CMSC 330 HOMEWORK EXERCISES #2

1. Give unambiguous grammars for the following languages:

   In all the grammars shown, single capital letters are used for non-terminals. Unless otherwise stated, S is the start symbol.

   (a) $\{a^n b^n \mid n \geq 0$ and n is even $\}$

   $$S \longrightarrow aaSbb \mid \epsilon$$

   (b) $\{a^i b^j c^{2i+1} d^k \mid i, j, k \geq 0\}$

   $$S \longrightarrow QD$$

   $$Q \longrightarrow Rc$$

   $$R \longrightarrow aRcc \mid P$$

   $$P \longrightarrow Pb \mid \epsilon$$

   $$D \longrightarrow Dd \mid \epsilon$$

   (c) $\{a_1 a_2 ... a_n a_n ... a_2 a_1 \mid a_i \in \{a, b\}, 1 \leq i \leq n\}$

   $$S \longrightarrow aSa \mid bSb \mid aa \mid bb$$

   (d) $\{a^m b^n a^{m+n} \mid m \geq 0$ and $n \geq 1 \}$
   rewrite as $a^m b^n a^n a^m$

   $$S \longrightarrow aSa \mid R$$

   $$R \longrightarrow bRa \mid ba$$

   (e) $\{a^n b^m a^{n-m} \mid n \geq m \geq 0 \}$
   rewrite as $a^{n-m} a^m b^m a^{n-m}$

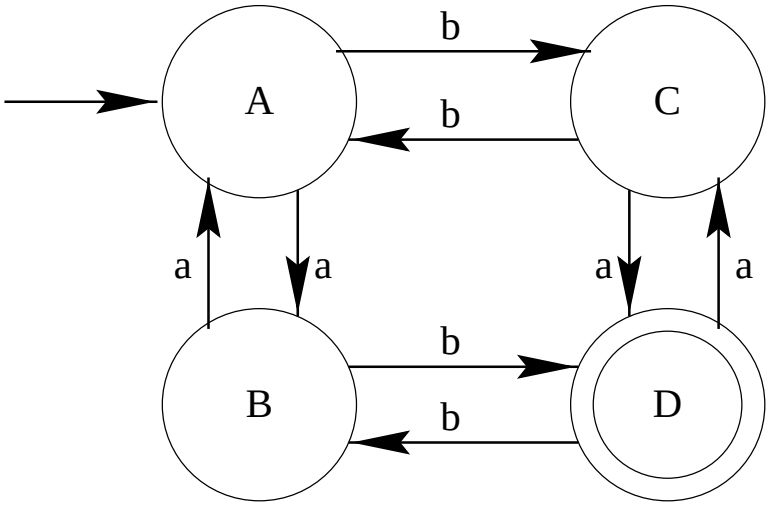
   $$S \longrightarrow aSa \mid R$$

   $$R \longrightarrow aRb \mid \epsilon$$

   (f) All possible sequences of balanced parentheses

   $$S \longrightarrow (S)S \mid \epsilon$$

(g) $\{w \mid w \in \{a,b\}^*$ and w has an odd number of a's and an odd number of b's $\}$.
Use this DFA, and the algorithm given in class to generate the grammar from the DFA. A is the start symbol.



