HW5CSE105W25: Homework assignment 5

Sample Solutions

Due: February 27th 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 and solved.

Resources: To review the topics for this assignment, see the class material from Weeks 6, 7, and 8. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Chapters 3 and 4. 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. **Equally expressive models** (10 points): The **Church-Turing Thesis** (Sipser p. 183) says that the informal notion of algorithm is formalized completely and correctly by the formal definition of a Turing machine. In other words: all reasonably expressive models of computation are equally expressive with the standard Turing machine. In this question, we will give support for this thesis by showing that some adaptations of the standard (Chapter 3) Turing machine model still gives us a new model that is equally expressive.

(a) (*Graded for completeness*) [1] Let's define a new machine model, and call it the **May-stay** machine. The May-stay machine model is the same as the usual Turing machine model, except that on each transition, the tape head may move L, move R, or Stay.

Formally: a May-stay machine is given by the 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where Q is a finite set with $q_0 \in Q$ and $q_{accept} \in Q$ and $q_{reject} \in Q$ and $q_{accept} \neq q_{reject}$, Σ and Γ are alphabets and $\Sigma \subseteq \Gamma$ and $\square \in \Gamma$ and $\square \notin \Sigma$, and the transition function has signature

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

[1] 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.

The notions of computation and acceptance are analogous to that from Turing machines.

Prove that Turing machines and May-stay machines are equally expressive. A complete proof will use the formal definitions of the machines.

Hint: Include two directions of implications. First, let M be an arbitrary Turing machine and prove that there's a May-stay machine that recognizes the language recognized by M. Next, let M_S be an arbitrary May-stay machine and prove that there's a Turing machine that recognizes the language recognized by M_S.

Solution:

First, let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ be an arbitrary Turing machine. We define the May-stay machine $M_{S_{new}} = (Q, \Sigma, \Gamma, \delta_{new}, q_0, q_{accept}, q_{reject})$ where $\delta_{new} : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ with

$$\delta_{new}((q, x)) = \delta((q, x)) \quad \text{for all } q \in Q, x \in \Gamma$$

Though $M_{S_{new}}$ has a different codomain for the transition function, the option for the tape head to stay (S) is never used. $M_{S_{new}}$ has the same deterministic transitions as M, therefore $M_{S_{new}}$ recognizes the same language recognized by M.

Next, let $M_S = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ be an arbitrary May-stay machine. We define the Turing machine $M_{new} = (Q_{new}, \Sigma, \Gamma, \delta_{new}, (q_0, 0), (q_{accept}, 0), (q_{reject}, 0))$ where $Q_{new} = Q \times \{0, 1\}$, and $\delta_{new} : Q_{new} \times \Gamma \to Q_{new} \times \Gamma \times \{L, R\}$ with the following definition for $q \in Q$, $x \in \Gamma$, $i \in \{0, 1\}$:

$$\delta_{new}(\ ((q, i), x)\ ) = \begin{cases} ((q', 0), y, d) & \text{if } \delta((q, x)) = (q', y, d),\ d \in \{L, R\},\ i = 0 \\ ((q', 1), y, R) & \text{if } \delta((q, x)) = (q', y, S),\ i = 0 \\ ((q, 0), x, L) & \text{if } i = 1 \end{cases}$$

The idea for δ_{new} is that whenever the May-stay machine's transition function says the tape head should stay, our new Turing machine will move the tape head right then left to simulate the stay effect. Specifically, when $\delta((q, x)) = (q', y, S)$, our M_{new} simulate this step by taking two steps: $\delta((q, 0), x) = ((q', 1), y, R)$ and then $\delta((q', 1), m) = ((q', 0), m, L)$. These two steps combined will achieve the effect of overwriting x with y, the tape head ends up in the original position, and we transition from state $(q, 0)$ to $(q', 0)$. States in $Q \times \{0\}$ are on the "main path" of the May-stay machines' computations, while states in $Q \times \{1\}$ are intermediate stations for simulating the stay move. Therefore we can see any string accepted by M_S is also accepted by M_{new}, and any string accepted by M_{new} is also accepted by M_S, thus $L(M_{new}) = L(M_S)$.

(b) (*Graded for correctness*) [2] Let's define a new machine model, and call it the **Double-move** machine. The Double-move machine model is the same as the usual Turing machine model, except that on each transition, the tape head may move L, move R one cell, or move R two cells. Formally: a Double-move machine is given by the 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where Q is a finite set with $q_0 \in Q$ and $q_{accept} \in Q$ and $q_{reject} \in Q$ and $q_{accept} \neq q_{reject}$, Σ and Γ are alphabets and $\Sigma \subseteq \Gamma$ and $\square \in \Gamma$ and $\square \notin \Sigma$, and the transition function has signature

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, T\}$$

where L means that the read-write head moves to the left one cell (or stays put if it's at the leftmost cell already), R means that the read-write head moves one cell to the right , and T means that the read-write head moves two cells to the right. The notion of computation and acceptance are analogous to that from Turing machines.

