Tracing OCaml Programs

Darius Foo, Wei-Ngan Chin
National University of Singapore
OCaml 2022
# Debugging in OCaml today

<table>
<thead>
<tr>
<th>#trace</th>
<th>ocamldebug</th>
<th>printf</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Getting an overview</td>
<td>✓ Reverse execution</td>
<td>✓ Control over output</td>
</tr>
<tr>
<td>× Too much output</td>
<td>× Hard to get an overview</td>
<td>× Modifying source</td>
</tr>
<tr>
<td>× Bytecode toplevel</td>
<td>× Needs inputs</td>
<td>× Inserting printers</td>
</tr>
<tr>
<td>× Code evaluation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Can a combined tool mitigate the downsides of each approach?
Type-aware record-and-replay debugging

• *Instrument* program to collect events
  • e.g. function calls and returns, with arguments and return value
• Run program and *record* an execution trace
• *Extract* information from trace
let cons x xs = x :: xs

let cons x xs =
  record_start "cons" [show x; show xs];
let res = x :: xs in
record_end "cons" (show res);
res
let rec fact n =
    if n = 0 then 1 else n * fact (n - 1)

let fact n =
    let fact_self n =
        if n = 0 then 1 else n * self (n - 1)
    in let rec aux n =
        record_start "fact" [show n];
        let res = fact_self aux in
        record_end "fact" (show res);
        res
    in aux n

Instrumentation
Typical ppx

- a.ml
  - ppx
  - a.pp.ml
    - typecheck
    - a.pp.cmt
      - compile
      - a.exe

- Error symbol
Typed ppx

(typpx, typedppxlib)
Typed ppx redux

The overhead of typechecking twice is difficult to avoid in the general case.

This is a specific case: we can separate the parts of the program which produce recorded values from the parts which consume them.

This allows compilation to be staged.
Typed ppx redux

- a.ml
  - ppx
  - a.pp.ml
  - typecheck
  - a.pp.cmt
  - compile
  - a.exe
  - binary trace

- reader.ml
  - ppx
  - reader.pp.ml
  - compile
  - reader.exe
  - trace

---

Common Trace Format

A FLEXIBLE, HIGH-PERFORMANCE BINARY TRACE FORMAT
Tradeoffs

- Makes debugging a build problem instead of a runtime problem
- Code must be recompiled to be instrumented
  - Library code is not instrumented, however we can see its output
- Fragile uses of Typedtree APIs
  - Less than vending typechecker
- Staged build a workaround for lack of ad hoc polymorphism?
  - Separating content from schema does lower (runtime, compile) overhead
- Scalability?
  - Lots of configuration for instrumentation process
  - In principle, could be no more expensive than regular printf
Other debugging methods

• Logging, tracing, testing (Runtime Events)
• gdb, rr, dtrace
• ocamli, Furukawa et. al’s stepper
  • magic-trace
  • hat
Work in progress

- Try on projects of all kinds and sizes
- Make build integration simpler
- More ways to query traces (backward slicing, evaluate code, ...)
- Concurrency

https://github.com/dariusf/ppx_debug

Thank you!