$$A \longrightarrow aB \mid bC$$

$$B \longrightarrow aA \mid bD \mid b$$

$$C \longrightarrow bA \mid aD \mid a$$

$$D \longrightarrow aC \mid bB$$

(h) $\{w \mid w \in \{a,b\}^*$ and w has an equal number of a's and b's $\}$

$$S \longrightarrow aX \mid bY \mid \epsilon$$

$$X \longrightarrow bS \mid aXX$$

$$Y \longrightarrow aS \mid bYY$$

(i) $\{a^m b^n c^p d^q \mid m + n = p + q \}$

$$S \longrightarrow aSd \mid aTc \mid bUd \mid bVc \mid \epsilon$$

$$T \longrightarrow aTc \mid bVc \mid \epsilon$$

$$V \longrightarrow bVc \mid \epsilon$$

$$U \longrightarrow bUd \mid bVc \mid \epsilon$$

(j) $\{a^m b^n \mid m \neq n$ and $m, n \geq 0\ \}$

$$S \longrightarrow AT \mid TB$$

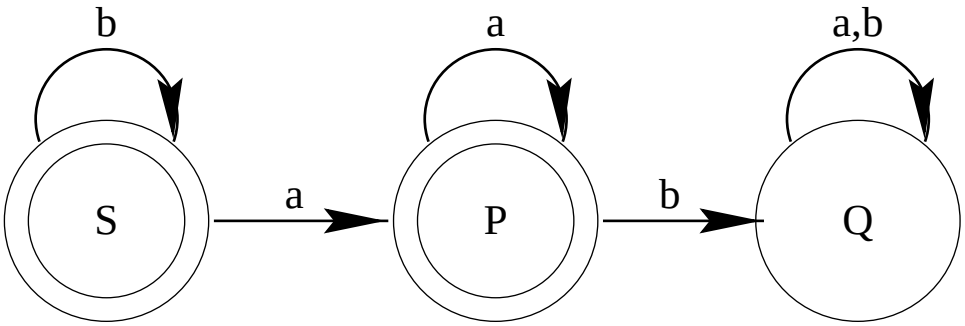
$$A \longrightarrow Aa \mid a$$

$$B \longrightarrow Bb \mid b$$

$$T \longrightarrow aTb \mid \epsilon$$

(k) $\{w \mid w \in \{a, b\}^*$ and w contains no substrings of the form $ab\}$

Use this DFA, and the algorithm given in class to generate the grammar from the DFA. S' is the start symbol.



$$S' \longrightarrow S \mid \epsilon$$

$$S \longrightarrow aP \mid a$$

$$S \longrightarrow bS \mid b$$

$$P \longrightarrow aP \mid a$$

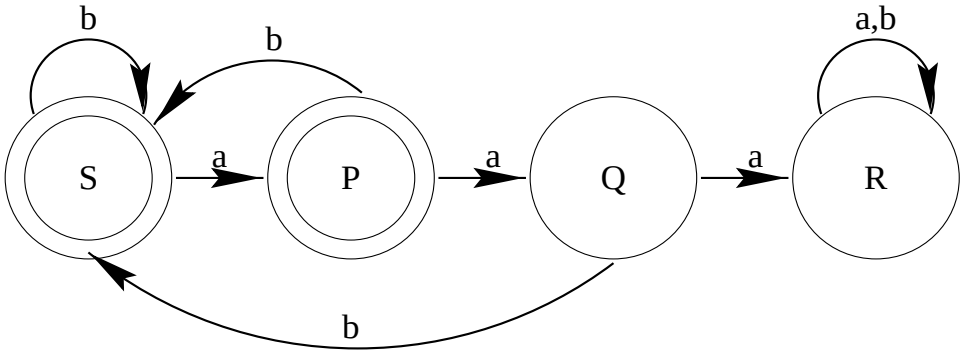
$$P \longrightarrow bQ$$

$$Q \longrightarrow aQ \mid bQ$$

Note that the productions from non-terminal Q are *dead* productions, meaning they can't produce a string of terminals (corresponding to the dead state in the DFA).

(l) $\{w \mid w \in \{a,b\}^*$ and every pair of a's is followed by at least one $b\}$

Use this DFA, and the algorithm given in class to generate the grammar from the DFA. S' is the start symbol.



$$S' \longrightarrow S \mid \epsilon$$
$$S \longrightarrow aP \mid a \mid bS \mid b$$
$$P \longrightarrow aQ \mid a \mid bS \mid b$$
$$Q \longrightarrow aR \mid bS \mid b$$
$$R \longrightarrow aR \mid bR$$

Again, the productions from non-terminal R are dead productions (so are any productions that generate R).

2. Write a grammar for the language

$$\{a^n b^n a^m b^m \mid m,n \geq 1\} \cup \{a^n b^m a^m b^n \mid m,n \geq 1\}$$

If your grammar is ambiguous, prove that it is.

$$S \longrightarrow P \mid Q$$
$$P \longrightarrow AA$$
$$A \longrightarrow aAb \mid ab$$
$$Q \longrightarrow aQb \mid aRb$$
$$R \longrightarrow bRa \mid ba$$

Show two leftmost derivations for the string ***abab***, to show the grammar is ambiguous.

$$S \Longrightarrow P \Longrightarrow AA \Longrightarrow abA \Longrightarrow abab$$
$$S \Longrightarrow Q \Longrightarrow aQb \Longrightarrow aRb \Longrightarrow abab$$

3. Consider a grammar with the following productions:

   <stmtlist> ::= <stmtlist> ; <stmt> | <stmt>
   <stmt> ::= <if stmt> | skip
   <if stmt> ::= if b then <stmtlist> <else part> fi
   <else part> ::= else <stmtlist> | $\in$

   Do these productions cure the "dangling" else problem by eliminating the ambiguity of which <if stmt> the final <else part> is associated with? Does the following grammar cure the problem?

