CS 420, Spring 2019
Homework 8 Solutions

1. Let M_1 be the Turing machine whose description is given in Example 3.9. Give the sequence of configurations that M_1 enters when started on the following input strings.

   (a) 011#011.

$q_1 011\#011$	$xq_7 11\#x11$	$xx1\#q_6 xx1$	$xxx\#xq_5 x1$	$xxxq_1\#xxx$
$xq_2 11\#011$	$q_7 x11\#x11$	$xx1q_6\#xx1$	$xxx\#xxq_5 1$	$xxx\#q_8 xxx$
$x1q_2 1\#011$	$xq_1 11\#x11$	$xxq_7 1\#xx1$	$xxx\#xq_6 xx$	$xxx\#xq_8 xx$
$x11q_2\#011$	$xxq_3 1\#x11$	$xq_7 x1\#xx1$	$xxx\#q_6 xxx$	$xxx\#xxq_8 x$
$x11\#q_4 011$	$xx1q_3\#x11$	$xxq_1 1\#xx1$	$xxxq_6\#xxx$	$xxx\#xxxq_8$
$x11q_6\#x11$	$xx1\#q_5 x11$	$xxxq_3\#xx1$	$xxq_7 x\#xxx$	$xxx\#xxx \sqcup q_{accept}$
$x1q_7 1\#x11$	$xx1\#xq_5 11$	$xxx\#q_5 xx1$		

   (b) 110#11.

$q_1 110\#11$	$x10q_6\#x1$	$xxq_3 0\#x1$	$xx0q_6\#xx$	$xxx\#q_4 xx$
$xq_3 10\#11$	$x1q_7 0\#x1$	$xx0q_3\#x1$	$xxq_7 0\#xx$	$xxx\#xq_4 x$
$x1q_3 0\#11$	$xq_7 10\#x1$	$xx0\#q_5 x1$	$xq_7 x0\#xx$	$xxx\#xxq_4$
$x10q_3\#11$	$q_7 x10\#x1$	$xx0\#xq_5 1$	$xxq_1 0\#xx$	$xxx\#xx \sqcup q_{reject}$
$x10\#q_5 11$	$xq_1 10\#x1$	$xx0\#q_6 xx$	$xxxq_2\#xx$	

   (c) 0#0#.

$q_1 0\#0\#$	$xq_1\#x\#$
$xq_2\#0\#$	$x\#q_8 x\#$
$x\#q_4 0\#$	$x\#xq_8\#$
$xq_6\#x\#$	$x\#x\#q_{reject}$
$q_7 x\#x\#$	

2. **Exercise 3.7**

   The description is not legitimate because there are infinitely many possible settings of the variables $x_1, \ldots, x_k$ to integral values, so a Turing machine cannot test them all and reject if none of them are roots. (It is legitimate for a Turing machine to test the possible integral values for $x_1, \ldots, x_k$ and accept if any of them are roots of p, but if p has no integral roots, the Turing machine will loop, not reject.)

3. (a) Give an implementation-level description of a one-tape Turing machine that decides the language

   $$\{w \in \{a, b, c\}^* | n_a(w) = n_b(w) \text{ and } n_a(w) > n_c(w)\}$$

   Solution:
   $M =$ "On input string w:

1. Repeat the following until it becomes impossible:
   2. Scan the input and cross off one a, one b and one c.
3. If there are no a's or no b's left on the tape *reject*.
4. If there are a's and b's left on the tape, repeat the following until it becomes impossible:
   5. Scan the tape and cross off one a and one b.
6. If all symbols are crossed off, *accept*, else *reject*.

(b) Give a more efficient multi-tape Turing machine to decide the language from Part (a).

Solution:

We define a four-tape Turing machine N to decide the same language. $N =$ "On input string w:

1. Scan the input, and copy all a's to tape 2, all b's to tape 3 and all c's to tape 4..
2. Return the tape heads to the first cells on tapes 2 ,3, and 4 and then scan to the right on each tape until the first blank is met on any of the three tapes.
3. If the tape head on tape 2 or tape 3 is reading a blank, *reject*.
4. Continue moving to the right on tapes 2 and 3 until a blank is read on one of these tapes. If both tape heads reach blanks at the same time, *accept*, else *reject*.."

