Module-4

- **Context-Free and Non-Context-Free Languages**
- **Where Do the Context-Free Languages Fit in the Big Picture?**
- **Showing that a Language is Context-Free**
- **Pumping theorem for CFL**
- **Important closure properties of CFLs**
- **Deterministic CFLs**
- **Algorithms and Decision Procedures for CFLs: Decidable questions**
- **Undecidable questions**
- **Turing Machine: Turing machine model**
- **Representation**
- **Language acceptability by TM**
- **Design of TM**
- **Techniques for TM construction.**

Context-Free and Non-Context-Free Languages

- The language $A^nB^n = \{a^nb^n \mid n \geq 0\}$ is context-free.
- The language $A^nB^nC^n = \{a^nb^nc^n \mid n \geq 0\}$ is not context free because a PDA's stack cannot count all three of the letter regions and compare them.

Where Do the Context-Free Languages Fit in the Big Picture?

THEOREM: The Context-Free Languages Properly Contain the Regular Languages.

Theorem: The regular languages are a proper subset of the context-free languages.

Proof: We first show that every regular language is context-free. We then show that there exists at least one context-free language that is not regular.

Every regular language is context-free : We show this by construction.

- If L is regular then it is accepted by some DFSM $M = (K, \Sigma, \delta, s, A)$.
- From M we construct a PDA
  $M' = (K', \Sigma', \Gamma', \Delta', s', A')$ to accept L. where Δ' is constructed as follows:
  For every transition (q_i,c,q_j)in δ, add to Δ' the transition $((q_i,c,\varepsilon),(q_i,\varepsilon))$,So $L(M)=L(M')$.

So, the set of regular languages is a subset of the CFL.

There exists at least one context-free language that is not regular : The regular languages are a *proper* subset the context-free languages because there exists at least one language $\mathbf{a^n b^n}$ that is context –free but not regular.

Theorem: There is a countably infinite number of context-free languages.

Proof:

Every context-free language is generated by some context-free grammar G = (V,Σ,R,S).

There cannot be more CFLs than CFGs. So there are at most a countably infinite number of context-free languages. There is not a one-to-one relationship between CFLs and CFGs, since there are an infinite number of grammars that generate any given language. But we know that, every regular language is context free and there is a countably infinite number of regular languages.

So there is at least and at most a countably infinite number of CFLs.

Showing That a Language is Context-Free

Two techniques that can be used to show that language L is context-free:

- Exhibit a context-free grammar for it.
- Exhibit a (possibly nondeterministic) PDA for it.

Theorem: The length of the yield of any tree T with height h and branching factor b is <= b^h.

Proof:

If h is 1, then a single rule applies. So the longest yield is of length less than or equal to b.

Assume the claim is true for h=n. We show that it is true for h=n+1.

Consider any tree with h=n+1. It consists of a root, and some number of subtrees, each of height <=n. By the induction hypothesis, the length of the yield of each of those subtrees is <= b^n. So the length of the yield must be $<=b.(b^n)=b^{n+1}=b^h$.

The Pumping Theorem for Context-Free languages

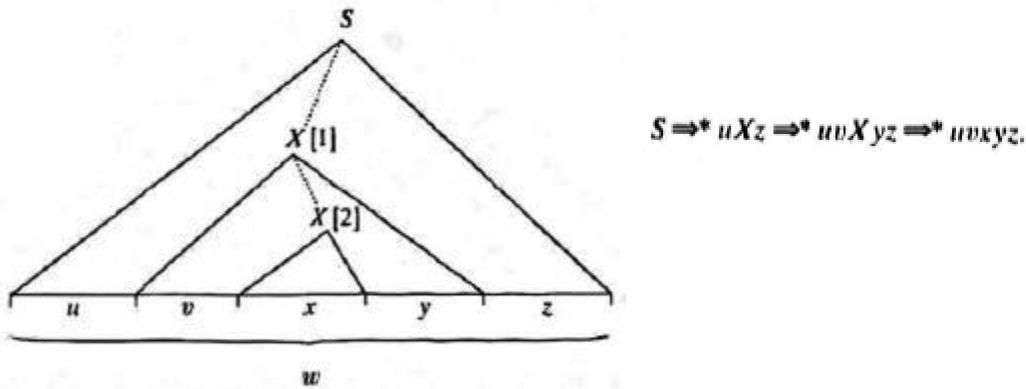
Statement: If L is CFL, then: ∃k≥1 (∀strings *w*∈L, where |w|≥k (∃u,v,x,y,z (*w*=uvxyz, vy≠Ɛ, |vxy| ≤ k and ∀q ≥ 0 ($uv^q xy^q z$ is in L))))

Proof: If L is context-free, then there exists a CFG G=(V,Σ,R,S) with n nonterminal symbols and branching factor b.

Let k be b^{n+1}.

Any string that can be generated by G and whose parse tree contains no paths with repeated nonterminals must have length less than or equal to b^n. Assuming that $b \geq 2$, it must be the case that $b^{n+1} > b^n$. So let w be any string in L(G) where $|w| \geq k$.

Let T be any smallest parse tree for w. T must have height at least n+1. Choose some path in T of length at least n + 1. Let X be the bottom-most repeated non terminal along that path. Then w can be rewritten as uvxyz as shown in below tree,



The tree rooted at [1] has height at most n+1.Thus its yield, vxy, has length less than or equal to b^{n+1},which is k. Further, $vy \neq \mathcal{E}$ .Since if vy were $\mathcal{E}$ then there would be a smaller parse tree for w and we choose T so that h at wasn't so.

Finally, v and y can be pumped: uxz must be in L because rule2 could have been used immediately at[1]. And, for any $q \geq 1$, $uv^q xy^q z$ must be in L because rule1 could have been used q times before finally using rule2.

Application of pumping lemma (Proving Language is Not Context Free)

Ex1: Prove that the Language $L = \{a^n b^n c^n | n>=0\}$ is Not Context-Free.

Solution: If L is CFL then there would exist some k such that any string w, where $|w|>=k$ must satisfy the conditions of the theorem.

Let $w = a^k b^k c^k$, where 'k' is the constant from the Pumping lemma theorem. For w to satisfy the conditions of the Pumping Theorem there must be some u,v,x,y and z, such that w=uvxyz, $vy \neq \mathcal{E}$, $|vxy| \leq k$ and $\forall q \geq 0$, $uv^q xy^q z$ is in L.

w=aaa…aaabbb…bbbccc…ccc, select v and y as follows:

w=aaa…a|**aabbb**|**..bbb**|**cc**|c…ccc

(v = **aabbb**, y = **cc**)

Let q=2, then

w=aaa…a|aabbaabb |b..bb|cccc|..ccc

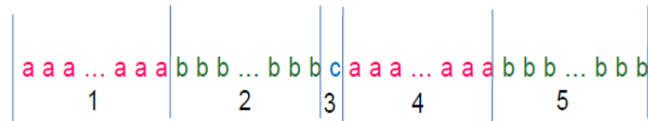
(v^2 = aabbaabb, y^2 = cccc)

The resulting string will have letters out of order and thus not in L.

So L is not context-free.

Ex 2: Prove that the Language L= {WcW: w∈{a,b}*}is Not Context-Free.

For w to satisfy the conditions of the Pumping Theorem there must be some u,v,x,y,and z, such that $w = uvxyz$, $vy \neq \varepsilon$, $|vxy| \leq k$ and $\forall q \geq 0$, uv^qxy^qz is in L. We show that no such u,v,x,y and z exist. Imagine w divided into five regions as follows:



Call the **part before the c the leftside and the part after the c the right side.** We consider all the cases for where v and y could fall and show that in none of them are all the condition so f the theorem met:

- If either v or y overlaps region 3, set q to 0. The resulting string will no longer contain a c and so is not in WcW.
- If both v and y occur before region 3 or they both occur after region 3, then set q to 2. One side will be longer than the other and so the resulting string is not in WcW.
- If either v or y overlaps region1 ,then set q to 2. In order to make the right side match. Something would have to be pumped into region 4. But any v,y pair that did that would violate the requirement that $|vxy| \leq k$.
- If either v or y overlaps region2, then set q to 2. In order to make the right side match, something would have to be pumped into region 5. But any v,y pair that did that would violate the requirement that $|vxy| \leq k$.
- **There is no way to divide w into uvxyz such that all the conditions of the Pumping Theorem are met . So WcW is not context-free.**

Some Important Closure Properties of Context-Free Languages

Theorem: The context- free languages are closed under Union, Concatenation, Kleene star, Reverse, and Letter substitution.

