HW5CSE105F24: Homework assignment 5 solution

CSE105F24

Due: November 19, 2024 at 5pm, via Gradescope

In this assignment,

You will practice analyzing, designing, and working with Turing machines. You will use general constructions and specific machines to explore the classes of recognizable and decidable languages. You will explore various ways to encode machines as strings so that computational problems can be recognized.

Resources: To review the topics for this assignment, see the class material from Weeks 6 and 7. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Sections 3.1, 3.3, 4.1 Chapter 3 exercises 3.1, 3.2, 3.5, 3.8. Chapter 4 exercises 4.1, 4.2, 4.3, 4.4, 4.5.

Assigned questions

1. **Classifying languages** (10 points): Our first example of a more complicated Turing machine was of a Turing machine that recognized the language $\{w\#w \mid w \in \{0,1\}^*\}$, which we know is not context-free. Let's call that Turing machine M_0. The language

$$L = \{ww \mid w \in \{0,1\}^*\}$$

is also not context-free.

(a) (*Graded for correctness*) [1] Choose an example string of length 4 in L that is in **not** in $\{w\#w \mid w \in \{0,1\}^*\}$ and describe the computation of the Turing machine M_0 on your example string. Include the contents of the tape, the state of the machine, and the location of the read/write head at each step in the computation.

Solution: Consider the string 0101 which is in $L = \{ww \mid w \in \{0,1\}^*\}$ but not in $\{w\#w \mid w \in \{0,1\}^*\}$. The computation of M_0 on 0101 is the following. We can see that the Turing machine rejects the string 0101 because the computation enters q_{rej}.

$q_1 \downarrow$						
0	1	0	1	$\sqcup$	$\sqcup$	$\sqcup$
	$q_2 \downarrow$					
x	1	0	1	$\sqcup$	$\sqcup$	$\sqcup$
		$q_2 \downarrow$				
x	1	0	1	$\sqcup$	$\sqcup$	$\sqcup$
			$q_2 \downarrow$			
x	1	0	1	$\sqcup$	$\sqcup$	$\sqcup$
				$q_2 \downarrow$		
x	1	0	1	$\sqcup$	$\sqcup$	$\sqcup$
					$q_{rej} \downarrow$	
x	1	0	1	$\sqcup$	$\sqcup$	$\sqcup$

(b) (*Graded for completeness*) [2] Explain why the Turing machine from the textbook and class that recognized $\{w\#w \mid w \in \{0,1\}^*\}$ does not recognize $\{ww \mid w \in \{0,1\}^*\}$. Use your example to explain why M_0 doesn't recognize L.

[1] This means your solution will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

[2] This means you will get full credit so long as your submission demonstrates honest effort to answer the question. You will not be penalized for incorrect answers. To demonstrate your honest effort in answering the question, we expect you to include your attempt to answer *each* part of the question. If you get stuck with your attempt, you can still demonstrate your effort by explaining where you got stuck and what you did to try to get unstuck.

Solution: Since the Turing machine M_0 recognizes $\{w\#w \mid w \in \{0,1\}^*\}$, the set of all strings that are accepted by M_0 would contain exactly one # symbol. But all strings in $L = \{ww \mid w \in \{0,1\}^*\}$ do not contain #, which means that they will not be accepted by M_0. Take the example given in part (a), we can see that the computation of M_0 on the string enters q_{rej} after encountering the first blank symbol on the tape. This will happen for any string in L since M_0 do not see the expected # symbol. Therefore, all strings in L will be rejected by M_0, i.e. M_0 doesn't recognize L.

