Problem B1 (60 pts). (1) Prove that the intersection, $L_1 \cap L_2$, of two regular languages, $L_1$ and $L_2$, is regular, using the Myhill-Nerode characterization of regular languages.

(2) Let $h: \Sigma^* \to \Delta^*$ be a homomorphism, as defined on pages 47-49 of the slides. For any regular language, $L' \subseteq \Delta^*$, prove that

$$h^{-1}(L') = \{w \in \Sigma^* | h(w) \in L'\}$$

is regular, using the Myhill-Nerode characterization of regular languages.

Proceed as follows: Let $\simeq'$ be a right-invariant equivalence relation on $\Delta^*$ of finite index $n$, such that $L'$ is the union of some of the equivalence classes of $\simeq'$. Let $\simeq$ be the relation on $\Sigma^*$ defined by

$$u \simeq v \iff h(u) \simeq' h(v).$$

Prove that $\simeq$ is a right-invariant equivalence relation of finite index $m$, with $m \leq n$, and that $h^{-1}(L')$ is the union of equivalence classes of $\simeq$.

To prove that that the index of $\simeq$ is at most the index of $\simeq'$, use $h$ to define a function $\hat{h}: (\Sigma^*/\simeq) \to (\Delta^*/\simeq')$ from the partition associated with $\simeq$ to the partition associated with $\simeq'$, and prove that $\hat{h}$ is injective.

Prove that the number of states of any minimal DFA for $h^{-1}(L')$ is at most the number of states of any minimal DFA for $L'$. Can it be strictly smaller? If so, give an explicit example.

Problem B2 (30 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,

$$p \equiv q \iff \forall z \in \Sigma^* (\delta^*(p, z) \in F \iff \delta^*(q, z) \in F),$$

and that $i$-equivalence is the equivalence relation $\equiv_i$ on $Q$, defined such that,

$$p \equiv_i q \iff \forall z \in \Sigma^*, |z| \leq i (\delta^*(p, z) \in F \iff \delta^*(q, z) \in F).$$
A relation $S \subseteq Q \times Q$ is a \textit{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 \textit{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 $R \approx$ containing $R$ is the relation $(R \cup R^{-1})^\ast$ (where $R^{-1} = \{(q, p) \mid (p, q) \in R\}$, and $(R \cup R^{-1})^\ast$ 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:

\[
R_0 = R \approx \\
R_{i+1} = (R_i \cup \{(\delta(p, a), \delta(q, a)) \mid (p, q) \in R_i, \ a \in \Sigma\})^\approx.
\]

The purpose of Problem B5(1) of HW2 was to prove that $R_{i_0+1} = R_{i_0}$ for some least $i_0$, and that $R_{i_0}$ is the smallest forward closure containing $R$. These facts need not be reproved.

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

(1) Prove that $p \equiv q$ iff the forward closure $R^\dagger$ of the relation $R = \{(p, q)\}$ is good.

\textit{Hint.} First, prove that if $R^\dagger$ is good, then

\[R^\dagger \subseteq \equiv .\]

For this, prove by induction that

\[R^\dagger \subseteq \equiv_i\]

for all $i \geq 0$.

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

\[R^\dagger \subseteq \equiv .\]

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

\textbf{Problem B3 (40 pts).} (1) Prove that the function, $f: \Sigma^* \rightarrow \Sigma^*$, given by

\[f(w) = w^R\]

is RAM computable. ($\Sigma = \{a_1, \ldots, a_N\}$).

(2) Prove that the function, $f: \Sigma^* \rightarrow \Sigma^*$, given by

\[f(w) = www\]

is RAM computable. ($\Sigma = \{a_1, \ldots, a_N\}$).
For simplicity, you may assume that $N = 2$.

You must run your interpreter from B5 on these two RAM programs for a few inputs. Show the two RAM programs as specified in the syntax of your interpreter in B5.

**Problem B5 (80 pts).** Write a computer program implementing a RAM program interpreter. You may want to assume that the instructions have five fields

\[
\begin{array}{cccccc}
N & X & \text{opcode} & j & Y \\
N & X & \text{opcode} & j & N_1
\end{array}
\]

with $j \in \{1, \ldots, k\}$, where $k$ is the number of symbols in $\Sigma$, and that the opcodes are

```
add  tail  clr  assign  gotoa  gotob  jmpa  jmpb  continue
```

where `gotoa` corresponds to jump above, `gotob` to jump below, `jpma` corresponds to jump above if condition is satisfied, and `jmpb` to to jump below if condition is satisfied. Depending on the opcode, some of the fields may be irrelevant (set them to 0).

The number of input registers is $n$ (so your memory must have at least $n$ registers), and the total number of registers is $p$. The number $k, n, p$ are input to your interpreter, as well as the program to be executed (a sequence of instructions). Assume that line numbers are integers. Also, to simplify matters, you may assume that you only consider alphabets of size at most 10, so that $a_1, \ldots, a_k$ ($k \leq 10$) are represented by the digits 0, 1, \ldots, 9.

Your program should output.

1. The input RAM program $P$
2. The input strings $w_1, \ldots, w_n$ to the RAM program $P$.
3. The value of the function being computed.
4. The sequence consisting of the memory contents and the current program counter as your interpreter executes the RAM program.

Test your interpreter on several RAM programs (and input strings), including the programs of B3.

To give you an idea of an implementation of this interpreter in Matlab here is the beginning of my program.

```
% % RAM interpreter
%
% opcodes are coded numerically as follows:
%`
```
% add = 1; tail = 2; clr = 3; assign = 4; gotoa = 5; gotob = 6; jmpa = 7;
% jpmb = 8; continue = 9
%
% Instructions have 5 fields
% N   X   opcode  j   Y
% N   X   opcode  j   N1
% where j corresponds to symbol a_j
% There are n input registers, a total number of p registers, and the
% alphabet size is k; symbols are coded as 1, 2, ..., k
% line numbers are nonnegative; unused line numbers are negative
% The registers are numbers 1, 2, ..., p
% The input RAM program is in RAMprog
% The program counter is pc
% input is a list indata containing the n input strings
%

function [res, pc, regs, counter] = RAMinterp(RAMprog, n, p, k, indata)
lenprog = size(RAMprog, 1);
%
% insert your code here
%
To run this program I used the following input file.

% % Running RAM interpreter
%
% concatenation of two strings

indata1{1} = [1 2 1 2]
indata1{2} = [2 1 2 1 1 2 2]

[res, pc, regs] = RAMinterp(RAMconcat,2,4,2,indata1)

% string reversal

indata2{1} = [1 2 2 2 2 1 1 2 1 2]
indata3{1} = [2 2 2 2 2 1 1 1 1 1]

[res, pc, regs] = RAMinterp(RAMrev,1,4,2,indata2)

% f(w) = www
Here is the program to concatenate two strings.

```matlab
% a RAM program to concatenate two strings

RAMconcat =
[-1 3 4 0 1;
 -1 4 4 0 2;
 0 4 8 1 1;
 -1 4 8 2 2;
 -1 0 6 0 3;
 1 0 1 1 3;
 -1 0 2 0 4;
 -1 0 5 0 0;
 2 0 1 2 3;
 -1 0 2 0 4;
 -1 0 5 0 0;
 3 1 4 0 3;
 -1 0 9 0 0]
```

TOTAL: 210 points.