(1) The context-free languages are closed under union:

- If L1 and L2 are context free languages then there exists a context-free grammar $G_1= (V_1,\Sigma_1, R_1,S_1)$ and $G_2=(V_2,\Sigma_2,R_2,S_2)$ such that $L_1=L(G_1)$ and $L_2=L(G_2)$.

• We will build a new grammar G such that $L(G)=L(G_1)UL(G_2)$. G will contain all the rules of both G_1 and G_2.

• We add to G a new start symbol S and two new rules. $S \rightarrow S_1$ and $S \rightarrow S_2$. The two new rules allow G to generate a string iff at least one of G_1 or G_2 generates it.

So, $G = (V_1 \text{ U } V_2 \text{ U } \{S\}, \Sigma_1 \text{ U } \Sigma_2, R_1 \text{ U } R_2 \text{ U } \{S \rightarrow S_1, S \rightarrow S_2\}, S)$

(2)The context-free languages are closed under concatenation

• If L_1 and L_2 are context free languages then there exist context-free grammar $G_1= (V_1,\Sigma_1,R_1,S_1)$ and $G_2=(V_2,\Sigma_2,R_2,S_2)$ such that $L_1= L(G_1)$ and $L_2= L(G_2)$.

• We will build a new grammar G such that $L (G) = L(G_1)L(G_2)$.

• G will contain all the rules of both G_1 and G_2.

• We add to G a new start symbol S and one new rule. $S \rightarrow S_1S_2$

So $G= (V_1UV_2U\{S\}, \Sigma_1 \text{ U } \Sigma_2, R_1UR_2U\{S \rightarrow S_1S_2),S)$

(3) The context-free Languages are closed under Kleene star:

• If L_1 is a context free language then there exists a context-free grammar $G_1=(V_1,\Sigma_1,R_1,S_1)$ such that $L_1= L(G_1)$.

• We will build a new grammar G such that $L(G)=L(G_1)^*$ G will contain all the rules of G_1.

• We add to G a new start symbol S and two new rules. $S \rightarrow \mathcal{E}$ and $S \rightarrow SS_1$

So $G = (V_1 \text{ U } \{S\}, \Sigma_1, R_1 \text{U } \{S \rightarrow \mathcal{E}, S \rightarrow SS_1), S)$

(4) The context-free languages are closed under reverse

• If L is a context free language then it is generated by some Chomsky Normal Form from grammar $G= (V,\Sigma,R, S)$.

• Every rule in G is of the form $X \rightarrow BC$ or $X \rightarrow a$, where X, B, and C are elements of $(V-\Sigma)$ and $a \in \Sigma$

• So construct, from G, a new grammar G^1 , Such that $L(G^1)= L^R$.

• $G^1= (V_G, \Sigma_G, R', S_G)$, Where R' is constructed as follows:

  - For every rule in G of the form $X \rightarrow BC$, add to R' the rule $X \rightarrow CB$
  - For every rule in G of the form $X \rightarrow a$ then add to R' the rule $X \rightarrow a$

(5)The context-free languages are closed under letter Substitution

• Consider two alphabets Σ_1 and Σ_2 .

•Let ***sub*** be any function from Σ_1 to Σ_2^*.

•Then ***letsub*** is a letter substitution function from L_1 to L_2 iff **letsub(L_1)** $=\{ w\in \Sigma_2^*: \exists y \in L1$ (w=y except that every character c of y has replaced by sub(c))}.

Example : Let y= VTU $\in L_1$ And sub(c) is given as : sub(V) = Visvesvaraya

sub(T) = Technological

sub(U)= University

Then , sub(VTU) = Visvesvaraya Technological University

Closure Under Intersection, Complement, and Difference

Theorem:The Context-free language are not closed under intersection, complement or difference.

1) The context-free languages are not closed under intersection

The proof is by counter example. Let: $L_1=\{a^nb^nc^m|n,m\geq 0\}$ $L_2=\{a^mb^nc^n|n,m\geq 0\}$ Both $L_1$and L_2 are context-free since there exist straight forward CFGs for them.

But now consider: L $=L_1\cap L_2=\{ a^nb^nc^n \mid n,m\geq 0\}$. If the context-free languages were closure under intersection. L would have to be context-free. But we have proved that L is not CFG by using pumping lemma for CFLs.

(2) The context-free languages are not closure under

Given any sets $L_1$and L_2 , $L_1\cap L_2 = \neg(\neg L_1\cup\neg L_2)$

• The context-free languages are closed under union.

• But we just showed that they are not, thus they are not closed under complement either.

• So, if they were also closed under complement, they would necessarily be closed under intersection.

(3) The context-free languages are not closed under difference (subtraction) :

Given any language L and $\neg L=\Sigma^*$- L.

Σ^* is context-free So, if the context-free languages were closed under difference, the complement of any CFL would necessarily be context-free But we just showed that is not so.

Closure Under Intersection With the Regular Languages

Theorem: The context-free languages are closed under intersection with the regular languages.

Proof: The proof is by construction.

- If L_1 is context-free, then there exists some PDA $M_1 = (K_1, \Sigma, \Gamma_1, \Delta_1, S_1, A_1)$ that accepts it.
- If L_2 is regular then there exists a DFSM $M_2 = (K_2, \Sigma, \delta, S_2, A_2)$ that accepts it.
- We construct a new PDA, M_3 that accepts $L_1 \cap L_2$. M_3 will work by simulating the parallel execution of M_1 and M_2.
- $M_3 = (K_1 X K_2, \Sigma, \Gamma_1, \Delta_3, (S_1, S_2), A_1 X A_2)$, Where Δ_3 is built as follows:
    - For each transition $((q_1, a, \beta), (p_1, \gamma))$ in Δ_1 and each transition $((q_2, a), p_2)$ in δ, add Δ_3 the transition: $(((q_1, q_2), a, \beta), ((p_1, p_2), \gamma))$.
    - For each transition $((q_1, \varepsilon, \beta), (p_1, \gamma))$ in Δ_1 and each state q_2 in k_2, add to Δ_3 the transition: $(((q_1, q_2), \varepsilon, \beta), ((p_1, p_2), \gamma))$.

Closure Under Difference with the Regular Language.

Theorem: The difference (L_1-L_2) between a context-free language L_1 and a regular language L_2 is context-free.

Proof: $L_1 - L_2 = L_1 \cap \neg L_2$

- If L_2 is regular, then, since the regular languages are closed under complement, $\neg L_2$ is also regular.
- Since L_1 is context-free, by Theorem we already proved that $L_1 \cap \neg L_2$ is context-free.

Using the Pumping Theorem in Conjunction with the Closure Properties

Languages that impose no specific order constraints on the symbols contained in their strings are not always context-free. But it may be hard to prove that one isn't just by using the Pumping Theorem. In such a case it is proved by considering the fact that the context-free languages are closed under intersection with the regular languages.

Deterministic Context-Free Languages

The technique used to show that the regular languages are closed under complement starts with a given (possibly nondeterministic) FSM M_1, we used the following procedure to construct a new FSM M_2 such that $L(M_2) = \neg L(M_1)$:

The regular languages are closed under complement, intersection and difference. Why are the context-free languages different? Because the machines that accept them may necessarily be nondeterministic.

1. From M_1, construct an equivalent DFSM M', using the algorithm ***ndfsmtodfsm***, presented in the proof of Theorem5.3. (If $M_1$is already deterministic. M'=M_1.)

2. M' must be stated completely. so if it is described with an implied dead state, add the dead state and all required transitions to it.

3. Begin building M_2 by setting it equal to M'. Then swap the accepting and the non-accepting states. So M_2 $M'= (K_{M'},.\Sigma,\delta_{M'},s_{M'},K_{M'}\text{-}A_{M'})$.

