CMSC 330, Summer 2019 — Midterm Solution

Name _____

Instructions

- Do not start this exam until you are told to do so.
- You have 80 minutes for this exam.
- This is a closed book exam. No notes or other aids are allowed.
- For partial credit, show all your work and clearly indicate your answers.

Honor Pledge

Please copy and sign the honor pledge: "I pledge on my honor that I have not given or received any unauthorized assistance on this examination."

_____

_____

_____

_____

Section	Points
Programming Language Concepts	10
Regular Expressions	8
Ruby Execution	10
Ruby Programming	16
OCaml Typing	10
OCaml Execution	12
OCaml Programming	18
Finite Automata	16
Total	100

Programming Language Concepts

1. [3 pts] Imagine a parallel universe in which Ruby is *statically typed*. Give the type of the `simple` method or if there is a type error, explain why there is one and if the same program would error with real Ruby.

```
def simple(x)
  if x == x then
    1
  else
    false
  end
end
```

Solution. The `true` branch of the conditional has `Integer` type, but the `false` branch has `Boolean` type. This is a type error. The same program would not error with real Ruby since the method is not typechecked. ❑

2. [3 pts] It also happens that in this universe Ruby has *no mutable state*. Variable assignment acts like OCaml's `let ... in ...` construct. What should be the result of the following piece of code and why? Does this return the same result as regular Ruby?

```
def sum(xs)
  total = 0
  xs.each do |x|
    total = total + x
  end
  total
end


sum([1, 2, 3])
```

Solution. This should return `0` since the assignment inside the `each` doesn't mutate `total`. It's scoped locally to the loop like OCaml's `let`. With real Ruby this would mutate `total` and hence return `6`. ❑

3. [3 pts] In our parallel universe OCaml is *dynamically typed*. What should the following code produce? If it errors, explain why. What would this code do under regular OCaml?

```
let simple x =
  if x = x then
    0
  else
    1 + (fun x -> x)
in simple 1
```

Solution. This should return `0` since the `else` branch is never taken. With real OCaml this would result in a type error since the `else` branch attempts to add an `int` and a function. ❑

4. [1 pts] What's your favorite programming language? (Any programming language is an acceptable answer here.)

Regular Expressions

1. [3 pts] Write a Ruby regular expression that *exactly* accepts strings containing zero or more occurrences of the words pie and jam separated by either a `,` or a `;`.

```
> foods?("pie,jam")
true
> foods?("pie,pie;pie,pie")
true
> foods?("jam&jam")
false
```

Fill in the blank below with your regular expression.

```
def foods?(str)
   str =~ /  ^((pie|jam)([,;](pie|jam))*)?$  /
end
```

2. [3 pts] Write a Ruby regular expression that accepts strings containing an odd number of a's.

```
> odd_as?("aardvark")
true
> odd_as?("sour")
false
```

Fill in the blank below with your regular expression.

```
def odd_as?(str)
   str =~ /  ^[^a]*a([^a]*a[^a]*a)*[^a]*$  /
end
```

3. [2 pts] Given the regular expression `/w{3}.\w+\d?$/` circle all of the strings that would be accepted.

   (a) www.google.com5

   (b) wwwumdedu

   (c) www.cmsc330com

Solution. (b), (c) ❑

Ruby Execution

Next to each Ruby snippet, write the output after executing it. If there is an error, then write "error."

1. [2 pts]

```
cmsc = [131, 132, 216, 250, 330, 351]
cmsc.reverse!
puts(cmsc[0])
```

Solution.

```
351
```

❑

2. [2 pts]

```
hash = Hash.new(-1)
ruby = "Ruby"
ocaml = "OCaml"
hash[ruby] = 3
hash[ocaml] = 4

puts(hash[ocaml + ruby])
puts(hash[ocaml] + hash[ruby])
```

Solution.

```
-1
7
```

❑

3. [3 pts]

```
def together(arr)
  str = ""
  arr.each do |x|
    str += x.to_s
  end
  str
end

puts(together([1, 2, 3, 4, 5]))
```

Solution.

```
12345
```

❑

4. [3 pts]

```
def double(val)
  yield(2 * val)
end

arr = ["Alpha", "Beta", "Charlie"]
double(1) { |x| puts(arr[x]) }
```

Solution.

```
Charlie
```

❑

Ruby Programming

You've recently been hired by the newest pizzeria in town and the owner has asked you to create a way to take orders and pass them along to the delivery driver. Each order comes as a string that contains pizza toppings and the associated address. Due to huge success, each address is limited to a single order at a given time, so if a new order comes from the same address, the old order is replaced.

1. [2 pts] Implement the `initialize` method. Decide on a data structure(s) to store the needed information.