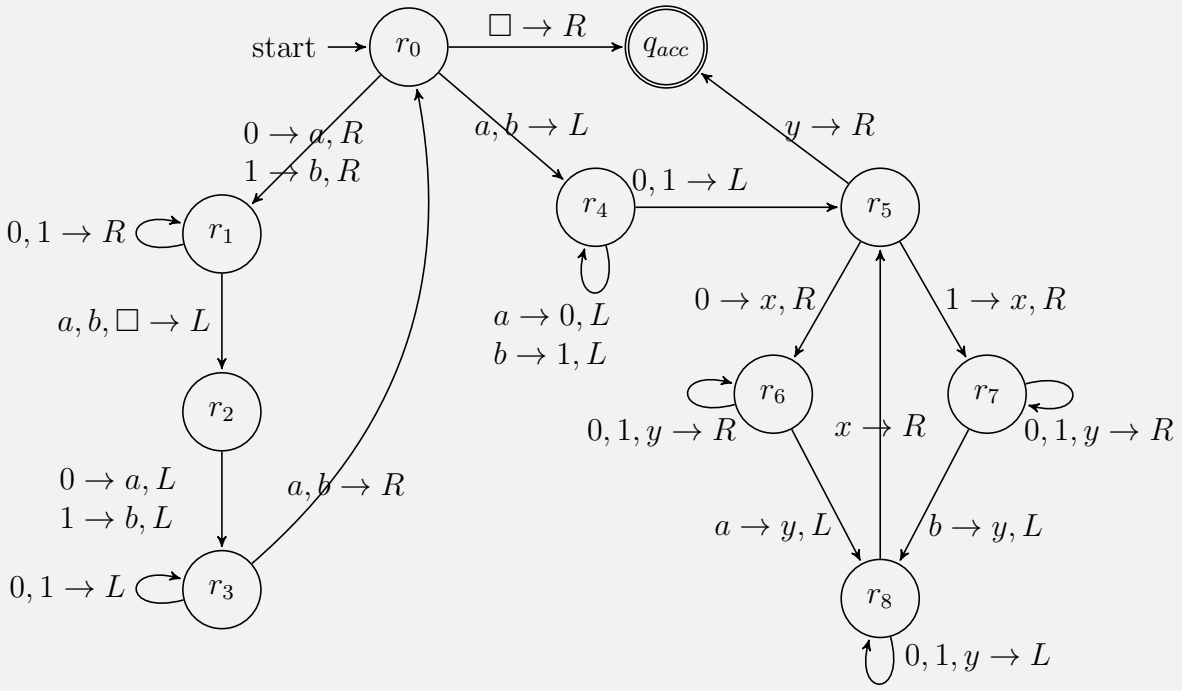
(c) (*Graded for completeness*) Explain how you would change M_0 to get a new Turing machine that does recognize L. Describe this new Turing machine using both an implementation-level definition and a state diagram of the Turing machine. You may use all our usual conventions for state diagrams of Turing machines (we do not include the node for the reject state $qrej$ and any missing transitions in the state diagram have value $(qrej, \square, R)$; $b \to R$ label means $b \to b, R$).

Solution: The new Turing machine need to first find the middle of the input string, and then compare the first half with the second half using the similar zig-zag strategy in M_0. The detail is as follows:

Implementation-level definition:
Zig-zag across the tape, mark the left-most symbol that is a 0 or 1 into a or b correspondingly, and mark the right-most symbol that is a 0 or 1 into a or b correspondingly. Repeat the above steps. If after marking a 0 or 1 on the left, there is no more 0 or 1 on the right to mark, reject, since the input is of odd length. Otherwise, after marking a 0 or 1 on the right and there is no more 0 or 1 on the left, the tape head is now pointing at the middle of the input. Scan left, convert the first half of the content back into 0's and 1's based on the marks (a becomes 0 and b becomes 1). The tape will now contain the original 0's and 1's representing the first half of the input string, followed by a's and b's representing the second half. The tape head should be back to the left-most position at this point.

Then, zig-zag across the tape to corresponding positions on the first half and the second half to check whether a 0 in the first half matches with an a in the second half, and whether a 1 in the first half matches with a b in the second half. If they do not, reject. If all of the pairs match, accept.

State diagram:
Consider the input alphabet $\Sigma = \{0,1\}$ and the tape alphabet $\Gamma = \{0,1,a,b,x,y,\square\}$.



Here is a short explanation for the state diagram: Empty string will be accepted immediately. States r_0, r_1, r_2, r_3, r_4 process the input string into 0's and 1's in the first half and a's and b's in the second half. Note that computations on odd-length strings will enter q_{rej} after entering r_2 and not finding another 0 or 1 to mark. For even-length strings (that are not empty string), the computation will reach r_5, and r_5, r_6, r_7, r_8 corresponds to checking whether the first half and the second half match. Iff they match, the computation will enter q_{acc}.

