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

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>10</td>
</tr>
<tr>
<td>Q2</td>
<td>15</td>
</tr>
<tr>
<td>Q3</td>
<td>15</td>
</tr>
<tr>
<td>Q4</td>
<td>15</td>
</tr>
<tr>
<td>Q5</td>
<td>20</td>
</tr>
<tr>
<td>Q6</td>
<td>15</td>
</tr>
<tr>
<td>Q7</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
</tr>
</tbody>
</table>
**Problem 1: Language Concepts**

Any regular expression can be expressed as a Context Free Grammar  **True**  **False**

let f x = x 4 is an example of a higher order function  **True**  **False**

One could theoretically code project 1 in lambda calculus  **True**  **False**

All statically typed languages use explicit (manifest) typing  **True**  **False**

FSMs are a subset of Turing Machines in terms of computational power  **True**  **False**

**Problem 2: Typing**

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  

(b) 'a -> 'a -> bool -> 'a

Given the following OCaml expressions, write down its type.

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

(d) fun a b c d -> if a && let x = b > c in x then d + 1 else b

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

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.  

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

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

\^[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.

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

Problem 4: Context Free Grammars

Consider the following Grammars:

<table>
<thead>
<tr>
<th>Grammar 1</th>
<th>Grammar 2</th>
<th>Grammar 3</th>
<th>Grammar 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S -&gt; AB</td>
<td>S -&gt; ASB</td>
<td>a</td>
<td>S -&gt; Sc</td>
</tr>
<tr>
<td>A -&gt; aAa</td>
<td>a</td>
<td>A -&gt; aA</td>
<td>a</td>
</tr>
<tr>
<td>B -&gt; bBbb</td>
<td>c</td>
<td>B -&gt; bbB</td>
<td>c</td>
</tr>
</tbody>
</table>

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

1 Grammar 1  2 Grammar 2  3 Grammar 3  N None

(b) Ambiguity

| "aabb" is an ambiguous string in Grammar 1 | Y | N |
| "aaabbb" is an ambiguous string in Grammar 2 | Y | N |
| "aaabbc" is an ambiguous string in Grammar 3 | Y | N |

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

A aaacbbb  B aaacbbb  C ccaaaaabbbb  D cacacbbb  E None
Problem 5: Finite State Machines

(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.

NFA:

DFA:

(b) Which of the following are the final states? Select all that apply

(c) Write a regex to describe the language of the above NFA
Problem 6: Lambda Calculus

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

(a) \((\lambda y.\, y y)((\lambda x.\, y)(\lambda y.\, x y))\) [2 pts]

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

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

(c) \((\lambda y.\, y y)((\lambda x.\, y)(\lambda y.\, x y))\) [2 pts]

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

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

- A \((\lambda z.\lambda x.z y)\)
- B \((\lambda y.\lambda x.x y)\)
- C \((\lambda z.\lambda x.x y)\)
- D \((\lambda x.\lambda y.x y)\)
- E None

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

- A \(c\)
- B \((\lambda x.\, x)c\)
- C \((\lambda y.\, y y)\)
- D \((\lambda x.x (c c))\)
- E \(c c\)
- F Infinite Recursion
- G None
Problem 7: Python Programming

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

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

```python
# 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]

```python
# 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
```

```python
def sumnum(s):
```
**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]

# 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]

# 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

---

**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

[r1|r2] r1 or r2 (eg. a|b means 'a' or 'b')
[r1|r2|r3] r1 or r2 or r3 (eg. [abc] is 'a' or 'b' or 'c')
[^r1] anything except r1 (eg. [^abc] is anything but an 'a', 'b', or 'c')
[r1-r2] 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
(m,n) at least m 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
(r1) capture the pattern r1 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): $(\Sigma, R, r_0, F_d, \sigma_n)$

$R \leftarrow \emptyset$

$r_0 \leftarrow \epsilon - \text{closure}(\sigma, q_0)$

while $\exists$ an unmarked state $r \in R$ do

mark $r$

for all $a \in \Sigma$ do

$E \leftarrow \text{move}(\sigma, r, a)$

$e \leftarrow e - \text{closure}(\sigma, E)$

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

$F_d \leftarrow \{r \mid \exists s \in r \text{ with } s \in F_n\}$

Grammars

<table>
<thead>
<tr>
<th>Regex</th>
<th>Lambda Calc</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>$\lambda x.e$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>$e$</td>
</tr>
<tr>
<td>$RR$</td>
<td>$e\ e$</td>
</tr>
<tr>
<td>$R</td>
<td>R$</td>
</tr>
<tr>
<td>$R^*$</td>
<td></td>
</tr>
</tbody>
</table>

Lambda Calc Encodings

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

$\lambda x.\lambda y.\ x = \text{true}$

$\lambda x.\lambda y.\ y = \text{false}$

$e_1\ e_2\ e_3 = \text{if } e_1 \text{ then } e_2 \text{ else } e_3$