FA TO RE

COMP 4200 – Formal Language

ARDEN'S THEOREM

ARDEN'S THEOREM

- If P and Q are RE over Σ, and if P does not contain ε then the following equation in R is given by $R = Q + RP$ has a unique solution $R = QP^*$.
- Proof:

R = Q + RP

Replace R with QP^*

R = Q + (QP^*)P = Q(ε + P^*P) = QP^*

Proved

- $R = Q + RP$

  $= Q + (Q + RP)P$

  $= Q + PQ + (Q+RP)P^2$

  $= Q + PQ + P^2Q + RP^3$

  $= Q + PQ + P^2Q + P^3Q + \ldots\ldots + P^{n+1}R$

  $= Q + PQ + P^2Q + P^3Q + \ldots\ldots + QP^n + QP^*P^{n+1}$

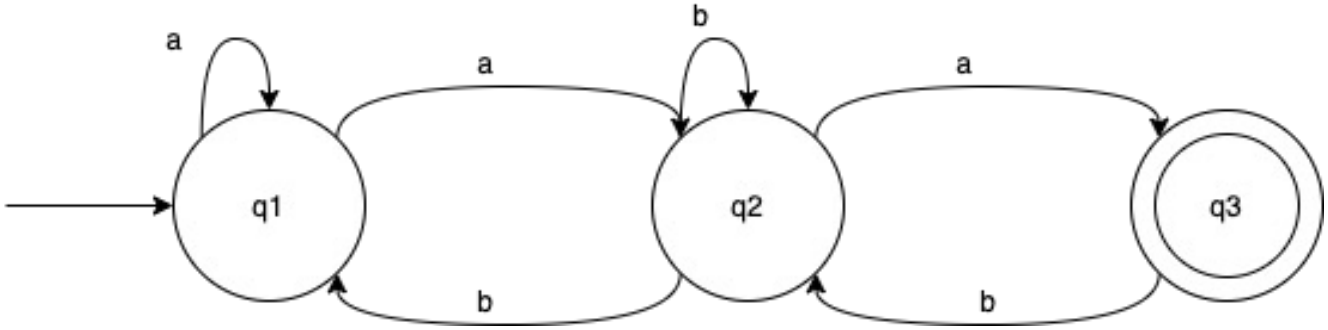
  $= Q[1+P+P^2+P^3+\ldots\ldots+P^n+P^*P^{n+1}]$

  $= QP^*$

DESIGNING RE FROM LANGUAGE

- Design RE for the following language over {a,b}
  - Language accepting strings of length exactly 2
    - L = {aa,ab,ba,bb}
    - R = aa+ab+ba+bb = a(a+b) + b(a+b) = (a+b)(a+b)
  - Language accepting strings of length atleast 2
    - L = {aa,ab,ba,bb,aaa,aab,….}
    - R = (a+b)(a+b)(a+b)*
  - Language accepting strings of length atmost 2
    - L = {ε , a, b, aa, ab, ba, bb}
    - R = ε + a + b + aa + ab + ba + bb = (ε + a + b) (ε + a + b)