Prove that Turing machines and Double-move machines are equally expressive. A complete proof will use the formal definitions of the machines.

Hint: Include two directions of implications. First, let M be an arbitrary Turing machine and prove that there's a Double-move machine that recognizes the language recognized by M. Next, let M_D be an arbitrary Double-move machine and prove that there's a Turing machine that recognizes the language recognized by M_D.

Solution:

First, let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ be an arbitrary Turing machine. Define a Double-move machine $M_{D_{new}} = (Q, \Sigma, \Gamma, \delta_{new}, q_0, q_{accept}, q_{reject})$ where $\delta_{new} : Q \times \Gamma \to Q \times \Gamma \times \{L, R, T\}$ with

$$\delta_{new}((q, x)) = \delta((q, x)) \quad \text{for all } q \in Q, x \in \Gamma$$

Though $M_{D_{new}}$ has a different codomain for the transition function, the option for read-write head moves two cells to the right (T) is never used. $M_{D_{new}}$ has the same deterministic transitions as M, therefore $M_{D_{new}}$ recognizes the same language recognized by M.

Next, let $M_D = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ be an arbitrary Double-move machine. Define the Turing machine $M_{new} = (Q_{new}, \Sigma, \Gamma, \delta_{new}, (q_0, 0), (q_{accept}, 0), (q_{reject}, 0))$ where $Q_{new} = Q \times \{0, 1\}$, and $\delta_{new} : Q_{new} \times \Gamma \to Q_{new} \times \Gamma \times \{L, R\}$ with the following definition for $q \in Q$, $x \in \Gamma$, $i \in \{0, 1\}$:

[2] 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.

$$\delta_{new}(\ ((q,i),x)\ ) = \begin{cases} ((q',0),y,d) & \text{if } \delta((q,x)) = (q',y,d),\ d \in \{L,R\},\ i = 0 \\ ((q',1),y,R) & \text{if } \delta((q,x)) = (q',y,T),\ i = 0 \\ ((q,0),x,R) & \text{if } i = 1 \end{cases}$$

The idea for δ_{new} is that whenever the Double-move machine's transition function says the tape head should move two cells to the right, our new Turing machine will move the tape head right then right again to simulate the double move effect. Specifically, when $\delta((q,x)) = (q',y,T)$, our M_{new} simulates this step by taking two steps: $\delta((q,0),x) = ((q',1),y,R)$ and then $\delta((q',1),m) = ((q',0),m,R)$. These two steps combined will achieve the effect of overwriting x with y, the tape head moves two cells to the right, and we transition from state $(q,0)$ to $(q',0)$. States in $Q \times \{0\}$ are on the "main path" of the Double-move machines' computations, while states in $Q \times \{1\}$ are intermediate stations for simulating the double move. Therefore we can see any string accepted by M_D is also accepted by M_{new}, and any string accepted by M_{new} is also accepted by M_D, thus $L(M_{new}) = L(M_D)$.

(c) (*Graded for completeness*) In your proofs of equal expressivity in the previous parts of this question, you proved that a language is recognizable by some Turing machine if and only if it is recognizable by some May-stay machine or by some Double-move machine. Do your proofs also prove that a language is decidable by some Turing machine if and only if it is decidable by some May-stay machine or by some Double-move machine? Justify your answer.

Solution: Yes, the same constructions can be used to prove that a language is decidable by some Turing machine if and only if it is decidable by some May-stay machine or by some Double-move machine.

Let's use the constructions in 1(a) to see why a language is decidable by some Turing machine if and only if it is decidable by some May-stay machine. Now, consider a language L that can be decided by a Turing machine M which is a decider. Then, we construct a May-stay machine $M_{S_{new}}$ following the method. We know that M accepts any $s_1 \in L$ in finitely many steps and rejects any $s_2 \notin L$ in finitely many steps because it is a decider. Since $M_{S_{new}}$ has the same deterministic transitions as M, we know that $M_{S_{new}}$ also accepts any $s_1 \in L$ in finitely many steps, and rejects any $s_2 \notin L$ in finitely many steps. Therefore, $L(M_{S_{new}}) = L(M) = L$ and $M_{S_{new}}$ is a decider May-stay machine, thus we proved that L can also be decided by a May-stay machine. On the other hand, consider a language L' that can be decided by a May-stay machine M_S which is a decider. Then, we construct a Turing machine M_{new} following the method. Since M_{new} translates each S transition of M_S into two transitions (R then L), and simulate each L or R transition of M_S with one transition, and since the computation of M_S on any string halts in finite steps,

the computation of M_{new} on any string also halts in finite steps. Also, any string accepted by M_S is also accepted by M_{new}, and any string rejected by M_S is also rejected by M_{new}. Therefore, $L(M_S) = L(M_{new}) = L'$, and M_{new} is a decider TM, thus we proved that L' can also be decided by a Turing machine.

