CMSC330 - Organization of Programming Languages
Spring 2024 - Exam 2 Solutions

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

- Please write legibly. **If we cannot read your answer you will not receive credit.**
- You may use anything on the accompanying reference sheet anywhere on this exam
- Please remove the reference sheet from the exam
- The back of the reference sheet has some scratch space on it. If you use it, you must turn in your scratch work
- 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.	6
P3.	8
P4.	14
P5.	12
P6.	14
P7.	15
P8.	16
P9.	5
Total	100

Problem 1: Language Concepts

[Total 10 pts]

	True	False
Context Free Grammars can describe strings that contain an arbitrary number of balanced parentheses	● T	F
Context Free Grammars cannot describe strings that contain an arbitrary number of balanced parentheses	T	● F
The lexing phase of an interpreter checks the grammar of the input.	T	● F
The evaluating phase of an interpreter checks the grammar of the input.	T	● F
Type checking is a separate process from evaluation	● T	F
Type checking is not a separate process from evaluation	T	● F
A language that uses dynamic typing will have type errors during runtime and not compile time	● T	F
A language that uses static typing will have type errors during runtime and not compile time	T	● F
Property based testing is intended to be a complete replacement to unit testing	T	● F
Property based testing is not intended to be a complete replacement to unit testing	● T	F
Operational semantics can be used to prove the correctness of a program	● T	F
Operational semantics cannot be used to prove the correctness of a program	T	● F
Every language uses the same typing rules.	T	● F
Ocaml is a statically typed language	● T	F
Ocaml is a dynamically typed language	T	● F
If a language is well-defined, it is also well-typed	T	● F
If a program is well-defined, it is also well-typed	T	● F
If a language is well-typed, then it is also well-defined.	T	● F
If a program is well-typed, then it is also well-defined.	T	● F
Context Free Grammars can describe all regular expressions	● T	F
Context Free Grammars cannot describe all regular expressions	T	● F

Problem 2: Context Free Grammars - Acceptance

[Total 6 pts]

Which of the following strings can be derived using CFG below?

$$
\begin{aligned}
E &\rightarrow M + E \mid M - E \mid M \\
M &\rightarrow N > M \mid N < M \mid N \\
N &\rightarrow n \mid b \mid (E)
\end{aligned}
$$

Note: $n \in \mathbb{Z}, b \in \{true, false\}$

- (A) `1 3 7`
- (B) `(((6 - 7)))`
- (C) `true > false`
- (D) `true < (6) + 7`
- (E) `((false + true) > (0 - 0))`
- (F) `() > true`
- (G) `6 + false`
- (H) `4 9`
- (I) `true < (true`
- (J) `true < (16 + (1))`
- (K)[a] `((false + true) > (0 - b))`
- (L) `true > 2 - 3`
- (M) `1 3`
- (N) `()`
- (O) `false < true`
- (P)[b] `true - false + (b)`
- (Q) `1 > false`
- (R) `true < {6} + 7`
- (S) `((false + (n) > (0 - 0))`
- (T) `(13) > true`

[a] if b is not a character
[b] if b is not a character

Problem 3: Context Free Grammars - Derivations

[Total 8 pts]

$$
\begin{aligned}
E \rightarrow &\quad M + E \mid M - E \mid M \\
M \rightarrow &\quad N > M \mid N < M \mid N \\
N \rightarrow &\quad n \mid b \mid (E)
\end{aligned}
$$

Note: $n \in \mathbb{Z}, b \in \{true, false\}$

Using only a **left-most** derivation, and the above grammar, derive the string "false > (true + 7)" (do not draw a tree).

$E \rightarrow M$
$\rightarrow N > M$
$\rightarrow false > M$
$\rightarrow false > N$
$\rightarrow false > (E)$
$\rightarrow false > (M + E)$
$\rightarrow false > (N + E)$
$\rightarrow false > (true + E)$
$\rightarrow false > (true + M)$
$\rightarrow false > (true + N)$
$\rightarrow false > (true + 7)$

Using only a **left-most** derivation, and the above grammar, derive the string "8 > (2 - false)" (do not draw a tree).

$E \rightarrow M$
$\rightarrow N > M$
$\rightarrow 8 > M$
$\rightarrow 8 > N$
$\rightarrow 8 > (E)$
$\rightarrow 8 > (M - E)$
$\rightarrow 8 > (N - E)$
$\rightarrow 8 > (2 - E)$
$\rightarrow 8 > (2 - M)$
$\rightarrow 8 > (2 - N)$
$\rightarrow 8 > (2 - false)$

Using only a **left-most** derivation, and the above grammar, derive the string "false > (7 - 0)" (do not draw a tree).

