1. IN order to Prove that the union operator is NOT closed under DCFL. we will utilize a proof by contradiction

   Let $L_1 = \{w | w \in \{a^*, b^*, c^*\}$ and w contains the same number of a's as b's$\}$

   Let $L_2 = \{w | w \in \{a^*, b^*, c^*\}$ and w contains the same number of a's as c's$\}$

   Assume all DCFL's are closed under the union operator. Since L_1 and L_2 are DCFLs, then $L_1 \cup L_2$ is also a DCFL.

   If $L_1 \cup L_2$ is a DCFL, then there is a DPDA which accepts it. but there is no DPDA which can accept it because when an automata reads the string w in, it must guess whether to begin popping A's to match B's in order to accept a word from L_1. but if it begins popping A's to match B's then it will not be able to pop A's to match C's to accept a word in L_2. Therefore without nondeterminism it is impossible to build an automata which can accept $L_1 \cup L_2$. But if there is no Deterministic PDA which accepts this language, then this language cannot be a DCFL. and there in lies a contradiction. Thus, we have proved $L_1 \cup L_2$ is not a DCFL, which is sufficient to show that the union operator is not closed under DCFL.

IN order to Prove that the Kleene Star operator is NOT closed under DCFL. we will utilize a proof by contradiction

Let $L_1 = \{w | w \in \{(a, b)a^n, b^n\}$ and n is some positive integer$\}$

Assume all DCFL's are closed under the Kleen Star operator. Since L_1 is a DCFL, then L_1^* is also a DCFL.

If L_1 is a DCFL, then there is a DPDA which accepts it. but there is no DPDA which can accept it because although a determinstic automaton can easily recognize L_1, nondeterminism is required to recognize multiple instances of strings in L_1. The reason is that the automaton must guess upon reading a 'b' if it is the last potentially "matching" 'b' of some word in L_1 or if it is the first 'b' of the next iteration of some new potentially valid string in L_1. Therefore without nondeterminism it is impossible to build an automata which can accept L_1^*. But if there is no Deterministic PDA which accepts this language, then this language cannot be a DCFL. There in lies a contradiction. Thus, we have proved L_1^* is not a DCFL as required, which is sufficient to show the Kleene Star is not closed under DCFL.

IN order to Prove that the concatenation operator is NOT closed under DCFL. we will utilize a proof by contradiction

Let $L_1 = \{w | w \in \{a^*, b^*\}$ and w contains the same number of a's as b's$\}$

Let $L_2 = \{w | w \in \{b^*, c^*\}$ and w contains the same number of b's as c's$\}$

Assume all DCFL's are closed under the concatenation operator. Since L_1 and L_2 are DCFLs, then $L_1 \circ L_2$ is also a DCFL.

If $L_1 \circ L_2$ is a DCFL, then there is a DPDA which accepts it. but there is no DPDA which can accept it because when an automata reads the string w in, it must guess after every 'b' if the first word is ended or not. Consider the example of string $w_1 =$ "aab" and word $w_2 =$ "bbbcc". There is no way to know where the first word ends (as the endmarker is lost when we supply a raw string to the automaton). Therefore without nondeterminism it is impossible to build an automata which will know which 'b's belong to which words. and thus no DPDA can be built to accept $L_1 \circ L_2$. But if there is no Deterministic PDA which accepts this language, then this language cannot be a DCFL. and there in lies a contradiction. Thus, we have proved $L_1 \circ L_2$ is not a DCFL, which is sufficient to show that the concatenation operator is not closed under DCFL.

---

2. In order to prove that if a CFL is inherently ambiguous it cannot be a DCFL we will utilize a proof by contradiction

   Assume that there is some CFL Z which is inherently ambiguous and is a DCFL. Since it is inherently ambiguous, by definition it can only be generated by an ambiguous grammar. If the grammar which generates Z is ambiguous, then it can generate the same string in multiple ways. If the same string in language Z can be generated in multiple ways, then there are multiple parse trees for the same string. If a string can be parsed in multiple different ways, then a deterministic automaton will not suffice for recognizing such a string. The reason for this is that in order to be deterministic there can be exactly one transition for each character of the input. Multiple interpretations of the same string would require an automaton to utilize non-determinism in guessing every possible interpretation starting from each rule of the grammar which could result in the current character being read in. If nondeterminism is required to recognize some word in Z, then there cannot be a DPDA which recognizes the language Z. if there is no DPDA which recognizes Z, then Z cannot be a DCFL. But that is a contradiction, because we assumed Z was a DCFL. Therefore we have shown Z is not a DCFL and thus proved that if a CFL is inherently ambiguous it cannot be be a DCFL.