We have no PDA equivalent of **ndfstodfsm** because there provably isn't one. We defined a PDA M to be deterministic iff:

- Δ_M contains opairs of transitions that compete with each other, and
- if q is an accepting state of M, then there is no transition ((q,ε,ε),(p,a)) for any p or a.

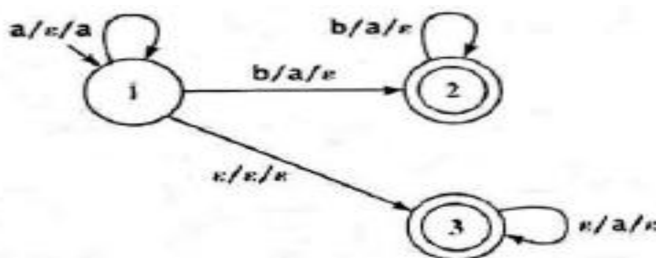
What is a Deterministic Context-Free language?

- Let $ be an end-of-string marker. A language L is deterministic context-free iff L$ can be accepted by some deterministic PDA.

EXAMPLE: Why an End-of-String Marker is Useful

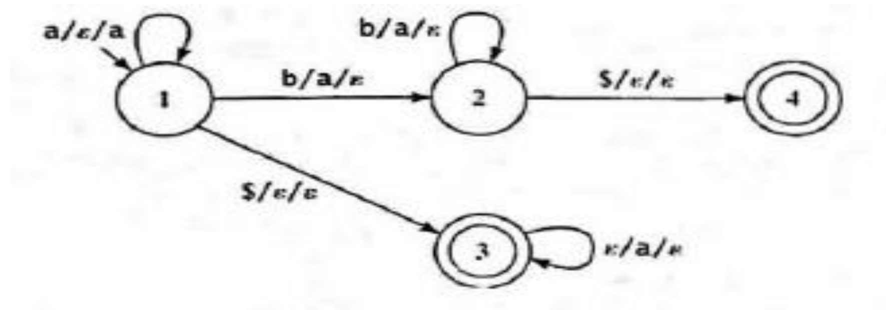
Let L= $a^* \cup \{ a^n b^n | n>0\}$

Consider any PDA M that accepts L. When it begins reading a's, M must push them onto the stack in case there are going to be b's. But if it runs out of input without seeing b's, it needs a way to pop those a's from the stack before it can accept. Without an end-of-string marker, there is no way to allow that popping to happen only when all the input has been read.



For example, the PDA accepts L, but it is nondeterministic because the transition to state3 (where the a's will be popped) can compete with both of the other transitions from state1.

With an end-of-string marker, we can build the deterministic PDA, which can only take the transition to state3, the a-popping state. When it sees the $:



NOTE: Adding the end-of-string marker cannot convert a language that was not context-free into one that is.

CFLs and Deterministic CFLs

Theorem: Every deterministic context-free language is context-free.

Proof:

If L is deterministic context-free, then L$ is accepted by some deterministic PDA $M=(K,\Sigma,\Gamma,\Delta,s,A)$. From M, we construct M' such that L (M') = L. We can define the following procedure to construct M':

without$(M:PDA)=

1.Initially. set M' to M.

/*Make the copy that does not read any input.

2.For every state q in M, add to M' a new state q'.

3.For every transition $((q, \varepsilon, \gamma_1),(p,\gamma_2))$ in Δ M do:

3.1. Add to Δ_M the transition $((q',\varepsilon,\gamma_1),(p',\gamma_2))$.

/*Link up the two copies.

4.For every transition $((q,\$,\gamma_1),(p,\gamma_2))$ in Δ M do:

4.1. Add to $\Delta_{M'}$ the transition $((q,\varepsilon,\gamma_1),(p',\gamma_2))$.

4.2. Remove $((q,\$,\gamma_1),(p,\gamma_2))$ from $\Delta_{M'}$

/*Set the accepting state s of M'.

5.$A_{M'} = \{q':q \in A\}$.

Closure Properties of the Deterministic Context-Free Languages

1) Closure Under Complement

Theorem: The deterministic context-free languages are closed under complement.

Proof: The proof is by construction. If L is a deterministic context-free language over the alphabet Σ, then L$ is accepted by some deterministic PDA M = (K, Σ U{$}, ***Γ***, ***Δ***, s, A).

We need to describe an algorithm that constructs a new deterministic PDA that accepts (¬L)$.

We defined a construction that proceeded in two steps:

- Given an arbitrary FSM, convert it to an equivalent DFSM, and then
- Swap accepting and non accepting states.

A deterministic PDA may fail to accept an input string *w* for any one of several reasons:

1. Its computation ends before it finishes reading *w*.
2. Its computation ends in an accepting state but the stack is not empty.
3. Its computation loops forever, following ε-transitions, without ever halting in either an accepting or a non accepting state.
4. Its computation ends in a non accepting state.

If we simply swap accepting and non accepting states we will correctly fail to accept every string that M would have accepted (i.e., every string in L$). But we will not necessarily accept every string in (¬L)$. To do that, we must also address issues 1through 3 above.

An additional problem is that we don't want to accept ¬L(M). That includes strings that do not end in $. We must accept only strings that do end in $ and that are in (¬L)$.

2) Non closure Under Union

Theorem: The deterministic context-free languages are not closed under union.

Proof: We show a counter example:

Let, L_1={ $a^ib^jc^k$| i,j,k ≥0 and i≠ j } and L_2={ $a^ib^jc^k$ | i,j,k≥0 and j ≠ k}

Let, L' = L_1U L_2= { $a^ib^jc^k$ | i,j,k≥0 and ((i≠ j) and (j ≠ k)) }.

Let, L'' = ¬L'.

$=\{ a^ib^jc^k \mid i,j,k\geq 0$ and $(i=j=k)\}$ U $\{w\in\{a,b,c\}^*$: the letters are out of order$\}$.

Let, $L'''=L''\cap a^*b^*c^* = \{a^nb^nc^n \mid n \geq 0\}$

But L''' is $A^nB^nC^n=\{ a^nb^nc^n \mid n\geq 0\}$,which we have shown is not context-free.

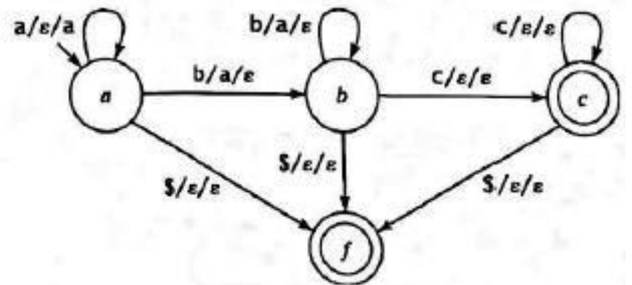
3) Non Closure Under Intersection

Theorem: The deterministic context-free languages are not closed under inter section.

Proof: We show a counter example:

Let, $L_1= \{ a^ib^jc^k \mid i,j,k\geq 0$ and $i= j \}$ and $L_2=\{ a^ib^jc^k \mid i,j,k\geq 0$ and $j=k\}$

Let, $L' = L_1\cap L_2 =\{ a^nb^nc^n \mid n \geq 0\}$



L_1 and L_2 are deterministic context-free. The deterministic PDA shown accepts $L_1\$$, A similar one accepts L_2. But we have shown that their intersection L' is not context-free much less deterministic context-free.

A hierarchy within the class of context-free languages

Some CFLs are not Deterministic

Theorem: The class of deterministic context-free languages is a proper subset of the class of context-free languages. Thus there exist nondeterministic PDAs for which no equivalent deterministic PDA exists.

Proof: We show that there exists at least one context-free language that is not deterministic context-free.

Consider $L = \{ a^ib^jc^k \mid i,j,k \geq 0$ and $((i\neq j)$ or $(j \neq k)) \}$. L is context-free.

If L were deterministic context-free, then, its complement

$L'=\{ a^ib^jc^k \mid i,j,k \geq 0$ and $(i=j=k)$ $\}$ U $\{w\in\{a,b,c\}^*$:the letters are out of order$\}$

Would also be deterministic context-free and thus context-free. If L' were context-free, then $L''=L'\cap a^*b^*c^*$ would also be context-free (since the context-free languages are closed under inter section with the regular languages).