$E \rightarrow M$
$\rightarrow N > M$
$\rightarrow false > M$
$\rightarrow false > N$
$\rightarrow false > (E)$
$\rightarrow false > (M - E)$
$\rightarrow false > (N - E)$
$\rightarrow false > (7 - E)$
$\rightarrow false > (7 - M)$
$\rightarrow false > (7 - N)$
$\rightarrow false > (7 - 0)$

Using only a **left-most** derivation, and the above grammar, derive the string "true > (1 - false)" (do not draw a tree).

$E \rightarrow M$
$\rightarrow N > M$
$\rightarrow true > M$
$\rightarrow true > N$
$\rightarrow true > (E)$
$\rightarrow true > (M - E)$
$\rightarrow true > (N - E)$
$\rightarrow true > (1 - E)$
$\rightarrow true > (1 - M)$
$\rightarrow true > (1 - N)$
$\rightarrow true > (1 - false)$

Problem 4: Context Free Grammars - Creation

[Total 14 pts]

Design a Context Free Grammar using the alphabet $\{$a,b$\}$.

- Accepted strings must be of length 0 or more
- Accepted strings must contain an equal number of a's and b's
- You must accept strings with a's and b's in any order (ab-babbaa)

$S \longrightarrow SaSb \mid SbAs \mid \epsilon$

Design a Context Free Grammar using the alphabet $\{$c,d$\}$.

- Accepted strings must be of length 2 or more
- Accepted strings must contain an equal number of c's and d's
- You must accept strings with c's and d's in any order (cdd-cddcc)

$S -> TcTd \mid TdTc$
$T -> TcTd \mid TdTc \mid \epsilon$

Design a Context Free Grammar using the alphabet $\{$p,m$\}$.

- Accepted strings must be of length 0 or more
- Accepted strings must contain an equal number of p's and m's
- You must accept strings with p's and m's in any order (pmmpmmpp)

$S \longrightarrow SpSm \mid SmAp \mid \epsilon$

Design a Context Free Grammar using the alphabet $\{$x,y$\}$.

- Accepted strings must be of length 0 or more
- Accepted strings must contain an equal number of x's and y's
- You must accept strings with x's and y's in any order (xyyxyyxx)

$S \longrightarrow SxSy \mid SyAx \mid \epsilon$

Problem 5: Lexing Parsing and Evaluating

[Total 12 pts]

Given the following CFG, and assuming the **Ocaml** type system and semantics, at what stage of language processing would each expression **fail**? Mark **'Valid'** if the expression would be accepted by the grammar and evaluate properly. Assume the only symbols allowed are those found in the grammar. Choose only one choice for each expression. Grammar:

$$
\begin{aligned}
E &\rightarrow \quad M + E \mid M - E \mid M \\
M &\rightarrow \quad N > M \mid N < M \mid N \\
N &\rightarrow \quad n \mid b \mid (E)
\end{aligned}
$$

Note: $n \in \mathbb{Z}, b \in \{true, false\}$

	Lexer	Parser	Evaluator	Valid
`let x = 4 in 5`	● L	○ P	○ E	○ V
`(true) - false > 8`	○ L	○ P	● E	○ V
`8 * 5 - 15`	● L	○ P	○ E	○ V
`-1 - -10`	○ L	○ P	○ E	● V
`(((((false)))))`	○ L	○ P	○ E	● V
`1.4 > 4`	● L	○ P	○ E	○ V
`(b) - false > 8`	● L	○ P	● E	○ V
`8 + true - 15`	○ L	○ P	● E	○ V
`(((((123))))`	○ L	● P	○ E	○ V
`1 < 4.1`	● L	○ P	○ E	○ V
`if 3 > 4 then 5 else 9`	● L	○ P	○ E	○ V
`((E))`	● L	○ P	○ E	○ V
`3 > 8 > 0`	○ L	○ P	● E	○ V
`(true > (false))`	○ L	○ P	○ E	● V
`true false`	○ L	● P	○ E	○ V
`2 > true + 9`	○ L	○ P	● E	○ V
`(true) - false > n`	● L	○ P	● E	○ V
`8 * 5 - false`	● L	○ P	○ E	○ V
`true > 4`	○ L	○ P	● E	○ V

