

CS 51 Code Review 4

Modules and Functors in OCaml

Sam Green and Gabbi Merz

Harvard University

Table of contents

1. Modules and Abstraction
2. Functors
3. Binary Heaps

Modules and Abstraction

- A **module** is a collection of values (and remember, functions are values) and types.
- A **module signature** or **module type** describes the contents of a module.
- While not precisely true, this analogy may help:

$$\frac{\text{type}}{\text{value}} \cong \frac{\text{signature}}{\text{module}}$$

Modules

Here's a definition of the Math module:

```
# module Math =  
  struct  
    let pi = 3.14159  
    let cos = cos  
    let sin = sin  
    let sum = (+.)  
    let max (lst : float list) =  
      match lst with  
      | [] -> None  
      | hd :: tl -> Some (List.fold_right max tl hd)  
  end ;;
```

Important syntax here: `module`, `struct`, `end`. What are these for? Are they analogous to other syntax we've seen so far?

Module Signatures

The type analog for modules is the module signature. For example:

```
# module type TF =  
  sig  
    type info  
    val info : info  
    val hometown : string  
    val print_info : unit -> unit  
    val grade_assignment : int -> string  
    val favorite_function : float -> float -> float  
    val fold : int list -> int -> int  
  end ;;
```

We could then apply this signature to the Sam or Gabbi module.
(Remember, files are module by default!)

```
module TFGabbi = Gabbi : TF ;;
```

Exercise: a BigInt Module.

You may have noticed that

Functors

Binary Heaps
