CMSC 330 Spring 2020 Final Exam Solutions

Q1 Introduction

0 Points

[omitted]

Q2 T/F - OCaml tuples

1 Point

OCaml tuples are similar to C structs, in that they are both fixed-sized collections of data.

True

Q3 T/F - Rust Lifetimes

1 Point

In Rust, a reference can have a longer lifetime than the value it is referencing.

False

Q4 T/F - Lambda CBV vs CBN

1 Point

The end result of a lambda calculus reduction that terminates will always be the same, regardless of whether it was reduced under call-by-name or call-by-value evaluation.

True

Q5 Weak vs Strong Typing

4 Points

Describe an operation that could be performed in a *dynamically-typed* language, but not in a *statically-typed* one.

- **Reassigning a variable to a differently-typed value**
- **Conditionals based on the type of a value at runtime**

Q6 Ruby vs OCaml Memory Management

4 Points

Describe the similarities between memory management in Ruby and OCaml, and provide one benefit of their approach.

Automatic garbage collection, which makes coding easier and prevents most accidental memory leaks or use-after-free.

Q7 Rust References

4 Points

You are writing a Rust program and include the following String definition

```
let my_string = String::from("Hello, world!");
```

Which of the following scenarios is valid? Select all that apply.

- Having a single mutable reference to my_string
- **Having 10 non-mutable references to my_string at the same time**
- Having a single mutable reference and a single non-mutable reference at the same time.

Q8 RegExp Construction

3 Points

Write a regular expression that matches strings which are a comma-separated list of one or more *words*. A *word* is defined (for this question) as an underscore followed by one or more uppercase/lowercase English letters. There should be no spaces between words.

The following are examples of strings which should match:
`_cmsc,_three,_thirty`
`_hello`

The following are examples of strings which should **not** match:
`_a, _b` (no spaces allowed)
`_cmsc330!` (no numbers/symbols)
` ` (must be at least one word)

`/^_[A-Za-z]+(,_[A-Za-z]+)*$/`

Q9 DFA Construction

3 Points

Construct a **DFA** (not an NFA) that accepts strings consisting of the characters `3` and `6`, and have the property that the sum of all digits in the string is **exactly** 9. You can write your DFA in the format from project 3, or upload a picture.

```
let dfa_ex = {
    sigma = ['a'; 'b'; 'c'];
    qs = [0; 1; 2];
    q0 = 0;
    fs = [2];
    delta = [(0, Some 'a', 1); (1, Some 'b', 0); (1, Some 'c', 2)]
}
```

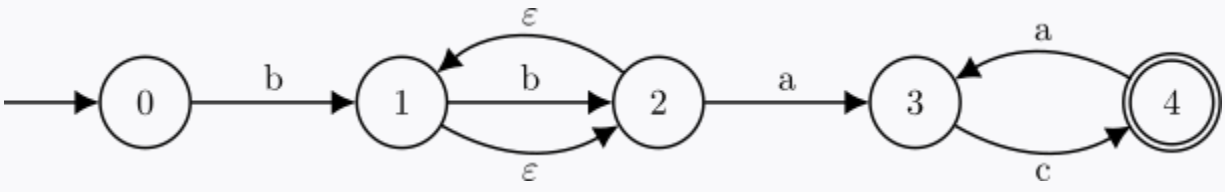
Note that the DFA above is an example purely for formatting, its actual contents are not related to this problem.

```
let dfa_ex = {
    sigma = ['3'; '6'];
    qs = [0; 1; 2; 3];
    q0 = 0;
    fs = [3];
    delta = [(0, Some '3', 1); (0, Some '6', 2); (1, Some '3', 2);
             (1, Some '6', 3); (2, Some '3', 3)]
}
```

Q10 NFA → RegExp

2 Points

Convert the following NFA to a regular expression




```
/^b+(ac)+$/
```

Q11 CFG Construction

3 Points

Construct a CFG that generates strings of the form $a^x b^x c^y$, where $x \geq 0,\ y > 1$.

S → AB
A → aAb | ε
B → cB | c

Q12 CFG → RegExp

3 Points

Write a regular expression equivalent to the following CFG:

```
S → A | B
A → Aa | a
B → Bb | ε
```

/^a+|b*$/

Q13 Operational Semantics Fill-in-the-Blank

3 Points

Using the provided rules, indicate what should be written in place of each of the 5 marked areas.

