SOLUTION

CMSC330 - Organization of Programming Languages
Fall 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
P1	10
P2	15
P3	15
P4	15
P5	15
P6	15
P7	15
Total	100

Problem 1: Language Concepts

[Total 10 pts]

(a) True/False [8 pts]

	True	False
OCaml is statically and latently (implicitly) typed	(T)	(F)
if `f` is a function passed into map, `f` can also be passed into reduce Map's function takes in a single value, whereas fold's needs 2	(T)	(F)
`x = x + 1` and `x += 1` are semantically the same they both are adding 1 to x	(T)	(F)
`lambda x:x(3)` is an example of a higher order function in Python x is being treated as a function so we can assume this is taking in a function as input	(T)	(F)
Python does not track the types associated with variables so it will never generate an error for type mismatches like `3+"hi"` Why python does not track types with variables, you will get an error for 3 + "hi"	(T)	(F)
Examining a Python expression like `x+y`, one can determine that it will never generate a runtime type error. We know nothing of the types of x and y since python is dynamically typed	(T)	(F)
Examining an OCaml expression like `x+y`, one can determine that it will never generate a runtime type error. OCaml will check types before runtime to make sure variables are being used correctly	(T)	(F)
Extended Regex syntax such as `+ ? . [a-z]` that are often supported by Regular Expression engines CANNOT be translated to the fundamental Regex building primitives of concatenation, union, and Kleene closure. These are just shortcuts for the basics of concat,union and kleene	(T)	(F)

(b) Unlike Python, OCaml requires special syntax to allow "side effects" and "mutation" to occur. Describe in 2 sentences an advantage this confers to OCaml programs. [2 pts]

OCaml programs tend to be more functional, involve less mutability, and prefer aspects like "referential transparency." This makes them easier to reason about and formally or informally prove they behave correctly.

Problem 2: 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]+.3|(a|A)*[a-z][0-9]3$
```

(A) ABC123 (B) abc123 (C) Ab12 (D) ABCa134 (E) None

(b) Write a regular expression that recognizes Engineering / Exponential numbers. Examples of these are shown below. In this format, numbers may start with an optional -/+, followed by exactly 1 non-zero digit, followed by a decimal point. After the decimal, there is a upper/lower case E, followed by an optional +/-, and ending with one or more digits. [5 pts]

Examples: `1.2345e20  -3.14159E+00  +5.67e-1  -2.0e+123`

`[-+]?[1-9]\.[0-9]+[eE][-+]?[0-9]+ OR [-+]?[1-9]\.[0-9]*[eE][-+]?[0-9]+`

(c) Write a regular expression that accepts mathematical expressions that could be put into a 4-function integer calculator. [5 pts]

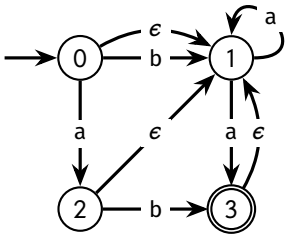
Examples: `1+2, 2-3,  -5*6,  12/3/2,  1+3-5,  36*122+5/6`

`[+-]?[0-9]+([+-*/][+-]?[0-9]+)* OR [-]?[0-9]+([+-*/][-]?[0-9]+)*`

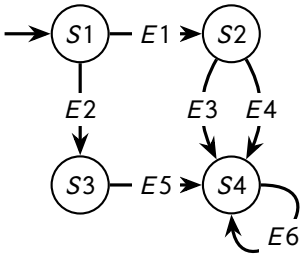
Problem 3: Finite State Machines [Total 15 pts]

(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. [12 pts]

NFA: Scratch Space (if needed)



DFA:



S1: 0,1 S2: 1,2,3 S3: 1 S4: 1,3

E1: a E2: b E3: a

E4: b E5: a E6: a Note: E3 and E4 could be swapped.

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

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

Problem 4: Higher Order Programming [Total 15 pts]

(a) What's the return value? If the code throws an error put "ERROR". [3 pts]

```
return map(lambda x: len(x[0]), [["hello"],[[1,2],3]], [])
```

Empty list [] (technically correct but not expected) OR ERROR (identifying map() doesn't take a 3rd arg) OR [5,2] (took original list); "Map Object" also technically correct but not for full credit.

(b) What segment(s) of code would add up the values in `lst` after cubing each value? Mark all that apply [4 pts]

(A) `reduce(lambda x,y:  x + y, map(lambda x:  x * x * x,lst),0)`
(B) `reduce(lambda x,y:  x * x * x + y, lst, 0)`
(C) `reduce(lambda x,y:  x + y, reduce(lambda x,y:  x * x * x,lst,0),0)`
(D) None

(c) Convert the following function to a shorter version that uses a one or more appropriate **higher-order functions**. Missing opportunities to utilize higher-order functions will result in a loss of credit. You may use Lambda expressions or helper functions. [8 pts]

