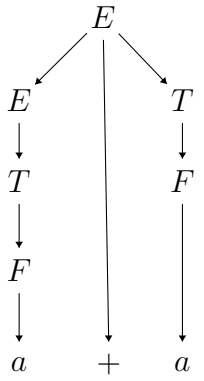
Jason Downing
Email: jason_downing@student.uml.edu
Foundations of Computer Science
Homework #3 - Chapter 2
10/31/2016

2.1 Recall the CFC G4 that we gave in Example 2.4.

a. Parse tree for a:

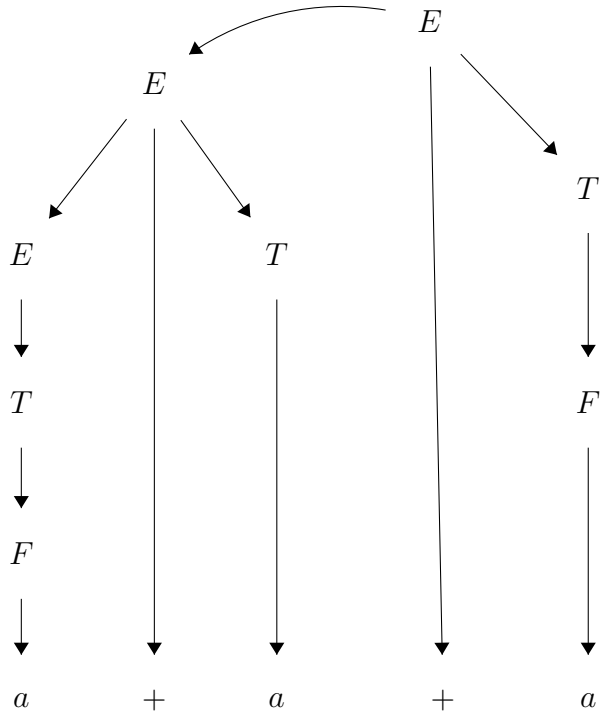
Derivation for a: $E \to T \to F \to a$

b. Parse tree for $a + a$:



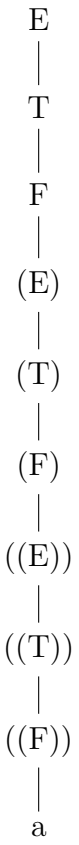
Derivation for $a + a$: $E \to E + T \to T + T \to T + F \to F + a \to a + a$

c. Parse tree for $a + a + a$:



Derivation for $a + a + a$: $E \to E + T \to T + T + T \to F + T + F \to F + F + T \to F + F + F \to a + F + F \to a + a + F \to a + a + a$

d. Parse tree for $((a))$:



Derivation for $((a))$: $E \to T \to F \to (E) \to (T) \to (F) \to ((E))$
$\to ((T)) \to ((F)) \to ((a))$

2.4
Give context-free grammars that generate the following languages.
In all parts, the alphabet $\sum$ is $\{0, 1\}$.

b. {w— w starts and ends with the same symbol}
$S \to 0P0|1P1|\epsilon$
$P \to 0P|1P|\epsilon$

c. {w— the length of w is odd}
$S \to 0|1|00S|01S|10S|11S$

e. {w— w = w^{R}, that is, w is a palindrome}
$S \to 0|1|0S0|1S1|\epsilon$

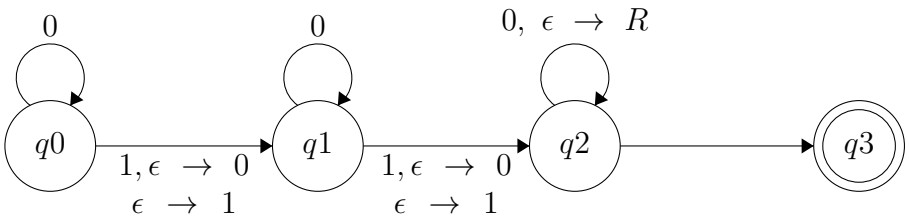
f. The empty set
$S \to \epsilon$

2.5 Give informal descriptions and state diagrams of pushdown automata for the languages in Exercise 2.4.

a. {w— w contains at least three 1s}

Informal: In this grammar, there is no need of a stack (as seen in the diagram). In the grammar only the input is read, and the input is only accepted when there is at least three 1's. The only time the machine will accept a string in this case is when at least three 1's are present; otherwise it is rejected.

