Theory of Computation
Sample Final; solution set.

1. (**10 points.**) Give a regular expression for the following language A over the alphabet $\{a, b\}$.

$$A = \{w \mid w \text{ does not contain the substring } aaa\}.$$

Solution.
$(\lambda \cup a \cup aa)\{b \cup ba \cup baa\}^*$.

2. (**10 points.**) Either give a context free grammar to generate the following language B, or give a PDA that recognizes B.

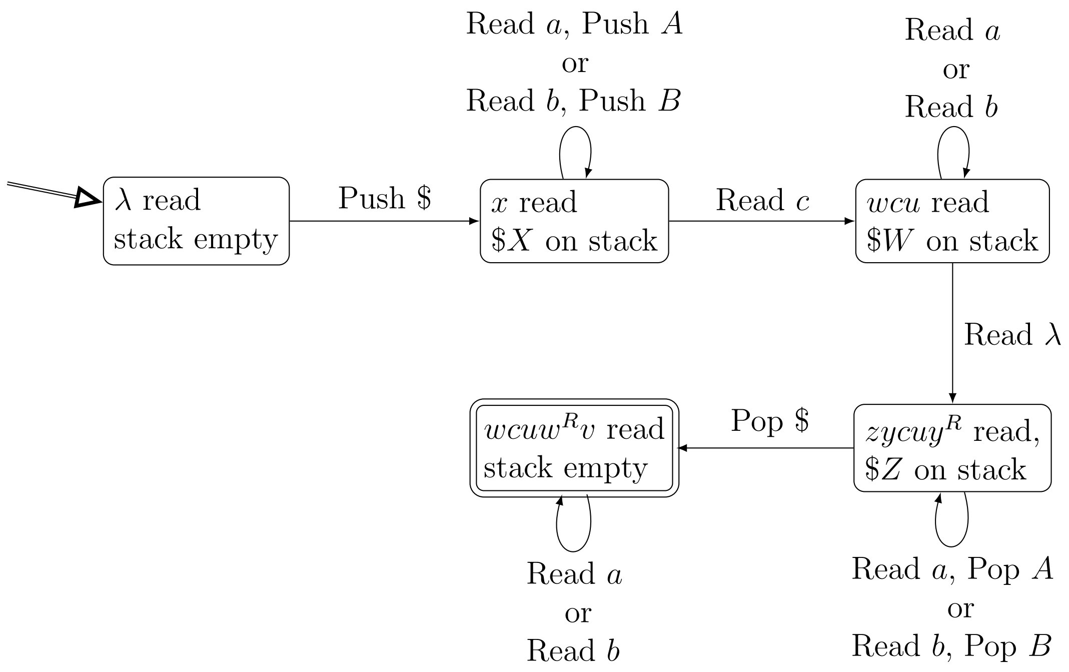
$$B = \{wcuw^R v \mid w, u, v \in \{a, b\}^*\}.$$

If you give a CFG, you should include a specification of the set of strings that each variable generates.
If you give a PDA either a clear English description or a PDA with vertex specifications suffices.

CFG solution.
X will generate $\{a, b\}^*$; P will generate the string $wcuw^R$ or equivalently $wcXw^R$.

$$\begin{aligned} S &\leftarrow PX \\ P &\leftarrow aPa \mid bPb \mid cX \\ X &\leftarrow a \mid b \mid XX \mid \lambda \end{aligned}$$

PDA solution.
The PDA will use a \$-shielded stack. It will begin by copying w onto the stack as it reads w, ending this process when the first (and only) c is reached; it will then read the c and a subsequent arbitrary string of a's and b's (string u), whose end is determined non-deterministically. This is followed by the reading of w^R, which is confirmed by matching the read characters with the characters on the stack, which are popped one by one in tandem with the reading of w^R. This process completes when the stack is emptied back down to the \$-shield. Finally, another arbitrary string of a's and b's (string v) is read. If the end of the string is reached at this point, the input is recognized. The PDA is illustrated in the following figure.



λ read stack empty
Push \$
x read \$$X$ on stack
Read a, Push A or Read b, Push B
Read c
wcu read \$$W$ on stack
Read a or Read b
Read λ
$zycuy^R$ read, \$$Z$ on stack
Read a, Pop A or Read b, Pop B
Pop \$
$wcuw^Rv$ read stack empty
Read a or Read b

3. (**10 points.**) Show that the following language C is not context free.

$$C = \{a^i ba^j ba^k ba^l \mid i = k \text{ and } j = l\}.$$

Solution.

