Preface

This is a book about hacking in ocaml. It's assumed that you already understand the underlying theory. Happy hacking Most parts are filled with code blocks, I will add some comments in the future. Still a book in progress. Don't distribute it.

☺

Acknowledgements

write later

Contents

Todo list

Chapter 1

Tool Chain

1.7 Standard OCaml Develop Tools

ocaml	toplevel top
ocamlrun	bytecode interpreter
ocamlc	bytecode batch compiler
ocamlopt	native code batch compiler
ocamlc.opt	*optimized* bytecode batch compiler
ocamlopt.opt	*optimized* native code batch compiler
ocamlmktop	new *toplevel* constructor

Table 1.1: Ocaml Compiler Tools

The optimized compilers are themselves compiled with the Objective Caml native compiler. They compile *faster* but are otherwise *identical* to their unoptimized counterparts.

Compilation Unit For the interactive system, the unit of compilation corresponds to a phrase of the language. For the batch compiler, the unit of compilation is two files: the source file, and the interface file. The *compiled interface* is used for both the bytecode and native code compiler.

extension	meaning
.ml	source
.mli	interface
.cmi	compiled interface
.cmo	object file (byte)
.cma	library object file(bytecode)
.cmx	object file (native)
.cmxa	library object file(native)
.c	c source
.o	c object file (native)
.a	c library object file (native)

Table 1.2: ocaml file name extension

-a	construct a runtime library
-annot	save information in <filename>.annot
-c	compile *without* linking
-o name_of_executabular	specify the name of the executabular
-linkall	link with *all* libraries used
-i	*display all* compiled global declarations, generate .mli file
-pp	preprocessor
-unsafe	*turn off* index checking
-v	display version
-w list	choose among the list the level of warning message
-impl file	indicate that *file* is a caml source(.ml)
-intf file	as a caml interface(.mli)
-I dir	add directory in the list of directories
-thread	light process
-g, -noassert	linking with debug information
-custom, -cclib, -ccopt, -cc	standalone executeblel
-make-runtime, -use-runtime	runtime
-output-obj	output a c object file instead of an executable
-vmthread	VM-level thread support
-dparsetree	generate the parse output
-drawlambda	s-expression
-dlambda	s-experssion
-dinstr	generate asm

Table 1.3: ocamlc options

A/a	enable/disable all messages
F/f	partial application in a sequence
P/p	incomplete pattern matching
U/u	missing cases in pattern matching
X/x	enable/disable all other messages
M/m and V/v	for hidden object

Table 1.4: warning option

About warning messages (Table 1.4) , the compiler chooses the **(A)** by default. turn off some warnings sometimes is helpful, for example

```
ocamlbuild -cflags -w,aPF top_level.cma
```

-compact	optimize the produced code for space
-S	keeps the assembly code in a file
-inline level	set the aggressiveness of inlining
-S	keep assembly output

Table 1.5: ocamlopt option

-I dir	adds the directory
-unsafe	no bounds checking

Table 1.6: toplevel option

1.8 ocamlmktop

OCAMLMKTOP(Table 1.6) is ofen used for *pulling native object* code libraries(typically written in C) into a new toplevel. *-cclib libname, -ccopt optioin, -custom, -I dir -o exectuable*

For example:

```
ocamlmktop -custom -o mytoplevel graphics.cma \
   -cclib -I/usr/X11/lib -cclib -lX11
```

This *standalone* exe(-custom) wil be *linked* to the library X11(libX11.a) which in turn will be looked up in the path */usr/X11/lib*

A standalone exe is a program that *does not* depend on OCaml installation to run. The OCaml *native* compiler produces standalone executabulars by default. But *without -custom* option, the *bytecode* compiler produces an executabular which requires the *bytecode interpreter ocamlrun*

```
ocamlc test.ml -o a
ocamlc -custom test.ml -o b
-rwxr-xr-x   1 bob  staff    12225 Dec 23 16:31 a
-rwxr-xr-x   1 bob  staff   198804 Dec 23 16:31 b
```

You can see the size of `b`with the ocamlrun is way more bigger than `a`

```
bash-3.2$ cat a | head -n 1
#!/Users/bob/SourceCode/ML/godi/bin/ocamlrun
```

Without *-custom*, it depends on *ocamlrun*. With *-custom*, it contains the *Zinc* interpreter as well as the program bytecode, this file can be executed directly or copied to another machien(using the same CPU/Operating System). Still, the inclusion of machine code means that stand-alone executalbes are not protabular to other systems or other architectures.

Optimization It is necessary to not create *intermediate closures* in the case of application on several arguments. For example, when the function *add* is applied with two integers, it is not useful to create the first closure corresponding to the function of applying add to the first argument. It is necessary to note that the creation of a closure would *allocate* certain memory space for the environment and would require the recovery of that memory space in the future. *Automatic memory recovery* is the second major performance concern, along with environment.

Chapter 2

Lexing

Chapter 3

Parsing

Chapter 4

Camlp4

Camlp4 stands for Preprocess-Pretty-Printer for `OCaml`, it's extremely powerful and hard to grasp as well. It is a source-to-source level translation tool.

Chapter 5

Libraries

5.16 Modules

Chapter 6

Runtime

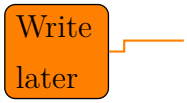
Chapter 7

GC

Should be re-written later

Chapter 8

Object-oriented


Write
later

Chapter 9

Language Features

9.9 The module Language

Chapter 10

subtle bugs

Chapter 11

Interoperating With C

Write later

Chapter 12

Pearls

Chapter 13

XX

Chapter 14

Topics