4. **Problem 3.9**

(a) This part follows from Part (b), but we give a separate proof. We know that 1-PDAs are just PDAs, so since $A = \{0^n 1^n 2^n | n \geq 0\}$ is not a CFL, there is no 1-PDA that recognizes A. We will describe a 2-PDA M that recognizes A. This will show 2-PDAs are more powerful than 1-PDAs.

M will stop in a nonaccept state if it sees a 0 after it sees 1s or 2s or if it sees a 1 after it sees 2s. Initially, M marks the bottom of both its stacks. While it is reading 0s, it pushes the 0s onto both of its stacks. While it reads 1s, it matches the 1s against the 0s on the first stack. If there are more 0s than 1s, it stops in a nonaccept state. Once the first stack is empty, M does not read any more 1s, but instead matches 2s from the tape with the 0s on the second stack. When the second stack is empty, M goes into an accept state and doesn't read any more symbols.

(b) First, we argue that any 3-PDA can be simulated by a TM. Let M be a 3-PDA. A nondeterministic TM M' to simulate M will have 4 tapes. The first tape will hold the input and the other three tapes will hold the three stacks of M, one per tape, with the bottom of the stack at the left end of the tape. M' nondeterministically simulates one possible computation of M on its input step-by-step, simulating the changes to the stack that M makes by writing on its tape. If this computation of M reaches an accepting state after reading its whole input, then M' accepts. If the computation of M gets stuck before reading the whole input or halts in a nonaccepting state after reading the whole input and without reaching an accepting state after reading the whole input, then M' rejects. If the computation of M goes into an infinite sequence of ε moves, then M' loops.

Now, we argue that every TM can be simulated by a 2-PDA. Let M be a TM. A 2-PDA P to simulate M works as follows. P first marks the bottom of its two stacks. Then it reads its input, pushing it all onto its first stack. Then P pops all but the first symbol from the first stack and pushes these symbols onto the second stack. At this point, P has the first symbol of the input on its first stack and all the rest of the input, in left-to-right order, on its second stack, with the right end of the input at the bottom of the stack.

P now simulates M one step at a time, using moves that do not read any symbols from P's tape. The current symbol of M is always at the top of the first stack. If M moves right, then P replaces the top of the first stack with the symbol written by M and pops the top symbol off the second stack and pushes it onto the first stack. If the marker is at the top of the second stack, then P leaves this symbol alone instead of popping it and pushes a blank onto the first stack.

If M moves left, then P pops a symbol off the first stack and pushes onto the second stack the symbol that M writes in its move. If this brings the marker to the top of the first stack, then this means that M has tried to move left from the left end of the tape, so P pops the symbol it just put onto the second stack and puts it back on the first stack.

If M reaches its accept state, then P goes into an accept state. If M goes into its reject state, then P stops simulating M, but does not go into an accept state. If M loops, then P has an infinite sequence of ε moves after reading its whole input. M and P recognize the same language.

Now we have

- Every 2-PDA can be simulated by a 3-PDA. (Since a 3-PDA can just ignore its third stack.)
- Every 3-PDA can be simulated by a TM.
- Every TM can be simulated by a 2-PDA.

Thus, 2-PDAs, 3-PDAs and TMs are all equally powerful.

5. **Problem 3.11**

We must show that every ordinary Turing machine can be simulated by a Turing machine with doubly infinite tape and that every Turing machine with doubly infinite tape can be simulated by an ordinary Turing machine.

First suppose that M is an ordinary Turing machine. We define a Turing machine M' with doubly infinite tape to simulate M as follows. M' begins by moving left one cell and printing a special symbol \$. Then, M' moves right and goes into the start state of M. From then on, M' just simulates M, except that whenever M' reads \$, it moves right and stays in the same state.

