HW6 : Computational Problems, Recognizability, Decidability **Sample Solutions**

CSE105Sp23

Assigned questions

1. **Explicit encodings** (8 points):
   In a computational problem, the elements of the language are encodings of machines. For example, consider the language

   $$E_{\text{DFA}} := \{\langle M\rangle \mid M \text{ is a DFA, and } L(M) = \varnothing\}$$

   where each string $\langle M\rangle$ in the language encodes a DFA $M = (Q, \Sigma, \delta, q_0, F)$. Usually, we purposefully drop the details about how this encoding is done because they can distract from the central computational properties of the language. In fact, any encoding can be used so long as there exists a decider for syntactic questions about the DFAs being encoded. In this question, we will build some specific explicit examples of encodings of DFAs to get more comfortable with these ideas.

   (a) (*Graded for completeness*) [1] *Encoding with delimiters:* Perhaps the most straightforward way to create an encoding is to have it mirror the structure of the tuple $(Q, \Sigma, \delta, q_0, F)$ for the DFA. Your task: describe an encoding that maps each DFA M to a distinct string $\langle M\rangle$ that uniquely identifies M. That is, if you "decode" the encoding, you get the exact same machine back.

   Hints, tips, notes of caution:

   - You may use special characters like # and $ as delimiters in your encoding to separate the various components.
   - Your encoding alphabet must be finite

[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 ask that you 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.

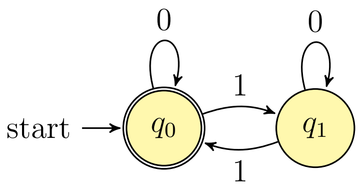
Solution: We can create an encoding using the finite alphabet $\{0, 1, \times, \rightarrow, \$, \#\}$ to map any DFA M to a distinct string $\langle M \rangle$ that uniquely identifies M by listing each element of the 5-tuple $(Q, \Sigma, \delta, q_0, F)$, each encoded as described below and separated by the special delimiting character #. We encode each of the elements of the five tuple necessary to describe a DFA as follows:

- Q: We can encode the information we need about the set of states by first giving the number of states, $|Q|$, encoded in binary. *Notice that we are not encoding the specific labels of the states, just the underlying graph structure.*
- Σ: Similarly, we can encode the number of symbols in Σ in binary.
- δ: To describe our transition function, we can list all combinations of a state, a symbol, and its resulting state separated by another delimiting character \$. For each individual combination, we can encode a unique identification for the states and symbol as their position in a list of all of the states or symbols, and then encode this in binary. The input state's binary encoding can be followed by the delimiting character *times*, then the binary encoding of the input symbol, followed by a $\rightarrow$, and finally the binary encoding of the output state.
- q_0: We can encode the start state's identification as described above in binary.
- F: We can list the encodings of all accept states' identification in binary as described above, separated by \$ characters.

Using this procedure, a string encoding of a DFA will take the following form:

[number of states] # [number of symbols] # [state id] $\times$ [symbol id] $\rightarrow$ [state id] \$ … # [state id] # [state id] \$ …

(b) (*Graded for completeness*) Use your encoding from part (a) to produce the string encoding the DFA below:



Solution: First, we describe the DFA as shown in its 5-tuple form:

$$(\{q_0, q_1\}, \{0, 1\}, \{\delta(q_0, 0) = q_0, \delta(q_0, 1) = q_1, \delta(q_1, 0) = q_1, \delta(q_1, 1) = q_0\}, q_0, \{q_0\})$$

Now, we can translate this more easily using our encoding for each part:

- Q: We encode $\{q_0, q_1\}$ as the size of Q, which is 2, in binary as 10
- Σ: Similarly, we encode the number of symbols in $\{0, 1\}$ in binary as 10.

δ: We assign each state in $\{q_0, q_1\}$ a unique identifier in binary: q_0 as 0 and q_1 as 1. Similarly, the symbols in the alphabet are represented by unique identifiers in binary, which in this case happen to be themselves. Now, we translate each part of our transition function according to our encoding scheme as follows:

A. $\delta(q_0, 0) = q_0$ is encoded as $0 \times 0 \to 0$

B. $\delta(q_0, 1) = q_1$ is encoded as $0 \times 1 \to 1$

C. $\delta(q_1, 0) = q_1$ is encoded as $1 \times 0 \to 1$

D. $\delta(q_1, 1) = q_0$ is encoded as $1 \times 1 \to 0$

We separate these with the delimiter \$ to produce the string: $0 \times 0 \to 0\$0 \times 1 \to 1\$1 \times 0 \to 1\$1 \times 1 \to 0$

q_0: We can encode the start state q_0's identification in binary as 0.

