CMSC330 - Organization of Programming Languages
Fall 2023 - Final

CMSC330 Course Staff
University of Maryland
Department of Computer Science

Name: ______________________

UID: ______________________

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination

Signature: ______________________

Ground Rules

- You may use anything on the accompanying reference sheet anywhere on this exam
- Please write legibly. **If we cannot read your answer you will not receive credit**
- You may not leave the room or hand in your exam within the last 10 minutes of the exam
- If anything is unclear, ask a proctor. If you are still confused, write down your assumptions in the margin

Question	Points
P1	15
P2	5
P3	5
P4	10
P5	10
P6	10
P7	10
P8	10
P9	15
P10	10
EC	5
Total	100 + 5

Problem 1: Language Concepts

[Total 15 pts]

Indicate True or False for each of the below statements. 1 point per question.

		True	**False**
(A)	OCaml and Rust's static type system force each function or data structure to be created for only one specific type of data making it hard to create generalized code in those languages.	Ⓣ	Ⓕ
(B)	While Rust is a "memory-safe" language, OCaml and Python are not "memory-safe" and have features which may lead to memory corruption.	Ⓣ	Ⓕ
(C)	Any language described by a CFG can also be described by a regular expression.	Ⓣ	Ⓕ
(D)	Lambda Calculus can mimic the behavior of a Finite State Machine.	Ⓣ	Ⓕ
(E)	Python associates types with values, not variables.	Ⓣ	Ⓕ
(F)	A Context Free Grammar is Ambiguous if there is a single left most and single right most derivation for all strings of literals in the grammar.	Ⓣ	Ⓕ
(G)	"Double Freeing" (repeatedly de-allocating the same memory) is prevented in Rust by its semantics and compiler.	Ⓣ	Ⓕ
(H)	Operational Semantics specifies the meaning of language syntax such as whether `a = 7` means "check for equality" or "bind this value to this variable".	Ⓣ	Ⓕ
(I)	While Python and OCaml often share data between different parts of the program, Rust must copy data often as sharing pointers to the same block of memory is not permitted for safety reasons.	Ⓣ	Ⓕ
(J)	Due to its limited nature, the simply typed Lambda Calculus we studied is not powerful enough to represent ALL types of data or programs available in "real" programming languages.	Ⓣ	Ⓕ

Indicate which of the following choices best answers the question given. 2 points per answer.

(K) Python programs with **type problems** such as dividing an integer by a string are usually detected... [1 pts]

Ⓐ The semantics the language prevent this kind of problem from happening

Ⓑ At Compile Time before a program is created

Ⓒ At Run Time when code causing the problem executes

Ⓓ This type of error is not detected and can lead to memory corruption.

(L) OCaml programs with **memory problems** such a using memory after it has been de-allocated are detected... [2 pts]

Ⓐ The semantics the language prevent this kind of problem from happening

Ⓑ At Compile Time before a program is created

Ⓒ At Run Time when code causing the problem executes

Ⓓ This type of error is not detected and can lead to memory corruption.

(M) Rust programs with **memory problems** such a using memory after it has been de-allocated are detected... [2 pts]

Ⓐ The semantics of the language prevent this kind of problem from happening

Ⓑ At Compile Time before a program is created

Ⓒ At Run Time when code causing the problem executes

Ⓓ This type of error is not detected and can lead to memory corruption.

Problem 2: Regular Expressions

[Total 5 pts]

(a) Which of the following strings are accepted by the regular expression below? [2 pts]

```
L*DS|[OB]{2}
```

(O|B)·2

Select NONE if none of the first five (5) options match.

● DS (B) OBOB (C) DOL (D) LSBB ● OB (F) NONE

(b) **Write a Regular Expression** Consider the OCaml variant: [3 pts]

```
    type shape = Rect of int * int|Circ of float
```

Write a regular expression that would describe an OCaml expression which binds a shape to a variable.