Now suppose that N is a Turing machine with doubly infinite tape. We define an ordinary Turing machine N' that simulates N. N' will be a two-tape Turing machine. The book shows that a two-tape TM can be simulated by a one-tape TM. The first tape of N' contains the cells from the first symbol of the original input to the right. The second tape contains the cells on N's tape to the left of the original input, in reverse order. Initially, N' transforms its input $w_1 \cdots w_n$ on the first tape into $\#w_1 \cdots w_n$ and puts a # in the first cell of tape 2. The tape head on tape one sits on w_1 and the tape head on tape two sits on #. N' then starts simulating moves of N one at a time. At the start of each move simulation, exactly one of the two tape heads will be sitting on #. If the first tape head is not on #, then a move is simulated on tape 1. If after the move is simulated, the tape head on tape 1 reads #, then the tape head on tape 2 is moved one cell to the right. If at the beginning of a move simulation the tape head on tape 2 is not reading #, then the move is simulated on tape 2, but a left move for N causes the tape head to move right on the second tape of N' and a right move for N causes N' to move left on tape 2.

6. In our definition of Turing machine, if the machine tries to move left from the first tape cell, then it stays put. An alternative defintion would be that if the Turing machine tries to move left from its first cell, then it halts and rejects. (So in the alternative definition, there are two ways to reject - going to the reject state and trying to move left from the first cell.) Show how to transform a Turing machine M of the type we defined in class into a Turing machine M' using this alternative defintion in such a way that M and M' recognize the same language.

   Solution:

   Given a Turing machine M of the type we defined in class, we define a Turing machine M' that uses the alternative definition that simulates M as follows. M' begins by shifting its input one cell to the right and putting a new symbol $*$ in the first cell. It then moves to it's second cell and starts simulting M one move after another. If M' ever reads $*$ (which means that M tried to move left from its first cell), M' moves right and continues simulating M. Since M' never moves left from its first cell, the only reason it would reject is if M rejects, and M and M' recognize the same language.

7. (a) **Problem 3.15d**

   Let M be a Turing machine that decides a language A. Then a Turing machine M' that decides the complement of the language A

is the same as M except that the accepting and rejecting states are reversed. Thus, the complement of A is decidable.

(b) **Problem 3.15e**

Let M_1 and M_2 be Turing machines that decide the languages A_1 and A_2, respectively. Then, a 2-tape TM M that decides $A_1 \cap A_2$ works as follows.

$M =$ "On input string w

1. Copy w onto tape 2.
2. Simulate M_1 on w using tape 1 and M_2 on w using tape 2.
3. If both M_1 and M_2 accept, then *accept*; else *reject*."

M is a decider since both M_1 and M_2 are deciders. If the input w is in $A_1 \cap A_2$, then both M_1 and M_2 will accept w, so M accepts w. If w is not in $A_1 \cap A_2$, then either M_1 or M_2 (or both) will reject w, so M rejects w. Thus M decides $A_1 \cap A_2$ and $A_1 \cap A_2$ is decidable.

8. (a) **Problem 3.16b**

Let M_1 and M_2 be Turing machines that recognize the languages A_1 and A_2, respectively. Then, a nondeterministic, 3-tape TM N that recognizes $A_1 A_2$ works as follows.

$N =$ "On input string w

1. Nondeterministically divide w into $w = uv$. Copy u onto tape 2 and v onto tape 3.
2. Simulate M_1 on u using tape 2. If M_1 rejects, then *reject*. If M_1 accepts, then go to Step 3.
3. Simulate M_2 on v using tape 3. If M_2 accepts, then *accept*. If M_2 rejects, then *reject*."

(Note that if either M_1 loops on u or M_2 loops on v, then the given computation of N loops.) If the input w is in $A_1 A_2$, then there will be some computation of N on w (the one that makes the right guess where to break up w into u and v) that accepts. If w is not in $A_1 A_2$, then all computations of N on w either reject or loop. Thus, N recognizes $A_1 A_2$ and $A_1 A_2$ is Turing-recognizable.