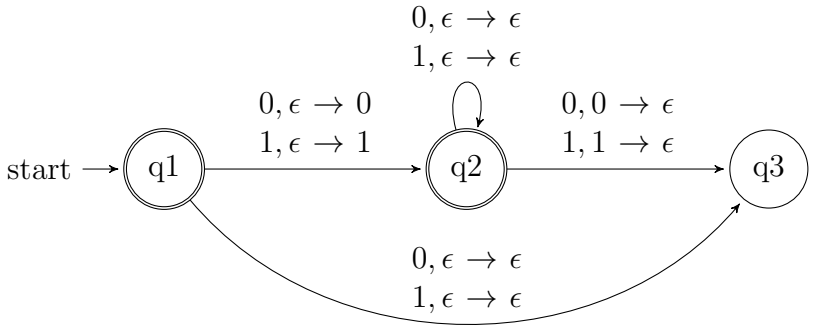
Pushdown Automata:



b. {w— w starts and ends with the same symbol}

Informal: In this grammar, there is an assumption that there is only one symbol in the string that can be accepted without the use of a stack. If there is more than one symbol in the string, then all of the symbols are put into the stack in order. We also check to see whether the last symbol is the one being read. If it is, and this symbol matches with the symbol on the stack, and if all the input has been checked with no input left, then we accept that state. Otherwise we reject the input.

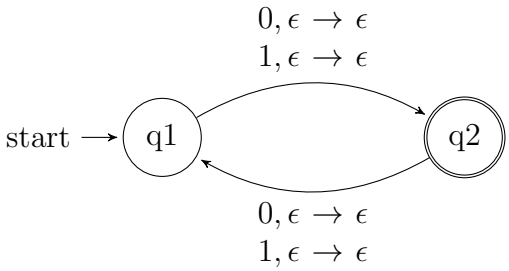
Pushdown Automata:



c. {w— the length of w is odd}

Informal: In this grammar, there is no need of a stack. In this grammar, we only read the input and this input is also only accepted when there is an odd length.

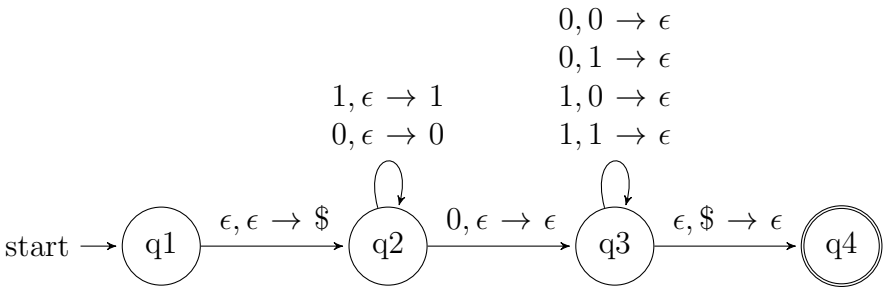
Pushdown Automata:



d. {w— the length of w is odd and its middle symbol is a 0}

Informal: In this grammar, the input is read by the pushdown automata and each symbol is placed onto the stack. At each point, the pushdown automata guesses if the input in the middle or if the next symbol read is in the middle of the string. If the next symbol read is the middle of the string then it does not get read onto the stack. After this, we pop symbols off the stack if they match the input symbols that were read. If the symbols that are popped match the symbols that were pushed earlier, and the stack is empty as the input is finished, we accept the state. Otherwise we reject the state.

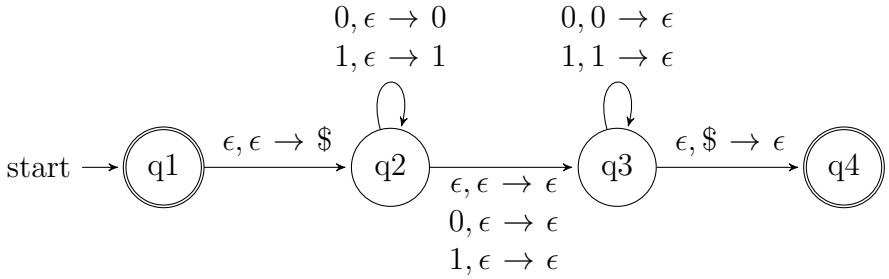
Pushdown Automata:



e. {w— w = w^{R}, that is, w is a palindrome}

Informal: In this grammar, the input is read by the pushdown automata and each symbol is placed onto the stack. At each point, the pushdown automata guesses if the input in the middle or if the next symbol read is in the middle of the string. If the next symbol read is the middle of the string then it does not get read onto the stack. After this, we pop symbols off the stack if they match the input symbols that were read. If the symbols that are popped match the symbols that were pushed earlier, and the stack is empty as the input is finished, we accept the state. Otherwise we reject the state.

