2.25 For any language A, let $SUFFIX(A) = \{v \mid uv \in A \text{ for some string } u\}$. Show that the class of context-free languages is closed under the $SUFFIX$ operation.

Proof: We give two ways to prove the claim. One is by constructing a PDA that can accept suffixes of A, the other way is by designing a CFG that can generate suffixes of A. Both proofs are listed below; either is good enough to prove the claim.

1. We can prove the claim by showing there is a PDA M which can accept the suffixes of A.

   Because A is context-free, there must be a PDA accepting A, denoted as M1. Then, we do the following:

   (1) Make a copy of M1, and denote it as M2. M2 has exactly the same states, stack, and transitions as M1. M1 and M2 together will form the whole PDA, which is M.
   (2) Change the input part of transitions in M2 from what it was to ε, while keeping the stack unchanged. For example, if the original transition was $a, b \to \varepsilon$, we change it to $\varepsilon, b \to \varepsilon$.
   (3) For each state in M2, add a transition $\varepsilon, \varepsilon \to \varepsilon$ to the corresponding state in M1.
   (4) Let the start of M2 be the start state of the whole PDA.

   We can see that M will construct the stack appropriately and ignore the input. At some point of time, M nondeterministically transits from the state in M2 to the corresponding state in M1, then starts to take the first character of the suffix as the input, and makes transition in M1 from then on. Hence, all the suffixes of the string belongs to A will be accepted by M.

2. We can prove the claim by showing there is a CFG which can generate the suffixes of A.

   Because A is a CFL, there is a context-free grammar in Chomsky normal form, denoted as G, that generates A. We form a new CFG G' that can generate $SUFFIX(A)$. In order to do this, we modify G in the following way:

   (1) For each variable X of G, add a new variable X'. (X' will generate the suffixes of the language that is generated by using X as the start variable.)

   (2) Whenever $X \to YZ$ is a rule of G, we add two new rules, $X' \to Y'Z$ and $X' \to Z'$.

   (3) Also, for each variable in G, we add $X' \to X$ and $X' \to \varepsilon$

   (4) If S is the start variable of G, we let S' be the start variable of G'.

   Now the new CFG G' can generate the suffix of A, and hence context-free language is closed under the suffix operation.

3.1 Solution:

Here we use $\square$ to represent blank.

(a) 0.

$q_1 0$, $\square q_2 \square$, $\square\square q_{accept}$

(c) 000.

$q_1 000$, $\square q_2 00$, $\square \times q_3 0$, $\square \times 0 q_4 \square$, $\square \times 0 \square q_{reject}$

(d) 000000.

$q_1 000000$, $\square q_2 00000$, $\square \times q_3 0000$, $\square \times 0 q_4 000$, $\square \times 0 \times q_3 00$, $\square \times 0 \times 0 q_4 0$, $\square \times 0 \times 0 \times q_3 \square$, $\square \times 0 \times 0 q_5 \times \square$, $\square \times 0 \times q_5 0 \times \square$, $\square \times 0 q_5 \times 0 \times \square$, $\square \times q_5 0 \times 0 \times \square$, $\square q_5 \times 0 \times 0 \times \square$, $q_5 \square \times 0 \times 0 \times \square$, $\square q_2 \times 0 \times 0 \times \square$, $\square \times q_2 0 \times 0 \times \square$, $\square \times \times q_3 \times 0 \times \square$, $\square \times \times \times q_3 0 \times \square$, $\square \times \times \times 0 q_4 \times \square$, $\square \times \times \times 0 \times q_4 \square$, $\square \times \times \times 0 \times \square q_{reject}$

3.2 Solution:

(b) 1#1

$q_1 1\#1$, $\times q_3 \#1$, $\times \# q_5 1$, $\times q_6 \# \times$, $q_7 \times \# \times$, $\times q_1 \# \times$, $\times \# q_8 \times$, $\times \# \times q_8 \square$, $\times \# \times \square q_{accept}$

(c) 1##1

$q_1 1\#\#1$, $\times q_3 \#\#1$, $\times \# q_5 \#1$, $\times \#\# q_{reject} 1$

