Effective ML

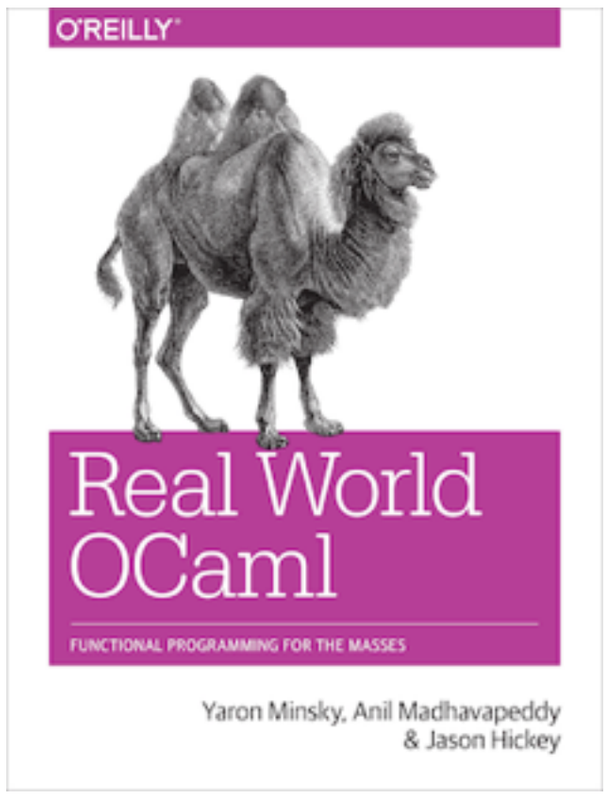
Lessons from 10 years of OCaml at work

Lessons

- Favor readers over writers
- Create uniform interfaces
- Make illegal states unrepresentable
- Code for exhaustion
- Make common errors obvious
- Prefer explicit to terse
- Avoid boilerplate
- Types first
- Don’t be puritanical about purity
- Avoid complex type hackery

Learn more

- OPAM: package manager for OCaml
- http://janestreet.github.io
  - Core: a replacement for OCaml stdlib
  - Async: cooperative concurrency library
- Real World OCaml
  http://realworldocaml.org



Sound like fun?

http://janestreet.com/apply

Internships and
Full-time positions

Phantom Types

Hack your Compiler

```
type sexp = | Atom of string
            | List of sexp list

(* Example.
     ((this) (is an s-expression))
   would be written like this:
*)
let example =
  List [List [Atom "this"];
        List [Atom "is";Atom "an";Atom "s-expression"]]
```

```
type order =
  { time:  Time.t;
    sym:   Trading_symbol.t;
    price: float;
    size:  int;
    dir:   Dir.t;
  }


let example_string =
  "((time \"2009-03-12 11:27:32\")
    (sym C)
    (price 3.87)
    (size 50_000)
    (dir Buy))"
```

```
type order =
  { time:  Time.t;
    sym:   Trading_symbol.t;
    price: float;
    size:  int;
    dir:   Dir.t;
  }

let sexp_of_order o =
  List [ List ["time"  ; Time.sexp_of_t           o.time  ];
         List ["sym"   ; Trading_symbol.sexp_of_t o.sym   ];
         List ["price" ; sexp_of_float            o.price ];
         List ["size"  ; sexp_of_int              o.size  ];
         List ["dir"   ; Dir.sexp_of_t            o.dir   ];
       ]
```

Can’t a machine do this for me?

Yes, with camlp4!

```
type order =
  { time:  Time.t;
    sym:   Trading_symbol.t;
    price: float;
    size:  int;
    dir:   Dir.t;
  }
with sexp
```

More Syntactic Fun...

- binary protocols
- comparison functions
- hash functions
- type hashes
- first-class fields and variants