CMSC330 Spring 2016 Midterm #2
9:30am/12:30pm/3:30pm
Solution

Name:

____

Discussion Time: **10am** **11am** **12pm** **1pm** **2pm** **3pm**
TA Name (Circle): **Adam** **Anshul** **Austin** **Ayman** **Damien**
Daniel **Jason** **Michael** **Patrick** **William**

Instructions

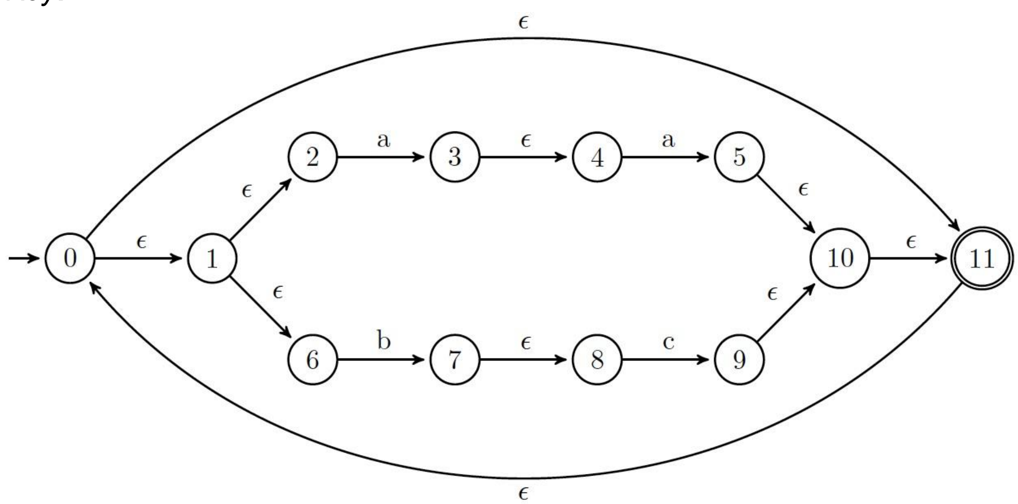
- Do not start this test until you are told to do so!
- You have 75 minutes to take this midterm.
- This exam has a total of 100 points, so allocate 45 seconds for each point.
- This is a closed book exam. No notes or other aids are allowed.
- Answer essay questions concisely in 2-3 sentences. Longer answers are not needed.
- For partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit cannot be given for illegible answers.

	Problem	Score
1	Finite Automata	/20
2	Context Free Grammars	/15
3	Parsing	/10
4	OCaml	/20
5	Programming Language Concepts	/13
6	Operational Semantics	/10
7	Lambda Calculus	/12
	Total	/100

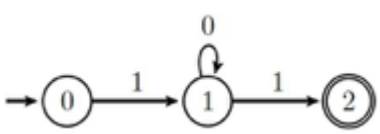
1. Finite Automata (20 pts)

1) (5 pts) Construct an NFA for the following regular expression: (aa|bc)*

Key:

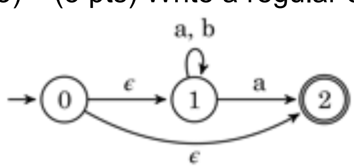


2) (5 pts) Describe the strings accepted by the following NFA.



Key: Binary strings that start and end with 1, and zero or more 0 between them.

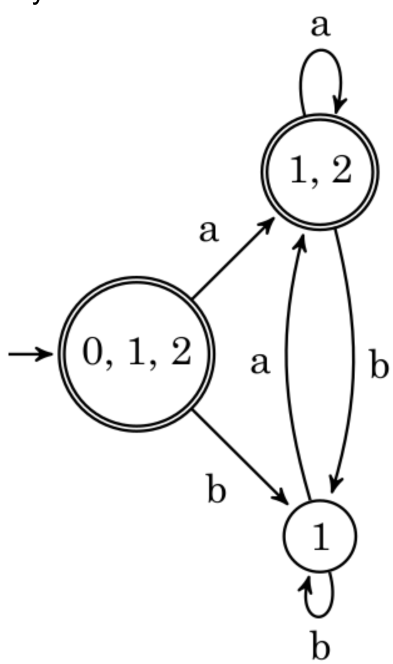
3) (5 pts) Write a regular expression which accepts the same strings as the following NFA.



Key:
ϵ|(a|b)*a

4) (5 pts) Convert the above NFA to a DFA

Key:



2. Context Free Grammars (15 pts)

1) (4 pts) Create a context-free grammar which accepts strings of the following form: $a^n b^{2n+1}$ (n>0) Strings include "b", "abbb", "aabbbbb", etc.

