CMSC330 - Organization of Programming Languages
Summer 2023 - Exam 1

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
Q1	10
Q2	15
Q3	15
Q4	15
Q5	20
Q6	15
Q7	10
Total	100

Problem 1: Language Concepts

[Total 10 pts]

	True	False
Any regular expression can be expressed as a Context Free Grammar	T	F
`let f x = x 4` is an example of a higher order function	T	F
One could theoretically code project 1 in lambda calculus	T	F
All statically typed languages use explicit (manifest) typing	T	F
FSMs are a subset of Turing Machines in terms of computational power	T	F

Problem 2: Typing

[Total 15 pts]

Write an expression of the following types in OCaml. You cannot use type annotations, and all pattern matching must be exhaustive.

(a) `string -> 'a -> string` [2 pts]

(b) `'a -> 'a -> bool -> 'a` [3 pts]

Given the following OCaml expressions, write down its type.

(c) `fun a b -> let c = a = b in if c then 2 else 3` [2 pts]

(d) `fun a b c d -> if a && let x = b > c in x then d + 1 else b` [3 pts]

(e) Which of the following choices could be the type of the python lambda below? Select all that apply. [2 pts]

```
lambda x,y: x + y
```

(A) int -> int -> int (B) string -> int -> string (C) list -> list -> list (D) float -> int -> float

(E) None of the above

(f) Which of the following python lambdas could have the type of string list -> int list? Select all the apply. [3 pts]

(A) `lambda x: [1,2] if x == ["hello"] else [0]` (B) `lambda x: [len(x[0])]`
(C) `lambda x: map(lambda y:  len(y),x)` (D) `lambda x: len(x)`
(E) None of the above

Problem 3: Regular Expressions

[Total 15 pts]

(a) Which of the following strings are an exact match of the following Regular Expression? Mark all that apply. [5 pts]

```
^[A-Z][a-z0-9]+: ([0-9]{3}|[CS330]+)$
```

(A) Major: CS (B) Age: 25 (C) Class: CS330 (D) Finitial: C (E) None

(b) Write a regular expression that accepts phone numbers of all the following formats and rejects everything else. You may assume that any X can be any digit. [5 pts]

XXX-XXX-XXXX   XXX-XXXXXXX   XXXXXXXXXX   (XXX)-XXX-XXXX   (XXX)-XXXXXXX   (XXX)XXXXXXX

(c) Write a regular expression that would accept all strings of odd length and have at least 1 lowercase vowel (a,e,i,o,u) and reject anything else [5 pts]

Problem 4: Context Free Grammars

[Total 15 pts]

Consider the following Grammars:

Grammar 1	Grammar 2	Grammar 3	Grammar 4
S -> AB	S -> ASB|a	S -> Sc|AB	S -> ASB|cScc|c
A -> aAa|a	A -> aA|a	A -> aA|a	A -> aaA|a
B -> bBbb|ϵ	B -> bbB|c	B -> bbB|b	B -> bbB|b

(a) Which grammars (of 1, 2, and 3) accept both "aabbbbc" and "aaabbcc"? Select all that apply. [4 pts]

(1) Grammar 1 (2) Grammar 2 (3) Grammar 3 (N) None

(b) Ambiguity [6 pts]

	Yes	**No**
"aaabbb" is an ambiguous string in Grammar 1	(Y)	(N)
"aaabbc" is an ambiguous string in Grammar 2	(Y)	(N)
"aaabcc" is an ambiguous string in Grammar 3	(Y)	(N)

(c) Which strings are accepted by Grammar 4? Select all that apply. [5 pts]

(A) aaacbbb (B) aaacbbbb (C) ccaaabbbbcc (D) cacacbbbb (E) None

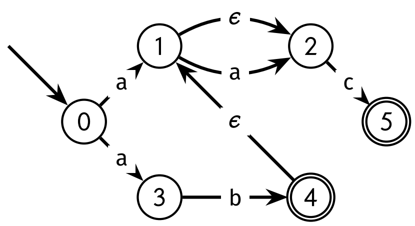
Problem 5: Finite State Machines

[Total 20 pts]

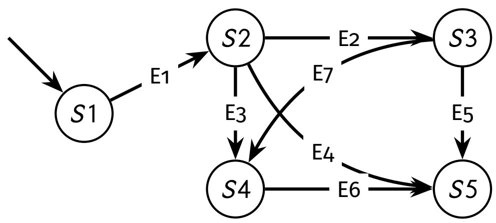
(a) Using the subset algorithm, convert the following NFA to a DFA, and fill in the blanks appropriately matching the DFA provided with the right nodes and transitions. Only the blanks will be graded. [12 pts]

