1) a) A 2-NPDA can be defined as $(Q, \Sigma, \Gamma_1, \Gamma_2, \delta, q_0, F)$ where $Q, \Sigma, \Gamma_1, \Gamma_2$ are finite states.

$Q$: set of all states

$\Sigma$: alphabet of the input tape

$\Gamma_1$: the stack alphabet from the first stack of 2-NPDA

$\Gamma_2$: the stack alphabet from the second stack

$\delta: Q \times \Sigma_\epsilon \times \Gamma_{1_\epsilon} \times \Gamma_{2_\epsilon} \to P(Q \times \Gamma_{1_\epsilon} \times \Gamma_{2_\epsilon})$ | $\delta$ represents the transition function

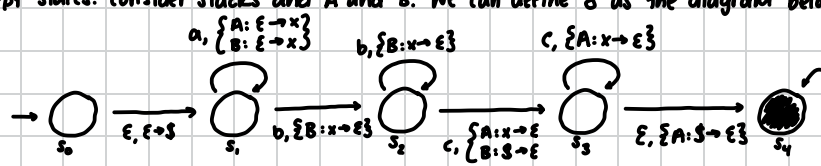
$q_0 \in Q$ is the start state

$F \subseteq Q$ is the subset of accept states

A 2-NPDA, which we will call M, computes as follows: it accepts input w if w can be written as $w = w_1 w_2 w_3 \ldots$ where $w_i \in \Sigma_\epsilon$ and sequences of states $r_0, r_1, r_2, \ldots r_n \in Q$, strings $s_0, s_1, s_2 \ldots s_n \in \Gamma_1^*$ and $p_0, p_1, p_2 \ldots p_n \in \Gamma_2^*$ exist that satisfy the following conditions ($s_i$ and $p_i$ represent the sequence of stack contents that M has on the accepting branch of the computation):

1. $r_0 = q_0$ and $s_0 = \epsilon$ and $p_0 = \epsilon$. This indicates that M starts in the start state with an empty queue.
2. for $i = 0, \ldots m-1$ we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_{1_\epsilon}$ and $t \in \Gamma_1^*$.
3. for $i = 0, \ldots, m-1$ we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where $p_i = at$ and $p_{i+1} = bt$ for some $a, b \in \Gamma_{2_\epsilon}$ and $t \in \Gamma_2^*$.
4. $r_m \in F$. This condition says that an accept state occurs at the end of the input.

b) Consider the 2-NPDA where $M = (Q, \Sigma, \Gamma_A, \Gamma_B, \delta, q_0, F)$ where $Q = s_0, s_1, s_2, s_3, s_4$ represents all the states, $\Sigma = \{a, b, c\}$, $\Gamma_A = \{x, \$\}$, $\Gamma_B = \{x, \$\}$, $q_0 = \{s_0\}$ and represents the start state, and $F = \{s_4\}$ represents the set of all accept states. Consider stacks A and B. We can define $\delta$ as the diagram below:




Thus we can decide the language L in the following steps as shown in the diagram. We start by pushing $ to both stacks. ∀ a read from the tape, we push x onto both stack A and B. When a b is read, we move to the next state and pop x from stack B. ∀ b read from the tape, we pop x from stack B. If the number of b's is greater than a's, the string will die. Otherwise when a c is read, we move to the next state and pop x from stack A and $ from stack B. If the number of a's > number of b's, the string will die as B cannot pop the $. Next, ∀ c read from the tape, we pop from A. If there are more c's than a's, the string will die. Additionally, if stack A and B only have $ and the input tape is empty, we pop $ and move to the accept state. Notice that if the a's, b's, and c's aren't in order, the 2-NPDA won't accept since each state only reads specific letters. ∴ we have constructed a 2-NPDA that decides $L = \{a^n b^n c^n : n \geq 0\}$.

1) c) We want to prove that 2-NPDA's are equivalent to Turing machines. We will prove the forward direction, which is that if a language is decided by a 2-NPDA, then it is decided by a Turing machine. We can do this by showing that an arbitrary 2-NPDA can be written as a Turing machine. We will start by defining a 2-NPDA where $M = (Q, \Sigma, \Gamma_1, \Gamma_2, \delta, q_0, F)$ with

$Q$ as the set of all states

$\Sigma$ is the tape alphabet

$\Gamma_1$ and $\Gamma_2$ are the stack alphabets for the two stacks $S_1$ and $S_2$

$\delta : Q \times \Sigma_\varepsilon \times \Gamma_{1_\varepsilon} \times \Gamma_{2_\varepsilon} \to P(Q \times \Gamma_{1_\varepsilon} \times \Gamma_{2_\varepsilon})$ is the transition function

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of all accept states

Given an input $w$, a 2-NPDA reads from the tape and pushes or pops from the stacks based on transition functions. Now we will define a similar Turing machine. From the theorem mentioned in class, every multitape Turing machine has an equivalent single tape. We can define a multitape Turing machine $M'$ with 3 tapes that has the same function as $M$.