---

3. (a)
   WE WILL SHOW THAT FOR EVERY WORD OF L(G), THERE ARE AN EQUAL NUMBER OF A'S AND B'S

   We will show this with an inductive proof on the size n of any word w in L(G). for each step in our proof we will look at every rule of grammar G which is applicable to the situation and determine that for every case, there is an equal number of a's and b's

BASE CASE: let n = 0

Then we know the word is 0 character long. In this case the start state will map directly to the output string (emptyString) there are 0 a's and 0 b's. So we have proved the base case.

Inductive Hypothesis (notice for every rule in the grammar exactly 0 or 2 terminals are added to the derived word, so then we do not pay attention to odd length words, because there are none):

——for all words of size (n = k):

Suppose all words of k length have the same number of a's and b's

then for each word of size k + 2 there are also the same number of a's and b's

we consider each case possible by looking at every rule of G

From the start state we transition to T endmarker. No terminals are added so we still have the same number of a's and b's

From T we have three derivation options.

1. $T \rightarrow TaTb$ in which case exactly one a and one b are added to the derived word plus 2 times all other possible mappings of T.

2. $T \rightarrow TbTa$ in which case exactly one a and one b are added to the derived word plus 2 times all other possible mappings of T.

3. $T \rightarrow \epsilon$ in which case exactly zero a's and b's are added to the derived word.

Since we have shown for every rule of L(G) there is either an equal number of a and b terminals or an equal number of Variable T which recursively generate an equal number of a and b terminals or another instance of itself, then for every word generated by G there will be an equal number of a's and b's.

Therefore, we have proved that every word in L(G) has the same number of a's and b's by induction on the length of w as required.

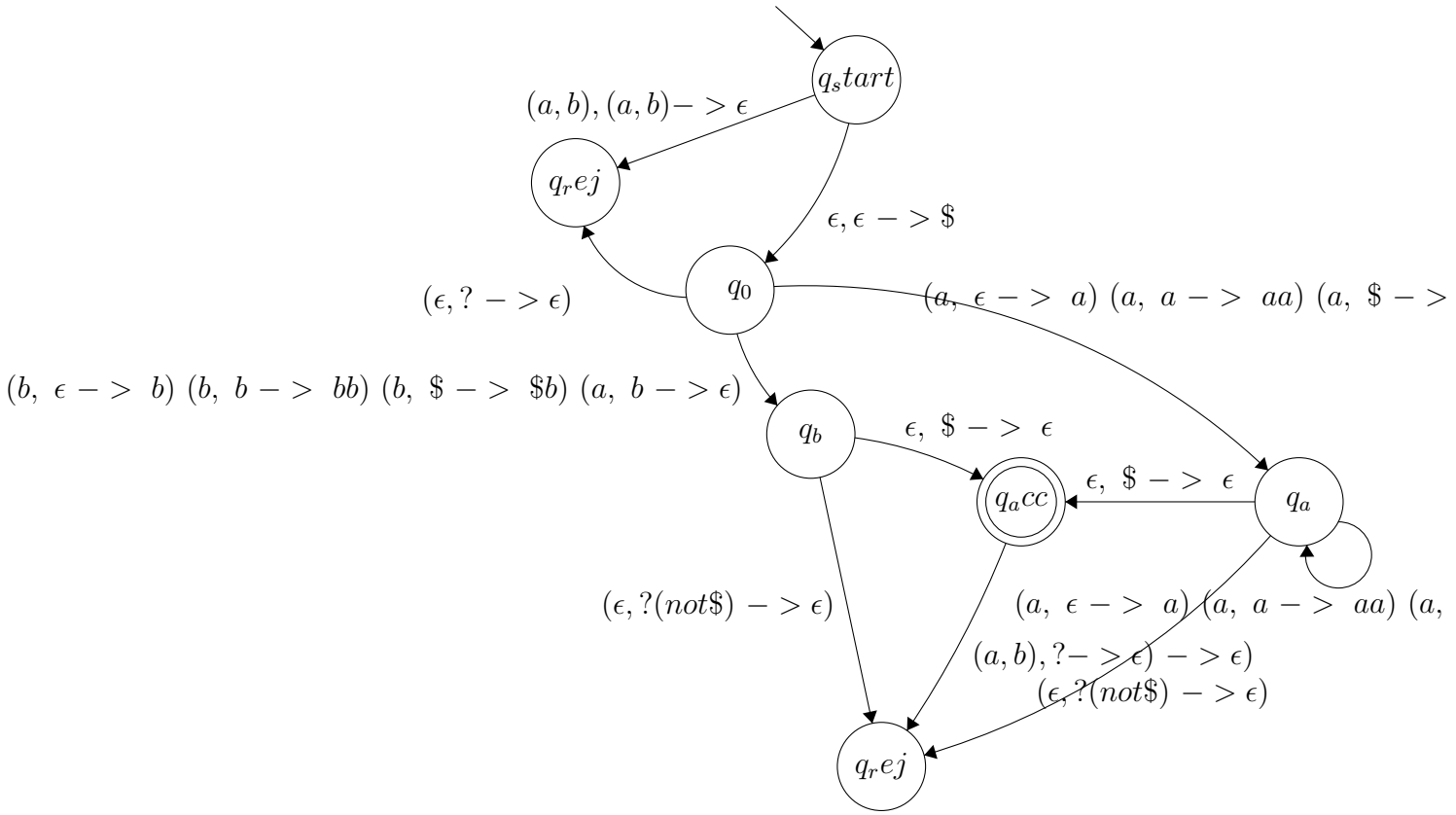
(c)

