Homework #3

Artemis Kelly
CS 424 - Theory of Computation

April 14, 2020

Problem 1: 3.2(b - e)

Question: This exercise concerns TM M_1, whose description and state diagram appear in Example 3.9. In each of the parts, give the sequence of configurations that M_1 enters when started on the indicated input string.

3.2b: 1#1

$q_1 1\#1 \rightarrow \mathrm{x}q_3\#1 \rightarrow \mathrm{x}\#q_5 1 \rightarrow \mathrm{x}q_6\#\mathrm{x} \rightarrow q_7\mathrm{x}\#\mathrm{x} \rightarrow \mathrm{x}q_1\#\mathrm{x} \rightarrow \mathrm{x}\#q_8\mathrm{x} \rightarrow \mathrm{x}\#\mathrm{x}q_8\sqcup \rightarrow \mathrm{x}\#\mathrm{x}\sqcup q_{accept}$

3.2c: 1##1

$q_1 1\#\#1 \rightarrow \mathrm{x}q_3\#\#1 \rightarrow \mathrm{x}\#q_5 1\#1 \rightarrow \mathrm{x}\#\#q_{reject}1$

3.2d: 10#11

$q_1 10\#11 \rightarrow \mathrm{x}q_3 0\#11 \rightarrow \mathrm{x}0q_3\#11 \rightarrow \mathrm{x}0\#q_5 11 \rightarrow \mathrm{x}oq_6\#\mathrm{x}1 \rightarrow \mathrm{x}q_7 0\#\mathrm{x}1 \rightarrow q_7\mathrm{x}0\#\mathrm{x}1 \rightarrow \mathrm{x}q_9 0\#\mathrm{x}1 \rightarrow \mathrm{xx}q_2\#\mathrm{x}1 \rightarrow \mathrm{xx}\#q_4\mathrm{x}1 \rightarrow \mathrm{xx}\#\mathrm{x}q_4 1 \rightarrow \mathrm{xx}\#\mathrm{x}1q_{reject}$

3.2e: 10#10

$q_1 10\#10 \rightarrow \mathrm{x}q_3 0\#10 \rightarrow \mathrm{x}0q_3\#10 \rightarrow \mathrm{x}0\#q_5 10 \rightarrow \mathrm{x}0q_6\#\mathrm{x}0 \rightarrow \mathrm{x}q_7 0\#\mathrm{x}0 \rightarrow q_7\mathrm{x}0\#\mathrm{x}0 \rightarrow \mathrm{x}q_1 0\#\mathrm{x}0 \rightarrow \mathrm{xx}q_2\#\mathrm{x}0 \rightarrow \mathrm{xx}\#q_4\mathrm{x}0 \rightarrow \mathrm{xx}\#\mathrm{x}q_4 0 \rightarrow \mathrm{xx}\#q_6\mathrm{xx} \rightarrow \mathrm{xx}q_6\#\mathrm{xx} \rightarrow \mathrm{x}q_7\mathrm{x}\#\mathrm{xx} \rightarrow \mathrm{xx}q_1\#\mathrm{xx} \rightarrow \mathrm{xx}\#q_8\mathrm{xx} \rightarrow \mathrm{xx}\#\mathrm{x}q_8\mathrm{x} \rightarrow \mathrm{xx}\#\mathrm{xx}q_8\sqcup \rightarrow \mathrm{xx}\#\mathrm{xx}\sqcup q_{accept}$

Problem 2: 3.6

Question: In Theorem 3.21, we showed that a language is Turing-recognizable iff some enumerator enumerates it. Why didn't we use the following simpler algorithm for the forward direction of the proof? As before, s_1, s_2,... is a list of all strings in Σ^*
E = "Ignore the input.
1.) Repeat the following for i = 1,2,3...
2.) Run M on s_i
3.) If it accepts, print out s_i.

Answer: It is important to note that Machine M recognizes the language and does not decide the language. By definition, this means that there could be some string s_j in Σ^* for which the computation of M never halts, or is infinite, as M can fail to reject or accept it. When M comes upon this string, it will not be able to evaluate the following strings s_{j+1} and so on in Σ^* because computation is stuck on the previous string. Some of those following strings could be members of the language, and should be printed, but never will if we follow this "simpler algorithm." Therefore, we must use the more complex algorithm which contains a finite number of steps in order to ensure all strings are looked at.

Problem 3: 3.8(b)

Question: Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet {0,1}.

