CH-5 & CH-6

11/23/2016

Foundations of Comp. Science

HW-2

Kiran C. Shettar
UML ID - 01605800

5.1.1 c) The set of all strings of a's & b's that are not of the form ww, that is, not equal to any string repeated.

$$S \rightarrow a \mid b \mid \epsilon$$

5.1.1 d) The set of all strings with twice as many 0's as 1's.

$$S \rightarrow ASB$$
$$A \rightarrow 00A \mid \epsilon$$
$$B \rightarrow 1B \mid \epsilon$$

5.1.2 b) 1101

Left most

$S \rightarrow A1B$
$S \xrightarrow[lm]{*} 1B$
$S \xrightarrow[lm]{*} 10B$
$S \xrightarrow[lm]{*} 100B$
$S \xrightarrow[lm]{*} 1001B$
$S \xrightarrow[lm]{*} 1001$

Right most

$S \xrightarrow[rm]{*} A1B$
$S \xrightarrow[rm]{*} A10B$
$S \xrightarrow[rm]{*} A100B$
$S \xrightarrow[rm]{*} A1001B$
$S \xrightarrow[rm]{*} A1001$
$S \xrightarrow[rm]{*} 1001$

5.1.2 c) 00011

Left most derivation

$S \xrightarrow[lm]{*} A1B$

$S \xrightarrow[lm]{*} 0A1B$

$S \xrightarrow[lm]{*} 00A1B$

$S \xrightarrow[lm]{*} 000A1B$

$S \xrightarrow[lm]{*} 000 1B$

$S \xrightarrow[lm]{*} 000 11B$

$S \xrightarrow[lm]{*} 00011$

Right most derivation

$S \xrightarrow[rm]{*}$

$S \xrightarrow[rm]{*}$

$S \xrightarrow[rm]{*}$

$S \xrightarrow[rm]{*}$

$S \xrightarrow[rm]{*}$

$S \xrightarrow[rm]{*}$

$S \xrightarrow[rm]{*}$

5.1.3 Show that every regular language is a context free-language.

Base case: We start this with single charact with no operations. To recognize a single character 'a', we can just have a ~~single~~ CFG with a single transition.

for ex: $S \rightarrow a$

Inductive step: Now, suppose that for any regular expression with fewer than 'k' operators, we can construct a CFG that produces the same language.

Then take a language 'L' be represented by the regular expression R, such that 'R' has 'k' operators.

Then we have three possible operations.

Union: If $R = R_1 + R_2$, then assume that R_1 is produced by the CFG with a start symbol S_1, and R_2 is produced by a CFG with a start symbol S_2. We created a new start state S with the production rule $S \rightarrow S_1 | S_2$. If the first production we use is $S \rightarrow S_1$, then we will produce exactly the string matched by R_1, by our inductive hypothesis. If the first production we use is $S \rightarrow S_2$, then we will produce exactly the strings matched by R_2.

Concatenation: If $R = R_1 R_2$, then we create a new CFG with start state S and the production rule $S \rightarrow S_1 S_2$. By our inductive hypothesis, S_1 produces exactly the strings matched by R_1, and S_2 produces exactly the strings matched by. So 'S' will produce exactly

the strings in $R_1 R_2$

kleen star: If $R = R_1^*$, then we create a new CFG with start state ~~$S \Rightarrow S_1 S_1$~~ $S \to S_1 S_1 | \epsilon$. Then we can prove by induction that a string with any number of copies of string matched by R_1 can be generated.

∴ Since every regular expression has an equivalent CFG, every regular language is context free.

5.1.4 a) Every right linear grammar generates a regular language.

Right linear grammar language is a 4-tuple $< T, N, S, R >$

where,

(i) T: Set of finite terminal including empty string.

(ii) N: Finite set of terminals

(iii) S: Start symbol.

(iv) R: Finite set of re writing rules of the form $A \to xB$ or $A \to x$, where A & B stand for non-terminals & x stands for terminal

We need to show that languages that're generated by right linear should always be grammars.

This is done by constructing an NFA that is similar to the derivations of a right linear grammar. We also have to keep in mind that right linear sentential forms are of a special form in which there is exact one variable & it occurs as the right most symbol. Suppose now that we have a step in derivation.

ab.. cD → ab.. cdE arrived at by productio D → dE. The corresponding NFA can initiate this step by going from state D to state E, when a symbol 'd' is encountered. In this scheme, the state of the automation correspon to the variable in the sentential form. Hence this is proved.

5.1.4 b> Every regular language has a right linear grammar.

<i> S: start state

<ii> Associate each rule of the form A→wB into a transition in the FSA from state

A to state B on the input 'w'.

(iii) Associate each rule of the form $A \to w$ into a transition in the FSA from state A on the input 'w' to a final state 'F'.

A grammar $G = (V, T, P, S)$ is said to be right linear if the productions are of the form

$A \to xB$

$A \to xyB$

$A \to xy$

A regular grammar is one that is either right-linear or left-linear. In the regular grammar atmost 1 variable appear on the right side of any production. Furthermore that variable must be consistently be either the right most or left most symbol of the right side of any production

Ex: $G_1 = (\{S\}, \{a, b\}, S, P_1)$ with P_1 given as

$S \to abS \mid a$ is right linear.

$G_2 = (\{S, S_1, S_2\}, \{a, b\}, S, P_2)$ with productions

$S \to S_1 ab$

$S_1 \to S_1 ab \mid S_2$

$S_2 \to a$ is left linear.

Both G_1 & G_2 are regular grammars. (7)

The sequence $S \to ab$ $S \to abab$ $S \to ababa$ is derivation with G_1. from this single instance it is to say that $L(G_1)$ the language denoted by the regular expression $r = (ab)^* a$. Similarly $L(G_2)$ is the regular language.

5.1.8 Given CFG G

$S \to aSbS \mid bSaS \mid \epsilon$

Now, we have to prove $L(G)$ is the set of all strings with equal number of a's & b's.

Then assume

$S \to aB \mid bA$

$A \to a \mid aS \mid bAA$

$B \to b \mid bS \mid aBB$

Here, a, b are terminals. S, A & B are non-terminals. This given language CFG generate the set of all strings with an equal number of a's & b's. The language begins as follows.

