COMP 330 Winter 2021
Final

Belle Pan 260839939

22nd April 2021, 2:00 PM

Solution 1.
Classify the following language as either

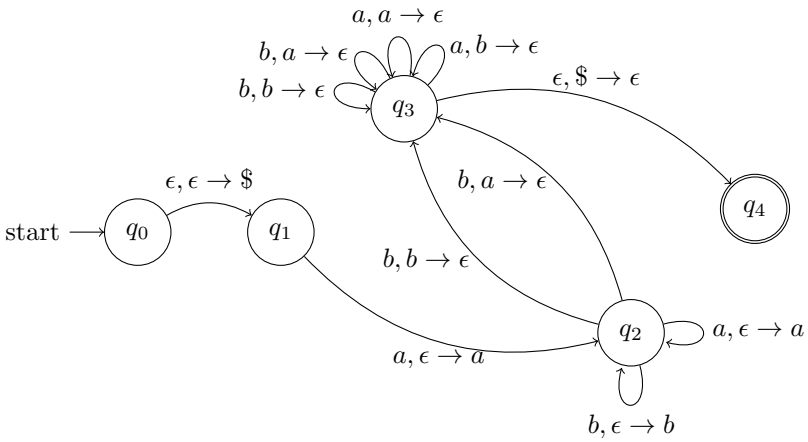
1. **regular or**
2. **context-free but not regular or**
3. **decidable but not context-free**

L consists of even length non-empty strings over the alphabet {a, b}, so every word in L can be written as w_1w_2 with $|w_1| = |w_2|$ (i.e. $|w_1|$ and $|w_2|$ have the same length); we require that w_1 starts with a and w_2 starts with b. If you think it is regular it is sufficient to give an NFA. If you think that it is context-free but not regular you have to give a PDA or CFG and a pumping lemma proof that it is not regular; you do not have to prove that your CFG or PDA is correct but the description should be clear. For the last case you need to give a pumping lemma proof that it is not context-free as well as an algorithm for deciding whether a given string is in the language.

This language is not regular. Below is a pumping lemma proof using the demon-angel strategy:

- The demon picks a number p.
- The angel picks the word a^pb^p.
- The demon is forced to pick a y value that is made exclusively of a's due to the constraints $|xy| \leq p$ and $|y| > 0$. Suppose that $y = a^k$, where $|y| = k$.
- The angel picks $i = 2$ such that the new word is $a^{p+k}b^p$ and $|a^{p+k}b^p| = 2p + k$. Words in L must be made of two words of the same length, with the first one beginning in a and the second beginning in b, but this is impossible with $a^{p+k}b^p$; in this case, w_1 must be $a^{p+\frac{k}{2}}$ and w_2 must be $a^{\frac{k}{2}}b^p$, which violates the structure of words in L. Thus, $a^{p+k}b^p$ must not exist in L
- Therefore, by the pumping lemma, this language is indeed not regular.

Below is the PDA for the language, which indicates that it is context-free:



The PDA above ensures that our first input letter is an a, and then begins to push inputs onto the stack. Then, using non-determinism (see q_2), the PDA is able to find the start of w_2; the start of w_2 must be a b, so

the machine gives the choice of either continuing to push inputs onto the stack or to begin popping off elements from the stack when it reads a b. At this point, the machine had already pushed $|w_1|$ items onto the stack, so by reading w_2 and popping an element off the stack for every input, we should be left with an empty stack at the end of the word, as $|w_1| = |w_2|$. The PDA accepts only if the stack is empty at the end.

Solution 2.
Suppose that I am given two regular languages R_1 and R_2 from the same fixed two-letter alphabet. Describe in outline how you would algorithmically test whether $R_1 \subseteq R_2$. You can assume that you are given explicit descriptions of DFA's to recognize the languages. You have to describe in outline the steps you could take to answer the question. The algorithm has to work without any special assumptions about the languages beyond the fact that they are regular. Your description should be high-level : for example, if you are going to say as some stage that you are testing whether an accept state is reachable from the start state you can just say that, you do not have to give the details of an algorithm to test reachability. Similarly if you want to say that you want to test something about R^* you need not give an explicit description of how you construct a DFA to recognize $R*$. I am not suggesting that either of these steps is part of the required answer; they are just examples.

Let us first note some closure properties sets that may be applied to regular languages, namely that if a regular language is a subset of another regular language, then the intersection of the former and the complement of the latter must be empty. So, in this case, if $R_1 \subseteq R_2$, then $R_1 \cap \overline{R_2} = \emptyset$.
Keeping this property in mind, we begin constructing an algorithm to test if $R_1 \subseteq R_2$:

1. Given explicit descriptions of the DFA's that recognize the languages, let us construct a new DFA that keeps track of pairs of states, with the first being a state in the DFA that recognizes R_1 and the second being a corresponding state in the DFA that recognizes R_2.

2. This new DFA begins tracking at the start state and collects a list of reachable pairs of states from both DFA's. This DFA should also keep track of whether or not the states in each pair are accept states.

3. If $R_1 \subseteq R_2$, there should not be any accept states from the the former DFA paired with a reject state from the latter DFA, as that would indicate that R_1 has strings that are not present in R_2 which violates the aforementioned closure property.

4. Similarly, if we see that all accept states from the former DFA are paired with accept states in the latter DFA, we know that all strings in R_1 are in R_2, and thus $R_1 \subseteq R_2$.

Solution 3.
One of the following sets is CE and the other is co-CE.

