

Solutions for cs154 Homework #4

Easy Problems

Part (a) Let $EQ_{DFA/Regex} = \{\langle DFA \rangle, \langle Regex \rangle \mid L(DFA) = L(Regex)\}$ where $\langle DFA \rangle$ encodes a DFA and $\langle Regex \rangle$ encodes a regular expression. In one of the lectures (and also in Sipser), we saw an encoding for TM's. The encodings for $\langle DFA \rangle$ and $\langle Regex \rangle$ are along the same lines.

$EQ_{DFA/Regex}$ is decidable because there is a TM (program) that *halts on all inputs* and accepts (halts in q_{accept}) exactly for those strings that have the form $\langle DFA \rangle, \langle Regex \rangle$ and where $L(DFA) = L(Regex)$.

A TM that recognizes $EQ_{DFA/Regex}$ works as follows. First, it performs syntax checking on the input. If the input violates the rules for encoding $\langle DFA \rangle$ or $\langle Regex \rangle$, the TM halts in q_{reject} . Note that we have not formally proved that such a syntax checker exists. Our *proof*, or rather assertion, is by *hand waving*. You might want to convince yourself that such a syntax checker indeed is easy to create.

If the input has the form $\langle DFA \rangle, \langle Regex \rangle$, then the TM converts $\langle Regex \rangle$ into an NFA, which it converts into a DFA DFA' . To check whether DFA and DFA' are equivalent, it creates $DFA_{new} = DFA \cap \overline{DFA'}$. If $L(DFA_{new}) = \{\}$, then the TM halts in q_{accept} . Otherwise, it halts in q_{reject} . Note that each of the steps $Regex \rightarrow NFA \rightarrow DFA' \rightarrow \overline{DFA'}$ is a procedure that always halts in finite time. The same holds for creating DFA_{new} and checking whether $L(DFA_{new}) = \{\}$.

Part (b) Proof by contradiction. Let us assume that there exists a one-to-one mapping $f : \mathcal{N} \rightarrow \mathcal{B}$. For sequence $b \in \mathcal{B}$ and $i \geq 1$, let $b[i]$ denote the i^{th} member of b . Consider the sequence $d = \langle f(1)[1], f(2)[2], f(3)[3], \dots \rangle$. Sequence d consists of bits on the diagonal corresponding to the mapping f . Now consider \bar{d} which is defined as $\langle 1 - d[1], 1 - d[2], 1 - d[3], \dots \rangle$. It is clear that $\bar{d} \in \mathcal{B}$. However, there exists no $n \in \mathcal{N}$ such that $f(n) = \bar{d}$. This is because for any $n \in \mathcal{N}$, \bar{d} differs from $f(n)$ in the bit that lies on the diagonal. This means that f does not exist. Therefore \mathcal{B} is uncountable.

Part (c) We know that $\mathcal{N} \times \mathcal{N} \times \mathcal{N}$ is countable. We also know that if $S \subseteq C$ where S is infinite and C is a countable set, then S is also countable. We are given $T = \{(i, j, k) \mid i \leq j \leq k \text{ and } i, j, k \in \mathcal{N}\}$. Now T is infinite and $T \subseteq \mathcal{N} \times \mathcal{N} \times \mathcal{N}$. Therefore, T is countable.

Part (d) We will reduce ALL_{CFG} to EQ_{CFG} . Here are the definitions of the two languages:

$$EQ_{CFG} = \{\langle CFG_1, CFG_2 \rangle \mid L(CFG_1) = L(CFG_2)\}$$

$$ALL_{CFG} = \{\langle CFG \rangle \mid L(CFG) = \Sigma^*\}$$

ALL_{CFG} is shown to be undecidable in Sipser.

The reduction $ALL_{CFG} \rightarrow EQ_{CFG}$ is straightforward. We convert the string $\langle CFG \rangle$ into the string $\langle CFG, CFG_for_Sigma_*\rangle$, where $CFG_for_Sigma_*$ denotes a CFG that generates Σ^* (such a CFG is easy to create). If EQ_{CFG} is decidable, then the reduction implies that ALL_{CFG} is decidable, a contradiction. Therefore, EQ_{CFG} must be undecidable.

Part (e) $EQ_{CFG} = \{\langle CFG_1, CFG_2 \rangle \mid L(CFG_1) = L(CFG_2)\}$.