$L(G) = \{ aabb, abab, baab, baba, bbaa \ldots \}$

If a word in L(G), starts with an 'a' the remainder of the word is a string with the property i.e has in total exactly one more b than a's.

If a word in L(G) starts with a,b it must be of the form bA where from 'A' generates any string in which the number of a's is one more than number of b's.

Now, all the words derivable ~~from~~ from 'S' are the words in L(G) & all the words in L(G) are generated by S.

∴ L(G) is the set of all strings with an equal number of a's & b's.

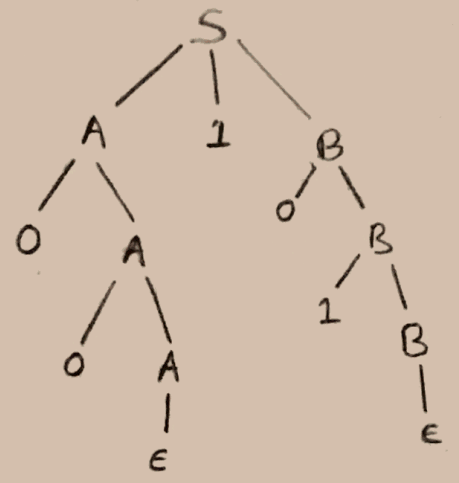
5.2.1 Grammar:

$S \rightarrow A\,1\,B$
$A \rightarrow 0A \mid \epsilon$
$B \rightarrow 0B \mid 1B \mid \epsilon$

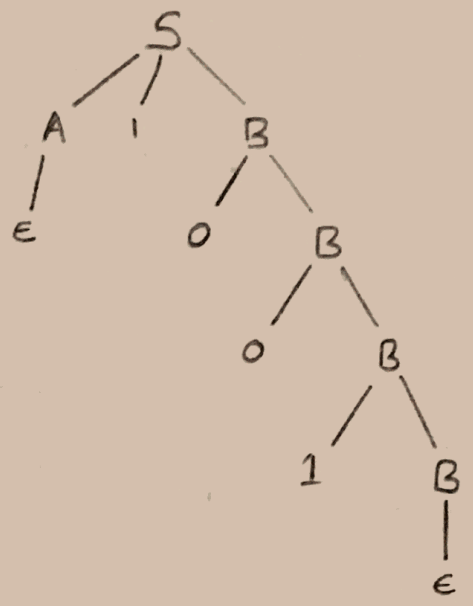
Parse tree for:

1. 00101
2. 1001
3. 00011

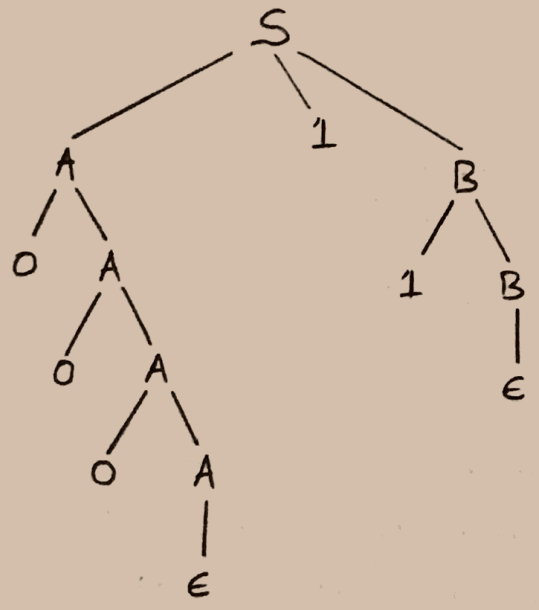
(1)



(2)



(3)



5.3.1 $B \rightarrow BB \mid (B) \mid \epsilon$.

A string of paranthesis is termed as balanced if and only if it's scanned balanced. i.e if the number of open paranthesis should be equal to the number of closed paranthesis.

There's actually only one expansion that inserts paranthesis in this grammar and it inserts an open paranthesis before inserting a closed paranthesis. This means that the count of open paranthesis should always be equal to or more than closed paranthesis. Therefore the strings produced will be scan balanced and thus balanced.

5.3.4 b> & c>

```
<html>
    <ol>
        <li> row </li>
        <li> header </li>
        <li> data </li>
    </ol>
    <ul>
        <li> table row </li>
        <li> table header </li>
        <li> table data </li>
    </ul>
    <table>
        <tr>
            <th> first_column </th>
            <th> second-column </th>
            <th> third_colum </th>
        </tr>
    </table>
```

```
    <tr>
        <td> a </td>
        <td> b </td>
        <td> c </td>
    </tr>
    <tr>
        <td> A </td>
        <td> B </td>
        <td> C </td>
    </tr>
</table>
</html>
```

5.3.5

Derivation One	Derivation Two
S	S
aS	aSbS
aasbS	aaSbS
aaεbS	aaεbS
aabε	aabε
aab	aab

Table: Different left most derivations for aab.

```
course = course list;
course list = course, course list;
course: name, professor, optimal student list,
        optimal assistant
name = 'character data'
professor = 'character data'
```

```
optimal student list = student list 1 ;
student list = student, student list;
optimal assignment = assistant 1 ;
assistant = 'character data' ;
```

5.4.1 a> Parse trees.

Showing ambiguity : You can derive the string aab as follows:
(at every step we expand the leftmost non-terminal).

$S \rightarrow aSbS \rightarrow aaSbS \rightarrow aabS \rightarrow aab$
$S \rightarrow aS \rightarrow aaSbS \rightarrow aabS \rightarrow aab.$

These two parses correspond to ~~the~~ associating the 'b' with the first or the second 'a'. This is some what analogous to the problem of the dangling else, where the last else may be associated with the previous then or with an earlier one.

5.4.2 The proof that all prefixes have atleast as many a's as b's follows from the given productions. It is clearly true if the length of the string is 1 (it can only be a single 'a'). or 2 (aa or bb). Assume that it is true for all strings of length up to 'n'. Then, using the given productions, we can construct strings of length n+1. & longer, for which the property clearly holds because we have added an 'a' to any previous prefix. Therefore the property holds for all 'n'.

5.4.5 a) Consider the grammar.

