We strongly recommend that you work through this exam under realistic conditions rather than just flipping through the problems and seeing what they look like. Setting aside three hours in a quiet space with your notes and making a good honest effort to solve all the problems is one of the single best things you can do to prepare for this exam. It will give you practice working under time pressure and give you an honest sense of where you stand and what you need to get some more practice with.

This practice final exam is a (slightly modified) version of the final exam we gave out in Fall 2018. The exam policies are the same for the midterms – closed-book, closed-computer, limited note (one double-sided sheet of 8.5” × 11” paper decorated however you’d like).

You have three hours to complete this exam. There are 63 total points.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Graders</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Discrete Structures I</td>
<td></td>
<td>/ 13</td>
</tr>
<tr>
<td>(2) Discrete Structures II</td>
<td></td>
<td>/ 16</td>
</tr>
<tr>
<td>(3) Formal Languages I</td>
<td></td>
<td>/ 10</td>
</tr>
<tr>
<td>(4) Formal Languages II</td>
<td></td>
<td>/ 13</td>
</tr>
<tr>
<td>(5) Finite Automata</td>
<td></td>
<td>/ 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/ 63</td>
</tr>
</tbody>
</table>
Problem One: Discrete Structures I  
(13 Points)  
(We recommend spending about 30 minutes on this problem.)

On Problem Sets One and Two, you explored tournaments and their properties. This problem explores other concepts in tournaments and how they relate to the ones you’ve seen so far.

Let’s begin by refreshing some definitions. A tournament is a contest among \( n \) players. Each player plays a game against each other player, and either wins or loses the game (let’s assume that there are no draws). No one plays a game against herself. A sample tournament is shown below:

```
A     B
|     |
\|     \|
C     D
```

Suppose \( T \) is a tournament involving the set of players \( P \). We say that a factor of \( T \) is a set \( F \subseteq P \) where
- \( F \neq \emptyset \), and
- for any player \( p \in F \) and any player \( q \in P - F \), player \( p \) won her game against player \( q \).

This question is all about factors and their properties.

i. (3 Points) Let \( T \) be an arbitrary tournament involving the set of players \( P \). Prove that if \( P \) is nonempty, then \( T \) has at least one factor.
(Extra space for your answer to Problem One, Part (i), if you need it.)
Refreshing the definition from the previous page, suppose $T$ is a tournament involving the set of players $P$. We say that a factor of $T$ is a set $F \subseteq P$ where

- $F \neq \emptyset$, and
- for any player $p \in F$ and any player $q \in P - F$, player $p$ won her game against player $q$.

Tournaments can have multiple factors, and if they do, those factors will be closely connected.

ii. **(4 Points)** Let $T$ be a tournament involving the set of players $P$. Prove that if both $F_1$ and $F_2$ are factors of $T$, then $F_1 \subseteq F_2$ or $F_2 \subseteq F_1$ (or both).
(Extra space for your answer to Problem One, Part (ii), if you need it.)
Recapping the definition from the previous page, if $T$ is a tournament involving the set of players $P$, then a \textit{factor of $T$} is a set $F \subseteq P$ where

- $F \neq \emptyset$, and
- for any player $p \in F$ and any player $q \in P - F$, player $p$ won her game against player $q$.

As a refresher from the problem set, a \textit{tournament champion} is a player $c$ where, for each other player $p$ in the tournament, either

- $c$ won her game against $p$, or
- there’s a player $q$ where $c$ won her game against $q$ and $q$ won his game against $p$.

Champions interact with factors in interesting ways.

iii. \textbf{(6 Points)} Let $T$ be a tournament with set of players $P$. Prove that if $c$ is a champion in $T$, then $c$ is an element of every factor of $T$. 
(Extra space for your answer to Problem One, Part (iii), if you need it.)
Problem Two: Discrete Structures II

(We recommend spending about 50 minutes on this problem.)

On Problem Sets Three, Four, and Five, you explored graph automorphisms and their properties. This question explores a related concept.

