

# CMSC330 - Organization of Programming Languages Spring 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
Q1	10
Q2	7
Q3	15
Q4	15
Q5	12
Q6	15
Q7	18
Q8	8
EC	5
Total	100 + 5

**Problem 1: Language Concepts**

[Total 10 pts]

$(\lambda x.abx)$ is alpha-equivalent to $(\lambda c.xyc)$	<b>True</b>	<b>False</b>
For statically typed languages, type checking occurs during the parsing phase	<b>True</b>	<b>False</b>
Dangling Pointers are prevented in Rust	<b>True</b>	<b>False</b>
Lifetimes are part of a variable's type in Rust	<b>True</b>	<b>False</b>
"Missing semicolon on line 12" is an error that would be raised during evaluation	<b>True</b>	<b>False</b>
$S \rightarrow S - S   n$ is an ambiguous grammar	<b>True</b>	<b>False</b>
Grammar is a subset of Syntax	<b>True</b>	<b>False</b>
Mark and Sweep is faster than Reference Counting on average	<b>True</b>	<b>False</b>
A rust function with the following header will compile: <code>fn myst(a:&amp;str, b:&amp;u32, c:&amp;u32) -&gt; &amp;str</code>	<b>True</b>	<b>False</b>
OCaml's <code>let x = x + 1 in x</code> is operationally the same as Ruby's <code>x = x + 1</code>	<b>True</b>	<b>False</b>

**Problem 2: Regex**

[Total 7 pts]

(a) Which of the following strings are accepted by the regular expression below?

 $/[\lambda\delta\sigma]^+\omega|\beta/$ 

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

[3 pts]

 $\lambda\lambda\beta$  $\delta$  $\delta\omega\lambda$  $\sigma\lambda\beta\beta$  $\omega\beta$ 

NONE

(b) Write a regular expression that describes a comma separated integer list of odd length.

[4 pts]

	Valid	Invalid
Examples:	1	1,2
	1,2,3	1.3
	-6,-1,-3	

### Problem 3: Higher Order Functions

Given the following type, write an expression that matches that type. You may not use type annotations and all pattern matching must be exhaustive. **You must use map or fold in your answer**

(a) `string list -> string`

(b) `'a list -> 'b list -> ('a list -> 'b -> 'a list) -> ('a -> 'c) -> 'c list`

Given the expression, write down its type. **You will need to evaluate it first**

(c) `fun a b c -> if a b then [b+1] else c`

(d) `(fun x -> fun y -> y x) ((fun y -> y + 1) 5)`

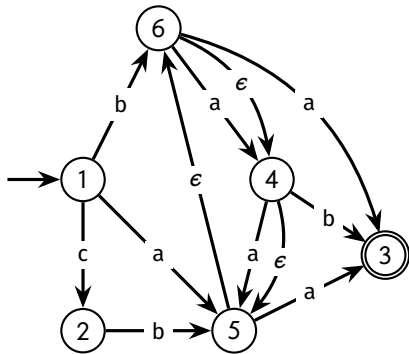
(e) `let c = if true then false else true in fun a -> fun b c -> b c > a c`

## Problem 4: Finite State Machines

[Total 15 pts]

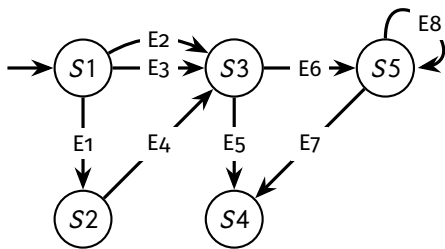
Using the subset algorithm, convert the following NFA to a DFA, and fill in the blanks appropriately matching the DFA provided with the right nodes and transitions. Only the blanks will be graded.

NFA:



Scratch Space (if needed)

DFA:



S1:  S2:  S3:  S4:  S5:

E1:  E2:  E3:  E4:

E5:  E6:  E7:  E8:

Final States:

S1

S2

S3

S4

S5

## Problem 5: Operational Semantics

Consider the following rules for 2 Languages, using Ruby as the Metalanguage:

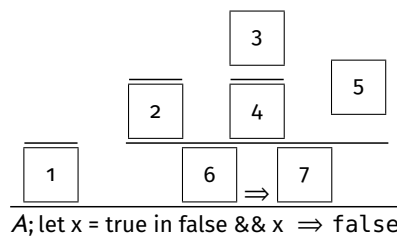
Language 1:	Language 2
$\frac{}{\text{true} \rightarrow \text{true}}$	$\frac{}{\text{true} \rightarrow \text{true}}$
$\frac{}{\text{false} \rightarrow \text{false}}$	$\frac{}{\text{false} \rightarrow \text{false}}$
$\frac{A(x) = v}{A; x \Rightarrow v}$	$\frac{A(x) = v}{A; x \Rightarrow v}$
$\frac{A; e_1 \Rightarrow v_1 \quad A; e_2 \Rightarrow v_2 \quad v_3 = v_1 \text{ and } v_2}{A; e_1 \ \&\& \ e_2 \Rightarrow v_3}$	$\frac{A; e_1 \Rightarrow v_1 \quad A; e_2 \Rightarrow v_2 \quad v_3 = v_1 \text{ and } v_2}{A; (\lambda x. \lambda y. x y x) e_1 \ e_2 \Rightarrow v_3}$
$\frac{A; e_1 \Rightarrow v_1 \quad A, x : v_1; e_2 \Rightarrow v_2}{A; \text{let } x = e_1 \text{ in } e_2 \Rightarrow v_2}$	$\frac{A; e_2 \Rightarrow v_1 \quad A, x : v_1; e_1 \Rightarrow v_2}{A; (\lambda x : e_1) e_2 \Rightarrow v_2}$