$M' = (Q', \Sigma', \Gamma', \delta', q_0, q_{accept}, q_{reject})$.

$Q'$ is the set of all states

$\Sigma' = \Sigma$ is the input alphabet

$\Gamma'$ is the tape alphabet where $B \in \Gamma'$ ($B$ is the blank symbol) and $\Gamma_1 \cup \Gamma_2 \subseteq \Gamma'$

$\delta' = Q' \times \Sigma' \times \Gamma' \to P(Q' \times \Sigma' \times \Gamma' \times \{R, L\})$

$q_0' \in Q'$ is the start state

$q_{accept} \in Q'$ is the accept state

$q_{reject} \in Q'$ is the reject state, where $q_{reject} \neq q_{accept}$

We then start by initializing a tape in the Turing machine that replicates the input tape of the 2-NPDA by reading the input string and writing right until the entire input string is copied down. We then create two more tapes that correspond to the two stacks $S_1$ and $S_2$ of $M$. Each tape will push the starting symbol # first.

If we push to $S_1$, then the corresponding tape 1 will write onto tape 1 and move the tape head right (same thing for $S_2$ if we push to $S_2$ then move right and write on tape 2). If we pop from $S_1$, the corresponding tape 1 will move the tape head to the right until it finds the first empty string or $B$, moving the tape head left and writing $B$ over the character (same for $S_2$ and tape 2). Note that if we read #, we do not move left as that is the bottom of the stack. ∴ we have a Turing machine with 3 tapes that decides $L$ where $L$ is decided by a 2-NPDA.

Next we will prove backwards that if a language $L$ is decided by a Turing machine, then it is decided by a 2-NPDA. We can construct an arbitrary Turing machine with:

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

$Q$ is the set of states of the Turing machine

$\Sigma$ is the input alphabet and $B \notin \Sigma$

→

1) c) $\Gamma$ is the tape alphabet where $B \in \Gamma$ and $\Sigma \subseteq \Gamma$

$\delta = Q \times \Sigma \times \Gamma \rightarrow P(Q \times \Sigma \times \Gamma \times \{R, L\})$

$q_0 \in Q$ is start state

$q_{accept}$ is accept state and $q_{reject}$ is reject state where $q_{reject} \neq q_{accept}$

We will construct a 2-NPDA with the same function as M. We can define a 2-NPDA with

$M' = (Q', \Sigma', \Gamma_1, \Gamma_2, \delta', q_0', F)$ with

$Q'$ is set of states

$\Sigma'$ is the tape alphabet

$\Gamma_1$ and $\Gamma_2$ are the stack alphabets for $S_1$ and $S_2$ where $B, \$ \in \Gamma_1$ and $\$ \in \Gamma_2$

$\delta' = Q' \times \Sigma'_\epsilon \times \Gamma_{1_\epsilon} \times \Gamma_{2_\epsilon} \rightarrow P(Q' \times \Gamma_{1_\epsilon} \times \Gamma_{2_\epsilon})$ is the transition function

$q_0' \in Q'$ is start state

$F \subseteq Q'$ is set of accept states

We will replicate a Turing machine with a 2-NPDA by letting our 2 stacks R and L represent the right and left of the turing tape head. We will start by pushing $ onto both stacks and pushing each character of the input tape onto stack R. When the turing machine's tape head moves right, we can imitate this with the 2-NPDA M' by pushing the character read onto stack L and popping from R ($ should not be popped from either stack). Since turing tapes are infinite, the tapehead can move right infinitely reading blanks. ∴ if a tapehead moves right moving blanks, we can replicate this by pushing empty symbols B onto stack L to keep track of where the turing head is in reference to our 2-NPDA. When the tape head moves left, we can replicate by pushing the symbol read onto stack R and popping the top character from L. Recall that since turing machine tapes are infinite, if we are moving left and reading blanks we will not push these blanks onto R and only pop from L. If the tape head writes to the right, we push the new written symbol onto L and pop the top character from R and vice-versa if the tape head is moving left. Thus we have constructed a 2-NPDA that decides L where L is decided by a turing machine. ∴ we have proven that L is decided by a 2-NPDA iff it is decided by a Turing machine.