NFA TO RE



Incoming transitions

$q3 = q2a \rightarrow 1$

$q2 = q2b + q1a + q3b \rightarrow 2$

$q1 = \varepsilon + q1a + q2b \rightarrow 3$

$R = Q + RP$

$R = QP^*$

$q_1 \rightarrow aq_1 + bq_2 + \varepsilon$

$q_2 \rightarrow aq_1 + bq_2 + bq_3$

$q_3 \rightarrow aq_2$

$q_3 = [q_2 b + q_1 a + q_3 b]\, a$

$q_3 = q_2 ba + q_3 aa + q_3 ba$

$q_2 = q_2 b + q_1 a + q_3 b$

$= q_2 b + q_1 a + q_2 ab$

$q_2 = q_2 [b + ab] + q_1 a$

R R P Q

$q_2 = q_1 a [b + ab]^*$

$q_1 = \varepsilon + q_1 a + [q_1 ab [b + ab]^*]$

$q_1 = \varepsilon + q_1 a + q_1 ab (b + ab)^*$

$q_1 = q_1 [a + ab(b + ab)^*] + \varepsilon$

R R P Q

$q_1 = \varepsilon [a + ab(b + ab)^*]^*$

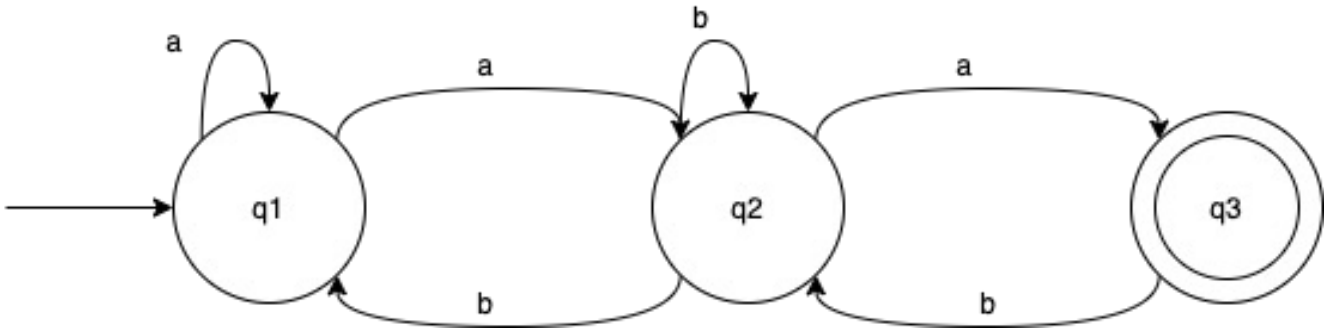
$q_2 = [a + ab)(b + ab)^*]^* a$

$(b + ab)^*$

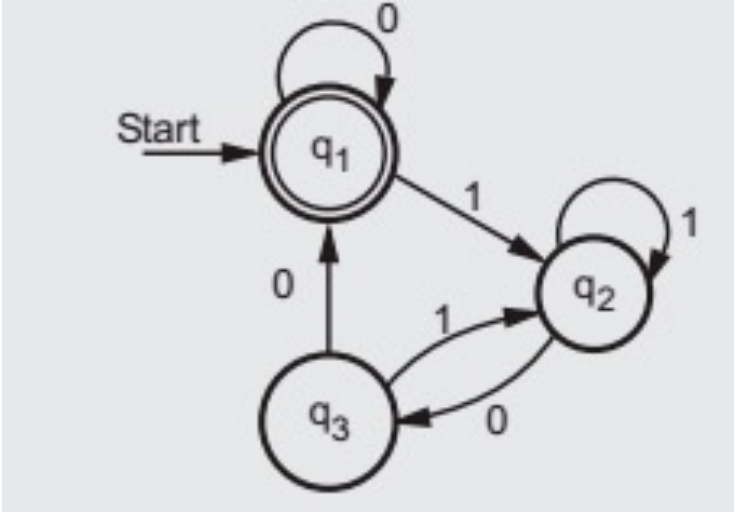
$q_3 = [(a + ab)(b + ab)^*]^* a (b + ab)^* a$

- q3 = q2a
  - = [q1a+q2b+q3b]a
  - = q1aa+q2ba+q3ba → 4
- q2 = q1a + q2b + q3b
  - putting value of q3 from eq(1)
  - = q1a+q2b+(q2a)b
  - = q1a+q2b+q2ab
  - = q1a+q2(b+ab) → R = Q+RP → R = q2, Q = q1a, P = (b+ab) → R = QP*
  - = q1a(b+ab)* → 5
- q1 = ε + q1a + q2b
  - Putting eq(5) in q1
  - = ε + q1a + [q1a(b+ab)*]b
  - = ε + q1[a + [a(b+ab)*]b] → R = Q+RP → R = q1, Q= ε , P= a + [a(b+ab)*]b → R = QP*
  - = ε[a + [a(b+ab)*]b]* → ε.R = R
  - = [a + [a(b+ab)*]b]* → 6

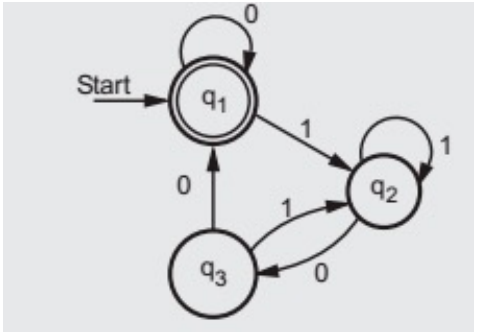
- Final state, q3 = q2a, q2 from eq(5),
  - = [q1a(b+ab)*]a
  - = [a + [a(b+ab)*]b]* a(b+ab)*a → q1 from eq(6)
  - q3 = [a + [a(b+ab)*]b]* a(b+ab)*a
  - Final expression is in terms of input symbols.



EXAMPLE



EXAMPLE



- Incoming transitions
  - q1 = ε + 0q1 + 0q3 → 1
  - q2 = 1q1 + 1q2 + 1q3 → 2
  - q3 = 0q2 → 3
- For getting RE we need to solve the final state q1.
  - q2 = 1q1 + 1q2 + 1q3
    - = 1q1 + 1q2 + 1(0q2) → q3 from eq(3)
    - = 1q1 + q2(1+10) → R = Q + RP → R = q2, Q = 1q1,P = (1+10) → R=QP*
    - = 1q1(1+10)* → 4
  - q1 = ε + 0q1 + 0q3
    - = ε + 0q1 + 0(0q2) → q3 from eq(3)
    - = ε + 0q1 + 00q2
    - q2 from eq(4)
    - = ε + 0q1 + 00[1q1(1+10)*]
    - = ε + 0q1 + 001q1(1+10)*
    - = ε + q1[0 + 001(1+10)*] → R = Q + RP → R = q1, Q = ε,P = [0 + 001(1+10)*] → R=QP*
    - = ε [0 + 001(1+10)*] * → εR = R
    - [0 + 001(1+10)*] *

$q_1 = \varepsilon + aq_1$

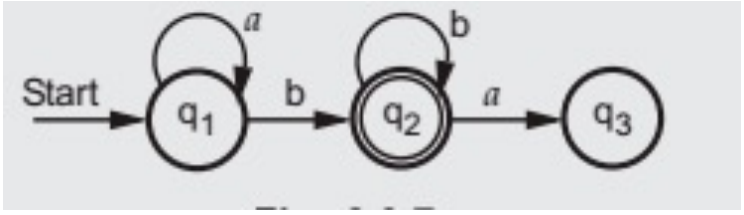
$q_2 = bq_2 + bq_1$

$q_3 = aq_2$

$R = RP + Q$

$R = QP^*$

TRY YOURSELF!