But $L''= A^nB^nC^n =\{a^nb^nc^n \mid n\geq 0\}$,which is not context free.

So L is context-free but not deterministic context-free.

Since L is context-free, it is accepted by some (non deterministic) PDA M. M is an example of an on deterministic PDA for which no equivalent deterministic PDA L exists. If such a deterministic PDA did exist and accept L, it could be converted into a deterministic PDA that accepted L$. But, if that machine existed. L would be deterministic context-free and we just showed that it is not.

Inherent Ambiguity versus Non determinism

There are context-free languages for which unambiguous grammars exist and there are others that are inherently ambiguous, by which we mean that every corresponding grammar is ambiguous.

Example:

The language $L_1 = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } ((i= j) \text{ or } (j = k))\}$ can also be described as $\{a^n b^n c^m | n,m \geq 0\} U \{a^n b^m c^m | n,m \geq 0\}$. L_1 is inherently ambiguous because every string that is also in $A^n B^n C^n = \{a^n b^n c^n | n \geq 0\}$ is an element of both sub languages and so has at least two derivations in any grammar for L_1.

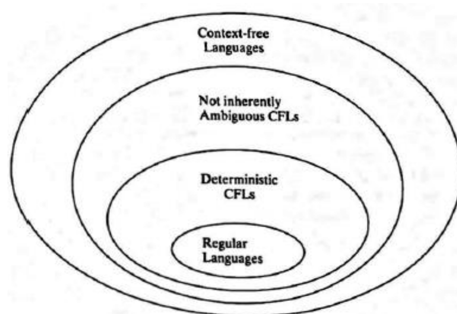
• Now consider the language $L_2 = \{a^n b^n c^m d | n,m \geq 0\} U \{a^n b^m c^m e | n,m \geq 0\}$ is not inherently ambiguous.

• Any string in is an element of only one of them (since each such string must end in d or e but not both).

There exists no PDA that can decide which of the two sublanguages a particular string is in until it has consumed the entire string.

What is the relationship between the deterministic context-free languages and the languages that are not inherently ambiguous?

The answer is shown in below Figure.



There exist deterministic context-free languages that are not regular. One example is $A^n B^n = \{a^n b^n | n,m \geq 0\}$.

•There exist context-free languages and not inherently ambiguous. Examples:

(a) $PalEven = \{ww^R : w \in \{a,b\}^*\}$.

(b) $\{a^n b^n c^m d | n,m \geq 0\} U \{a^n b^m c^m e | n,m \geq 0\}$.

•There exist languages that are in the outer donut because they are inherently ambiguous. Two examples are:

- $\{a^i b^j c^k | i,j,k \geq 0 \text{ and } ((i \neq j) \text{or} (j \neq k))\}$
- $\{a^i b^j c^k | i,j,k \geq 0 \text{ and } ((i=j) \text{ or } (j=k))\}$

Regular Language is Deterministic Context-Free

Theorem: Every regular language is deterministic context-free.

Proof: The proof is by construction. {$} is regular. So, if L is regular then so is L$ (since the regular languages are closed under concatenation).So there is a DFSM M that accepts it. Using the construction to show that every regular language is context-free Construct, from M a PDA P that accepts L$. P will be deterministic.

Every Deterministic CFL has an Unambiguous Grammar

Theorem: For every deterministic context-free language there exists an unambiguous grammar.

Proof: If a language L is deterministic context-free, then there exists a deterministic PDA M that accepts L$. We prove the theorem by construction of an unambiguous grammar G such that L(M)= L(G). We construct G as follows:

The algorithm PDAtoCFG proceeded in two steps:

1.Invoke ***convenPDAtorestricted(M)*** to build M', an equivalent PDA in restricted normal form.

2. Invoke ***buildgrammar(M'),*** to build an equivalent grammar G

So the construction that proves the theorem is:

buildunambiggrammar(M:deterministicPDA)=

1. Let G=*buildgrammar*(***convertPDAtodetnormalform***(**M**)).
2. Let G' be the result of substituting ε for $ in each rule in which $ occurs.
3. Return G'.

NOTE: The algorithm ***convertPDAtodetnormalform,*** is described in the theorem that proves the deterministic context-free languages are closed under complement.

The Decidable Questions

Membership

"Given a language L and a string *w*, is *w* in L?'

This question can be answered for every context-free language and for every context-free language L there exists a PDA M such that M accepts L. But existence of a PDA that accepts L does not guarantee the existence of a procedure that decides it.

It turns out that there are two alternative approaches to solving this problem, both of which work:

- **Use a grammar:** Using facts about every derivation that is produced by a grammar in Chomsky normal form, we can construct an algorithm that explores a finite number of derivation paths and finds one that derives a particular string w iff such a path exists.
- **Use a PDA** : While not all PDAs halt, it is possible, for any context-free language L, to craft a PDA M that is guaranteed to halt on all inputs and that accepts all strings in L and rejects all strings that are not in L.

Using a Grammar to Decide

Algorithm for deciding whether a string *w* is in a language L:

***decideCFLusingGrammar*(L: CFL,*w*: string) =**

1. If L is specified as a PDA, use ***PDA to CFG***, to construct a grammar G such that L(G) =L (M).

2. If L is specified as a grammar G, simply use G.

3. If w =ε then if S_G is nullable then accept, otherwise reject.

4. If w ≠ ε then:

4.1. From G, construct G' such that L (G') = L(G)-{ε} and G' is in Chomsky normal form.

4.2. If G derives to, it does so in (2 • |*w*| - 1) steps. Try all derivations in G of that number of steps. If one of them derives w, accept. Otherwise reject.

Using a PDA to Decide

A two-step approach.

- We first show that, for every context-free language L, it is possible to build a PDA that accepts L-{ε} and that has no ε-transitions.
- Then we show that every PDA with no ε-transitions is guaranteed to halt

Elimination of ε-Transitions

Theorem: Given any context-free grammar G=(V,Σ,R,S), there exists a PDA M such that L(M)=L(G)-{ε} and M contains no transitions of the form

$((q1,\varepsilon,\alpha),(q2,\beta))$. In other words, every transition reads exactly one input character.

Proof: The proof is by a construction that begins by converting G to Greibach normal form. Now consider again the algorithm ***cfgtoPDAtopdown***, which builds, from any context-free grammar G, a PDA M that, on input w, simulates G deriving w, starting from S.

M= $(\{p,q\},\Sigma,V,\Delta, p,\{q\})$, where Δ contains:

1. The start-up transition $((p,\varepsilon,\varepsilon),(q,S))$, which pushes the start symbol on to the stack and goes to state q.
2. For each rule $X \to s_1s_2...s_n$, in R, the transition $((q,\varepsilon,X),(q,s_1s_2...s_n))$, which replaces X by $s_1s_2...s_n$. If n=0 (i.e., the right-hand side of the rule is ε), then the transition $((q, \varepsilon, X), (q, \varepsilon))$.
3. For each character $c \in \Sigma$. the transition $((q, c, c), (q,\varepsilon))$, which compares an expected character from the stack against the next input character.

If G contains the rule $X \to cs_2...s_n$, (where $c \in \Sigma$ and s_2 through s_n, are elements of V-Σ), it is not necessary to push *c* onto the stack, only to pop it with a rule from step 3.

Instead, we collapse the push and the pop into a single transition. So we create a transition that can be taken only if the next input character is *c*. In that case, the string s_2 through s_n is pushed onto the stack.

Since terminal symbols are no longer pushed onto the stack. We no longer need the transitions created in step3 of the original algorithm.