   <Stmt> ::= <UStmt> | <CStmt>
   <UStmt> ::= skip | begin <StmtList> end
   <CStmt> ::= if b then <UStmt> else <Stmt> | if b then <UStmt>
   <StmtList> ::= <StmtList> ; <Stmt> | <Stmt>

   Both these grammars cure the ambiguity. The first uses the *fi* ending an *if* statement to allow the else to be correctly associated with the proper *if* statement.

   The second grammar also cures the problem, be requiring the *then* part of the *if* statement to be surrounded by a *begin*/*end*.

   The best way to see this is to try derivations for each grammar.

4. Grammars can be used to express concepts like associativity and precedence. For each of the following languages, give unambiguous grammars capturing the appropriate associativity and precedence.

   For these grammars, we use Backus-Naur form, meaning that non-terminals are surrounded by <>.

   (a) Operators in APL are right associative and have equal precedence, unless altered by parentheses. The operators +, -, *, / can appear as both monadic (unary) and dyadic (binary) operators. Assignment ($\leftarrow$) is also treated as a binary operator that returns the value assigned as its result. Write an unambiguous context free grammar for APL expressions containing the operands a, b, and c.

   The start symbol is $< apl - expr >$.

$$< apl - expr > \longrightarrow < binary - expr > | < unary - expr >$$

$$< binary - expr > \longrightarrow < unary - expr >< binary - op >< apl - expr >$$

$$< unary - expr > \longrightarrow < unary - op >< unary - expr > | < operand >$$

$$< binary - op > \longrightarrow < unary - op > | \leftarrow$$

$$< unary - op > \longrightarrow \ + \mid - \mid / \mid *$$

$$< operand > \longrightarrow \ a \mid b \mid c \mid (< apl - expr >)$$

(b) Propositional calculus formulas having operators unary $\sim$ and binary $\supset$, operands p and q, and parentheses. The operators should both be right associative and have their usual precedence.

In this problem, usual precedence means $\sim$ has higher precedence than $\supset$.
$< formula >$ is the start symbol.

$$< formula > \longrightarrow < binary - formula >$$

$$< binary - formula > \longrightarrow < unary - formula > \supset < binary - formula > \mid < unary - formula >$$

$$< unary - formula > \longrightarrow \sim < unary - formula > \mid (< formula >) \mid < operand >$$

$$< operand > \longrightarrow \ p \mid q$$

(c) C++ expressions with identifiers `a` and `b` and operators `->`, `*`, and `++`. `->` is a left associative binary operator, `*` is a right associative unary operator, and `++` is a right associative postfix operator. The unary operators have precedence lower than the postfix expressions but higher than all binary operators. For example, the expression `*a++` increments `a` before dereferencing it.

$< expr >$ is the start symbol.

$$< expr > \longrightarrow < binary - expr >$$

$$< binary - expr > \longrightarrow < binary - expr > \rightarrow < unary - expr > \mid < unary - expr >$$

$$< unary - expr > \longrightarrow \ * < unary - expr > \mid < postfix - expr >$$

$$< postfix - expr > \longrightarrow < postfix - expr > ++ \ \mid < id >$$

$$< id > \longrightarrow \ a \mid b$$

5. Context-free grammars cannot express all the restrictions normally placed on programming languages, e.g., that all procedure statements have the same number of actual parameters as the procedure's definition had formals. Consider a LISP-like language which permits a single procedure definition and procedure statement (each with an arbitrary number of parameters) defined by the following grammar:

<S>	::=	(defun p (<Formals>) <Body>) (p <Actuals>)
<Formals>	::=	<Formals> <Id> | $\in$
<Actuals>	::=	<Actuals> <Exp> | $\in$
<Body>	::=	...
<Id>	::=	...
<Exp>	::=	...

Develop another grammar for this language that enforces the restriction that a call must have the same number of arguments as the declaration (you can treat <Body>, <Id> and <Exp> as non-terminals for this purpose).

$$< S > \longrightarrow (defun\ p\ (< M >)$$

$$< M > \longrightarrow < Formal >< M >< Actual > |\ ) < Body >) (p$$

$$< Formal > \longrightarrow < Id >$$

$$< Actual > \longrightarrow < Exp >$$

Is it possible to generalize this to a language in which arbitrarily many calls to a procedure can occur?

The idea of the grammar is that it matches the first formal parameter (in the procedure definition) with the last actual parameter (in the procedure call), the second formal with the second to last actual, etc. So, it doesn't work correctly for more than one procedure call.

As a matter of fact, it is not possible to do this sort of match with a context free grammar (CFG), since a CFG cannot match arbitrarily many symbols.

For example, the language $L = \{a^n b^n c^n \mid n \geq 0\}$ is not a context free language. And this is the formal language that essentially describes matching exactly 2 procedure calls with one procedure definition.