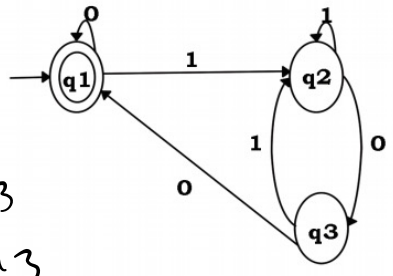


(a)

$q_1 = \varepsilon a^*$

$q_2 = b a^* + bq_2$

$q_2 = ba^*b^*$



$q_1 = \varepsilon + 0q_1 + 0q_3$

$q_2 = 1q_2 + 1q_1 + 1q_3$

$q_3 = 0q_2$

(b)

$q_2 = 1q_1 + 10q_2 + 1q_2$

$q_2 = 1q_1 + q_2[10+1]$

$q_2 = 1q_1(10+1)^*$

$$q_3 = 1q_1 + 1q_3 + 1q_2$$

$$= q_3 = 01q_1 + 01q_3 + 01(1q_1(10+1)^*)$$

$$q_3 = q_1[01 + 011[10+1]^*) + 01q_3$$

$$q_3 = q_1[01 + 011(10+11)^*](01)^*$$

$$q_1 = \varepsilon + 0q_1 + 0(q_1[01 + 011(10+11)^*](01)^*)$$

$$q_1 = \underset{R}{\varepsilon} + \underset{Q}{\varepsilon} + \underset{R}{q_1}(0 + 0[01 + 011(10+11)^*](01)^*)$$

$$q_1 = \varepsilon(0 + 0[01 + 011(10+11)^*](01)^*)^*$$

EQUIVALENCE OF TWO FA

EQUIVALENCE OF TWO FA

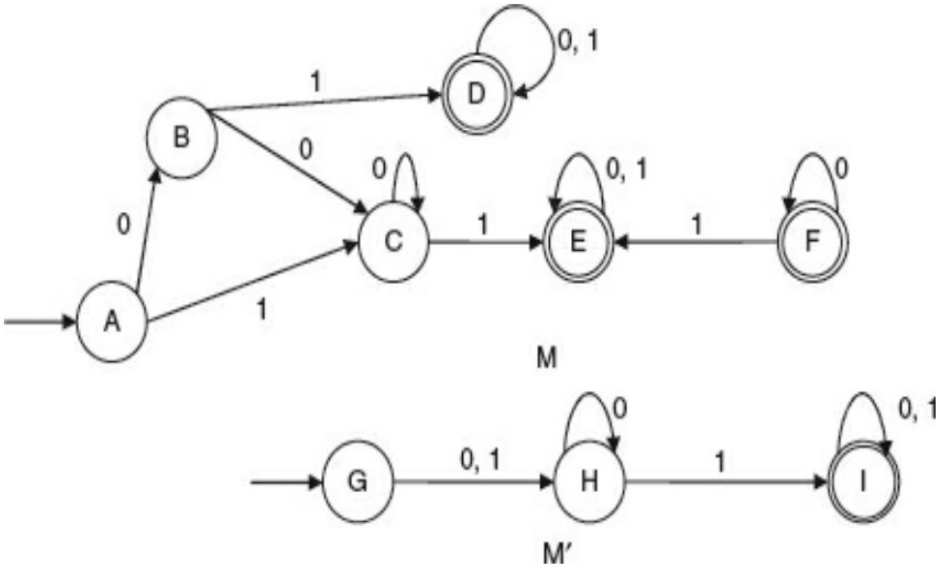
- Two finite automata are equivalent if they accept the same set of strings over L.
- Algorithm:
  - Construct comparison table consisting of n+1 columns, where n is number of input symbols .
  - In the first column, there will be a pair of vertices (q, q′) where q ∈ M and q′ ∈ M′. The first pair of vertices will be the initial states of the two machines M and M′.
  - The second column consists of (q_a, q'_a), where q_a is reachable from the initial state of the machine M for the first input, and q'_a is reachable from the initial state of the machine M′ for the first input.
  - The other n − 2 columns consist of a pair of vertices from M and M′ for n − 1 inputs, where n = 2, 3,..... n − 1.
  - If any new pair of states appear in any of the n − 1 next state columns, which were not taken in the first column, take that pair in the present state column and construct subsequent column elements like the first row.

- If a pair of states (q,q′) appear in any of the n columns for a pair of states in the present state column, where q is the final state of M and q′ is the non-final state of M′ or vice versa, terminate the construction and conclude that M and M′ are not equivalent.
- If no new pair of states appear, which were not taken in the first column, stop the construction and declare that M and M′ are equivalent.
- Simple words:
  - For any pair of states (q_i,q_j), the transition input a ∈ Σ is defined by (q_i,q_j) where,

$$\delta(q_i, a) = q_a \; and \; \delta(q_j, b) = q_b$$

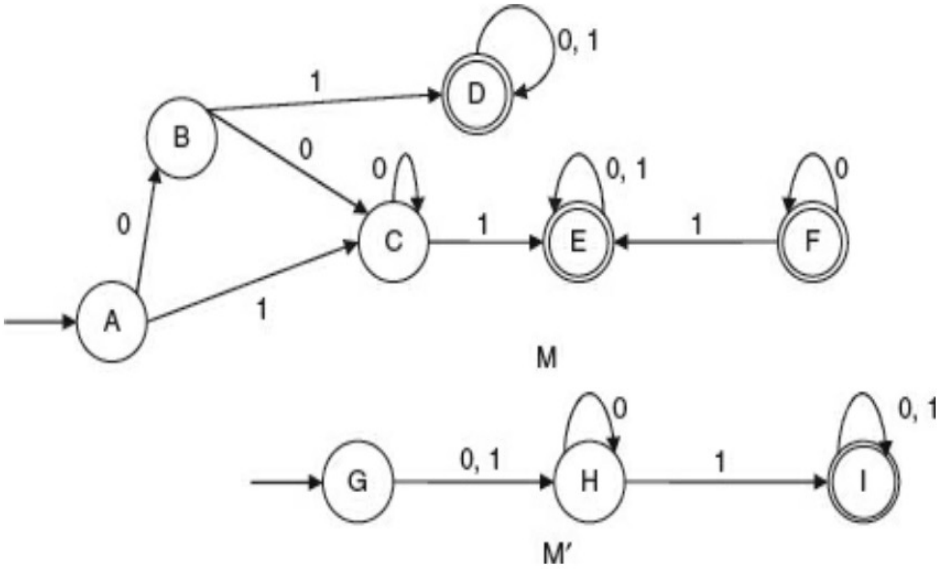
  The two automaton are not equivalent if for a pair (q_a,q_b) one is intermediate state and other is final state.
  - If the initial state is final state of one automaton, then in second automaton also initial state must be final state for them to be equivalent.