So, M=$(\{p,q\},\Sigma,V,\Delta,p,\{q\})$, where Δ contains:

1. The start-up transitions: For each rule $S \to cs_2...s_n$ the transition $((p,c,\varepsilon),(q,s_2...s_n))$.
2. For each rule $X \to cs_2...s_n$ (where $c \in \Sigma$ and s_2 through s_n, are elements of V-Σ), the transition $((q,c,X),(q,s_2...s_n))$.

cfgtoPDAnoeps(G:context-freegrammar)=

1. Convert G to Greibach normal form, producing G'.
2. From G' build the PDA M described above.

Halting Behavior of PDAs Without ε-Transitions

Theorem: Let M be a PDA that contains no transitions of the form $((q_1,\varepsilon,s1),(q_2,s_2))$. i.e., no ε-transitions. Consider the operation of M on input $w \in \Sigma^*$. M must halt and either accept or reject w. Let n=|w|.

We make three additional claims:

a) Each individual computation of M must halt within *n* steps.

b) The total number of computations pursued by M must be less than or equal to b^n, where *b* is the maximum number of competing transitions from any state in M.

c) The total number of steps that will be executed by all computations of M is bounded by nb^n

Proof:

a) Since each computation of M must consume one character of w at each step and M will halt when it runs out of input, each computation must halt within n steps.

b) M may split into at most b branches at each step in a computation. The number of steps in a computation is less than or equal to n. So the total number of computations must be less than or equal to b^n.

c) Since the maximum number of computations is b^n and the maximum length of each is n, the maximum number of steps that can be executed before all computations of M halt is nb^n.

So a second way to answer the question, "Given a context-free language L and a string w, is w in L?" is to execute the following algorithm:

decideCFLusingPDA(L:CFL,w:string)=

1. If L is specified as a PDA, use ***PDAtoCFG*,** to construct a grammar G such that L(G)=L(M).
2. If L is specified as a grammar G, simply use G.
3. If w=ε then if S_G is nullable then accept, otherwise reject.
4. If w≠ε then:
    - 4.1. From G, construct G' such that L(G')=L(G)-{ε} and G' is in Greibach normal form.
    - 4.2. From G' construct, using ***cfgtoPDAnoeps***, a PDA M' such that L(M')=L(G') and M' has no ε-transitions.
    - 4.3. We have proved previously that, all paths of M' are guaranteed to halt within a finite number of steps. So run M' on w, Accept if M' accepts and reject otherwise.

Emptiness and Finiteness

Decidability of Emptiness and Finiteness

Theorem: Given a context-free language L. There exists a decision procedure that answers each of the following questions:

1. Given a context-free language L, is L=∅?
2. Given a context-free language L, is L infinite?

Since we have proven that there exists a grammar that generates L iff there exists a PDA that accepts it. These questions will have the same answers whether we ask them about grammars or about PDAs.

Proof :

***decideCFLempty*(G: context-free grammar) =**

1. Let G' =***removeunproductive(G).***
2. If S is not present in G' then return True else return False.

***decideCFLinfinite*(G:context-freegrammar)=**

1. Lexicographically enumerate all strings in Σ^* of length greater than b^n and less than or equal to $b^{n+1}+b^n$.
2. If, for any such string w, ***decideCFL*(L,w)** returns *True* then return *True*. L is infinite.
3. If, for all such strings w, ***decideCFL*(L,w)** returns *False* then return *False*. L is not infinite.

The Undecidable Questions

• Given a context-free language L, is $L=\Sigma^*$?

• Given a CFL L, is the complement of L context-free?

• Given a context-free language L, is L regular?

• Given two context-free languages L_1 and L_2 is $L_1=L_2$?

• Given two context-free languages L_1 and L_2, is $L_1 \subseteq L_2$?

• Given two context-free languages L_1 and L_2, is $L_1 \cap L_2=\emptyset$?

• Given a context-free language L, is L inherently ambiguous?

• Given a context-free grammar G, is G ambiguous?

TURING MACHINE

The Turing machine provides an ideal theoretical model of a computer. Turing machines are useful in several ways:

• Turing machines are also used for determining the undecidability of certain languages and

• As an automaton, the Turing machine is the most general model, It accepts type-0 languages.

• It can also be used for computing functions. It turns out to be a mathematical model of partial recursive functions.

• Measuring the space and time complexity of problems.

Turing assumed that while computing, a person writes symbols on a one-dimensional paper (instead of a two dimensional paper as is usually done) which can be viewed as a tape divided into cells. In

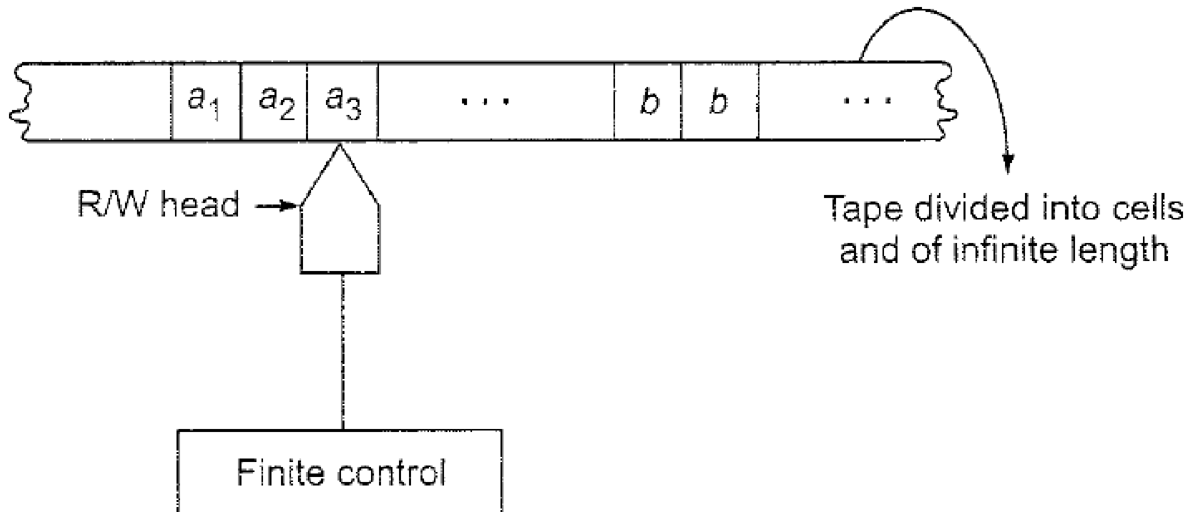
Turing machine one scans the cells one at a time and usually performs one of the three simple operations, namely:

(i) Writing a new symbol in the cell being currently scanned,

(ii) Moving to the cell left of the present cell, and

(iii) Moving to the cell right of the present cell.

Turing machine model



•Each cell can store only one symbol.

•The input to and the output from the finite state automaton are affected by the R/W head which can examine one cell at a time.

In one move, the machine examines the present symbol under the R/W head on the tape and the present state of an automaton to determine:

(i) A new symbol to be written on the tape in the cell under the R/W head,

(ii) A motion of the R/W head along the tape: either the head moves one cell left (L),or one cell right (R).

(iii) The next state of the automaton, and

(iv) Whether to halt or not.

Definition:

Turing machine M is a 7-tuple, namely (Q, Σ, $\mathbf{\Gamma}$, $\boldsymbol{\delta}$, q_0, b, F), where

1. Q is a finite nonempty set of states.
2. $\mathbf{\Gamma}$ is a finite nonempty set of tape symbols,
3. $b \in \mathbf{\Gamma}$ is the blank.
4. Σ is a nonempty set of input symbols and is a subset o f $\mathbf{\Gamma}$ and $b \notin \Sigma$.

5. **δ** is the transition function mapping (q,x) onto (q',y,D) where D denotes the direction of movement of R/W head; D=L orR according as the movement is to the left or right.
6. $q_0 \in Q$ is the initial state, and
7. $F \subseteq Q$ is the set of final states.

Notes:

(1)The acceptability of a string is decided by the reachability from the initial state to some final state.

(2) **δ** may not be defined for some elements of QX **Γ**.

REPRESENTATION OF TURINGMACHINES

We can describe a Turing machine employing

(i) Instantaneous descriptions using move-relations.