Let’s begin with a new definition. Suppose that $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are graphs. We’ll say that a **homomorphism from $G_1$ to $G_2$** is a function $h : V_1 \rightarrow V_2$ with the following property:

$$\forall u \in V_1. \forall v \in V_1. \{\{u, v\} \in E_1 \rightarrow \{h(u), h(v)\} \in E_2 \}.$$

This is similar to how we defined graph automorphisms, but has some key differences:

- An automorphism is a function from a graph’s nodes to themselves. Graph homomorphisms, on the other hand, can map from one graph to another.
- An automorphism is, by definition, a bijection. The definition of a homomorphism does not require the function to be a bijection.
- An automorphism preserves both adjacency and non-adjacency. A homomorphism only has to preserve adjacency.

This question explores properties of homomorphisms and connections between homomorphisms and other concepts you’ve seen this quarter.

i.  (1 Point) Below are pictures of two graphs $G_1$ and $G_2$. Find a homomorphism from $G_1$ to $G_2$. To give your answer, label each node of $G_1$ with the corresponding node in $G_2$ that the homomorphism maps it to.

\[\text{The graph } G_1\]

\[\text{The graph } G_2\]
As a refresher from the previous page, if $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are graphs, a **homomorphism from $G_1$ to $G_2$** is a function $h : V_1 \to V_2$ with the following property:

$$\forall u \in V_1. \forall v \in V_1. (\{u, v\} \in E_1 \to \{h(u), h(v)\} \in E_2).$$

Now, a new definition. A graph $G = (V, E)$ is called a **complete graph** if the following is true:

$$\forall u \in V. \forall v \in V. (u \neq v \leftrightarrow \{u, v\} \in E).$$

This question explores how complete graphs interact with homomorphisms.

ii. **(8 Points)** Let $G = (V, E)$ be a complete graph and let $h : V \to V$ be an arbitrary function. Prove that $h$ is injective if and only if it's a homomorphism from $G$ to itself.

Just to make sure you didn't miss it, the statement to prove is a biconditional.
(Extra space for your answer to Problem Two, Part (ii), if you need it.)
If \( G_1 = (V_1, E_1) \) and \( G_2 = (V_2, E_2) \) are graphs, a **homomorphism from \( G_1 \) to \( G_2 \)** is a function \( h : V_1 \rightarrow V_2 \) with the following property:

\[
\forall u \in V_1. \forall v \in V_1. (\{u, v\} \in E_1 \rightarrow \{h(u), h(v)\} \in E_2).
\]

On Problem Set Four, you explored bipartite graphs. As you saw, an undirected graph \( G = (V, E) \) is **bipartite** if there exist two sets \( V_1 \) and \( V_2 \) such that

- every node \( v \in V \) belongs to exactly one of \( V_1 \) and \( V_2 \), and
- every edge \( e \in E \) has one endpoint in \( V_1 \) and the other in \( V_2 \).

This question explores the connection between bipartite graphs and graph homomorphisms.

iii. **(7 Points)** Let \( G = (V, E) \) be an arbitrary graph, and let \( H \) be the graph shown below over the set of nodes \( W = \{ \star, ☺ \} \). Prove that if there is a homomorphism from \( G \) to \( H \), then \( G \) is bipartite.

\[
\begin{align*}
\star & \quad \cdots \quad ☺
\end{align*}
\]

*The graph \( H \).*
(Extra space for your answer to Problem Two, Part (iii), if you need it.)
Problem Three: Formal Languages I

(We recommend spending 30 minutes on this problem.)

Let’s begin with a new definition. A language \( L \) over an alphabet \( \Sigma \) is called **forgiving** if
\[
\forall w \in \Sigma^*, \exists x \in \Sigma^*. wx \in L.
\]
A language that is not forgiving is given the ominous title of **unforgiving**.

i. **(4 Points)** Below are a collection of languages described in various ways. For each language, decide whether that language is forgiving. No justification is necessary.