```
# OLD VERSION
def only_odd_update(lst):           # EXAMPLES
  newlst = []                       # >>> only_odd_update([7, 2, 4, 9, 11])
  for x in lst:                     # [22, 28, 34]
    if x % 2 == 1:                  # >>> only_odd_update([6])
      new = x*3 + 1                 # []
      newlst.append(new)            # >>> only_odd_update([12,14,33])
  return newlst                     # [100]

from functools import *  # all higher-order functions available

# NEW VERSION

### SOLUTION 1
def only_odd_update(lst):
  return list(map(lambda x:3*x+1, filter(lambda x:x%2==1, lst)))

### SOLUTION 2
def only_odd_update(lst):
  return list(reduce(lambda acc,cur: acc+[3*cur+1] if cur%2==1 else acc,
                     lst, []))
```

Problem 5: Regex / FSM Relations [Total 15 pts]

(a) Con V. Eertit claims to have found a regular expression R that has no equivalent Finite State Machine to match it. Con believes this will earn him great acclaim while the claim seems dubious to you. If Con showed you the regex R, how would you use it to disprove his claim? *Answer in 2-3 sentences.* [8 pts]

An easy way to disprove Con's claim is to examine R and convert it to an NFA. This can be done by examining small parts such as individual character matches, creating NFAs for those, then combining those NFAs into larger NFAs as Union, Concatentation, and Kleene closure appear. If Con used any fancy + or ?, convert those to the equivalent Union/Concat/Star operations. This will build an NFA which is a FSM proving Con wrong.

(b) Equi Valent has spent a considerable amount of time writing a slick NFA matching routine. It takes an NFA and a string and produces a true or false based on whether the string would be accepted by the NFA. Her supervisor, Didno Take (330th in his line) insists that Equi must now also create a DFA matching algorithm for the company. Should Equi be worried about staying late at work to create the DFA matching routine? Explain your reasoning based on the relationship between NFAs and DFAs. *Answer in 2-3 sentences.* [7 pts]

This is not a problem for Equi. DFAs are a subset of NFAs so the NFA matching routine will work without modification for any DFA passed to it. Equi can head home early.

Problem 6: Type Inference and Static Typing in OCaml [Total 15 pts]

Consider the following OCaml function which is being defined in an interactive loop.

```
# let trip_max a b =
  if a > b then
    3*a
  else
    3*b;;
```

(a) What is the type inferred for `trip_max`? [3 pts]

```
int -> int -> int
```

Tup Le'Type is coming from Python and learning OCaml. He tries to invoke the above function and gets an error.

```
# trip_max(1,2);;
Line 1, characters 8-13:
1 | trip_max(1,2);;
            ^^^^^
Error: This expression has type 'a * 'b but an expression was expected of type int
```

(b) Explain why this error is occurring and instruct Tup on how to correctly invoke the function; use 1-2 sentences. [3 pts]

Tup is passing the first argument as a pair (tuple) of 2 integers. The function expects a single integer so compiler is identifying the type problem. Properly calling the function would use no parentheses as in: `trip_max 1 2`

Write down the types inferred for the following OCaml expressions. If there is a type error: write "type error".

(c) [3pts]

```
if true || false then
  if false then
    3 > 4
  else
    false
else
  1 = 9
```

```
bool
```

(d) [3pts]

```
let f a b c =
  if b > c then
    a
  else
    c + 1
in f
```

```
int -> int -> int -> int
```

(e) [3pts]

```
let f =
  fun a b ->
    if a then b
    else b
```

```
bool -> 'a -> 'a
```

Problem 7: Python Programming [Total 15 pts]

An email address is divided into 3 parts. With `user@terpmail.umd.edu` as an example these three parts are

- `user`: The 'local' id which appears before the @ sign and must be at least 3 characters long
- `terpmail`: Subdomains which appear immediately after the @ sign and may contain several 'dotted' portions.
- `umd.edu`: The Root Domain which is the last dotted part of the email

Assume only upper/lowercase letters, numbers, periods (.), and exactly one @ sign may be appear in email addresses, but no other characters.

Write a Python function `popular_email_counts` that takes in a list of email addresses as strings and counts the number of occurrences of each of the following root domains:

`(1) gmail.com (2) yahoo.com (3) umd.edu`

The counts are returned as a dictionary with the root domains as keys and the counts as values. If an allowed Root Domain occurs 0 times, it may be included in the dictionary with 0 count or excluded from it.

Email addresses that do not follow specifications or are not in one of the allowed Root Domains above are ignored and do not contribute to any count.

EXAMPLE:

Valid Emails	**Invalid Emails**
abc@umd.edu	no-dashes@umd.edu
def@umd.edu	notValidDomain@yandex.edu
abc@sub.gmail.com	ab@umd.edu
ABC@gmail.com	caseMattersForDomain@GMAIL.com
123@terpmail.umd.edu	no@allowed@gmail.com
allowed@gmail.com	
DEF@sub.dom.gmail.com	

The results of processing the adjacent email addresses as a list is one of:

```
{'umd.edu':3, 'gmail.com':4}
   OR
{'umd.edu':3, 'yahoo.com':0, 'gmail.com':4}
```

CONSTRAINT: You may NOT use Python's `split()` method. Instead, use other Regular Expression-based processing functions to check for email matches and dissect their parts.

```
import re
from functools import reduce

## SOLUTION 1
def popular_email_counts(emails):
  domains = {"gmail.com",
             "yahoo.com",
             "umd.edu"}
  counts = {}
  regex = r"[A-Za-z0-9.]{3,}@([A-Za-z0-9.]+\.)*([A-Za-z0-9]+\.[A-Za-z0-9.]+)"
  for email in emails:
    m = re.match(regex,email)
    if m and m[2] in domains:
      if m[2] not in counts:
        counts[m[2]] = 0
      counts[m[2]] = counts[m[2]]+1
  return counts

## More solutions below
```

```
## SOLUTION 2
def popular_email_counts(emails):
  email_re = r"([a-zA-Z0-9\.]{3,})+@([a-z]+\.)*(umd\.edu|((gmail|yahoo)\.com))"
  roots = reduce(lambda x,y: x+[y.group(3)] if y else x,
                 list(map(lambda x: re.fullmatch(email_re,x),emails)),[])
  ret = {}
  for x in roots:
    if x not in ret:
        ret[x] = 0
    ret[x] = ret[x] + 1
  return ret
```

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
```

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')
[abc]	match any character in abc
[^r_1]	anything except r_1 (eg. [^abc] 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
^	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

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