Problem 6: Coding and Debugging

[Total 14 pts]

Recall the interpreter code done in discussion/project 4/lecture. Debug the following code used to parse the grammar. There a variety of type and logic errors. You only need to identify (1) type error and two (2) logic bugs. For the logic bugs, we provide an input that returns the incorrect value. Things that would cause warnings are not bugs in this case.
The following is the correct code with no errors

```
Grammar:
E -> M + E | M - E | M
M -> N > M | N < M | N
N -> n | b | (E)
(* n is any int, and b is any bool *)

type token = Tok_Plus | Tok_Minus | Tok_LT | Tok_GT | Tok_LParen | Tok_RParen
             | Int of int | Boolean of bool
type ast = Add of ast * ast | Sub of ast * ast
           | LT of ast * ast | GT of ast * ast | Num of int | Bool of bool

let match_token toks tok = match toks with
     [] -> raise (Failure("Error"))
    |h::t when h = tok -> t
    |h::_ -> raise (Failure("Error"))

let lookahead toks = match toks with
     h::t -> h
    |_ -> raise (Failure("Error"))

Parser Code:
let rec parse toks =
    let (toks, tree) = parse_E toks in
    if toks = [] then tree else raise (Failure("Nope"))

and parse_E toks = let (toks,tree1) = parse_M toks in match lookahead toks with
    Tok_Plus ->  let t = match_token toks Tok_Plus in
                 let (toks,tree2) = parse_E t in (toks,Add(tree1,tree2))
   |Tok_Minus -> let t = match_token toks Tok_Minus in
                 let (toks,tree2) = parse_E t in (toks,Sub(tree1,tree2))
   | _ -> (toks,tree1)

and parse_M toks = let (toks,tree1) = parse_P toks in match lookahead toks with
    Tok_LT -> let t = match_token toks Tok_LT in
              let (toks,tree2) = parse_M t in (toks,LT(tree1,tree2))
   |Tok_GT -> let t = match_token toks Tok_GT in
              let (toks,tree2) = parse_M t in (toks,GT(tree1,tree2))
   | _ -> (toks,tree1)

and parse_P toks = match lookahead toks with
    Int(x) -> let t = match_token Int(x) in (t,Num(x))
   |Boolean(x) ->  let t = match_token Boolean(x) in (t,Bool(x))
   |Tok_LParen -> let t = match_token toks Tok_LParen in
                   let (toks,tree) = parse_E t in (match toks with
                                                  Tok_RParen::t -> t,tree
                                                  |_ -> raise (Failure("Nope")))
Incorrect input:
(parse [Boolean(false); Tok_LT; Tok_LParen; Int(5); Tok_RParen])
```

These errors are dependent on your version, as there are a variety for each version. Please make sure you look at what version you have. Additionally, these solutions state what the error is, the fix is above.

(a) **Type Errors** [4 pts]

Line	Fix
10	make sure you are machining lookahead and not just with the toks.
17	need to return both token and tree, not just tree
18	need to return both token and tree, not just tree

(b) **Logic Errors** [5 pts]

Line	Fix
4	The parse call should be parse_M, not parse_E
6	The parse call should be parse_E, not parse_M
10	The parse call should be parse_P, not parse_M
14	The parse call should be parse_P, not parse_M
20	make sure to match the right token list (toks vs t)
21	make sure to return the right token list (toks vs t)

Problem 7: Property Based Testing

[Total 15 pts]

Consider the following functions and type definitions:

```
type transition = (int * char option * int)
type nfa = {alphabet: char list; Qs: int list; q0: int; fs: int list; delta: transition list}

let rec e_closure nfa state =
    fold_left (fun a (s,c,d) -> if c = None then d::a else a) [state] nfa.delta

let rec move nfa state symbol =
    fold_left (fun a (s,c,d) -> if c = symbol && s = state then d::a else a) [] nfa.delta
```

Below is a description of the property being tested and its attempted implementation. Please indicate if the function does in fact test the property, and if the property is valid to test. If the property is valid, indicate if the property will catch the bugs in the above code **regardless of the implementation**. If the property is invalid, put NA to catch bugs

(a) Property 1 [5 pts]

Property: E-closure should always have at least one element

Implementation: `fun nfa state -> List.len(e_closure nfa state) > 0`

Valid implementation: **(Y)** (N) Valid property: **(Y)** (N) Would catch bugs: (Y) **(N)** (na)