Key:

```
S -> Tb
T -> aTbb|ε
```

2) (6 pts)

a. Show that this grammar CFG is ambiguous,

```
S -> SaS|T
T -> Tb|b
```

***Key:* "babab" (and other strings) can be produced through multiple left-most derivations.**

b. Rewrite the CFG so that it is not ambiguous.

New grammar:

```
S -> TaS|T
T -> Tb|b
```

3) (5 pts) Is the language defined by the following CFG a *regular language*? If so, give an equivalent regular expression. If not, say why it is not regular.

```
S -> AyAyAS|ε
  A -> xA|ε
```

Key: **Regex: (x*yx*yx*)***

3. Parsing (10 pts)

1) (4 pts) Given this CFG, what would be the first set that is generated by S?

S -> ASB | B
A -> wA | w
B -> cB | hB | iB | kB | eB | nB

wchiken

2) (6 pts) I've made the world's best language and named it Sequel. All sequel can do is add and subtract numbers or expressions and output the value of an expression.

type token = | Tok_Num of int | Tok_LParen | Tok_RParen | Tok_Semi | Tok_Print | Tok_Sum | Tok_Sub	type ast = | Num of int | Sum of ast * ast | Sub of ast * ast | Print of ast

Grammar:
printExp -> 'output ' additiveExp ';'
additiveExp -> subtractiveExp ('+' subtractiveExp)*
subtractiveExp -> primaryExp ('-' primaryExp)*

primaryExp -> '(' additiveExp ')' | INITLIT
INTLIT-> ('0' | ('1'..'9') ('0'..'9')*)

Given that you have a lookahead lst, parse_sub lst, parse_primary lst, parse_print lst and parse_int lst of type **token list -> ast * token list**. Write a **parse_add lst** that parses an additive expression. The lookahead function is defined as follows:

```
parse_add lst =
        let (a,h::t) = parse_sub lst in
            match h with
            | Tok_Sum -> let (c,d) = parse_add t in (Sum (a, c), d)
            | _ -> (a,h::t)
```

4. OCaml (20 pts)

```
let rec map f l = match l with
      [] -> []
      | h::t -> let r = f h in r :: map f t
;;

let rec fold f a l = match l with
      [] -> a
   |h::t -> fold f (f a h) t
;;
```

1) (6 pts) Write a function **app** of type
'a list -> ('a -> 'b) list -> 'b list list
that, given a list and a list of functions, applies each function in the function list to the ‘a list. **You should use at least one of fold and map. You cannot have rec in the function definition or in the definition of any helper functions.** (Note: the order of the lists in the list list does not matter).

Examples:

f [1;2;3;4;5] [(fun x -> x); (fun x -> x * x); (fun x -> x * x * x)] =
[[1; 8; 27; 64; 125]; [1; 4; 9; 16; 25]; [1; 2; 3; 4; 5]]
OR
[[1; 2; 3; 4; 5]; [1; 4; 9; 16; 25]; [1; 8; 27; 64; 125]]

let app lst f_lst =

Key:

```
let f lst f_lst = fold (fun a h -> (map h lst)::a) [] f_lst;;
```

2) (8 pts) Recall the concept of an an AST, an intermediate representation of code used before execution. Given the following definition of an AST, write a recursive function `eval_ast` of type `ast -> int` that will evaluate a given AST.

```
type ast =
      |     Sum of ast * ast          (* Adds two sides *)
   |     Double of ast               (* Doubles the result of the child node *)
   |     Num of int                   (* Contains a normal integer *)
;;
```

Example: `eval_ast (Sum(Sum(Num(3),Num(4)),Double(Num(8)))) = 23`

```
let rec eval_ast = function
        | Sum (x, y) -> (eval_ast x) + (eval_ast y)
        | Double x -> 2 * (eval_ast x)
        | Num x -> x
;;
```

3) (6 pts) Two trees are identical when they have same data and arrangement of data is also same. Given the definition of tree

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

Write a function **equal (t1, t2)** of type tree * tree -> bool, which returns true if t1 and t2 are equal and returns false otherwise.

```
let rec equal x =
  match x with
  |(Leaf, Node(_,_,_))->false
  |(Node(_,_,_),Leaf)->false
```

```
  |(Leaf,Leaf)->true
  |(Node(l1,v1,r1),Node(l2,v2,r2))->
          if equal (l1,l2) && v1=v2 && equal (r1,r2) then true else false
;;
```

5. Programming Language Concepts (13 pts)

1) (3 pts) T/**F** OCaml's @ operator is an example of *ad hoc polymorphism*

2) (4 pts) Write the output of following OCaml code.

```
let x = 10 ;;
let f y = x + y;;
let x = 5 ;;
let y = 7 ;;
   let z = f (x + y) ;;
```

What is the value of z:

With Static Scoping	With Dynamic Scoping
22	**17**

3) (3 pts) Consider the following OCaml code fragment.

```
let foo a b = a + b in foo 5 6;;
```

Does the code fragment above produce any side effects?

a) Yes, because foo returns the result of a + b.
b) Yes, because if you typed this into the interpreter you would get back - : int = 11.
c) No, because it doesn't mutate any state.
d) No, because the modifications it makes to the environment are contained in a closure.

