UNIT IV
PROPERTIES OF CONTEXT-FREE LANGUAGES

Normal forms for CFG – Pumping Lemma for CFL - Closure Properties of CFL – Turing Machines – Programming Techniques for TM.

4.1. INTRODUCTION

If a language is a context free language, then it should have a grammar in some specific form. In order to obtain the form, we need to simplify the context free grammars.

4.2. NORMAL FORMS FOR CFG

4.2.1. Simplification of CFG:

In a CFG, it may not be necessary to use all the symbols in V T on all the productions in P for deriving sentences. So we try to eliminate the symbols and productions in G which are not useful for derivation of sentences. To simplify a CFG we need to eliminate,

(a) Eliminating Useless Symbols.
(b) Eliminating ε – Productions.
(c) Eliminating Unit Productions.

(a) Eliminating Useless Symbols:

The productions from a grammar that can never take part in any derivation is called useless symbols.

4.2.3. Definition:

Let G = (V, T, P, S) be a grammar, A grammar 'X' is useful, if there is a derivation S αXβ w, where 'w' is in T. A symbol 'X' is not useful, we say it is useless. There are two ways to eliminating useless symbols,

(1) First, eliminate non generating symbols.
(2) Second, eliminate all symbols that are not reachable in the grammar G.

Ex.1:

Eliminate useless symbols from the given grammar,

$S \rightarrow AB \mid a$

$A \rightarrow a$

Soln:

We find that no terminal string is derivable from B. So, to eliminate the symbol B and the production S→AB. After eliminating useless symbols and the productions are,

$S \rightarrow a$

$A \rightarrow a$

Ex.2:

Eliminate useless symbols from the given grammar,

$S \rightarrow aSbS \mid C \mid \varepsilon$

$C \rightarrow aC$

Soln:

The production $S \rightarrow C$ is useless production, since C cannot derive any terminal string. To eliminate the variable C, and the productions $S \rightarrow C$, $C \rightarrow aC$.

After eliminating all useless symbols the productions are,

$S \rightarrow aSbS \mid \varepsilon$

(b) <u>Eliminating ε – Productions:</u>

Any productions of CFG of the form $A \rightarrow \varepsilon$ is called ε – production.

Any variable A for which the derivations A ε is possible, is called as nullable.

Steps:

(i) Find the set of nullable variables of G, for all productions of the form $A \rightarrow \varepsilon$. Put A to Vnullable, if $B \rightarrow A_1A_2 \ldots\ldots A_n$, where A_1, A_2, $\ldots\ldots A_n$ are in Vnullable then put B also in Vnullable.

(ii) Construct a new set of production P'. For each production in P, put that production and all the production generated by replacing nullable variables for all possible combinations into P'.

Ex.1:

Eliminate ε – productions from the grammar,

$S \rightarrow abB$

$B \rightarrow Bb \mid \varepsilon$

Soln:

(i) $B \rightarrow \varepsilon$ is a null production. Vnullable = {B}

(ii) Construct the production in P'

S → abB | ab
B → Bb | b

Ex.2:

Eliminate ε – productions from the grammar,

S → BabC
B → a | ε
C → b | ε

Soln:

(i) Find all nullable variables, Vnullable = {C, B}.

(ii) Construct the production in P'.

S → BabC | abC | Bab | ab
B → a
C → b

Ex.3:

Eliminate ε – productions from the grammar,

S → AB
A → aAA | ε B → bBB | ε

Soln:

(i) Find all nullable variables, Vnullable = {A, B]

(ii) Construct the production in P'.

S → AB | B | A
A → aAA | aA | a
B → bBB | bB | b

(c) <u>Eliminating Unit Productions:</u>

A production of the form A → B, where A, B V is called unit productions.

Steps:

(1) Add all non – unit productions (or) to eliminate all unit productions of P into P'.

(2) For each unit production A → B, add A → α, to new production set P', where B → α is a non – unit production in P.

Ex.1:

Eliminate all unit productions from the grammar,

S → AaB | C B → A | bb

A → a | bC | B C → a

Soln:

(1) To eliminate all unit productions to P'.

S → AaB B → bb

A → a | bC C → a

(2) S → C, A → B and B → A are unit productions, they are derivable, hence removing unit productions to get,

S → AaB | a B → a | bC | bb

A → a | bC | bb C → a

Ex.2:

Eliminate all unit productions from the grammar,

S → ABA | BA | AA | AB | A | B

A → aA | a

B → bB | b

Soln:

(1) To eliminate all unit productions to P'.

S → ABA | BA | AA | AB

A → aA | a

B → bB | b

(2) S → A, S → B are unit productions, they are derivable, hence removing unit productions to get,

S → ABA | BA | AA | AB | aA | a | bB | b

A → aA | a

B → bB | b

4.2.4. <u>Types of Normal Forms:</u>

There are two normal forms, they are,

(1) Chomsky Normal Form (CNF)

(2) Greibach Normal Form (GNF)

(1) Chomsky Normal Form:

Every CFL is generated by a CFG in which all productions are of the form $A \rightarrow BC$ or $A \rightarrow a$, where A, B, C are variables, and 'a' is a terminal. This form is called Chomsky Normal Form(CNF).

Rules for Converting a Grammar into CNF:

(i) Simplify the grammar by, eliminating ε – productions, unit productions and useless symbols.

(ii) Add all the productions of the form $A \rightarrow BC$ and $A \rightarrow a$ to the new production set P'.

(iii) Consider a production $A \rightarrow A_1A_2 \ldots\ldots A_n$, if A_i is a terminal say a_i then add a new variable C_{ai} to the set of variables, say V' and a new production $C_{ai} \rightarrow a_i$ to the new set of production P'. Replace A_i and A production of P by C_{ai}.

(iv) Consider $A \rightarrow A_1A_2 \ldots\ldots\ldots A_n$, where $n \geq 3$ and all A_i's are variables then introduce new productions. $A \rightarrow X_1C_1$, $C_1 \rightarrow X_2C_2$,.....$C_{n-2} \rightarrow X_{n-1}C_n$ to the new set of productions P' and the new variables C_1, C_2, $\ldots\ldots\ldots$ C_{n-2} into new set of variables V'.

(v)

PROBLEMS:

Ex.1:

Convert the following grammar into CNF,

$S \rightarrow AAC$

$A \rightarrow aAb \mid \varepsilon$

$C \rightarrow aC \mid a$

Soln:

Step 1: Simplify the Grammar:

- **Eliminate ε – Productions:**
  - Find nullable variables Vnullable = {A}
  - Construct the production in P'.

P' : $S \rightarrow AAC \mid AC \mid C$

$A \rightarrow aAb \mid ab$

$C \rightarrow aC \mid a$

- **Eliminate Unit Productions:**

  $S \rightarrow C$ is a Unit Production. Replace C by its productions,

  $S \rightarrow AAC \mid AC \mid aC \mid a$

  $A \rightarrow aAb \mid ab$

  $C \rightarrow aC \mid a$

- **Eliminate Useless Symbols:**

  There is no useless symbols. All the variables are generating terminal string.

Step 2: Reduce the given grammar into CNF:

Add all the productions of the form $A \rightarrow BC$ or $A \rightarrow a$.

- $S \rightarrow AAC$

  $S \rightarrow AD_1$; $D_1 \rightarrow AC$

- $S \rightarrow aC$

  $S \rightarrow C_aC$; $C_a \rightarrow a$

- $A \rightarrow ab$

  $A \rightarrow C_aC_b$

- $A \rightarrow aAb$

  $A \rightarrow C_aAC_b$; $C_b \rightarrow b$

  $A \rightarrow C_aD_2$; $D_2 \rightarrow AC_b$

- $C \rightarrow aC$

  $C \rightarrow C_aC$

The Resultant Grammar is,

$S \rightarrow AD_1 \mid AC \mid C_aC \mid a$
$D_1 \rightarrow AC$
$C_a \rightarrow a$
$A \rightarrow C_aD_2 \mid C_aC_b$
$D_2 \rightarrow AC_b$
$C_b \rightarrow b$
$C \rightarrow C_aC \mid a$

Ex.2:

Convert the following grammar into CNF,

$S \rightarrow aAa \mid bBb \mid \varepsilon$

$A \rightarrow C \mid a$

$B \rightarrow C \mid b$

$C \rightarrow CDE \mid \varepsilon$

$D \rightarrow A \mid B \mid ab$

Soln:

Step 1: Simplify the Grammar:

- **Eliminate ε – Productions:**
  - Find nullable variables Vnullable = {S, C}
  - Construct the production in P'.

  P' : S → aAa | bBb C → CDE | DE
  A → C | a D → A | B | ab
  B → C | b

- **Eliminate Unit Productions:**

  A → C, B → C, D → A, D → B is Unit Productions. After eliminating unit productions grammar is,

  S → aAa | bBb C → CDE | DE
  A → CDE | DE | a D → CDE | DE | a | b | ab
  B → CDE | DE | b

- **Eliminate Useless Symbols:**

  Since C and E does not produce terminal string, so remove the symbols and productions from the grammar. After eliminating useless symbols grammar is,

  S → aAa | bBb B → b
  A → a D → a | b | ab

Step 2: Reduce the given grammar into CNF:

Add all the productions of the form A → BC or A → a.