EXAMPLES:

```
 Valid              | Invalid
 let x = Rect(10,2) | let x = Rect 10,2 (* need parens around constructor *)
 let y = Circ(1.23) | let yz = Circ(1.2) (* variable name too long *)
 let z = Rec(0,-10) | let a = Rect(10.2,5.6) (* Rect requires ints in its pair, not floats *)
 let w = Circ(-1.2) | let B = Circ(-1.2) (* variable name uppercase *)
```

Constraints and limitations:

1. The variable name will be a single lowercase alphabetic character
2. One or more spaces can appear between the `let` keyword and the variable name
3. Zero or more spaces can appear on either side of the equal sign = in the strings
4. The constructor of the `shape` must use parenthesis.
5. No spaces may appear within the parentheses ( ) of the constructor
6. Multi-digit numbers MAY have leading zeros (i.e. 0123 is accepted)
7. Only unary negation of numbers is accepted, NOT unary plus.
8. Floating point values always have a decimal point with trailing digits.

let \s+[a-z]\s*=\s*^Rect([

Problem 3: Context Free Grammars

[Total 5 pts]

Consider the following Grammar:

```
    S -> S is S|U
    U -> Not U|P
    P -> That|It
```

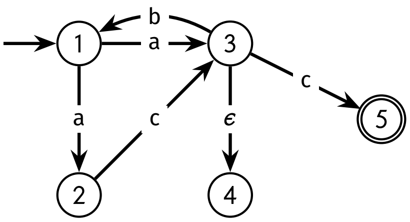
Derive the string "`That is Not It`". Use a leftmost derivation and show all steps for full credit

Problem 4: Finite State Machines

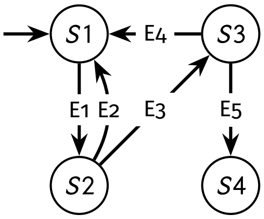
[Total 10 pts]

Using the Subset Construction algorithm, convert the following NFA to a DFA, and fill in the blanks appropriately matching the DFA provided with the right nodes and transitions. Only answer in the blank boxes / multiple choice below will be graded.

NFA:

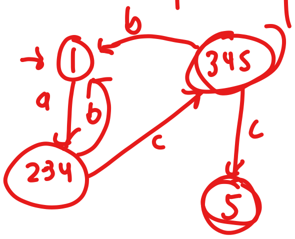


DFA:



Scratch Space (if needed)

R	a	b	c
1	234	∅	∅
234	∅	1	3,4, 5
345	∅	1	5
5	∅	∅	∅



(a) Blanks

[9 pts]

S1: 1

S2: 2,3,4

S3: 3,4,5

S4: 5

E1: a

E2: b

E3: c

E4: b

E5: c

(b) Final States

[1 pts]

S1 S2 [✓ S3] [✓ S4]

Problem 5: Language Analysis

[Total 10 pts]

Consider the following two code fragments which attempt to compute similar results involving only integers or integer tuples.

```
# Python
def ratio(x,y,as_tup):
  if as_tup:
    return (x,y,x//y)
  else:
    return x//y
```

```
(* OCaml *)
let ratio x y as_tup =
  if as_tup then
    (x,y,x/y)
  else
    x/y
```

(a) **Python Correctness:** What would the Python version do on an input like `ratio(15,3,True)` vs `ratio(15,3,False)` ? Would it function correctly? [3 pts]

(b) **OCaml Correctness:** The OCaml version will not compile. Why not? Describe the problems the compiler will identify in a few sentences. [2 pts]

(c) **OCaml Equivalence:** Describe how to get the same effect in the OCaml function as is present in the Python program so that the same basic data can be returned in the OCaml version. Mention any relevant features of OCaml that would be useful here. [3 pts]

(d) **Rust Safety:** Nearby is a partial Rust version of the `ratio()` function along with `main()` that calls it several times. Describe whether the Rust compiler will compile this program and the results of running it. [2 pts]

```
// Rust
fn ratio(x: i32, y: i32) -> i32{
  return x / y;
}
fn main(){
  println!("ratio(15,3): {}",ratio(15,3));
  println!("ratio(15,0): {}",ratio(15,0));
  println!("ratio(15,3): {}",ratio(16,2));
}
```

Problem 6: Operational Semantics

[Total 10 pts]