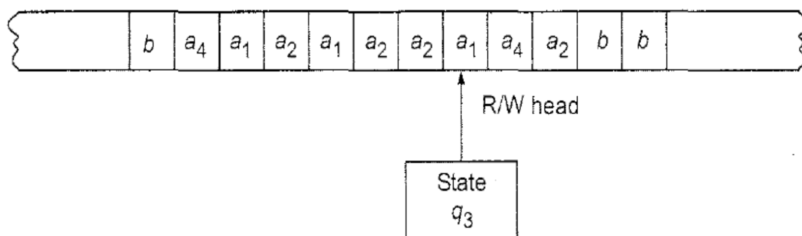
(ii) Transition table, and

(iii) Transition diagram (Transition graph).

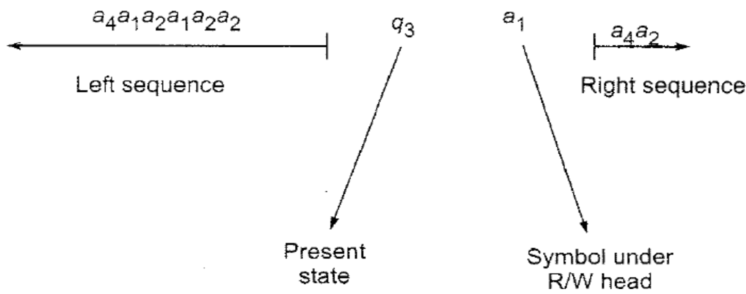
REPRESENTATIONBY INSTANTANEOUSDESCRIPTIONS

Definition: An ID of a Turing machine M is a string $\alpha\beta\gamma$, where β is the present state of M, the entire input string is split as $\alpha\gamma$, the first symbol of γ is the current symbol $\boldsymbol{a}$ under the R/W head and γ has all the subsequent symbols of the input string, and the string α is the substring of the input string formed by all the symbols to the left of $\boldsymbol{a}$.

EXAMPLE: A snapshot of Turing machine is shown in below Fig. Obtain the instantaneous description.



The present symbol under the R/W head is a_1. The present state is q_3. So a_1 is written to the right of q_3 The nonblank symbols to the left of al form the string $a_4a_1a_2a_1a_2a_2$, which is written to the left of q_3. The sequence of nonblank symbols to the right of a_1 is a_4a_2. Thus the ID is as given in below Fig.



Notes: (1) For constructing the ID, we simply insert the current state in the input string to the left of the symbol under the R/W head.

(2) We observe that the blank symbol may occur as part of the left or right substring.

REPRESENTATION BY TRANSITION TABLE

We give the definition of **δ** in the form of a table called the transition table If (q, a)=(γ,α,β). We write $\alpha\beta\gamma$ under the α-column and in the q-row. So if we get $\alpha\beta\gamma$ in the table, it means that α is written in the current cell, β gives the movement of the head (L or R) and γ denotes the new state into which the Turing machine enters.

EXAMPLE:

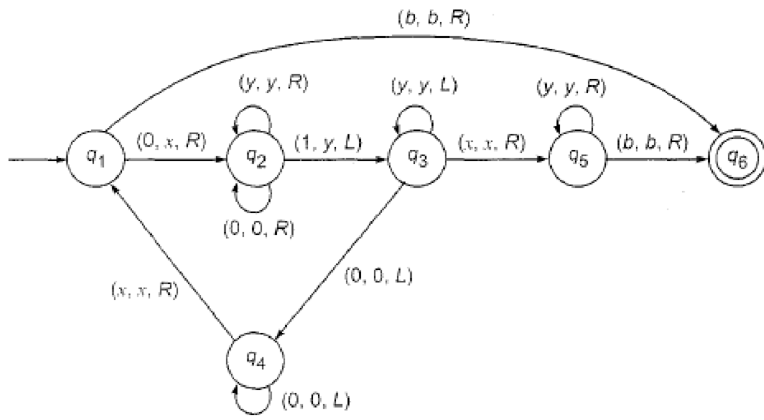
Consider, for example, a Turing machine with five states q_1,...,q_5 where q_1 is the initial state and q_5 is the (only) final state. The tape symbols are 0,1and b. The transition table given below describes **δ**:

Present state	*Tape symbol*		
	b	0	1
$\rightarrow q_1$	$1Lq_2$	$0Rq_1$	
q_2	bRq_3	$0Lq_2$	$1Lq_2$
q_3		bRq_4	bRq_5
q_4	$0Rq_5$	$0Rq_4$	$1Rq_4$
(q_5)	$0Lq_2$		

REPRESENTATION BY TRANSITION DIAGRAM (TD)

The states are represented by vertices. Directed edges are used to represent transition of states. The labels are triples of the form (α,β,γ)where $\alpha,\beta\in\Gamma$ and $\gamma\in\{L,R\}$.When there is a directed edge from q_i to q_j with label (α,β,γ),it means that **δ**(q_i,α)=(q_j,β,γ).

EXAMPLE:



LANGUAGE ACCEPTABILITY BY TURING MACHINES

Let us consider the Turing machine M=(Q,Σ,**Γ**,**δ**,q0,b,F). A string w in Σ* is said to be accepted by M, if $q_0w \overset{*}{\vdash} \alpha_1 p \alpha_2$ for some P∈F and $\alpha_1,\alpha_2 \in \Gamma^*$.

EXAMPLE: Consider the Turing machine M described by the table below

Present state	Tape symbol				
	0	1	x	y	b
→ q_1	xRq_2				bRq_5
q_2	$0Rq_2$	yLq_3		yRq_2	
q_3	$0Lq_4$		xRq_5	yLq_3	
q_4	$0Lq_4$		xRq_1		
q_5				$yxRq_5$	bRq_6
(q_6)					

IDs for the strings (a) 011 (b)0011 (c)001

(a) $q_1011 \vdash xq_211 \vdash q_3xy1 \vdash xq_5y1 \vdash xyq_51$

As (q_5,1) is not defined, M halts; so the input string 011 is not accepted

(b) $q_10011 \vdash xq_2011 \vdash x0q_211 \vdash xq_30y1 \vdash q_4x0y1 \vdash xq_10y1$
$\vdash xxq_2y1 \vdash xxyq_21 \vdash xxq_3yy \vdash xq_3xyy \vdash xxq_5yy$
$\vdash xxyq_5y \vdash xxyyq_5b \vdash xxyybq_6$

M halts. As q_6 is an accepting state, the input string 0011is accepted byM.

(c) $q_1 001 \vdash xq_2 01 \vdash x0q_2 1 \vdash xq_3 0y \vdash q_4 x0y \vdash xq_1 0y$
$\vdash xxq_2 y \vdash xxyq_2$

M halts. As q_2 is not an accepting state,001 is not accepted by M.

DESIGN OF TURING MACHINES

Basic guidelines for designing a Turing machine:

1. The fundamental objective in scanning a symbol by the R/W head is to know what to do in the future. The machine must remember the past symbols scanned. The Turing machine can remember this by going to the next unique state.

2. The number of states must be minimized. This can be achieved by changing the states only when there is a change in the written symbol or when there is a change in the movement of the R/W head.

EXAMPLE 1

Design aTuring machine to recognize all strings consisting of an even number of 1's.

Solution:

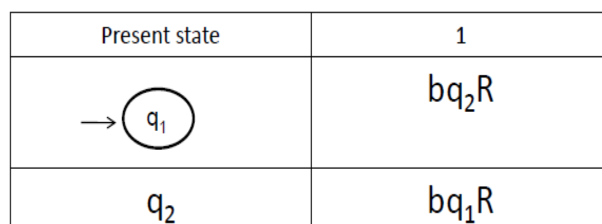
The construction is made by defining moves in the following manner:

(a) q_1 is the initial state. M enters the state q_2 on scanning 1 and writes b.

(b) If M is in state q_2 and scans 1, it enters q_1 and writes b.

(c) q_1 is the only accepting state.

Symbolically M= $(\{q_1,q_2\},\{1,b\},\{1,b\},\boldsymbol{\delta},q,b,\{q_1\})$, Where $\boldsymbol{\delta}$ is defined by ,



Present state	1
$\rightarrow (q_1)$	bq_2R
q_2	bq_1R

Let us obtain the computation sequence of 11:

$q_1 11 \vdash bq_2 1 \vdash bbq_1$

As q_1 is an accepting state 11 is accepted.