(d) 10#11

$q_1 10\#11$, $\times q_3 0\#11$, $\times 0 q_3 \#11$, $\times 0 \# q_5 11$, $\times 0 q_6 \# \times 1$, $\times q_7 0 \# \times 1$, $q_7 \times 0 \# \times 1$, $\times q_1 0 \# \times 1$, $\times \times q_2 \# \times 1$, $\times \times \# q_4 \times 1$, $\times \times \# \times q_4 1$, $\times \times \# \times 1 q_{reject} \square$

(e) 10#10

$q_1 10\#10$, $\times q_3 0\#10$, $\times 0 q_3 \#10$, $\times 0 \# q_5 10$, $\times 0 q_6 \# \times 0$, $\times q_7 0 \# \times 0$, $q_7 \times 0 \# \times 0$, $\times q_1 0 \# \times 0$, $\times \times q_2 \# \times 0$, $\times \times \# q_4 \times 0$, $\times \times \# \times q_4 0$, $\times \times \# q_6 \times \times$, $\times \times q_6 \# \times \times$, $\times q_7 \times \# \times \times$, $\times \times q_1 \# \times \times$, $\times \times \# q_8 \times \times$, $\times \times \# \times q_8 \times$, $\times \times \# \times \times q_8 \square$, $\times \times \# \times \times \square q_{accept}$

3.6 Solution:

If we use this simpler algorithm and come across a string s_i that will make the Turing Machine M loop (neither accept nor reject), then M will loop infinitely and hence we never enumerate the other strings. This implies that the simpler algorithm cannot construct the enumerator that enumerates the language accepted by M.

3.9 (a) Show that 2-PDAs are more powerful than 1-PDAs.

Proof:

(1) Clearly, 2-PDAs are as good as 1-PDAs.

(2) Now we give an example that only 2-PDA can accept, in order to show that 2-PDAs are more powerful.

Consider the language $L = \{a^n b^n c^n \mid n \geq 0\}$, it is already proved in Example 2.36 that L is not context-free. Hence, L cannot be accepted by 1-PDAs.

However, we can construct a 2-PDA M in the following way such that it can accept L. Every time M reads in an 'a', it pushes 'a' onto both of the two stacks. After M begins to read in 'b', no more 'a' is allowed in the input. When M reads in a 'b', it pops an 'a' from the first stack only. Similarly, after M begins to read in 'c', no more 'a' or 'b' is allowed in the input, and when M reads in a 'c', it pops an 'a' from the second stack only. If by the end of the input, both of the stacks are empty, M accepts the string. Hence, in this way, L can be accepted by this 2-PDA, which implies that 2-PDAs are more powerful than 1-PDAs.

3.15 Proof:

b) concatenation

For any two decidable languages L_1 and L_2, let M_1 and M_2 be the deterministic TMs that decide them. We construct a 2-tape nondeterministic TM M' that decides $L_1 L_2$.

"On input *w:*

1. Scan the input tape (tape 1) from left to right until a blank is encountered. For each tape cell read, nondeterministically choose either to write the same symbol, or to write the symbol with a mark on it. Make sure that exactly one tape symbol receives a mark.
2. Copy all symbols from the first tape up to and including the marked symbol onto the second tape, removing the mark on the last symbol.

3. Run M_1 on the contents of the second tape. If it rejects, then $M^{'}$ rejects. If it accepts, continue to step 4.

4. Blank out the second tape, and then copy all symbols from the first tape following the marked symbol and through to the first blank onto the second tape.

5. Run M_2 on the second tape. If it accepts, then $M^{'}$ accepts. If it rejects, then $M^{'}$ rejects."

c) Star:

Suppose that *M* is a deterministic 1-tape TM that decides *L*. We form a nondeterministic 2-tape TM *M'* that decides *L**.

"On input *w:*

Scan the input tape (tape 1) from left to right until a blank is encountered. For each tape cell read, nondeterministically choose either to write the same symbol, or to write the symbol with a mark on it. Any number of tape cells can be marked in this process. Position the tape head on the first tape at the left hand end.