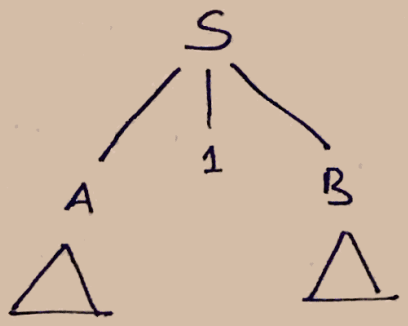
$S \rightarrow A1B$

$A \rightarrow 0A \mid \epsilon$

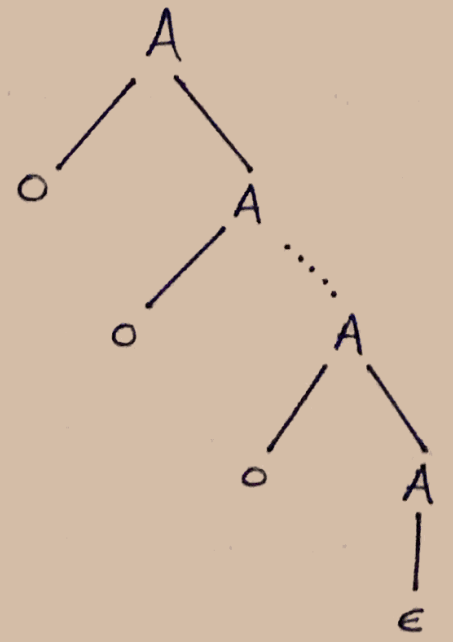
$B \rightarrow 0B \mid 1B \mid \epsilon$

We should prove that it is not possible to have two different parse trees for a string produced by grammar.

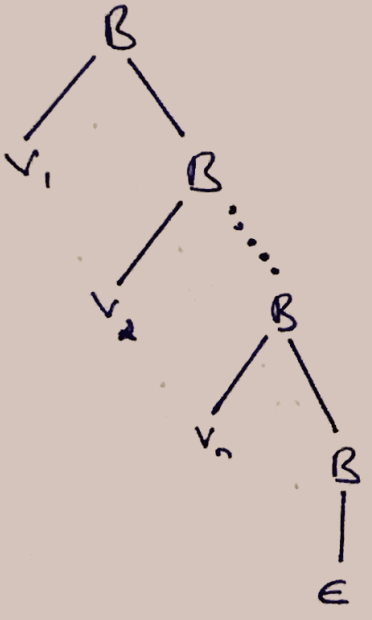
Proof: Since S is the start symbol and $S \to A1B$ the only rule containing S any parse tree for a string w produced by the grammar must have the form



'A' can produce only strings of 0's & clearly the parse tree below 'A' will always have the structure



Thus if there are two parse trees for a string 'w' they must differ below B. So far we can see that the string 'w' must be of the form $w = 00\ldots01v$, let us assume that v is of length n & $v = v_1 v_2 \ldots v_n$ where $v_i \in \{0,1\}$ for $i \in \{1,2\ldots n\}$. However since $B \to 0B \mid 1B \mid \epsilon$ with both the 0 and the 1 on the left we will always have a parse tree of the form



It is clear that, two different trees of this structure cannot produce the same string hence we are unable to produce two different parse trees for a string 'w'.

5.4.5 b) find a gramanar for the same language that is ambiguous & demonstrate

Given grammar:

$S \to A\,1B$

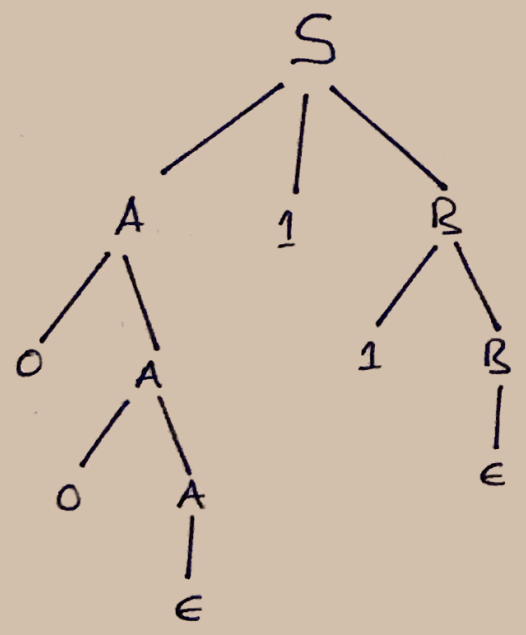
$A \to 0A \mid \epsilon$

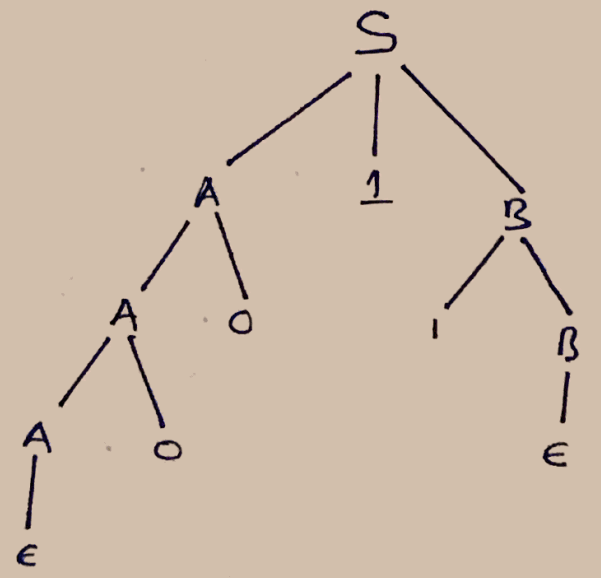
$B \to 0B \mid 1B \mid \epsilon$

The grammar for the same language that is ambiguous:

$S \rightarrow A\ 1\ B$
$A \rightarrow 0A \mid A0 \mid \epsilon$
$B \rightarrow 0B \mid 1B \mid \epsilon$

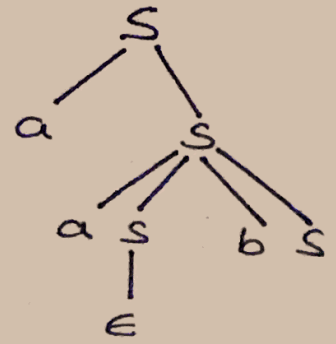
* for a string $w = 0011$, two parse trees are produced. Therefore this grammar is ambiguous.



