You have one hour and 15 minutes to complete the exam.
Do not open the exam until instructed to do so.
No notes, texts, computers, calculators, or communication devices are permitted.
Write all answers on the examination paper itself.

**BUDGET YOUR TIME WELL!**

**SHOW ALL WORK!**

<table>
<thead>
<tr>
<th>Question 1</th>
<th>[13]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 2</td>
<td>[10]</td>
</tr>
<tr>
<td>Question 3</td>
<td>[8]</td>
</tr>
<tr>
<td>Question 4</td>
<td>[9]</td>
</tr>
<tr>
<td>Question 5</td>
<td>[10]</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>[50]</td>
</tr>
</tbody>
</table>

Bonus question [1 mark]: What is the language $\emptyset^*\lambda$? It is $\{\lambda\}$ (Note that $\emptyset^* = \lambda$.)

Question 1. [13 marks total]
Let G be the grammar with start variable S, terminals \{0,1\}, and productions:
\[
\begin{align*}
    S & \rightarrow 0A \mid 0M \\
    A & \rightarrow 0A \mid 1B \mid \lambda \\
    B & \rightarrow 0C \\
    C & \rightarrow 0A \\
    M & \rightarrow 1N \\
    N & \rightarrow 1P \\
    P & \rightarrow 1P \mid 0M \mid \lambda
\end{align*}
\]
Call its language L.

(a) [3 marks] Using grammar G and the algorithms developed in class, give an NFA for L.
(Describe briefly the steps that you follow.)
I make every variable a state, and make the start state be the start variable and the final states be every variable X for which \( X \rightarrow \lambda \). Every production \( X \rightarrow aY \) makes a transition from X to Y labeled \( a \).

So I get an NFA with states \( Q = \{S, A, B, C, M, N, P\} \), start state \( S \), final states \( \{A, P\} \), \( \Sigma = \{0, 1\} \), and transition function \( \delta \) defined by \( \delta(S, 0) = \{A, M\} \), \( \delta(A, 0) = \{A\} \), \( \delta(A, 1) = \{B\} \), \( \delta(B, 0) = \{C\} \), \( \delta(C, 0) = \{A\} \), \( \delta(M, 1) = \{N\} \), \( \delta(N, 1) = \{P\} \), \( \delta(P, 0) = \{M\} \), \( \delta(P, 1) = \{P\} \). When \( \delta(X, a) \) is not defined here, it is \( \emptyset \).

(It is probably easier to draw a transition graph, and most people did.)

(b) [5 marks] Using your answer to (a) and the algorithms developed in class, give a DFA for \( L \). (Describe briefly the steps that you follow.)

My DFA has states corresponding to sets of states of the NFA from the (a) part. For each set of states that I encounter and each symbol in \( \Sigma \), I make a transition to the set of all states that the NFA can get to by reading the specified symbol. I start with the state \( \{S\} \). I determine whether a state is final according to whether its set contains a final state of the NFA.

<table>
<thead>
<tr>
<th>State of DFA</th>
<th>Transition on 0 to state</th>
<th>Transition on 1 to state</th>
</tr>
</thead>
<tbody>
<tr>
<td>{S}</td>
<td>{A,M}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{A,M} – final!</td>
<td>{A}</td>
<td>{B,N}</td>
</tr>
<tr>
<td>{A} – final!</td>
<td>{A}</td>
<td>{B}</td>
</tr>
<tr>
<td>{B,N}</td>
<td>{C}</td>
<td>{P}</td>
</tr>
<tr>
<td>{B}</td>
<td>{C}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{C}</td>
<td>{A}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{P} – final!</td>
<td>{M}</td>
<td>{P}</td>
</tr>
<tr>
<td>{M}</td>
<td>\emptyset</td>
<td>{N}</td>
</tr>
<tr>
<td>{N}</td>
<td>\emptyset</td>
<td>{P}</td>
</tr>
</tbody>
</table>

(c) [5 marks] Using your answer to (a) and the algorithms developed in class, give a regular expression for \( L \). (Describe briefly the steps that you follow.)

