Note: The exam is open-textbook/open-notes. However, you may not refer to other resources (e.g., online, classmates, or software). It is important to review the course material before you take the exam.

The following provides a list of the knowledge and skills you should have in order to succeed on the first exam. To prepare for the exam, you should make sure that you have done all assigned reading, attempted all the homework problems and understood the released solutions, and looked at all lecture notes.

1 Alphabets, Strings, Languages

- Know the definitions of alphabet, string, language.
- Be able to describe various languages of strings in set notation.
- Understand the effect of concatenation, Kleene star, and union on strings and as operators on languages.
- Be able to understand and create recursive/inductive definitions of sets of strings.
- Be able to do proofs about strings and languages by induction on the lengths of strings.

2 Deterministic Finite Automata

- Know the definition of DFA as a transition diagram or 5-tuple.
- Understand the transition function $\delta$ and extended transition function $\hat{\delta}$, where $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x))$, $a$ for all states $q$, strings $x$, and symbols $a$.
- Be able to design a DFA to meet a given specification.
- Understand the essence of DFAs: Finite memory. States capture all and only that which needs to be remembered to allow the computation to proceed. Intuition: Counting for some fixed constant $n$ is possible, but counting and remembering an arbitrary value for later comparison is not.
- Be able to describe the language accepted by a given DFA.

3 Nondeterministic Finite Automata

- Understand the basic definition of NFAs, both as a transition diagram and a tuple.
- Understand that an NFA accepts if there is at least one accepting computation.
- Be able to take advantage of nondeterminism by designing a simple NFA where designing a DFA would be much more difficult.
- Understand use of $\epsilon$-transitions and the convenience they offer.
- Be able to eliminate $\epsilon$-transitions from an NFA (compute the $\epsilon$-close).
- Be able to create a DFA that is equivalent to a given NFA by using the subset-of-states construction. Understand not only how to do the construction, but why it works.
• Be able to design **NFAs** to show various closure properties of regular languages, including union, concatenation, and Kleene star, closure under reversal and complementation by creating an **NFA** from a **DFA** by means such as reversing edges, swapping final and non-final states, etc.

4 Regular Expressions

• Be able to give the recursive definition of regular expressions.
• Understand the precedence of the operators in a regular expression (*, concatenation, ∪)
• Given a regular expression \( r \), be able to describe the language \( L(r) \) it denotes.
• Given a **DFA** \( M \), be able to produce a regular expression \( r \) such that \( L(r) = L(M) \), using the state elimination method.
• Given a regular expression \( r \), be able to produce an **NFA** \( M \) such that \( L(M) = L(r) \).

5 Pumping Lemma

• Be able to state the Pumping Lemma for regular languages.
• Understand how (and why) the Pumping Lemma works. This involves understanding that if a string \( w \) is accepted by some \( M \) and \( w \) is as long or longer than the number of states, then \( w \) must “loop” back to some state \( M \) en route to accept. Therefore, we can eliminate (pump 0 times) or repeat (pump 1 or more times, up to infinity) the segment of \( w \) that labels the loop to obtain new accepted strings.
• Be able to use the Pumping Lemma to prove that a given language is not regular.
  – Make sure your argument is coherent, and has all the “glue” so that it reads like a proof.
  – Make sure that your argument does not assume some particular \( n \), or some particular \( x \), \( y \), and \( z \). That is, know that you do not get to choose \( n \) or \( x \), \( y \), \( z \), but that you do get to choose \( w \) and the \( i \) in \( xy^iz \).
• Don’t mistakenly think that the Pumping Lemma can be used to prove that a language is regular.

6 Closure Properties of Regular Languages

• Be able to show that regular languages are closed under the following operations: union, concatenation, Kleene star, complement, intersection, reversal.
• Be familiar with various techniques to show closure properties of regular languages:
  – By constructing **DFAs** or **NFAs** that depend on given **DFAs** or **NFAs**
  – By constructing regular expressions
  – By arguing set-theoretically (for example, if \( L = L_1 - L_2 \) is regular if \( L_1 \) and \( L_2 \) are, because this is just \( L_1 \) intersected with the complement of \( L_2 \), and regular languages are closed under intersection and complement).
• Be able to use closure properties to prove that a language \( L \) is **not** regular, for example, by showing that if \( L \) were regular, by applying closure properties, we could obtain a known non-regular language \( L' \) from \( L \) (and perhaps other regular languages), and therefore we can conclude that since \( L' \) is known to be non-regular, \( L \) is not regular.