2. **Closure** (18 points): Suppose M is a Turing machine over the alphabet $\{0, 1\}$. Let $s_1, s_2, \ldots$ be a list of all strings in $\{0, 1\}^*$ in string (shortlex) order. We define a new Turing machine by giving its high-level description as follows:

M_{new} = "On input w :

1. For $n = 1, 2, \ldots$
2. For $j = 1, 2, \ldots n$
3. For $k = 1, 2, \ldots, n$
4. Run the computation of M on $s_j w s_k$
5. If it accepts, accept.
6. If it rejects, go to the next iteration of the loop"

Recall the definitions we have: For each language L over the alphabet $\Sigma_1 = \{0, 1\}$, we have the associated sets of strings

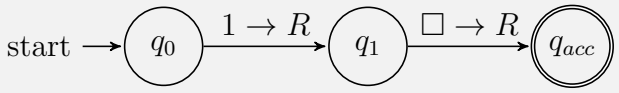
$$SUBSTRING(L) = \{w \in \Sigma_1^* \mid \text{there exist } x, y \in \Sigma_1^* \text{ such that } xwy \in L\}$$

and

$$EXTEND(L) = \{w \in \Sigma_1^* \mid w = uv \text{ for some strings } u \in L \text{ and } v \in \Sigma_1^*\}$$

(a) (*Graded for correctness*) Prove that this Turing machine construction **cannot** be used to prove that the class of decidable languages over $\{0,1\}$ is closed under the $EXTEND$ operation. A complete and correct answer will give a counterexample which is a set A over Σ_1 that is decidable, along with a definition of Turing machine M_A that decides A (with a justification why this Turing machine accepts all strings in A and rejects all strings not in A), and then either a description of the language of M_{new} that results when setting the Turing machine $M = M_A$ and an explanation why $L(M_{new}) \neq EXTEND(A)$ or a description why M_{new} is not a decider and therefore can't witness that $EXTEND(A)$ is decidable.

Solution: Take $A = \{1\}$ as a counterexample. The state diagram of M_A that decides A is the following, where input alphabet $\Sigma_1 = \{0,1\}$ and tape alphabet $\Gamma = \{0,1,\square\}$:



M_A accepts string 1 with the following accepting computation: $q_01,\ 1q_1, 1\square q_{acc}$. Since 1 is the only string in A, we proved that M_A accepts all strings in A. Each string not in A is either the empty string or starts with 0 or starts with 1 and has length greater than one: the computation of M_A on the empty string or on strings that start with 0 will enter q_{rej} after its initial configuration (the missing arrows from q_0 with first label 0 or $\square$ are assumed to go to q_{rej}); the computation of M_A on strings that start with 1 and have length greater than one will start by transitioning from $q0$ to $q1$ (leaving the tape unchanged) and then will transition to q_{rej} from $q1$ (the missing arrows from q_0 with first label 0 or 1 are assumed to go to q_{rej}). Thus, all strings not in A will be rejected by M_A. Therefore, M_A decides A.

When setting $M = M_A$, we can trace the construction of M_{new} to analyze $L(M_{new})$. Consider the string 10. This string is in $EXTEND(A)$, as witnessed by taking $u = 1$ and $v = 0$ in the definition of $EXTEND$. When we take $w = 10$ as the input to M_{new}, any choice of s_j and s_k will result in $s_jws_k = s_j10s_k$ is a string that has length at least two and so (from our analysis of M_A above) s_jws_k will be rejected by M_A. This means that the computation of M_{new} on 01 will loop forever, as n gets incremented and new choices of s_jws_k are input to M_A. In particular, $01 \notin L(M_{new})$. Therefore, $L(M_{new}) \neq EXTEND(A)$.

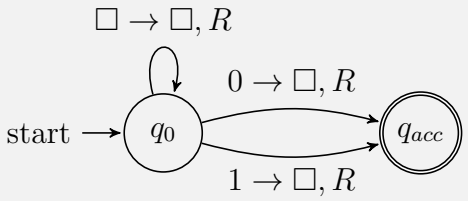
Since we've show that M_{new} loops on input 10, we've shown that M_{new} is not a decider and so (even if it accepted the right strings to recognized $EXTEND(A)$), it still couldn't be used to prove that the class of decidable languages over $\{0,1\}$ is closed under any operation.

We notice that $L(M_{new}) = \{\varepsilon, 1\} = SUBSTRING(A)$ (details omitted since this isn't part of the question, but it's worth thinking through how to prove this).

(b) (*Graded for correctness*) Prove that this Turing machine construction cannot be used to prove that the class of recognizable languages over $\{0, 1\}$ is closed under the $SUBSTRING$ set operation. A complete and correct answer will give a counterexample of a specific language B and Turing machine M_B recognizing it (with a justification why this Turing machine accepts all and only strings in B), and then a description of the language of M_{new} that results when setting the Turing machine $M = M_B$ and an explanation why $L(M_{new}) \neq SUBSTRING(B)$

Solution: Essentially, the problem is that in line 4, M_new runs M indefinitely until it accepts or rejects. So if the computation of M never halts, M_{new} doesn't halt either. This is a problem if a later choice of s_j, s_k could witness membership in $SUBSTRING(L(M))$.