$$\frac{}{A; n \to n} \qquad \frac{A(x) = v}{A; x \to v} \qquad \frac{A; e_1 \to v_1 \qquad A, x : v_1; e_2 \to v_2}{A;\ \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \to v_2}$$

$$\frac{A; e_1 \to n_1 \qquad A; e_2 \to n_2 \qquad n_1 > n_2}{A; e_1 > e_2 \to \textbf{true}} \qquad \frac{A; e_1 \to n_1 \qquad A; e_2 \to n_2 \qquad n_1 \leq n_2}{A; e_1 > e_2 \to \textbf{false}}$$

$$\frac{A; e_1 \to \textbf{true} \qquad A; e_2 \to v}{A;\ \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \to v} \qquad \frac{A; e_1 \to \textbf{false} \qquad A; e_3 \to v}{A;\ \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \to v}$$

$$\frac{\dfrac{}{A, \boxed{2}} \qquad \dfrac{\dfrac{\dfrac{}{A, \boxed{5}}}{A, x : 5; x \to 5} \qquad \dfrac{}{A, x : 5; 3 \to 3} \qquad \boxed{4}}{A, x : 5; x > 3 \to \textbf{true}} \qquad \dfrac{}{A, \boxed{3}}}{A, x : 5;\ \textbf{if}\ x > 3\ \textbf{then true else false} \to\ \textbf{true}}}{A;\ \textbf{let}\ \boxed{1}\ \textbf{in if}\ x > 3\ \textbf{then true else false} \to \textbf{true}}$$

Make sure to clearly label each part.

1. **x = 5**
2. **5 → 5**
3. **x : 5; true → true**
4. **5 > 3**
5. **x : 5 (x) = 5**

Q14 Operational Semantics Mystery Operator

3 Points

Given the following operational semantics for **myst**, which logical operator does **myst** represent?

$$\frac{A; e_1 \to \textbf{true} \qquad A; e_2 \to \textbf{true}}{A; e_1 \ \textbf{myst} \ e_2 \to \textbf{true}} \qquad \frac{A; e_1 \to \textbf{true} \qquad A; e_2 \to \textbf{false}}{A; e_1 \ \textbf{myst} \ e_2 \to \textbf{false}}$$

$$\frac{A; e_1 \to \textbf{false} \qquad A; e_2 \to \textbf{true}}{A; e_1 \ \textbf{myst} \ e_2 \to \textbf{false}} \qquad \frac{A; e_1 \to \textbf{false} \qquad A; e_2 \to \textbf{false}}{A; e_1 \ \textbf{myst} \ e_2 \to \textbf{false}}$$

Logical AND, conjunction, &&

Q15 Lambda Calculus - Alpha Conversions

4 Points

For each pair of λ-expressions, indicate whether they are α-equivalent.

Q15.1

1 Point

λx.(λx.x) x vs λy.(λx.x) y

Equivalent

Q15.2

1 Point

λx.(λx.x) vs λy.(λx.y)

Not equivalent

Q15.3

1 Point

λx.(λy.y) y vs λx.(λz.z) y

Equivalent

Q15.4

1 Point

λz.(λy.(λx.x) x y) z vs λz.(λx.(λx.x) x x) z

Not equivalent

Q16 Lambda Calculus - Beta Reduction

8 Points

Reduce each of the following λ-expressions to β-normal form. Be sure to show each reduction step for full credit. If the expression reduces infinitely, write "infinite reduction." You may upload an image if you prefer to write the answers by hand, but please include both answers in a single image, labeled and in order.

(λy.(λx.x x) y) (λz.x y z) a
(λx.x x) (λz.x y z) a
(λz.x y z) (λz.x y z) a
x y (λz.x y z) a

(λb.((λx.x x) (λx.x x)) b) a b
((λx.x x) (λx.x x)) a b
infinite reduction

Q17 Lambda Calculus OCaml Representation

2 Points

Given the following AST data type in OCaml:

```
type id = string
type exp = Var of id
         | Lam of id * exp
         | App of exp * exp
```

For example, λx.x y would be represented as

```
Lam("x", App(Var("x"), Var("y")))
```

Write out the representation (in OCaml) of λx.a b λy.x y a.

```
Lam("x", App(App(Var("a"), Var("b")), Lam("y",
  App(App(Var("x"), Var("y")), Var("a")))))
```

Q18 CBN vs CBV

2 Points

Name one advantage of call-by-name evaluation over call-by-value evaluation.
CBN will not waste time evaluating arguments that it does not use

Q19 Evaluation

4 Points

You are given the following Abstract Syntax Tree (AST) OCaml type:

```
type expr = Plus of expr * expr
          | Mult of expr * expr
          | Sub  of expr * expr
          | Int  of int
```

Where `Plus`, `Mult`, and `Sub` represent addition, multiplication, and subtraction respectively; and `Int` represents an integer.

