CMSC330 Fall 2017 Final Exam
Solution

Name (PRINT YOUR NAME IN ALL CAPS):

_____

Discussion Time (circle one) **10am 11am 12pm 1pm 2pm 3pm**

Discussion TA (circle one) **JT Greg Justin Michael BT Daniel David Derek**

Cameron Eric Kesha Shriraj Pei-Jo Michael Bryan Kameron

Instructions

- The exam has 17 pages (front and back); make sure you have them all.
- Do not start this test until you are told to do so!
- You have 120 minutes to take this exam.
- This exam has a total of 130 points, so allocate 55 seconds for each point.
- This is a closed book exam. No notes or other aids are allowed.
- Answer essay questions concisely in 2-3 sentences. Longer answers are not needed.
- For partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit cannot be given for illegible answers.

#	**Problem**	**Score**
1	PL Concepts	/10
2	Lambda Calculus	/10
3	OCaml Typing	/10
4	OCaml Execution	/10
5	OCaml Programming	/12
6	Ruby	/16
7	Prolog	/20
8	Regexps, FAs, CFGs	/18
9	Parsing	/8
10	Operational Semantics	/6
11	Security	/10
	TOTAL	/130

1 PL Concepts [10 pts]

1. **[8 pts]** Circle True or False for each of the following statements (1 point each)

    1. True / **False** Structural ("deep") equality implies physical equality

    2. **True** / False In Prolog, green cuts can be used to eliminate redundant computations

    3. True / **False** Static type checking occurs during run time in OCaml

    4. True / **False** Both Ruby and OCaml have type inference

    5. True / **False** In SmallC, the lexing stage outputs an abstract syntax tree

    6. **True** / False Mark and sweep garbage collection can collect cyclic data structures

    7. True / **False** In general, static type systems are sound and complete

    8. **True** / False In lambda calculus, some beta reductions may never terminate

2. **[2 pts]** Fill in the blank

    A ___**Module Signature**_____ in OCaml can be used to hide the implementation details of a module.

2 Lambda Calculus [10 pts]

1. **[2 pts]** Make the parentheses explicit in the following expression

```
(   λx.  y   x   y   λa.  a   b   )
```

(λx.(y x y (λa.(a b))))

2. **[3 pts]** Beta reduce the following expression until it reaches normal form. Show each step of the reduction.

```
(λx.λy.x y) (λy.y)  x  (t a)
```

(λx.λa.x a)(λy.y)(x)(t a)
(λa.(λy.y) a)(x)(t a)
((λy.y) x)(t a)
(x)(t a)

3. **[5 pts]** Here are the Church encodings for boolean conditionals:

`true`	`λx.λy.x`
`false`	`λx.λy.y`
`if a then b else c`	`a b c`

Give the lambda calculus encoding of the boolean "or" function. For example, **`or true false`** should reduce to **`true`,** and **`or false false`** should reduce to **`false`**. To make it more clear, do not substitute true, false with their lambda encodings. (hint: your answer should look like `or = λa.λb. some application of a and b here`)

`λa.λb.a a b`

3 OCaml Types [10 pts]

1. Write the types of the following OCaml expressions or write “type error” if the expression has no type:

   1) **[2 pts]**

```
[(“I”,4.0); (“R”,0.0); (“S”,1)]
```

   Type Error

   2) **[2 pts]**

```
fun a -> fun b -> (a b) + 1
```

   (‘a -> int) -> ‘a -> int

2. Provide expressions (without type annotations) that have the following types:

   1) **[3 pts]**

```
int -> int list -> bool list
```

   fun i l -> List.map ((=) (i + 1)) l;;

   2) **[3 pts]**

```
(’a->’b) -> (’c->’a) -> ’c -> ’b
```

   fun a b c -> a (b c)

4 OCaml Execution [10 pts]

Code for map and fold is provided for reference:

```
let rec map f l = match l with
 [] -> []
 | h::t -> let r = f h in r :: map f t;;

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

What is the output of the following OCaml expressions?

1. **[3 pts]**

```
fold (fun a x -> if x mod 2 = 0 then a else x::a) [] [1; 2; 3; 4; 5];;
```

[5;3;1]

2. **[3 pts]**

```
type taxpayer = Person of float
              |Company of float;;
let tax y =
  let income_tax a x = a *. x in
    match y with
     |Person i -> income_tax i 0.1
     |Company j -> income_tax j 0.2
in
  (tax (Person 100.0), tax (Company 200.0));;
```

(10.0, 40.0)

3. **[4 pts]**

```
let f =
  let c = ref 0 in
      fun a x -> c:= x+a; !c in
fold f 0 [1;2;3];;
```

6

5 OCaml Programming [12 pts]

A matrix is a two-dimensional list of floats defined as:

```
type row = float list
type matrix = row list
```

1. **[6 pts]** Write a function that, given matrix `m` and ints `i` and `j`, returns the element at row `i` and column `j`. If `i` or `j` is out of bounds, return `None`. For example:

```
let m = [[1.0 ; 2.0]; [3.5 ; 4.5]] in
get_element m 0 0 → Some 1.0
get_element m 0 1 → Some 2.0
get_element m 1 0 → Some 3.5
get_element m 3 0 → None
```

Write your answer below. (Hint: Write a helper function to get an element at an index from a list).