(b) **Problem 3.16d**

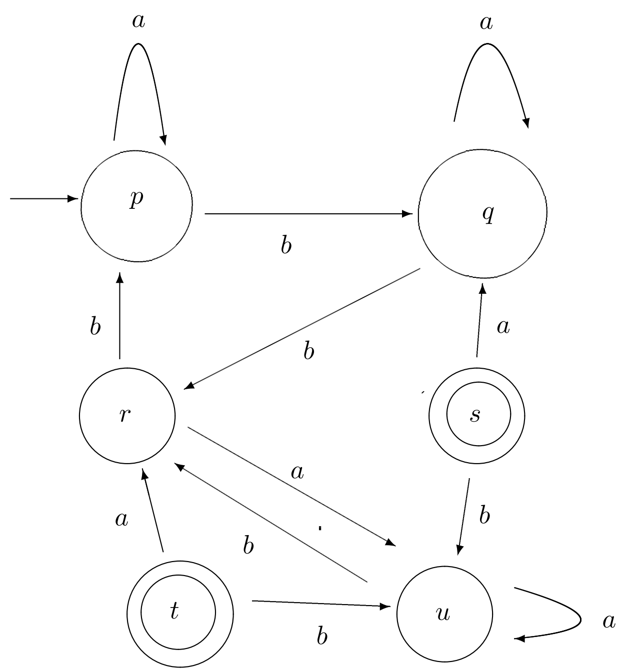
Let M_1 and M_2 be Turing machines that recognize the languages A_1 and A_2, respectively. Then, a 2-tape TM M that recognizes $A_1 \cap A_2$ works as follows.

$M =$ "On input string w

1. Copy w onto tape 2.
2. Simulate M_1 on w using tape 1. If M_1 rejects, then *reject*. If M_1 accepts, then go to Step 3.
3. Simulate M_2 on w using tape 2. If M_2 accepts, then *accept*. If M_2 rejects, then *reject*."

(Note that if either M_1 or M_2 loops on w, then M loops on w.) If the input w is in $A_1 \cap A_2$, then both M_1 and M_2 will accept w, so M accepts w. If w is not in $A_1 \cap A_2$, then either M_1 or M_2 (or both) will reject or loop on w, so M either rejects or loops on w. Thus M recognizes $A_1 \cap A_2$ and $A_1 \cap A_2$ is Turing-recognizable.

9. Apply the method from class that decides E_{DFA} to the following DFA and answer the questions below.



(a) List the states you mark in the order they get marked.
$\underline{p, q, r, u}$

(b) Does the DFA belong to E_{DFA}? Yes

(c) How does your answer to (b) follow from your answer to (a)?
No accept state is marked.

10. Apply the method from class that decides E_{CFG} to the following CFG and answer the questions below.

$$S \rightarrow aTbU|aSTb$$

$$\begin{array}{rcl} T & \rightarrow & YaU|bT \\ U & \rightarrow & aYbY|VW \\ V & \rightarrow & aV|bW \\ W & \rightarrow & aW|bV \\ X & \rightarrow & bX|\varepsilon \\ Y & \rightarrow & aY|a|TU \end{array}$$

(a) List the terminals and variables you mark in the order they get marked. (List each terminal and variable only the first time you mark it. There is more than one possible order.)
$\underline{a, b|X, Y|U|T|S}$

(b) Does the CFG belong to E_{CFG}? No

(c) How does your answer to (b) follow from your answer to (a)?
S is marked

11. The language EQ_{REX} is defined as $\{\langle R, S\rangle | R, S$ are regular expressions and $L(R) = L(S)\}$. Prove that EQ_{REX} is decidable.

Solution:

A Turing machine M that decides the language EQ_{REX} is given by

$M =$ "On input $\langle R, S\rangle$ where R and S are regular expressions,

1. Combining the methods of Lemma 1.55 and Theorem 1.39, produce DFAs B and C with $L(B) = L(R)$ and $L(C) = L(S)$.
2. Run the TM F that decides EQ_{DFA} on $\langle B, C\rangle$.
4. If F accepts, then *accept*. If F rejects, then *reject*."

12. Let $ALL_{NFA} = \{\langle A\rangle | A$ is an NFA and $L(A) = \Sigma^*\}$. Show that ALL_{NFA} is decidable.

Solution:

ALL_{NFA} is decided by the following Turing machine M.

$M =$ "On input $\langle A\rangle$ where A is an NFA,

1. Using the method of Theorem 1.39, construct a DFA B that is equivalent to A.
2. Obtain a DFA C from B by reversing accept and reject states.
3. Run the TM T that decides E_{DFA} on $\langle C\rangle$.
3. If T accepts, then *accept*. If T rejects, then *reject*."

Note that you have to transform A into a DFA because the complementation construction does not always work for NFAs.