First I modify the NFA from (a) by giving it a single final state \( F \). To do this, I add transitions \( \delta(A, \lambda) = \{F\} \) and \( \delta(P, \lambda) = \{F\} \); make \( F \) a final state; and make A and P non-final. Now I repeatedly eliminate a non-start, non-final state by figuring out all the ways to enter and leave it and making a transition labeled with the corresponding regular expression.

To start, eliminate \( B \) and the two transitions involving it, and add the transition \( \delta(A, 10) = \{C\} \).
Then eliminate \( C \), and replace the transition \( \delta(A, 0) = \{A\} \) by \( \delta(A, 0+100) = \{A\} \).
Then eliminate \( N \) and the two transitions involving it, and add the transition \( \delta(M, 11) = \{P\} \).
Then eliminate \( M \) and the three transitions involving it, add \( \delta(S, 011) = \{P\} \), and replace \( \delta(P, 1) = \{P\} \) by \( \delta(P, 1+011) = \{P\} \).
Finally eliminate \( A \) and \( P \) to get a two-state NFA with \( \delta(S, 0(0+100)^* + 011(1+011)^*) = \{F\} \).
Now read out the regular expression, \( 0(0+100)^* + 011(1+011)^* \).

There are many reduction orders, and the answer can look different.
Question 2. [10 marks]

(a) [3 marks] Show that if \( L \) is a regular language, then the reverse of \( L \), \( L^R \), is also a regular language.

Many solutions are possible. Here is one. Because \( L \) is regular, \( L \) has an NFA. By adding \( \lambda \)-transitions, we can make an NFA that has exactly one final state. Now reverse all of the transitions, make the start state final and the final state the start state. This new NFA accepts the reverse of \( L \), so it is regular.

(b) [5 marks] Suppose that for a language \( L \), \( \text{prefix}(L) = \{w : wy \in L\} \). Show that if \( L \) is regular, then \( \text{prefix}(L) \) is also regular.

Many solutions are possible. One way uses DFAs, one way uses regular expressions. I will show the second. Because \( L \) is regular, it has a regular expression \( r \).

\[
\begin{align*}
&\text{If } r = \emptyset, \text{prefix}(r) = \emptyset. \\
&\text{If } r = \lambda, \text{prefix}(r) = \lambda. \\
&\text{If } r = a, \text{prefix}(r) = a + \lambda. \\
&\text{If } r = r_1 + r_2, \text{prefix}(r) = \text{prefix}(r_1) + \text{prefix}(r_2). \\
&\text{If } r = r_1r_2, \text{prefix}(r) = r_1 \text{prefix}(r_2) + \text{prefix}(r_1). \\
&\text{If } r = (r_1)^*, \text{prefix}(r) = (r_1)^{*} \text{prefix}(r_1).
\end{align*}
\]

Because \( \text{prefix}(r) \) has a regular expression, it represents a regular language.

Note: this does NOT follow from closure under concatenation.

(c) [2 marks] Use your answers to the (a) and (b) parts to show that if \( L \) if regular then \( \text{suffix}(L) = \{w : yw \in L\} \)is also regular.

\[
\text{suffix}(L) = \{w^R : w \in \text{prefix}(L^R)\} = (\text{prefix}(L^R))^R.
\]

Use closure under reversal and prefix to get closure under suffix.

Question 3 [8 marks]

(a) [2 marks] Define each of the following types of grammar: left linear; right linear; regular; linear.

- **left-linear**: all productions of the form \( A \rightarrow Bw \) or \( A \rightarrow w \) with \( A, B \in V, w \in T^* \).
- **right-linear**: all productions of the form \( A \rightarrow wB \) or \( A \rightarrow w \) with \( A, B \in V, w \in T^* \).
- **regular**: the grammar is either right-linear or left-linear.
- **linear**: all productions of the form \( A \rightarrow wB, A \rightarrow Bw, \) or \( A \rightarrow w \) with \( A, B \in V, w \in T^* \).

(b) [2 marks] If \( L \) has a regular grammar, is \( L \) (always) a regular language? Explain.

Yes. If the grammar is right-linear we can convert it to an NFA (as in Question 1(a)). If it is left-linear, reversing each right hand side gives a right-linear grammar for the reverse of the language, and we have closure under reversal, so it is again regular.

