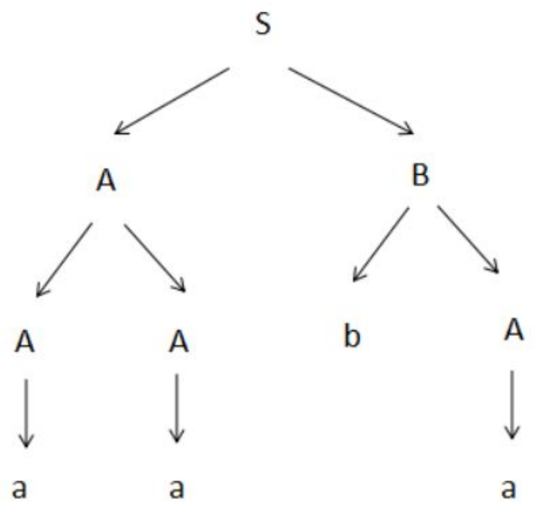
Final

1. Obviously B is not generating, so we can just simply delete it. So the CFG is turned into only A→aA|a. It is not reachable, so the grammar cannot generate any language.

2. Since there is no string given, I'll assume a string in the CFG and give the parse tree and derivation.

   Assume the string is aaba, which is generated by the grammar.

   Parse tree:



   Leftmost derivation: S→AB→AAB→aAB→aaB→aabA→aaba

   Rightmost derivation: S→AB→AbA→Aba→AAba→Aaba→aaba

3. (a) S→AaBaA|AbBbA|CB|BC|bA|Ab
   A→aA|bA|ε
   B→aBb|ab
   C→aA|bA
   (b)S→A#B#A|B
   A→aA|bA|#A|ε
   B→aBa|bBb|aCa|bCb
   C→#A#|#

4. It contains strings that consist of 0's and broken into 3 or 2 parts by #, the number of 0's can be 0, if it is broken into 2 parts, then the left part of # contains half number of 0's than the right part of #.

   It is not regular, using pumping lemma: for string $0^n\#0^{2n}$ which is generated by the grammar, break it into xyz. Since $|xy| \le n$, y only consist of 0 and is not empty. So, let k=0, then in string xy^kz, the number of 0's left to # is less than n and cannot be half of 2n. That is to say xy^kz is not in the language, so it is not regular.

5. A→AB|BA|BB|CC
   B→CC
   C→0

6. (a)

(1) the TM starts at the left of the tape with state q_0, read to right and jump by all 0's with notion * (which means checked) and all 1's. Find the first 0 with notion B (which means not-checked-yet), change B into * and state into q_1, go right.

(2) from (1), when find another 0, write * and change state into q_2, go left.

(3) jump by every symbol, when arrive the start of the tape, change state into q_3, go right.

(4) from (3), jump by every 1's with * and 0's, find the first 1 with B, write *, state into q_4, go left.

(5) jump by every symbol, when arrive the start of the tape, change state into q_0, go right, begin step(1).

(6) in (2), if it cannot find another 0, then halt.

(7) in (4), if it cannot find 1, then halt.

(8) in (1), if it cannot find the first 0 and goes to the end of the tape, state into q_5, go left.

(9) from (8), jump by every symbol with *, when find any symbol with B, halt.

(10) from (8), jump by every symbol with *, if it arrives the start of the tape, state into q_6, q_6 is the accept state.

(b)

Most of the steps can simply copy from (a), and do a little adjustment to the rest.

(1) same as (a).

(2) same as (a).

(3) same as (a).

(4) same as (a).

(5) same as (a).

(6) in (2), if it cannot find another 0, state into q_6, q_6 is the accept state.

(7) in (4), if it cannot find 1, state into q_6, q_6 is the accept state.

(8) same as (a).

(9) from (8), jump by every symbol with *, when find any symbol with B, state into q_6, q_6 is

the accept state.

(10)from (8), jump by every symbol with *, if it arrives the start of the tape, halt.

7. 000111:TRUE 10001:FALSE 0101:t=TRUE 0000:TRUE B:TRUE

0110011→1100110 001110→011100 100100→100100

This Turing machine M move every 1's in the string forward by one, but if the string start with 1, then the string remains the same.

8. (a) is Turing-recognizable, it is easy to design an algorithm to accept it. In brief, give 3 distinct inputs and simulating M on these inputs. As for (b), from our textbook p398, the Rice's Theorem, both (a) and (b) are undecidable. Since (b) is complement of (a), so (b) is not Turing-recognizable.

9. (a) The start state is q_0

(1) $\delta(q_0,\$)=(q_1,\$,R)$, from q_0, when see in a \$, change state into q_1, go right.

(2) $\delta(q_1,\$)=(q_2,\$,R)$, from q_1, when see in a \$, change state into q_2, go right.

(3) $\delta(q_2,\$)=(q_0,\$,R)$, from q_2, when see in a \$, change state into q_0, go right. This time, write a \$ on another tape.

(4) $\delta(q_0,B)=HALT$, from q_0, when arrive the end of the string, the TM halt and write # on the other tape.

(5) $\delta(q_1,B)=HALT$, from q_1, when arrive the end of the string, the TM halt and write #\$ on the other tape.

(6) $\delta(q_2,B)=HALT$, from q_2, when arrive the end of the string, the TM halt and write #\$\$ on the other tape.

(b) To generalize to any arbitrary symbol, suppose the symbol is %. Just simply change every \$ into % in the TM illustrated in (a).

(c) suppose now we generalize it by "to i", then we will need i states $\{q_0,q_1,...,q_{i-1}\}$. Step (1)-(3) in (a) can be similarly generalized to (1)-(i) steps for states $\{q_0,q_1,...,q_{i-1}\}$. (i) step should be, $\delta(q_{i-1},\$)=(q_0,\$,R)$, from q_{i-1}, when see in a \$, change state into q_0, go right. This time, write a \$ on another tape. Step (4)-(6) is generalized into step (i+1)-(2i), and it should be: $\delta(q_k,B)=HALT$, from q_k, when arrive the end of the string, the TM halt and write $\#(\$)^k$ on the other tape.

10. (a) $P \cup P = P$ that means the problem that is to solve 2 problems in P is also in P.

(b) $NP \cup NP = NP$ because if we can verify a certificate in polynomial time, then verify 2 certificate will also cost polynomial time.

(c) $NPC \cup NPC = NPC$ because if we can reduce every NP problem to a problem which is a NPC problem, then we can reduce it to 2 NPC problems.

11. This language is recursive. In problem 9(c) I illustrated how to divide a string. Using the similar strategy, if the length of a input string is n, we can do division from 2 to n-1(or smaller, to n square, that doesn't matter), that is in polynomial time. If divide by some k yield there is no remainder, accept and halt. If there is always a remainder, just halt. That means we build a TM that accept the language and the TM is always halt. So the language is recursive.