Do Problems B1, B2, and B5.

Do either Problem B3 or Problem B4.

If you do both B3 and B4 you will receive extra credit for the extra problem done.

Problem B1 (60 pts). (i) Consider the following language $L$ over $\{a, b\}^*$:

$$L = \{w \mid \exists n \geq 1, \exists x_i \in a^+, \exists y_i \in b^+, 1 \leq i \leq n, n \text{ is not prime}, w = x_1y_1 \cdots x_ny_n\}.$$

1. Prove that $L$ is not regular.

2. Prove that there is an integer $m \geq 1$ such that, for every $w \in L$, if $|w| \geq m$, then there is some decomposition of $w$ of the form $w = uxv$ satisfying the following properties:

   a. $x \neq \epsilon$.

   b. $ux^iv \in L$ for all $i \geq 0$.

   c. $|ux| \leq m$.

   In other words, the conclusion of the pumping lemma holds for $L$, yet $L$ is not regular.

   **Hint.** Actually we must have $m \geq 8$. Beware of the case $i = 0$ in (b).

Problem B2 (60 pts). Give context-free grammars for the following languages:

1. $L_5 = \{wcw^R \mid w \in \{a, b\}^\ast\}$ ($w^R$ denotes the reversal of $w$)

2. $L_6 = \{a^mb^n \mid 1 \leq m \leq n \leq 2m\}$

   For any fixed integer $K \geq 2$,

   $$L_7 = \{a^mb^n \mid 1 \leq m \leq n \leq Km\}$$

   c. $L_8 = \{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$

   d. $L_9 = \{a^mb^n a^m b^p \mid m, n, p \geq 1\} \cup \{a^mb^{4n} a^p b^{4n} \mid m, n, p \geq 1\}$
(e) \( L_{10} = \{ xy | |x| = 2|y|, x, y \in \{a, b\}^* \} \)

In each case, give a justification of the fact that your grammar generates the desired language.

**Problem B3 (80 pts).** The purpose of this problem is to get a fast algorithm for testing state equivalence in a DFA. Let \( D = (Q, \Sigma, \delta, q_0, F) \) be a deterministic finite automaton. Recall that *state equivalence* is the equivalence relation \( \equiv \) on \( Q \), defined such that,

\[
px \equiv qx \iff \forall z \in \Sigma^* (\delta^*(px, z) \in F \iff \delta^*(qx, z) \in F),
\]

and that *\( i \)-equivalence* is the equivalence relation \( \equiv_i \) on \( Q \), defined such that,

\[
px \equiv_i qx \iff \forall z \in \Sigma^*, \ |z| \leq i (\delta^*(px, z) \in F \iff \delta^*(qx, z) \in F).
\]

A relation \( S \subseteq Q \times Q \) is a *forward closure* iff it is an equivalence relation and whenever \((p, q) \in S\), then \((\delta(p, a), \delta(q, a)) \in S\), for all \( a \in \Sigma\).

We say that a forward closure \( S \) is good iff whenever \((p, q) \in S\), then \( \text{good}(p, q) \), where \( \text{good}(p, q) \) holds iff either both \( p, q \in F \), or both \( p, q \notin F \).

Given any relation \( R \subseteq Q \times Q \), recall that the smallest equivalence relation \( R_\infty \) containing \( R \) is the relation \((R \cup R^{-1})^* \) (where \( R^{-1} = \{(p, q) | (p, q) \in R\} \), and \((R \cup R^{-1})^* \) is the reflexive and transitive closure of \((R \cup R^{-1})\)). We define the sequence of relations \( R_i \subseteq Q \times Q \) as follows:

\[
R_0 = R_\infty
R_{i+1} = (R_i \cup \{(\delta(p, a), \delta(q, a)) | (p, q) \in R_i, \ a \in \Sigma\})_\infty.
\]

(i) Prove that \( R_{i_0+1} = R_{i_0} \) for some least \( i_0 \). Prove that \( R_{i_0} \) is the smallest forward closure containing \( R \).

*Hint.* First, prove that

\[
R_i \subseteq R_{i+1}
\]

for all \( i \geq 0 \). Next, prove that \( R_{i_0} \) is forward closed.

If \( \sim \) is any forward closure containing \( R \), prove by induction that

\[
R_i \subseteq \sim
\]

for all \( i \geq 0 \).

We denote the smallest forward closure \( R_{i_0} \) containing \( R \) as \( R^\dagger \), and call it the *forward closure of \( R \).*

(ii) Prove that \( p \equiv q \) iff the forward closure \( R^\dagger \) of the relation \( R = \{(p, q)\} \) is good.
**Hint.** First, prove that if $R^i$ is good, then

$$R^i \subseteq \equiv.$$  

For this, prove by induction that

$$R^i \subseteq \equiv_i$$

for all $i \geq 0$.

Then, prove that if $p \equiv q$, then

$$R^i \subseteq \equiv.$$  

For this, prove that $\equiv$ is an equivalence relation containing $R = \{(p,q)\}$ and that $\equiv$ is forward closed.

**Problem B4 (80 pts).** This problem is based on the method proved correct in Problem B3 of this Homework. Also, consult Section 2.5 of the notes.

Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$, for any two states $p, q \in Q$, a fast algorithm for computing the forward closure of the relation $R = \{(p,q)\}$, or detecting a bad pair of states, can be obtained as follows: An equivalence relation on $Q$ is represented by a partition $\Pi$. Each equivalence class $C$ in the partition is represented by a tree structure consisting of nodes and (parent) pointers, with the pointers from the sons of a node to the node itself. The root has a null pointer. Each node also maintains a counter keeping track of the number of nodes in the subtree rooted at that node.

Two functions `union` and `find` are defined as follows. Given a state $p$, `find(p, \Pi)` finds the root of the tree containing $p$ as a node (not necessarily a leaf). Given two root nodes $p, q$, `union(p,q, \Pi)` forms a new partition by merging the two trees with roots $p$ and $q$ as follows: if the counter of $p$ is smaller than that of $q$, then let the root of $p$ point to $q$, else let the root of $q$ point to $p$.

In order to speed up the algorithm, using an idea due to Tarjan, we can modify `find` as follows: during a call `find(p, \Pi)`, as we follow the path from $p$ to the root $r$ of the tree containing $p$, we redirect the parent pointer of every node $q$ on the path from $p$ (including $p$ itself) to $r$.

Say that a pair $\langle p, q \rangle$ is bad iff either both $p \in F$ and $q \notin F$, or both $p \notin F$ and $q \in F$. The function `bad` is such that `bad(\langle p, q \rangle) = true` if $\langle p, q \rangle$ is bad, and `bad(\langle p, q \rangle) = false` otherwise.

For details of this implementation of partitions, see pages 33-36 of the notes.

Then, the algorithm is as follows:
function unif\([p,q,\Pi,dd]\): flag;
begin
trans := left(d); ff := right(d); pq := (p,q); st := (pq); flag := 1;
k := Length(first(trans));
while st \neq () \land flag \neq 0 do
uv := top(st); uu := left(uv); vv := right(uv);
pop(st);
if bad(ff,uv) = 1 then flag := 0
else
u := find(uu,\Pi); v := find(vv,\Pi);
if u \neq v then
union(u,v,\Pi);
for i = 1 to k do
u1 := delta(trans,uu,k - i + 1); v1 := delta(trans,vv,k - i + 1);
uv := (u1,v1); push(st,uv)
endfor
endif
endif
endwhile
end

The initial partition \(\Pi\) is the identity relation on \(Q\), i.e., it consists of blocks \(\{q\}\) for all state \(q \in Q\). The algorithm uses a stack \(st\). We are assuming that the DFA \(dd\) is specified by a list of two sublists, the first list, denoted \(left(dd)\) in the pseudo-code above, being a representation of the transition function, and the second one, denoted \(right(dd)\), the set of final states. The transition function itself is a list of lists, where the \(i\)-th list represents the \(i\)-th row of the transition table for \(dd\). The function \(delta\) is such that \(delta(trans,i,j)\) returns the \(j\)-th state in the \(i\)-th row of the transition table of \(dd\). For example, we have a DFA
\[
\begin{align*}
\text{dd} & = \left( ((2,3),(2,4),(2,3),(2,5),(2,3),(7,6),(7,8),(7,9),(7,6)), (5,9)) \right)
\end{align*}
\]
consisting of 9 states labeled 1,\ldots,9, and two final states 5 and 9. Also, the alphabet has two letters, since every row in the transition table consists of two entries. For example, the two transitions from state 3 are given by the pair \((2,3)\), which indicates that \(\delta(3,a) = 2\) and \(\delta(3,b) = 3\).

Implement the above algorithm, and test it at least for the above DFA \(dd\) and the pairs of states \((1,6)\) and \((1,7)\). Pay particular attention to the input and output format. In particular, output the current partition at every round through the \textbf{while} loop. Explain your data structures.
Please, consult the instructions posted on the web page for CIS262, Homework section, for instructions on the format for the input and output for this computer program.

**Problem B5 (25 pts).** Consider the language

\[ L = \{a^{4n+3} \mid 4n + 3 \text{ is prime}\}. \]

Assuming that \( L \) is infinite, prove that \( L \) is not regular.

**Extra Credit (25 pts).** Prove that there are infinitely many primes of the form \( 4n + 3 \).

The list of such primes begins with

\[ 3, 7, 11, 19, 23, 31, 43, \cdots \]

Say we already have \( n + 1 \) of these primes, denoted by

\[ 3, p_1, p_2, \cdots, p_n, \]

where \( p_i > 3 \). Consider the number

\[ m = 4p_1p_2\cdots p_n + 3. \]

If \( m = q_1 \cdots q_k \) is a prime factorization of \( m \), prove that \( q_j > 3 \) for \( j = 1, \ldots k \) and that no \( q_j \) is equal to any of the \( p_i \)'s. Prove that one of the \( q_j \)'s must be of the form \( 4n + 3 \), which shows that there is a prime of the form \( 4n + 3 \) greater than any of the previous primes of the same form.

**TOTAL: 225 points + 105 points Extra credit**