- S → aAa
  $S \rightarrow C_aAC_a$; $C_a \rightarrow a$
  $S \rightarrow C_aD_1$; $D_1 \rightarrow AC_a$

- S → bBb
  $S \rightarrow C_bBC_b$; $C_b \rightarrow b$
  $S \rightarrow C_bD_2$; $D_2 \rightarrow BC_b$

- D → ab
  $D \rightarrow C_aC_b$

The Resultant Grammar is,

$S \rightarrow C_aD_1 \mid C_bD_2$
$C_a \rightarrow a \qquad C_b \rightarrow b$
$D_1 \rightarrow AC_a \qquad D_2 \rightarrow BC_b$
$A \rightarrow a$
$B \rightarrow b$
$D \rightarrow a \mid b \mid C_aC_b$

Ex.3:

Convert the following grammar into CNF,

$S \rightarrow abAB$

$A \rightarrow bAB \mid \varepsilon$

$B \rightarrow BAa \mid A \mid \varepsilon$

Soln:

Step 1: Simplify the Grammar:

- **Eliminate ε – Productions:**

  - Find nullable variables Vnullable = {A, B}
  - Construct the production in P'.

  P' : $S \rightarrow abAB \mid abB \mid abA \mid ab$

  $A \rightarrow bAB \mid bB \mid bA \mid b$

  $B \rightarrow BAa \mid Ba \mid Aa \mid a \mid A$

- **Eliminate Unit Productions:**

  $B \rightarrow A$ is a Unit Production. Replace A by its productions,

  $S \rightarrow abAB \mid abB \mid abA \mid ab$

  $A \rightarrow bAB \mid bB \mid bA \mid b$

  $B \rightarrow BAa \mid Ba \mid Aa \mid a \mid bAB \mid bB \mid bA \mid b$

- **Eliminate Useless Symbols:**

  There is no useless symbols. All the variables are generating terminal string.

Step 2: Reduce the given grammar into CNF:

Add all the productions of the form $A \rightarrow BC$ or $A \rightarrow a$.

- $S \rightarrow abAB$
  $S \rightarrow C_aC_bAB$; $C_a \rightarrow a$; $C_b \rightarrow b$
  $S \rightarrow C_aC_bD_1$; $D_1 \rightarrow AB$
  $S \rightarrow C_aD_2$; $D_2 \rightarrow C_bD_1$
- $S \rightarrow abB$
  $S \rightarrow C_aC_bB$
  $S \rightarrow C_aD_3$; $D_3 \rightarrow C_bB$
- $S \rightarrow abA$
  $S \rightarrow C_aC_bA$
  $S \rightarrow C_aD_4$; $D_4 \rightarrow C_bA$
- $S \rightarrow ab$
  $S \rightarrow C_aC_b$
- $A \rightarrow bAB$
  $A \rightarrow C_bAB$
  $A \rightarrow D_4B$
- $A \rightarrow bB$
  $A \rightarrow C_bB$
- $A \rightarrow bA$
  $A \rightarrow C_bA$
- $B \rightarrow BAa$
  $B \rightarrow BAC_a$
  $B \rightarrow BD_5$; $D_5 \rightarrow AC_a$
- $B \rightarrow Ba$
  $B \rightarrow BC_a$
- $B \rightarrow Aa$
  $B \rightarrow AC_a$
- $B \rightarrow bAB$
  $B \rightarrow C_bAB$
  $B \rightarrow C_bD_1$
- $B \rightarrow bB$
  $B \rightarrow C_bB$
- $B \rightarrow bA$
  $B \rightarrow C_bA$

The Resultant Grammar is,

$S \rightarrow C_aD_2 \mid C_aD_3 \mid C_aD_4 \mid C_aC_b$
$C_a \rightarrow a$; $C_b \rightarrow b$
$D_2 \rightarrow C_bD_1$; $D_1 \rightarrow AB$
$D_3 \rightarrow C_bB$; $D_4 \rightarrow C_bA$
$A \rightarrow D_4B \mid C_bB \mid C_bA \mid b$ $B \rightarrow BD_5 \mid BC_a \mid AC_a \mid a \mid C_bD_1 \mid C_bB \mid C_bA \mid b$
$D_5 \rightarrow AC_a$

(2) Greibach Normal Form (GNF)

A context free grammar G is reduced to GNF of every productions of the form A→aα., where 'a' is terminal, $\alpha \in (V \cup T)$ also 'α' can be empty if α→ε then A→α. The construction of GNF depends on two lemmas.

Lemma 1:

Let G = (V, T, P, S) be a CFG.

Let $A \rightarrow \alpha_1 B \alpha_2$ is a production in G.

Let $B \rightarrow \beta_1 \mid \beta_2 \mid \ldots\ldots\ldots \mid \beta_m$.

Then a new grammar G_1 can be constructed by replacing 'B' by its productions.

$\therefore A \rightarrow \alpha_1\beta_1\alpha_2 \mid \alpha_1\beta_2\alpha_2 \mid \ldots\ldots\ldots \alpha_1\beta_m\alpha_2$

Lemma 2:

Let G = (V, T, P, S) be a CFG. Let the set of A – productions be,

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots\ldots\ldots \mid A\alpha_m$

$A \rightarrow \beta_1 \mid \beta_2 \mid \ldots\ldots\ldots \mid \beta_n$

(or)

$A \rightarrow A\alpha_i \mid \beta_i$

Introduce a new variable say 'B', 'P_1' can be formed by replacing the A – production by,

$A \rightarrow A\alpha \mid \beta$
$A \rightarrow \beta \mid \beta B$
$B \rightarrow \alpha \mid \alpha B$

Reducing a CFG to GNF:

To reduce a grammar into GNF form, the following actions have to be performed.

(1) Construct a grammar CFG, G in CNF generating the CFL L. Rename the variables in V as $\{A_1, A_2, \ldots A_n\}$ with start symbol as A_1.

(2) Modify the productions, such that, $A_i \rightarrow A_j\gamma$, where $i < j$.

(3) If $A_k \rightarrow A_j\gamma$ is a production with $j < k$, generated a new set of productions by substituting for A_j.

(4) By repeating this we obtain the productions of the form $A_k \rightarrow A_l\gamma$, $l \geq k$. The production with l=k are replaced according to lemma2 by introducing a new variable.

PROBLEMS:

Ex.1:

Construct a GNF for the following grammar,

$S \rightarrow AA \mid a$

$A \rightarrow SS \mid b$

Soln:

Step 1:

The given grammar is in CNF. Rename the variables 'S' and 'A' as 'A_1' and 'A_2'. Modify the productions,

$A_1 \rightarrow A_2A_2 \mid a$

$A_2 \rightarrow A_1A_1 \mid b$

Step 2:

Check $A_i \rightarrow A_j$, where $i < j$.

- A_1 productions are in the required form. (ie, $i < j$, where $i = 1, j = 2$)
- A_2 productions are not in the required form.

  ie) $A_2 \rightarrow A_1A_1 \mid b$ ($i > j$, where $i = 2, j = 1$)

∴ Replace the leftmost symbol by its productions,

$A_2 \rightarrow (A_2A_2 \mid a)\ A_1 \mid b$

$A_2 \rightarrow A_2A_2A_1 \mid aA_1 \mid b$

Now, the production $A_2 \rightarrow A_2A_2A_1$ is of the form $A_k \rightarrow A_k$.

Step 3:

By introducing a new variable 'B', ie, in the form of;

$A \rightarrow A\alpha \mid \beta$

$A \rightarrow \beta \mid \beta B$

$B \rightarrow \alpha \mid \alpha B$

$A_2 \rightarrow A_2A_2A_1 \mid aA_1 \mid b$

Here, $\alpha = A_2A_1$

$\beta = aA_1 \mid b$

$A_2 \rightarrow aA_1 \mid b \mid (aA_1 \mid b)\ B$

$A_2 \rightarrow aA_1 \mid b \mid aA_1B \mid bB$

$B \rightarrow A_2A_1 \mid A_2A_1B$

Now the productions are,

$$\begin{bmatrix} A_1 \rightarrow A_2A_2 \mid a \\ A_2 \rightarrow aA_1 \mid b \mid aA_1B \mid bB \\ B \rightarrow A_2A_1 \mid A_2A_1B \end{bmatrix}$$

Step 4:

- $A_1 \rightarrow A_2A_2 \mid a$, replace leftmost A_2 by,
  $A_1 \rightarrow (aA_1 \mid b \mid aA_1B \mid bB)\ A_2 \mid a$
  $A_1 \rightarrow aA_1A_2 \mid bA_2 \mid aA_1BA_2 \mid bBA_2 \mid a$
- $B \rightarrow A_2A_1 \mid A_2A_1B$, replace leftmost A_2 by,
  $B \rightarrow (aA_1 \mid b \mid aA_1B \mid bB)\ A_1 \mid (aA_1 \mid b \mid aA_1B \mid bB)\ A_1B$
  $B \rightarrow aA_1A_1 \mid bA_1 \mid aA_1BA_1 \mid bBA_1 \mid aA_1A_1B \mid bA_1B \mid aA_1BA_1B \mid bBA_1B$

∴ The resultant GNF is,

$A_1 \rightarrow aA_1A_2 \mid bA_2 \mid aA_1BA_2 \mid bBA_2 \mid a$
$A_2 \rightarrow aA_1 \mid b \mid aA_1B \mid bB$
$B \rightarrow aA_1A_1 \mid bA_1 \mid aA_1BA_1 \mid bBA_1 \mid aA_1A_1B \mid bA_1B \mid aA_1BA_1B \mid bBA_1B$

Ex.2:

Convert the following grammar in GNF,

$A_1 \rightarrow A_2A_3$
$A_2 \rightarrow A_3A_1 \mid b$
$A_3 \rightarrow A_1A_2 \mid a$

Soln:

Step 1:

The given grammar is in CNF and it is in the form of $A_1, A_2, \ldots A_n$. So there is no change in step 1.