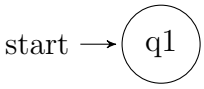
Pushdown Automata:



f. The empty set

Informal: In this grammar, the language only accepts an empty set so it implies that it does not terminate any derivation. And if none of the derivations are terminated, that means that no string is accepted by the CFG, including the empty string.

Pushdown Automata:



2.6
b. The complement of the language $\{a^n b^n \mid n >= 0\}$ The language is:
$S \rightarrow S_1 \quad | \quad S_2$
$S_1 \rightarrow aS_1b \quad | \quad T \quad | \quad U$
$S_2 \rightarrow RbaR$
$T \rightarrow aT \quad | \quad a$
$U \rightarrow Ub \quad | \quad a$
$R \rightarrow RR \quad | \quad a \quad | \quad b \quad | \quad \epsilon$

d. $\{x_1\#x_2\# \ldots \#x_k \mid k \geq 1, each\ x_i \epsilon \{a,b\}^*, and\ for\ some\ i\ and\ j,\ x_i = x_j^R\}$
The language is:
$R \rightarrow S \quad | \quad J\#S\#J \quad | \quad J\#S \quad | \quad S\#J$
$S \rightarrow aSa \quad | \quad bSb \quad | \quad \# \quad | \quad \#J\#$
$J \rightarrow aJ \quad | \quad bJ \quad | \quad \#J \quad | \quad \epsilon$

2.9 $A = \{a^i b^j c^k \mid i = j \text{ or } j = k, \text{ where } i, j, k \geq 0\}$
The context free grammar that generates the language is:

First, split the languages into two parts:
$A_1 = \{a^i b^j c^k \mid i, j, k \geq 0,\ i = j\}$
$A_2 = \{a^i b^j c^k \mid i, j, k \geq 0,\ j = k\}$

Now we can define the grammar for the language A:
$S \to S_1 \mid S_2$

In A_1, the values of i and j are equal. So, there must be an equal number of A's and B's in this language.

CGF for language A_1 is:
$S_1 \to S_1 c \mid E \mid \epsilon$
$E \to aEb \mid \epsilon$

In A_2, j and k are equal, so there are an equal number of B'c and C's.

CGF for language A_2 is:
$S_2 \to aS_2 \mid F \mid \epsilon$
$F \to bFc \mid \epsilon$

Since we can use either A_1 or A_2 to generate the string, either S_1 or S_2 can be used. Thus, the context free grammar for the language A is ambiguous.

2.10 Give an informal description of a pushdown automaton that recognizes the language A in Exercise 2.9.

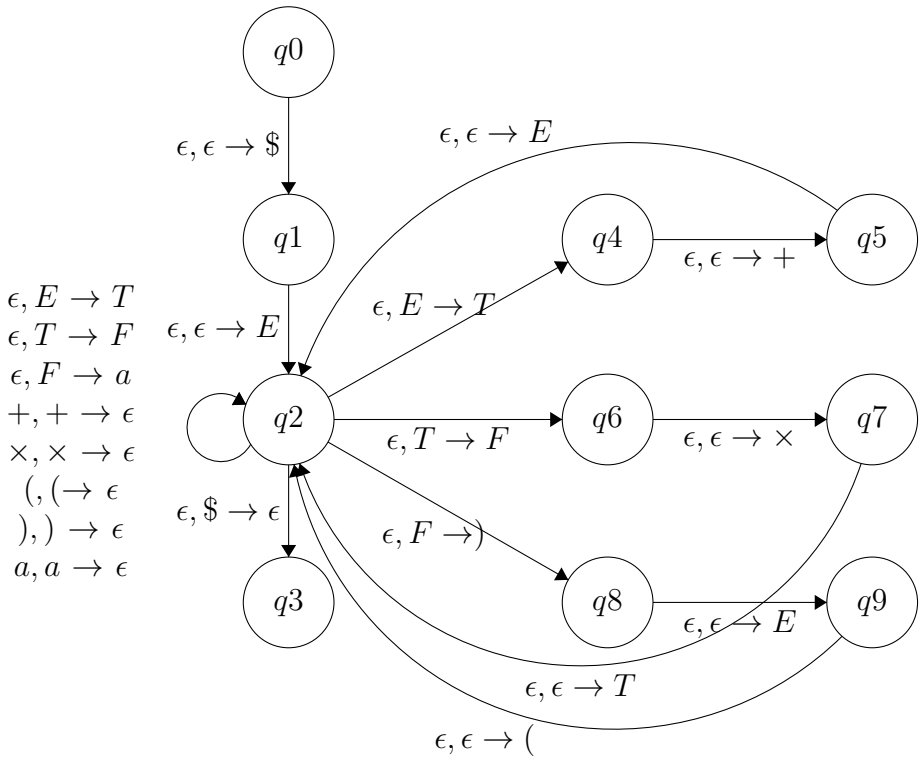
At the beginning, we break into two branches. In the first branch, for each a that is read, we push an a onto the stack. Then, we guess when the first b is read, and begin popping symbols from the stack for every b that we read. When the stack is empty, the only input that is remaining should be C, and then we read the input without adjusting the stack. If we read the symbols in the proper order, that is a followed by b followed by c, and the stack is empty by the time the B's are done reading, then we accept the state. Otherwise, we reject the state.

