Theory of Computation, CSCI 438 spring 2022
Final review, May 5th

New material is in yellow.

Regular Languages

- Know the definition of a DFA.
- Be able to convert:

Conversion	Using What
Picture ⇒ M= (…), i.e. the formal definition	Definition
Picture ⇐ M= (…)	

- Be able to convert:

Language description ⇒ DFA	Problem solving
Language description ⇐ DFA	

- Know the signature of δ^* for a DFA and be able to use it to define what it means for a DFA to accept a string, and therefore for the DFA to recognize a language
- Know the definition of a regular language
- Be able to prove that regular languages are closed under complementation, union and intersection

Nondeterminism

- Know the definition of an NFA
- Be able to convert:

DFA ⇒ NFA	Theorem: A language is regular iff some NFA recognizes it
DFA ⇐ NFA	

- Be able to prove that a language is regular iff some NFA recognizes it.
- Be able to prove closure of regular languages under concatenation and star-closure

Regular Expressions

- Know the definition of regular expressions
- Know situations where regular expressions are used
- Be able to convert:

Language description ⇒ Reg-ex	Problem solving
Language description ⇐ Reg-ex	

- Be able to convert:

NFA ⇒ Reg-ex	Problem solving. Once you have a solution you could prove that your solution works via induction on the length of the string, but I won't ask you to do this.
NFA ⇐ Reg-ex	

- Know the pumping lemma for regular languages and be able to prove that a language is not regular using the pumping lemma for regular languages

Context-Free Languages

- Know the definition of a context-free grammar
- Know situations where context-free grammars are used
- Know what it means for a variable to be able to "**derive**" a string
- Know the definition of a context free language
- Know what it means for a grammar to be "**ambiguous**" and what it means for a language to be "**inherently ambiguous**"
- Be able to prove that context-free languages are closed under union, concatenation and star-closure
- Be able to convert:

Language description ⇒ CFG	Problem solving
Language description ⇐ CFG	

- Know the definition of a PDA.
- Be able to convert:

Language description ⇒ PDA	Problem solving
Language description ⇐ PDA	

- Know the format of a grammar which is in Chomsky Normal Form.
- Be able to convert:

CFG ⇒ CFG in Chomsky normal form	Theorem 2.29: Any context-free language is generated by a context-

	free grammar in Chomsky normal form.

- Be able to convert:

PDA $\Leftarrow$ CFG	Theorem 2.20: A language is context-free iff some PDA recognizes it. For both directions you only need to be able to do the conversion

- Know the pumping lemma for context-free languages and be able to prove that a language is not context-free using the pumping lemma for context-free languages

Turing Machines

Turing machines are describable at various levels:

- Know the definition of a TM
- Know what it means for a language to be Turing-decidable and Turing-recognizable.
- Be able to convert:

Language description $\Rightarrow$ TM	Problem solving.

- Given a variation on the definition of a TM, be able to show that it computes the same class of languages as a TM (that is, it computes the class of Turing-recognizable languages).

Variation of a TM $\Rightarrow$ TM	Use constructive proofs
Variation of a TM $\Leftarrow$ TM	

  (You should know that a non-deterministic TM can be converted into a regular TM and you can describe the proof from a high level, but I will not ask detailed questions on the proof.)
- Know the Church-Turing Thesis and its implications.
- Know what is meant by an algorithm and know the format in which we will be writing algorithms in this class.
- Know what is meant by an algorithm and be able to write high level algorithms given a problem description.

Decidability

- Know what is meant by acceptance problems (A_{DFA}, A_{NFA}, A_{CFG}, etc.), be able to formulate them, and to prove if they are decidable or recognizable.
- Similar to the above, know what is meant by the "empty" (E_{DFA}, E_{NFA}, E_{CFG}, etc.) problems, and the "equivalence" problems ($EQ_{m1,m2}$), be able to formulate them, and to prove if they are decidable or recognizable.

- Be able to prove that A_{TM} is not decidable, yet is Turing-recognizable.
- Know what a Universal Turing machine is.
- Know what it means for two sets to be the same size, even when the sets may be infinite.
- Know what it means for a set to be countable.
- Know Cantor's diagonalization process and be able to use it to show that the real numbers are not countable.
- Be able to draw the Chomsky hierarchy and give sample languages within each class.

Complexity Theory

- Know the difference between computability theory and complexity theory.
- Know that computability theory does not depend on the model of computation (as long as that model is "reasonable") while complexity theory does depend on the computation model.
- Know what is meant by a verifier for a language and be able to describe what the certificate is.
- Know that a language is in P if and only if it is decided on a deterministic single tape TM in polynomial time Turing machine.
- Be able to prove that a language is in P.
- Know that a language is in NP if and only if it can be verified on a deterministic single tape TM in polynomial time Turing machine (note that this is our definition of NP).
- Be able to prove that a language is in NP.
- Know that a language is in NP if and only if it is decided by some nondeterministic polynomial time Turing machine.
- Know and understand that P is the class of languages for which membership can be decided quickly while NP is the class of languages for which membership can be verified quickly.