(c) [4 marks] If \( L \) has a linear grammar, is \( L \) (always) a regular language? Explain.

No. Here is a linear grammar: \( S \rightarrow aB \mid \lambda, B \rightarrow Sb \). Its language is \( \{a^n b^n : n \geq 0\} \), which we know is not regular.
It is not enough to just say that one cannot make an NFA or DFA.

Question 4 [9 marks]
(a) [2 marks] Build a DFA to accept the language of strings over \{a,b\} that start and end with an a.

I allow the case that there is only one letter, and then the first and the last letters are the same.

<table>
<thead>
<tr>
<th>State</th>
<th>Transition on input a to</th>
<th>Transition on input b to</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$ – start state</td>
<td>$q_1$</td>
<td>$q_{dead}$</td>
</tr>
<tr>
<td>$q_{dead}$</td>
<td>$q_{dead}$</td>
<td>$q_{dead}$</td>
</tr>
<tr>
<td>$q_1$ – final state</td>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

(b) [7 marks] Let $L$ be the language of strings over \{a,b\} that start and end with an a and contain at least as many a’s as b’s. Is $L$ regular? Answer yes or no, and show that your answer is correct.

No! The easiest is to use pumping, but let me use a different argument. Suppose to the contrary that there is a DFA for $L$ with start state $q_0$ and transition function $\delta$. Let’s define an infinite sequence of states by $q_{i+1} = \delta(q_i, a)$ for all $i \geq 0$. By the pigeonhole principle, there must be an $i$ and $j$ with $i < j$ for which $q_i = q_j$.

Because $ab^{j+1}a$ is in $L$, it follows that $\delta^*(q_j, b^{j+1}a)$ is a final state.

But $q_i = q_j$ so $\delta^*(q_i, b^{j+1}a)$ is a final state, and hence $ab^{j+1}a$ is in $L$.

This is a contradiction because $i < j$.

(Arguments that just say “not enough memory”, while true, are not convincing at all. Arguments that presume that the numbers of a’s and b’s are the same are not correct.)
Question 5 [10 marks]
(a) [5 marks] Outline a general method for converting a regular expression with language L into a regular expression whose language is the complement of L.

1. Convert the regular expression to an NFA.
2. Convert the NFA to a DFA.
3. Swap final and nonfinal states to get a DFA M for the complement of L. (Note: you need this step – you cannot do this on an NFA!)
4. Change the DFA so that it has only one final state – this may make it an NFA M’.
5. Eliminate non-start non-final states one at a time in M’ (as in 1(c)) to recover an r.e. for the complement.

(b) [5 marks] Apply your method to the regular expression \((ab + ba)^*\) to produce a regular expression for its complement.

First I make an NFA – and simplify it – and convert to a DFA to get

<table>
<thead>
<tr>
<th>State</th>
<th>Transition on input a to</th>
<th>Transition on input b to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0) – start state and final state</td>
<td>(q_1)</td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>(q_2)</td>
<td>(q_0)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(q_0)</td>
<td>(q_3)</td>
</tr>
<tr>
<td>(q_3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Then I swap final and nonfinal states and add a new single final state to get the NFA:

<table>
<thead>
<tr>
<th>State</th>
<th>Transition on a to</th>
<th>Transition on b to</th>
<th>Transition on (\lambda) to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0) – start state</td>
<td>(q_1)</td>
<td>(q_2)</td>
<td></td>
</tr>
<tr>
<td>(q_1)</td>
<td>(q_3)</td>
<td>(q_0)</td>
<td>(q_f)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(q_0)</td>
<td>(q_3)</td>
<td>(q_f)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>(q_3)</td>
<td>(q_3)</td>
<td>(q_f)</td>
</tr>
<tr>
<td>(q_f) – final state</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

And now I eliminate the three states that are non-start, non-final, one at a time, to get:

\((ab+ba)^*(a + b + (aa+bb)(a+b))^*\)

Again, the specific NFA that you start with, and the reduction sequence, can make the answer look different.

Check for yourself: If you do not convert the NFA to a DFA before swapping final and non-final states, you may not get the complement of the language!