2. [6 pts] Implement the `take_order` method. The method will take in a string that is formatted to contain a list of toppings followed by an address. The toppings should contain only alphabetic characters and are separated by single spaces. The address should be a single non-negative four digit number. Because it is a small town, there is only one street (Main Street), going straight through town. Don't add an order if the input string is malformed.

   ```
   > take_order("Pepperoni Spinach 3900")
   > take_order("Cheese Pepperoni Peppers Mushrooms Spinach 4131")
   > take_order("Cheese Ham Pineapple 7000")
   > take_order("Cheese 9001")
   ```

3. [4 pts] Implement the `give_order` method that takes a current address (as a string) and returns (again as a string) the closest address with a pending order. If there are no orders it should return `false`.

   ```
   > give_order("4000")
   "3900"
   > give_order("6000")
   "7000"
   ```

4. [4 pts] Your delivery driver can handle up to three orders at once. The owner wants the driver to always go to the nearest location from their current location. Implement the `route` method to return (as a list of strings) the three closest orders, in order of how close they are to the most recent delivery—starting from the pizzeria at 5000 Main Street. These orders will then be removed from the list of pending orders.

   ```
   > route()
   ["4131", "3900", "7000"]
   > route()
   ["9001"]
   > route()
   []
   ```

Implement your solution in the class skeleton below.

Solution.

```
class Pizzeria
  def initialize()
    @orders = {}
  end

  def take_order(order)
    if order =~ /^([A-z]+ )*(\d{4})$/ then
      @orders[$2] = $1.split(" ")
    end
  end

  def give_order(location)
    return false if @orders.empty?
    @orders.keys.min_by { |x| (x.to_i - location.to_i).abs }
  end

  def route()
    arr = []
    location = "5000"
    3.times do
      location = give_order(location)
      if location then
        arr << @orders.delete(location)
      else
        break
      end
    end
    arr
  end
end
```

❑

OCaml Typing

Recall the definition of `'a option`.

```
type 'a option = Some of 'a | None
```

Determine the type of the following expressions. If there is an error, write "error" and give a brief explanation.

1. [2 pts]

   `fun x -> [5] :: x`

   Solution. `int list list -> int list list` ❑

2. [2 pts]

   `[3; 4.0]`

   Solution. Type error. ❑

3. [2 pts]

   `let f x y z = x +. y +. z in f 1.0 2.0`

   Solution. `float -> float` ❑

Write an expression that has the following type, without using type annotations.

4. [2 pts] `int option -> int list`

   Solution. `fun (Some x) -> [x + 1]` ❑

5. [2 pts] `'a list -> 'a list list`

   Solution. `fun (_ :: xt) -> [xt]` ❑

OCaml Execution

Recall the definitions of `map`, `fold_left`, and `fold_right`.

```
let rec map f xs =
  match xs with
  | [] -> []
  | x :: xt -> (f x) :: (map f xt)
```

```
let rec fold_left f a xs =
  match xs with
  | [] -> a
  | x :: xt -> fold_left f (f a x) xt
```

```
let rec fold_right f xs a =
  match xs with
  | [] -> a
  | x :: xt -> f x (fold_right f xt a)
```

Write the final value of the following OCaml expressions next to each snippet. If there is an error, write "error."

1. [2 pts]

```
let x = 3 in
let f x y = x + y in
f 5 2
```

Solution. `7` ❑

2. [3 pts]

```
let f a x = x :: a in
fold_left f [] [1; 2; 3]
```

Solution. `[3; 2; 1]` ❑

3. [3 pts]

```
map (fun a x ->
       if (x mod 3) = 0 then
         x :: a
       else
         a)
    [1; 3; 11; 27]
```

Solution. Error. ❑

4. [4 pts]

```
let f x y z =
  if y = x then
    y :: z
  else
    z
in fold_right (f 3) [1; 2; 3; 3; 2; 1] []
```

Solution. `[3; 3]` ❑

OCaml Programming

Consider the `argmin (f : (int -> int)) (xs : int list) : int` function that returns the last element of the list `xs` that minimizes the given function `f`. You may assume the function only operates on non-empty lists. Here are some examples:

```
# argmin (fun x -> x) [0; 2; 1]
0
# argmin (fun x -> -x) [0; 2; 1]
2
# argmin (fun _ -> 0) [0; 2; 1]
1
```

1. [6 pts] Fill in the following blanks to correctly implement `argmin`.

```
let argmin (f : int -> int) ((x :: xt) : int list) : int =
  List.fold_left
    (fun acc x ->
       if [          (f x) <= (f acc)          ] then
          [                x                ]
       else
          [                acc              ] )
    [ x ]
    [ xt ]
```

2. [2 pts] Suppose the list `xs` has length n. In terms of n, **exactly** how many times is the function `f` evaluated in the above implementation of `argmin`?