Consider the following operational semantics for some list operations with OCaml as the metalanguage:

$$\frac{}{[] \Rightarrow []} \qquad \frac{}{[x] \Rightarrow x :: []}$$

$$\frac{}{[x, y] \Rightarrow x :: [y]} \qquad \frac{}{[x, y, z] \Rightarrow x :: [y, z]}$$

$$\frac{}{len([]) \Rightarrow 0} \qquad \frac{v \Rightarrow h :: t \quad len(t) \Rightarrow l \quad n = l + 1}{len(v) => n}$$

$$\frac{len(v) \Rightarrow l \quad l > 0 \quad v => x :: xs}{v.get(0) \Rightarrow x} \qquad \frac{len(v) \Rightarrow l \quad l > x \quad v \Rightarrow y :: ys \quad z = x - 1 \quad ys.get(z) \Rightarrow n}{v.get(x) => n}$$

$$\frac{len(v) \Rightarrow l \quad l < x}{v.get(x) \Rightarrow None}$$

Construct a proof that $len([1, 2]) => 2$. Use the space below and follow the structure of the tree provided. Letters indicate blanks that must be filled in.

```
                 =========================  ==========================


                 (D)                        (E)                        (F)
===============  =================================================================


(A)                        (B)                                                        (C)
======================================================================================================
                                          len([1,2])=>2
```

Problem 7: Rust Ownership

[Total 10 pts]

```
1  fn append_one(v: &mut Vec<&i32>){
2    let n = 1;
3    v.push(&n);
4  }
5
6  fn main() {
7    let three = 3;
8    let two = 2;
9    let mut down = vec![&three,&two];
10   append_one(&mut down);
11   println!("{:?}",down);
12 }
```

Nearby is a short Rust snippet. When compiled it will give an error to the effect that `Line 3: borrowed value &n does not live long enough.` Explain in a two sentences why this error is occurring

Explain code changes that would fix the `append_one()` function and allow it to append 1 to vectors. Your changes may include alterations to the function and `main`.

Problem 8: Type Inference

[Total 10 pts]

Give the type inferred for the following OCaml expressions.

```
fun a b c ->                          (* A *)
    (List.map a c) > b || c > b
```

```
fun x y z ->                          (* B *)
    let _ = List.fold_left x
                          (0,0)
                          [true;false;y]
    in z
```

```
fun a b c d ->                        (* C *)
    List.fold_left a b (map c d)
```

Problem 9: OCaml Coding

[Total 15 pts]

Below are types and a description of the less_traveled_by function. Analyze the examples provided and implement this function in OCaml.

```
type path =                   (* Branching paths with travel counts along Roads *)
  | Road of int*path          (* count of travel along Road AND remaining path *)
  | Diverge of path*path      (* left path AND right path *)
  | End;;                     (* end of the path *)

(* EXAMPLE 1 *)                                (* EXAMPLE 2 *)
# let road_to_nowhere = End;;                  # let cormack_road =
# less_traveled_by road_to_nowhere;;               Road(1, Road(2, Road(5, Road(8, End))));;
string list * int = ([], 0)                    # less_traveled_by cormack_road;;
                                               - : string list * int = ([], 16)

(* EXAMPLE 3 *)
# let branching =                             (* Left/right path and count for each *)
    Diverge( Diverge( Diverge( Road(4,End),   (* (["left","left","left"],4)  *)
                               Road(2,End)),  (* (["left","left","right"],2) <- LEAST TRAVELED *)
                      Road(3,End)),           (* (["left","right"],3)        *)
             Diverge( Road(7,End),            (* (["right","left"],7)        *)
                      Road(6,End)));;         (* (["right","right"],6)       *)
# less_traveled_by branching;;
- : string list * int = (["left"; "left"; "right"], 2)

(* EXAMPLE 4 *)
# let yellow_wood =
Road(2, Diverge( Road(4, Road(1, Diverge( Road(3, End),       (* (["left";"left"], 10)        *)
                                          End))),             (* (["left";"right"], 7)        *)
                 Road(1, Diverge( Diverge( Road(5, End),      (* (["right";"left";"left"], 8) *)
                                           Road(2, End)),     (* (["right";"left";"right"], 5) LEAST*)
                                  Road(1, Road(4, End))))));; (* (["right";"right"],8)        *)
# less_traveled_by yellow_wood;;
- : string list * int = (["right"; "left";"right"], 5)
```

Write your implementation on the next page.