Let us obtain the computation sequence of 111:

$q_1 111 \vdash bq_2 11 \vdash bbq_1 1 \vdash bbbq_2$

As q_2 is an not accepting state 111 is not accepted.

EXAMPLE 2: Design a TM that accepts $\{0^n1^n | n \geq 0\}$

Solution: We require the following moves:

(a) If the leftmost symbol in the given input string *w* is 0, replace it by *x* and move right till we encounter a leftmost 1in *w*. Change it to *y and* move backwards.

(b) Repeat (a) with the leftmost 0. If we move back and forth and no 0 or 1 remains. Move to a final state.

(c) For strings not in the form 0^n1^n, the resulting state has to be non-final.

we construct a **TM *M as follows:*** $M = (Q, \Sigma, \Gamma, \delta, q_0, b, F)$

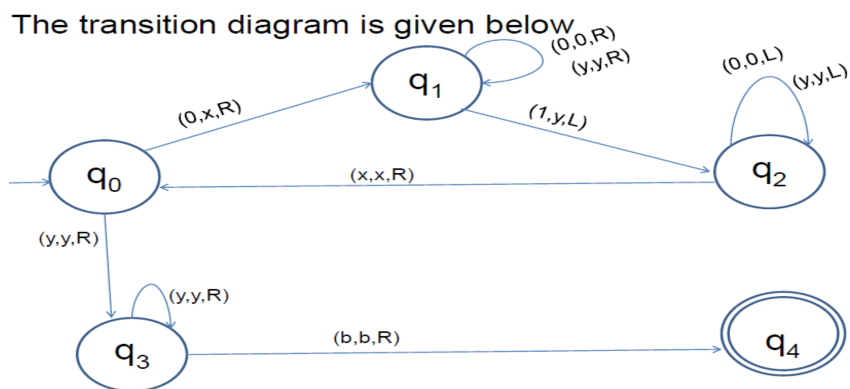
$Q = \{q_0, q_1, q_2, q_3, q_f\}$

$F = \{q_f\}$

$\Sigma = \{0,1\}$

$\Gamma = \{0,1,x,y,b\}$

The transition diagram is given below



Computation sequence of 0011:

$q_0 0011 \vdash xq_1 011 \vdash x0q_1 11 \vdash xq_2 0y1 \vdash q_2 x0y1 \vdash xq_0 0y1$
$\vdash xxq_1 y1 \vdash xxyq_1 1 \vdash xxq_2 yy \vdash xq_2 xyy \vdash xxq_0 yy \vdash xxyq_3 y$
$\vdash xxyyq_3 = xxyyq_3 b \vdash xxyybq_4 b$

q_4 is final state, hence 0011 is accepted by M.

TECHNIQUES FOR TM CONSTRUCTION

1. TURING MACHINE WITH STATIONARY HEAD

Suppose, we want to include the option that the head can continue to be in the same cell for some input symbol. Then we define (q,a) as (q',y,S).This means that the TM, on reading the input symbol a, changes the state to q' and writes y in the current cell in place of a and continues to remain in the same cell. In this model (q, a) =(q', y, D) where D = L, R or S.

2. STORAGE IN THE STATE

We can use a state to store a symbol as well. So the state becomes a pair*(q,a)* where q is the state and a is the tape symbol stored in (q, a). So the new set of states becomes Qx**Γ**.

EXAMPLE: Construct a TM that accepts the language 0 1* + 1 0*.

We have to construct a TM that remembers the first symbol and checks that it does not appear afterwards in the input string.

So we require two states, q_0, q_1. The tape symbols are 0,1 and b. So the TM, having the **'storage facility in state'**, is M=({q_0,q_1}X{0,1,b},{0,1},{0,1,b},**δ**,[q_0,b],[q_1,b]})

We describe **δ** by its implementation description.

1. In the initial state, M is in q_0 and has b in its data portion. On seeing the first symbol of the input sting w, M moves right, enters the state q_1 and the first symbol, say a, it has seen.

2. M is now in [q_1,a].

   (i) If its next symbol is b, M enters [q_1,b], an accepting state.

   (ii) If the next symbol is a, M halts without reaching the final state (i.e.**δ** is not defined).

   (iii) If the next symbol is ā, (ā=0 if a=1 and ā=1 if a=0), M moves right without changing state.

3. Step2 is repeated until M reaches [q_1,b] or halts (**δ** is not defined for an input symbol in w).

3. MULTIPLE TRACK TURING MACHINE

In a multiple track TM, a single tape is assumed to be divided into several tracks. Now the tape alphabet is required to consist of k-tuples of tape symbols, k being the number of tracks. In the case of the standard Turing machine, tape symbols are elements of r; in the case of TM with multiple tracks, it is Γ^k.

4. SUBROUTINES

First a TM program for the subroutine is written. This will have an initial state and a 'return' state. After reaching the return state, there is a temporary halt for using a subroutine, new states are introduced. When there is a need for calling the subroutine, moves are effected to enter the initial state for the subroutine. When the return state of the subroutine is reached, return to the main program of TM.

EXAMPLE: Design a TM which can multiply two positive integers.

Solution: The input (m,n), m,n being given ,the positive integers represented by 0^m10^n. M starts with 0^m10^n in its tape. At the end of the computation, 0^{mn} (mn in unary representation) surrounded by b's is obtained as the output.

The major steps in the construction are as follows:

1. 0^m10^n1 is placed on the tape (the output will be written after the rightmost 1).
2. The leftmost 0 is erased.
3. A block of n 0's is copied onto the right end.
4. Steps 2 and 3 are repeated m times and 10^m10^{mn} is obtained on the tape.
5. The prefix $10^m1$of 10^m10^{mn} is erased, leaving the product 0^{mn} as the output.

For every 0 in 0^m, 0^n is added onto the right end. This requires repetition of step3. We define a subroutine called COPY for step3. For the subroutine COPY the initial state is q_1 *and the final state is* q_5 is given by the transition table as below:

The transition table for the SUBROUTINE COPY

State	*Tape symbol*			
	0	1	2	*b*
q_1	$q_2 2R$	$q_4 1L$	—	—
q_2	$q_2 0R$	$q_2 1R$	—	$q_3 0L$
q_3	$q_3 0L$	$q_3 1L$	$q_1 2R$	—
q_4	—	$q_5 1R$	$q_4 0L$	—
q_5	—	—	—	—

The Turing machine M has the initial state q_0. The initial ID for M is $q_0 0^m 10^n$. O^n seeing 0,the following moves take place

$$q_0 0^m 10^n 1 \vdash bq_6 0^{m-1} 10^n 1 \overset{*}{\vdash} b0^{m-1} q_6 10^n 1 \vdash b0^{m-1} 1 q_1 0^n 1$$

q_1 is the initial state of COPY. The following moves take place for M_1:

$$q_1 0^n 1 \vdash 2q_2 0^{n-1} 1 \overset{*}{\vdash} 20^{n-1} 1 q_3 b \vdash 20^{n-1} q_3 10 \overset{*}{\vdash} 2q_1 0^{n-1} 10$$

After exhausting 0s, q_1 encounters 1. M_1 moves to state q_4. All 2's are converted back to 0's and M_1 halts in q_5. The TM M picks up the computation by starting from q_5 The q_0 and q_6 are the states of M. Additional states are created to check whether reach 0 in 0^m gives rise to 0^m at the end of the rightmost 1 in the input string. Once this is over, M erases 10^n1 and finds 0^{mn} in the input tape.

M can be defined by M=({q_0,q_1,....q_{12}}{0,1},{0,,2,b},**δ**,q_0,b,{q_{12}}) where **δ** is defined by table given below:

	0	1	2	b
q_0	q_6bR	—	—	—
q_6	q_60R	q_11R	—	—
q_5	q_70L	—	—	—
q_7	—	q_81L	—	—
q_8	q_90L	—	—	$q_{10}bR$
q_9	q_90L	—	—	q_0bR
q_{10}	—	$q_{11}bR$	—	—
q_{11}	$q_{11}bR$	$q_{12}bR$	—	—

ADDITIONAL PROBLEMS

1. Design a Turing machine to obtain complement of a binary number.

IDEA OF CONSTRUCTION:

1) If symbol is 0 change it to 1, move read write head to RIGHT
2) If symbol is 1 change it to 0, move read write head to RIGHT
3) Symbol is b (blank) don't change, move read write head to RIGHT, and HALT.