Step 2:

Check $A_i \rightarrow A_j$, where $i < j$.

- A_1 productions are in the required form. (ie, $i < j$, where $i = 1, j = 2$)
- A_2 productions are in the required form. (ie, $i < j$, where $i = 2, j = 3$)
- A_3 productions are not in the required form
  ie) $A_3 \rightarrow A_1A_2$ ($i > j$, where $i = 3, j = 1$)

∴ Replace the leftmost symbol by its productions,

$A_3 \rightarrow (A_2A_3)\ A_2 \mid a$
$A_3 \rightarrow A_2A_3A_2 \mid a$

This is also not in the required form, (ie, $i < j$, where $i = 3, j = 2$). So replace the leftmost symbol by,

$A_3 \rightarrow (A_3A_1 \mid b)\ A_3A_2 \mid a$
$A_3 \rightarrow A_3A_1A_3A_2 \mid bA_3A_2 \mid a$

Now, the production $A_3 \rightarrow A_3A_1A_3A_2$ is of the form $A_k \rightarrow A_k$.

Step 3:

By introducing a new variable 'B', ie, in the form of;

$$A \rightarrow A\alpha \mid \beta$$
$$A \rightarrow \beta \mid \beta B$$
$$B \rightarrow \alpha \mid \alpha B$$

$A_3 \rightarrow A_3A_1A_3A_2 \mid bA_3A_2 \mid a$

Here, $\alpha = A_1A_3A_2$

$\beta = bA_3A_2 \mid a$

$A_3 \rightarrow bA_3A_2 \mid a \mid (bA_3A_2 \mid a)\ B$

$A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2B \mid aB$

$B \rightarrow A_1A_3A_2 \mid A_1A_3A_2B$

Now the productions are,

$$\begin{cases} A_1 \rightarrow A_2A_3 \\ A_2 \rightarrow A_3A_1 \mid b \\ A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2B \mid aB \\ B \rightarrow A_1A_3A_2 \mid A_1A_3A_2B \end{cases}$$

Step 4:

- $A_2 \rightarrow A_3A_1 \mid b$, replace leftmost A_3 by,
  $A_2 \rightarrow (bA_3A_2 \mid a \mid bA_3A_2B \mid aB)\ A_1 \mid b$
  $A_2 \rightarrow bA_3A_2A_1 \mid aA_1 \mid bA_3A_2BA_1 \mid aBA_1 \mid b$
- $A_1 \rightarrow A_2A_3$, replace leftmost A_2 by,
  $A_1 \rightarrow (bA_3A_2A_1 \mid aA_1 \mid bA_3A_2BA_1 \mid aBA_1 \mid b)\ A_3$
  $A_1 \rightarrow bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2BA_1A_3 \mid aBA_1A_3 \mid bA_3$
- $B \rightarrow A_1A_3A_2 \mid A_1A_3A_2B$, replace leftmost A_1 by,
  $B \rightarrow (bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2BA_1A_3 \mid aBA_1A_3 \mid bA_3)\ A_3A_2 \mid (bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2BA_1A_3 \mid$
  $aBA_1A_3 \mid bA_3)\ A_3A_2B$
  $B \rightarrow bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_2BA_1A_3A_3A_2 \mid aBA_1A_3A_3A_2 \mid bA_3A_3A_2 \mid bA_3A_2A_1A_3A_3A_2B \mid$
  $aA_1A_3A_3A_2B \mid bA_3A_2BA_1A_3A_3A_2B \mid aBA_1A_3A_3A_2B \mid bA_3A_3A_2B$
  ∴ The resultant GNF is,

$A_1 \rightarrow bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2BA_1A_3 \mid aBA_1A_3 \mid bA_3$
$A_2 \rightarrow bA_3A_2A_1 \mid aA_1 \mid bA_3A_2BA_1 \mid aBA_1 \mid b$
$A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2B \mid aB$
$B \rightarrow bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_2BA_1A_3A_3A_2 \mid aBA_1A_3A_3A_2 \mid bA_3A_3A_2 \mid$
$bA_3A_2A_1A_3A_3A_2B \mid aA_1A_3A_3A_2B \mid bA_3A_2BA_1A_3A_3A_2B \mid aBA_1A_3A_3A_2B \mid$
$bA_3A_3A_2B$

4.3. PUMPING LEMMA FOR CFL

Pumping lemma for CFL states that in any sufficiently long string in a CFL, it is possible to find at most two short substrings close together that can be repeated, both of the strings same number of times.

4.3.1. Statement of the Pumping Lemma:

The pumping lemma for CFL's is similar to the pumping lemma for regular language, but we break each string 'z' in the CFL, L into five parts.

Theorem:

Let L be any CFL. Then there is a constant 'n', depending only on L, such that if 'z' is in L and $|z| \geq n$, then we may write z=uvwxy such that,

(1) $|vx| \geq 1$
(2) $|vwx| \leq n$ and
(3) For all $i \geq 0$ uv^iwx^iy is in L.

Proof:

If 'z' is in L(G) and 'z' is long then any parse tree for 'z' must contain a long path. If the parse tree of a word generated by a Chomsky Normal Form grammar has no path of length greater than 'i' then the word is of length no greater than 2^{i-1}.

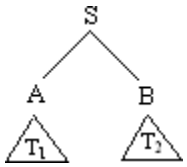
To prove this,

Basis:

Let i=1, the tree must look like,

Induction:

Let i > 1, the root and its sons be,



If there are no paths of length greater than i-1 in trees T_1 and T_2 then the trees generate words of 2^{i-2}. Let G have 'k' variable and let $n=2^k$. If 'z' is in L(G) and $|z| \geq n$ then since $|z| > 2^{k-1}$ any parse for 'z' must have a path of length atleast k+1. But such a path has atleast k+2 vertices. Then there must be some variables that appear twice in a path since there are only 'k' variables.

Let 'P' be a path that is a long than any path in the tree. Then there must be 2 vertices V_1 and V_2 on the path satisfying following conditions,

(1) The vertices V_1 and V_2 both have same label say 'A'.
(2) Vertex V_1 is closer to the root than vertex V_2.
(3) The portion of the path from V_1 to leaf is of length atmost k+1.

The subtree T_1 with root 'r_1' represents the derivation of length atmost 2^k. There is no path in T_1 of length greater than k+1, since 'P' was the longest path.

4.3.2. Applications of the Pumping Lemma:

Pumping Lemma can be used to prove a variety of languages not to be context free. To show that a language L is not context free, we use the following steps;

(i) Assume L is context free. Let 'n' be the natural number obtained by using pumping lemma.
(ii) Choose $|z| \geq n$, write z = uvwxy using the lemma.
(iii) Find a suitable integer 'i' such that $uv^iwx^iy \notin L$. This is a contradiction, and so L is not context free.

PROBLEMS:

Ex.1:

Show that $L = \{a^nb^nc^n \mid n \geq 1\}$ is not context free.

Soln:

Assume L is context free.
L = {abc, aabbcc, aaabbbccc,..... }
Let z = uvwxy.
Take a string in L = aabbcc [Take any string in L]
To prove, aabbcc is not regular.

Case 1:

z = aabbcc ; n = 6

Now, divide 'z' into uvwxy.

Let u = aa, v = b, w = b, x = ε, y = cc [∴ $|vx| \geq 1$, $|vwx| \leq n$]

Find, uv^iwx^iy. When i = 2,

uv^iwx^iy = aabbbcc

∴ aabbbcc ∉ L.

So L is not context free.

Case 2:

z = aabbcc ; n = 6

Now, divide 'z' into uvwxy.

Let u = a, v = a, w = b, x = ε, y = bcc [∴ $|vx| \geq 1$, $|vwx| \leq n$]

Find, uv^iwx^iy. When i = 2,

uv^iwx^iy = aaabbcc

∴ aaabbcc ∉ L.

So L is not context free.

Ex.2:

Show that $L = \{0^n1^n2^n \mid n \geq 1\}$ is not context free.

Soln:

Assume L is context free.

L = {012, 001122, 000111222,.....}

Let z = uvwxy.

Take a string in L = 001122 [Take any string in L]

To prove, 001122 is not regular.

Case 1:

z = 001122 ; n = 6

Now, divide 'z' into uvwxy.

Let u = 00, v = 1, w = 1, x = ε, y = 22 [∴ $|vx| \geq 1$, $|vwx| \leq n$]

Find, uv^iwx^iy. When i = 2,

uv^iwx^iy = 0011122

∴ 0011122 ∉ L.

So L is not context free.

Case 2:

z = 001122 ; n = 6

Now, divide 'z' into uvwxy.

Let u = 0, v = 0, w = 1, x = ε, y = 122 [∴ |vx| ≥ 1, |vwx| ≤ n]

Find, uv^iwx^iy. When i = 2,

$uv^iwx^iy = 0001122$

∴ 0001122 ∉ L.

So L is not context free.

4.4. CLOSURE PROPERTIES OF CFL

We now consider some operations that preserve context free languages. The operations are useful not only in constructing or proving that certain languages are context free but also in proving certain languages not to be context free.

4.4.1. Closure Under Union:

Theorem:

Context Free Languages are closed under union.