The same reasoning applies to the justification of why a language is decidable by some Turing machine if and only if it is decidable by some Double-move machine, using the constructions in 1(b).

2. **Modifying machines** (12 points)

(a) (*Graded for correctness*) Suppose a friend suggests that the following construction can be used to prove that the class of decidable languages is closed under intersection.

Construction: given deciders M_1 and M_2 build the following machine M

M = "On input w :
1. Run M_1 on input w.
2. If M_1 accepts w, accept.
3. Run M_2 on input w.
4. If M_2 accepts w, accept.
5. If M_2 rejects w, reject."

Build a counterexample that could be used to convince your friend that this construction doesn't work. A complete counterexample will include (1) a high-level description of M_1, (2) a high-level description of M_2, (3) a justification for why they provide a counterexample (that references the definitions of M , decidable languages, and intersection).

Ungraded bonus: Is it possible to change one line of the construction to make it work?

Solution: As the counterexample, let's defined the following two Turing machines by giving their high-level descriptions

M_1 = "On input w :
1. If w is the empty string, accept.
2. Else, reject."

M_2 = "On input w :

1. If w is the empty string, reject.
2. Else, accept."

Notice that both M_1 and M_2 are deciders because checking whether a string has any characters can be done within finitely many steps and the conditional branch takes finitely many steps.
Also, notice that the languages of the machines are complements of each other: $L(M_1) = \{\varepsilon\}$ and $L(M_2) = \{w \in \Sigma^* \mid w \neq \varepsilon\}$. The intersection of these two sets is therefore empty (because no string is both empty and nonempty). Thus, if $L(M) = L(M_1) \cap L(M_2)$, then M should not accept any string. However, let's trace the computation of M on ε: in step 1, we run M_1 on ε, and in step 2, we accept because M_1 accepts the string. Therefore, M does not decide $L(M_1) \cap L(M_2)$ so it cannot be used to prove that the class of decidable languages is closed under intersection.

Ungraded bonus: If we were to change step 2 to "If M_1 rejects w, reject", M would work. We know that both M_1 and M_2 are deciders, so there shouldn't be any looping in the new Turing machine M since strings will be accepted or rejected in finite time, thus M is a decider. If and only if M_1 and M_2 both accepts the input string, M accepts. Otherwise, it rejects. Therefore, the modified M can decide the intersection of $L(M_1)$ and $L(M_2)$.

(b) (*Graded for correctness*) Suppose a friend suggests that the following construction can be used to prove that the class of recognizable languages is closed under intersection.
Construction: given Turing machines M_1 and M_2 build the following machine M'

M' = "On input w :

1. Run M_1 on input w.
2. If M_1 rejects w, reject.
3. Run M_2 on input w.
4. If M_2 rejects w, reject."

Build a counterexample that could be used to convince your friend that this construction doesn't work. A complete counterexample will include (1) a high-level description of M_1, (2) a high-level description of M_2, (3) a justification for why they provide a counterexample (that references the definition of M', recognizable languages, and intersection).
Ungraded bonus: Is it possible to change one line of the construction to make it work?

Solution: Our alphabet will be $\Sigma = \{0, 1\}$. To build our counterexample, we start by defining the following Turing machine, M_1, where $L(M_1) = \Sigma^*$:

$$M_1 = \text{“On input } w :$$
1. Accept.”

We also define the Turing machine M_2, which is designed so that $L(M_2) = L((0 \cup 1)^+)$ or the set of all strings except the empty string.

$$M_2 = \text{”On input } w :$$
1. If $w \neq \varepsilon$, accept.
2. Else, loop.

