CMSC 330: Organization of Programming Languages

Functional Programming with OCaml

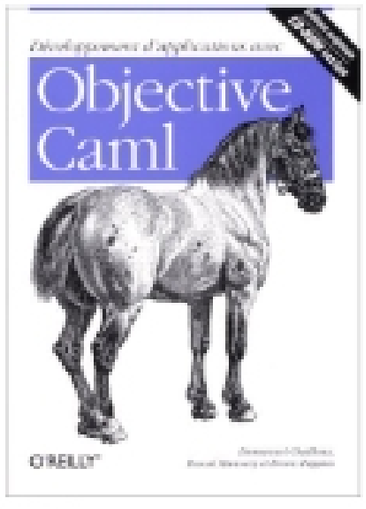
Background

- ML (Meta Language)
  - Univ. of Edinburgh, 1973
  - Part of a theorem proving system LCF
    - The Logic of Computable Functions
- SML/NJ (Standard ML of New Jersey)
  - Bell Labs and Princeton, 1990
  - Now Yale, AT&T Research, Univ. of Chicago (among others)
- OCaml (Objective CAML)
  - INRIA, 1996
  - French Nat'lInstitute for Research in Computer Science

Dialects of ML

- Other dialects include MoscowML, ML Kit, Concurrent ML, etc.
  - But SML/NJ and OCaml are most popular
  - O = "Objective," but probably won't cover objects
- Languages all have the same core ideas
  - But small and annoying syntactic differences
  - So you should not buy a book with ML in the title
    - Because it probably won't cover OCaml

More Information on OCaml



- Translation available on the class webpage
  - *Developing Applications with Objective Caml*
- Webpage also has link to another book
  - *Introduction to the Objective Caml Programming Language*

Features of ML

- Higher-order functions
  - Functions can be parameters and return values
- "Mostly functional"
- Data types and pattern matching
  - Convenient for certain kinds of data structures
- Type inference
  - No need to write types in the source language
    - But the language is statically typed
  - Supports *parametric polymorphism*
    - Generics in Java, *templates* in C++
- Exceptions
- Garbage collection

Functional languages

- In a pure functional language, every program is just an expression evaluation

```
let add1 x = x + 1;;

let rec add (x,y) = if x=0 then y else add(x-1, add1(y));;

add(2,3) = add(1,add1(3)) = add(0,add1(add1(3)))
         = add1(add1(3)) = add1(3+1) = 3+1+1
         = 5
```

OCaml has this basic behavior, but has additional features to ease the programming process.

- Less emphasis on data storage
- More emphasis on function execution

A Small OCaml Program- Things to Notice

Use (* *) for comments (may nest)

Use let to bind variables

No type declarations

```
(* A small OCaml program *)
let x = 37;;
let y = x + 5;;
print_int y;;
print_string
  "\n";;
```

Need to use correct print function
(OCaml also has printf)

;; ends a top-level expression

Line breaks, spacing ignored (like C, C++, Java, not like Ruby)

Run, OCaml, Run

- OCaml programs can be compiled using ocamlc
  - Produces .cmo ("compiled object") and .cmi ("compiled interface") files
    - We'll talk about interface files later
  - By default, also links to produce executable a.out
    - Use -o to set output file name
    - Use -c to compile only to .cmo/.cmi and not to link
    - You'll be given a Makefile if you need to compile your files

Run, OCaml, Run (cont'd)

- Compiling and running the previous small program:

ocaml1.ml:

```
(* A small OCaml program *)
let x = 37;;
let y = x + 5;;
print_int y;;
print_string "\n";;
```

```
% ocamlc ocaml1.ml
% ./a.out
42
%
```


Run, OCaml, Run (cont'd)

Expressions can also be typed and evaluated at the top-level:

```
# 3 + 4;;
- : int = 7

# let x = 37;;
val x : int = 37

# x;;
- : int = 37

# let y = 5;;
val y : int = 5

# let z = 5 + x;;
val z : int = 42

# print_int z;;
42- : unit = ()

# print_string "Colorless green ideas sleep furiously";;
Colorless green ideas sleep furiously- : unit = ()

# print_int  "Colorless green ideas sleep furiously";;
This expression has type string but is here used with type int
```

gives type and value of each expr

"-" = "the expression you just typed"

unit = "no interesting value" (like void)

Run, OCaml, Run (cont'd)

- Files can be loaded at the top-level

ocaml1.ml:

```
(* A small OCaml program *)
let x = 37;;
let y = x + 5;;
print_int y;;
print_string "\n";;
```

```
% ocaml
        Objective Caml version 3.08.3

# #use "ocaml1.ml";;
val x : int = 37
val y : int = 42
42- : unit = ()

- : unit = ()
# x;;
- : int = 37
```

#use loads in a file one line at a time

Basic Types in OCaml

- Read e : t has "expression e has type t"

42 : int	true : bool
"hello" : string	'c' : char
3.14 : float	() : unit (* don't care value *)

- OCaml has static types to help you avoid errors
  - Note: Sometimes the messages are a bit confusing

```
# 1 + true;;
This expression has type bool but is here used with
  type int
```

  - Watch for the underline as a hint to what went wrong
  - But not always reliable

More on the Let Construct

- let is more often used for local variables
  - let x = e1 in e2 means
    - Evaluate e1
    - Then evaluate e2, with x bound to result of evaluating e1
    - x is *not* visible outside of e2

```
let pi = 3.14 in pi *. 3.0 *. 3.0;;
pi;;
```

bind pi in body of let

floating point multiplication

error

More on the Let Construct (cont'd)

- Compare to similar usage in Java/C

```
let pi = 3.14 in
  pi *. 3.0 *. 3.0;;
pi;;
```

```
{
  float pi = 3.14;

  pi * 3.0 * 3.0;
}
pi;
```

- In the top-level, omitting in means "from now on":

```
# let pi = 3.14;;
(* pi is now bound in the rest of the top-level scope *)
```


Nested Let

- Uses of let can be nested

```
let pi = 3.14 in
let r = 3.0 in
  pi *. r *. r;;
(* pi, r no longer in scope *)
```

```
{
  float pi = 3.14;
  float r = 3.0;

  pi * r * r;
}
/* pi, r not in scope */
```


Defining Functions

use let to define functions

list parameters after function name

```
let next x = x + 1;;
next 3;;
let plus (x, y) = x + y;;
plus (3, 4);;
```

no parentheses on function calls

no return statement

Local Variables

- You can use let inside of functions for locals

```
let area r =
  let pi = 3.14 in
  pi *. r *. r
```

  - And you can use as many lets as you want

```
let area d =
  let pi = 3.14 in
  let r = d /. 2.0 in
  pi *. r *. r
```


Function Types

- In OCaml, -> is the function type constructor
  - The type `t1 -> t2` is a function with argument or *domain* type `t1` and return or *range* type `t2`

- Examples
  - `let next x = x + 1 (* type int -> int *)`
  - `let fn x = (float_of_int x) *. 3.14`
    `(* type int -> float *)`
  - `print_string (* type string -> unit *)`

- Type a function name at top level to get its type