Proof:

Let 'L_1' and 'L_2' be CFL's generated by CFG's.

$G_1 = \{V_1, T_1, P_1, S_1\}$, $G_2 = \{V_2, T_2, P_2, S_2\}$

For $L_1 \cup L_2$, construct grammar G_3,

$G_3 = \{V_1 \cup V_2 \cup \{S_3\}, T_1 \cup T_2, P_3, S_3\}$

Where, P_3 is $P_1 \cup P_2$ plus the production $S_3 \rightarrow S_1 \mid S_2$.

If a string 'w' is in L_1 then the derivation, $S_3 \underset{G_3}{\Rightarrow} S_1 \underset{G_1}{\Rightarrow}{}^*$ w is in derivation in G_3, as every production of G_1 is a production of G_1 is a production of G3.

Thus $L_1 \subseteq L(G_3)$.

Similarly for a string w_1 in L_2, $S_3 \underset{G_3}{\Rightarrow} S_2 \underset{G_2}{\Rightarrow}{}^*$ w_1 is a derivation in G_3, as every of G_2 is a production of G_3.

Thus $L_2 \subseteq L(G_3)$. Hence $L(G_3) = L_1 \cup L_2$

4.4.2. Closure Under Concatenation:

Theorem:

Context Free Languages are closed under concatenation.

Proof:

Let 'L_1' and 'L_2' be CFL's generated by CFG's.

$G_1 = \{V_1, T_1, P_1, S_1\}$, $G_2 = \{V_2, T_2, P_2, S_2\}$

For $L_1 . L_2$, construct grammar G_1,

$G_1 = \{V_1 \cup V_2 \cup \{S_4\}, T_1 \cup T_2, P_4, S_4\}$

Where, P_4 is $P_1 \cup P_2$ plus the production $S_4 \rightarrow S_1S_2$.

$\therefore L(G_4) = L(G_1) . L(G_2)$

4.4.3. Closure Under Kleene Closure:

Theorem:

Context Free Languages are closed under kleene closure.

Proof:

Let 'L_1' and 'L_2' be CFL's generated by CFG's.

$G1 = \{V_1, T_1, P_1, S_1\}$

For closure, let $G_5 = \{V_1 \cup \{S_5\}, T_1, P_5, S_5\}$

Where, P_5 is P_1 plus the production $S_5 \rightarrow S_1S_5 \mid \varepsilon$.

If a string 'w' is in L then the derivation, $S_5 \Rightarrow S_1S_5 \Rightarrow wS_5 \Rightarrow_w$ $[S_5 \Rightarrow \varepsilon]$

$\therefore L(G_5) = L(G_1)^*$

4.4.4. Closure Under Substitution:

Theorem:

Context Free Languages are closed under substitution.

Proof:

Let L be a CFL, $L \subseteq \Sigma^*$ and for each 'a' in Σ, let L_a be a CFL. Let L be a L(G) and for each 'a' in Σ, let L_a be $L(G_a)$. Construct a grammar G' as follows;

- The variables of G' are all the variables of G and G_a's.
- The terminals of G' are the terminals of the G_a's.
- The start symbols of G' are all the production of the G_a's.
- The productions of G' are all the productions of the G_a's together with those productions formed by taking a production $A \rightarrow \alpha$ of G and substituting S_a, the start symbol of G_a, for each instance of an 'a' in Σ appearing in α.

Ex:

Let L be the set of words with an equal number of a's and b's,

$L_a = \{0^n1^n \mid n \geq 1\}$ and $L_b = \{ww^R \mid w \text{ in } (0+2)^*\}$

For G we may choose, $S \rightarrow aSbS \mid bSaS \mid \varepsilon$.

For G_a take, $S_a \rightarrow 0S_a1 \mid 01$

For G_b take, $S_b \rightarrow 0S_b0 \mid 2S_b2 \mid \varepsilon$

If 'f' is the substitution $f(a) = L_a$ and $f(b) = L_b$, then, f(L) is generated by grammar G' as,

$$S \rightarrow S_aSS_bS \mid S_bSS_aS \mid \varepsilon$$
$$S_a \rightarrow 0S_a1 \mid 01$$
$$S_b \rightarrow 0S_b0 \mid 2S_b2 \mid \varepsilon$$

4.4.5. Closure Under Intersection:

Theorem:

Context Free Languages are closed under intersection.

Proof:

Let L_1 and L_2 be CFL's, then $L_1 \cap L_2$ is a language, where it satisfies both the properties of L_1 and L_2 which is not possible in CFL.

Ex:

Let $L_1 = \{a^m b^n \mid m \geq 1, n \geq 1\}$, $L_2 = \{a^n b^m \mid n \geq 1, m \geq 1\}$

$L = L_1 \cap L_2$ is not possible.

Because L_1 requires that there be 'm' number of a's and 'n' mumber of b's.

So CFL's are not closed under intersection.

4.4.6. Closure Under Homomorphism:

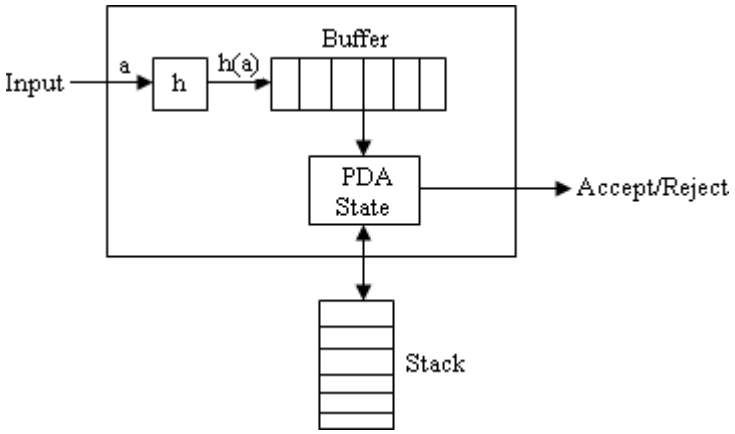
Let L be a CFL over the alphabet Σ and 'h' is a homomorphism on Σ. Let 'S' be the substitution that replaces each symbol 'a' in Σ by the language consisting of one string h(a).

$S(a) = \{h(a)\}$ for all 'a' in Σ.

Thus $h(L) = S(L)$.

4.4.7. Closure Under Inverse Homomorphism:

If 'h' is a homomorphism and 'L' is any language then $h^{-1}(L)$ is the set of strings 'w' such that h(w) is in L. Thus CFL's are closed under inverse homomorphism. The following fig., shows the inverse homomorphism of PDA's.



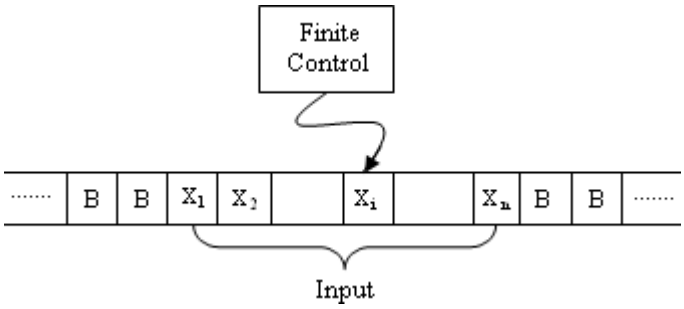
After getting the input 'a', h (a) is placed in a buffer. The symbols of h(a) are used one at a time and fed to the PDA being simulated. While applying homomorphism the PDA checks whether the buffer is empty. If it is empty, then the PDA read the input symbols and applies the homomorphism.

4.5. TURING MACHINES (TM)

- A Turing Machine is a simple, abstract mathematical model of a computer.
- It is developed by "Alan Turing", during the year 1936.
- Turing Machine is introduced as a tool for studying computability of mathematical functions.
- Turing Machine is mostly used to define languages and to compute integer functions.

4.5.1. Notation for the Turing Machine:

- The Turing Machine consists of a finite control, and a input tape.
- Finite control has a finite set of states.
- Input tape is divided into cells, each cell can hold any one of a finite number of symbols.



- Initially, the input, which is a finite length of symbols, that is placed on the tape.
- All other tape cells, extending infinitely to the left and, right can hold a special symbol called **Blank**.
- The blank is a tape symbol, but not an input symbol.
- The model of the turing machine also has a **tape head**, that is always positioned at one of the tape cells.
- Initially the tape head is pointing the leftmost cell that holds the input.

4.5.2. Move of the Turing Machine:

In one move, the turing machine depending upon the symbol scanned by the tape head and the state of finite control.

(1) It changes the state.
(2) Writes a tape symbol.
(3) Moves tape head one cell, to its left or right.

4.5.3. Formal Definition of Turing Machine:

A turing machine M has 7 – tuples,

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Where, Q – The finite set of states.

Σ – The finite set of input symbols.

Γ – Finite set of tape symbols.

δ - The transition function. The arguments of δ is,

$\delta(q, A) = (p, B, L)$

where, 'q' is the current state.

A, B is the input symbols.

L is the direction (L → Left, R → Right)

'p' is the next state.

q_0 – The start state.

B – The blank symbol, blank symbol is in tape symbol, but not in input symbols.

F – The set of final or accepting states.