2) a) A queue automaton is defined as $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where $Q, \Sigma, \Gamma$ are all finite states and

Q represents the set of all states

$\Sigma$ represents the input alphabet or tape alphabet

$\Gamma$ represents the queue alphabet

$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow P(Q \times \Gamma_\epsilon)$ is the transition function

$q_0 \in Q$ is start state

$F \subseteq Q$ is the set of accept states

A queue automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows: it accepts input w if w can be written as $w = w_1 w_2 w_3 \dots w_n$ where $w_i \in \Sigma_\epsilon$ and sequences of states $r_0, r_1, r_2 \dots r_n \in Q$ and strings $s_0, s_1, s_2, \dots s_n \in \Gamma^*$ exist that satisfy the following conditions (strings $s_i$ represent the sequence of queue contents that M has on the accepting branch):

→

2) a) 1. $r_0 = q_0$ and $s_0 = \epsilon$. This indicates M starts in the start state with an empty queue.

2. for $i = 0, \ldots m-1$ we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where $s_i = ta$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\epsilon$ and $t \in \Gamma^*$

3. $r_m \in F$. This says that an accept state occurs at the end of the input

b) we will start $\Rightarrow$ by showing that if a language L is decided by a queue automaton then it is decided by a Turing Machine. We will define an arbitrary queue automaton where $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

Q represents the set of all states

$\Sigma$ represents the input alphabet or tape alphabet

$\Gamma$ represents the queue alphabet

$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow P(Q \times \Gamma_t)$ is the transition function

$q_0 \in Q$ is start state

$F \subseteq Q$ is the set of accept states

Given some input string w, the queue automaton reads from the input tape and enqueues or dequeues symbols onto the queue based on transition functions. We will construct a Turing Machine M' with the same function as M. Every multitape turing machine has an equivalent single tape, so we can define a multitape turing machine M' that has two tapes with the same function as M.

Let $M' = (Q', \Sigma', \Gamma', \delta', q_0', q_{accept}, q_{reject})$ where

Q' is the set of states in the turing machine

$\Sigma' = \Sigma$ is the input alphabet and $B \notin \Sigma'$

$\Gamma'$ is the tape alphabet where $B \in \Gamma'$ and $\Sigma' \subseteq \Gamma'$

$\delta' = Q' \times \Sigma' \times \Gamma' \rightarrow P(Q' \times \Sigma' \times \Gamma' \times \{R, L\})$

$q_0' \in Q'$ is the start state

$q_{accept} \in Q'$ is the accept state

$q_{reject} \in Q'$ is reject state; $q_{rej} \neq q_{acc}$

we will have two tapes: the first will replicate the input tape of the queue automaton by reading the input string and writing right until the entire input string is copied down and the second tape will replicate the queue with the following transitions $\delta'$ (the operations will only be performed on the second tape):

1. we start by adding a start symbol to the second turing tape or B.
2. To imitate dequeue, we will move the turing tapehead left until we reach a blank or B, we then move the tape head right and write over the first non-blank character with B, then move the tape-head right. If there is no non-blank symbol, then we do nothing.
3. To imitate enqueue, we will move the turing tape head right until we reach the first B in which we will write over the blank with the enqueue symbol.

$\therefore$ we are able to construct a Turing Machine that recognizes the same language that a queue automaton recognizes.

$\rightarrow$

2) b) An example for the process in a turing machine:

input tape: B a b c B —(dequeue a)→ tape 2: B B b c B —(enqueue a d)→ tape 3: B B b c d

For the backward direction ⇐, if a language $L$ is decided by a Turing Machine, it is determined by a queue automaton. We can define $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$:

$Q$ is set of states in the turing machine

$\Sigma$ is the input alphabet and $B \notin \Sigma$

$\Gamma$ is the tape alphabet where $B \in \Gamma$ and $\Sigma \subset \Gamma$

$\delta : Q \times \Sigma \times \Gamma \to P(Q \times \Sigma \times \Gamma \times \{R, L\})$

$q_0 \in Q$ is start state

$q_{acc}$ : accept state and $q_{rej}$ : reject state, where $q_{rej} \neq q_{acc}$

We will construct a queue automaton with the same functions. Consider queue automaton $M' = (Q', \Sigma', \Gamma', \delta', q_0', F)$:

$Q'$ is set of all states

$\Sigma'$ is the tape alphabet

$\Gamma'$ is the queue alphabet

$\delta' : Q' \times \Sigma_\epsilon' \times \Gamma_\epsilon' \to P(Q' \times \Gamma_\epsilon')$ is the transition function

$q_0' \in Q'$ is the start state

$F \subseteq Q'$ is the set of accept states

We can replicate a turing machine with a queue automaton using the following transitions $\delta'$: we will construct our queue | it will always have the tape of the Turing Machine and we let \$ represent where the tape head is. ∴ a queue that looks like ab\$cd where $c \in \Sigma$ and $a, b, d \in \Sigma^*$ indicates that the tapehead is pointing at c. We initialize the queue by enqueueing \$ and then enqueueing the string input. We imitate moving right and possibly writing by cyclically shifting the \$ right. To do this, we define a state $S_n \in Q'$ that allows us to recall the last symbol read | in this state, each symbol is remembered as a 2-tuple of $(i, j)$ where $i$ is the previous symbol (symbol to the left) of $j$. This is possible since we know the tape alphabet and the alphabet is finite. ∴ we have the steps:

1. enqueue a # to the end of our queue
2. encode each symbol starting at the front of queue as a 2-tuple of the previous and current head in $S_n$
3. dequeue each tuple $(i, j)$ and enqueue i until we read a tuple $(i, j)$ with $i$ = \$
4. If the turing head is moving right, dequeue tuple (\$, j) and enqueue j from the tuple (\$, j) then enqueue \$. If we are writing, we dequeue the tuple (\$, j) and enqueue the new symbol then enqueue \$.
5. After this, we continue to dequeue $(i, j)$ and enqueue $j$ until we reach $(i, \#)$
6. Once we reach $(i, \#)$, dequeue the tuple and arrive at our new queue →

2) b) Now we can replicate a turing machine's tapehead moving left and possibly writing. In order to do this, we define a state $s_m \in Q'$ that allows us to recall the last symbol read | in this state each symbol is remembered as a 2-tuple $(i, j)$ where $i$ is the previous symbol (symbol to the left of $j$). This is possible since we know the tape alphabet and it is finite. We have the steps:

1. Start by enqueuing a # to the end of the queue
2. Encode each character starting at the current head as a 2-tuple of the previous and current symbol in state $s_m$
3. dequeue 2-tuples $(i,j)$ and enqueue $i$ until we read a \$ as the current symbol (the second character of the tuple or something in the form $(i,\$)$ for some value $i$)
4. dequeue $(i,\$)$ and enqueue a \$ then enqueue the previous symbol $i$ associated with the \$ which we remember
5. if the turing machine is <u>Moving</u> left, we dequeue the tuple that looks like $(\$, i)$ and enqueue $i$. If the turing machine is <u>writing</u> left, we dequeue the the tuple that looks like $(\$, i)$ for an arbitrary $i$ and enqueue the new symbol.
6. continue to dequeue 2-tuples $(i,j)$ and enqueue $j$ until we reach the last tuple, which is in form $(i, \#)$ and we dequeue it. We should then reach our new queue.

$\therefore$ We have constructed a queue automaton that recognizes the same language that a Turing machine recognizes. $\Rightarrow$ $L$ is decided by a queue automaton iff it is decided by a Turing machine.

3) We want to prove that a language is RE iff it can be expressed as $L = \{x:$ there exists $y$ for which $\langle x,y\rangle \in R\}$ where $R$ is a decidable language. In the forward direction $\Rightarrow$ if $L$ is RE, then it can be expressed as $L = \{x:$ there exists $y$ for which $\langle x,y\rangle \in R\}$. Since $L$ is RE, it is recognized by a Turing machine. We know that a language is Turing recognizable iff some enumerator enumerates it. $\therefore$ there is an enumerator $E$ that writes out a sequence of strings that are in $L$. We can define $R = \{\langle x,y\rangle :$ where $x$ is the string printed by $E$ after running $y$ times$\}$. Note that $y$ is a string but treated like an integer. Thus $R$ is decidable since it returns true if $x$ is in $R$ after $y$ steps and false if $x$ is not in $R$ after $y$ steps. Notice that $L$ is the set of strings $x$ where $\exists$ some $y \mid \langle x,y\rangle \in R$ and since $x$ is written by $E$, then $x \in L$. $\therefore L = \{x: \exists\ y$ for which $\langle x,y\rangle \in R\}$.

$\Leftarrow$ $L = \{x: \exists\ y$ for which $\langle x,y\rangle \in R\}$, then $L$ is RE and recognized by a turing machine. Consider some input string $x \in L$. $E$ will be our enumerator of the turing machine that prints strings that are recognized by the turing machine. We can define $R = \{\langle x,y\rangle : x$ is the string printed by $E$ after $E$ runs $y$ times$\}$. Note $y$ is a string that is treated like an integer. Again $R$ is decidable since it returns true if $x$ is in $R$ after $y$ steps and false if $x$ is not in $R$ after $y$ steps. With the input string $x$, we can test every $y$ in lexicographic order to see if $\langle x,y\rangle \in R$. If we find a $y$ that satisfies $\langle x,y\rangle \in R$, then accept. Else our enumerator continues printing strings to an upper bound $y \leq m$ in which we reject $x$. $\therefore$ the set of accepted strings by the turing machine are exactly $L$. $L$ is RE and recognized by a turing machine. Thus, a language is RE iff it can be expressed as $L = \{x:$ there exists $y$ for which $\langle x,y\rangle \in R\}$ ($R$ is decidable).