Write your implementation here.

```
(* Returns a pair of (string list * int); the list shows one sequence of left/right
   choices through the path with smallest total count along roads in the path. *)
let rec less_traveled_by path =
```

Problem 10: Lambda Calculus

[Total 10 pts]

(a) **Lazy Evaluation, Single Step** Perform a single step of Beta Reduction using the Lazy / Call by Name Evaluation Strategy on the given Lambda Calculus expression. If the expression cannot be reduced, select *"Beta Normal Form"*. The options offered may be alpha equivalent to what you calculated. [3 pts]

$$(\lambda x.xx)((\lambda y.yy)(\lambda z.zz))$$

Ⓐ $(\lambda x.xx)(\lambda z.zz)$
Ⓑ $(\lambda x.xx)((\lambda z.zz)(\lambda z.zz))$
Ⓒ $((\lambda y.yy)(\lambda z.zz))((\lambda y.yy)(\lambda z.zz))$
Ⓓ Beta Normal Form
Ⓔ None of the above

$$(\lambda x.(\lambda y.y\ z)a)$$

Ⓐ $(\lambda x.\lambda a.a\ z)$
Ⓑ z
Ⓒ $(\lambda x.a\ z)$
Ⓓ Beta Normal Form
Ⓔ None of the above

(b) **Eager Evaluation, Single Step:** As before, perform a single step of Beta Reduction but this time use the Eager / Call by Value Evaluation Strategy. [3 pts]

$$(\lambda x.xx)((\lambda y.yy)(\lambda z.zz))$$

Ⓐ $(\lambda x.xx)(\lambda z.zz)$
Ⓑ $(\lambda x.xx)((\lambda z.zz)(\lambda z.zz))$
Ⓒ $((\lambda y.yy)(\lambda z.zz))((\lambda y.yy)(\lambda z.zz))$
Ⓓ Beta Normal Form
Ⓔ None of the above

$$(\lambda x.(\lambda y.y\ z)a)$$

Ⓐ $(\lambda x.\lambda a.a\ z)$
Ⓑ z
Ⓒ $(\lambda x.a\ z)$
Ⓓ Beta Normal Form
Ⓔ None of the above

(c) **Reduce to Normal Form:** Convert the following to Beta Normal Form: $(\lambda x.\lambda y.\lambda z.x\ y\ z)\ a\ b\ (\lambda d.d\ c)\ e$ [2 pts]

Ⓐ $a\ b\ e\ c$ Ⓑ $a\ b\ (\lambda d.d\ c)$ Ⓒ $a\ b$ Ⓓ $a\ b\ e\ c\ y\ z$
Ⓔ Already in Beta Normal Form Ⓕ Diverges like infinite recursion Ⓖ None

(d) **Reduce to Normal Form:** Convert the following to Beta Normal Form: $(\lambda y.y\ y\ y)(\lambda y.y\ y\ y)$ [2 pts]

Ⓐ $(\lambda y.y\ y\ y)$ Ⓑ $y\ y\ y$ Ⓒ y Ⓓ $(\lambda y.y\ y\ y)(\lambda y.y\ y\ y)(\lambda y.y\ y\ y)$
Ⓔ Already in Beta Normal Form Ⓕ Diverges like infinite recursion Ⓖ None

The blank area below may be used for calculations

Problem 11: Extra Credit

[Total 5 pts]

(a) What is your TA's name and what is your section id? [3 pts]

(b) What is your favorite pun? [2 pts]