Suppose, for a contradiction, that C were context free. Then by the Pumping Lemma, C has a pumping constant $p \geq 1$, such that all strings $s \in C$ with $|S| \geq p$ can be pumped. Choose $s = a^p ba^p ba^p ba^p$. As $|s| \geq p$ and $s \in C$, by the Pumping Lemma we can write s as $uvwxy$ with $|vwx| \leq p$, $|vx| \geq 1$, and $uv^i wx^i y \in C$ for all integer $i \geq 0$. Consider the string $s' = uwy$. By the Pumping Lemma, $s' \in C$. If vx lies in a single block of a's, then s' has one if its four blocks of a having $|vx|$ fewer a's, and then $s' \notin C$ as $|vx| \geq 1$, which is a contradiction. If vx contains a b, then s' contains only two b's, and again $s' \notin C$, which is a contradiction. Finally, if vx overlaps two blocks of a's, which are necessarily adjacent blocks as $|vwx| \leq p$, then in s' those two blocks each have fewer a's than the other two blocks; as a result, blocks 1 and 3 have different numbers of a's, as do blocks 2 and 4. Once more $s' \notin C$, which is once more a contradiction. As there is a contradiction in every case, the initial assumption must be incorrect; consequently, C is not context free.

4. (**10 points.**) Suppose that you are given an algorithm $\mathcal{A}_D$ to decide the following language D:

$$D = \{\langle Q\rangle \mid Q \text{ halts on at least one input}\}.$$

Using $\mathcal{A}_D$ as a subroutine, give an algorithm $\mathcal{A}_H$ to decide $H = \{\langle P, w\rangle \mid P \text{ halts on input } w\}$.

Solution.

The program A_H has the following form.

On input $\langle P, w\rangle$:

```
    Step 1. Construct a Program $Q_{P,w}$.
    Step 2. Run $\mathcal{A}_D(\langle Q_{P,w}\rangle)$.
    Step 3. Report the result from Step 2.
```

In order for this to determine whether P halts on input w, we want:

$$\mathcal{A}_D(\langle Q_{P,w}\rangle) = \mathcal{A}_H(\langle P, w\rangle) = \begin{cases} \text{“Recognize”} & \text{if } P(\langle w\rangle) \text{ halts} \\ \text{“Reject”} & \text{if } P(\langle w\rangle) \text{ runs forever} \end{cases}$$

In other words, we need:

$$\begin{array}{lcl} Q_{P,w} \text{ halts on some input} & \Longleftrightarrow & P(\langle w\rangle) \text{ halts} \\ Q_{P,w} \text{ runs forever on every input} & \Longleftrightarrow & P(\langle w\rangle) \text{ runs forever} \end{array}$$

Accordingly, we define $Q_{P,w}$ as follows:

```
On input $x$:
    Run $P(\langle w\rangle)$.
```

Clearly, on any input x, $Q_{P,w}(x)$ halts exactly if $P(\langle w\rangle)$ halts, and $Q_{P,w}(x)$ runs forever on every input x if $P(\langle w\rangle)$ runs forever.

5. (**10 points.**) Show that the following Must Traverse Edges problem (MTE for short) has a polynomial time verifier.
Input: A directed graph $H = (W, F)$ and a subset $I \subseteq F$.
Question: Is there a simple cycle in H which includes every edge in I, and possibly others? (A simple cycle is one with no repeated vertices.)

Solution.

The a candidate certificate consists of a sequence of vertices; the certificate is valid if all the vertices are in W, and in their given order, they form a simple cycle in H which includes every edge in I.

A valid certificate has length $O(W)$ which is linear in the size of the input. Checking whether a candidate certificate is correct is straightforward. We need to verify (i) that it is a sequence of vertices in W in which each vertex in W occurs at most once, (ii) that successive vertices in the sequence are joined by edges, including an edge from the last vertex back to the first vertex, (iii) that every edge in I is in this sequence. (Strictly speaking, we should also check that the input meets its specifications.) Each check is readily implemented in time $O((I + W) \cdot F)$ (more efficient implementations are possible). This is polynomial in the input size, as desired.

Finally, we note that if $(H, I) \in$ MTE, then the cycle demonstrating this property provides a certificate, and if there is a certificate it provides the cycle demonstrating the property.

6. (**10 points.**) Suppose you are given a polynomial time algorithm to decide MTE. Then show that there is also a polynomial time algorithm for DHC, where DHC denotes the Directed Hamiltonian Cycle problem.
Hint. Given an input $G = (V, E)$ that is an input to the DHC problem, you need to show how to construct a suitable pair (H, I). Note that if there is a Hamiltonian Cycle in G, that is a cycle that traverses every $\underline{\text{vertex}}$, then there needs to be a cycle in H that traverses at least the specified subset of the $\underline{\text{edges}}$ in H.
Solution.

Our algorithm to test whether G has a Hamiltonian Cycle begins by building a pair (H, I) such that $(H, I) \in$ MTE if and only if $G \in$ DHC. It then runs the algorithm for MTE on input (H, I) and reports this answer.

(H, I) is constructed as follows. For each vertex $v \in V$, vertices v^{in} and v^{out} are added to W, and edge $(v^{\text{in}}, v^{\text{out}})$ is added to both F and I. In addition, for each edge $(u, v) \in E$, edge $(u^{\text{out}}, v^{\text{in}})$ is added to F. Clearly, this construction takes time linear in the size of G.

Now we show that $(H, I) \in$ MTE if and only if $G \in$ DHC.

First, suppose that G has a Hamiltonian Cycle, $v_1, v_2, \ldots, v_n$ (we are letting $n = |V|$). Then the following cycle in H goes through every edge in I: $v_1^{\text{in}}, v_1^{\text{out}}, v_2^{\text{in}}, v_2^{\text{out}}, \ldots, v_n^{\text{in}} v_n^{\text{out}}$.