1. $\{M : |L(M)| \leq 330\}$,
2. $\{M : |L(M)| \geq 330\}$,

where M is a Turing machine description, $L(M)$ means the language recognized by M and $|L(M)|$ is the size of this language. Identify which set is CE and which is co-CE and give proofs for both. There is no credit for guessing, you must give the proofs. You can take it for granted that neither set is decidable.

It is clear that the two sets are complements - one set is for all Turing machines that recognize no more 330 words, and the other set is for all Turing machines that recognize no less than 330 words - so if one set is CE then the other must be co-CE.
The computability of the two sets can be seen using the following algorithm:

- Let us construct a Turing machine M' that simulates all Turing machines on all the possible words of the given alphabet.
- If the machine accepts a word, we note it down and move onto the next input.
- If at some point the machine accepts more than or equal to 330 inputs, then M' halts and we simulate the next machine.

Through the above algorithm, a Turing machine that accepts equal to or more than 330 inputs would be easily identified. Therefore, we know that the set $\{M : |L(M)| \geq 330\}$ must be CE. And through our first observation, that the two sets are complements, we also know that $\{M : |L(M)| \leq 330\}$ is co-CE.

To further prove that $\{M : |L(M)| \leq 330\}$ is not CE, we can use a reduction from the complement of the Halting Problem, which we denote as $\overline{H_{TM}}$.

- Let there be an encoding of a Turing machine $\langle M, x\rangle = \langle M'\rangle$ that takes as input w and completely ignores it. It instead simulates M on x and accepts if M halts on x.
- If M does not halt on x then M' does not accept any input, and thus $|L(M')| \leq 330$ and $M' \in \{M : |L(M)| \leq 330\}$. From this relation, we know that $\langle M, x\rangle \leq_m \overline{H_{TM}}$
- However, if M does halt on x then M' accepts all inputs, and thus $|L(M')| > 330$; thus $M' \notin \{M : |L(M)| \leq 330\}$.

As such, we know that $\{M : |L(M)| \leq 330\}$ must not be CE and is in fact co-CE.

Solution 4.
One of the following questions is decidable, the other is undecidable. For the decidable question give an outline of an algorithm to solve it. For the undecidable question give a short proof of undecidability by showing that it is is equivalent to a question that we know to be undecidable. Here R is a regular language and L is a context-free language specified however you prefer and the question in each case is whether the indicated inclusion holds.

- $R \subseteq L$
- $L \subseteq R$

$R \subseteq L$ is undecidable.

- If $R \subseteq L$ and $R = \Sigma^*$, then L must $= \Sigma^*$. However, whether or not a CFL contains all strings for a given alphabet (i.e. CFL $= \Sigma^*$) is undecidable. Therefore, $R \subseteq L$ is undecidable.
- In fact, such a reduction exists and we may use it within our proof: $\overline{A_{TM}} \leq ALL_{CFG}$, where $ALL_{CFG} = \{\langle G \rangle | G \text{ is a CFG that generates all strings}\}$ and $\overline{A_{TM}} = \{\langle M, w \rangle | M \text{ is a Turing machine that rejects or loops on } w\}$.
  - Suppose we have a Turing machine M_1 that decides ALL_{CFG}; M_1 will accept a grammar G if G generates all strings for a given alphabet, and will reject G if it does not generate all strings.
  - Then, we construct a Turing machine M_2 that accepts as input $\langle M, w \rangle$ and creates a CFG based on the strings that are rejected by or looping in M. In other words, we are relaying information about all the strings that are rejected by M and ignoring the strings that are accepted. It outputs this grammar, G, to M_1
  - So, if M rejects or loops on an input w then the output grammar, G, generates all strings and if w is accepted then G must not generate all strings.
  - However, we can easily realize that this requires us to solve $\overline{A_{TM}}$, as we need to identify whether or not the machine rejects or loops for a given input, which is known to be undecidable.
- Therefore, $R \subseteq L$ must be undecidable

$L \subseteq R$ is decidable.

- $L \subseteq R$ iff $L \cap \overline{R} = \emptyset$. We know that regular languages are closed under complement, hence $\overline{R}$ is regular; and because regular languages are also context-free, and the intersection of context-free languages is also context-free, we know that $L \cap \overline{R}$ must be context-free.
- The emptiness problem is decidable for context-free languages, so we know that $L \cap \overline{R} = \emptyset$ must also be decidable. Thus, we know that $L \subseteq R$ must be decidable given that we can easily check if $L \cap \overline{R} = \emptyset$.
- The algorithm to check if $L \cap \overline{R} = \emptyset$ and $L \subseteq R$ is as follows:
  1. First, we generate a context-free grammar for $L \cap \overline{R}$.
  2. Then, we create a set of all instructions from the grammar that are generating, which we denote GEN- i.e. if an instruction produces a terminal symbol, it belongs to GEN.
  3. We first put all terminal symbols in the set, then we verify if any instructions produce only symbols in this set; if there are instructions that do so, we place them in the set as well. This process repeats until there are no more changes in GEN.
  4. Finally, we check if the start instruction S is within GEN. If it is, then we know that $L \cap \overline{R} \neq \emptyset$ as the grammar can generate strings. If S is not present in the set, then we know that S cannot generate any strings and thus $L \cap \overline{R} = \emptyset$.
  5. Then, if we know that $L \cap \overline{R} = \emptyset$, we must also know that $L \subseteq R$.

Solution 5.
Here are 10 multiple-choice questions; each question has 4 choices. In every case, exactly one statement is correct. You must indicate the correct answer. You are not required to give any reasons for your answer. Each question is worth 4 points. Please just write the letter corresponding to your choice for each question. Do not write any explanations.

1. (d)
2. (c)
3. (d)
4. (a)
5. (b)
6. (a)
7. (c)
8. (a)
9. (b)
10. (d)