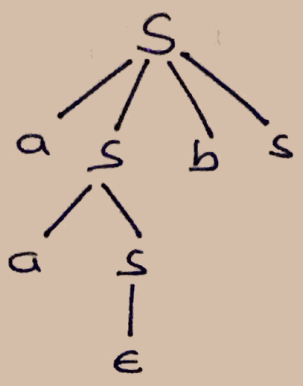


5.4.7 a) Using the string $+*-xyxy$ and grammar

$\epsilon \rightarrow +EE \mid *EE \mid -EE \mid x \mid y$



This is a possible parse tree for aab.



Another possible parse tree for aab

Derivation 1

S
aS
aaSbS
aaSbϵ
aaϵb
aab

Derivation 2-

S
aSbS
aaSbS
aaSbϵ
aaϵb
aab

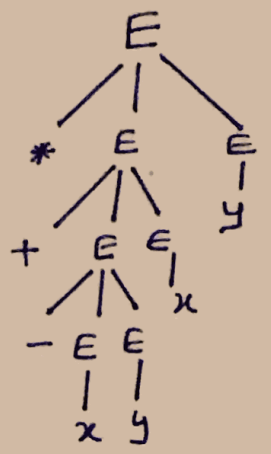
Table: Different rightmost derivation for aab.

ϵ
+ϵϵ
+ϵy
+*ϵϵy
+*ϵxy
+*-ϵϵxy
+*-ϵyxy
+*-xyxy

Table: Left most derivation for aab.

ϵ
+ϵϵ
+*ϵϵϵ
+*-ϵϵϵϵ
+*-xϵϵϵ
+*-xyϵϵ
+*-xyxϵ
+*-xyxy

Derivation tree for + * - x y x y



5.4.7 b) In this grammar, the application of each rule generates a string starting with 'a' unique terminal symbols (+, *, -, x, y)

For any string w that belongs to context free language, when we consider a left most variable E in the left most derivation of the string, there is only one rule that can be used to continue the derivation. The rule is uniquely determined by the next symbol in 'w' to be derived. So there's only one left most derivation for 'w' & hence the unambiguity of the grammar.

6.1.1 b) 0011

$(q, 0011, Z_0) \vdash (q, 011, XZ_0)$

$\vdash (q, 11, XXZ_0), (P, 011, Z_0)$

$\vdash (q, 1, XXZ_0), (P, 11, XZ_0)$

$\vdash (q, \epsilon, XXZ_0), (P, 1, XZ_0), (P, 11, Z_0),$

$(P, 1, XXZ_0)$

$\vdash (P, \epsilon, XZ_0), (P, \epsilon, XXZ_0), (P, 1, Z_0),$

$(P, 1, \epsilon), (P, 1, XZ_0), (P, \epsilon, XXXZ_0)$

$\vdash (P, \epsilon, Z_0), (P, \epsilon, \epsilon)$

6.1.1 c) 010

$(q, 010, Z_0) \vdash (q, 10, XZ_0)$

$\vdash (q, 0, XZ_0), (P, 10, Z_0)$

$\vdash (q, \epsilon, XXZ_0), (P, 0, Z_0), (P, 0, \epsilon)$

$\vdash (P, \epsilon, XZ_0)$

$\vdash (P, \epsilon, Z_0)$

6.2.1 b) $(\{q, P, r\}, \{0, 1\}, \{X, Z_0\}, \delta, q, Z_0)$

$\delta(q, 0, Z_0) = \{(q, X)\}$

$\delta(q, 0, X) = \{(q, XX)\}$

$\delta(q, 0, XX) = \{(q, XXX)\}$

$\delta(q, 1, XXX) = \{(P, XX)\}$

~~$\delta(r, x)$~~ $\delta(P, 1, X) = \{(r, X)\}$

$\delta(r, 1, X) = \{(r, \epsilon)\}$

6.2.1 c) $(\{q, p\}, \{0, 1\}, \{x, Z_0\}, \delta, q, Z_0)$

$\delta(q, 0, Z_0) = \{(q, x)\}$

$\delta(q, 0, x) = \{(q, xx)\}$

$\delta(q, 1, x) = \{(p, \epsilon)\}$

$\delta(p, 1, x) = \{(p, \epsilon)\}$

6.2.2 b) Begin with start state 'q_0', with start symbol 'Z_0', and imidiately guess whether to check for

$\delta(q_0, 0, Z_0) = (q_1, 00Z_0)$

$\delta(q_0, 1, Z_0) = (q_1, 1Z_0)$

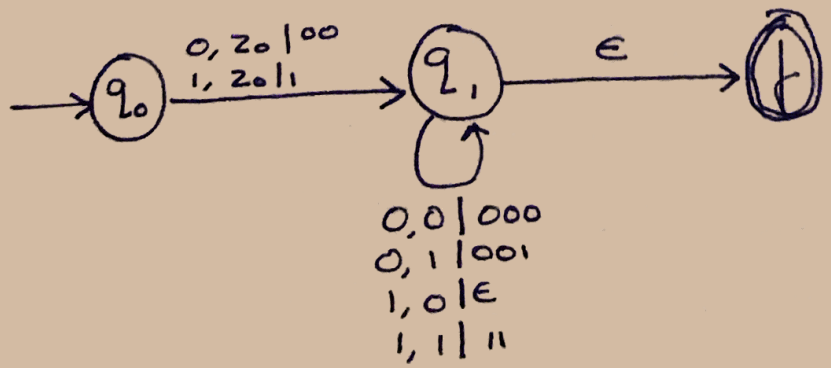
$\delta(q_1, 0, 0) = (q_1, 00)$

$\delta(q_1, 1, 0) = (q_1, \epsilon)$

$\delta(q_1, 0, 1) = (q_1, 01)$

$\delta(q_1, 1, 1) = (q_1, 11)$

$\delta(q_1, \epsilon, Z_0) = (f, \epsilon)$



6.2.3 a) $\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$

1. $i \neq j \neq 0$ (state q_1)
2. $i \neq j > 0$ (state q_2)
3. $j = k$ (state q_3)