4.5.4. Instantaneous Descriptions(ID) for Turing Machines:

Instantaneous Description (ID) of the turing machine M is denoted by $\alpha_1 q \alpha_2$. Here 'q' is the current state, α_1 and α_2 is the string. We define a move of M as follows;

Let $X_1 X_2 \ldots\ldots X_{i-1}\ q\ X_i X_{i+1} \ldots\ldots X_n$ be an ID, suppose $\delta(q, X_i) = (p, Y, L)$, then, we write move as,

$X_1 X_2 \ldots\ldots X_{i-1}\ q\ X_i X_{i+1} \ldots\ldots X_n \} X_1 X_2 \ldots\ldots p\ X_{i-1} Y\ X_{i+1} \ldots\ldots X_n$

Alternatively, $\delta(q, X_i) = (p, Y, R)$, then, we write move as,

$X_1 X_2 \ldots\ldots X_{i-1}\ q\ X_i X_{i+1} \ldots\ldots X_n \} X_1 X_2 \ldots\ldots X_{i-1} Y\ p\ X_{i+1} \ldots\ldots X_n$

PROBLEMS:

Ex.1:

Design a TM to accept the language $L = \{0^n 1^n \mid n \geq 1\}$

Soln:

Step 1: Place $0^n 1^n$ on the tape following by infinite blank symbols.

Step 2: Replace leftmost '0' by 'X' and move right to find the leftmost '1'.

Step 3: The leftmost '1' is replaced by 'Y', moves left to find rightmost 'X', then moves one cell right to the

leftmost '0' and repeats the cycle.

Step 4: While searching, for a '1', if M finds a blank, then M halts without accepting.

Step 5: If after changing '1' to 'Y', M finds no more 0's then M checks whether there is any more 1's left out.

Step 6: If none, then it accepts the string else not.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, assume the set of states $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0,1,X,Y,B\}$, $q_0 = \{q_0\}$, $F = \{q_4\}$.

Assume n= 2 then the input string is: 0011 [ie., $0^2 1^2$]

Assume n= 2 then the input string is: 0011 [ie., $0^2 1^2$]

(i) 0011BBB ⊢ X011BBB ⊢ X011BBB ⊢ X0Y1BBB ⊢ X0Y1BBB ⊢ X0Y1BBB ⊢ XXY1BBB
$\uparrow_{q_0}$ $\uparrow_{q_1}$ $\uparrow_{q_1}$ $\uparrow_{q_2}$ $\uparrow_{q_2}$ $\uparrow_{q_0}$ $\uparrow_{q_1}$

⊢ XXY1BBB ⊢ XXYYBBB ⊢ XXYYBBB ⊢ XXYYBBB ⊢ XXYYBBB ⊢ XXYYBBB ⊢ XXYYBBB
$\uparrow_{q_1}$ $\uparrow_{q_2}$ $\uparrow_{q_2}$ $\uparrow_{q_0}$ $\uparrow_{q_2}$ $\uparrow_{q_3}$

$\uparrow_{q_4}$ ∴ Accepted.

(ii) Suppose the input string is 0010 then,

0010BBB ⊢ X010BBB ⊢ X010BBB ⊢ X0Y0BBB ⊢ X0Y0BBB ⊢ X0Y0BBB ⊢ XXY0BBB
$\uparrow_{q_0}$ $\uparrow_{q_1}$ $\uparrow_{q_1}$ $\uparrow_{q_2}$ $\uparrow_{q_2}$ $\uparrow_{q_0}$ $\uparrow_{q_1}$

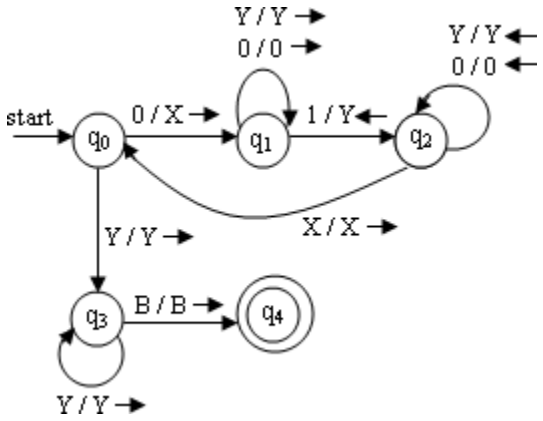
⊢ XXY0BBB ⊢ XXY0BBB

$\uparrow_{q_1}$ $\uparrow_{q_1}$ ∴ Not accepted.

Transition Table:

State	0	1	X	Y	B
→q_0	(q_1,X,R)	-	-	(q_3,Y,R)	-
q_1	(q_1,0,R)	(q_2,Y,L)	-	(q_1,Y,R)	-
q_2	(q_2,0,L)	-	(q_0,X,R)	(q_2,Y,L)	-
q_3	-	-	-	(q_3,Y,R)	(q_4,B,R)
*q_4	-	-	-	-	-

Transition Diagram:



Ex.2:

Design a Turing Machine for the language L = $\{0^n1^n0^n \mid n \geq 1\}$

Soln:

Let M = (Q, Σ, Γ, δ, q_0, B, F), assume the set of states Q = {q_0, q_1, q_2, q_3, q_4, q_5}, Σ = {0, 1}, Γ = {0,1,X,Y,Z,B}, q_0 = {q_0}, F = {q_5}.

Assume n= 2 then the input string is: 001100 [ie., $0^2 1^2 0^2$]

001100BBB ⊢ X01100BBB ⊢ X01100BBB ⊢ X0Y100BBB ⊢ X0Y100BBB ⊢ X0Y1Z0BBB

⊢ X0Y1Z0BBB ⊢ X0Y1Z0BBB ⊢ X0Y1Z0BBB ⊢ X0Y1Z0BBB ⊢ XXY1Z0BBB

⊢ XXY1Z0BBB ⊢ XXYYZ0BBB ⊢ XXYYZ0BBB ⊢ XXYYZZBBB ⊢ XXYYZZBBB

⊢ XXYYZZBBB ⊢ XXYYZZBBB ⊢ XXYYZZBBB ⊢ XXYYZZBBB ⊢ XXYYZZBBB

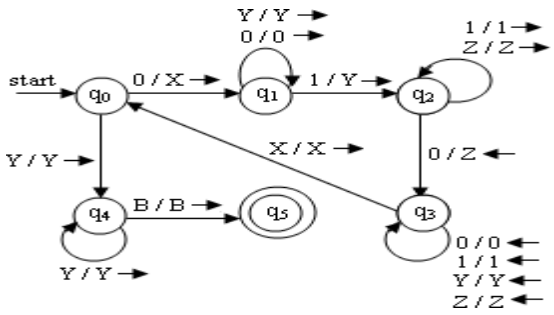
⊢ XXYYZZBBB ⊢ XXYYZZBBB ⊢ XXYYZZBBB

∴ Accepted.

Transition Table:

State	0	1	X	Y	Z	B
→q_0	(q_1,X,R)	-	-	(q_4,Y,R)	-	-
q_1	(q_1,0,R)	(q_2,Y,R)	-	(q_1,Y,R)	-	-
q_2	(q_3,Z,L)	(q_2,1,R)	-	-	(q_2,Z,R)	-
q_3	(q_3,0,L)	(q_3,1,L)	(q_0,X,R)	(q_3,Y,L)	(q_3,Z,L)	-
q_4	-	-	-	(q_4,Y,R)	(q_4,Z,R)	(q_5,B,R)
*q_5	-	-	-	-	-	-

Transition Diagram:



Ex.3:

Design a TM to recognize the language, $L = \{ww^R \mid w \text{ in } (0+1)^*\}$

Soln:

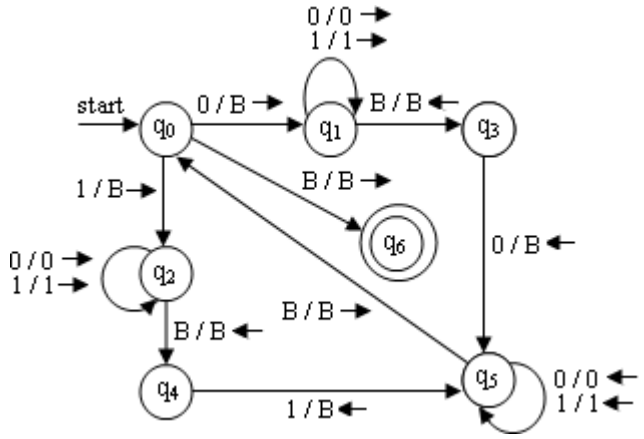
Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, assume the set of states $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0,1,B\}$, $q_0 = \{q_0\}$, $F = \{q_6\}$.

Assume the input string is: 010010 [ie.,w = 010, w^R = 010]

Transition Table:

State	0	1	B
$\rightarrow q_0$	(q_1,B,R)	(q_2,B,R)	(q_6,B,R)
q_1	$(q_1,0,R)$	$(q_1,1,R)$	(q_3,B,L)
q_2	$(q_2,0,R)$	$(q_2,1,R)$	(q_4,B,L)
q_3	(q_5,B,L)	-	-
q_4	-	(q_5,B,L)	-
q_5	$(q_5,0,L)$	$(q_5,1,L)$	(q_0,B,R)
$*q_6$	-	-	-

Transition Diagram:



Ex.1:

Construct a TM that performs addition.

