

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

| | |
|--|-----------|
| Preface | 3 |
| Acknowledgements | 5 |
| 1 Tool Chain | 17 |
| 1.1 Ocamlbuild | 18 |
| multiple directories | 21 |
| grouping targets | 22 |
| With lex yacc, ocamlfind | 23 |
| 1.1.1 Principles | 24 |
| 1.1.2 Write plugin | 25 |
| Samples | 28 |
| Mixing with C stubs | 30 |
| 1.1.3 OCamlbuild in the toplevel | 36 |
| 1.1.4 Building interface files | 37 |
| Interaction with Git | 37 |
| 1.2 Godi,otags | 38 |
| 1.2.1 CheatSheet | 38 |
| 1.3 Ocamlfind | 39 |
| 1.3.1 CheatSheet | 39 |
| 1.3.2 META file | 39 |
| 1.4 toplevel | 40 |
| 1.4.1 directives | 40 |

| | | |
|----------|---|-----------|
| 1.4.2 | Module toplevel | 40 |
| 1.4.3 | Env | 41 |
| 1.5 | OcamlDoc | 43 |
| 1.6 | ocamlmktop | 46 |
| 1.7 | Standard OCaml Develop Tools | 47 |
| 1.8 | ocamlmktop | 49 |
| 1.9 | Git | 51 |
| 2 | Lexing | 53 |
| 2.1 | Lexing | 54 |
| 2.1.1 | Ulex interface | 59 |
| 2.2 | Ocamllex | 67 |
| 3 | Parsing | 71 |
| 3.1 | Ocamlyacc | 72 |
| 3.2 | MENHIR Related | 86 |
| 4 | Camlp4 | 89 |
| 4.1 | Predicate Parser | 90 |
| 4.2 | Basic Structure | 91 |
| 4.2.1 | Experimentation Environment | 91 |
| 4.2.2 | Camlp4 Modules | 91 |
| 4.2.3 | Camlp4 Make Toplevel | 92 |
| 4.2.4 | Command Options | 95 |
| 4.2.5 | Module Components | 96 |
| 4.2.6 | Simple Experiment | 97 |
| 4.3 | Camlp4 SourceCode Exploration | 98 |
| 4.3.1 | Camlp4 PreCast | 98 |
| 4.3.2 | OCamlInitSyntax | 100 |
| 4.3.3 | Camlp4.Sig | 105 |
| 4.3.4 | Camlp4.Struct.Camlp4Ast.mlast | 105 |
| 4.3.5 | AstFilters | 106 |

| | | |
|--------|--|-----|
| 4.3.6 | Camlp4.Register | 106 |
| 4.3.7 | Camlp4Ast | 111 |
| 4.3.8 | TestFile | 139 |
| 4.4 | Extensible Parser | 140 |
| 4.4.1 | Examples | 140 |
| 4.4.2 | Mechanism | 141 |
| 4.4.3 | Parsing OCaml using Camlp4 | 146 |
| | Fully Utilize Camlp4 Parser and Printers | 146 |
| | Otags Mini | 146 |
| | Parsing Json AST | 148 |
| 4.5 | STREAM PARSER | 150 |
| 4.6 | Grammar | 151 |
| 4.6.1 | LEVEL | 153 |
| 4.6.2 | Grammar Modification | 153 |
| | Example: Expr Parse Tree | 156 |
| 4.7 | QuasiQuotations | 161 |
| 4.7.1 | Quotation Introduction | 162 |
| 4.7.2 | Quotation Expander | 163 |
| | Lambda Example | 167 |
| 4.8 | Ast Transformation | 171 |
| 4.9 | Quotation Cont | 174 |
| 4.9.1 | Quotation module | 174 |
| 4.10 | Antiquotation Expander | 176 |
| 4.11 | Revised syntax | 180 |
| 4.12 | Filters in camlp4 | 186 |
| 4.12.1 | Map Filter | 186 |
| 4.12.2 | Filter Examples | 186 |
| | Example: Map Filter | 186 |
| | Linking Problem | 188 |
| | Example: Add Zero | 189 |
| | Fold filter | 189 |