EQ_{CFG} would be the complement of a TM-recognizable language if and only if $\overline{EQ_{CFG}}$ is a TM-recognizable language. Now, $\overline{EQ_{CFG}} = \{w \mid w \text{ does not have the form } \langle CFG_1, CFG_2 \rangle\} \cup \{\langle CFG_1, CFG_2 \rangle \mid L(CFG_1) \neq L(CFG_2)\}$. We claim that the first set (consisting of strings that violate the syntax for encoding $\langle CFG_1, CFG_2 \rangle$) is easy to recognize. We do not provide a formal proof though. The second set, namely $\{\langle CFG_1, CFG_2 \rangle \mid L(CFG_1) \neq L(CFG_2)\}$, can be recognized as follows. We convert both CFG_1 and CFG_2 into Chomsky Normal Form. Then we start enumerating strings in Σ lexicographically (where Σ is the alphabet used by CFG_1 and CFG_2). This means that we start producing strings in this sequence: **a, b, c, ..., aa, ab, ac, ..., aaa, aab, aac, ...**, where $\Sigma = \{a, b, c, \dots\}$. For each string w , we check whether it can be generated by CFG_1 and by CFG_2 . If both or none of the CFGs can generate w , the TM moves on to produce the next string in lexicographic sequence. Otherwise, exactly one of the CFG's generates the string; the TM halts in q_{accept} . Why did we convert the CFGs into Chomsky Normal Form? The reason is that there is a procedure that always halts for checking whether a CFG in Chomsky Normal Form can generate a particular string w or not.

Problem 1 (10 Points)

Let $A = \{\langle R, S \rangle \mid R \text{ and } S \text{ are regular expressions and } L(R) \subseteq L(S)\}$. Show that A is decidable.

Key Idea: To test $L(R) \subseteq L(S)$, we need to figure out if $L(R) - L(S)$ is empty or not. This is equivalent to the question $L(R) \cap \overline{L(S)} = \{\}$? We can determine the answer by constructing $DFA(R) \cap \overline{DFA(S)}$, where $DFA(R)$ and $DFA(S)$ correspond to DFAs for regular expressions R and S respectively. Each of the following constructions: $DFA(R)$, $DFA(S)$, $\overline{DFA(S)}$, $DFA(R) \cap \overline{DFA(S)}$ and checking whether $L(DFA(R) \cap \overline{DFA(S)}) = \{\}$ requires procedures that always halt.

There is a detail we need to take care of: A TM that accepts language A should halt *on all inputs*. This means that the TM should halt in q_{reject} for all strings in \overline{A} . Now $\overline{A} = \{w \mid w \text{ does not have the right format}\} \cup \{\langle R, S \rangle \mid R \text{ and } S \text{ are regular expressions and } L(R) \supset L(S)\}$. Strings that do not have the right format are easy to identify (we do not present a formal proof here). The second set of strings lead the TM to q_{reject} , as outlined earlier.

Problem 2 (10 Points)

Show that $\{\langle G \rangle \mid G \text{ is a CFG over } \{0, 1\}^* \text{ and } 1^* \subseteq L(G)\}$ is a decidable language.

Using the construction in Problem 3, it is possible to construct a CFG for the language $L(G) \cap 1^*$. This language is context-free. Moreover, its alphabet consists of only one symbol: 1. It can be shown that a CFL over a unary alphabet is regular. And it is possible to convert the CFG for $L(G) \cap 1^*$ into a DFA that accepts exactly the same language. We then simply check whether the DFA accepts 1^* or not.

Alternate solution: Let p be the constant in the Pumping Lemma for this CFG. Test whether all strings of the form 1^k are generated by the CFG where $0 \leq k \leq p! + p$. If yes, then $1^* \subseteq L(CFG)$, otherwise not.

Proof by induction.

Base Case: We have already ensured that all strings of the form 1^k belong to $L(CFG)$ where $0 \leq k \leq p! + p$.

Induction Step: Let $n > p! + p$. Assuming that all strings of the form 1^k are generated by CFG where $0 \leq k < n$, we will show that 1^n can also be generated by the CFG.

Claim: the string $1^{n-p!}$ can be pumped to generate 1^n .

Proof of claim: By induction hypothesis, we know that $1^{n-p!}$ belongs to the language. Since $n > p! + p$, the length of $1^{n-p!}$ is at least p . Therefore, Pumping Lemma can be applied. This means that there is some $\ell \in [1, p]$ such that $\forall x \geq 0 : 1^{n-p!+x\ell}$ also belongs to the language. Now there exists an x such that $p! = x\ell$. This is because $\ell \in [1, p]$ and therefore, divides $p!$. This means 1^n belongs to the language.

Problem 3 (10 Points)

Let $E = \{\langle M \rangle \mid M \text{ is a DFA that accepts some string of the form } ww^R \text{ for } w \in \{0, 1\}^*\}$. Show that E is decidable.

Key Idea: Let C denote a CFG such that $L(C) = \{ww^R \mid w \in \{0, 1\}^*\}$. Then $L(C) \cap L(M)$ is a CFL. We need to test whether $L(C) \cap L(M) = \{\}$ or not. Testing a CFL for emptiness is decidable (Sipser).