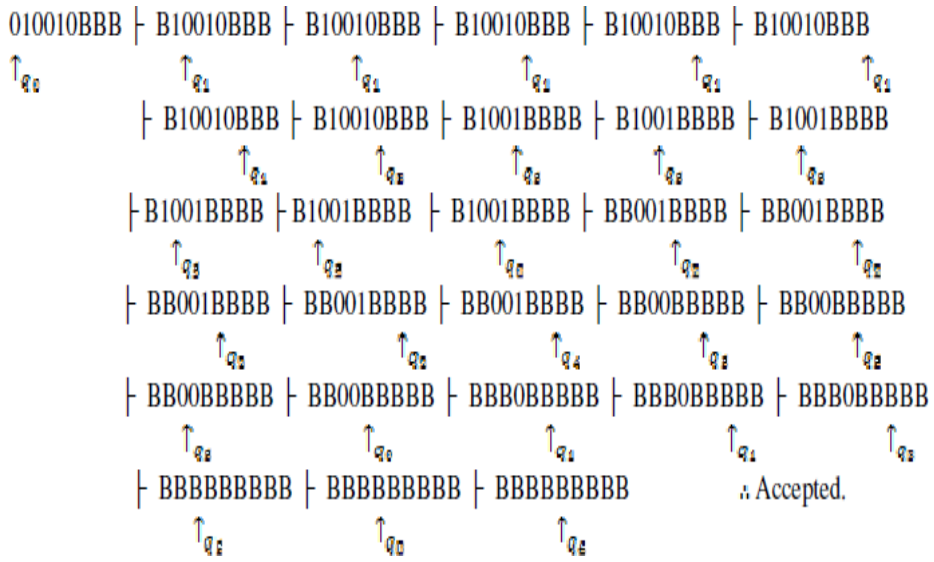
Soln:

Procedure:

- The function is defined as f(x, y) = x+y.
  'x' is given by 0^x.
  'y' is given by 0^y.
- The input is placed on tape as $0^x|0^y$, where '|' is the separator.
- Then the output will be 0^{x+y}.
- Starting from the first zero in the 0^x, the tape head moves till it finds a separator '|' and replaces it by '0', move right to find the blank symbol.
- Then moves left one cell and replace the zero in that cell by a blank symbol.

Let M = (Q, Σ, Γ, δ, q_0, B, F), assume the set of states Q = {q_0, q_1, q_2, q_3}, Σ = {0, 1}, Γ = {0,1,B}, q_0 = {q_0}, F = {q_3}.

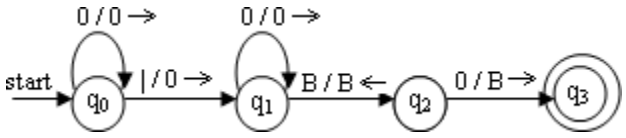
Assume x = 3, y = 2 then, input string is: $0^3|0^2$ ==> 000|00BBB



Transition Table:

State	0	|	B
→q_0	(q_0,0,R)	(q_1,0,R)	-
q_1	(q_1,0,R)	-	(q_2,B,L)
q_2	(q_3,B,R)	-	-
*q_3	-	-	-

Transition Diagram:



Ex.2:

Construct a TM to compute the function, f(x) = x+1

Soln:

- 'x' is given by 0^x.
- ∴ f(x) = x+1 = 0^{x+1}.
- The output contains one more '0' than the input.
- Initially the TM is at q_0.
- At q_0 if it reads a blank symbol by skipping 0's, replace it with '0' and enters final state.

Let M = (Q, Σ, Γ, δ, q_0, B, F), assume the set of states Q = {q_0, q_1}, Σ = {0}, Γ = {0, B}, q_0 = {q_0}, F = {q_1}.

Assume x = 3 then, input string is: 0^3 ==> 000BBB

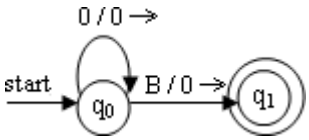
000BBB } 000BBB } 000BBB } 000BBB } 0000BB

↑ ↑ ↑ ↑ ↑

Transition Table:

State	0	B
→q_0	(q_0,0,R)	(q_1,0,R)
*q_1	-	-

Transition Diagram:



Ex.3:

Design a TM to compute proper subtraction.

Soln:

Proper subtraction is defined by m ∸ n.

ie) m ∸ n = max(m – n, 0)

$$\begin{cases} m - n & \text{if } m > n \end{cases}$$

$m \dot{-} n$ is

$$0 \quad \text{if} \quad m < n$$

Procedure:

- The TM start its operation with $0^m|0^n$ on its input tape.
- Initially the TM is at state q_0.
- At q_0, it replaces the leading '0' by blank and search right looking for first '|'.
- After finding it, the TM searches right for '0' and change it to '|'.
- Then move the tape head to left till reaches the blank symbol. And then enter state q_0 to repeat the cycle.

The repetition ends if:

(1) Searching right for a '0', TM encounters a blank. Then n 0's in $0^m|0^n$ have all been changed to B. Replace the $(n+1)^{th}$ '|' by '0' and n B's. Leaving m-n 0's on its tape.

(2) TM cannot find a '0' to change it to blank during the beginning of the cycle. Change all zero's and 1's to blank and the result in zero.

Let M = (Q, Σ, Γ, δ, q_0, B, F), assume the set of states Q = {q_0, q_1, q_2, q_3, q_4, q_5, q_6}, Σ = {0, 1}, Γ = {0,1,B}, q_0 = {q_0}, F = {q_6}.

(i) Assume m=2, n=1 then, Input string is: 00|0

(ii)

$$\begin{array}{l} 00|0BBB \vdash B0|0BBB \vdash B0|0BBB \vdash B0|0BBB \vdash B0|1BBB \vdash B0|1BBB \vdash B0|1BBB \vdash B0|1BBB \\ \uparrow_{q_0} \quad \uparrow_{q_1} \quad \uparrow_{q_1} \quad \uparrow_{q_2} \quad \uparrow_{q_3} \quad \uparrow_{q_3} \quad \uparrow_{q_3} \quad \uparrow_{q_0} \\ \vdash BB|1BBB \vdash BB|1BBB \vdash BB|1BBB \vdash BB|1BBB \vdash BB|BBBB \vdash BBBBBBB \\ \quad \uparrow_{q_1} \quad \uparrow_{q_2} \quad \uparrow_{q_2} \quad \uparrow_{q_4} \quad \uparrow_{q_4} \quad \uparrow_{q_4} \\ \vdash B0BBBBB \\ \quad \uparrow_{q_6} \end{array}$$

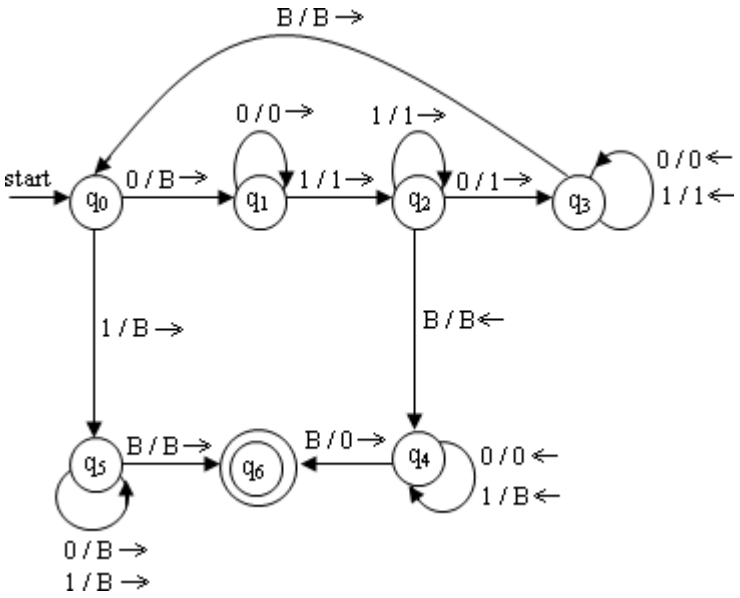
(iii) Assume m=1, n=2 then, input string is: 0|00

$$\begin{array}{l} 0|00BB \vdash B|00BB \vdash B|00BB \vdash B|10BB \vdash B|10BB \vdash B|10BB \vdash BB10BB \vdash BBB0BB \\ \uparrow_{q_0} \quad \uparrow_{q_1} \quad \uparrow_{q_2} \quad \uparrow_{q_3} \quad \uparrow_{q_3} \quad \uparrow_{q_0} \quad \uparrow_{q_2} \quad \uparrow_{q_5} \\ \vdash BBBBBB \vdash BBBBBB \\ \quad \uparrow_{q_5} \quad \uparrow_{q_6} \end{array}$$

Transition Table:

State	0	1	B
$\rightarrow q_0$	(q_1,B,R)	(q_5,B,R)	-
q_1	$(q_1,0,R)$	$(q_2,1,R)$	-
q_2	$(q_3,1,L)$	$(q_2,1,R)$	(q_4,B,L)
q_3	$(q_3,0,L)$	$(q_3,1,L)$	(q_0,B,R)
q_4	$(q_4,0,L)$	(q_4,B,L)	$(q_6,0,R)$
q_5	(q_5,B,R)	(q_5,B,R)	(q_6,B,R)
$*q_6$	-	-	-

Transition Diagram:



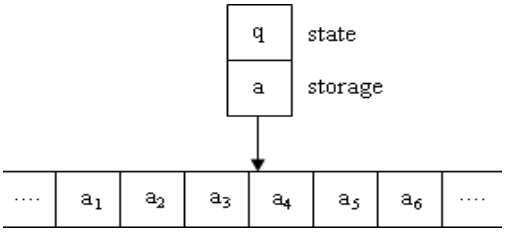
4.6. PROGRAMMING TECHNIQUES FOR TM

There are different techniques are used for constructing Turing Machine. They are,

(1) Storage in the state.
(2) Multiple Tracks
(3) Subroutines

(1) Storage in the state:

The finite control holds a finite amount of information. Then the state of the finite control is represented as a **pair of elements**. The first element represents the **state** and the second element represents **storing a symbol**.