| | |
|--|-----|
| Meta filter | 189 |
| Lift filter | 191 |
| Macro Filter | 191 |
| 4.12.3 Example Tuple Map | 192 |
| 4.12.4 Location Strip filter | 192 |
| 4.12.5 Camlp4Profiler | 193 |
| 4.12.6 Camlp4TrashRemover | 193 |
| 4.12.7 Camlp4ExceptionTracer | 193 |
| 4.13 Examples | 194 |
| 4.13.1 Pa_python | 194 |
| 4.13.2 Pa_list | 199 |
| 4.13.3 Pa_abstract | 201 |
| 4.13.4 Pa_apply | 202 |
| 4.13.5 Pa_ctyp | 202 |
| 4.13.6 Pa_exception_wrapper | 203 |
| 4.13.7 Pa_exception_tracer | 206 |
| 4.13.8 Pa_freevars | 207 |
| 4.13.9 Pa_freevars_filter | 207 |
| 4.13.10 Pa_global_handler | 207 |
| 4.13.11 Pa_holes | 207 |
| 4.13.12 Pa_minimm | 207 |
| 4.13.13 Pa_plus | 207 |
| 4.13.14 Pa_zero | 207 |
| 4.13.15 Pa_printer | 207 |
| 4.13.16 Parse_arith | 208 |
| 4.13.17 Pa_estring | 208 |
| 4.13.18 Pa_holes | 217 |
| 4.14 Useful links | 218 |
| 4.15 Camlp4 CheatSheet | 219 |
| 4.15.1 Camlp4 Transform Syntax | 219 |
| Example: semi opaque | 219 |

| | | |
|----------|-----------------------------------|------------|
| 4.15.2 | Parsing | 220 |
| 5 | Libraries | 221 |
| 5.1 | batteries | 222 |
| | syntax extension | 222 |
| 5.1.1 | Dev | 222 |
| 5.2 | Format | 223 |
| 5.2.1 | Indentation Rules | 224 |
| 5.2.2 | Boxes | 224 |
| 5.2.3 | Directives | 226 |
| 5.2.4 | Example: Print | 226 |
| 5.1.2 | BOLT | 223 |
| 5.2 | Mikmatch | 224 |
| 5.3 | pa-do | 237 |
| 5.4 | num | 238 |
| 5.5 | caml-inspect | 239 |
| 5.6 | ocamlgraph | 244 |
| 5.7 | pa-monad | 253 |
| 5.8 | bigarray | 257 |
| 5.9 | sexplib | 258 |
| 5.10 | bin-prot | 261 |
| 5.11 | fieldslib | 262 |
| 5.12 | variantslib | 263 |
| 5.13 | delimited continuations | 264 |
| 5.14 | shcaml | 270 |
| 5.15 | deriving | 271 |
| 5.16 | Modules | 272 |
| 6 | Runtime | 275 |
| 6.1 | ocamlrun | 276 |
| 6.2 | FFI | 278 |
| 6.2.1 | Data representation | 279 |

| | | |
|-----------|--|------------|
| 6.2.2 | Caveats | 289 |
| 7 | GC | 291 |
| 8 | Object-oriented | 299 |
| 8.1 | Simple Object Concepts | 300 |
| 8.2 | Modules vs Objects | 304 |
| 8.3 | More about class | 305 |
| 9 | Language Features | 307 |
| 9.1 | Stream Expression | 308 |
| 9.2 | GADT | 312 |
| 9.3 | First Class Module | 313 |
| 9.4 | Pahantom Types | 317 |
| 9.4.1 | Useful links | 324 |
| 9.5 | Positive types | 325 |
| 9.6 | Private Types | 326 |
| 9.7 | Subtyping | 328 |
| 9.8 | Explicit Nameing Of Type Variables | 329 |
| 9.9 | The module Language | 330 |
| 10 | subtle bugs | 331 |
| 10.1 | Reload duplicate modules | 332 |
| 10.2 | debug | 334 |
| 10.3 | Debug Cheat Sheet | 335 |
| 11 | Interoperating With C | 337 |
| 12 | Pearls | 339 |
| 12.1 | Write Printf-Like Function With Ksprintf | 340 |
| 12.2 | Optimization | 340 |
| 12.3 | Weak Hashtbl | 340 |
| 12.4 | Bitmatch | 340 |

| | |
|---|------------|
| 12.5 Interesting Notes | 341 |
| 12.6 Polymorphic Variant | 342 |
| 13 XX | 347 |
| 13.0.1 tricks | 348 |
| 13.0.2 ocaml blogs | 352 |
| 14 Topics | 353 |
| 14.1 First Order Unification | 354 |
| 14.1.1 First-order terms | 354 |
| 14.1.2 Substitution | 355 |
| 14.1.3 Unification in Various areas | 355 |
| 14.1.4 Occurs check | 356 |
| 14.1.5 Unification Examples | 356 |
| 14.1.6 Algorithm | 356 |
| 14.2 LLVM | 360 |

Todo list

| | |
|---|-----|
| write later | 5 |
| mlpack file | 20 |
| Glob Patterns | 23 |
| parser-help to coordinate menhir and ulex | 66 |
| predicate parsing stuff | 90 |
| Should be re-written later | 291 |
| Write later | 306 |
| read ml 2011 workshop paper | 312 |
| Read the slides by Jacques Garrigue | 313 |
| write later with subtyping | 325 |
| write later | 330 |
| polymorphic comparison | 332 |
| Write later | 337 |

Chapter 1

Tool Chain

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

5.2 Format

Format is a pretty printer library.

```

1  open_box 0; print_string "x ="; print_space ();
2  print_int 1; close_box (); print_newline ()
3  (** equivalent *)
4
5  printf "@[x =@ %i@]." 1.
```

In case of interactive use, the system closes all opened boxes and flushes all pending text (as with the `print_newline` function) after each phrase. Each phrase is therefore executed in the initial state of the pretty-printer.