For example, the expression $1 * (2 + 3) - 4$ would be represented as

```
Sub(
  Mult(
    Int(1),
    Plus(
      Int(2),
      Int(3)
    )
  ),
  Int(4)
)
```

Write a function `eval` that takes an expression and evaluates it to a value. Here are some examples:

```
eval Int(7) = 7
eval Plus(Int(1), Int(2)) = 3
eval Sub(Int(3), Int(7) = -4
eval Plus(Mult(1, 2), Int(3)) = 5
```

You may enter your code in the box below, or upload a file.

```
let rec eval ast = match ast with
  | Int(i) -> i
  | Plus(a, b) -> (eval a) + (eval b)
  | Mult(a, b) -> (eval a) * (eval b)
  | Sub(a, b) -> (eval a) - (eval b)
```

Q20 Attacks in C vs Rust

2 Points

Name and/or describe one type of attack that is possible in C, but not in Rust, and give a brief explanation of why it is not possible in Rust.

- **Buffer overflows: all strings have a length component, not terminated by a null byte**
- **Double free/use-after-free/memory leak: memory is deallocated automatically when there are no more references to it.**

Q21 Input Validation

2 Points

What is one issue/challenge with validating input by whitelisting.

You have to know all possible valid inputs beforehand, and it is easy to forget some which will cause valid inputs to be rejected.

Q22 Cross-Site Scripting

4 Points

Briefly describe how a reflected XSS attack is carried out.

The user enters JavaScript code in an input field (e.g. a search box), and when the next page loads, it renders the contents of that input field, along with any HTML elements and scripts that were entered into it.

Q23 Blacklisting vs Sanitizing

4 Points

Describe the difference between blacklisting and escaping input in the context of input sanitization.

Blacklisting: removing bad characters from input
Escaping: keeping bad characters, but adding escape chars (usually \) to force them to be recognized as data rather than code.

Q24 SQL Injection

8 Points

The backend of bank.com's server uses an SQL database. The following Ruby code is used on the backend server to verify withdrawal transactions.

```
def verify_withdraw(username, password, amount)
    # Get the old balance
    curr_balance = db.execute "SELECT balance FROM accounts
        WHERE username='#{username}' AND password='#{password}';"
    # Update with new balance
    db.execute "UPDATE accounts SET balance='#{curr_balance - amount}'
        WHERE username='#{username}' AND password='#{password}';"
end
```

Q24.1 Exploit the Vulnerability

4 Points

Assume that a user named `bob` exists in the database. What inputs to the username, password, and amount fields would allow you to withdraw money from `bob`'s account without knowing his password? You can leave a field blank if you don't think it's necessary to enter anything. Assume no validation takes place, and the password is not hashed.

username:
bob';--

password:
whatever

amount:
100 (any numerical amount, as long as it is valid and does not comment out username)

Q24.2 Fix the Vulnerability

4 Points

What are two techniques that could be used to fix this vulnerability?

- **Prepared statements**
- **Sanitizing input (blacklisting or escaping)**
- **Blacklisting bad inputs (rejecting them altogether)**
- **Whitelisting (list of allowed characters for username and password)**

Q25 OCaml Typing

2 Points

Write an OCaml expression with the following type without using type annotations. All pattern matching must be exhaustive.

```
('a -> int) -> 'a list -> int
```

```
fun a b -> match b with
  | [] -> 0
  | h::_ -> a h
```

Q26 OCaml Typing

2 Points

Write an OCaml expression with the following type without using type annotations. All pattern matching must be exhaustive.

```
int list -> a' -> (a' -> b') -> (b' -> string) -> (int * string)
```

```
fun a b c d -> match a with
  | [] -> (0, "")
  | h::t -> (h, d (c b))
```

Q27 OCaml Coding - Family Tree

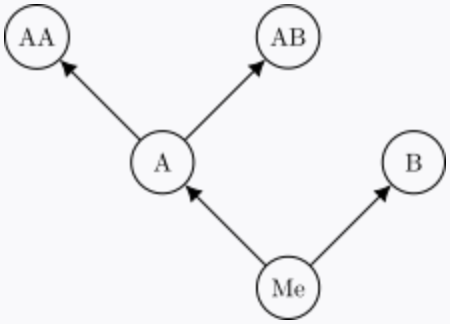
7 Points

Given that we define a binary tree type called `family_tree`, write the `find_gens_older` function which takes a `family_tree` and returns an in-order traversal of all family members removed from the root by greater than or equal to `num_gens` generations (the root is generation 0). You may make the function recursive and/or write recursive helper functions if you would like. The `@` operator is allowed. Assume that the tree is non-empty.

```
type family_tree = Member of family_tree * string * family_tree
                 | Nil
```

Example:

Consider the family tree shown below:



The representation of this tree is shown with the variable `t` below:

```
let t = Member(
          Member(
            Member(Nil, "Grandparent AA", Nil),
            "Parent A",
            Member(Nil, "Grandparent AB", Nil)
          ),
          "Me",
          Member(Nil, "Parent B", Nil)
        )

(* Examples *)
find_gens_older t 0 = ["Grandparent AA"; "Parent A"; "Grandparent AB";
                       "Me"; "Parent B"]
find_gens_older t 1 = ["Grandparent AA"; "Parent A"; "Grandparent AB";
                       "Parent B"]
find_gens_older t 2 = ["Grandparent AA"; "Grandparent AB"]
find_gens_older t 3 = []
```

Note that this is drawn upside-down to more closely match the structure of a family tree.