Note that the machine M_2 is not a decider because it does not reject ε (nor does it accept it), it loops instead.
Since $L(M_2) \subseteq L(M_1)$, all (and only) the strings in $L(M_2)$ are in both $L(M_1)$ and $L(M_2)$, hence

$$L(M_1) \cap L(M_2) = L(M_2) = \{w \in \Sigma^* \mid w \neq \varepsilon\}$$

Consider the string $w = 1$. (Any nonempty string will work for our example, we're just choosing one for definiteness.) Since $w \in L(M_1) \cap L(M_2)$, M' should accept the string in finite time. Tracing the computation of M' on $w = 1$, in step 1 we simulate the computation of M_1 on w, which accepts immediately so M''s computation continues to step 2, evaluates the conditional's condition as false, and moves to step 3. In this step, we simulate the computation of M_2 on w, which accepts since $w = 1 \neq \varepsilon$, so the execution returns to M' and in step 4 evaluates the conditional's condition as false and so never halts. In particular, since M' never explicitly accepts, M' doesn't accept $w = 1$ (even though it should have). More generally, M' can't accept any input at all, only reject it or loop on it. Thus the language of M' is always the empty language. When, like in this example, we have $L(M_1) \cap L(M_2) \neq \emptyset$ then M' does not recognize their intersection. Thus, M' cannot be used to prove that the class of recognizable languages is closed under intersection.

Ungraded bonus: We cannot alter just one line of the construction to make it work. We'd have to add an additional 5th step at the end that says “Otherwise, accept.” This way, if M_1 and M_2 both accept the string, the new Turing machine also accepts. The computation on strings that aren't in the intersection will either reject or loop. And thus M' recognizes the intersection of $L(M_1)$ and $L(M_2)$.

3. **Closure** (12 points):

For each language L over an alphabet Σ, we have the associated sets of strings (also over Σ)

$$L^* = \{w_1 \cdots w_k \mid k \geq 0 \text{ and each } w_i \in L\}$$

and

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

and

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

(a) (*Graded for correctness*) Prove whether this Turing machine construction below **can** or **cannot** be used to prove that the class of recognizable languages over Σ is closed under the Kleene star operation or the $SUBSTRING$ operation or the $EXTEND$ operation.

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

M_a = "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 $s_j w s_k$ for at most n steps
5. If that computation halts and accepts within n steps, accept.
6. Otherwise, continue with the next iteration of this inner loop"

A complete and correct answer will either identify which operation works and give the proof of correctness why, for any Turing machine M, $L(M_a)$ is equal to the result of applying this operation to $L(M)$; **or** give a counterexample (a recognizable set A and a Turing machine M recognizing A and a description of why $L(M_a)$ where M_a is the result of the construction applied to M doesn't equal A^* and doesn't equal $SUBSTRING(A)$ and doesn't equal $EXTEND(A)$.

Solution: The operation $SUBSTRING$ works.

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_a defined above. We will show that $L(M_a) = SUBSTRING(L_1)$.

$\subseteq$ Let $w \in L(M_a)$, i.e. M_a accepts the string w. The only way for M_α to accept is in step 5. Tracing the definition of M_a, arriving in step 5 means there exist some strings s_j, s_k and some integer n such that $s_j w s_k$ is accepted by M_1 within n steps. 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 $a, b \in \Sigma^*$ 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_a 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_a will accept w.

(b) (*Graded for correctness*) Prove whether this Turing machine construction below **can** or **cannot** be used to prove that the class of recognizable languages over Σ is closed under the Kleene star operation or the $SUBSTRING$ operation or the $EXTEND$ operation.

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

M_b = "On input w :

1. For $n = 1, 2, \ldots$
2. For $j = 0, \ldots |w|$
3. Let u be the string consisting of the first j characters of w
4. Run the computation of M on u for at most n steps
5. If that computation halts and accepts within n steps, accept.
6. Otherwise, continue with the next iteration of this inner loop"

A complete and correct answer will either identify which operation works and give the proof of correctness why, for any Turing machine M, $L(M_b)$ is equal to the result of applying this operation to $L(M)$; **or** give a counterexample (a recognizable set B and a Turing machine M recognizing B and a description of why $L(M_b)$ where M_b is the result of the construction applied to M doesn't equal equal B^* and doesn't equal $SUBSTRING(B)$ and doesn't equal $EXTEND(B)$.

Solution: The operation $EXTEND$ works.

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_b defined above. We will show that $L(M_b) = EXTEND(L_1)$.

- $\subseteq$ Let $w \in L(M_b)$, i.e. M_b accepts the string w. The only way for M_b to accept is in step 5. Tracing the definition of M_b, arriving in step 5 means there exists some u that is a prefix of w accepted by M_1. This means $u \in L(M_1) = L_1$, so $w \in EXTEND(L_1)$ by definition of $EXTEND$.
- $\supseteq$ Consider $w \in EXTEND(L_1)$. By definition of $EXTEND$, there exists $a, b \in \Sigma^*$ such that $w = ab$, $a \in L_1$, and $b \in \Sigma^*$. Moreover, let's say the length of a

is J, and the computation of M_1 on a takes N steps to get to the accept state. When $n = N$ and $j = J$, the nested for loop in lines 1 and 2 of the definition of M_b has an iteration where $a = u$ is considered and in step 4 feeds a 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_b accepts w.