```
let rec get_element (m : matrix) (i : int) (j : int) : float option =
match (at_index m i) with
| None -> None
| Some r -> at_index r j

and at_index (m : matrix) (lst : 'a list) : 'a option =
match lst with
| [] -> None
| h::t -> if i = 0 then (Some h) else (at_index t (i-1))
```

2. **[6 pts]** Write a function that, given float `c` and matrix `m`, returns a new matrix with elements in `m` multiplied by `c`.

```
let m = [[1.0 ; 2.0] ; [3.5 ; 4.5]] in
scalar_mult m 0.0 → [[0.0 ; 0.0] ; [0.0 ; 0.0]]
scalar_mult m 1.5 → [[1.5 ; 3.0] ; [5.25 ; 6.75]]
```

Write your answer below. (Hint: You know a higher-order function that makes this question very easy).

```
let rec scalar_mult (m : matrix) (c : float) : matrix =
map (fun r ->
      map (fun x -> x *. c) r) m
```

```
let scalar_mult m c = map (map (fun x -> x *. c)) m;;
```

6 Ruby [16 pts]

For this question, you need to write several methods to work with a data file containing transactions. Each line of the file has three columns: **date, amount,** and **location**. **Date** is in **YYYY-MM-DD** format, **amount** is a decimal with two digits after the point, and **location** is an arbitrary string. The amount does not start with a decimal point (but it can start with a zero). Each item is separated by a comma with no whitespace. Here is an example file:

```
2017-11-27,-250.00,Amazon.com
2017-11-16,-9.50,Chipotle
2017-11-15,10.00,UMD Paycheck
```

Write the following methods for a class `Transactions`:

1. **[5 pts] initialize(datafile):** The constructor takes as input a string containing the filename of the transactions, and it loads them into field **`@t`** of the current object. The `@t` field should be an array where each array element is itself an array `[date, amount, location]`, where the first and last items are strings and the amount is a float (use `String#to_f. Example: "1234.5".to_f => 1234.5`). You must validate the formatting of the file. If the file is invalid, raise an exception (hint: use `raise "Invalid Input"`). You can use the following pattern to read lines from a file:

```
IO.foreach("file") { |line| puts line }
```

```
def initialize(datafile)

      @transactions = []
      IO.foreach("file") { |line|
            if line =~ /(\d{4}-\d{2}-\d{2}),(-?\d+\.\d{2}),(.*)/
                  @transactions << {Transaction.new($1, $2, $3)}
            else
                  return false
            end
      }
      return true
end
```

2. **[5 pts] get_net_growth(start_date,end_date):** Returns as a float the total amount of money made (or lost) between the given start date and end date, inclusive. You may assume both dates are valid. To help you out, you may assume there is a function **`date_cmp(date1, date2)`**, which will return < 0 if date1 is earlier than date2, 0 if they are the same date, and > 0 if date1 is later than date2.

```
def get_net_growth(start_date, end_date)

sum = 0
      @transactions.each { |t|
            if date_cmp(start_date, t.date) <= 0 and date_cmp(t.date,
end_date) <= 0
                  sum += t.amount
            end
      }
      return sum



end
```

3. **[6 pts] most_frequent:** Takes no arguments, and returns the location that occurs most frequently in the transaction list. If there is a tie, it does not matter which one you pick.

```
def most_frequent


counts = Hash.new(0)
      @transactions.each { |t|
            Counts[t.location] += 1
      }
      m = counts.values.max
      return (counts.keys.select {|k| counts[k] == m})[0]







end
```

7 Prolog [20 pts]

1. **[6 pts]** consider the following definitions:

```
job(alice, programmer).
job(bob, manager).
job(celine, ceo).

supervises(A, B) :- job(A, manager), job(B, programmer).
supervises(A, B) :- job(A, ceo), job(B, manager).

can_fire(A, B) :- supervises(A, B).
can_fire(A,B) :-  supervises(A,C), can_fire(C,B)

low_level(A) :- \+ can_fire(A, _).     # \+ is the same as not
```

For each query, list the substitutions that Prolog produces that makes the query true.

1) `?- supervises(celine, X).`

   X = bob

2) `?- can_fire(X, alice).`

   {X = bob; X = celine}

3) `?- low_level(X).`

   false

2. **[4 pts]** List all substitutions that satisfy the following query, or write false if no such solution exists.