F: We can list the encoding of the only accept state q_0's identification in binary as 0.

Our complete encoding is the concatenation of the individual encodings separated by the delimiter #:

$$10\#10\#0 \times 0 \to 0\$0 \times 1 \to 1\$1 \times 0 \to 1\$1 \times 1 \to 0\#0\#0$$

(c) (*Graded for completeness*) Show that it is possible to have the same kind of delimited encoding without using special delimiter characters. In particular, prove that for every DFA M, we can assume that $\langle M \rangle \subseteq \{0, 1\}^*$.

Solution: We can create an encoding that represents the same information as above, all of the elements in a 5-tuple describing a DFA, using only the characters 0 and 1. At a high-level, we can represent numerical values using 1s, use 00 as our delimiter between the five major components, and 0 as a delimiter as necessary within these components. This essentially replaces the delimiting character # used above between the elements of the 5-tuple with 00, replaces other delimiting characters within components with 0, and replaces binary representations of numerical values with that number of 1s.

Q: The number of states is represented as that number of 1s. For example, if the DFA has 5 states, it would be represented as 11111.

Σ: Same as above for the number of symbols.

δ: In the prior encoding, we used a different delimiter character \$ to separate the parts of the transition function, and the special characters $\times$ and $\to$ in each of these parts. However, we don't actually need these different characters to differentiate between delimiters, because we already know exactly how many parts of the transition function there will be (the product of the number of states and the number of symbols), and that there are exactly three numerical values (which we can represent with 1s) we need to represent each part. Therefore, we can use the same 0 delimiter in place of all three of these special characters, and still be able to reconstruct the information exactly.

q_0: We can represent the start state's numerical identifier with that number of 1s.

F: Finally, we can list the IDs of the accept states using 1s instead of in binary form, and 0s used as delimiters between them.

Challenge; not graded:
For the delimited encoding schemes above, there are strings over the encoding alphabet (Σ) that nevertheless do not correspond to a valid DFA. Prove/disprove: There exists an encoding scheme for which this is not true; that is,

$$\{\langle M\rangle \mid M \text{ is a DFA}\} = \Sigma^*.$$

Solution: The delimited encoding scheme does not have this property, but there are encoding schemes that do. For example, take any encoding scheme over the alphabet Σ, and sort all encodings of DFA using that scheme in string order $(e_1, e_2, \ldots)$. Also consider the strings over Σ in string order $(s_1, s_2, \ldots)$. The new encoding scheme is as follows: string s_i corresponds to DFA encoded by e_i. We want to prove that producing the encoding e_i from s_i takes finite time. To see this, notice that from input s_i, you can compute i itself. Iterating over all strings in string order, keep a counter of the number of strings you encouter that are also valid encodings (recall that testing if an encoding is valid is decidable). Once you get to the ith valid encoding, that is your encoding corresponding to s_i. Since each s_i corresponds to a valid encoding and $\cup_i s_i = \Sigma^*$, we have proved the statement.

2. **Closure** (18 points):
Let $\Sigma = \{0,1\}$ and $\Gamma = \{0,1,2\}$. Recall the functions

$$\begin{aligned}\textsc{Substring}(K) &:= \{w \in \Gamma^* \mid \text{there exist } a, b \in \Gamma^* \text{ such that } awb \in K\}\\ \textsc{Rep}(L) &:= \{w \in \Gamma^* \mid \text{between every pair of successive 2s in } w \text{ is a string in } L\}\\ &= \{w \in \Gamma^* \mid \text{for all } v \in \Sigma^* \text{ if } 2v2 \in \textsc{Substring}(\{w\}), \text{ then } v \in L\}\end{aligned}$$

(a) (*Graded for correctness*) [2] Prove that, given any deterministic decider over Σ, M_L, there is a deterministic decider over Γ that recognizes

$$\textsc{Rep}(\ L(M_L)\ )$$

In other words, you will prove that for any Turing-decidable language L over Σ, $\textsc{Rep}(L)$ is also Turing-decidable. A complete answer will include both a precise construction of the machine and a (brief) justification of why this machine works as required.

[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.

Solution: Let L be a language over Σ and suppose we have M_L, a deterministic decider over Σ that recognizes L. We will use it as a subroutine in a new machine to check whether the string between every successive pair of 2's is in $L(M_L)$. Specifically, define the Turing machine M_1 over Γ:

$M_1 =$ "On input $w \in \Gamma^*$:

1. If w has at most one 2 in it, accept. Otherwise, decompose input w as $w = w_1 2 s_1 2 s_2 2 \cdots 2 s_k 2 w_2$ where $w_1, w_2, s_1, \ldots, s_k \in \{0,1\}^*$.
2. Run M_L on each of the strings $s_1, s_2, \ldots, s_k$ in turn.
3. If M_L rejects any these strings, then reject; otherwise, accept."