The DPDA which recognizes L(G) begins by writing a "start character" to the stack. It then simply writes either an 'a' or 'b' onto the stack (whatever it receives first) and then if

at any time it gets another of the same character it writes another instance of the character onto the stack. If at any time it gets the other character in the alphabet of L(G), then it pops one character from the stack. If the string ends when the "start character" is on top of the stack, it accepts the input. Otherwise, it rejects the input.

below is a diagram of such a PDA. the ? is a wild card character meaning anything. the not $ means any character that is not the $. there should also be a looping circle from qb to qb which has the same rules as q0 to qb. the arrows from q0 to qa and qa to qa are similar to that from q0 to qb but with a's instead of b's and vice versa. They were there but went off the screen. There should also be a rule from q0 to qaCC– $\epsilon, \$ - - > \epsilon$

In the picture there are also two qrej states both depicting one collective garbage state.



---

4. IN order to Prove that the language in question 2.57 is not a DCFL. we will utilize a proof by contradiction

   Let L_1 = the language in 2.57

   Assume L_1 is a DCFL

   If L_1 is a DCFL, then there is a DPDA which accepts it. but there is no DPDA which can accept it because when an automata reads some word w in L in, as it reaches the b's and begins scanning them in, it must guess whether to begin popping a's to match b's in

order to accept a word from L_1. but if it begins popping a's to match b's then it will not be able to pop a's to match c's to accept a different word in L_1 in which there are the same number of c's as a's. Therefore without nondeterminism it is impossible to build an automata which can accept L_1. But if there is no Deterministic PDA which accepts this language, then this language cannot be a DCFL. and there in lies a contradiction. Thus, we have proved L_1 is not a DCFL as required.

---

5. IN order to Prove that the language in question 2.58 is not a DCFL. we will utilize a proof by contradiction

   Let L_1 = the language in 2.58

   Assume L_1 is a DCFL

   If L_1 is a DCFL, then there is a DPDA which accepts it. but there is no DPDA which can accept it because when an automata reads some word w in L in, it does not know the length of w. Since it does not know the length of w it is impossible for it to know the midpoint in order to begin popping characters from the stack as they match w^r. In order for any automata to recognize a palindrome, it must use non-determinism to guess that each character just scanned as input is the midpoint of the string of input. Therefore without nondeterminism it is impossible to build an automata which can accept L_1. But if there is no Deterministic PDA which accepts this language, then this language cannot be a DCFL. and there in lies a contradiction. Thus, we have proved L_1 is not a DCFL as required.