```
?- member(X, [0, 2]), length(Q, X), append(A, Q, [1, 2, 3, 4, 5]).
```

{X = 0, Q = [], A = [1, 2, 3, 4, 5]; X = 2, Q = [4, 5], A = [1, 2, 3]}

3. **[5 pts]** Implement `nim(N, L)`, where if N is a non-negative integer, then L is a list of 1s and 2s that add up to N.

```
?- nim(0, L).
L = [].
?- nim(3, L).
L = [1, 1, 1]; L = [1, 2]; L = [2, 1].
```

```
nim(0, []).

nim(N, [1|T]) :-
    N >= 1,
    N2 is N - 1,
    nim(N2, T).

nim(N, [2|T]) :-
    N >= 2,
    N2 is N - 2,
    nim(N2, T).
```

4. **[5 pts]** Implement `merge(A, B, C)`, where if A and B are sorted lists of integers, then C is the lists merged together. In other words, C is a sorted list of integers which contains all elements in both A and B. You CANNOT use the predicate *sort(+List, -Sorted)*.

```
?- merge([], [], C).
C = [].
?- merge([], [1, 2], C).
C = [1, 2].
?- merge([1, 4], [2, 3, 5], C).
C = [1, 2, 3, 4, 5].
```

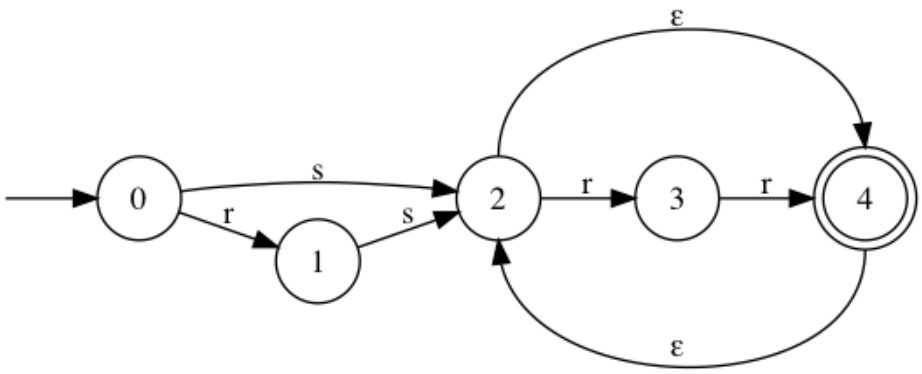
```
merge([], B, B).
merge(A, [], A).

merge([A|AT], [B|BT], [A|T]) :- A =< B, merge(AT, [B|BT], T).
merge([A|AT], [B|BT], [B|T]) :- A > B, merge([A|AT], BT, T).
```

8 Regexps, FAs, CFGs [18 pts]

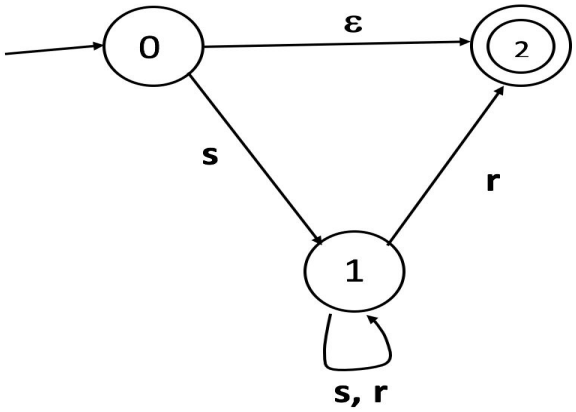
1. **[3 pts]** Write a regular expression for describing all the even length strings over the alphabet {a, b}.

   (aa|ab|ba|bb)*

2. **[3 pts]** Construct an NFA corresponding to the regular expression (s|rs)(rr)*.



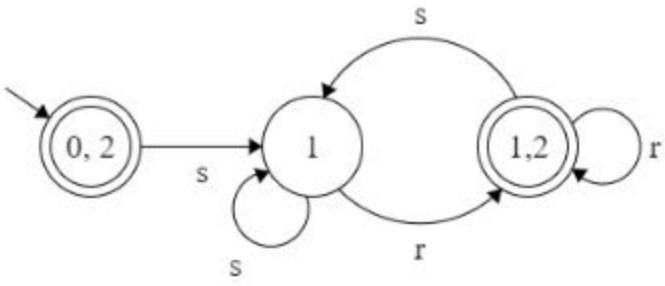
3. Consider the following NFA:



1) **[3 pts]** What regex does this NFA represent?

   (s (s|r)* r) | ε

2) **[5 pts]** Convert the NFA to a DFA.



4. **[4 pts]** Write a context-free grammar for the language of strings over the alphabet {a, b} containing an equal number of a's and b's. For example, "ab", "ba", "aabb", "abab", and the empty string are in the language, but "a", "b", "aba", and "aab" are not.

Some examples of correct solutions are:

- **S -> aSb | bSa | SS | ε**
- **S -> aSbS | bSaS | ε**

9 Parsing [8 pts]

S → BS | C
B → aB | bB | cB | ε
C → x | y

1. **[3 pts]** Given this CFG, calculate the first sets of B and S (ε represents the empty string)

   First(B) = { **a, b, c, ε** }

   First(S) = { **a, b, c, x, y** }

For the next part you are given the following utilities for parsing.

`lookahead`: Variable holding next terminal
`match(x)` : Function to match next terminal to x
`error()`: Signals a parse error

2. **[5 pts]** Write OCaml code for `parse_S()` for the CFG from part 1 using the above utilities. Assume `parse_B()`, and `parse_C()` are given. Any errors should be handled by raising an exception.