3.8b: { w | w contains twice as many 0s as 1s}

Answer: M = "On input w:
1.)Scan the input from left to right searching for a unmarked 1
    a.) If no 1 is found, go to step 3
    b.)If a 1 is found, mark it with an X. Go back to start of the tape and go to step 2
2.)Scan the input from left to right searching for two unmarked 0's
    a.) if you reach the end of the tape and didn't cross off two unmarked 0's, reject
    b.) if you did cross off two unmarked zeros, go to beginning of tape and back to step 1
3.)Go to beginning of tape and scan the input from left to right to see if there are any unmarked 1's or 0's
    a.) if there are none and you reach the blank spaces, accept
    b.)if there are any unmarked 1's or 0's, reject

Problem 4: 3.9

Question: Let a k-PDA be a pushdown automaton that has k stacks. Thus a 0-PDA is an NFA and a 1-PDA is a conventional PDA. You already know that 1-PDAs are more powerful (recognize a larger class of languages) than 0-PDAs.
A.) Show that 2-PDAs are more powerful than 1-PDAs.
B.) Show that 3-PDAs are not more powerful that 2-PDAs.
Hint: Simulate a Turing machine tape with two stacks

3.9a: Show that 2-PDAs are more powerful than 1-PDAs

Answer: We will prove that 2-PDAs are more powerful than 1-PDA by showing that a 2-PDA can do more than a 1-PDA. We know that 2-PDAs are at least as powerful as 1-PDA because a 2-PDA can stimulate a 1-PDA by using 1 stack.
To prove a 2-PDA is more powerful, we will use the language $\{a^n\ b^n\ c^n \mid n \geq 0\}$. This language is not context free, and therefore, can not be accepted by a 1-PDA. (This language

is proven to not be context free in example 2.36 in the book, so I will provide a summary of that proof.)

The Language is not context free so a 1-PDA can not accept it

Here I will show that this language is not context free, using the pumping lemma. To start, assume the language $\{a^n\ b^n\ c^n \mid n \geq 0\}$ is context free. We will let the pumping length be p and let s = $a^p\ b^p\ c^p$. In this case, the length of s is longer than the pumping length, | s| = 3p, so s may be divided into five pieces s = uvxyz. Using conditions 2 either v or y must be non empty pieces. Using condition 3, |vxy| $\leq$ p. There are two different ways that v and y can be grouped to contain letters of the string. But we show that no matter how we divide s into the 5 parts, one of the three conditions of the lemma will be violated.

The first is if both v and y are made up of only one type of alphabet symbol. Neither v or y contain both a's and b's or b's and c's. In this case, when you pump the string, for example uv^2xy^2z, there can not be an equal number of all a's, b's and c's. This violates condition 1 of the lemma.

The second is if either v or y contains more than one letter from the alphabet, containing both a's and b's or b's and c's. Then, when the string is pumped in the same fashion as above, it may contain the right number of letters but they will no longer be in the right order. In this case, the string is no longer a member of the language and we have a contradiction. because both cases resulted in a contradiction, we know that the language is not a context free language. Because of this, a 1-PDA can not accept it.

Proof that the 2-PDA can accept it

However, we can show how it can be read by a 2-PDA, G, that recognizes L:

1.)G starts in the accept state, q_accept. This is to make sure that we accept the empty string. If an 'a' is read, push the 'a' into stack one and the PDA moves into state q_1. If anything else is read in before an 'a', reject the string / go to the reject state, q_reject.

2.) Now in state q_1. When an 'a' is read, push it into stack 1 and stay in q1. If a 'b' is read, go to q_2, push the 'b' onto stack 2 and pop an 'a' from stack 1. If a 'c' is read, go to the reject state and reject the string.

3.)Now in state q_2. Each time we read a 'b', pop an 'a' from stack 1 and push a 'b' onto stack 2. If there is no more 'a' to pop from stack 1 as a b is read, go to the reject state and reject the string. If a 'c' is read in, go to state q_3 and pop a 'b' from stack 2.

4.) Now in state q_3. Each time a 'c' is read, pop a 'b' from stack 2. If an 'a' or a 'b' is read, go to the reject state and reject the string. If another c is read and there are no more b's, go to the reject state and reject the string.

5.) Only accept if both of the stacks are empty. Otherwise, reject the string.

Because we have just proved that there is at least one language that can be recognized by a 2-PDA and not a 1-PDA, we have proved that 2-PDAs are more powerful than 1-PDAs.

3.9b: Show that 3-PDAs are not more powerful that 2-PDAs.

Answer: A 2-PDA is able to simulate a Turing machine. The first stack will have the entirety of the input tape on it, with the top of the stack representing the beginning of the tape, where the head is. If the head were to "move right", the top item from the first stack would be popped and pushed onto the second stack. If the head were to move left, the top item in the second stack would be popped and pushed onto the first stack. In this way, a

2-PDA is perfectly able to simulate a turing machine. Therefore, a 2-PDA is just as powerful as a Turing machine. In addition, a turing machines can simulate 2-PDA and 3-PDA by using 2 or 3 tapes, respectively. In fact, a turing machine can simulate a k-PDA for k $\geq$ 2 by using k-tapes. The i-th tape is a simulation of the i-th stack by being a copy of the values on the stack. The left side of the tape represents the bottom of the stack and the head of the tape will look at the last value of the tape, on the right, which will represent the top of the stack. It can only move to simulate a push or pop. In the case of a push, the head will move right and write the new value. In the case of a pop, the head will move left and erase the last value. Because we know that every multi tape turing machine has an equivalent single tape turing machine (theorem 3.13) we can conclude a a Turing machine is at least as powerful as a k-PDA for k $\geq$ 2. And since we know a 2-PDA is equivalent in power to a turing machine, we know a 2-PDA is equivalent to a k-tape turing machine, and therefore we can conclude that for k $\geq$ 2, k-PDA's are equivalent in power, including 3-PDA and 2-PDA.