EXAMPLE



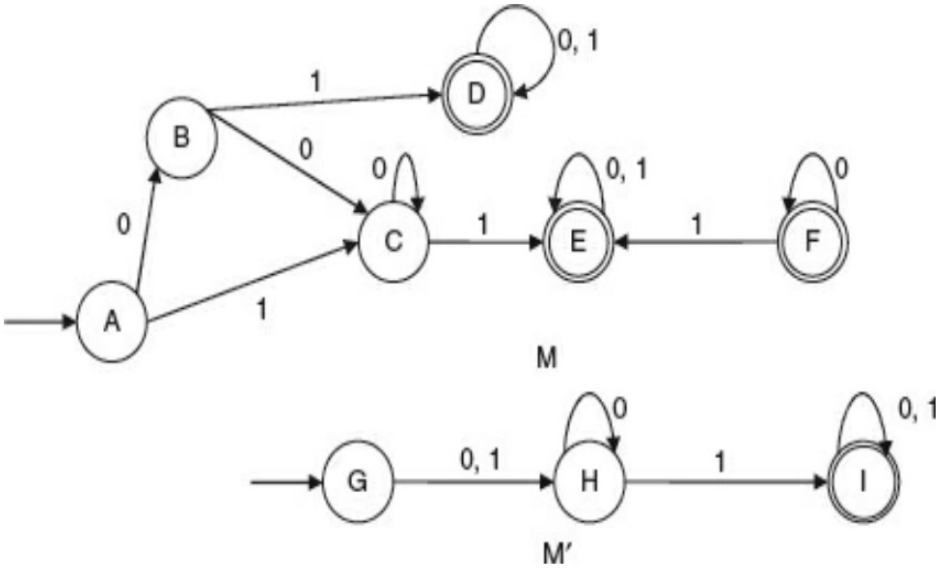
States	0	1
(A,G)	(B,H)	(C,H)
(B,H)	(C,H)	(D,I)
(C,H)	(C,H)	(E,I)
(D,I)	(D,I)	(D,I)
(E,I)	(E,I)	(E,I)

EXAMPLE



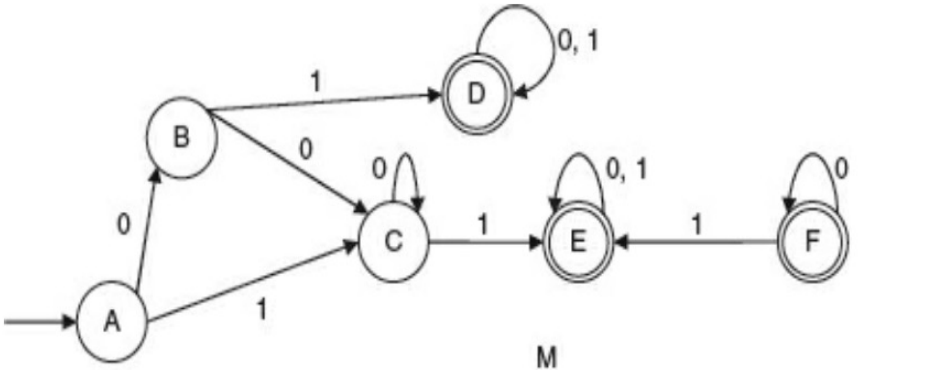
States	0	1
(A,G)	(B,H)	(C,H)

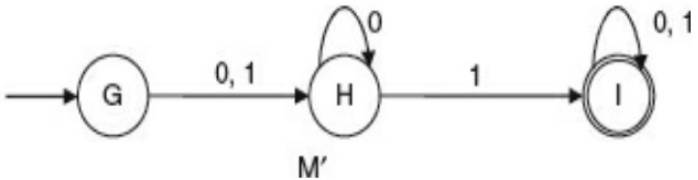
EXAMPLE



States	0	1
(A,G)	(B,H)	(C,H)
(B,H)	(C,H)	(D,I)