Claim: If M is a DFA and C is a CFG, then $L(C) \cap L(M)$ is a Context Free Language (CFL). In fact, it is possible to construct a CFG C' such that $L(C') = L(C) \cap L(M)$.

Proof: Convert CFG C into PDA P . We can construct a PDA P' such that $L(P') = L(P) \cap L(M)$. The key idea is to run P and M *in parallel* and to accept only if *both* of them are in their respective final states. Formally, let $P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, F_P)$. Let $M = (Q_M, \Sigma, \delta_M, q_M, F_M)$. We will create $P' = (Q_{P'}, \Sigma, \Gamma, \delta_{P'}, q_{P'}, F_{P'})$ as follows. Let $Q_{P'} = Q_P \times Q_M$. The initial state is $[q_P, q_M]$ where q_P and q_M are the initial states of P and M respectively. The new set of final states is $F_{P'} = F_P \times F_M = \{[p, m] \mid p \in F_P \text{ and } m \in F_M\}$. The stack alphabet Γ remains the same. The new transition function is $\delta_{P'}([p, m], a, \gamma) = \{([p', m'], \gamma') \mid ((p', \gamma')) \in \delta_P(p, a, \gamma) \text{ and } \delta(m, a) = m'\}$. The new PDA P' accepts exactly the language $L(P) \cap L(M)$. We can now convert P' into a CFG C' such that $L(P') = L(C')$. ⊙

The above construction is useful in solving a few other problems as well.

A TM that accepts E works as follows. It first checks whether the input has the right format or not. Constructing such a syntax checker is easy. Strings with incorrect format are rejected. The TM then creates a CFG C such that $L(C) = \{ww^R \mid w \in \{0, 1\}^*\}$. Then it creates a CFG C' such that $L(C') = L(C) \cap L(M)$, using the construction outlined above. Finally, it tests whether $L(C') = \{\}$ or not. Each of these steps requires a procedure that is guaranteed to halt.

Problem 4 (10 Points)

Let $J = \{w \mid w = 0x \text{ for some } x \in A_{TM} \text{ or } w = 1y \text{ for some } y \in \overline{A_{TM}}\}$.

Show that neither J nor \overline{J} is Turing-recognizable.

We reduce $\overline{A_{TM}}$ to J . We know that $\overline{A_{TM}}$ is not Turing-recognizable. The mapping function is simple: when the input string is w , A_{TM} simply feeds $1w$ to J and accepts if and only if J accepts. If J is Turing-recognizable, then A_{TM} is Turing-recognizable, a contradiction. Therefore, J is not Turing-recognizable.

$\overline{J} = \{w \mid w = 1x \text{ for some } x \in A_{TM} \text{ or } w = 0y \text{ for some } y \in \overline{A_{TM}}\} \cup (\Sigma^* - (0+1)\Sigma^*)$. The regular expression at the end consists of all strings that do not begin with 0 or 1.

We reduce $\overline{A_{TM}}$ to \overline{J} . We know that $\overline{A_{TM}}$ is not Turing-recognizable. The mapping function is simple: when the input string is w , A_{TM} simply feeds $0w$ to \overline{J} and accepts if and only if \overline{J} accepts. If \overline{J} is Turing-recognizable, then A_{TM} is Turing-recognizable, a contradiction. Therefore, \overline{J} is not Turing-recognizable.

Problem 5 (10 Points)

Let $S_{TM} = \{\langle M \rangle \mid M \text{ is a TM that accepts } w^R \text{ whenever it accepts } w\}$.

Show that S_{TM} is undecidable.

We will reduce A_{TM} to S_{TM} where $A_{TM} = \{\langle M, w \rangle \mid \text{Turing Machine } M \text{ accepts } w\}$

Let us assume that S_{TM} is decidable. For convenience, we will denote the TM's that accept S_{TM} and A_{TM} by S and A respectively. A receives $\langle M, w \rangle$ as input. It rejects the input if the syntax is incorrect. Otherwise, it feeds $\langle X \rangle$ to S where X is a machine we will shortly define. A accepts $\langle M, w \rangle$ if and only if S accepts $\langle X \rangle$.

TM X works as follows. It scans the input and accepts if the string has the form 00^*11^* . Otherwise, it simulates M on w . If M accepts w , X accepts irrespective of what the input for X was. Note that the description of M and w is *hard-coded* into X ; it is not fed as input to X . Note that by construction, X accepts 00^*11^* if M rejects w ; otherwise, X accepts Σ^* . Therefore $\langle X \rangle$ will be accepted by S if and only if M accepts w . This means that A will accept $\langle M, w \rangle$ if and only if S accepts $\langle X \rangle$. By assumption, S is decidable. This implies A is decidable, a contradiction.