(b) Property 2 [5 pts]

Property: E-closure upon a state should always have that state in the result

Implementation: `fun nfa state -> List.mem state (e_closure nfa state)`

Note: `List.mem x lst returns true if x is an element of lst`

Valid implementation: **(Y)** (N) Valid property: **(Y)** (N) Would catch bugs: (Y) **(N)** (na)

(c) Property 3 [5 pts]

Property: Move upon a state with Epsilon should result in the same as the eclsoure of that state

Implementation: `fun nfa state -> move nfa state None = e_closure nfa state`

Valid implementation: **(Y)** (N) Valid property: (Y) **(N)** Would catch bugs: (Y) (N) **(na)**

Problem 8: Type Checking

[Total 16 pts]

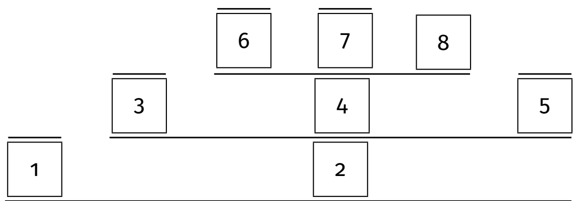
Consider the following Typing Rules for Ocaml:

$$\frac{}{G \vdash \text{true} : \text{bool}} \qquad \frac{}{G \vdash \text{false} : \text{bool}} \qquad \frac{}{G \vdash n : \text{int}}$$

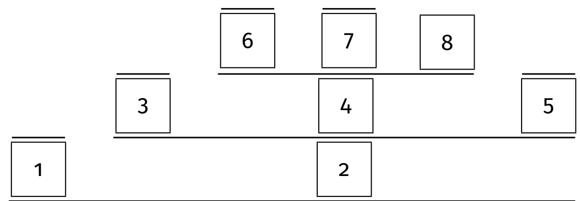
$$\frac{}{G \vdash x : G(x)} \qquad \frac{G \vdash e_1 : int \qquad G \vdash e_2 : int \qquad + = (int, int, int)}{G \vdash e_1 + e_2 : int}$$

$$\frac{G \vdash e_1 : t_1 \qquad G, x : t_1 \vdash e_2 : t_2}{G \vdash let\ x = e_1 \text{ in } e_2 : t_2} \qquad \frac{G \vdash e_1 : bool \qquad G \vdash e_2 : t \qquad G \vdash e_3 : t}{A; if\ e_1\ then\ e_2\ else\ e_3 : t}$$

Complete the typing proof for the following program to prove it is well typed.



$G \vdash$ let x = true in if x then 4 + 8 else 5 : `int`



$G \vdash$ let y = false in if y then 1 + 2 else 9 : `int`

Blank 1: $G \vdash true : bool$ | $G \vdash false : bool$

Blank 2: $G, x : bool \vdash if\ x\ then\ 4 + 8\ else\ 5 : int$ | $G, y : bool \vdash if\ y\ then\ 1 + 2\ else\ 9 : int$

Blank 3: $G, x : bool \vdash x : bool/G(x)$ | $G, y : bool \vdash y : bool/G(y)$

Blank 4: $G, x : bool \vdash 4 + 8 : int$ | $G, y : bool \vdash 1 + 2 : int$

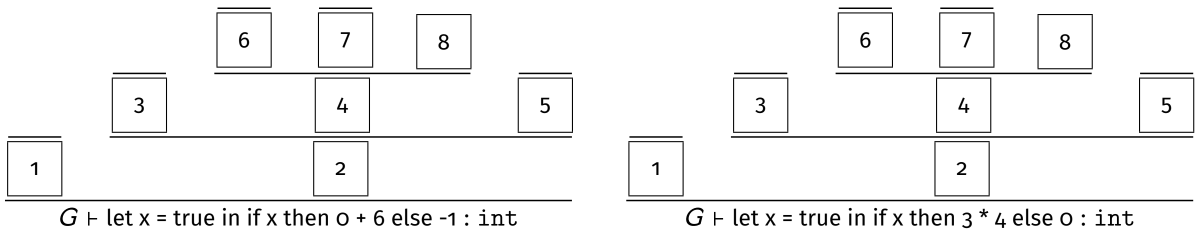
Blank 5: $G, x : bool \vdash 5 : int$ | $G, y : bool \vdash 5 : int$

Blank 6: $G, x : bool \vdash 4 : int$ | $G, y : bool \vdash 1 : int$

Blank 7: $G, x : bool \vdash 8 : int$ | $G, y : bool \vdash 2 : int$

Blank 8: $G, x : bool \vdash + = (int, int, int)$ | $G, y : bool \vdash + = (int, int, int)$

Complete the typing proof for the following program to prove it is well typed.