The material output by the following functions is delayed in the pretty-printer queue in order to compute the proper line breaking. Hence, you *should not mix* calls to the printing functions of the basic I/O system with calls to the functions of this module: this could result in some strange output seemingly unrelated with the evaluation order of printing commands.

Some important APIs:

open_box d opens a new pretty-printing box with *offset d*. This box is the general purpose pretty-printing box. Material in this box is displayed *horizontal or vertical*; break hints inside the box may lead to a new line, if there is no more room on the line to print the remainder of the box, or if a new line may lead to a new indentation (demonstrating the indentation of the box). When a new line is printed in the box, *d is added* to the current indentation.

print_as len str prints *str* in the current box. The *pretty-printer* formats *str* as if it were of length *len*

print_space () is used to separate items (typically to print a space between two words). It indicates that the line may be split at this point. It either prints one space or splits the line. It is equivalent to *print_break 1 0*.

print_cut () is used to mark a good break position. It indicates that the line may be split at this point. It either prints nothing or splits the line. This allows line splitting at the current point, without printing spaces or adding indentation. It is equivalent to *print_break 0 0*.

Inserts a break hint in a pretty-printing box. *print_break nspaces offset* indicates that the line may be split (a newline character is printed) at this point, if the contents of the current box does not fit on the current line. If the line is split at that point, offset is added to the current indentation. If the line is not split, nspaces spaces are printed.

Boxes, Break hints, and Indentation rules.

5.2.1 Indentation Rules

A box can state *the extra indentation of every new line* opened in its scope. This extra indentation is named *box breaking indentation*.

A break hint can also set the additional indentation of the new line it may fire. This extra indentation is named *hint breaking indentation*.

If break hint bh fires a new line within box b, then the indentation of the new line is simply the sum of: the current indentation of box b + the additional box breaking indentation, as defined by box b + the additional hint breaking indentation, as defined by break hint bh.

5.2.2 Boxes

h box, within this box, break hints do not lead to line breaks.

v box, within this box, every break hint leads to a new line.

hv box, if it is possible, the entire box is written on a single line; otherwise, every break hint within the box leads to a new line.

hov box. within this box, break hints are used to cut the line when there is no more room on the line.

Suppose we can write 10 chars before the right margin (that indicates no more room). We represent any char as a - sign; characters [and] indicates the opening and closing of a box and b stands for a break hint given to the pretty-printing engine.

The output “-b-b-” is displayed like this (the b symbol stands for the value of the break that is explained below):

within a “h” box: -b-b-

within a “v” box:

–b
–b
–

within a “hv” box: If there is enough room to print the box on the line: –b–b–
But “–b–b–” that cannot fit on the line is written

—b
—b
—

within a “hov” box: If there is enough room to print the box on the line: –b–b–
But if “–b–b–” cannot fit on the line, it is written as

—b—b
—

Break hints are also used to output spaces. You output a break hint using *print_break* *sp* *indent*, and this *sp* integer is used to print “sp” spaces. Thus *print_break sp ...* may be thought as: *print sp spaces or output a new line*.

For instance, if *b* is `break 1 0` in the output “–b–b–”, we get within a “h” box: –
– – within a “v” box: –

–
–

within a “hv” box: – – –

or, according to the remaining room on the line: –

–
–

and similarly for “hov” boxes.

5.2.3 Directives

“ “ outputs a breakable space (*print_space ()*)
“ , ” output a break hint (*print_cut ()*)
“ @; <n m> ” emit a “full” break hint (*print_break n m*)
“ @. ” end the pretty-printing, closing all the boxes still opened (*print_newline ()*)
“ @[<hov n> ” is equivalent to *open_hovbox n*.

5.2.4 Example: Print

Listing 53: Format

5.16 Modules

Chapter 6

Runtime

Chapter 7

GC

Should
be re-
written
later

Chapter 8

Object-oriented


An orange rounded rectangle with the text "Write later" inside. A small orange line extends from the right side of the rectangle.

Write
later

Chapter 9

Language Features

9.9 The module Language

An orange rectangular box with rounded corners. Inside the box, the words "write" and "later" are stacked vertically in a black, sans-serif font. A small orange L-shaped bracket is positioned to the right of the box, pointing towards the right margin.

Chapter 10

subtle bugs

Chapter 11

Interoperating With C

Write
later

Chapter 12

Pearls

Chapter 13

XX

Chapter 14

Topics