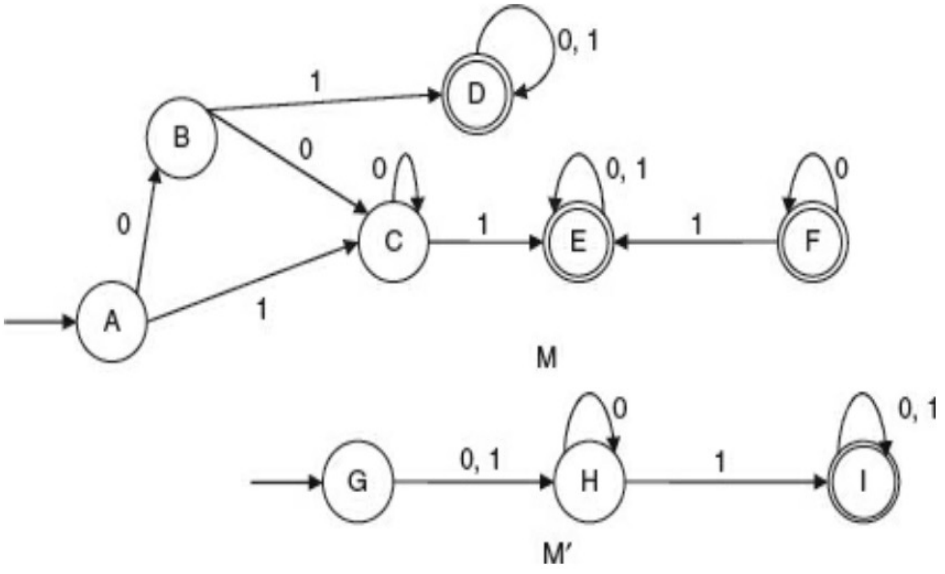
EXAMPLE





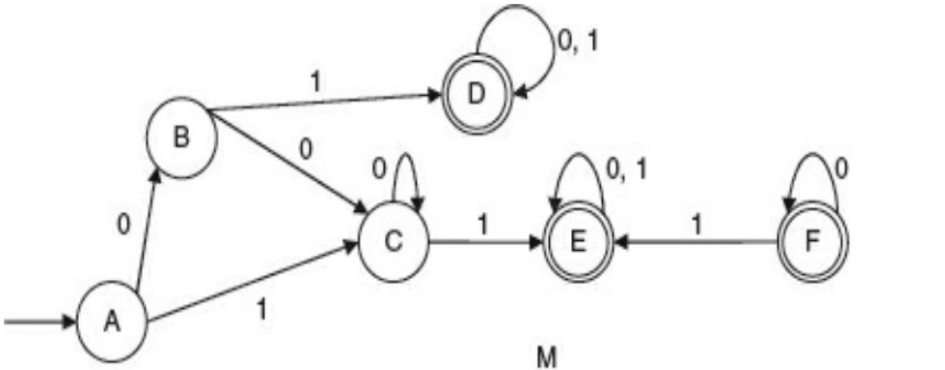
States	0	1
(A,G)	(B,H)	(C,H)
(B,H)	(C,H)	(D,I)
(C,H)	(C,H)	(E,I)

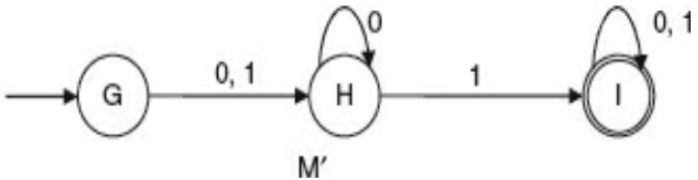
EXAMPLE



States	0	1
(A,G)	(B,H)	(C,H)
(B,H)	(C,H)	(D,I)
(C,H)	(C,H)	(E,I)
(D,I)	(D,I)	(D,I)

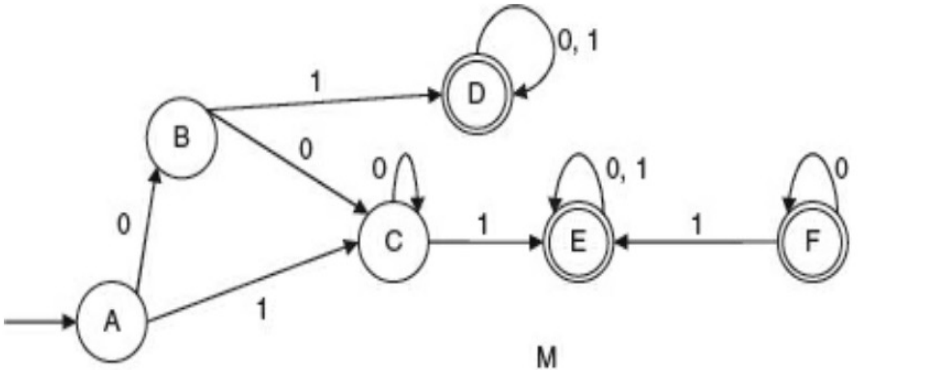
EXAMPLE

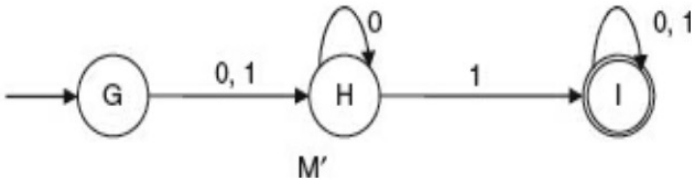




States	0	1
(A,G)	(B,H)	(C,H)
(B,H)	(C,H)	(D,I)
(C,H)	(C,H)	(E,I)
(D,I)	(D,I)	(D,I)
(E,I)	(E,I)	(E,I)

EXAMPLE





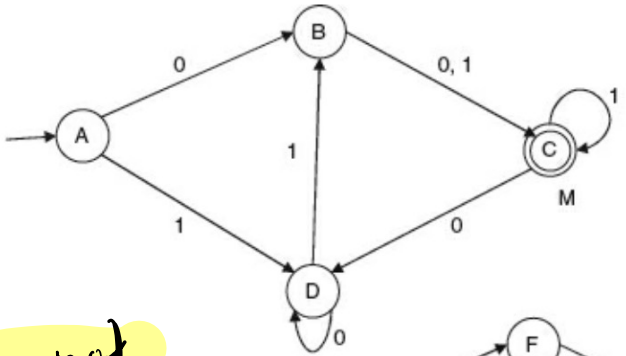
States	0	1
(A,G)	(B,H)	(C,H)
(B,H)	(C,H)	(D,I)
(C,H)	(C,H)	(E,I)
(D,I)	(D,I)	(D,I)
(E,I)	(E,I)	(E,I)

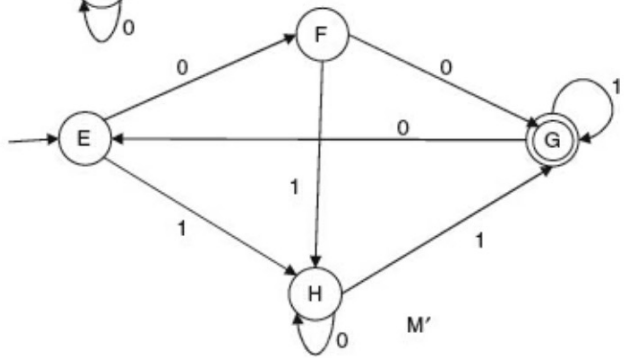
(B,H) → (IS,IS)
(C,H) → (IS,IS)
(D,I) → (FS,FS)
(E,I) → (FS,FS)

Therefore, the two DFAs are equivalent!!