NFA:

Scratch Space (if needed)



DFA:



S1: ______ S2: ______ S3: ______ S4: ______

S5: ______ E1: ______ E2: ______ E3: ______

E4: ______ E5: ______ E6: ______ E7: ______

(b) Which of the following are the final states? Select all that apply [3 pts]

(1) S1 (2) S2 (3) S3 (4) S4 (5) S5 (N) None

(c) Write a regex to describe the language of the above NFA [5 pts]

Problem 6: Lambda Calculus

[Total 15 pts]

For the following questions perform a single β-reduction using eager (call by value) evaluation on the outermost expression. If you cannot reduce it, write **Beta Normal Form**. You may **not** α-convert your final answer.

(a) $(\lambda y.yy)((\lambda x.y)(\lambda y.xy))$ [2 pts]

(b) $(\lambda x.\lambda x.xx)(z\ (\lambda a.a))$ [3 pts]

For the following questions perform a single β-reduction using lazy (call by name) evaluation on the outermost expression. If you cannot reduce it, write **Beta Normal Form**. You may **not** α-convert your final answer.

(c) $(\lambda y.yy)((\lambda x.y)(\lambda y.xy))$ [2 pts]

(d) $(\lambda x.\lambda x.xx)(z\ (\lambda a.a))$ [3 pts]

(e) Which of the following is alpha equivalent to $(\lambda x.x\lambda x.xy)$? Select all that apply. [2 pts]

(A) $(\lambda z.z\lambda x.zy)$ (B) $(\lambda y.y\lambda x.xy)$ (C) $(\lambda z.z\lambda x.xy)$ (D) $(\lambda x.x\lambda y.yz)$ (G) None

(f) Convert the following to Beta Normal Form: $(\lambda z.\lambda x.xz)(\lambda y.yy)c$ [3 pts]

(A) c (B) $(\lambda x.x\ x)c$ (C) $c\ (\lambda y.y\ y)$ (D) $\lambda x.x\ (c\ c)$ (E) $c\ c$ (F) Infinite Recursion (G) None

Problem 7: Python Programming [Total 10 pts]

(a) Write a function `mur` that has the same functionality of `map`, but uses `reduce`. [4 pts]

```
def mur(f,lst):
    return reduce(___BLANK____)

#mur(lambda x: x + 1,[1,2,3]) => [2,3,4]
#mur(lambda x: len(x),[[1,2,3],[4,5],[6]]) => [3,2,1]
#mur(lambda x: x,[1,2,3]) => [1,2,3]
```

Blank:

(b) Write a function `sumnum` that takes in a formatted string and returns the sum of all the numbers found in that string. [6 pts]

```
#sumnum("I have 2 apples and 30 oranges") => 32
#sumnum("There are no numbers here") => 0
#sumnum("I can have negatives like -2 and -4") => -6

def sumnum(s):
```

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"]

# 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 will be used to
# combine the items together, starting with
# start, and then going through each item
# in the list
```

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

# 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

# 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 parenthesis

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

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
$r_1 \mid r_2$	r_1 or r_2 (eg. a|b means 'a' or 'b')
[$r_1 r_2 r_3$]	r_1 or r_2 or $r3$ (eg. [abc] is 'a' or 'b' or 'c')
[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 ← {}
r0 ← ε − closure(σ, q0)
while ∃ an unmarked state r ∈ R do
    mark r
    for all a ∈ Σ do
        E ← move(σ, r, a)
        e ← ε − closure(σ, E)
        if e ∉ R then
            R ← R ∪ {e}
        end if
        σn ← σn ∪ {r, a, e}
    end for
end while
Fd ← {r | ∃s ∈ r with s ∈ Fn}
```

Grammars

Regex			Lambda Calc		
R	$\rightarrow$	$\emptyset$	e	$\rightarrow$	x
	$\mid$	σ		$\mid$	$\lambda x.e$
	$\mid$	ϵ		$\mid$	$e\ e$
	$\mid$	RR			
	$\mid$	$R\mid R$			
	$\mid$	R^*			

Lambda Calc Encodings

We will give you the encodings that you will need. They may or may not look like/include the following:

$\lambda x.\lambda y.x$	=	true
$\lambda x.\lambda y.y$	=	false
$e_1\ e_2\ e_3$	=	if e_1 then e_2 else e_3