In the second branch, for a that we read, we do not adjust the stack. We then guess when the first b is read and we begin pushing symbols onto the stack for every b that we read. We then guess when the first c is read - and we begin popping symbols from the stack for every c that is read. If the stack becomes empty right when the input string is finished, and the symbols were read in the right order, we accept. Otherwise, we reject the state.

2.11 Convert the CFG G_4 given in Exercise 2.1 to an equivalent PDA, using the procedure given in Theorem 2.20.

Given CFG:
$E \to E + T \mid T$
$E \to T \times F \mid F$
$F \to (E) \mid a$

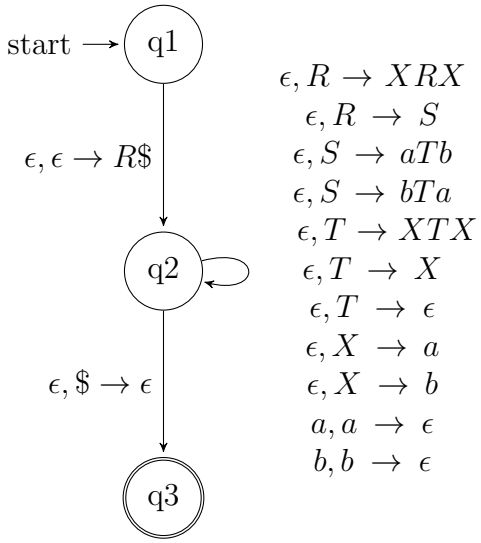
PDA for G_4:



2.12 Convert the CFG G given in Exercise 2.1 to an equivalent PDA, using the procedure given in Theorem 2.20.

CFG G given in 2.3:
$R \to XRX \mid S$
$S \to aTb \mid bTa$
$T \to XTX \mid X \mid \epsilon$
$X \to a \mid b$

PDA for G:



2.13 Let G = (V, $\sum$, R, S) be the following grammar,
V = {S, T, U}; = {0, #};

and R is the set of rules:
$S \to TT \mid U$
$T \to 0T \mid T0 \mid \#$
$U \to 0U00 \mid \#$

a) Describe L(G) in English.

$L(G)$ is the set of all strings that contain a string of 0's, and all #'s of the following forms:

1. Strings which contain exactly two #'s.
2. Strings that contain two #'s in the middle of the string, and any number of 0's on either sides of the string.
3. Strings that contain exactly one #, and the number of 0's to the left of the # are half the number of 0's that are to the right.

b) Prove that L(G) is not regular.

To prove that $L(G)$ is not regular, we will need to do a proof by contradiction that states that $L(G)$ is regular. Let us define A:

$A = L(G) \bigcup 0^*\#0^*$
We assume that $L(G)$ is regular, so A is regular too. Since A is regular, we can use pumping lemma to let p be the pumping length fo the regular language.