4) (3 pts) Signatures are the enabling mechanism for overloading of members in classes. A method's signature is the

a) name of the method and the type of its return value.

b) name of the method and the names of its parameters.
c) name of the method and the data types of its parameters.
d) name of the method, its parameter list, and its return type.

6. Operational Semantics (10 pts)

Domain of *n*: Z
Domain of *b*: {*true, false*}
Domain of *v*: {*n*, *b*}

Axioms:

Number: ________ True: __________ False: ______________________________________
N -> *n* true -> *true* false -> *false*

Arithmetic/Comparison operations:

Op: <u>E1 -> *n1* E2 -> *n2*</u>
E1 op E2 -> *n1 op n2*

(Hint: Arithmetic operations evaluate to values in the domain of *n*. Comparisons evaluate to *b*)

Booleans:

If-T : <u>E1 -> *true* E2 -> *v2*</u>
If E1 then E2 else E3 -> *v2*

If-F: <u>E1 -> *false* E3 -> *v3*</u>
If E1 then E2 else E3 -> *v3*

Fill in the missing pieces of the following deductions.

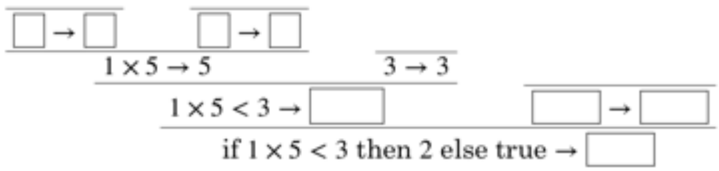
1) (4 pts)

$$\dfrac{\dfrac{\dfrac{}{2 \to 2} \qquad \dfrac{}{3 \to 3}}{2 \times 3 \to 6} \qquad \dfrac{\dfrac{}{\boxed{} \to \boxed{}} \qquad \dfrac{}{\boxed{} \to \boxed{}}}{3 + 4 \to \boxed{}}}{2 \times 3 < 3 + 4 \to \boxed{}}$$

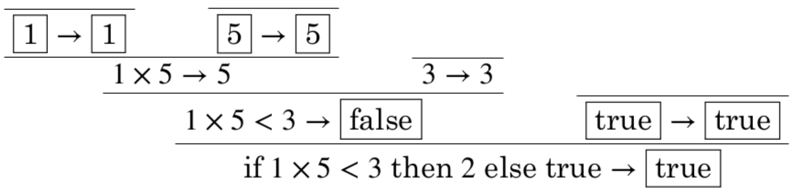
Key:

$$\dfrac{\dfrac{\dfrac{}{2 \to 2} \qquad \dfrac{}{3 \to 3}}{2 \times 3 \to 6} \qquad \dfrac{\dfrac{}{\boxed{3} \to \boxed{3}} \qquad \dfrac{}{\boxed{4} \to \boxed{4}}}{3 + 4 \to \boxed{7}}}{2 \times 3 < 3 + 4 \to \boxed{\text{true}}}$$

2) (6 pts)



Key:



7. Lambda Calculus (12 pts)

1) (4 pts) Make all parentheses explicit in the following expression:

λc. λa. a b (λb. b a) c

Solution: **(**λc.**(**λa.(((a b) **(**λb.(b a)**)**) c)**))** (Bolded paren around boundaries of λ-exprs)

2) (4 pts each) Reduce the expressions as far as possible by showing the intermediate β-reductions and α-conversions.

(λx.λy.x y) (λy.y) b

Solution (+1 for each line):

(λx.λz.x z) (λy.y) b // α-conversion: rename y to z

(λz.(λy.y) z) b // β-reduction: replacing x with (λy.y)

(λy.y) b // β-reduction: replacing z with b

b // β-reduction: replacing y with b

Note: If student fails to α-convert at the beginning, max 2 points if the reduction following that path is correct

3) (4 pts) Given:

succ = λz.λf.λy.f (z f y)

0 = λf.λy.y

1 = λf.λy.f y

2 = λf.λy.f (f y)

Prove that succ 1 = 2

succ 1 =

Solution:

succ 1 =

(λz.λf.λy.f (z f y)) (λf.λy.f y)

λf.λy.f ((λf.λy.f y) f y)

λf.λy.f ((λy.f y) y)

λf.λy.f (f y) = 2