Next, suppose that H has a cycle that goes through every edge in I. As there are $|V| = n$ edges in I, and as they have no endpoints in common, this cycle must use every vertex in W. Consequently the cycle in H must be of the form $v_1^{\text{in}}, v_1^{\text{out}}, v_2^{\text{in}}, v_2^{\text{out}}, \ldots, v_n^{\text{in}} v_n^{\text{out}}$. Each edge $(v_i^{\text{out}}, v_{i+1}^{\text{in}})$ is present in H because (v_i, v_{i+1}) is an edge in E. Therefore $v_1, v_2, \ldots, v_n$ must be a Hamiltonian Cycle in G.

This shows that $(H, I) \in$ MTE if and only if $G \in$ DHC.

7. (**10 points.**) Let Σ be an alphabet. We define $\mathrm{DF}(w)$ to be the function that returns the string w with its first character removed; i.e. if $w = av$ for some $a \in \Sigma$ and $v \in \Sigma^*$, then $\mathrm{DF}(w) = v$.

Let L be a language. Then $\mathrm{DF}(L) = \cup_{w \in L \setminus \{\lambda\}} \mathrm{DF}(w)$.

Show that if L is a context free language then so is $\mathrm{DF}(L)$. Your construction may be based on a CFL G that generates L or a PDA M that recognizes L.

For your correctness argument the following suffices. For a CFL construction: a specification of the mapping between corresponding derivation trees for strings in L and in $L(\widetilde{G})$, where $\widetilde{G}$ is the new grammar. For a PDA construction: a specification of the mapping between corresponding recognizing paths for strings in L and in $L(\widetilde{M})$, where $\widetilde{M}$ is the new PDA.

CFL solution.
WLOG Suppose that G is a CNF grammar (for if not, given G we can construct a CNF grammar G' generating L). We now build a new context free grammar $\widetilde{G}$ which generates $\mathrm{DF}(L)$. For each variable X in G, $\widetilde{G}$ will have variables X^l and X^n. The variables X^l will occur only on the leftmost path of any derivation tree in the new grammar, while the variables X^n will occur at all other locations of each derivation tree. To this end, we define the rules in $\widetilde{G}$ as follows.

- If $A \to BC$ is a rule in G, then $A^l \to B^l C^n$ and $A^n \to B^n C^n$ are rules in $\widetilde{G}$.
- If $A \to b$ is a rule in G, then $A^l \to \lambda$ and $A^n \to b$ are rules in $\widetilde{G}$.
- Finally, the start variable in $\widetilde{G}$ is S^l.

Next, we explain the relationship between a derivation tree D in G for a string w, and a derivation tree $\widetilde{D}$ in $\widetilde{G}$ for a string v, where $v = \mathrm{DF}(w)$.

D and $\widetilde{D}$ have the same nodes and edges. When a non-leaf node in D has label X, the corresponding node in $\widetilde{D}$ has label X^l if it is on the leftmost path in $\widetilde{D}$, and otherwise has label X^n. The leaf labels are the same except for the leftmost leaf. In $\widetilde{D}$ the label is λ, while in D the label is some $b \in \Sigma$, with the additional constraint that if X labels the parent of the leftmost leaf in D, then $X \to b$ is a rule in G.

PDA solution.
We build a PDA M' to recognize $\text{DF}(L)$ as follows. M' consists of two copies of M, M^1 and M^2. The idea is that if M' is recognizing a string $\text{DF}(w)$, where $w = av$, then so long as the computation of w has not read anything, the corresponding computation of M' remains in M_1, but as soon as M reads the character a, M' uses a corresponding edge to switch to M_2. This edge has the Read a portion of its label changed to Read λ.

Now, we describe the construction precisely. The start vertex for M' is the copy of M's start vertex in M_1. The recognizing vertices are the copies of M's recognizing vertices in M_2. M_1 keeps only those edges whose labels include a Read λ. However, M_2 has copies of all the edges in M_1. In addition, if (p, q) is a edge in M whose label includes a Read a for some $a \in \Sigma$, then we add an edge (p_1, q_2) between the corresponding vertices in M_1 and M_2, resp.; furthermore, the label of this edge has the Read a changed to Read λ.

Next, we state the relationship between a recognizing path in M for a string w and a recognizing path in M' for a string v, where $v = \text{DF}(w)$.

If either $w \in L$ or $v \in L(M')$ (in fact, if one occurs so does the other), the following holds: there are recognizing paths P in M for w, and P' in M' for string v, which are related as follows.

We write P as P_1, e, P_2, where $e = (p, q)$ is the first edge on which a character is read. Similarly, we write P' as P'_1, e', P'_2, where $e' = (p'_1, q'_2)$ is the edge from M_1 to M_2 (there must be such an edge as the start vertex lies in M_1 but all recognizing vertices are in M_2). Then P_1 and P'_1 are identical paths, as are P_2 and P'_2; in addition $p = p'_1$ and $q = q'_2$.