We can now consider the string $w = 0^p\#0^{2p}$
It seems clear that the length of $w \in A$ is greater than p. This means that $|w| > p$. By pumping lemma, we now have $w = xyz$ such that:
$|xy| \leq p, y \neq \epsilon$ and $xy^iz \in L(G)$ for all $i \geq 0$

Since A is a regular language, we can now consider all the possible ways of cutting the string w:

1. If x contains the character #, then y will be on the right side of the #. As we pump y down, it increases the number of 0's on the left so that the number of 0's on the left of the # is not equal to half of the number of 0's to the right of the #.
2. If y contains a #, then as we pump y down, it increases the number of #'s in the string.
3. If z contains a #, then y should be on the left side of the #. As we pump y down in this case, it decreases the number of 0's to the right, such that the number of 0's to the left of the # are not equal to half the number of 0's to the right of the #.

The string that results from the above cases does not belong to A. As a result, we can say that A does not satisfy the pumping lemma. Basically, A is not a regular language. This is a contradiction to the statement that we assumed at the beginning of this proof: $L(G)$ is a regular language. Thus, $L(G)$ **is not a regular language.**

2.14 Convert the following CFG into an equivalent CFG in Chomsky normal form, using the procedure given in Theorem 2.9.

$A \to BAB \mid B \mid \epsilon$
$B \to 00 \mid \epsilon$

Let's use the **Chomsky Normal Form** to solve this problem. For a CFG to be in the **Chomsky Normal Form**, it must follow the following rules:
$A \to BC$
$A \to a$

Where: a is any terminal, A, B, and C are any variables, except that B and C may not be the starting variable. We also have the rule $S \to \epsilon$, which is where S is the starting variable.

With this in mind, we can now convert the given CFG to an equivalent CFG in CNF form. We will have the above rules, as well as another starting variable and rule. The grammar would therefore start out as:
$S_0 \to A$
$A \to BAB \mid B \mid \epsilon$
$B \to 00 \mid \epsilon$

We now need to start removing all rules that contain ϵ. We will remove the rule $A \to \epsilon$ and the rule $B \to \epsilon$. This gives us the following grammar:
$S_0 \to A \mid \epsilon$
$A \to BAB \mid BA \mid AB \mid A \mid B \mid BB$
$B \to 00$

$S_0 \to A \mid \epsilon$ is acceptable since S_0 is the starting variable, and this is allowed in **Chomsky Normal Form**. We can now rule the unit rules:

Removing $A \to A$ will give us:
$S_0 \to A \mid \epsilon$
$A \to BAB \mid BA \mid AB \mid B \mid BB$
$B \to 00$

Removing $S \to B$ will give us:
$S_0 \to A \mid \epsilon$
$A \to BAB \mid BA \mid AB \mid 00 \mid BB$
$B \to 00$

Removing $S_0 \to S$ will give us:
$S_0 \to BAB \mid BA \mid AB \mid 00 \mid BB \mid \epsilon$
$A \to BAB \mid BA \mid AB \mid 00 \mid BB$
$B \to 00$

We can also replace all of the terminal 0's with a new variable, U, as such:
$S_0 \to BAB \mid BA \mid AB \mid 00 \mid BB \mid \epsilon$
$A \to BAB \mid BA \mid AB \mid 00 \mid BB$
$B \to UU$
$U \to 0$

We can now shorten the right hand side of the rules by using only 2 variables each. To shortern these rules we will replace $S_0 \to BAB$ with two new rules:
$S_0 \to BA_1$ and
$A_1 \to AB$

We can also replace the rule $A \to BAB$ with two new rules:
$A \to BA_2$ and
$A_2 \to AB$

After we replace these rules, the final CFG in **Chomsky Normal Form** is:
$G = (V, \sum, R, S_0$
set of rules $V = \{S_0, S, B, U, A_1, A_2\}$
starting variable: S_0
The set of terminals: $\sum = \{0\}$
Rules R given by:
$S_0 \to BA_1 \mid BA \mid SB \mid UU \mid BB \mid \epsilon$
$A \to BA_2 \mid BA \mid SB \mid UU \mid BB$
$B \to UU$
$U \to 0$
$A_1 \to AB$
$A_2 \to AB$

2.26
Show that if G is a CFG in **Chomsky Normal Form** form, then for any string $w \in L(G)$ of length $n \geq 1$, exactly $2n-1$ steps are required for any derivation of w.

We can prove that G is in CNF form by using the induction method on the string w of length n.

Let's do this as follows:
$N = 1$, a string "a" is of length 1 in CNF form, so derivation that is valid for this is $S \to a$ where $a \in \sum$ and S is the starting symbol.