Ex.1:

Construct a TM, M=(Q, {0,1}, {0,1,B}, δ, [q_0,B], Z0, [q_1,B]), that look at the first input symbol records in the finite control and checks that symbol does not appear else where on its input.

Soln:

For the states of Q as, $Q \times \{0,1,B\} = \{q_0, q_1\} \times \{0,1,B\}$

$Q = \{[q_0, 0], [q_0, 1], [q_0, B], [q_1, 0], [q_1, 1], [q_1, B]\}$

In this, the finite control holds a pair of symbol, that is, both the state and the symbol.

(i) $\delta([q_0, B], a) = ([q_1, a], a, R)$; where, a=0 (or) 1

At 'q_0', the TM reads the first symbol 'a' and goes to state 'q_1'. The input symbol is coped into the second component of the state and moves right.

(ii) $\delta([q_1, a], \bar{a}) = ([q_1, a], \bar{a}, R)$; where, $\bar{a}$ is the complement of 'a'.

ie) if a = 0 then $\bar{a} = 1$

if a = 1 then $\bar{a} = 0$

At q_1, if the TM reads the other symbols, M skips over and moves right.

(iii) $\delta([q_1, a], B) = ([q_1, B], B, R)$

If M reaches the same symbol, it halts without enters accepting.

(iv) $\delta([q_1, a], B) = ([q_1, B], B, R)$

If M reaches the first blank, then it enters the accepting state.

Input String: 011BBB

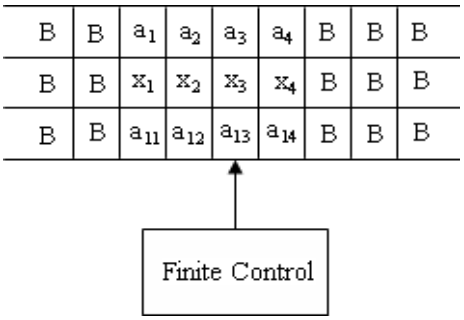
011BBB } 011BBB } 011BBB } 011BBB } 011BBB

↑ ↑ ↑ ↑ ↑

$[q_0, B]$ $[q_1,0]$ $[q_1,0]$ $[q_1,0]$ $[q_1, B]$

(2) Multiple Tracks:

It is possible that a Turing Machine's input tape can be divided into several tracks. Each track can hold symbols.



Ex.1:

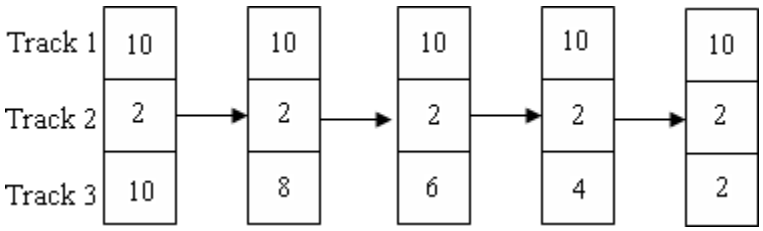
Construct a TM that takes an input greater than 2 and checks whether it is even or odd.

Soln:

Procedure:

(1) The input is placed into first tape or track.
(2) The integer 2 is placed on the second track.
(3) The input on the first track is copied into third track.
(4) The number on the second track is subtracted from the third track.
(5) If the remainder is same as the number in the second track then the number on the first track is even.
(6) If it is greater than 2, then continue this process until the remainder in the third track is <= 2, if it is equal to 2 then the number is even otherwise it is odd.

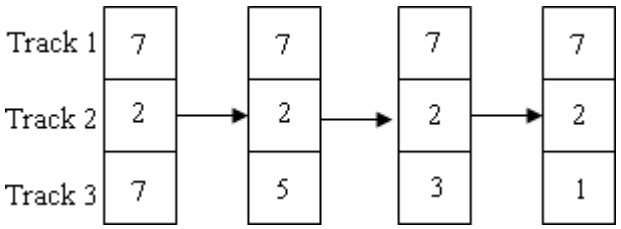
(i) Take the input ==> 10



Finally second track number and the third track number is equal.

∴ The given number is even.

(ii) Assume the input ==> 7



∴ The given number is odd.

Ex.2:

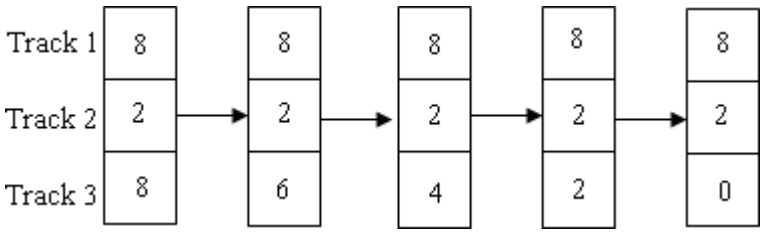
Design a TM that takes an input greater than 2 and checks whether the given input is prime or not.

Soln:

Procedure:

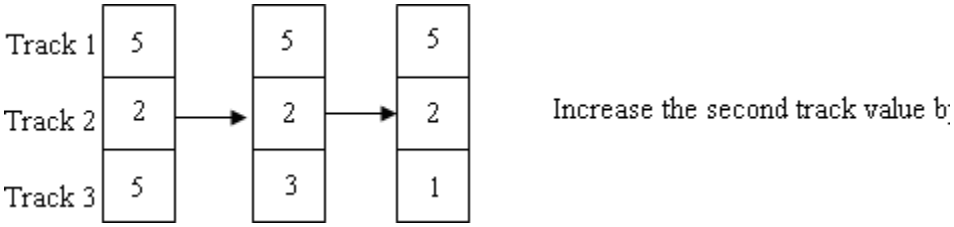
(1) The input is placed into first track.
(2) The integer 2 is placed on the second track.
(3) The input on the first track is copied into third track.
(4) The number on the second track is subtracted from the third track.
(5) If the remainder is zero, then the number on the first track is not a prime.
(6) If the remainder is non – zero, then increase the number on the second track by one.
(7) If the second track equals the first track, then the given number is prime.

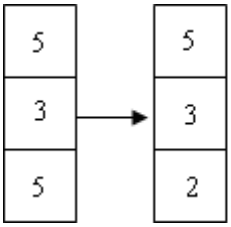
(i) Assume the input ==> 8



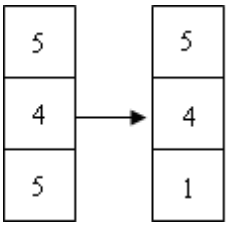
∴ The given number is not a prime number.

(ii) Assume the input ==> 5





Increase the second track value by 1



Increase the second track value by 1

5
5
5

Here the number on second track is equal to the number on the first track.

∴ The given number is prime.

(3) <u>Subroutines:</u>

Subroutines are used in computer languages, which performs some task repeatedly. A turing machine can simulate any type of subroutine found in programming languages. A part of the TM program can be used as subroutine. This subroutine can be called for any number of times in the main TM program.

Ex:

Design a TM to implement multiplication function, f(m,n) = m*n

Soln:

'm' is given by 0^m

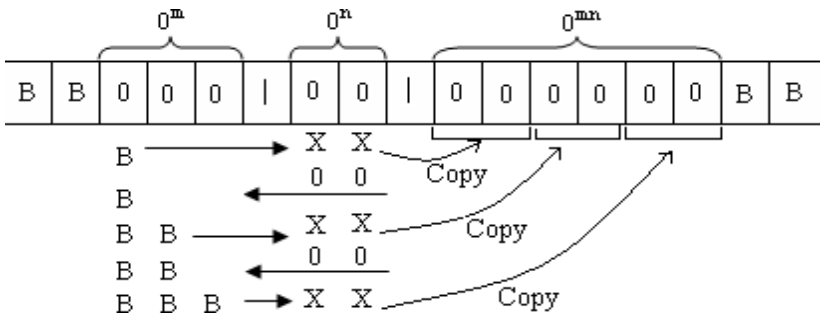
'n' is given by 0^n

Input is : $o^m | o^n$

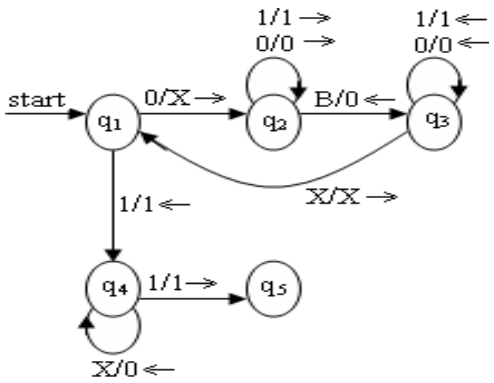
Output is : o^{mn}

Input and output is placed into the tape, that is, $o^m | o^n | o^{mn}$

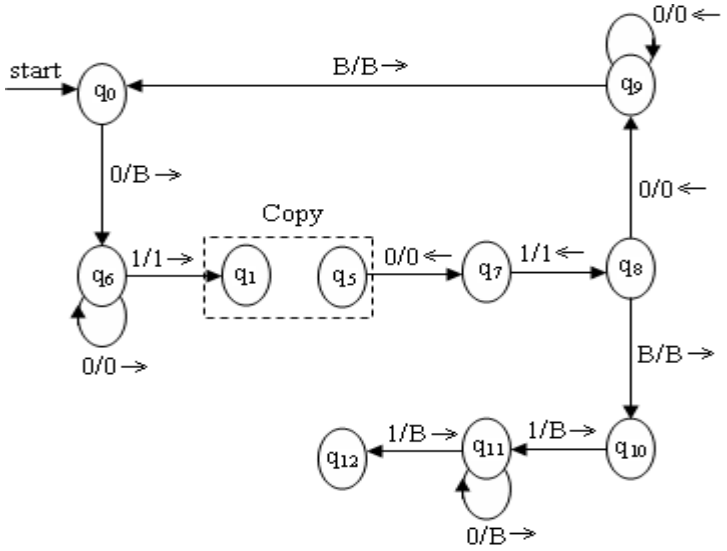
i/p o/p



The main concept is, it copy 'n' zero's 'm' times.