- The language of the above DFA over the alphabet \( \Sigma = \{a, b\} \).
  - ☐ Forgiving
  - ☐ Unforgiving

- The language of the above NFA over the alphabet \( \Sigma = \{a, b\} \).
  - ☐ Forgiving
  - ☐ Unforgiving

- \((a*b \cup a?c?)^*\)
  - The language of the above regex over the alphabet \( \Sigma = \{a, b, c\} \).
    - ☐ Forgiving
    - ☐ Unforgiving

- \(S \rightarrow aSc \mid bSc \mid cSa \mid \varepsilon\)
  - The language of the above CFG over the alphabet \( \Sigma = \{a, b, c\} \).
    - ☐ Forgiving
    - ☐ Unforgiving
As a refresher from the previous page, a language $L$ over $\Sigma$ is called **forgiving** if the following is true:

$$\forall w \in \Sigma^*. \exists x \in \Sigma^*. \, wx \in L.$$ 

The forgiving languages have a number of nice properties.

ii. **(6 Points)** Prove that the forgiving languages are closed under concatenation. That is, prove that if $L_1$ and $L_2$ are forgiving languages over some alphabet $\Sigma$, then $L_1L_2$ is also forgiving. As a refresher, language concatenation is defined as follows:

$$L_1L_2 = \{ \, wx \mid w \in L_1 \land x \in L_2 \, \}.$$
(Extra space for your answer to Problem Three, Part (ii), if you need it.)
Problem Four: Formal Languages II

(13 Points)

(We recommend spending about 30 minutes on this problem.)

Let $\Sigma = \{1, +, \equiv\}$ and consider this language $L_1$:

$$L_1 = \{1^m+1^n\equiv1^p \mid m, n, p \in \mathbb{N} \text{ and } (m + n) \equiv_2 p \}.$$ 

For example, these strings are in $L_1$:

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$11 + 11 \equiv 1111$</td>
<td>$1 + 11 \equiv 111 \equiv 1$</td>
</tr>
<tr>
<td>$11 + 11\equiv 11$</td>
<td>$1 + 11\equiv 1$</td>
</tr>
<tr>
<td>$\equiv$</td>
<td>$11 + 1\equiv 1111$</td>
</tr>
<tr>
<td>$1111 + \equiv$</td>
<td>$111 + 1\equiv 1111$</td>
</tr>
</tbody>
</table>

However, these strings are not in $L_1$:

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 + 1 \equiv 1$</td>
<td>$1 + \equiv$</td>
</tr>
<tr>
<td>$11 + 11 \equiv 111$</td>
<td>$11 \equiv 1 + 1$</td>
</tr>
<tr>
<td>$111 + 1 \equiv 111$</td>
<td>$\equiv + 1$</td>
</tr>
</tbody>
</table>

This language turns out to be regular.

i. (3 Points) In the space below, design an NFA for $L_1$. Then, write a brief description (at most fifty words) of how your NFA works.

As a hint, you may want to proceed as though you were designing a DFA for $L_1$, but simply leave out transitions corresponding to syntactically invalid strings.

Brief (at most fifty-word) description of how this NFA works:
Consider the following directed graph $G$:

![Directed Graph]

We can think of a path in this graph as a string over $\Sigma = \{ b, c, h, i, o, p, v \}$. For example, we could represent the path $b, i, h, p$ as the string $bihp$ and the path $v, o, i, c, b, i$ as $voicbi$.

Let $L_2 = \{ w \in \Sigma^* \mid w$ represents a path from $h$ to $b$ in $G \}$. Note that all strings in $w$ must start with $h$ and end with $b$. The paths represented this way do not necessarily need to be simple paths.

ii. **(3 Points)** Write a regular expression for $L_2$. Then, write a brief description (at most fifty words) of how your regular expression works.