In particular, let $B = \Sigma_1^*\backslash\{\varepsilon\}$ which is recognized by the Turing machine M_B with state diagram (with input alphabet $\Sigma_1 = \{0, 1\}$ and tape alphabet $\Gamma = \{0, 1, \square\}$):



M_B loops on empty string and accepts all other strings, so $L(M_B) = B$.

When setting $M = M_B$, we will show why the constructed M_{new} does not recognize $SUBSTRING(B)$. We know that $\varepsilon \in SUBSTRING(B)$ since $0\varepsilon 0 \in B$. However, when ε is input to the machine, the first iteration of the nested loops has $j = k = 1$ and $s_1 = \varepsilon$ so in step 4, M_{new} glues a copy of ε on each end of ε and runs M_B on $\varepsilon\varepsilon\varepsilon = \varepsilon$. This step never halt so we do not accept ε.

In fact, $L(M_{new}) = \Sigma_1^*\backslash\{\varepsilon\}$, i.e. M_{new} accepts all strings except for ε.

(c) (*Graded for completeness*) Define a new construction by slightly modifying this one that can be used to prove that the class of recognizable languages over $\{0, 1\}$ is closed under $SUBSTRING$. Justify that your construction works. The proof of correctness for the closure claim can be structured like: "Let L_1 be a recognizable language over $\{0, 1\}$ and assume we are given a Turing machine M_1 so that $L(M_1) = L_1$. Consider the new Turing machine M_{new} defined above. We will show that $L(M_{new}) = SUBSTRING(L_1)$... *complete the proof by proving subset inclusion in two directions, by tracing the relevant Turing machine computations*"

Solution: All we need to do is to limit the number of steps M can run. Change line 4 to "Run the computation of M on $s_j w s_k$ for at most n steps."

Let L_1 be a recognizable language over $\{0,1\}$ and assume we are given a Turing machine M_1 so that $L(M_1) = L_1$. Consider the new Turing machine M_{new} defined above. We will show that $L(M_{new}) = SUBSTRING(L_1)$.

- $\subseteq$ Let $w \in L(M_{new})$, i.e. M_{new} accepts the string w. The only way for M_{new} to accept is in step 5. Tracing the definition of M_{new}, arriving in step 5 means there exists some s_j, s_k such that $s_j w s_k$ is accepted by $M = M_1$. This means $s_j w s_k \in L(M_1) = L_1$, so $w \in SUBSTRING(L_1)$ by definition of $SUBSTRING$.
- $\supseteq$ Consider $w \in SUBSTRING(L_1)$. By definition of $SUBSTRING$, there exists $a, b \in \Sigma_1^*$ such that $awb \in L_1$. Moreover, let's say a is the J^{th} string in string order and b is the K^{th} string in string order and the computation of M_1 on awb takes N steps to get to the accept state. When $n = max(J, K, N)$, the nested for loop in lines 2 and 3 of the definition of M_{new} has an iteration where $a = s_J$ and $b = s_K$ are considered and in step 4 feed awb to M_1 and get accepted after N steps. (We are guaranteed to get to this step because each prior iteration of the loop takes no more than n steps.) Thus, M_{new} will accept w.

(d) (*Graded for completeness*) Prove that the class of recognizable languages over $\{0,1\}$ is closed under $EXTEND$.

Solution: Consider any recognizable language L over $\{0,1\}$. There exists a Turing machine M that recognizes L. Then we can show that there exists another Turing machine M' that recognizes $EXTEND(L)$:

M' = "On input w :

1. For $n = 1, 2, \ldots$
2. For $j = 1, 2, \ldots length(w)$
3. Run the computation of M on the first j characters of w for at most n steps
4. If it accepts, accept.
5. If not, go to the next iteration of the loop"

M' will accept all and only strings in $EXTEND(L)$. Therefore, $EXTEND(L)$ is also recognizable. Thus, the class of recognizable languages over $\{0,1\}$ is closed under $EXTEND$.

