IIT CS440: Programming Languages and Translators

Homework 0: OCaml Basics

Prof. Stefan Muller
TA: Xincheng Yang

Out: Thursday, Jan. 21
Due: Thursday, Feb. 4

Updated Jan. 22

This assignment contains 4 written tasks and 5 programming tasks, for a total of 75 points.

1 Logistics

Please read and follow these instructions carefully.

1.1 Collaboration

Please see the collaboration policy on the course website. Note that, in your written answers, you must include a statement about who, if anyone, you collaborated with (and in what way), and what outside sources, if any, you consulted. If you didn't collaborate or use outside sources, say that as well. This should include collaborations/citations for both the written and programming problems. Here's an example:

> I discussed alpha equivalence with A and B after class. I also got a little stuck on programming problem 4.2 and talked through a sketch of it with C. I read the Wikipedia article on alpha equivalence to understand it better.

1.2 Submission

Put your answers to the programming problems in the skeleton given in `hw0.ml`. Rename it to something like `Doe_John_440_hw0.ml` before submitting. **Do not alter any of the function headers/signatures we give you, though you may add your own.** Your written answers should be submitted as a pdf file (typeset in LaTeX is preferred, but other formats of pdf are acceptable) named something like `Doe_John_440_hw0.pdf`. Put the two files into a folder named something like `Doe_John_440_hw0` and zip the file to submit. Submit on Blackboard under the correct assignment.

1.3 Programming Problems

We will grade your OCaml files automatically. Follow the following rules to make sure you don't break our autograder:

- Do not alter any of the function headers/signatures in the provided skeleton (including to change types, make non-recursive functions recursive, etc.).
- Each function you're supposed to implement currently has as its body `raise ImplementMe`. This will make it so that the file will compile before you've implemented all of the functions. Remove this (but just this) and replace it with your code.

- Do not alter or remove any lines beginning in `(*>*`.
- Your file **must compile with ocamlc**. Copying and pasting code into the toplevel or TryOCaml can mislead you into thinking that code will compile when the whole file may not. Make sure to run ocamlc (e.g. `ocamlc -o hw0 Doe_John_440_hw0.ml`) before submitting.

1.4 Testing

Like many languages, OCaml has a form of `assert` statement that you can use for debugging and testing.

```
assert e
```

will trigger an exception if `e` evaluates to false, and will do nothing if `e` evaluates to true. You can use this to write unit tests for each function you write. We highly encourage you to write unit tests, using `assert`, for every function you write where possible. Your tests should cover every branch of your code: for example, if you have a recursive function over a list, with two branches (`[]` and `::`), you should have (at least) two tests, one for the empty list and one for a non-empty list. Cover any potentially tricky corner cases as well.

We will not grade you directly on testing, but the amount of testing you do is likely to indirectly impact your grade because tested code is more likely to be correct :). In addition, thinking about the tricky corner cases of your code may also expose cases you may not have thought of, leading you to discover errors in your code or questions about the specification you may want to clarify.

2 Basics, Types and Evaluation

Task 2.1 (Written, 10 points).

For each expression below, state whether it:

- Is not syntactically correct (explain why in ≤ 1 sentence)
- Is syntactically correct, but not well-typed (explain why in ≤ 1 sentence)
- Is syntactically correct and well-typed, but does not evalute to a value (explain why in ≤ 1 sentence)
- Evaluates to a value (give the value)
- Is a value

Try figuring each out yourself, but you can check yourself with OCaml.

a) `42`

b) `"42"`

c) `"47" - "5"`

d) `42/0`

e) `(42/0`

f) `fun x -> x / 0` (*Updated 1/22*: `fun`, not `fn`)

g) `if 42 < 84 { return "good"; } else  { return "bad"; }`

h) `if 42 < 84 then "good" else 0`

i) `(fun x -> x * 2) 21` (*Updated 1/22*: `fun`, not `fn`)

j) `let rec f () = f () in f ()`

Task 2.2 (Written, 10 points).

For each pair of expressions below, state whether or not they are α-equivalent (recall that α equivalence means you can rename variables to get the same expression). Explain why in ≤ 1 sentence.

a)
```
let x = 1 in let y = 2 in x + y
```
$\overset{?}{=}_{\alpha}$
```
let y = 1 in let x = 2 in y + x
```

b)
```
let x = 1 in (let x = 2 in x + x)
```
$\overset{?}{=}_{\alpha}$
```
let x = 1 in (let y = 2 in x + y)
```

c)
```
let x = 1 in (let x = 2 in x * 2) + x
```
$\overset{?}{=}_{\alpha}$
```
let y = 1 in (let z = 2 in z * 2) + y
```

d)
```
let x = 1 in (let y = 2 in y * x) + (let y = 3 in y * x)
```
$\overset{?}{=}_{\alpha}$
```
let x = 1 in (let a = 2 in a * x) + (let b = 3 in b * x)
```

e)
```
let x = 1 in let y = 2 in x
```
$\overset{?}{=}_{\alpha}$
```
let x = 1 in let x = 2 in x
```

3 Types and polymorphism

Task 3.1 (Written, 20 points).