$\delta(q_0, \epsilon, z_0) = \{(q_1, z_0), (q_2, z_0), (q_3, z_0)\}$

$\delta(q_1, c, z_0) = \{(q_1, z_0)\}$

$\delta(q_2, a, z_0) = \{(q_2, xz_0)\}$

$\delta(q_2, a, x) = \{(q_2, xx)\}$

$\delta(q_2, b, x) = \delta(q_4, b, x) = \{(q_4, \epsilon)\}$

$\delta(q_4, \epsilon, z_0) = \{(q_1, z_0)\}$

$\delta(q_3, a, z_0) = \{(q_3, z_0)\}$

$\delta(q_3, b, z_0) = \{(q_5, xz_0)\}$

$\delta(q_5, b, x) = \{(q_5, xx)\}$

$\delta(q_5, c, x) = \delta(q_6, c, x) = \{(q_6, \epsilon)\}$

$\delta(q_6, \epsilon, z_0) = \{(q_3, \epsilon)\}$

6.2.3 b) $\delta(q_0, \epsilon, z_0) = \{(q_1, z_0), (q_2, z_0)\}$

$\delta(q_0, a, z_0) = \delta(q_1, xz_0)$

$\delta(q_0, b, z_0) = \delta(q_1, xz_0)$

$\delta(q_1, a, x) = \delta(q_2, xx)$

$\delta(q_1, b, x) = \delta(q_2, xx)$

$\delta(q_2, a, xx) = \delta(q_3, \epsilon)$

$\delta(q_2, b, xx) = \delta(q_3, \epsilon)$

6.2.5 b)

$\delta(q_0, abb, Z_0) \vdash (q_1, AAZ_0)$

$\delta(q_1, bb, AAZ_0) \vdash (q_1, AZ_0)$

$\delta(q_1, b, AZ_0) \vdash (q_1, \epsilon)$

$\delta(q_1, \epsilon, Z_0) \vdash (q_0, Z_0)$

$\delta(q_0, \epsilon, Z_0) \vdash (f, \epsilon)$

6.2.5 c) $b^7a^4 \Rightarrow bbbbbbbaaaa$

$\delta(q_0, bbbbbbbaaaa, Z_0) \vdash (q_2, BZ_0)$

$\delta(q_2, bbbbbbaaaa, BZ_0) \vdash (q_2, BBZ_0)$

$\delta(q_2, bbbbbaaaa, BBZ_0) \vdash (q_2, BBBZ_0)$

$\delta(q_2, bbbbaaaa, BBBZ_0) \vdash (q_2, BBBBZ_0)$

$\delta(q_2, bbbaaaa, BBBBZ_0) \vdash (q_2, BBBBBZ_0)$

$\delta(q_2, bbaaaa, BBBBBZ_0) \vdash (q_2, BBBBBBZ_0)$

$\delta(q_2, baaaa, BBBBBBZ_0) \vdash (q_2, BBBBBBBZ_0)$

$\delta(q_2, aaaa, BBBBBBBZ_0) \vdash (q_3, BBBBBBZ_0)$

Contents of the stack would be 'a^3'.

6.2.5 d)

$L(P) = \{ w \mid w \in (a,b)$ such that number of b's is 1 more than number of a's $\}$

6.2.6 a)

$(\{q, p\}, \{0, 1, \epsilon\}, \{z_0, x\}, \delta, q, z_0, \{p\})$

$\delta(q_0, 0, z_0) = \{(q, x z_0)\}$
$\delta(q, \epsilon, z_0) = \{(p, \epsilon)\}$
$\delta(q, 0, x) = \{(q, xx)\}$
$\delta(q, 1, x) = \{(q, x)\}$
$\delta(q, \epsilon, x) = \{(p, \epsilon)\}$
$\delta(q, \epsilon, x) = \{(p, \epsilon)\}$
$\delta(q, 1, x) = \{(p, xx)\}$
$\delta(q, 1, z_0) = \{(p, \epsilon)\}$

b)

6.2.6 b) $(\{q, p\}, \{0, 1, \epsilon\}, \{z_0, x\}, \delta, q, z_0, \{p\})$

$\delta(q, 0, z_0) = \{(q, x z_0)\}$
$\delta(p, \epsilon, z_0) = \{(p, \epsilon)\}$
$\delta(q, 0, x) = \{(q, xx)\}$
$\delta(q, 1, x) = \{(q, x)\}$
$\delta(q, \epsilon, x) = \{(p, \epsilon)\}$
$\delta(p, \epsilon, x) = \{(p, \epsilon)\}$
$\delta(p, 1, x) = \{(p, xx)\}$
$\delta(p, 1, z_0) = \{(p, \epsilon)\}$

6.2.7 The main thing is to encode the stack alphabet in a unary manner (0s, with 1's used as a seperator). So, lets say we have an alphabet of the form A, B and C. A is encoded as 0, B as 00, and C as 000. Now, the transition $B \to AC$ can be re-encoded as $00 \to 01000$ (the 1 acts as a seperator b/n B & C).

6.3.2 The PDA might have just 1 state 'q' and its transition function is given by

$$\delta(q, \epsilon, S) = \{(q, aAA)\}$$
$$\delta(q, \epsilon, A) = \{(q, aS), (q, bS), (q, a)\}$$
$$\delta(q, a, a) = \{(q, \epsilon)\}$$
$$\delta(q, b, b) = \{(q, \epsilon)\}$$

6.3.5 a) $\{a^n b^m c^{2(n+m)} \mid n \geq 0, m \geq 0\}$

Let $M = (\{q_1, q_2, q_3\}, \{a, b, c\}, \{S, z_0\}, \delta, q_1, z_0, \phi)$ be a PDA defined by

$$\delta(q_1, a, A) = \{(q_1, SSA)\} \text{ for all } A \in \Gamma$$
$$\delta(q_1, \epsilon, A) = \{(q_2, A)\} \text{ for all } A \in \Gamma$$

