

Easy warm up (2pts each, 10 pts total):

1. Give the formal definition for $f(n) = \Theta(g(n))$.

Answer: We say that $f(n) = \Theta(g(n))$ if there exist positive constants c_1 , c_2 and n_0 such that $c_1g(n) \leq f(n) \leq c_2g(n)$ for all $n > n_0$.

2. What is the Church-Turing Thesis?

Answer: The Church-Turing thesis states that any algorithm can be mapped to an equivalent Turing Machine and vice-versa. In other words, any computable function can be represented by a TM. Sipser says it in a slightly different way: the intuitive notion of algorithms equals Turing machine algorithms (Figure 3.22). This is an important conclusion because it embodies the power of Turing machines to be able to perform any possible algorithm.

3. What is the difference when we say a TM decides a language vs. accepts a language?

Answer: A TM *decides* a language if it halts whether it is both accepting or rejecting a string in the language. It is said to *accept* a language when it halts only when the string is in the language (if the string is not in the language, the TM might not halt).

4. What do the letters “NP” in the complexity class NP stand for?

Answer: Non-deterministic Polynomial time, because they are the class of algorithms that run on a non-deterministic TM in polynomial time. Non-polynomial time is incorrect!

5. In class we talked about problems that are NP-complete, such as 3-SAT. Suppose in your research you come accross a new problem which you suspect of being NP-complete. How do you demonstrate that the new problem is NP-complete?

Answer: We need to show two things. First that the problem is in NP. Second, that all other problems in NP (or an existing NP-complete problem such as 3-SAT) can be reduced to it.

Requiring a little more thought (6pts each, 30pts total):

1. A language is called *Turing-recognizable* if there is some Turing machine that recognizes it, i.e. it accepts the language. Use the halting problem to show that there are languages that are not Turing-recognizable.

Answer: Although the halting problem is undecidable, it is Turing recognizable. We know that a language is decidable iff it is Turing-recognizable and co-Turing-recognizable. Hence the complement of the halting problem is an example of a language that is not Turing-recognizable. If it were, then the halting problem would be decidable.

2. In this course we have studied various classes of languages. Which class of languages is closed under complement and union but not under intersection?

Answer: This is not possible by DeMorgan's Law. If it is closed under complement and union it must be complement under intersection, because you can write intersection using only complement and union: $A \cap B = \overline{\overline{A} \cup \overline{B}}$.

3. What are the equivalence classes of the language $L = \{w \mid w \text{ has an odd number of 1's}\}$?

Answer: There are two equivalence classes: the first is the set of strings with an odd number of 1's, the second is the set of strings with an even number of 1's. They correspond directly to the two states in the minimal DFA that recognizes this language.

4. Show that the regular languages are closed under difference. That is, if L_1 and L_2 are regular languages, then $L_3 = L_1 - L_2$ (which is the set of strings in L_1 that are not in L_2) is also regular.

Answer: There are a couple of ways to show this. One way is to use boolean logic to show that $L_1 - L_2 = L_1 \cap \overline{L_2}$. Since regular languages are closed under complement and intersection, then L_3 is regular. Another way to show this is to construct a DFA that recognizes L_3 out of two DFA's M_1 and M_2 that recognize L_1 and L_2 respectively. The new DFA should have states that are the cross product of the states of M_1 and M_2 : $Q = Q_1 \times Q_2$, but the accept states are only those that are accepting for M_1 but not for M_2 . This new machine will recognize $L_1 - L_2$.

5. Show that decidable languages are closed under complement.

Answer: Suppose decidable language L is decided by a TM. Simply change the accept and reject states of the TM and now the TM will decide the complement. Hence the complement of a decidable language is also decidable.

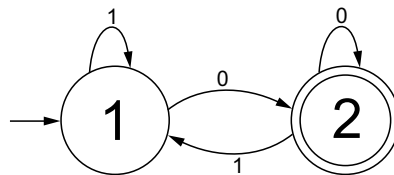
Regular problems (12pts each, 60 pts total):

1. Suppose you have set S which consists of 10 different integers between 0 and 100, inclusive. Prove that for *any* set S , there exist different subsets A and B (where $A \subset S$ and $B \subset S$ and $A \neq B$) such that sum of all the elements of A is equal to sum of all the elements of B .

