Compiling OCaml to C, and observing values with `liballocs`

Cheng Sun

CST Part II Project

16th June 2017

An example

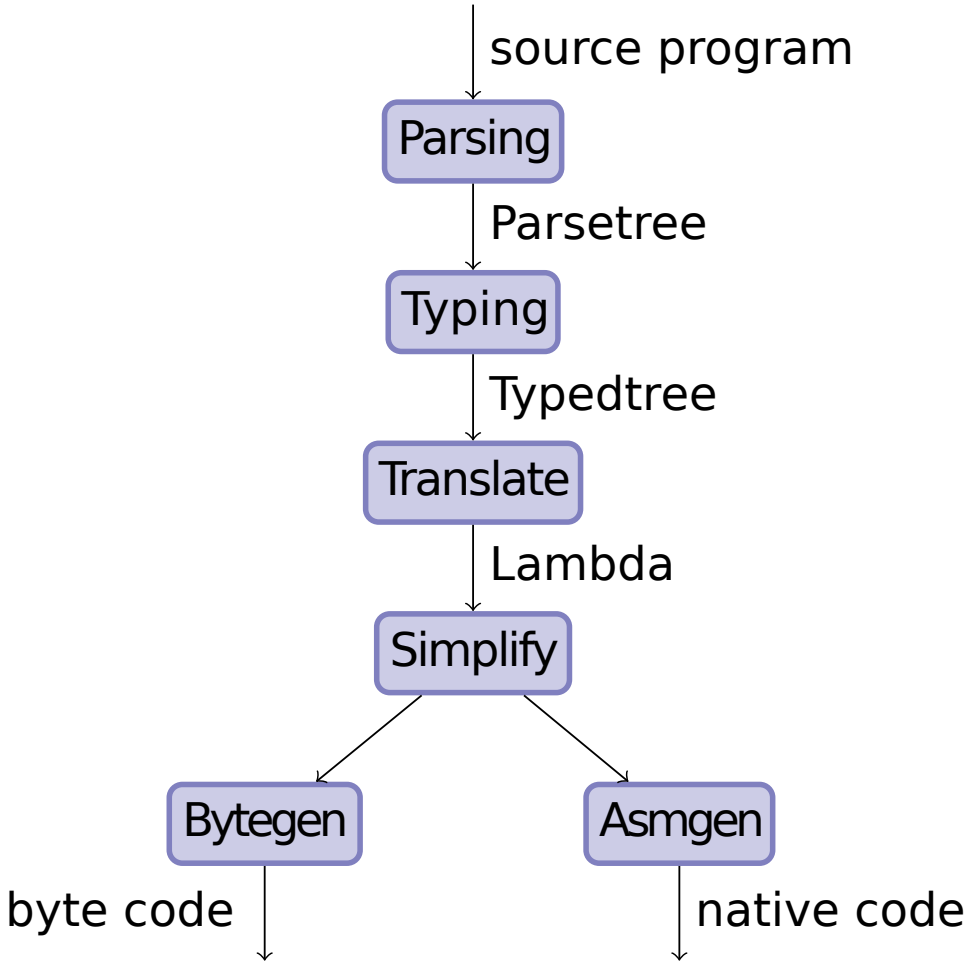
```
(* my_rev : 'a list -> 'a list *)
let my_rev xs =
    let rec loop accum = function
        | [] -> accum
        | x::xs -> loop (x::accum) xs
    in
    loop [] xs


let _ = my_rev [1;2;3]
```