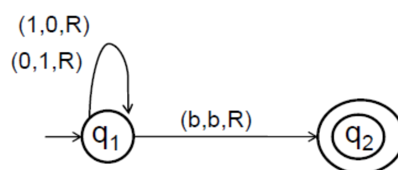
The construction is made by defining moves in the following manner:

(a) q_1 is the initial state. On scanning 1, no change in state and write 0 and move head to RIGHT.

(c) If M is in state q_1 and scans blank, it enters q_2 and writes b move to right.

(d) q_2 is the only accepting state.

Symbolically, $M=(\{q_1,q_2\},\{1,0,b\},\{1,0,b\},\boldsymbol{\delta},q_1,b,\{q_2\})$ Where $\boldsymbol{\delta}$ is defined by:



The computation sequence of 1010:

$q_1 1010 \vdash 0q_1 010 \vdash 01q_1 10 \vdash 010q_1 0 \vdash 0101q_1 b$

$\vdash 0101bq_2 b$

2. Design a TM that converts binary number into its 2's complement representation.

IDEA OF CONSTRUCTION:

- Read input from left to right until right end blank is scanned.
- Begin scan from right to left keep symbols as it is until 1 found on input file.

- If 1 found on input file, move head to left one cell without changing input.
- Now until left end blank is scanned, change all 1's to 0 and 0's to 1.

We require the following moves:

(a) Let q_1 be initial state, until blank is scanned, move head to RIGHT without changing anything. On scanning blank, move head to RIGHT change state to q_2 without changing the content of input.

(b) If q_2 is the state, until 1 is scanned, move head to LEFT without changing anything. On reading 1, change state to q_3, move head to LEFT without changing input.

(c) If q_3 is the state, until blank is scanned, move head to LEFT, if symbol is 0 change to 1, otherwise if symbol is 1 change to 0.On finding blank change state to q_4, move head to LEFT without Changing input.

(d) q_4 is the only accepting state.

We construct a **TM** ***M as follows:***

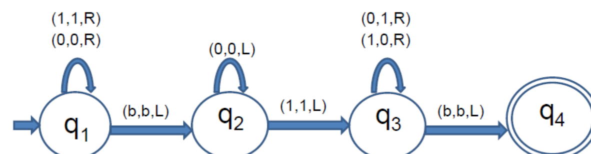
M = (Q, Σ,, **δ**, q_0,b, F)

Q = {q_1,q_2,q_3,q_4}

F = {q_4 }

Σ = {0,1}

Γ= {0,1,b}



3.Design a TM that add two integers

IDEA OF CONSTRUCTION:

- Read input from LEFT to RIGHT until blank (separator of two numbers) is found.
- Continue LEFT to RIGHT until blank (end of second number) is found.
- Change separator b to 1 move head to RIGHT.
- move header to Left (to point rightmost 1)
- Change 1 to b and move right, Halt.

We require the following moves:

(a) In q_1 TM skips1's until it reads b (separator),changes to1and goes to q_1

(b) In q_2 TM skips1's until it reads b (end of input), turns left and goes to q_3

(c) In q_3, TM reads 1 and changes to b go to q_4.

(d) q_4 is the final state, TM halts.

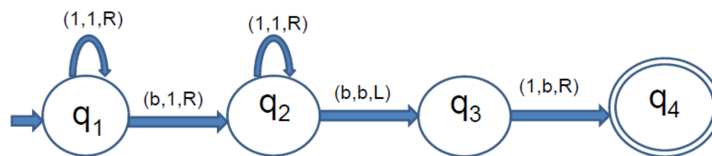
we construct a **TM M as follows:** M = (Q, Σ, **Γ**, **δ**, q_0,b, F)

Q = {q_1,q_2,q_3,q_4}

F = {q_4}

Σ = { b,1}

Γ= {1,b}



4. Design a TM that accepts the set of all palindromes over {0,1}*

IDEA OF CONSTRUCTION:

- If it is 0 and changes to X, similarly if it is 1, it is changed to Y, and moves right until it finds blank.
- Starting at the left end it checks the first symbol of the input,
- Nowmovesonestepleftandcheckwhetherthesymbolreadmatchesthemostrecentlychanged.Ifsoiti salsochangedcorrespondingly.
- Now machine moves back left until it finds 0 or 1.
- This process is continued by moving left and right alternately until all 0's and 1's have been matched.

We require the following moves:

1.If state is q0and it scans 0.

- Then go to state q1 and change the 0 to an X,
- move RIGHT over all 0's and 1's, until it finds either X or Y or B
- Now move one step left and change state to q_3
- It verifies that the symbol read is 0, and changes the 0 to X and goes to state q_5.

2. If state is q0 and it scans 1

- Then go to state q_2 and change the 1 to an Y,
- Move RIGHT over all 0's and 1's, until it finds either X or Y or B
  - Now move one step left and change state to q_4
  - It verifies that the symbol read is 1, and changes the 1 to Y and goes to state q_5.

3. If state is q_5

- Move LEFT over all 0's and 1's, until it finds either X or Y.
- Now move one step RIGHT and change state to q_0.

- Now at q_0 there are two cases:

  1. If 0's and 1's are found on input , it repeats the matching cycle just described.
  2. If X's and Y's are found on input, then it changes all the 0's to X and all the 1's to Y's. **The input was a palindrome of even length,** Thus, state changed to q_6.

4.If state is q_3 or q_4

If X's and Y's are found on input, it concludes that: **The input was a palindrome of odd length,** thus, state changed to q_6.

We construct a **TM *M as follows:***

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

$Q = \{q_0,q_1,q_2,q_3,q_4,q_5,q_6\}$

$F = \{q_6\}$

$\Sigma = \{ b,1,0\}$

$\Gamma = \{X,Y,b\}$

state	0	1	X	Y	B
q_0	(q_1,X,R)	(q_2,Y,R)	(q_6,X,R)	(q_6,Y,R)	(q_6,B,R)
q_1	$(q_1,0,R)$	$(q_1,1,R)$	(q_3,X,L)	(q_3,Y,L)	(q_3,B,L)
q_2	$(q_2,0,R)$	$(q_2,1,R)$	(q_4,X,L)	(q_4,Y,L)	(q_4,B,L)
q_3	(q_5,X,L)	-	(q_6,X,R)	(q_6,Y,R)	-
q_4	-	(q_5,Y,L)	(q_6,X,R)	(q_6,Y,R)	-
q_5	$(q_5,0,L)$	$(q_5,1,L)$	(q_0,X,R)	(q_0,Y,R)	-

PRACTICE PROBLEMS

1. Design a Turing machine to replace all a's with X and all b's with Y.
2. Design a Turing machine to accept $a^n b^m$ n>m.
3. Design a Turing machine to accept $a^n b^n$ n<m.
4. Design a Turing machine to accept (0+1)*00(0+1)* .
5. Design a Turing machine to increment a given input.
6. Design a Turing machine to decrement a given input.
7. Design a Turing machine to subtract two unary numbers.
8. Design a Turing machine to multiply two unary numbers.
9. Design a Turing machine to accept a string 0's followed by a 1.

10. Design a Turing machine to verify if the given binary number is an even number or not.
11. Design a Turing machine to shift the given input by one cell to left.
12. Design a Turing machine to shift the given input to the right by one cell .
13. Design a Turing machine to rotate a given input by one cell.
14. Design a Turing machine to erase the tape.
15. Design a Turing machine to accept $a^n b^n c^n$.
16. Design a Turing machine to accept any string of a's & b's with equal number of a's & b's.
17. Design a Turing machine to accept $a^n b^{2n}$.
18. Design a Turing machine to accept $a^n b^k c^m$: where n=m+k.
19. Design a Turing machine to accept $a^n b^k c^m$: where m=n+k.
20. Design a Turing machine to accept $a^n b^k c^m$: where k=m+n.