TRY YOURSELF!

	0	1
(A, E)	(B, F)	(D, H)
(B, F)	(C, G)	(C, H)

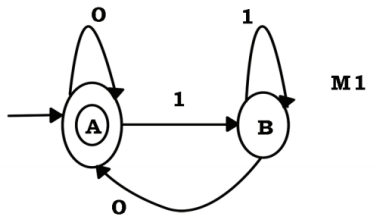


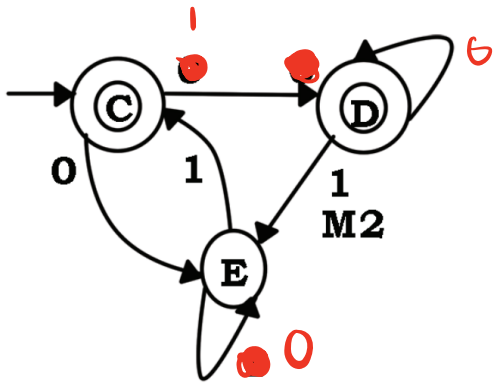


The 2 machines are not equivalent to each other.

	0	1
(A, C)	(A, E)	

TRY YOURSELF!





The 2 machines are not equivalent to each other.

PUMPING LEMMA

NON REGULAR LANGUAGE

REGULAR OR NON REGULAR LANGUAGE

$B = \{0^n 1^n \mid n \geq 0\}$.

- C = {w| w has an equal number of 0s and 1s}, and
- D = {w| w has an equal number of occurrences of 01 and 10 as substrings}.

$B = \{0^n 1^n \mid n \geq 0\}$.

PUMPING LEMMA

- Technique to prove non regularity → pumping lemma
- The Pumping Lemma states that all regular languages have a special pumping property.
- If a language does not have the pumping property, then it is not regular.
  - So one can use the Pumping Lemma to prove that a given language is not regular.
- The property states that all strings in the language can be "pumped" if they are at least as long as a certain special value, called the *pumping length.*

THEOREM: PUMPING LEMMA

- Let A be a regular language. There is a positive integer p such that any $s \in A$ with $|s| > p$, s can be divided into three pieces, $s = xyz$.
- (p is the pumping length of A.)
- This means that every string $s \in L$ contains a substring that can repeated any number of times (via a loop).
- The statement "s can be pumped" means that we can write

  s = xyz , where

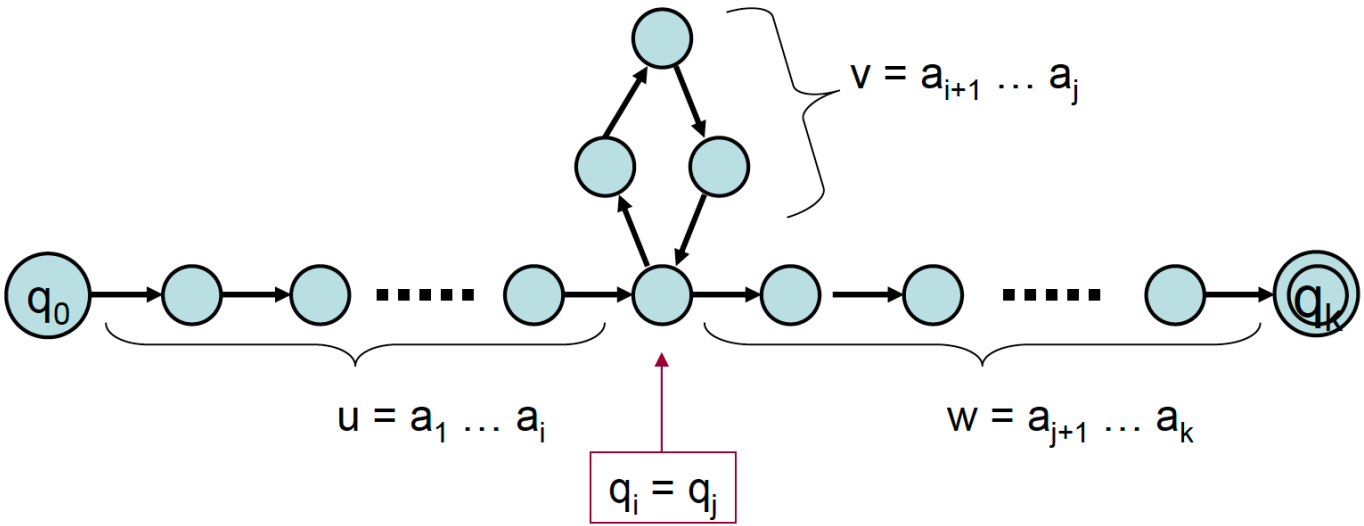
  1. $xy^iz \in A$ for all $i \geq 0$
  2. $|y| > 0$
  3. $|xy| \leq p$

THE PUMPING LEMMA FOR LANGUAGES

- Condition 1: $xy^i z \in A$ for all $i \geq 0$.
- Condition 2: $|y| > 0$.
- Condition 3: $|xy| \leq p$.
- All together, the conditions allow either x or z to be ε, but not both.
- When s is divided into xyz, either x or z may be ε, but condition 2 says that $y \neq \varepsilon$.
- Condition 3 states that the pieces x and y together have length at most p.