ocamldebug

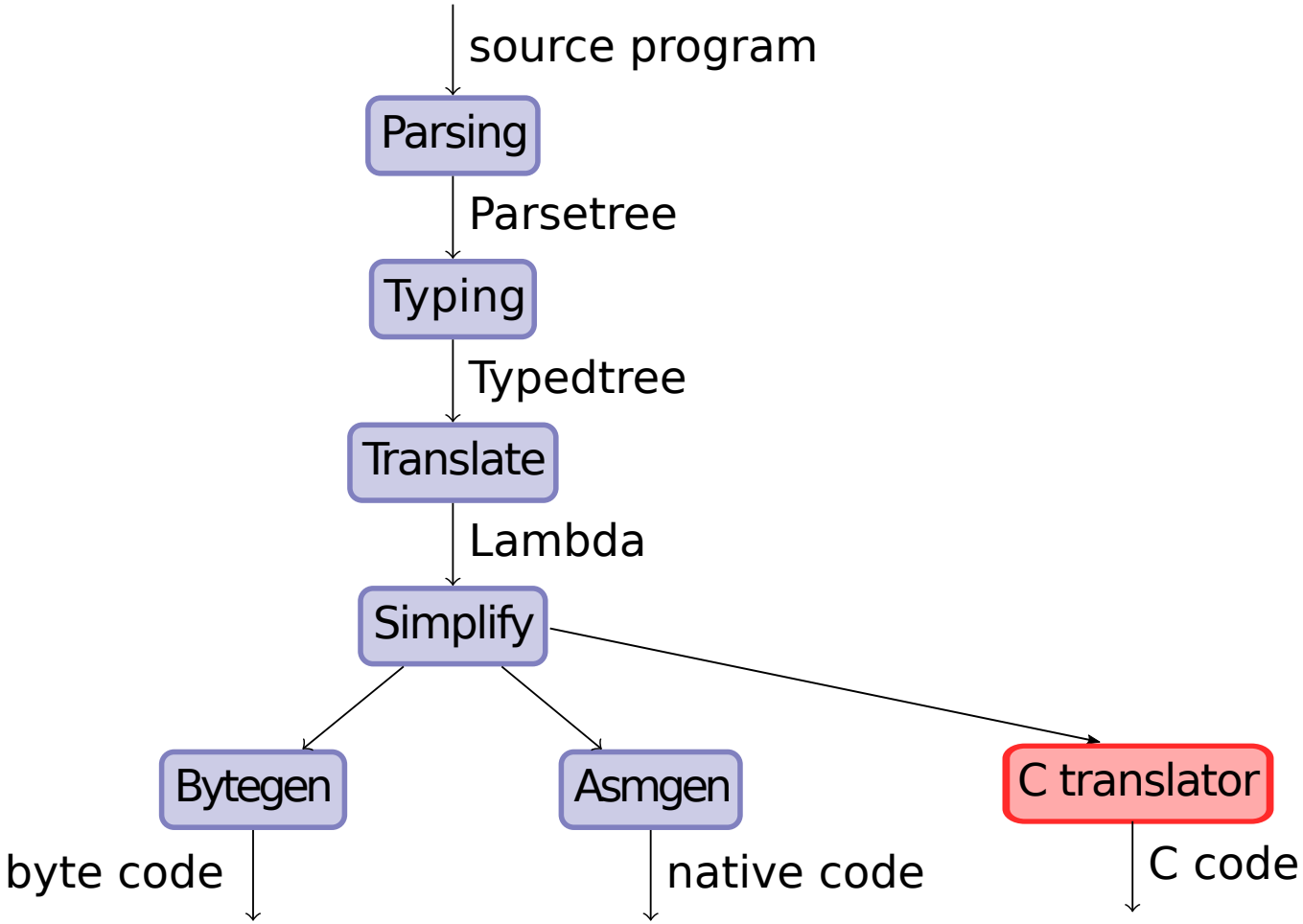
```
(ocd) break @ Test 5
Loading program... done.
Breakpoint 1 at 6188: file test.ml, line 3,
   characters 18-94
(ocd) run
Time: 14 - pc: 6220 - module Test
Breakpoint: 1
5          | x::xs -> <|b|>loop (x::accum) xs
(ocd) print x
x: 'a = <poly>
```