We claim that $L(M_1) = \text{REP}(\ L(M_L)\ )$: by definition, if w is in $\text{REP}(\ L(M_L)\ )$ then between each pair of successive 2's is a string in $L(M_L)$. In this case, the computation of M_L on each of these strings (halts and) accepts. Thus, running M_1 on w will either accept in step 1 (if w has at most one 2 in it) or will accept in step 3 (because each of the subroutines in step 2 halt and accept). Conversely, suppose w is not in $\text{REP}(\ L(M_L)\ )$. This means that there is a pair of successive 2s for which the string of 0s and 1s between them (let's call it x) is not in $L(M_L)$. In particular, since M_L is a decider, this means that M_L rejects x. Running M_1 on w, in step 1 w is decomposed into $w = w_1 2 s_1 2 s_2 2 \cdots 2 s_k 2 w_2$ with $x = s_j$ for some j. In step 2, the computations of M_L on each of the s_i are run in turn. All of these computations halt (because M_L is a decider) and at least one of these computations reject (namely the computation of M on $x = s_j$). Thus, in step 3, M_1 rejects w, as required.

(b) (*Graded for correctness*) Prove that, given any nondeterministic Turing machine over Γ, N_L, there is a nondeterministic Turing machine over Γ that recognizes

$$\text{SUBSTRING}(\ L(N_L)\ )$$

In other words, you will prove that the class of Turing-recognizable languages over Γ is closed under the SUBSTRING operation. A complete answer will include both a precise construction of the machine and a (brief) justification of why this machine works as required.

Solution: *Informally:* We will guess the strings $a, b \in \Gamma^*$ and then check whether $awb \in L(N_L)$. By "guess", we mean nondeterministically choose some string of finite length over Γ^*. It is important that all possible strings in Γ^* can be guessed.

More formally: Given a nondeterministic Turing machine N_L over Γ, we define the nondeterministic Turing machine

$M_2 =$ "On input $w \in \Gamma^*$

1. Guess strings $a \in \Gamma^*$ and $b \in \Gamma^*$
2. Run N_L on awb. If N_L accepts, accept."

We claim that $L(M_2) = \text{SUBSTRING}(\ L(N_L)\ )$: If $w \in \text{SUBSTRING}(L(N_L))$, then (by definition) there exists strings a and b such that $awb \in L(N_L)$. There is some computation of M_2 where these witness a and b are the strings chosen in step 1. and so we accept the string w in step 2 of this computation (because N_L recognizes $L(N_L)$). Since a nondeterministic Turing machine accepts a string when there's at least one accepting computation of the machine on the string, M_2 accepts w. Conversely, if $w \notin \textsc{Substring}(L(N_L))$, there is no a and b which lead to $awb \in L(N_L)$ and so every computation of M_2 on input w (no matter the nondeterministic guess in step 1) does not accept and hence M_2 does not accept w.

(c) (*Graded for completeness*) Give a different proof that the class of Turing-recognizable languages over Γ is closed under the $\textsc{Substring}$ operation, this time using only deterministic Turing machines. A complete answer will include both a precise construction of the machine and a (brief) justification of why this machine works as required.

Solution: *Informally:* We want to simulate a guess for the strings a and b that we used in part (b) using a TM that is deterministic. We will use the dovetailing trick we've seen a few times.

Formally: Let M_L be a deterministic Turing machine over the alphabet Γ. We will build a deterministic Turing machine that recognizes $\textsc{Substring}(L(M_L))$. To do this, we use an ordering on the set of all pairs of strings over Γ:

$$p_1 = (a_1, b_1), p_2 = (a_2, b_2) \dots$$

which exists because the Cartesian product of countably infinite sets is countably infinite.

We define M_3 = "On input $w \in \Gamma^*$

```
1. For $i = 1, 2, \dots$
    2. For $j = 1, 2, \dots i$
        3. Run the computation of $M_L$ on $a_j w b_j$ for at most $i$ steps.
    4. If $M_L$ ever accepts during these simulated computations, accept ; otherwise, go to next $i$"
```

Notice that if there exists some $a, b \in \Gamma^*$ such that $awb \in L(M_L)$, then there is some J for which $a = a_J$ and $b = b_J$ and there is some number of steps K for which the computation of M_L on awb halts and accepts within K steps. Thus, in the iteration of the loop on step 2 when $i = \max(J, K)$, for $j = J$, running the computation of M_L on $a_J w b_J$ for at most i steps will witness that M_L accepts $a_J w b_J$ and M_3 will accept w. Otherwise, if there is no such pair, M_L will loop forever on input w.