You may enter your code in the box below, or upload a file.

```
let rec find_gens_older tree num_gens =
  let rec helper tree level = match tree with
    | Nil -> []
    | Member(l, m, r) -> let l = (helper l (level + 1)) in
                         let r = (helper r (level + 1)) in
                         if level >= num_gens then
                           l @ [m] @ r
                         else
                           l @ r
  in helper tree 0
```

Q28 OCaml Coding - Increasing Sums

5 Points

Write a function called `increasing_sums` that takes a two-dimensional array of integers, and determines whether the sums of each sublist are increasing. That is, the sum of all elements in the second sublist are greater than or equal to the sum of all elements in the first, and so on. You may **not** use the `rec` keyword, or any function in the `List` module. You may use the two functions provided below. You may define any non-recursive helper functions you would like.

```
let rec fold_left f a l =
  match l with
  | [] -> a
  | h::t -> fold f (f a h) t

let rec map f l =
  match l with
  | [] -> []
  | h::t -> (f h)::(map f t)
```

Example:

```
increasing_sums [[-3]; []; [2; 3]; [-1; 6]; [7]; [3; 10; -2]] = true
increasing_sums [[1]] = true
increasing_sums [] = true
increasing_sums [[4; 3]; [5]] = false
```

You may enter your code in the box below, or upload a file.

```
let sum lst = fold_left (+) 0 lst

let increasing_sums lst = match lst with
  | [] -> true
  | h::t -> let sums = map sum t in
            fst (fold_left (fun (flag, prev) x ->
                        flag && x >= prev, x) (true, sum h) sums)
```

Q29 Ruby Programming - Grade Calculation

9 Points

You are the head TA for CMSC331, and the final exam grade deadline is tomorrow night. You're scrambling to get the grades in, but fortunately your fellow TAs have compiled a file of results for you, which contains lines of the form

`Firstname Lastname, M1_score, M2_score, FE_score, Extra Credit`

For example:

```
Anwar Mamat, 40, 80, 92, E.C.
Roger Eastman, 100, 80, 20
```

First and last names must begin with a capital letter, which is followed by zero or more lowercase letters. You can assume that every line is valid, and contains a name, both midterm scores, and a final exam score, all integers within the range 0-100. Some students will have extra credit, denoted `E.C.` at the end of the line, which is worth one extra point. Each part is separated by a comma and a space, as shown above.

You may assume all files are well-formed, following this format precisely.

Q29.1 File Parsing

5 Points

Write the function `addGrades`, which will open the file called "grades.txt" and process each of the lines by storing the relevant pieces of information in the `@grades` hash. Here is starter code for you (**please copy this code to your answer and add to it**):

```
@grades = {}
def addGrades
    File.foreach("grades.txt") do |line|
        # TODO: your code here
    end
end
```

```
@grades = {}
def addGrades
  File.foreach("grades.txt") do |line|
    if line =~ /^(.*), (\d+), (\d+), (\d+)(?:, (E.C.))?$/
      @grades[$1] = [$2.to_i, $3.to_i, $4.to_i, $5 == "E.C." ? 1 : 0]
    end
  end
end
```

Q29.2 Grade Calculation

4 Points

Now that you have stored the data in the `@grades` hash, write the function `finalGrades`, which will return a new hash with students' names mapped to their final grades. The actual computation of an individual's final grade will be handled by a code block that is passed into the function. A example call to `finalGrades` is shown below:

```
finalGrades do |e1, e2, f, ec|
   [[e1, e2].sum / 2, f].max + ec
end
```

where `e1` is the first midterm score, `e2` is the second midterm score, `f` is the final exam score, and `ec` is either 1 or 0 depending on whether the student received extra credit. The function should return a new hash mapping names to final grades.

Using the example file from above as a reference, this call to `finalGrades` should output the following hash:

```
{"Anwar Mamat" => 93, "Roger Eastman" => 90}
```

Note: you should not be performing any mathematical computation in your function. Instead, pass the four values to the code block and use the final grade returned by it. You can assume that a code block was given.

Here is starter code for you (**please copy this code to your answer and add to it**):

```
def finalGrades
    # TODO: your code here
end
```

```
def finalGrades
  fin = {}
  @grades.each do |k, v|
    fin[k] = yield v[0], v[1], v[2], v[3]
  end
  fin
end
```