THE LOOP

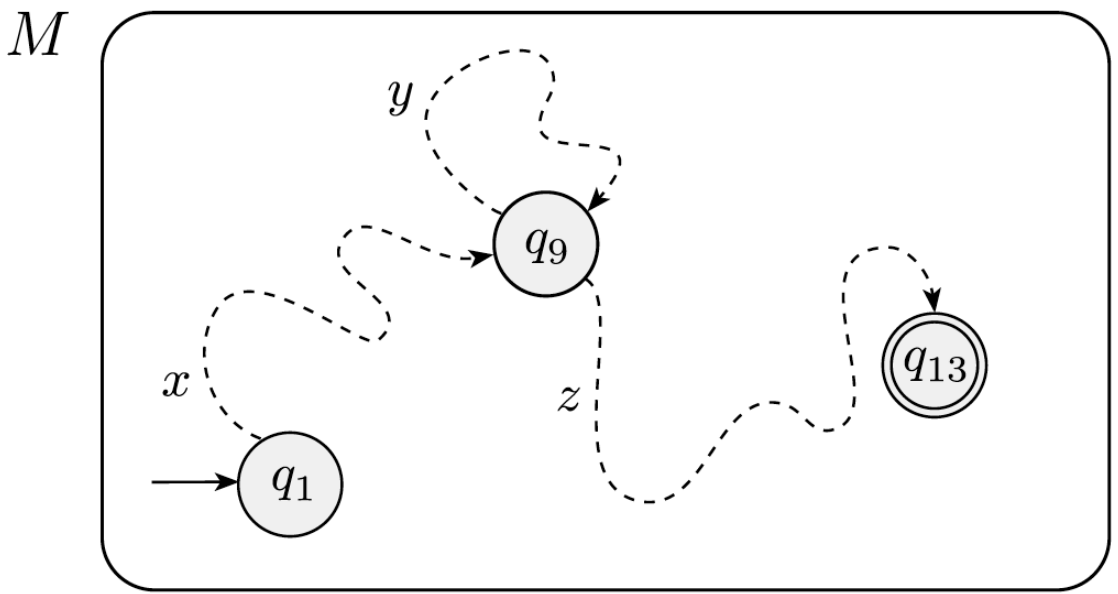
- u v^mw is accepted, since it follows the loop m times (possibly 0 times).



PROOF

- Let M = (Q,Σ, δ, q1, F) be a DFA that recognizes A
- Let p = number of states in the FA
- String s ∈ A, length at least p may be broken into the three pieces xyz, i.e., | s |> p
- If s in A has length at least p, consider the states that FA goes through for s.
- If we let n be the length of s, then sequence of state is n+1.
- Since there are only p states and |s| > p, one state must be repeated (via pigeonhole principle)
- So, there is a loop.

$$s = \begin{array}{cccccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & & & s_n & \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \cdots & \uparrow & \uparrow \\ q_1 & q_3 & q_{20} & \textcircled{q_9} & q_{17} & \textcircled{q_9} & q_6 & & q_{35} & q_{13} \end{array}$$



STEPS(PROOF BY CONTRADICTION)

- Assume A is regular.
- It should have a pumping length as ‘p’.
- All strings longer than ‘p’ must be pumped $|s| \geq p$.
- Now find a string s in A, such that $|s| \geq p$.
- Divide s into xyz.
- Show that $xy^iz \notin A$, for some i.
- Consider all the ways that s can be divided into xyz.
- Show that none of these can satisfy all the three conditions at same time.
- S cannot be pumped.

B BE THE LANGUAGE $\{0^N1^N \mid N \geq 0\}$. USE THE PUMPING LEMMA TO PROVE THAT B IS NOT REGULAR.

- Assume to the contrary that B is regular.
- Choose pumping length, p
- Choose string s from the language, $s = 0^p1^p$, s is a member of B, $|s| \geq p$.
- Split $s = xyz$, where for any $i \geq 0$ the string xy^iz is in B.
- three cases:
  - string y consists only 0's → xyyz has more 0's than 1's. violates the condition 1.
  - String y consists only 1's → xyyz has more 1's than 0's. violates the condition 1.
  - String y consist of both 0's and 1's → may have equal number of 0's and 1's →may be out of order hence not member.
- By contradiction, B is not regular.

C = {w | w has an equal number of 0's and 1's}.

0^p1^p

xy^iz

$|xy| \leq p$

- Assume C is regular.
- Choose pumping length, p
- Choose string s from the language, $s = 0^p1^p$, s is a member of C, $|s| \geq p$.
- Split s = xyz, where for any $i \geq 0$ the string xy^iz is in C.
- If we pick $x = z = \varepsilon$ and $y = 0^p1^p$, can we pump it and have pumped string $xy^i z \in C$?
- Yes! But this choice breaks condition $|xy| \leq p$.
- Suppose we choose x, y , z such that $|xy| \leq p$ and $|y| > 0$. Since $|xy| \leq p$, y consists only of 0's. Hence xyyz C (too ε many zeros).

$$F = \{ ww \mid w \in \{0, 1\}^* \}.$$

|0|

- F = {ε, 00, 11, 0000, 0101, 1010, 1111, . . . }
- Assume F is regular.
- Choose pumping length, p
- Choose string s from the language, s = 0^p10^p1, s is a member of F, |s| ≥ p.
- Split s = xyz, where for any i ≥ 0 the string xy^iz is in F.
- if we let x and z be the empty string, we can pump s. With condition 3 the proof follows because y must consist only of 0s, so xyyz ∉ F, since 0's separated by 1 must be equal.