The number of steps required can be found by doing:
$2n - 1 = 2(1) - 1$
$= 2 - 1$
$= 1$
Thus it is true that for $n = 1$ that $2n-1$ steps are required to derive a string a.

Now we can find the steps required for the case $n = k$. This would be a string of length $k \geq n$ in CNF form, so a valid derivation for this should take $2k - 1$ steps.

The number of steps required can be found by doing:
$2n - 1 = 2(k) - 1$
$= 2k - 1$

We will assume a string length of at most $k \geq n$ terminal symbols, and that it has a string length of $n = k + 1$ in CNF form.

Since we know that $n > 1$, we can consider a language that follows CNF:
$S \to BC$
$B \to *x$
$C \to *y$

Where the derivation starts with the given symbol S. The length of the string with this starting symbol S is $|w| = xy$ where $|x| > 0$ and $|y| > 0$.

We can use the inductive hypothesis now:

$$1 + (2|x| - 1) + (2|y| - 1) = 2|x| + 2|y| + 1 - 1 - 1 = 2(|x| + |y|) - 1$$

Which shows that $n = |x| + |y|$.

We've found that $B \to *x$ has a length of $|x|$ and that $C \to *y$ has a length $|y|$. Thus, it is proven that it requires $2n - 1$ steps to get the derivation of the string $w \in L(G)$ in **CNF** form.

2.30 Use the pumping lemma to show that the following languages are not context free.

a. Given language is: $\{0^n1^n0^n1^n \mid n \geq 0\}$

Let $A = \{0^n1^n0^n1^n \mid n \geq 0\}$
p will be there pumping length of A given by pumping lemma.

We can now prove that A is not a **Context-Free Language (CFL)**. In order to prove this, we will need to show that a string $s = 1^p0^p1^p0^p$ cannot be pumped. We will consider s in the form of $s = uvxyz$.

1. If both v and y contain at most one alphabet symbol type, the string will be in the form of uv^2xy^2z, which is a string of $0's$ and $1's$ of unequal length. In this case, the string s cannot be a member of A.

2. If either v or y contain more than one alphabet symbol type, the string will end up in the form of uv^2xy^2z which does not contain the symbols in the required order. In this case, the string s cannot be a member of A.

Since we cannot pump the string s without violating the conditions of pumping lemma, A is not a **CFL**.

b. $\{0^n\#0^{2n}\#0^{3n} \mid n \geq 0\}$

Let $B = \{0^n\#0^{2n}\#0^{3n} \mid n \geq 0\}$
p will be there pumping length of B given by pumping lemma.

We can now prove that B is not a **Context-Free Language (CFL)**. In order to show this, we will consider the string $s = 0^p\#0^{2p}\#0^{3p}$, and we will show that string s cannot be pumped by pumping lemma. We will consider s in the form of $s = uvxyz$.

1. Neither v nor y can contain the symbol #, otherwise xv^2wy^2z will contain more than two "#" symbols. If this string is divided into three segments of #s, then at least one of the segments 0^p, 0^{2p}, and 0^{3p} cannot be contained within either v or y.

2. Because the ratio of length between the segments is not maintained in a $1:2:3$ ratio, xv^2wy^2z cannot be contained within B. Which means that string s cannot be a member of B.

Since we cannot pump the string s without violating the conditions of pumping lemma, B is not a **CFL**.

c. $\{w\#t \mid w \text{ is a substring of } t, \text{ where } w,\ t \in \{a,b\}^*\}$

Let $C = \{w\#t \mid w \text{ is a substring of } t, \text{ where } w,\ t \in \{a,b\}^*\}$
p will be there pumping length of C given by pumping lemma.

We can now prove that C is not a **Context-Free Language (CFL)**. In order to show this, we will consider the string $s = a^pb^p\#a^pb^p$, and we will show that string s cannot be pumped by pumping lemma. We will consider s in the form of $s = uvxyz$.

1. Neither v nor y can contain a symbol #, otherwise uv^0xy^0z will not contain a #. In this case, string s cannot be a member of C.

2. If both v and y are not empty, and they both occur on the left side of the #, then the string uv^2xy^2z is no longer on the left side of the # symbol. In this case, string s cannot be a member of C.

3. We can also consider when both v and y are not empty, and they both occur on the right side of the # symbol. In this case, the string uv^0xy^0z is no longer on the right side of the # symbol. In this case, string s cannot be a member of C.

