    match eff with
    | Xchg n -> Some (fun (k: (a, _) continuation) ->
        continue k (n+1))
    | _ -> None }
```

Sequential compatibility

- Mostly identical performance for sequential code
- Full compatibility with OCaml 4.14
- ... except for some long deprecated modules
- ... with compatibility libraries available when 5.0 release

Conclusion

Conclusion

- Regular releases, better user experience, internal refactoring, better development analytic
- A lot of progress ... a lot of work remaining
- A new exciting major version is on the horizon
- And stay tuned for the next talk on longer-term evolutions for OCaml

Thank you for your attention