Brief (at most fifty-word) description of how this regular expression works:
iii. **(7 Points)** Below is a Venn diagram showing the overlap of different classes of languages we've studied so far. We have also provided you a list of numbered languages. For each of those languages, draw where in the Venn diagram that language belongs. As an example, we've indicated where Language 1 and Language 2 should go. No proofs or justifications are necessary, and there is no penalty for an incorrect guess.

As a refresher from Problem Set Nine, we’ve defined **co-RE** = { \( L \mid \overline{L} \in \text{RE} \) }.

1. \( \Sigma^* \)
2. \( L_D \)
3. \( \{ w \in \{C, S, 1, \theta, 3\}^* \mid w = \text{CS103} \} \)
4. \( \{ w \in \{C, S, 1, \theta, 3\}^* \mid \text{the sum of all the digits in } w \text{ (including repeated digits) is } 103 \} \)
5. \( \{ w \in \{C, S, 1, \theta, 3\}^* \mid w \text{ contains } 103 \text{ copies of the substring } \text{CS103} \} \)
6. \( \{ \langle M \rangle \mid M \text{ is a Turing machine and } \mathcal{L}(M) \subseteq \{C, S, 1, \theta, 3\}^* \} \)
7. \( \{ \langle M \rangle \mid M \text{ is a Turing machine and } \{C, S, 1, \theta, 3\}^* \subseteq \mathcal{L}(M) \} \)
8. \( \{ \langle M \rangle \mid M \text{ is a Turing machine and } \text{CS103} \in \mathcal{L}(M) \} \)
9. \( \{ \langle M \rangle \mid M \text{ is a Turing machine and } \mathcal{L}(M) \in \text{CS103} \} \)
Problem Five: Finite Automata  

(We recommend spending about 40 minutes on this problem.)

On Problem Sets Six, Seven, and Eight, you explored the 5-tuple definition of DFAs. This question is designed to let you show us what you've learned in the process.

As a refresher from the problem sets, a DFA is formally defined as a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where \(Q\) is the set of states in the DFA, \(\Sigma\) is the DFA's alphabet, \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function, \(q_0\) is the start state, and \(F\) is the set of accepting states.

Given a DFA \(D = (Q, \Sigma, \delta, q_0, F)\), for any \(q \in Q\) and any \(w \in \Sigma^*\), we'll have \(q \bowtie w\) denote the state that you'd end up in if you started in state \(q\) and read string \(w\). This idea generalizes the extended transition function you saw in Problem Sets Seven and Eight; for example, \(q_0 \bowtie w = \delta^*(w)\), since both sides of the equality represent “where you end up if you start in \(q_0\) and read the string \(w\)”.

Just as we defined the function \(\delta^*\) recursively, the easiest way to formally specify what \(\bowtie\) means is with a recursive definition.

i. (3 Points) Fill in the following blanks to complete the recursive definition of \(\bowtie\). Your answer should be given purely symbolically, making appropriate reference to the 5-tuple components of \(D\), and without using plain English.

**Base case:** if \(q \in Q\), then

\[q \bowtie \varepsilon = \text{_______________________________}\]

**Recursive case:** if \(q \in Q\), \(w \in \Sigma^*\), and \(a \in \Sigma\), then

\[q \bowtie wa = \text{_______________________________}\]
The recursive definition of $\bowtie$ that you developed on the previous page allows us to prove formal results about the behavior of the $\bowtie$ operation.

ii. **(8 Points)** Let $q \in Q$ be an arbitrary state and $x \in \Sigma^*$ be some string. Prove that the following is true for any $y \in \Sigma^*$:

$$q \bowtie xy = (q \bowtie x) \bowtie y.$$  

To do so, use your recursive definition of $\bowtie$ from the previous page. While we recommend using your intuition about DFAs to get a better handle on what this result says, we’re expecting you to reference the formal recursive definition in your proof.

Fun fact: this is the very last proof of CS103! *Congratulations!* 😊
(Extra space for your answer to Problem Five, Part (ii), if you need it.)