```
parse_S() {
        If (lookahead == 'x' || lookahead == 'y') {
                parse_C();
        } elsif (lookahead == 'a' || lookahead == 'b' || lookahead == 'c')) {
                parse_B();
                parse_S();
        } else {
                error("ParseError");
        }
}
```

10 Operational Semantics [6 pts]

1. **[3 pts]** Give a derivation showing that `let x = let x = 2 in 1 in x` evaluates to `1` in the empty environment using the following operational semantics rules:

$$\frac{A(x) = v}{A; x \Rightarrow v} \qquad \frac{}{A; n \Rightarrow n} \qquad \frac{A; e_1 \Rightarrow v_1 \quad A, x_1 : v_1; e_2 \Rightarrow v_2}{A; \texttt{let } x_1 = e_1 \texttt{ in } e_2 \Rightarrow v_2}$$

$$\frac{\dfrac{\dfrac{}{\bullet; 2 \Rightarrow 2} \quad \dfrac{}{\texttt{x} : 2; 1 \Rightarrow 1}}{\bullet; \texttt{let x = 2 in 1} \Rightarrow 1} \quad \dfrac{}{\texttt{x} : 1; \texttt{x} \Rightarrow 1}}{\bullet; \texttt{let x = let x = 2 in 1 in x} \Rightarrow 1}$$

2. **[3 pts]** Given the following rule for a while loop:

$$\frac{A_1; b_1 \Rightarrow \texttt{false} \quad A_1; s_1 \Rightarrow A_2}{A_1; \texttt{while } b_1 \texttt{ do } s_1 \Rightarrow A_2} \qquad \frac{A_1; b_1 \Rightarrow \texttt{true} \quad A_1; s_1 \Rightarrow A_2 \quad A_2; \texttt{while } b_1 \texttt{ do } s_1 \Rightarrow A_3}{A_1; \texttt{while } b_1 \texttt{ do } s_1 \Rightarrow A_3}$$

What unexpected behavior for `while` will the resulting language have?

Evaluates s1 even if b1 is false

11 Security [10 pts]

Multiple choice questions [1 point each]

1. What should you do to validate user input?
    a. Reject the input with unwanted characters
    b. Escape unwanted characters
    c. Remove unwanted characters
    d. A**ll of the above**

2. What is the security benefit of using *prepared statements*?
    a. They ensure user data is parsed as input, not code
    b. User data will always be encrypted
    c. They make constructing queries easier
    d. They prevent server-side errors

3. Consider the following code:

```
def sanitize_input(userString)
      reg = /[-\/\\^$*+?.()@|{}]/
      return userString.gsub(reg, '')
end
```

(Hint: gsub(pattern, replacement) returns a string where all occurrences of pattern are substituted for the second argument.)

This snippet is an example of validating user input by
    a. Whitelisting
    b. Blacklisting
    c. Escaping characters
    d. XSS prevention

4. A stored XSS attack typically must be prevented by
    a. The user's web browser
    b. A network administrator
    c. The database executing SQL code
    d. The web server

5. The following is a function used in a very buggy Ruby file transfer application.

```
1   def get_directory_contents(directory, userName)
2      if !directory.include?(userName) then
3           return nil
4      else
5           return `ls #{directory}` # executes shell command
6      end
7   end
```

1) **[1 pts]** In THREE WORDS OR FEWER, name a vulnerability that exists in the above code

   Command injection

2) **[2 pts]** Give an example of a value for the `directory` parameter that exploits your given vulnerability:

   alice; rm -rf /

3) **[3 pts]** Write a short code that fixes the vulnerability while maintaining intended functionality. Indicate at which line you want to insert your code.

Insert before line 2

```
if directory =~ /;/ then
      puts "illegal argument"
      exit 1
end
```