3. **Computational problems** (12 points): Recall the definitions of some example computational problems from class

Acceptance problem		
... for DFA	A_{DFA}	$\{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$
... for NFA	A_{NFA}	$\{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$
... for regular expressions	A_{REX}	$\{\langle R, w\rangle \mid R \text{ is a regular expression that generates input string } w\}$
... for CFG	A_{CFG}	$\{\langle G, w\rangle \mid G \text{ is a context-free grammar that generates input string } w\}$
... for PDA	A_{PDA}	$\{\langle B, w\rangle \mid B \text{ is a PDA that accepts input string } w\}$
Language emptiness testing		
... for DFA	E_{DFA}	$\{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$
... for NFA	E_{NFA}	$\{\langle A\rangle \mid A \text{ is a NFA and } L(A) = \emptyset\}$
... for regular expressions	E_{REX}	$\{\langle R\rangle \mid R \text{ is a regular expression and } L(R) = \emptyset\}$
... for CFG	E_{CFG}	$\{\langle G\rangle \mid G \text{ is a context-free grammar and } L(G) = \emptyset\}$
... for PDA	E_{PDA}	$\{\langle A\rangle \mid A \text{ is a PDA and } L(A) = \emptyset\}$
Language equality testing		
... for DFA	EQ_{DFA}	$\{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$
... for NFA	EQ_{NFA}	$\{\langle A, B\rangle \mid A \text{ and } B \text{ are NFAs and } L(A) = L(B)\}$
... for regular expressions	EQ_{REX}	$\{\langle R, R'\rangle \mid R \text{ and } R' \text{ are regular expressions and } L(R) = L(R')\}$
... for CFG	EQ_{CFG}	$\{\langle G, G'\rangle \mid G \text{ and } G' \text{ are CFGs and } L(G) = L(G')\}$
... for PDA	EQ_{PDA}	$\{\langle A, B\rangle \mid A \text{ and } B \text{ are PDAs and } L(A) = L(B)\}$

(a) (*Graded for completeness*) Pick five of the computational problems above and give examples (preferably different from the ones we talked about in class) of strings that are in each of the corresponding languages. Remember to use the notation $\langle \cdots \rangle$ to denote the string encoding of relevant objects. *Extension, not for credit:* Explain why it's hard to write a specific string of 0s and 1s and make a claim about membership in one of these sets.

Solution:

i. Let N be an NFA over $\{0, 1\}$ that accepts every string, namely, define

$$N = (\{q_0\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

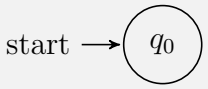
where $\delta : \{q_0\} \times \{0, 1, \varepsilon\} \to \mathcal{P}(\{q_0\})$ is given by $\delta((q_0, 0)) = \delta((q_0, 1)) = \{q_0\}$ and $\delta((q_0, \varepsilon)) = \emptyset$. With this definition, $\langle N, \varepsilon\rangle \in A_{NFA}$

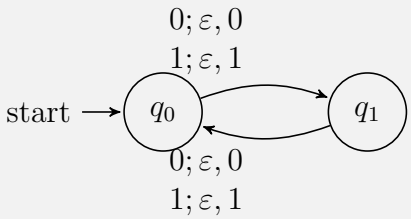
ii. $\langle 101^*, 10111\rangle \in A_{REX}$

iii. $\langle(\{S\}, \{a, b\}, \{S \to S\}, S)\rangle \in E_{CFG}$

iv. $\langle 101^*, 10 \cup 1011^*\rangle \in EQ_{REX}$

v. Define P_1, P_2 PDAs with input alphabet $\{0,1\}$ and stack alphabet $\{0,1\}$ and state diagrams:

$P_1 =$ 

$P_2 =$ 

$\langle P_1, P_2 \rangle \in EQ_{PDA}$

(b) (*Graded for completeness*) Computational problems can also be defined about Turing machines. Consider the two high-level descriptions of Turing machines below. Reverse-engineer them to define the computational problem that is being recognized, where $L(M_{DFA})$ is the language corresponding to this computational problem about DFA and $L(M_{TM})$ is the language corresponding to this computational problem about Turing machines. *Hint*: the computational problem is not acceptance, language emptiness, or language equality (but is related to one of them).