Blank 1:	$G \vdash true : bool$	$G \vdash true : bool$
Blank 2:	$G, x : bool \vdash if\ x\ then\ 0 + 6\ else\ -1 : int$	$G, x : bool \vdash if\ y\ then\ 3 * 4\ else\ 0 : int$
Blank 3:	$G, x : bool \vdash x : bool/G(x)$	$G, x : bool \vdash y : bool/G(y)$
Blank 4:	$G, x : bool \vdash 0 + 6 : int$	$G, x : bool \vdash 3 * 4 : int$
Blank 5:	$G, x : bool \vdash -1 : int$	$G, x : bool \vdash 0 : int$
Blank 6:	$G, x : bool \vdash 0 : int$	$G, x : bool \vdash 3 : int$
Blank 7:	$G, x : bool \vdash 6 : int$	$G, x : bool \vdash 4 : int$
Blank 8:	$G, x : bool \vdash + = (int, int, int)$	$G, y : bool \vdash * = (int, int, int)$

Problem 9: Operational Semantics

[Total 5 pts]

Consider the following rules for 2 Languages:

Language 1:

$$\frac{}{\text{true} \longrightarrow \text{true}}$$

$$\frac{}{\text{false} \longrightarrow \text{false}}$$

$$\frac{A(x) = v}{A; x \Rightarrow v}$$

$$\frac{A; e_1 \Rightarrow v_1 \qquad A; e_2 \Rightarrow v_2 \qquad v_3 = v_1 \, and \, v_2}{A; e_1 \ \&\& \ e_2 \Rightarrow v_3}$$

$$\frac{A; e_1 \Rightarrow v_1 \qquad A; e_2 \Rightarrow v_2 \qquad v_3 = v_1 \, or \, v_2}{A; e_1 \ || \ e_2 \Rightarrow v_3}$$

$$\frac{A; e_1 \Rightarrow v_1 \qquad A, x : v_1; e_2 \Rightarrow v_2}{A; let x = e_1 \text{ in } e_2 \Rightarrow v_2}$$

Language 2

$$\frac{}{\text{true} \longrightarrow \text{true}}$$

$$\frac{}{\text{false} \longrightarrow \text{false}}$$

$$\frac{A(x) = v}{A; x \Rightarrow v}$$

$$\frac{A; e_1 \Rightarrow v_1 \qquad A; e_2 \Rightarrow v_2 \qquad v_3 = v_1 \, and \, v_2}{A; ((\texttt{fun } x \ y \ \longrightarrow \texttt{if } x \texttt{ then } y \texttt{ else } x) \ e_1 \ e_2) \Rightarrow v_3}$$

$$\frac{A; e_1 \Rightarrow v_1 \qquad A; e_2 \Rightarrow v_2 \qquad v_3 = v_1 \, or \, v_2}{A; ((\texttt{fun } x \ y \ \longrightarrow \texttt{if } x \texttt{ then } x \texttt{ else } y) \ e_1 \ e_2) \Rightarrow v_3}$$

$$\frac{A; e_2 \Rightarrow v_1 \qquad A, x : v_1; e_1 \Rightarrow v_2}{A; (funx \longrightarrow e_1) \ e_2 \Rightarrow v_2}$$

Convert the following Language 1 sentence to its language 2 counterpart

A; let x = true in false && x

$(fun \ x \longrightarrow ((fun \ x \ y \longrightarrow \ if \ x \ then \ y \ else \ x) \ false \ x)) \ true$

A; let x = false in false || x

$(fun \ x \longrightarrow ((fun \ x \ y \longrightarrow \ if \ x \ then \ x \ else \ y) \ false \ x)) \ false$

Convert the following Language 1 sentence to its language 2 counterpart

A; let x = false in x && true

$(fun \ x \longrightarrow ((fun \ x \ y \longrightarrow \ if \ x \ then \ y \ else \ x) \ x \ true)) \ false$

A; let x = false in true || x

$(fun \ x \longrightarrow ((fun \ x \ y \longrightarrow \ if \ x \ then \ x \ else \ y) \ true \ x)) \ false$