Problem 5: 3.15(c,e)

Question: Show that the collection of decidable languages is closed under the operation of

3.15c: Show that the collection of decidable languages is closed under the operation of star.

Answer: Let L be a decidable language. There is a Turing Machine M that decides L by definition. We can construct a non deterministic 2-tape Turing Machine M_* that will help us show closure under star operation. M_* will serve as a nondetermistic decider because M is a decider for L.
M_*: "On input w:
1.) Nondeterminalistcally select a non empty left-most piece of the input w and copy it on the second tape.
2.) Run M on the selected string on the second tape
3.) If the whole input was processed and M accepted, then M_* accepts as well. If M accepted but some parts of the input w still need to be processed, the second tape must be cleared and we continue with step 1. If M rejects, then M_* will reject as well.
Because any 2-tape nondetermistinistic decider is equivalent to a single tape deterministic decider, we have shown there is a decider for the Star operation. In addition, going off of corollary 3.19: A language is decidable if and only if some nondeterministic Turing Machine decides it, like in this case.

3.15e: Show that the collection of decidable languages is closed under the operation of intersection.

Answer: Given two decidable languages, L_1 and L_2, we take their intersection to be the set of all strings that are in both L_1 and L_2, $L_1 \cap L_2$. Let D_1 and D_2 be the turing machines that decide L_1 and L_2. Using this, we will construct a turing machine D that decides $L_1 \cap L_2$.

D = "On the input w
1.) Run D_1 on w. If it rejects, reject. If it accepts, go on to part 2
2.) Run D_2 on w. If it rejects, reject. If it accepts, D will accept
D will halt on input w and must be a decider because steps 1 and 2 rely on D_1 and D_2, which both halt on any input w because they are deciders. Therefore, D is a decider, showing that the intersection of decidable langauges must be closed under interestions.

Problem 6: 3.16(d)

Question: Show that the collection of Turing-recognizable languages is closed under the operation of intersection.
Given two turing-recognizable languages, L_1 and L_2, we take their intersection to be the set of all strings that are in both L_1 and L_2, $L_1 \cap L_2$. Let D_1 and D_2 be the turing machines that recognize L_1 and L_2. Using this, we will construct a turing machine D that recognizes $L_1 \cap L_2$.
D = "On the input w
1.) Run D_1 on w. If it rejects, halt computation in D and have D reject. If it accepts, go on to part 2. If D_1 neither accepts or rejects, computation would never halt and computation for D will never halt either.
2.) Run D_2 on w. If it rejects, halt computation in D and have D reject as well. If it accepts, D will accept as well. If D_2 neither accepts or rejects, computation would never halt and computation for D will never halt either.
D must be a recognizer because steps 1 and 2 rely on D_1 and D_2. D_1 and D_2 both take a finite number of steps to accept strings in its language, so therefore, D only takes a finite number of steps to accept strings in its language. D will never halt on inputs that neither D_1 or D_2 halt on.

Problem 7: 3.13

Question: A Turing machine with stay put instead of left is similar to an ordinary Turing machine, but the transition function has the form $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{ R , S \}$. At each point, the machine can move its head right or let it stay in the same position. Show that this Turing machine variant is not equivalent to the usual version. What class of languages do these machines recognize?
The main thing about this kind of a machine is it can never reread symbols it has passed. Staying in put does not prove to be very useful in this analysis because performing the stay put operation once vs 1000 times achieves the same thing in terms of what is going on with the tape. In addition, surrounding a right move by a single or many stay put transitions just moves right once. Effectively, this machine can only move right and read the input string a single time. Because of this, we can rewrite the machine as a DFA. We are able to cut out the middle transitions and we are able to have only a single transition arrow at each point of this machine, if we were to draw it out. Therefore, this machine would be able to recognize regular languages.

Lets start with a turing machine that has stay put instead of left. M = (Q, Σ, Γ, δ, q_0, q_{accept}, q_{reject}). We will construct a DFA, D = (Q^*, Σ^*, δ^*, q_0^*, F). We will modify M to help us define D in ways that do not modify the language that is recognized by either.

1.) Add a new symbol, for example *, that M will write instead of writing blanks on the tape. We will extend the transition function of M so that when it sees this symbol, it will follow the same transition as if it had seen a blank.

2.) When M transitions to the accept or reject state, the reading head will move right, not stay put. This will not change the language that it accepts.

3.) Make $Q^* = Q$, $\Sigma^* = \Sigma$ and $q_0^* = q_0$

4.) Consider the transition function:

$$\delta^*(q, \sigma) = \begin{cases} q, & \text{if } q \in \{q_{accept}, q_{reject}\} \\ q_{reject}, & \text{if M in state q reads } \sigma \text{ and stays put} \\ q^*, & \text{where } q^* \text{ is the state M enters when it first moves right when in state q and reading } \sigma \end{cases}$$