$\delta(q_2, b, A) = \{(q_2, SSA)\}$

$\delta(q_2, \epsilon, A) = \{(q_3, A)\}$ for all $A \in T$

$\delta(q_3, c, S) = \{(q_2, \epsilon)\}$

$\delta(q_3, \epsilon, Z_0) = \{(q_3, \epsilon)\}$

6.3.5 b) $\{a^i b^j c^k \mid i = 2j \text{ or } j = 2k\}$

following is a CFG generating the language:

$S \rightarrow PC \mid AQ$

$P \rightarrow aaPb \mid \epsilon$

$C \rightarrow cC \mid \epsilon$

$A \rightarrow aA \mid \epsilon$

$Q \rightarrow bbQc \mid \epsilon$

Let $M = (\{q\}, \{a,b,c\}, \{a,b,c,A,B,P,Q,S\}, \delta, q, S, \phi)$

where

$\delta(q, \epsilon, S) = \{(q, PC), (q, AQ)\}$

$\delta(q, \epsilon, P) = \{(q, aaPb), (q, \epsilon)\}$

$\delta(q, \epsilon, C) = \{(q, cC), (q, \epsilon)\}$

$\delta(q, \epsilon, A) = \{(q, aA), (q, \epsilon)\}$

$\delta(q, \epsilon, Q) = \{(q, bbQc, (q, \epsilon)\}$

$\delta(q, a, a) = \{q, \epsilon\}$

$\delta(q, b, b) = \{q, \epsilon\}$

$\delta(q, c, c) = \{q, \epsilon\}$

6.3.5 c) $\{0^n 1^m \mid n \leq m \leq 2n\}$

Following grammar generates the language:

$$S \to 0S1 \mid 0S11 \mid \epsilon$$

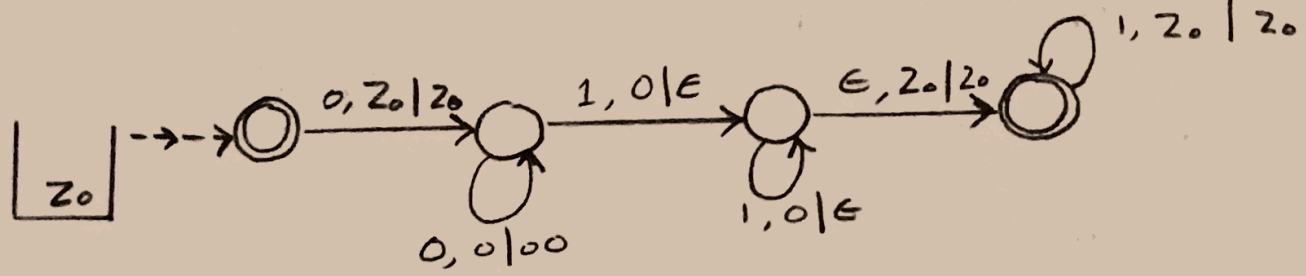
Let $M = (\{q\}, \{0,1\}, \{0,1,S\}, \delta, q, S, \emptyset)$, where

$\delta(q, \epsilon, S) = \{(q, 0S1), (q, 0S11), (q, \epsilon)\}$.

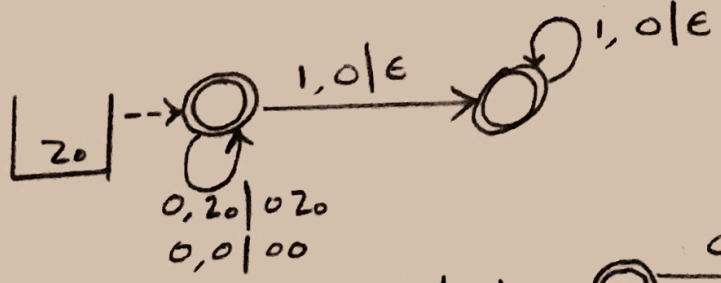
$\delta(q, 0, 0) = \{(q, \epsilon)\}$

$\delta(q, 1, 1) = \{(q, \epsilon)\}$

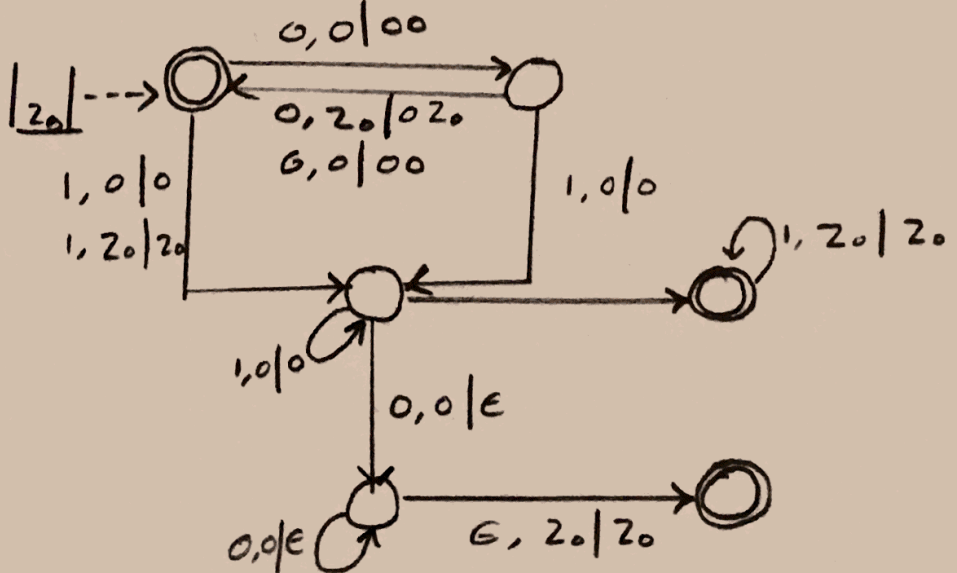
6.4.2 a) $\{0^n 1^m \mid n \leq m\}$



b)



c)



6.3.4

$$S \to [qZ_0q] \mid [qZ_0p]$$

$$qZq \to 0(qXq)(qZq) \mid 0(qXp)(pZq)$$
$$qXq \to 0[qXq][qXq] \mid 0[qXp][pXq]$$
$$qXp \to \epsilon \mid 0[qXq][qXp] \mid 0[qXp][pXp]$$
$$pXp \to \epsilon \mid 1[pXp][pXp] \mid 1[pXq][qXp]$$
$$pZp \to \epsilon$$
$$qZp \to 0[qXq][qZp] \mid 0[qXp][pZp]$$
$$pXq \to 1[pXp][pXq] \mid 1[pXq][qXq]$$

6.3.7

Let 'S' be the start symbol. All symbols of form $[pXq]$ where p & q are states in Q & X is a stack symbol in T.

