ocamli: Interpreted OCaml

JOHN WHITINGTON UNIVERSITY OF LEICESTER

INTRODUCTION

ocamli is a *proof-of-concept* interpreter for OCaml, which can show the steps of evaluation of an OCaml program for teaching or debugging. For example:

```
$ ocamli -e "1 + 2 * 3" -show-all
   1 + 2 * 3
=> 1 + 6
=> 7
```

There are options to search and prune the output, to stop execution early, and to load libraries such as the OCaml Standard Library or the Unix library.

OBJECTIVES

Teaching It's useful to draw this kind of evaluation diagram out by hand, in small-group teaching sessions. But, it takes a lot of time, and doesn't scale well. The interpreter ocamli can help by providing the ability to interactively experiment.

Debugging Whilst ocamli is not yet complete enough to act as a reliable debugger, extending it to do the job should be possible. It would need to support the whole language, and cope with mixed C/OCaml programs, and so on. If it can't run any program, it might not be able to run my program, and will not be an attractive debugger.

SAMPLE EXECUTION

```
$ ocamli programs/factorial.ml ...
    factorial 4
n = 4   =>  if n = 1 then 1 else n * factorial (n - 1)
n = 4   =>  n * factorial (n - 1)
        =>  4 * factorial 3
n = 3   =>  4 * (if n = 1 then 1 else n * factorial (n - 1))
n = 3   =>  4 * (n * factorial (n - 1))
        =>  4 * (3 * factorial 2)
n = 2   =>  4 * (3 * (if n = 1 then 1 else n * factorial (n - 1)))
n = 2   =>  4 * (3 * (n * factorial (n - 1)))
        =>  4 * (3 * (2 * factorial 1))
n = 1   =>  4 * (3 * (2 * (if n = 1 then 1 else n * factorial (n - 1))))
        =>  4 * (3 * (2 * 1))
        =>  24
```

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.

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.

ocamli's facilities for conversing with C code (such as OCaml `%external` declarations) are based on reading and writing OCaml heap values.

STATUS

The ocamli interpreter is a proof-of-concept. Do not expect it to run all or even most of your 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.

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.

ocamli can be downloaded from GitHub, and contains simple documentation and some example programs.

MORE SAMPLE EXECUTIONS

We can search through the output, showing only certain steps of execution.

"I want to see the last few steps before `if true`":

```
$ 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)))
```

The interpreter can be used at runtime, and the resulting value bought back into the caller:

```
# 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 generates

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

REFERENCES

[1] compiler-libs: The ocaml compiler front-end http://caml.inria.fr/pub/docs/manual-ocaml/parsing.html.

[2] J. Whitington and T. Ridge. Visualizing the Evaluation of Functional Programs for Debugging. *SLATE '17*, 2017.

FUTURE WORK

- Extend ocamli to the full language
- Make it fast enough for general use
- Easy invocation as a debugger regardless of build system or source structure
- Better tools for searching and eliding
- Allow interpretation of just one module – other modules run at full speed
- An interactive interface

CONTACT INFORMATION

Web cs.le.ac.uk/people/jw642/
Email jw642@le.ac.uk
GitHub github.com/johnwhitington/ocaml-i (tag "OCaml17")