While the symbol on the first tape head under the tape head is not a blank,

1. Make the second tape entirely blank. (Start at the left hand end, and move right writing blanks until the first blank is read.)

2. Copy all symbols from the first tape starting at the current position of the tape head up to and including the marked symbol onto the second tape (starting at the left hand end of the second tape), removing the mark on the last symbol. Leave the tape head on the first tape positioned on the cell right after the last (marked) symbol copied.

3. Run *M* on the contents of the second tape. If it rejects, then $M^{'}$ rejects and stops. If it accepts, continue.

End while

Accept."

d) complementation

For any decidable language L, let M be the TM that decides it. We construct a TM $M^{'}$ that decides the complementation of L:

On input w:

Run M on w. If it accepts, reject it in $M^{'}$. If it rejects, accept it in $M^{'}$.

$M^{'}$ accepts w if M rejects it, and rejects w if M accepts it.

e) intersection

For any two decidable languages L_1 and L_2, let M_1 and M_2 be the TMs that decide them. We construct a TM M' that decides the intersection of L_1 and L_2 :

On input w:

1. Run M_1 on w.
2. Run M_2 on w.

If M_1 and M_2 accept it, M' accepts, otherwise, M' rejects.

3.16 Proof:

b) concatenation

For any two Turing-recognizable languages L_1 and L_2, let M_1 and M_2 be the deterministic TMs that recognize them. We construct a 2-tape nondeterministic TM M' that recognizes L_1L_2.

"On input *w:*

Scan the input tape (tape 1) from left to right until a blank is encountered. For each tape cell read, nondeterministically choose either to write the same symbol, or to write the symbol with a mark on it. Make sure that exactly one tape symbol receives a mark.

Copy all symbols from the first tape up to and including the marked symbol onto the second tape, removing the mark on the last symbol.

Run M_1 on the contents of the second tape. If it rejects, then M' rejects. If it runs forever then M' also runs forever. If it accepts, continue to step 4.

Blank out the second tape, and then copy all symbols from the first tape following the marked symbol and through to the first blank onto the second tape.

Run M_2 on the second tape. If it accepts, then M' accepts. If it runs forever then M' also runs forever. If it rejects, then M' rejects."

By using nondeterminism to break the string up into two pieces (by placing a mark on the string), we need only consider one way to partition up the string. If we used determinism, we would have to try all possible ways. (This could be done too, but is more complicated to explain.)

c) star

Suppose that *M* is a deterministic 1-tape TM that recognizes *L*. We form a nondeterministic 2-tape TM *M'* that recognizes *L**.

"On input *w:*

Scan the input tape (tape 1) from left to right until a blank is encountered. For each tape cell read, nondeterministically choose either to write the same symbol, or to write the symbol with a mark on it. Any number of tape cells can be marked in this process. Position the tape head on the first tape at the left hand end.

While the symbol on the first tape head under the tape head is not a blank,

1. Make the second tape entirely blank. (Start at the left hand end, and move right writing blanks until the first blank is read.)

2. Copy all symbols from the first tape starting at the current position of the tape head up to and including the marked symbol onto the second tape (starting at the left hand end of the second tape), removing the mark on the last symbol. Leave the tape head on the first tape positioned on the cell right after the last (marked) symbol copied.

3. Run *M* on the contents of the second tape. If it rejects, then $M^{'}$ rejects and stops. If it runs forever, then $M^{'}$ runs forever. If it accepts, continue.

End while

Accept."

By using nondeterminism to break the string up into some number of pieces (by placing marks on the string), we need only consider one way to partition up the string. If we used determinism, we would have to try all possible ways. (This could be done too, but is more complicated to explain.)

d) intersection : For any two Turing-recognizable languages L_1 and L_2, let M_1 and M_2 be the TMs that recognize them. We construct a TM $M^{'}$ that recognizes the intersection of L_1 and L_2:

On input w:

Run M_1 and M_2 on w alternatively step by step. If both accept, $M^{'}$ accepts. If either of these two halts and rejects, $M^{'}$ rejects. Otherwise, $M^{'}$ may loop forever.