We know that $[P_0 Z_0 p]$ is intended to generate all those strings 'w' that cause 'P' to pop Z_0 from its stack while going from state q_0 to state P.

Let $\delta(q, a, x)$ contain pair $(r, Y_1, Y_2 \ldots Y_k)$ where 'a' is either a symbol in Σ or $a = \epsilon$. 'k' can be any number '0' in which pair (r, ϵ)

Then ∀ list states $r_1, r_2, r_3 \ldots r_k$ has the product $[q X r_k] \rightarrow a[r_1, r_1][r_1, r_2, r_2] \ldots [r_{k-1}, r_k, r_k]$

Basis: (P, ϵ) must be in $\delta(q, \omega, x)$ & 'ω' is either a single symbol or ϵ.

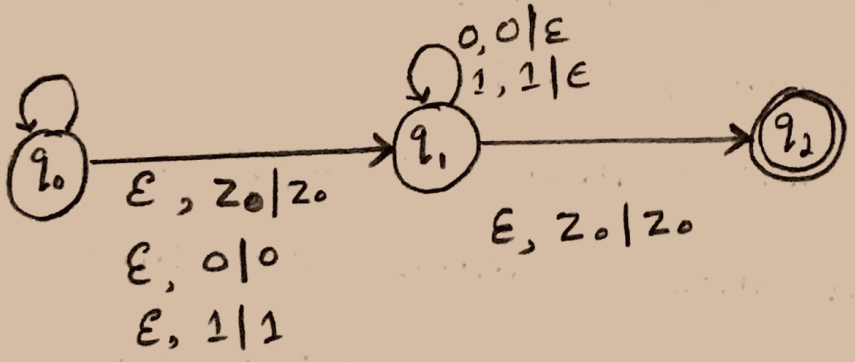
Induction: Suppose the sequence takes 'n' steps. $(q, \omega, x) \vdash (r_0, x, y_1, y_2 \ldots y_k) \vdash^* (P, t, t)$

Hence the tight upper bound will be 'k' variables.

6.4.1 a) This is not a DPDA

0, z_0 | 0z_0, 1, z_0 | 1z_0, 0, 0 | 00, 0, 1 | 01, 10 | 10, 1

1 | 11



We can see that, there're many transitions from the same state of an input hence it is non-deterministic.

$\delta(q_0, 0, z_0) = \{(q_0, 0, z_0)\}$ & $\delta(q_0, \epsilon, z_0) = \{(q_1, z_0)\}$

6.4.1 c) It is not a DPDA

The rules that violate it are:

$\delta(q, 0, x) = \{(p, x)\}$

$\delta(q, \epsilon, x) = \{(q, \epsilon)\}$

$\delta(q, \lambda, \lambda) = \lambda$

6.4.3 b) Let a DPDA accept both w & wx by an empty stack where x is not ϵ.

Then $(q_0, wxz_0) \vdash^* (q)(\epsilon)$ for some state q where q_0 & z_0 are the start state & symbol of P. It is not possible that $(qx, \epsilon) \vdash^* (p)(\epsilon)$ for some state 'p' because we know 'x' is not ϵ. PDA cannot have an empty stack. This observation contradicts assumption that wx is in $w(P)$.

To find P' we modify 'P' in the following ways.

Add a new state 1 and a new start symbol P' with this state & symbol, push the start symbol of 'P' on top of the

stack & then go to start state of P'. The purpose of the new start symbol is to make sure that P doesn't accidentally accept by an empty stack.

Add a new popping state to P. In this state, P pops every symbol it sees on the stack using ε input. If P enters an accepting state, P' enters popping state. Hence for some DPDA P there exists a DPDA P' such that $L = L(P')$.

6.4.4 If we change the state set from 'Q' into $Q \times \{0,1\}$ or change the constructions so that upon reading a symbol, we switch between 0 & 1. Additionally when reading an even numbered symbol in original machine, new one will read empty word.

Thus $(P, 0, x, a, \alpha) \Rightarrow (P^0, 0, X, q', \alpha) \mid (P_1, \lambda, x, q_0, \alpha)$.

assuming $0 \in \Sigma$. (belongs to).

We also get:

$(P, \lambda, x, q, \alpha) \Rightarrow (P^0, \lambda, x, q_0, \alpha)\ \&\ (P_1, \lambda, x, q_1, \alpha)$

Assume that we run a variant of DPDA for the given language. when the new PDA sees the first final state, we ignore it. when this happened, we create some other input other than 1. This is because the deterministic computation of $0^n 1^n$ must be a prefix of $a^n b^{2n}$.

Since we get a contradiction, there's no DPDA for the original language.

5.1.6 a)

Basis: Let $n=1$ $\therefore w = y_1$

From definition wkt

$\alpha = [redacted]\,\gamma_1$ $\therefore a^* \Rightarrow \beta$

$\beta = y_1$

Induction:

Let R+1 be the last string alphabet.

$\alpha = [redacted]\,\gamma_1$

$\beta = [redacted]\,\gamma_{n+1}$

As $\beta = y$, α can become β by one or more steps if α is [redacted]$\cdot\gamma$

Since any string derives itself

$\alpha \overset{*}{\Rightarrow} \beta$ if and only if there is a sequence of 1 or more strings.

5.1.6 b)

Basis: Let the number of steps for $\alpha \overset{*}{\Rightarrow} \beta$ be 1

$\beta \overset{*}{\Rightarrow} \gamma$ be 1

$\therefore \beta = \gamma$

$\alpha = \gamma$ in two steps

$\alpha = \beta = \gamma$

Induction:

Let 'p' be the number of steps required for $\beta \overset{*}{\Rightarrow} \gamma$

Let 'q' be the number of steps required for $\alpha \overset{*}{\Rightarrow} \beta$

Since α takes 'q' steps to get to β and β takes 'p' steps to reach γ.

