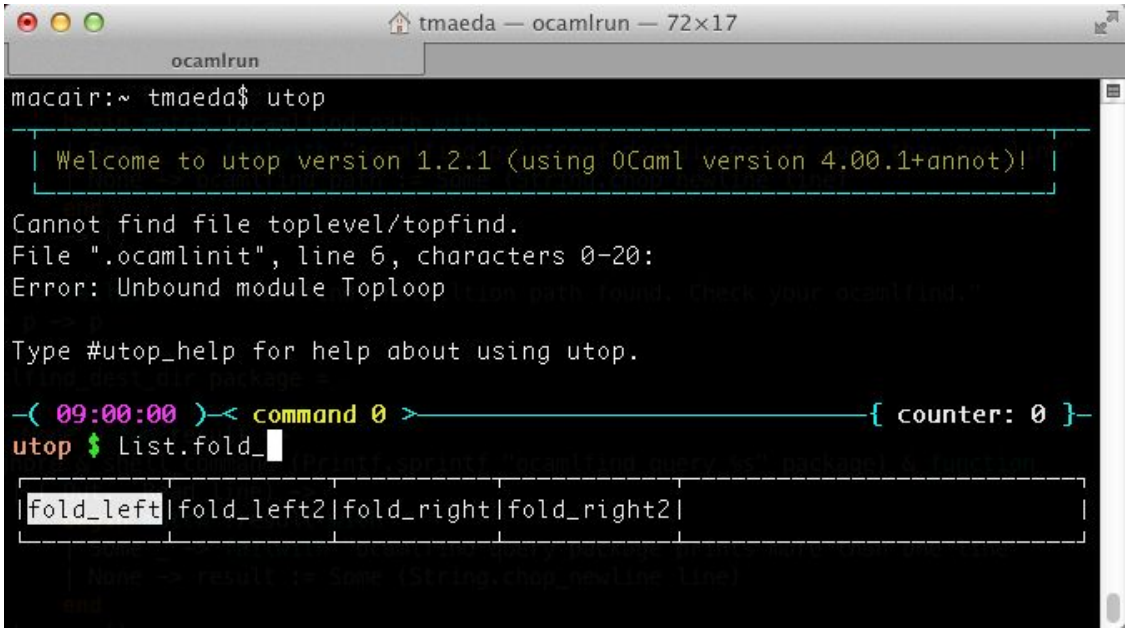
OCaml Standard Tools

Letterio Galletta

PA Lecture

19/10/2015

Interactive Toplevel



Advantages

- Immediate feedbacks
- Rapid prototyping
- Interactive programming

Disadvantages

- No standalone applications
- No optimized native code

Today's lecture: others OCaml tools

List of presented tools

- `ocamlc`
- `ocamlopt`
- `ocamlrun`
- `ocamlmktop`
- `ocamlbuild`
- `ocamlcp/ocamloptp`
- `ocamlprof`
- `ocamldoc`

References

- OCaml Manual
- OCaml Standard Tools Cheat Sheets

Compilers

Bytecode

`ocamlc[.opt]`

- `*.cmo` bytecode object
- `*.cmi` interface object
- `*.cma` bytecode library

Native-code

`ocamlopt[.opt]`

- `*.cmx` & `*.o` asm object
- `*.cmi` interface object
- `*.cmxa` & `*.a` native library

Demo: bytecode compiler

- `ocamlc -c ex1.ml`
  - compile a source file
  - generate `ex1.cmo` & `ex1.cmi`
  - the option `-c` means “only compilation”
- `ocamlc ex1.ml -o ex1`
  - build a bytecode executable
  - generate `ex1.cmo` & `ex1.cmi` & `ex1`
  - the option `-o` specifies the executable name
- `./ex1` & `ocamlrun ex1` to run the executable

Demo: native-code compiler

- `ocamlopt -c ex1.ml`
  - compile a source file
  - generate `ex1.cmx` & `ex1.cmi` & `ex1.o`
  - the option `-c` means “compile only”
- `ocamlopt ex1.ml -o ex1`
  - build a native executable
  - generate `ex1.cmx` & `ex1.cmi` & `ex1.o` & `ex1`
  - the option `-o` specifies the executable name
- `./ex1` runs the program

Demo: use a library

- `ocamlc graphics.cma -o sierpinski sierpinski.ml`
  - build the bytecode executable `sierpinski`
  - `graphics.cma` bytecode library
  - `ocamlc` links together `graphics.cma` and `sierpinski.cmo`
- `ocamlopt graphics.cmxa -o sierpinski sierpinski.ml`
  - build a native executable
  - `graphics.cmxa` native-code library
  - `ocamlopt` links together our application and the library
- `#load "graphics.cma";;` in the OCaml interactive toplevel
- `ocamlmktop graphics.cma -o graphicstop`
  - a toplevel with code of a library preloaded at startup

Demo: create a library

- Generate a library (`*.cma`)
  - `ocamlc -c util.ml`
  - `ocamlc -a -o libutil.cma util.cmo`
- Compile your application
  - `ocamlc libutil.cma -o main main.ml`
- Native-code
  - `ocamlopt -c util.ml`
  - `ocamlopt -a -o libutil.cmxa util.cmx`
  - `ocamlopt libutil.cmxa main.ml -o main`

`ocamlbuild`: a generic build system

- Simplify the compilation of ocaml projects
  - determine the sequence of calls to the compiler
- in many cases automatically discover the various source files and dependencies of a project

Demo: simple use of `ocamlbuild`

- bytecode application
  - `ocamlbuild ex1.byte` bytecode application
  - `ocamlbuild -libs graphics sierpinsky.byte`
  - `ocamlbuild util.cma` bytecode library
- native-code application
  - `ocamlbuild ex1.native` native-code application
  - `ocamlbuild -libs graphics sierpinsky.native`
  - `ocamlbuild util.native` native-code library

Other tools

- Profiling (`ocamlprof`)
  - `ocamlcp/ocamloptp ex1.ml -o ex1` compile the program
  - `./ex1` run
  - `ocamlprof ex1.ml` call `ocamlprof`
- Documentation (`ocamldoc`)
  - special comments in the code
  - `mkdir doc && ocamldoc -html -d doc ex1.ml`
  - other output format
    - -latex
    - -texi
    - -man