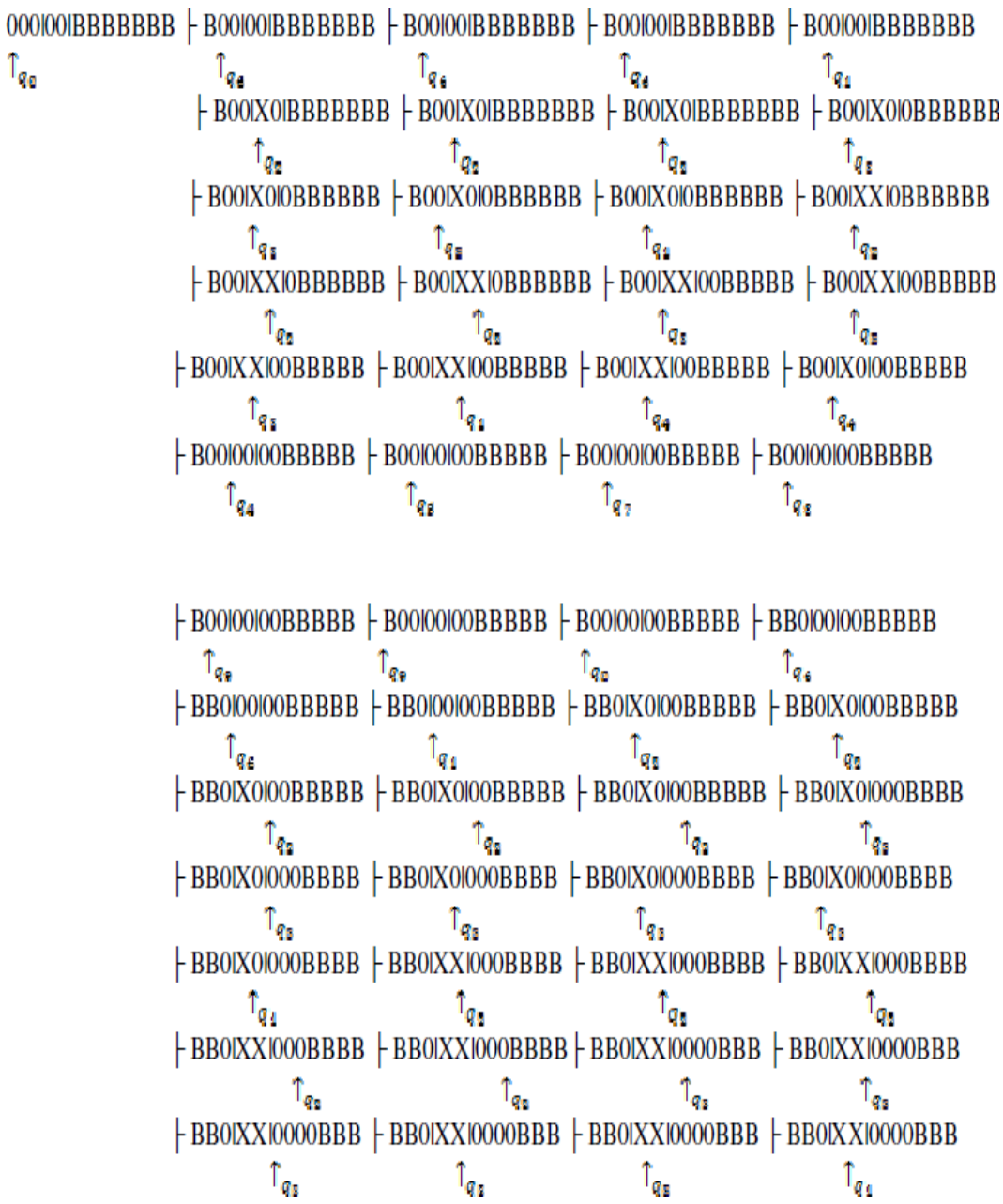


The Subroutine Copy



The Complete Multiplication Program Uses in Subroutine Copy

Assume m = 3, n = 2 then, input is $0^3|0^2$, that is placed on the input tape as,


000|00|BBBBBBB ├ B00|00|BBBBBBB ├ B00|00|BBBBBBB ├ B00|00|BBBBBBB ├ B00|00|BBBBBBB
├ B00|X0|BBBBBBB ├ B00|X0|BBBBBBB ├ B00|X0|BBBBBBB ├ B00|X0|0BBBBBB
├ B00|X0|0BBBBBB ├ B00|X0|0BBBBBB ├ B00|X0|0BBBBBB ├ B00|XX|0BBBBBB
├ B00|XX|0BBBBBB ├ B00|XX|0BBBBBB ├ B00|XX|00BBBBB ├ B00|XX|00BBBBB
├ B00|XX|00BBBBB ├ B00|XX|00BBBBB ├ B00|XX|00BBBBB ├ B00|X0|00BBBBB
├ B00|00|00BBBBB ├ B00|00|00BBBBB ├ B00|00|00BBBBB ├ B00|00|00BBBBB
├ B00|00|00BBBBB ├ B00|00|00BBBBB ├ B00|00|00BBBBB ├ BB0|00|00BBBBB
├ BB0|00|00BBBBB ├ BB0|00|00BBBBB ├ BB0|X0|00BBBBB ├ BB0|X0|00BBBBB
├ BB0|X0|00BBBBB ├ BB0|X0|00BBBBB ├ BB0|X0|00BBBBB ├ BB0|X0|000BBBB
├ BB0|X0|000BBBB ├ BB0|X0|000BBBB ├ BB0|X0|000BBBB ├ BB0|X0|000BBBB
├ BB0|X0|000BBBB ├ BB0|XX|000BBBB ├ BB0|XX|000BBBB ├ BB0|XX|000BBBB
├ BB0|XX|000BBBB ├ BB0|XX|000BBBB ├ BB0|XX|0000BBB ├ BB0|XX|0000BBB
├ BB0|XX|0000BBB ├ BB0|XX|0000BBB ├ BB0|XX|0000BBB ├ BB0|XX|0000BBB


⊢ BB0|XX|0000BBB ⊢ BB0|X0|0000BBB ⊢ BB0|00|0000BBB ⊢ BB0|00|0000BBB
↑q4 ↑q4 ↑q4 ↑q8
⊢ BB0|00|0000BBB ⊢ BB0|00|0000BBB ⊢ BB0|00|0000BBB ⊢ BB0|00|0000BBB
↑q7 ↑q8 ↑q8 ↑q10
⊢ BBB|00|0000BBB ⊢ BBB|00|0000BBB ⊢ BBB|X0|0000BBB ⊢ BBB|X0|0000BBB
↑q6 ↑q4 ↑q5 ↑q5
⊢ BBB|X0|0000BBB ⊢ BBB|X0|0000BBB ⊢ BBB|X0|0000BBB ⊢ BBB|X0|0000BBB
↑q5 ↑q5 ↑q5 ↑q5
⊢ BBB|X0|0000BBB ⊢ BBB|X0|00000BB ⊢ BBB|X0|00000BB ⊢ BBB|X0|00000BB
↑q5 ↑q5 ↑q5 ↑q5
⊢ BBB|X0|00000BB ⊢ BBB|XX|00000BB ⊢ BBB|XX|00000BB ⊢ BBB|XX|00000BB
↑q4 ↑q2 ↑q2 ↑q2
⊢ BBB|XX|00000BB ⊢ BBB|XX|00000BB ⊢ BBB|XX|00000BB ⊢ BBB|XX|00000BB
↑q5 ↑q5 ↑q5 ↑q5
⊢ BBB|XX|000000B ⊢ BBB|XX|000000B ⊢ BBB|XX|000000B ⊢ BBB|XX|000000B
↑q5 ↑q5 ↑q5 ↑q5
⊢ BBB|XX|000000B ⊢ BBB|XX|000000B ⊢ BBB|XX|000000B ⊢ BBB|XX|000000B
↑q5 ↑q3 ↑q3 ↑q4
⊢ BBB|XX|000000B ⊢ BBB|X0|000000B ⊢ BBB|00|000000B ⊢ BBB|00|000000B
↑q4 ↑q4 ↑q4 ↑q6
⊢ BBB|00|000000B ⊢ BBB|00|000000B ⊢ BBB|00|000000B ⊢ BBBB00|000000B
↑q7 ↑q8 ↑q10 ↑q11
⊢ BBBBB0|000000B ⊢ BBBBBB|000000B ⊢ BBBBBBB000000B
↑q11 ↑q11 ↑q12