Answer: The sum of all the numbers in S is upper-bounded at 1000 because all 10 numbers are less than 100, so it is impossible that the sum could be greater than $10 \times 100 = 1000$). Note that the strict upperbound is actually less than that because there cannot be any duplicates, but this loose upperbound suffices for our proof. Likewise, we can see that the sum of the numbers in S is lower-bounded at 0. Therefore, the sum of the elements of A and B must also be between 0 and 1000 because $A \subset S$ and $B \subset S$. While there are $2^{10} = 1024$ different subsets of S possible, there are only 1001 possible sums that they could add up to (from 0 to 1000). Therefore, by the pigeon-hole principle, at least two different subsets must have the same sum. There are simply not enough unique sum totals to go around so that each subset can have their own unique sum.

2. Suppose $L = \{w \mid w \text{ is an even integer written in unsigned binary}\}$ where you ignore the leading zeros. For example, the string “00110” is binary for 6 (which is even) and so it is in L . However, the string “111” is binary for 7 and so it is not in L . We assume that “0” is in L because it is even, and that $\epsilon \notin L$. Is L a regular language? If so write an DFA that recognizes it. If not, explain why it isn't regular using an argument based on equivalence classes or the pumping lemma.

Answer: Yes, L is a regular language. In fact, it is the language of strings that end in 0. The DFA that recognizes L is shown below.



3. Show that the regular languages are closed under the *cycle* operation, which is defined as $cycle(L) = \{w \mid \text{where } w = xy \text{ such that } yx \in L\}$. For example, if $L = \{01, 011\}$, then $cycle(L) = \{01, 10, 011, 110, 101\}$. Show that if L is regular, then $cycle(L)$ is also regular.

Answer: L is regular, so there exists a DFA D that recognizes it. To show that $cycle(L)$ is regular, we show how to construct an NFA that recognizes $cycle(L)$ using the DFA D . First, duplicate D so that there are as many copies of it as it has states. Since D has a finite number of states, there will be a finite number of copies of D , each with a finite number of states. Now create a start node for our NFA and attach ϵ transitions from it to a single state in each of the copies of D . Each transition will go to a different state of D , which is why we need the same number of copies of D as it has states. For each of the copies of D , we make the following modifications. First, we add ϵ transitions between its accept states and its own start node. We then change its accept states into non-accept states and turn the node which is connected to the start node of the NFA the final accept state for each copy of the DFA.

So now every copy of D in the NFA will have a single unique accept state, which is pointed to by an ϵ transition from the NFA start node. As the string is processed, the NFA tries in parallel for each copy of D to reach the original accept states (which recognizes x) and then is sent back to the start node of the DFA to see if y sends it back to the original internal state within the DFA from which it started. We note that this NFA will recognize $cycle(L) \cup \epsilon$ because of the way we have ϵ transitions to the new accept states in every copy of the DFA. However, since regular languages are closed under union and $\{\epsilon\}$ is a regular language then $cycle(L)$ is a regular language.

4. Show that the language $A = \{w \mid w \in \{a, b, c\}^* \text{ and contains equal numbers of } a\text{'s, } b\text{'s and } c\text{'s}\}$ is not CFL. Use the fact that we know that $\{a^n b^n c^n\}$ is not context free and the closure properties for CFLs.

Answer: We can write $\{a^n b^n c^n\}$ as $A \cap \{a^* b^* c^*\}$, where $\{a^* b^* c^*\}$ is a regular language. We know that the intersection between a CFL and a regular language is CFL. Since $\{a^n b^n c^n\}$ is not CFL, then A must not be CFL either.

5. The language of odd-length palindromes with a symbol in the middle $L = \{w\#w^R \mid w \in \{0, 1\}^*\}$ and its complement \bar{L} are both CFL. Is L DPDA? If so, construct a DPDA that recognizes it by drawing the DFA and using the notation “ $a, b \rightarrow c$ ” (when machine reads symbol a from the input string it should replace b at the top of the stack with c). Assume the input string alphabet of $\Sigma = \{0, 1, \#, \sqcup\}$ (the string is padded with a “ \sqcup ” after it finishes) and a stack alphabet $\Gamma = \{0, 1, \epsilon\}$ (an empty stack means that the top of the stack contains ϵ). Naturally, pushing a symbol onto the stack can be written as $\epsilon \rightarrow x$ and popping as $x \rightarrow \epsilon$. If the language is not a DPDA, explain why it does not exist. Regardless of whether it is DPDA or not, we do know that it is CFL, so write out the grammar that generates L .

Answer: In the homework we had a similar question, except there was no intermediate symbol and we showed the language was not DPDA. However, here we have the intermediate symbol “ $\#$,” so in this case the DPDA knows when to start popping the stack. The DPDA is shown below. The grammar to generate the language is:

$$S \rightarrow 1S1, 0S0, \#$$