(a) Convert the following Language 1 sentence to it's language 2 counterpart

$A; \text{let } x = \text{true in false} \ \&\& \ x$

(b) Complete the opsem proof for the following program using Language 1:

$\text{let } x = \text{true in false} \ \&\& \ x$



$A; \text{true} \Rightarrow \text{true} \quad A; x : \text{true}; \text{false} \ \&\& \ x \Rightarrow \text{false}$

Blank 1:  $A; \text{true} \Rightarrow \text{true}$

Blank 2:  $\text{false} \Rightarrow \text{false}$

Blank 3:  $\text{false} \Rightarrow \text{false}$

Blank 4:  $\text{false} \Rightarrow \text{false}$

Blank 5:  $\text{false} \Rightarrow \text{false}$

Blank 6:  $A; x : \text{true}; \text{false} \ \&\& \ x$

Blank 7:  $\text{false}$

## Problem 6: Lambda Calculus

[Total 15 pts]

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

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

[3 pts]

Perform a single  $\beta$ -reduction using Eager (call by value) evaluation on the outermost expression. If you cannot reduce it, write **Beta Normal Form**. Do **not**  $\alpha$ -convert your final answer.

(b)  $(\lambda x. a b c)((\lambda x. (x x)) x)$

[3 pts]

Convert the following expressions to Beta Normal Form. If it is already in Beta Normal Form, circle BNF. If the answer is not given, circle None.

(c)  $(\lambda x. \lambda y. x y)((\lambda b. b b) y)$

[3 pts]

 $\lambda y. y y y$  $\lambda y. x x y$  $\lambda a. y y a$  $y y y$ 

BNF

infinite recursion

None

(d)  $(\lambda x. x x x) (\lambda x. x x x)$

[3 pts]

 $(\lambda x. x x x)$  $x x x$  $(\lambda x. x x x)(\lambda x. x x x)$  $x$ 

BNF

infinite recursion

None

(e)  $\lambda x. (\lambda b. a b) (\lambda b. a b)$

[3 pts]

 $\lambda x. (\lambda b. a b)$  $(\lambda b. a b)$  $a b$  $\lambda x. a \lambda b. a b$ 

BNF

infinite recursion

None

## Problem 7: Coding

Consider the following Grammar, where  $n$  is any integer:

$$\begin{aligned} S &\rightarrow N + S|(N) \\ N &\rightarrow n \end{aligned}$$

(a) Ruby Lexer

Write a lexer for this grammar in **Ruby**, you may use the following as tokens

```
# tokens: n, "Plus", "RParen", "LParen"
# example input-output
lex("2 * -5 + 6") = IOError
lex("2 -7 9 -10") = ["2", "-7", "9", "-10"]
lex("(-2) + (3)") = ["LParen", "-2", "RParen", "Plus", "LParen", "3", "RParen"]
#If an error occurs, you may raise an error
raise IOError.new("Error")
```

```
def lex(str)
```

## (b) Ocaml Parser

[10 pts]

Using the same grammar as before, where  $n$  is any integer:

$$\begin{aligned} S &\rightarrow N + S | (N) \\ N &\rightarrow n \end{aligned}$$

Write a parser for the  $S$  non-terminal in **OCaml**. You may use the following types and functions:

```
type tok = Int of int | Plus | RParen | LParen
```

```
type tree = Add of tree * tree | Leaf of int
```

```
let lookahead toks = match toks with [] -> None | h::t -> Some h
```

```
let match_tok toks tok = match toks [] -> raise Error | h::t when h = tok -> t | _ -> raise Error  
(* You may assume raise Error is valid and compiles *)
```

You may assume there is a `parse_n` function of type `tok list → (tree * tok list)` and that it is correct.

The type of `parse_s` is `tok list → (tree * tok list)`

```
let rec parse_s toks =
```



## Problem 8: Rust

```

1 fn main(){
2     let m = String::from("Hello");
3     let t = String::from("World");
4     { let y = m;
5       { let c = myfunc(y,t);
6         let d = &c;
7       }
8     }
9 }
10
11 fn myfunc<'a>(a:String, b: String) -> String{
12     if a.len() > b.len() {a} else {b}
13 }

```

## Problem 9: Extra Credit

What is your favorite pun?

## Problem 10: Extra Credit

Who is your discussion TA and what is your section number?

## Ownership

If there is no owner, write "NONE".

Who is the owner of "Hello" immediately after line 11 is run?

Who is the owner of "World" immediately after line 5 is run?

## Lifetimes

What is the last line executed before "Hello" dropped?

What is the last line executed before "World" dropped?

You may use this area as scratch space