$\therefore \alpha$ will take $p+q$ steps to reach γ.

$\therefore \alpha \overset{*}{\Rightarrow} \gamma$ if $\beta \overset{*}{\Rightarrow} \gamma$

5.3.3 Seeing the production

$S \rightarrow \varepsilon | SS | iS | iSeS$

looking for first e to the left of i we can see that none exist.

Hence the string with e at the beginning fails.

Seeing the production

$S \rightarrow \varepsilon | SS | iS | iSeS$

Considering a string with no e.

The string can be an empty one.

The production can be $S \rightarrow \varepsilon$ if the string has only i.

5.2.2

Basis: Let the length of 'w' be 1.

Hence 'w' is a terminal that is end node of the tree

The number of steps needed for this is 1. Hence, if length of 'w' is 1 then the number of steps needed is also 1. So, parse tree has two nodes. The start node & the terminal node.

Induction: Let the length of w be 'n'. The derivation of 'w' has 'm' steps.

Let 'A' be the start node of parse tree. Assume that deviation has m+1 steps & for m or fewer steps the statement holds

$$A \Rightarrow X_1 X_2 \dots X_k \overset{*}{\Rightarrow} w$$

Break 'w' as $w_1, w_2 \dots w_k$

If x_i is a terminal then $x_i = w_i$. There's only one production.

If x_i is a variable then w_i was previously inferred to be in language of x_i. Hence it takes almost m steps out of m+1 steps.

∴ for each interference of 'w' where 'w' ha length 1 the number of steps required is 'm Hence parse tree has m+1 nodes. If length of steps is 'n' then the number of nodes are m+n.

5.2.3 Let length of 'w' be 1.

Since the production has ε as the right side. If 'w' is a terminal then the number of nodes are 2.

Sine ε is also the part of the production the number of total nodes is 3. The maximum number of nodes cannot exceed 3 if ε is taken into consideration.

Induction: Let the number of steps be 'm'. The length of 'w' be r. 'A' is the start node of the tree.

Assume derivation has m+1 ϕ for 'm' or fewer steps the statement holds

$\therefore A \Rightarrow x_1 x_2 \dots x_k \overset{*}{\Rightarrow} w$

break w as $w_1, w_2 \dots w_k$

If x_i is the terminal that are 2 terminals including ε. Hence the total number of nodes in the length of the string plus the number of steps.

If x_i is a variable it makes at most 'm' steps out of m+1 steps to reach the end.

Hence for a parse tree of string 'w' other than ε is the maximum no. of nodes is n+2m-1

5.2.4 Basis: Let there be a production $A \to w$. Hence in this variation there's only step.

while $A \Rightarrow x_1 x_2 \cdots x_k \overset{*}{\Rightarrow} w$ the derivation is of '0' steps.

$\therefore$ if $x_1 x_2 x_3 \ldots x_k \overset{*}{\Rightarrow} \alpha$

Since the production leads only to terminals

Induction: Let 'w' be a language of 'A' is inferred after $n+1$ inference steps.

Let inference use some production.

$A \to x_1 x_2 x_3 \ldots x_k$ where x_i is either a terminal or a variable

Assuming $x_1 x_2 x_3 \cdots x_k \overset{*}{\Rightarrow} \alpha$

if $i < j$

the production will be used to derive x_i such that $x_i = \alpha$

Then it proceeds to x_j to derive x_j such that $x_j = \alpha$

As $i < j$ x_i is derived before x_j.

Hence the position of α from x_i is to the left of all position that come from x_j is $i < j$.

5.4.4 Proving $\omega \in \alpha$ then G generates every ω.

Base case: $n = 1$

Let $\omega = a^n b^{n-k}$ where $0 \leq k \leq n$

then $k = 0$ or 1

If $k = 0$

$\omega = ab$

generated by

$S \rightarrow aSbS \Rightarrow ab$

If $k = 1$

$\omega = a$

generated by

$S \rightarrow aS \Rightarrow a$

Inductive hypothesis:

Assume $n = i$ that is for any k

$0 \leq k \leq i$, the string is $\omega = a^i b^{i-k}$

Let $\omega = a^{i+1} b^{(i+1)-k}$ with $0 \leq k \leq i+1$

Case 1: $0 \leq k \leq i$ Then $\omega = a^{i+1} b^{(i+1)-k}$

Inductive step

Prove that every derivation with $n+1$ step

generate string in α.

Since $n+1 > 1$ the first step cannot be

$S \Rightarrow \epsilon$

Hence derivation starts with either

$S \rightarrow aS$ or $S \rightarrow aSbS$

For $S \to aS$

'w' be in the form aw_1 where w_1 is a string derived from S in 'n' steps.

By induction hypothesis $w_1 \in L$

$w_1 = a^i b^j$ with $i \geq 1$ and $j \leq i$

$w = aa^i b^j = a^{i+1} b^j \in \alpha$

Hence, $i \geq j$ & $i+1 \geq j+1$

For $S \to aSbS$

'w' is in the form aw_1, bw_2 where w_1 & w_2 are derived ~~from~~ in 'n' steps from S. Hence, by inductive hypothesis

$w_1 = a^i b^j$ where $i \geq j$

$w_2 = a^p b^q$ where $p \geq q$

$= aa^i b^{i+1-k} = aa^i b^{i-k} b$

By inductive hypothesis the string $a^i b^{i-k}$ for $0 \leq k \leq i$ is generated by grammar.

$S \overset{*}{\Rightarrow} a^i b^{i-k}$ and hence

~~$S \to aS \to aa$~~

$S \to aSbS \to aa^i b^{i-k} b = w$ is a

Deviation //

Case 2: $k = i+1$

$$\det \omega = a^{i+1} b^{(i+1)-i+1}$$
$$= a^{i+1} b^{0}$$
$$= a^{i+1}$$

∴ Since,

$$S \overset{*}{\Rightarrow} \omega \overset{*}{\Rightarrow} a^{i+1}$$

This is generated by,

$S \to aS \to aa^{i+1}$ is a derivation of 'ω'.

∴ String of the form $a^i b^{i-k} a^{i+1} \notin \Sigma$. //