ocamli: Interpreted OCaml

John Whitington, University of Leicester

July 12, 2017

We present a proof-of-concept tool, ocamli, which runs ordinary OCaml programs by direct interpretation of the abstract syntax tree, taking advantage of compiler-libs for everything other than the actual evaluation. While this method of execution of an OCaml program is, clearly, rather slow, it has a number of intriguing advantages for teaching and debugging.

ocamli can run programs from source files, or from a program given on the command line. For example, using something from the Standard Library (which ocamli knows how to load and interpret):

```
$ ocamli -e "List.fold_left ( + ) 0 [1; 2; 3]" -show
6
```

ocamli knows how to print the stages of a computation, underlining the reducible expression, much as we may do by hand when teaching:

```
$ ocamli programs/functionapp.ml -show-all
    let f x = x + 1 in f (3 + 3)
=>  let f x = x + 1 in f 6
=>  let x = 6 in x + 1
=>  6 + 1
=>  7
```

We can modify the output, for example by removing **let rec** definitions, for clarity. The following example would look very crowded if the definition of `factorial` were to be included on each line:

```
$ ocamli programs/factorial.ml -show-all -remove-rec-all
   factorial 3
=> let n = 3 in if n = 1 then 1 else n * factorial (n - 1)
=> let n = 3 in if false then 1 else n * factorial (n - 1)
=> let n = 3 in n * factorial (n - 1)
=> let n = 3 in 3 * factorial (n - 1)
=> 3 * factorial (3 - 1)
=> 3 * factorial 2
=> 3 * (let n = 2 in if n = 1 then 1 else n * factorial (n - 1))
=> 3 * (let n = 2 in if false then 1 else n * factorial (n - 1))
=> 3 * (let n = 2 in n * factorial (n - 1))
=> 3 * (let n = 2 in 2 * factorial (n - 1))
=> 3 * (2 * factorial (2 - 1))
=> 3 * (2 * factorial 1)
=> 3 * (2 * (let n = 1 in if n = 1 then 1 else n * factorial (n - 1)))
=> 3 * (2 * (let n = 1 in if true then 1 else n * factorial (n - 1)))
=> 3 * (2 * 1)
=> 3 * 2
=> 6
```

The beginnings of a debugger

Eventually, we propose for ocamli to form the basis of a debugger. Since it has access to the full text at any point of evaluation, we expect it to have certain advantages over existing kinds of debugger.

For now, there are very basic debugging facilities. For example, we can add a regular expression search term to restrict the output to a line containing that term and, say, the three lines before it, to locate a bug:

```
$ ocamli programs/factorial.ml -show-all -search "if true" -upto 3
=> 3 * (2 * factorial (2 - 1))
=> 3 * (2 * factorial 1)
=> 3 * (2 * (let n = 1 in if n = 1 then 1 else n * factorial (n - 1)))
=> 3 * (2 * (let n = 1 in if true then 1 else n * factorial (n - 1)))
```

Using ocamli from other programs

Linking with ocamli allows one to evaluate arbitrary OCaml source at runtime, returning a string of the result:

```
# let x = Runeval.eval_string "List.split [(1, 2); (3, 4)]");;
val x : string = "([1; 3], [2; 4])"
```

More usefully, we can convert this to a real OCaml value, though the types must match:

```
# let x : int list * int list =
  Tinyocaml.to_ocaml_value
    (Runeval.eval_string "List.split [(1, 2); (3, 4)]");;
val x : int list * int list = ([1; 3], [2; 4])
```

A PPX rewriter, PPX_eval is provided, so that writing

```
let compiler_command = [%compiletimestr "Sys.argv.(0)"])
```

in a normal compiled OCaml program with PPX_eval might generate

```
let compiler_command = "ocamlopt"
```

Status

The ocamli interpreter is a proof-of-concept. Do not expect it to run your larger programs! It supports just enough of the language to load (and run the module initialisation of) the OCaml Standard Library. This is quite a large number of constructs, though, including functors, first class modules and so on. In particular, ocamli can run almost all the programs in the *OCaml from the Very Beginning* textbook. The examples are included in the download.

ocamli currently makes no guarantee of computational complexity, even when the steps of evaluation are not shown. The extent to which such a guarantee can be given is an open research question.

Implementation Details

The ocamli interpreter requires no patches to the OCaml compiler; it is implemented entirely using compiler-libs. The OCaml program is lexed and parsed by compiler-libs, typechecked, then the parse tree is converted to a simpler, more direct datatype, called Tinyocaml. The Tinyocaml representation is then interpreted step-by-step, optionally printing each stage out. We use a custom prettyprinter, but hope to eventually contribute back to the standard one. ocamli's facilities for conversing with C code (such as OCaml `%external` declarations) are based on reading and writing OCaml heap values.

Keeping ocamli in sync with OCaml will involve updating it to reflect any changes in the compiler-libs API and adding any new features added to the OCaml languages. So, we will have ocamli 4.05 for OCaml 4.05 and so on.

Download

`https://github.com/johnwhitington/ocamli` (tag "Ocaml17")

Paper

A position paper describing the possiblities of ocamli as a debugger is available: John Whitington and Tom Ridge *Visualizing the Evaluation of Functional Programs for Debugging* SLATE '17, Oporto, 2017
`http://www.cs.le.ac.uk/people/jw642/visfunc.pdf`