Solution. $2(n-1)$ ❑

3. [8 pts] Fill in the following blanks to correctly implement `argmin`.

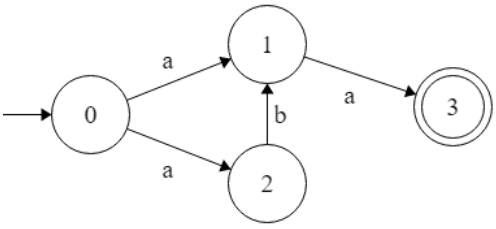
```
let argmin (f : int -> int) ((x :: xt) : int list) : int =
  let (min_x, _) =
    List.fold_left
      (fun (x', y') x ->
         let y =  f x   in
         if  y <= y'  then
          (x, y)
         else
          (x', y')  )
    (x, f x)
    xt
  in min_x
```

4. [2 pts] Suppose the list `xs` has length n. In terms of n **exactly** how many times is the function `f` evaluated in this new implementation of `argmin`?

Solution. n ❑

Finite Automata

1. [2 pts]



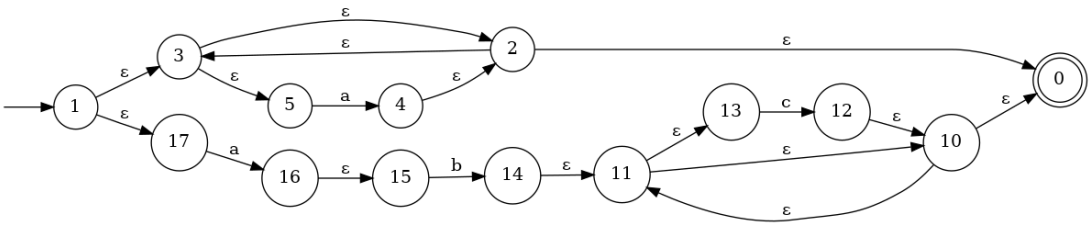
Is the above finite automaton an NFA or DFA? (Select the most specific option.)

Solution. It's an NFA, but not a DFA. ❑

2. [6 pts] Draw a finite automaton (either an NFA or a DFA) that accepts the same strings as the regular expression:

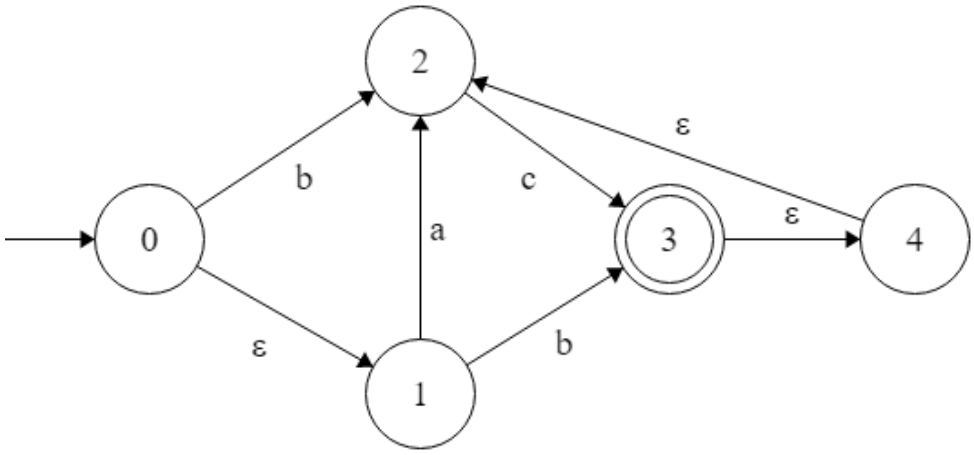
$$(abc^\star)|(a^\star)$$

Solution.

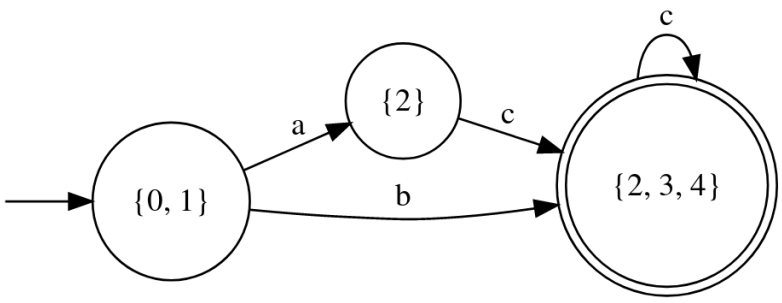


❑

3. [8 pts] Convert the following NFA to a DFA using the subset construction algorithm. Label your states with their appropriate subsets.



Solution.



❑