3. **Computational problems** (24 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 brief (3-4 sentences or so) justification for your answer.

(a) (*Graded for correctness*) For each regular language K, the language

$$\{\langle M\rangle \mid M \text{ is a DFA and } L(M) = K\}$$

is decidable.

Solution: True.
Let K be a regular language.
By definition, this means there exists a DFA N such that $L(N) = K$.

Moreover, since EQ_{DFA} is a decidable language, there exists a Turing machine that decides EQ_{DFA}. Let's call that machine M_{EQ}.

Define $R =$ "On input w,
1. Type check if $w = \langle M\rangle$ is a valid encoding of a DFA. If not, reject.
2. Run M_{EQ} on input $\langle N, M\rangle$. If accepted, accept. Else, reject."

Since M_{EQ} always halts, machine R always halts. Moreover, $L(R) = \{\langle M\rangle \mid M \text{ is a DFA and } L(M) = K\}$, as required.

(b) (*Graded for correctness*) For each regular language L, the language

$$\{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are both DFA and } L(M_1) \subseteq L \text{ and } L(M_2) \subseteq \overline{L}\}$$

is decidable.

Solution: True.

Informally: We use the set identity that $X \subseteq Y$ iff $X \cup Y = Y$. *More formally:* Let L be a regular language. By definition there exists a DFA A such that $L(A) = L$ and from the construction proving the class of regular languages is closed under complementation, we can also build DFA B with $L(B) = \overline{L}$ (by keeping the same state diagram as A but flipping the status of accept / non-accept states).

Moreover, since EQ_{DFA} is a decidable language there exists a Turing machine such that it decides on EQ_{DFA}. Let's call it M_{EQ}.

Define $S =$ "On input w,
1. Type check if $w = \langle M_1, M_2\rangle$ is a valid encoding of an ordered pair of DFAs. If not, reject.
2. Construct DFA D_1 such that $L(D_1)$ is the union of $L(A)$ and $L(M_1)$ (using Chapter 1 Cartesian product construction).
3. Run M_{EQ} on input $\langle D_1, A\rangle$. If rejected, reject.
4. Construct DFA D_2 such that $L(D_2)$ is the union of $L(B)$ and $L(M_2)$.
5. Run M_{EQ} on $\langle D_2, B\rangle$. If rejected, reject. Else accept."

On any input w, the computation of S halts because the type check takes finitely many steps, the construction of D_1 takes finitely many steps, M_{EQ} is a decider so its computation on $\langle D_1, A\rangle$ takes finitely many steps, the construction of D_2 takes finitely many steps, and the computation of M_{EQ} on $\langle Y, B\rangle$ takes finitely many steps. Moreover, $L(S) = \{\langle M_1, M_2\rangle \mid M_1, M_2$ are both DFA and $L(M_1) \subseteq$

L and $L(M_2) \subseteq \overline{L}\}$, using the set identity mentioned above (omitting some of the details of tracing through the bidirectional set equality proof).

(c) (*Graded for correctness*) Let Model $\in \{\text{DFA}, \text{NFA}, \text{REX}, \text{CFG}, \text{PDA}\}$. If EQ_{Model} is decidable, then E_{Model} is decidable.

Solution: True.
For any of the Models in the set, suppose we assume EQ_{Model} is decidable. Thus, there exists a Turing Machine M_{EQ} that decides EQ_{Model}.

Since the empty language is a finite language, it is regular, and hence also recognizable by a NFA, describable by a regular expression, generated by some CFG, and recognizable by some PDA (see results in Chapter 1 and beginning of Chapter 2). Thus, there is an instance X in the Model such that $L(X) = \emptyset$.

Define T = "On input w,
1. Type check if $w = \langle M \rangle$ is a valid encoding of Model.
2. Run M_{EQ} on input $\langle X, M \rangle$. If accepted, accept. Else reject."

Since M_{EQ} always halts, machine T always halts and $L(T) = E_{\text{Model}}$.
Thus, if EQ_{Model} is decidable, then E_{Model} is decidable.

Challenge; not graded: Let Model $\in \{\text{DFA}, \text{NFA}, \text{REX}, \text{CFG}, \text{PDA}\}$. *If A_{Model} is decidable, then EQ_{Model} is decidable.*

Solution: False.
A counterexample to this example is the model CFG. Theorem 4.7 in the book says A_{CFG} is decidable (Theorem 4.7). However, Page 200 in the book says that EQ_{CFG} is not decidable. Thus, the statement is false.