Raising a family is hard. Especially if you're not a necromancer

Cheat Sheet

Python

```
# Lists
lst = []
lst = [1,2,3,4]
lst[2] # returns 3
lst[-1] # returns 4
lst[0] = 4 # list becomes [4,2,3,4]
lst[1:3] # returns [2,3]
# Strings
string = ”hello”
len(string) # returns 5
string[0] # returns h
string[2:4] # returns ll
string = ”this is a sentence”
string.split(” ”)
# returns [”this”, ”is”, ”a”, ”sentence”]

# Dictionary
# {’a’:0,’b’:1}.keys() #returns [’a’,’b’]
# {’a’:0,’b’:1}.values() #returns [0,1]
# ’a’ in {’a’:0,’b’:1} # returns True

# Map and Reduce

map(function, lst)
# returns a map object corresponding to the
# result of calling function to each item in lst
# typically needs to be cast as a list

reduce(function,lst,start)
# returns a value that is the combination of all
# items in lst. function(accum,cur) will be used to
# combine the items together, starting with start,
# and then going through each item in the list.

reduce(function,lst)
# omitting start uses the first element being
# reduced as ’accum’ in the above version
```

```
# List functions
lst = [1,2,3,4,5]
len(lst) # returns 5
sum(lst) # returns 15
lst.append(6) # returns None. lst now [1,2,3,4,5,6]
lst.pop() # returns 6. lst is now [1,2,3,4,5]
[1,2,3] + [4,5,6] # returns [1,2,3,4,5,6]

# regex in python
re.fullmatch(pattern,string)
# returns a match object if string is a
# full/exact match to string.
# returns None otherwise

re.search(pattern,string)
# returns a match object corresponding to
# the first instance of pattern in string.
# returns None otherwise

re.findall(pattern, string)
# returns all non-overlapping matches
# of pattern in string as a list

re.finditer(pattern, string)
# returns an iterator over the string
# each iteration gives a match object

# match objects
m = re.search(”([0-9]+) ([0-9]+)”, ”12 34”)
m.groups() # returns (”12”, ”34”)
# returns a tuple of all things that were
# captured with parentheses

m.group(n) # m.group(1) = ”12”, m.group(2) = ”34”
# returns the string captured by the nth
# set of parentheses
```

OCaml

```
(* Lists *)
let lst = []
let lst = [1;2;3;4]

(* :: (cons) has type ’a->’a list -> ’a list *)
1::2::3::[] = [1;2;3]

(* @ (append) has type ’a list -> ’a list-> ’a list *)
[1;2;3] @ [4;5;6] = [1;2;3;4;5;6]

(* variants *)
type linkedlist = Cons of int * linkedlist|Nil
Cons(1,Cons(2,Cons(3,Nil)))

(* pattern matching *)
match linkedlist with
     Cons(x,y) -> ”List”
    |Nil -> ”Nil”
```

```
(* Anonymous Functions *)
(fun a b c -> a + b + c *)

(* Map and Fold *)
let rec map f l = match l with
   [] -> []
  |x::xs -> (f x)::(map f t)

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

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

Rust

```
// Vectors
let vec = Vec::new();  // makes a new vector
let vec1 = vec![1,2,3]

vec.push(ele);  // Pushes the element ’ele’
                // to end of the vector ’vec’
// Strings
let string = String::from(”Hello”);

string.push_str(&str); // appends the str
                       // to string

vec.to_iter();  // returns an iterator for vec

string.chars()  // returns an iterator of chars
                // over the a string
```

```
iter.rev();       // reverses an iterators direction

iter.next();      // returns an Option of the next
                  // item in the iterator.

struct Building{   // example of struct
    name:String,
    floors:i32,
    locationx:f32,
    locationy:f32,
}

enum Option<T>{ Some(T); None } //enum Option type
option.unwrap();  // returns the item in an Option or
                  // panics if None
```

Regex

*	zero or more repetitions of the preceding character or group
+	one or more repetitions of the preceding character or group
?	zero or one repetitions of the preceding character or group
.	any character
$r_1 \vert r_2$	r_1 or r_2 (eg. a—b means 'a' or 'b')
[abc]	match any character in abc
[ˆr_1]	anything except r_1 (eg. [ˆabc] is anything but an 'a', 'b', or 'c')
[r_1-r_2]	range specification (eg. [a-z] means any letter in the ASCII range of a-z)
{n}	exactly n repetitions of the preceding character or group
{n,}	at least n repetitions of the preceding character or group
{m,n}	at least m and at most n repetitions of the preceding character or group
ˆ	start of string
$	end of string
(r_1)	capture the pattern r_1 and store it somewhere (match group in Python)
\d	any digit, same as [0-9]
\s	any space character like \n, \t, \r, \f, or space

NFA to DFA Algorithm (Subset Construction Algorithm)

NFA (input): $(\Sigma, Q, q_0, F_n, \sigma)$, DFA (output): $(\Sigma, R, r_0, F_d, \sigma_n)$