Give the most general types of the following functions. Again, try figuring each out yourself, but you can check yourself with OCaml.

a) `let f1 (x, y) = (y, x)` (Hint: you may need two type variables)

b) `let f2 x y = [[x]; y]`

c) `let f3 x y z = [x::z; y::z]`

d) `let f4 x y = [x; x + y]`

e) `let rec f5 x = match x with [] -> [] | x::t -> [x; x]::(f5 t)`

4 Lists and Recursion

Task 4.1 (Programming, 5 points).

Write a function `stutter: int -> 'a -> 'a list`: the call `stutter n x` should result in a list with `x` repeated `n` times, e.g.

- For all `x`, `stutter 0 x = []`
- `stutter 2 5 = [5; 5]`
- `stutter 3 "yadda" = ["yadda"; "yadda"; "yadda"]`

Task 4.2 (Programming, 5 points).

Write a function `filter: ('a -> bool) -> 'a list -> 'a list`. The call `filter f l` should return `l` with all and only the items for which `f` returns true; these items should appear in the same order in which they appear in the original list.

Some examples:

- `filter (fun x -> x > 2) [5; 3; 1; 2; 4] = [5; 3; 4]`
- `filter (fun x -> x > 5) [5; 3; 1; 2; 4] = []`

Task 4.3 (Programming, 5 points).

Write a function `find: ('a -> bool) -> 'a list -> 'a option` that takes the same arguments as `filter`. If `x` is the first element in the list for which `f x = true`, then `find` should return `Some x`. If there is no such element, it should return `None`.

Some examples:

- `find (fun x -> x > 2) [5; 3; 1; 2; 4] = Some 5`
- `find (fun x -> x < 3) [5; 3; 1; 2; 4] = Some 1`
- `find (fun x -> x > 5) [5; 3; 1; 2; 4] = None`

Task 4.4 (Programming, 8 points).

One important syntactic task of programming in most languages is making sure your parentheses match! Write a function `parens` that takes a list of characters including open parens `'('` and close parens `')'`. It should return true if and only if all parentheses are correctly matched.

Hints/additional specifications:

- You will probably want a recursive helper function that scans the list left to right and tracks some additional information.
- If you reach the end of the list and there are unmatched open parens, the parens are not matched.
- If you see a close paren that doesn't match an open paren, the parens are not matched.
- A list with no parentheses is considered correctly matched.
- You can ignore any characters other than `'('` and `')'`.

Continued on next page

Task 4.5 (Programming, 12 points).

Believe it or not, possibly the hardest part of writing a compiler is producing good error messages (if you've ever used a new language or compiler and found the error messages frustratingly unhelpful, this is probably why!) An important part of this is identifying *where* in the code the error occurs (generally line and column number). Imagine how frustrating the following would be when you're trying to compile a large program:

```
$ ocamlc mycode.ml
Syntax Error.
$
```

While you won't be finding syntax errors in a program (yet!), you will start by writing a function similar to `find` above, but which identifies *where* the found item occurs in a list. It also finds an item in a list of lists, representing a two-dimensional array of characters (like a program written as a text file).

Implement the function `find2D: ('a -> bool) -> 'a list list -> (int * int) option`. If `file` contains a list of rows, where each row is a list of characters, `find2D f file` should return `Some (x, y)` where `x` is the row and `y` is the character of the first item for which `f` returns true, or return `None` if there is no such item.

Both rows and columns (characters) should be indexed from 0. When determining which item is "first", you should read from left to right within each row, then move to the next row (just like reading).

As an example, if `file` is

```
[[1;2;3;4;5];
 [5;4;3;2;1];
 [10;9;8;7;6];
 [2;4;6;8;10;12]]
```

then

- `find2D ((=) 5) file = Some (0, 4)` because the first 5 appears in row 0 at character 4.
- `find2D ((=) 15) file = None` because 15 does not appear in the file.

Task 4.6 (Written, 0 points).

How long (approximately) did you spend on this homework, total? Your honest feedback will help us with future homeworks.