# OCaml 4.02

ygrek

OCaml@Singapore

Nov 25, 2014

# Disclaimer

This talk is a shameless ripoff and compilation of OCaml changelog, documentation, bugtracker and ICFP'14 slides. All respect goes to original authors, all mistakes and misrepresentations are mine.

# Past

- *3.12* (August 2010)
  - polymorphic recursion
  - first-class modules
  - record fields punning
- *4.00* (July 2012)
  - GADTs
  - `-bin-annot` typedtree dump
- *4.01* (April 2014)
  - `-ppx` parsetree rewriters
  - type-level disambiguation of record labels
  - `(|>)` and `(@@)` in `Pervasives`

# Current

*4.02* (August 2014)

- attributes
- module aliases
- `-safe-string` read-only strings
- match case for exceptions
- generative functors
- open types

# Future?

- ephemerons
- smarter inliner
- modular implicits (aka scoped typeclasses)

# Attributes

Standard way to put annotations on AST nodes from the source code. Those annotations can later be consumed by `ppx` rewriters and the compiler itself. Possible (future) uses:

- anchors for code generation
- per-scope enabling of certain compiler features (options, warnings)
- deprecation markers (`[@@ocaml.deprecated]`)
- documentation comments

# Attributes (example)

```
type t = {
  x : int;
  y [@name "z"] : int;
} [@@generate]
```

ocamlc -c -dparsetree attributes.ml :

```
[
  structure_item (attributes.ml[1,0+0]..[4,45+14])
    Pstr_type
    [
      type_declaration "t" (attributes.ml[1,0+5]..[1,0+6]) (attributes.ml[1,0+5]..[4,45+14])
        attribute "generate"
          []
...
```

# Module aliases

New signature item

```
module A = B
```

which records and tracks the equality of module paths. The aliases are expanded only at the place of usage, not at definition.
Benefits :

- solution to the long-standing issue with functors depending on syntactic module paths equality
- smaller compiled objects and interfaces
- faster compilation
- less link-time dependencies and smaller binaries (`-no-alias-deps`)

# Read-only strings

Introduces `type bytes` equal to `string` (preserving backward compatibility) unless option `-safe-string` is used, in which case it is opaque. Module `Bytes` in standard library can be used to create values of this type.
The intended usage is: `Bytes` for writable buffers, `String` for read-only strings. Option `-safe-string` is likely to become default in the future compiler release.

# Read-only strings (cont.)

```
# #show_module Bytes;;
module Bytes :
  sig
    ...
    val make : int -> char -> bytes
    val init : int -> (int -> char) -> bytes
    val of_string : string -> bytes
    val to_string : bytes -> string
    val sub : bytes -> int -> int -> bytes
    val sub_string : bytes -> int -> int -> string
    ...
  end
# #show_module String;;
module String :
  sig
    external length : string -> int = "%string_length"
    external get : string -> int -> char = "%string_safe_get"
    external set : bytes -> int -> char -> unit = "%string_safe_set"
    ...
  end
```

# Match case for exceptions

Well-known inconvenience for seasoned OCaml programmers: catching exceptions in tail recursive function.

```
(* Bad *)
let rec lines ch acc =
  try
    lines ch (input_line ch :: acc)
  with
    End_of_file -> List.rev acc
```

# Match case for exceptions

Well-known inconvenience for seasoned OCaml programmers: catching exceptions in tail recursive function.

```
(* Bad *)
let rec lines ch acc =
  try
    lines ch (input_line ch :: acc)
  with
    End_of_file -> List.rev acc
```

```
(* Ugly *)
let rec lines ch acc =
  match try Some (input_line ch) with End_of_file -> None with
  | None -> List.rev acc
  | Some s -> lines ch (s::acc)
```

# Match case for exceptions

Well-known inconvenience for seasoned OCaml programmers: catching exceptions in tail recursive function.

```
(* Bad *)
let rec lines ch acc =
  try
    lines ch (input_line ch :: acc)
  with
    End_of_file -> List.rev acc
```

```
(* Ugly *)
let rec lines ch acc =
  match try Some (input_line ch) with End_of_file -> None with
  | None -> List.rev acc
  | Some s -> lines ch (s::acc)
```

```
(* Good *)
let rec lines ch acc =
  match input_line ch with
  | s -> lines ch (s::acc)
  | exception End_of_file -> List.rev acc
```

# Generative functors

Consider functor applied to the same module twice:

```
module F (X : sig end) = struct type t end
module B1 = F(String)
module B2 = F(String)
let f (x:B1.t) : B2.t = x
(* val f : B1.t -> B2.t = <fun> *)
```

Should $B1.t$ be considered equal to $B2.t$ or not? OCaml's functors are *applicative* by default, which means that functor applications with equal parameters yield modules with equal type components. But sometimes *generative* functors are desired, which generate new types for each application.

```
module F (X : sig end) () = struct type t end
module B1 = F(String)()
module B2 = F(String)()
let f (x:B1.t) : B2.t = x
(* Error: This expression has type B1.t but an expression was expected of type B2.t *)
```

# Open types

OCaml already had one extensible type that could have constructors added after it's definition. This is the type of exceptions : `exn`. It is special - one couldn't define new types with such behaviour. Not anymore:

```
# type t = ..;;
type t = ..
# type t += A | B of int;;
type t += A | B of int
# B 2;;
- : t = B 2
# C 3.;;
Error: Unbound constructor C
# type t += C of int;;
type t += C of float
# C 3.;;
- : t = C 3.
```

Obviously, every pattern match on the value of open type should include wildcard case.

# FIN

And all this goodness is just one command away. Do it now :

```
opam switch 4.02.1
```

# Community efforts

- OPAM
- merlin
- github pull requests

# OPAM

New OPAM 1.2 release adds several useful features.

- usable `opam pin`
- `opam source`
- `dev-repo` field in *opam* files
- `{build}` filter for `depends`

# merlin

Code browsing tool for OCaml projects.
Integrates with Vim, Emacs, acme and Sublime Text, can be extended to support any capable editor. Uses *bin-annot* files produced by the compiler and handles partial builds.
Provides:

- type throwback for values under cursor
- "Go to definition" across the whole project (and possibly for external libraries)
- background build and on-the-fly error messages

# github

Since some time ago there is an official mirror of OCaml compiler on github.
More importantly the pull requests against that repo are considered by Inria team.
Getting the patch into OCaml compiler had never been easier.
Push it while it lasts!

# References

1) A Guide to Extension Points in OCaml
2) https://ocaml.org/meetings/ocaml/2014/
3) The State of OCaml
4) Ephemerons meet OCaml GC
5) https://github.com/ocaml/ocaml