```
$R \leftarrow \{\}$
$r_0 \leftarrow \epsilon - \text{closure}(\sigma, q_0)$
**while** $\exists$ an unmarked state $r \in R$ **do**
    mark $r$
    **for all** $a \in \Sigma$ **do**
        $E \leftarrow \text{move}(\sigma, r, a)$
        $e \leftarrow \epsilon - \text{closure}(\sigma, E)$
        **if** $e \notin R$ **then**
            $R \leftarrow R \cup \{e\}$
        **end if**
        $\sigma_n \leftarrow \sigma_n \cup \{r, a, e\}$
    **end for**
**end while**
$F_d \leftarrow \{r \mid \exists s \in r \text{ with } s \in F_n\}$
```

Structure of Regex

Regex

$$\begin{array}{lll} R & \rightarrow & \emptyset \\ & - & \sigma \\ & - & \epsilon \\ & - & RR \\ & - & R|R \\ & - & R^* \end{array}$$

T	P	R	T	C	U	S	A	V	H	K	N	A	I	R	O	T	C	I	V
T	E	J	A	T	A	H	R	N	J	A	N	A	R	L	R	P	B	S	A
K	C	Z	D	J	S	U	S	H	A	W	N	I	U	E	D	W	I	N	N
T	Y	H	L	X	A	T	H	O	C	O	G	C	W	N	S	N	I	X	D
G	O	Q	S	L	P	R	U	S	L	T	I	O	A	L	M	A	N	A	E
X	L	A	W	D	O	C	E	I	O	N	I	I	W	A	I	N	O	B	R
L	I	N	H	S	O	R	V	D	D	J	R	M	V	G	T	N	S	B	S
N	E	W	R	A	M	I	P	A	V	B	H	P	F	F	K	A	A	Y	B
X	A	I	M	A	A	N	F	F	A	A	E	Y	A	S	W	I	J	V	K
O	A	N	N	A	B	E	L	R	F	A	N	L	L	M	Y	K	J	R	R
K	I	H	X	H	E	M	S	A	V	E	J	S	Y	I	R	A	B	U	A
L	J	S	S	W	U	A	P	K	V	A	Q	D	H	Y	L	G	V	T	K
N	A	H	O	R	S	R	D	H	X	D	Q	G	E	I	B	H	K	I	Z
B	Z	M	Y	H	R	G	A	S	S	A	X	D	Y	C	K	Z	E	O	F
N	G	S	N	A	V	A	L	U	H	M	Z	M	A	A	E	A	Y	Q	G
T	O	M	A	S	D	R	T	O	E	D	N	O	E	N	I	A	Y	T	H
P	M	G	E	Y	D	E	O	N	L	H	E	L	T	S	E	R	G	T	L
H	U	J	U	A	N	T	N	A	D	M	R	L	A	L	V	S	A	U	L
W	P	G	N	U	S	H	O	C	O	R	E	I	N	R	X	R	H	M	U
C	G	O	H	I	I	I	Z	N	N	P	C	E	I	H	C	H	D	I	G

1. zoya
2. vruti
3. sheldon
4. adam
5. olivia
6. jared
7. edwin
8. joshua
9. brian
10. abby
11. tomas
12. danesh
13. vanshika
14. lily
15. annaika
16. annabel
17. smit
18. teja
19. arwen
20. shawn
21. margaret
22. vasu
23. ani
24. jason
25. mia
26. lucinda
27. anoushka
28. jana
29. rohan
30. tim
31. ceren
32. dalton
33. ohsung
34. maria
35. amr
36. mollie
37. anders
38. juan
39. roshni
40. victoria