4. **Computational problems** (8 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 three 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:

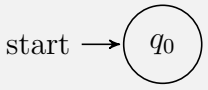
i. Let N be an NFA over $\{0, 1\}$ that accepts every string, namely, define

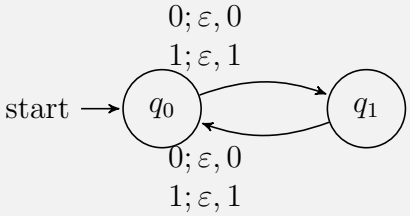
$$N = (\{q_0\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

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

iii. 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 for 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}.

5. **Computational problems** (8 points):

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

$\{\langle D \rangle \mid D \text{ is an NFA over } \{0,1\} \text{ and } D \text{ accepts at least 3 strings of length less than 5 } \}$

is decidable.

Solution: Let $s_1, s_2, \ldots$ be a list of all strings in $\{0,1\}^*$ in string (shortlex) order. The high level description of a decider that decides the above language is as follows:

M_1 = "On input $\langle D \rangle$ where D is an NFA over $\{0,1\}$:

1. Use macrostates (Theorem 1.39) to construct DFA M_D with $L(M_D) = L(D)$
2. Initialize the variable $numAcc = 0$, which we will use to record the number of accepted strings of length less than 5,
3. For $i = 1, 2, 3, \ldots, 31$
4. Run M_D on s_i. If it accepts, $numAcc$ += 1
5. If $numAcc \geq 3$, accept
6. Else, reject"

Once we type check, the construction in step 1 can be done in finite time. The for loop in steps 3 and 4 can also finish in finite time, since only the first $2^0 + 2^1 + 2^2 + 2^3 + 2^4 = 31$ strings over $\{0,1\}$ are having length less than 5, and the computation of M_D on each string also takes finitely many steps. Therefore, M_1 never loops on any input string and thus it is a decider.

If the input string is in the language, then $numAcc \geq 3$ and M_1 accepts the input string in step 5. If the input string is not in the language, there are two cases: (1) If the input string doesn't type check, M_1 rejects it directly after type checking. (2) If

the input string has the correct type but D does not accept at least 3 strings of length less than 5, then $numAcc < 3$ and M_1 rejects the input string in step 6. Therefore, M_1 accepts all and only the strings in the given language and reject all other strings. Thus we proved that the language is decidable.

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

$\{\langle R \rangle \mid R$ is a regular expression over $\{0,1\}$ and $L(R)$ has infinitely many strings starting with $0\}$

is decidable.

Solution: The high level description of a decider that decides the above language is as follows:

M_2 = "On input $\langle R \rangle$ where R is a regular expression over $\{0,1\}$:

1. Use recursive construction and then macrostates (Lemma 1.55 and then Theorem 1.39) to construct DFA $D = (Q, \Sigma, \delta, q_0, F)$ where $L(D) = L(R)$
2. Denote $u = \delta((q_0, 0))$. For each state $v \in F$:
3. Use DFS to get the set of all paths from u to v, denote as S
4. For each path $s \in S$ where $s = s_1, s_2, ..., s_k$ where $k \geq 1$, $s_1 = u$, $s_k = v$:
5. For $i = 1, 2, ..., k$:
6. If s_i has a self loop, accept the input string. Otherwise,
7. For $j = i + 1, ..., k$:
8. If s_j can reach s_i (check by DFS/BFS), accept the input string.
9. Reject the input string if we did not accept it in the above steps."

Once we type check, the construction in step 1 can be done in finite time. Since a DFA has finitely many states, each path between two states are finitely long, and running DFS/BFS and checking for self loop can all be done in finite time, the loops in steps 2-8 can terminate in finite time. Thus M_2 can always halt in finite time and it is a decider.

The algorithm's idea is after type checking, convert the regular expression R into an equivalent DFA D where $L(R) = L(D)$. From D's start state, read a 0 and land in state u. Then, if and only if u is reachable to an accept state, and on one path from u to an accept state there exists a loop, we know that D accepts infinitely many strings starting with 0, at which point we know that $\langle R \rangle$ is in the language and accept $\langle R \rangle$. If the input is not in the language, M_2 rejects it. Thus we proved that the language is decidable.