CS421 Summer 2012 Midterm #2

Name:	
NetID:	

- You have **75 minutes** to complete this exam.
- This is a **closed-book** exam. You are allowed one 3×5 inch (or smaller) card of notes (both sides may be used). This card is **not shared**. All other materials (e.g., calculators), except writing utensils are prohibited.
- Do not share anything with other students. Do not talk to other students. Do not look at another students exam. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.
- If you believe there is an error, or an ambiguous question, you may seek clarification from myself or one of the TAs. You must use a whisper, or write your question out. Speaking out aloud is not allowed.
- Including this cover sheet and rules at the end, there are 9 pages to the exam, including one blank page for workspace. Please verify that you have all 9 pages.
- Please write your name and NetID in the spaces above, and also in the provided space at the top of every sheet.

Question	Points	Score
1	15	
2	15	
3	42	
4	28	
Total:	100	

Problem 1. (15 points)

In each of the following parts, given the unification problem, mark **ALL** the rules or errors that are applicable to the underlined pair.

(a) (3 points)

$$\{(f(x,w,z), f(g(g(w)),z,w)), (f(x), f(y)), \underline{(g(x), z)}\}$$

◯ Delete ◯ Decompose √ **Orient** ◯ Eliminate ◯ Error

(b) (3 points)

$$\{(f(x,w,z), f(g(g(w)),z,w)), (f(x), f(y)), \underline{(z, g(x))}\}$$

◯ Delete ◯ Decompose ◯ Orient √ **Eliminate** ◯ Error

(c) (3 points)

$$\{\underline{(f(x,w,g(x)), f(g(g(w)),g(x),w))}, (f(x), f(y))\}$$

◯ Delete √ **Decompose** ◯ Orient ◯ Eliminate ◯ Error

(d) (3 points)

$$\{\underline{(x, g(g(w)))}, (w, g(x)), (g(x), w), (f(x), f(y))\}$$

◯ Delete ◯ Decompose ◯ Orient √ **Eliminate** ◯ Error

(e) (3 points)

$$\{\underline{(w, g(g(g(w))))}, (g(g(g(w))), w), (f(g(g(w))), f(y))\}$$

◯ Delete ◯ Decompose ◯ Orient ◯ Eliminate √ **Error**

Problem 2. (15 points)

(a) (12 points) Solve the following unification problem using the algorithm described in class. Write out intermediate steps, labeling which rule applies, and intermediate substitutions. Let lowercase letters denote constructors (and constants) and uppercase letters denote variables.

$$\{(f(C, l(n)), B); (A, f(C, D)); (A, B)\}$$

Solution: Eliminate $(A, f(C, D))$.

$$\{(f(C, l(n)), B); (f(C, D), B)\} \quad \{A \mapsto f(C, D)\}$$

Orient $(f(C, D), B)$.

$$\{(f(C, l(n)), B); (B, f(C, D))\} \quad \{A \mapsto f(C, D)\}$$

Eliminate $(B, f(C, D))$.

$$\{(f(C, l(n)), f(C, D))\} \quad \{A \mapsto f(C, D), B \mapsto f(C, D)\}$$

Decompose $(f(C, l(n)), f(C, D))$.

$$\{(C, C), (l(n), D)\} \quad \{A \mapsto f(C, D), B \mapsto f(C, D)\}$$

Delete (C, C).

$$\{(l(n), D)\} \quad \{A \mapsto f(C, D), B \mapsto f(C, D)\}$$

Orient $(l(n), D)$.

$$\{(D, l(n))\} \quad \{A \mapsto f(C, D), B \mapsto f(C, D)\}$$

Eliminate $(D, l(n))$.

$$\{\} \quad \{A \mapsto f(C, l(n)), B \mapsto f(C, l(n)), D \mapsto l(n)\}$$

(b) (3 points) Verify that your solution works by using your substitution on the original problem.