Let $s_1, s_2, \ldots$ be a list of all strings in $\{0,1\}^*$ in string (shortlex) order. Consider the following Turing machines

M_{DFA} = "On input $\langle D \rangle$ where D is a DFA :

1. for $i = 1, 2, 3, \ldots$
2. Run D on s_i
3. If it accepts, accept.
4. If it rejects, go to the next iteration of the loop"

and

M_{TM} = "On input $\langle T \rangle$ where T is a Turing machine :

1. for $i = 1, 2, 3, \ldots$
2. Run T for i steps on each input $s_1, s_2, \ldots, s_i$ in turn
3. If T has accepted any of these, accept.
4. Otherwise, go to the next iteration of the loop"

Solution: $L(M_{DFA})$ is the set of all DFA encodings such that the DFA has non-empty language. Suppose we have a DFA D whose language is non-empty, feeding

$\langle D \rangle$ to M_{DFA} would result in M_{DFA} trying all possible strings on D. Eventually, D will accept one, and M_{DFA} will also accept $\langle D \rangle$. On the other hand, if the input is not the encoding of a DFA, M_{DFA} will reject. Finally, if the input is the encoding of a DFA but the language is empty, M_{DFA} will loop forever. The same reasoning applies to M_{TM}, i.e. $L(M_{TM})$ is the set of all TM encodings such that the TM has non-empty language. We do need to watch out for infinite loops when running a TM, hence the difference in construction compared to M_{DFA}.

4. **Computational problems** (10 points): For each of the following statements, determine if it is true or false. Clearly label your choice by starting your solution with True or False and then provide a justification for your answer.

(a) (*Graded for correctness*) Prove that the language

$$\{\langle D \rangle \mid D \text{ is an NFA over } \{0,1\} \text{ and } L(D) = L(0^* \cup 1^*)\}$$

is decidable.

Solution: True. The high level description of the decider $M_{L(0^* \cup 1^*)}$ the decides the above language is as follows:

$M_{L(0^* \cup 1^*)}$ = "On input $\langle D \rangle$ where D is an NFA over $\{0,1\}$:

1. Construct DFA M_C with $L(M_C) = L(0^* \cup 1^*)$
2. Use macrostates (Theorem 1.39) to construct DFA M_D with $L(M_D) = L(D)$
3. Run a decider for EQ_{DFA} on $\langle M_C, M_D \rangle$
4. If it accepts, accept
5. If it rejects, reject"

We already know EQ_{DFA} is decidable and there's a decider that decides it (Week 7, pg. 7). Thus, once we type check, the construction in steps 1 and 2 can be done in finite time, and step 3 will also halt in finite time and tell us whether we reject or accept $\langle D \rangle$. Therefore, $M_{L(0^* \cup 1^*)}$ is a decider. Moreover, $\langle M_C, M_D \rangle \in EQ_{DFA} \iff L(0^* \cup 1^*) = L(M_C) = L(M_D) = L(D)$. Thus, $M_{L(0^* \cup 1^*)}$ accpets all and only strings in the above language.

(b) (*Graded for correctness*) Prove that the language

$$\{\langle R_1, R_2 \rangle \mid R_1, R_2 \text{ are regular expressions over } \{0,1\} \text{ and } L(R_1) \subseteq L(R_2)\}$$

is decidable.

Solution: True. We can solve this similar to how in class, we were able to create a Turing machine that decides EQ_{DFA} (Week 7, pg. 7). Let's say $L(R_1) = L_1$ and $L(R_2) = L_2$ for simplicity's sake. If $L_1 \subseteq L_2$, that just means that $L_1 \cap L_2 = L_1$. A high level description of a decider M_{SUBSET} that decides the above language is the following:

M_{SUBSET} = "On input $\langle R_1, R_2 \rangle$ where R_1, R_2 are regular expressions over $\{0, 1\}$:

1. Use recursive construction and then macrostates (Lemma 1.55 and then Theorem 1.39) to construct DFA D_1 and D_2 where $L(D_1) = L(R_1)$ and $L(D_2) = L(R_2)$
2. Use the method introduced in Week 2, pg. 8 to construct a new DFA D where $L(D) = L(D_1) \cap L(D_2)$
3. Run a decider for EQ_{DFA} on $\langle D_1, D \rangle$
4. If it accepts, accept.
5. If it rejects, reject"

Constructions in steps 1 and 2 can be done in finite time, and the decider's computation in step 3 will also halt in finite time. Therefore, M_{SUBSET} is a decider. Moreover, $\langle D_1, D \rangle \in EQ_{DFA} \iff L(R_1) = L(R_1) \cap L(R_2) \iff L(R_1) \subseteq L(R_2)$, thus M_{SUBSET} accepts all and only the strings in the given language.