4. If just one of either v and y is not empty, then we can assume that they both occurred on the same side of the # symbol. In this case, string s cannot be a member of C.

5. The final case is that if both v and y are not empty and they both include the # symbol. Then in this case, by the third pumping lemma condition of $|vxy| \leq p$, we will end up with v consisting of $b's$ and y will consist of $a's$. This means that uv^2xy^2z contains more $b's$ on the left side of the # symbol. In this case, string s cannot be a member of C.

Since we cannot pump the string s without violating the conditions of pumping lemma, C is not a **CFL**.

d. $\{t_1\#t_2\#...\#t_k \mid k \geq 2$, each $t_i \in \{a,b\}^*$, and $t_i = t_j$ for some $i \neq j\}$

Let $D = \{t_1\#t_2\#...\#t_k \mid k \geq 2$, each $t_i \in \{a,b\}^*$, and $t_i = t_j$ for some $i \neq j\}$ p will be there pumping length of D given by pumping lemma.

We can now prove that D is not a **Context-Free Language (CFL)**. In order to show this, we will consider the string $s = a^pb^p\#a^pb^p$, and we will show that string s cannot be pumped by pumping lemma. We will consider s in the form of $s = uvxyz$.

1. Neither v nor y can contain a symbol #, otherwise uv^0xy^0z will not contain a #. In this case, string s cannot be a member of C.

2. If just either v or y is not empty, than we can assume that they both occured on the same side of the # symbol. In this case, string s cannot be a member of C.

3. If both v and y are not empty, and they both occur on the left hand side of the # symbol, then the string uv^2xy^2z is not on the left hand side of the #. In this case, string s cannot be a member of C.

4. If both v and y are not empty, and they both occur on the right hand side of the # symbol, then the string uv^0xy^0z is no longer on the right hand side of the #. In this case, string s cannot be a member of C.

5. In the final case, if both v and y are not empty and they both include the # symbol, then by the third pumping lemma condition $|vxy| \leq p$, we have v that consists of $b's$ and y that consists of $a's$. This will lead to uv^2xy^2z containing more $b's$ on the left hand side of the #. In this case, string s cannot be a member of C.

Since we cannot pump the string s without violating the conditions of pumping lemma, D is not a **CFL**.

2.31 Let B be the language of all palindromes over $\{0, 1\}$ containing equal numbers of $0s$ and $1s$. Show that B is not context free.

To solve this problem, we will assume that B is a **Context-Free Language (CFL)**, and prove that it is not by contradiction.

p will be there pumping length of B given by pumping lemma.
We will consider a string $s = 0^p1^p1^p0^p$

We will assume that string s is a member of B, and that it has a length of p. We can show that anyway you divide the string into a form of $uvxyz$, one of the conditions of pumping lemma is violated. Pumping lemma is where $|xyz| \leq p$, and we can only place xyz into the following ways:

1. vxy fails completely in the first 0^p.
If we pump v and y, then the string that is created is no longer a valid palindrome, and the number of $0s$ will be greater than the number of $1s$. This is a contradiction.
2. vxy fails between 0^p and 1^p.
If v only contains $0s$ while y only contains $1s$, then if we pump string s, the new string that is created is no longer a palin drome. This is a contradiction.
3. vxy fails completely in the first 1^p.
This case is similar to **1**, after we pump string s, the number of $1s$ will be greater than the number of $0s$, which is a contradiction.
4. vxy falls between 1^p and 1^p
After pumping the string s, the number of $1s$ will be greater than the number of $0s$, which is a contradiction.
5. vxy falls completely within the second 1^p
This is the same case as **3**.
6. vxy falls between 1^p and 0^p
This is similar to case **2**, v will only contain $1s$ while y will only contain $0s$. If we pump the string s then the new string will no longer be a palindrome. This is a contradiction.
7. vxy falls completely within the second 0^p
This is the same case as **1**.

As shown above, B is not a **CFL**.

2.32 Let $\sum = \{1, 2, 3, 4\}$ and $C = \{w \in \sum^* \mid$ in w, the number of 1s equals the number of 2s, and the number of 3s equals the number of 4s$\}$.
Show that C is not a **context free language (CFL).**

We can consider the following language:
$C = \{w \in \{0, 1, 2, 3, 4\}^* |$ in w the number of $1's$ and the number of $2's$ are equal, and the number of $3's$ and the number of $4's$ are equal. $\}$

Let us consider that C is context free. Which means that C would have a pumping length of p.