My approach



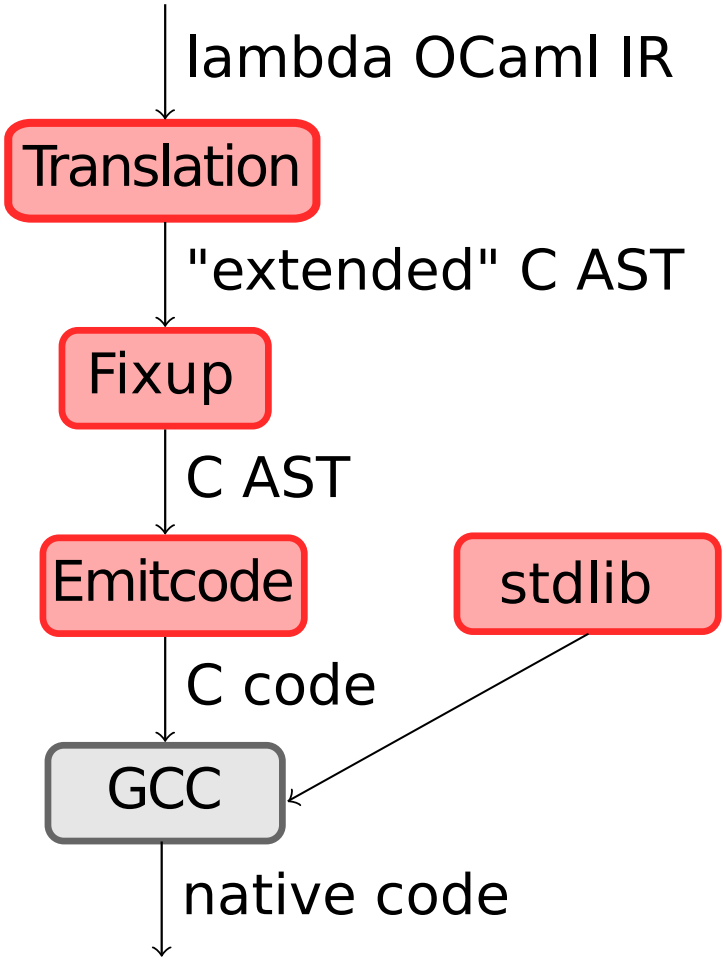
(adapted from Fischbach 2011)

My approach



(adapted from Fischbach 2011)

My approach


lambda OCaml IR
Translation
"extended" C AST
Fixup
C AST
Emitcode
stdlib
C code
GCC
native code

What's supported?

Enough to get many self-contained OCaml programs running.

- Basic types (tuples, records, argumentless variants)
- Most language constructs (match, if, for, while)
- Mutually recursive definitions
- Inter-module linking
- Exceptions (with arguments)
- Closures
- `List` module
- Standard input/output
- Runtime information to see through polymorphism when debugging

What's left to do?

Still some core functionality missing:

- Variants with arguments (block tags)
- Polymorphic comparison
- Submodules only partially supported
- Overapplication
- Various primitives (array, string, lazy)
- Other standard library modules
- More detailed type representation for debugging

NaN boxing

- A trick that allows any OCaml type to be placed into a 64-bit value
- In particular integers and doubles are not boxed
- The kind of value can be discriminated from the 64-bit value itself
- Required for polymorphic comparisons

Exceptions

- Linked list used to store exception handlers
- `setjmp` and `longjmp` used to perform stack unwinding
- Global variable stores current exception being handled

Closures

- All three cases handled in the compiler:
  1. let-binding (with non-empty `fvs`)
  2. anonymous lambda (with non-empty `fvs`)
  3. partial application
- Each closure involves creating a machine code stub at runtime
- Machine code carefully optimised for both speed and size
- Arbitrary number of arguments and closed variables supported
  - (Adheres to C calling standards: arguments passed on stack if necessary)

liballocs

- Written by my supervisor, Stephen Kell
- Combination of compilation toolchain and runtime library
- Provides allocation information at runtime
- Designed to work on unmodified C code
- My runtime uses liballocs to show values recursively during debugging