Preparation for the Midterm Exam (March 8)

Reading: The exam will be based on the Partee and Sipser chapters covered so far, including some of this week’s material. You are responsible for the following:

- **Partee, Chapter 1.** (Don’t worry about Russell’s paradox).
- **Sipser, Chapter 0:** Sections 0.1 and 0.2 except for 0.2.4 (Graphs) and 0.2.6 (Boolean Logic).
- **Sipser, Chapter 1.1:** Everything! But the exam will exclude the construction used in Theorem 1.12 (intersection and union of regular languages).
- **Sipser, Chapter 1.2,** up to p. 54 only. This includes the formal definition of an NFA.

Cheat sheet: You may bring with you any handwritten notes, diagrams etc. you can fit on one-half of a notebook page (written on one side only). No photocopies are allowed. If you prepare your sheet carefully, the preparation process will be an effective study aid.

Content: Pretty much what we have been doing in class, and in the homeworks. There will be some set theory problems, some questions related to formal definitions and concepts, some constructions of automata. You will need to be very clear on the meaning of the terminology and the notation (your “cheat sheet” will help you cut down on memorizing, but it is no substitute for understanding). You will especially need to know how to design simple deterministic and non-deterministic automata, like in the homework.

Preparation: Your best bet is to study your notes and the reading carefully, especially anything that didn’t make much sense the first time through. Verify your understanding by re-doing any past homeworks that you got wrong.

Be careful not to confuse NFAs and DFAs. They look similar but work according to different rules!

Key Concepts and Topics: A list is given in the following pages. You can use it as a study aid.
Key Concepts and Topics

1 Set Theory

Sets, finite and infinite, and the notation for specifying them. (E.g., what is \( \{ x \mid 5 \leq x \leq 10 \} \)?) Set operations: union, intersection, complement, power set, Cartesian product. Element vs. subset, sets of sets, etc. Ordered pairs and tuples. The standard sets \( \mathbb{N}, \mathbb{Z} \).

Functions: Understand how to write and use function tables for one- and two-argument functions; domain and range; using functions.

Strings: \( \varepsilon \); language, alphabet, length and reverse of a string.

2 Finite State Automata

Languages: Understand the meaning of languages given in set notation or in English; the difference between \( \phi \) (or \{\}) vs. \( \varepsilon \) vs. \{\varepsilon\} vs. \{0\} vs. \( 0 \).

FSA Basics: start state, accept and reject states; formal description (as a 5-tuple); the transition function; requirements for a well-formed FSA (no missing or multiple arrows, labels, etc).

Understanding FSAs: Given an automaton, you should be able to tell: how it works, and exactly how it will work if it is fed some string of my choice; what kinds of strings it accepts and what it rejects. Designing FSAs to order (see next page).

Negating an DFA: How to make an automaton that recognizes the complement of some language for which you have (or can write) a DFA.

NFA Basics: formal description; differences between NFAs and DFAs; \( \varepsilon \)-transitions. Designing simple NFAs.
3 Designing DFAs

General Advice:

1. Remember: it is just as important for your automaton to reject the bad strings as it is for it to accept the good ones!

2. Pay particular attention to what happens once you get to an accept state: if the string continues, do you accept it, reject it, or keep computing? The answer depends on the problem you are solving!

3. Ensure that you construct a valid DFA: no missing or multiple arrows, and don’t forget to circle the Accept states!

Types of Languages to Recognize:
There are several different types of problems, and for each type there are some tricks of the trade for solving it. Make sure you know how to go about solving each kind of problem.

1. Finite list of strings, e.g., \{ aba, bbab, a, \epsilon, aaabb \}.

2. All strings that begin or end with some given string (e.g., with 01001).

3. All strings that contain some substring anywhere inside them.

4. Counting by length; e.g., all strings of length exactly/at least/at most five.

5. Counting something (e.g., all strings that contain exactly/at least/at most three zeros).

6. Counting even/odd (or multiples of three, etc): strings of odd length, strings with an even number of zeros, etc.

7. Problems that do not fit in any of the above categories, or combinations of them: strings that begin with 1 and have odd length, etc. You must work out the approach to each one yourself; the principles of design and the techniques used in the other types will still be useful.