Solution:

$$\{(f(C, l(n)), f(C, l(n))); (f(C, l(n)), f(C, l(n))); (f(C, l(n)), f(C, l(n)))\}$$

Problem 3. (42 points)

(a) (6 points) Declare a datatype to represent the tokens of the following grammar:

```
<expr> ::= <prop> | <prop> && <expr> | %<prop>
<prop> ::= true | x | (<expr>)
```

Solution:

```
type token = TokT | TokX | LParen | RParen | Land | TokPer
```

(b) (15 points) Write lexing rules to lex this grammar. You do not need to write an entire `ocamllex` file, but should write the lexing rules (i.e., the part with `parse`).

Solution:

```
rule token = parse
  | "true" { TokT::token lexbuf }
  | "x" { TokT::token lexbuf }
  | "&&" { Land::token lexbuf }
  | "(" { LParen::token lexbuf }
  | ")" { RParen::token lexbuf }
  | EOF { [] }
```

(c) (6 points) Declare datatypes to represent the parse trees of the grammar.

Solution:

```
type expr = Prop of prop | And of (prop * expr) | Perc of prop
 and prop = True | X | Expr of expr
```

(d) (15 points) Write a recursive descent parser for the grammar. You may handle parser errors via pattern match failures. Assume that `<expr>` is the start symbol.

Solution:

```
let parse tokens =
  let rec expr list =
    match list with
    | TokPer::ts -> let (p,s) = prop ts in (Perc p, s)
    | _ -> let (p,s) = prop list
           in match s with
              | Land::ss -> let (e,s') = expr ss in (And (p,e),s')
              | _ -> (Prop p, s)
  and prop list =
    match list with
    | TokT::ts -> (True,ts)
    | TokX::ts -> (X,ts)
    | LParen::ts -> let (e,RParen::s) = expr ts in (Expr e,s)
  in (let (e,[]) = expr tokens in e)
```

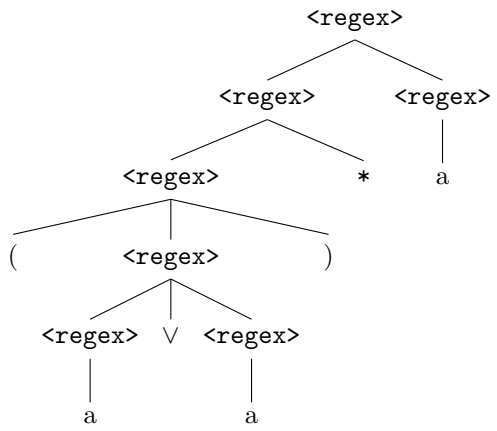
Problem 4. (28 points)

In this question we will use the grammar:

`<regex> ::= a | <regex>* | <regex> ∨ <regex> | <regex><regex> | (<regex>)`

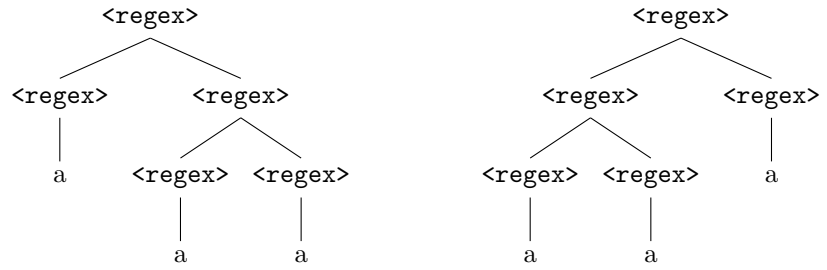
(a) (8 points) Write the parse tree for the string `(a∨a)*a` or write "No Parse" if one does not exist.

Solution:



(b) (7 points) Demonstrate that the grammar is ambiguous.

Solution: Consider the string `aaa` this has two parse trees (shown below).



(c) (13 points) Disambiguate the grammar. Assume that * binds most tightly, followed by juxtaposition (<regex><regex>) which associates right, and ∨ binds most loosely and associates left. Indicate which non-terminal is the start symbol.

Solution:
```
<regex>   ::= <concat> | <regex> ∨ <concat>
<concat>  ::= <base> | <base><concat>
<base>    ::= a | <base>* | (<regex>)
```
Start symbol is <regex>.

A Scratch Space