Let's use the string $s = 1^p 3^p 2^p 4^p \in C$ with $|s| > p$ as an example. There should be a string such as $uvxyz$ such that the following are true:
1. $uv^i xy^i z \in C$ for all $i \neg 0$.
2. $vy > 0$
3. $vxy \leq p$

Now let's prove all of these cases y contradiction, disregarding the value of $uvxyz$.
Case 1. If vxy contains a 1, then $uv^2xy^2z \notin C$, as it cannot be the same number of $1's$ and $2's$. Due to condition **3**, vxy cannot contain any $2's$.

Case 2. If vxy contains a 2, then $uv^2xy^2z \notin C$, as it cannot have the same number of $1's$ and $2's$. Due to condition **3**, vxy cannot contain any $1's$.

Case 3. If vxy contains a 3, then $uv^2xy^2z \notin C$, as it cannot have the same number of $3's$ and $4's$. Due to condition **3**, vxy cannot contain any $4's$.

Case 4. If vxy contains a 4, then $uv^2xy^2z \notin C$, as it cannot have the same number of $3's$ and $4's$. Due to condition **3**, vxy cannot contain any $3's$.

As we can see from condition **2**, this contradicts condition **1** in every case shown which proves that C is not a **CFL**.

2.47 Let $\sum = \{0, 1\}$ and let B be the collection of strings that contain at least one 1 in their second half.
In other words, $B = \{uv \mid u \in \sum^*, v \in \sum^* 1 \sum^* \text{ and } |u| \geq |v|\}$.

a. Give a PDA that recognizes B.

To construct the PDA, which I will call M, we need to define the formal description and determine how we will create the PDA. To start out, we will need to make sure that this PDA reads the string in, and pushes the string u onto the stack. The string u will need to contain $0's$ and $1's$. After the machine has finished pushing all of the u's onto the stack, it will need to compare them with that is in v. In this case, we should note that $u \geq v$. If we find that the amount of $1's$ in the input is at least one in the second half, then we can accept the string entirely. We can do this using a non deterministic PDA (push down automata).

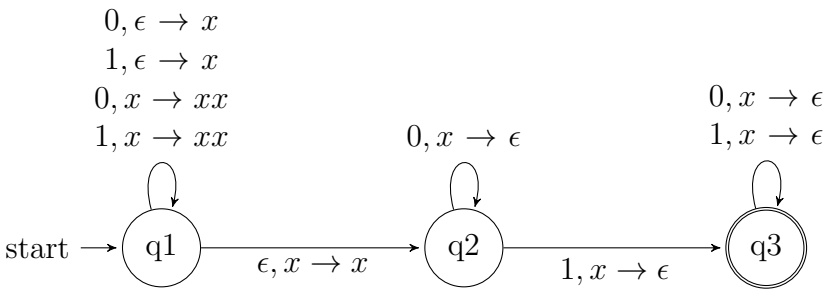
We can define the formal description of a PDA M as: Q, $\sum$, $\frac{1}{2}$, Γ, δ, q_0, F, where:

$Q = \{q_0, q_1, q_2, q_3\}$
$\sum = \{0, 1\}$
$\Gamma = \{x\}$
$F = \{q_3\}$

where the transition function, δ, is given by the following table:

Input:	0		1		ϵ	
stack	x	ϵ	x	ϵ	x	ϵ
q_0	$\{(q_0, xx)\}$	$\{(q_0, x)\}$	$\{(q_0, xx)\}$	$\{(q_0, x)\}$	$\{(q_1, x)\}$	
q_1	$\{(q_1, \epsilon)\}$		$\{(q_2, \epsilon)\}$			
q_2	$\{(q_2, \epsilon)\}$		$\{(q_2, \epsilon)\}$			

Now we can create the PDA M:



b. Give a **CFG** that generates B.

We can give the **CFG** as follows: $S \to UV$
$U \to AB$
$V \to A1A \mid A1B \mid A1U \mid B1U \mid U1U$
$A \to 00^* \mid \in$
$B \to 11^* \mid \in$

This is because:
1. The string s is a concatenation of U and V.
2. The string U can consist of any number of $0's$ and $1's$.
3. The string V can consist of at least one 1 or only 1.
4. The string A can consist of at least one 0 or more than one $0's$.
5. The string B can consist of at least one 1 or more than one $1's$.