$$E = \{0^i 1^j \mid i > j\}$$

- Assume E is regular.
- Let p be the pumping length for E given by the pumping lemma.
- Let s = $0^{p+1}1^p$.
- Then s can be split into xyz, satisfying the conditions of Pumping Lemma.
- By condition 3, y consists only of 0s.
- Let's examine the string xyyz to see whether it can be in E. Adding an extra copy of y increases the number of 0s. But, E contains all strings in 0*1* that have more 0s than 1s, so increasing the number of 0s will still give a string in E. No contradiction occurs. We need to try something else.
- Let's consider the string xy^0z = xz, Removing string y decreases the number of 0s in s. Recall that s has just one more 0 than 1. Therefore, xz cannot have more 0s than 1s, so it cannot be a member of E. Thus we obtain a contradiction.

$$D = \{ 1^{n^2} \mid n \geq 0 \}$$

- D contains all strings of 1s whose length is a perfect square
- D = {ε, 1, 1111, 11111111, . . . } We use the pumping lemma to prove that D is not regular. The proof is by contradiction.
- Let p be the pumping length given by the pumping lemma.
- Let s be the string 1^{p^2}
- Assume we have xyz ∈ D as per Pumping Lemma.
- By condition 3 of the pumping lemma, |xy | ≤ p and thus |y | ≤ p.
- We have |xyz| ≤ p^2, then |xy^2z| ≤ p^2 + p. But p^2 + p < p^2 + 2 p + 1 = (p + 1)^2 . Moreover, condition 2 implies that |y | > 0, and so |xy^2z| > p^2.
- So the length of xy^2z lies between two consecutive perfect squares p^2 and (p + 1)^2 , and hence $xy^2z \notin D$

$$A = \{a^n b^n \mid n \geq 0\}$$

- We need to keep the count of number of 'a' to get same number of 'b'
- FA cannot keep count, hence A is not regular.
- Proof:
  - Assume A is regular.
  - Pumping length, p
  - Choose string s, $s = a^p b^p$
  - Divide into 3 parts, xyz
  - Assume p = 7, $s = a^7 b^7$ = aaaaaaabbbbbbb
  - Case 1: The y is in the a part
    - aaaaaaa bbbbbbb
    - x y z
    - $xy^i z \rightarrow xy^2 z$, here i =2
    - aaaaaaaaaaabbbbbbb ≠ A

- Case 2: The y is in the b part
  - aaaaaaa bbbbbbb
    x y z
  - $xy^iz \rightarrow xy^2z$, here i =2
  - aaaaaaabbbbbbbbbbb ≠ A
- Case 3: The y is in the a and b part
  - aaaaaaa bbbbbbb
    x y z
  - $xy^iz \rightarrow xy^2z$, here i =2
  - aaaaaaabbaabbbbbbb ≠ A
- $|xy| \leq p$, here p = 7
  - Case 1: 6 < 7
  - Case 2: 13 ≮ 7
  - Case 3: 9 ≮ 7

CONTEXT FREE GRAMMAR

OVERVIEW

- Grammar
  - Derivation from a grammar
  - Types
- Context-Free Grammars
  - Formal Definition
  - Examples of CFG
  - Designing CFG
  - Parse/derivation tree
  - Ambiguity
  - Simplification of CFG
  - Chomsky Normal Form
  - Greibach Normal form
- Pushdown Automata
  - Formal Definition
  - Examples of PDA
  - Equivalence with CFGs
- Non-Context Free Languages
  - The pumping lemma for context-free languages

GRAMMAR

- Regular grammar → set of rules that we use for proper composition.
- Grammar can be formally described using 4 tuples as G = {V,T,S,P}
  - V → set of variables/non terminal symbols
  - T → Terminal symbols
  - S → Start symbol
  - P → Production rule for terminal and non terminal symbols.
- Production rule, has the form a→β, where a and β are strings on V U T and atleast one symbol of 'a' belongs to V.
- Example: G = ({S,A,B}, {a,b}, S, {S → AB, A→a,B→b})

STEPS

- Write down the start variable. It is the variable on the left-hand side of the top rule, unless specified otherwise.
- Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the right-hand side of that rule.
- Repeat step 2 until no variables remain.

EXAMPLE

$$A \rightarrow 0A1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$

- Grammar G1 generates the string 000#111. The sequence of substitutions to obtain a string is called a derivation.
- A derivation of string 000#111 in grammar G1 is

EXAMPLE

$$A \rightarrow 0A1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$

- Grammar G1 generates the string 000#111. The sequence of substitutions to obtain a string is called a derivation.
- A derivation of string 000#111 in grammar G1 is
- $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111.$

$G = (\{S,A,B\}, \{A,B\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

- V = {S,A,B}
- T = {a,b}
- S = s
- P = S→AB, A→a, B→b
  - S → AB, we know that A→a
  - S → aB
  - S → ab

DERIVATION FROM A GRAMMAR

- The set of all the strings that can be derived from a grammar is said to be the Language generated from that grammar.
- Example: G = ({S,A}, {a,b}, S, {S → aAb, A→aaAbb,A→ε})
  - S → aAb, A → aaAbb
  - Expanding, S → aaaAbbb
  - S → aaaAbbb, A→ ε
  - S → aaabbb, String derived from grammar G
  - L = $a^n b^n$

DERIVATION FROM A GRAMMAR

- The set of all the strings that can be derived from a grammar is said to be the Language generated from that grammar.
- Example: G = ({S,A}, {a,b}, S, {S → aAb, A→aaAbb,A→ε})

TYPES OF GRAMMAR

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable language	Turing Machine
Type 1	Context-sensitive grammar	Context-sensitive language	Linear-bounded automaton
Type 2	Context-free grammar	Context-free language	Pushdown Automaton
Type 3	Regular grammar	Regular language	Finite state automaton

CONTEXT FREE GRAMMAR

- Methods of describing languages:
  - Finite automata
  - Regular expressions
- Some simple languages, such as $\{0^n1^n \mid n \geq 0\}$, cannot be described by these languages.
- We study Context-Free Grammars, a more powerful method of describing languages. Such grammars can describe certain features that have a recursive structure.
- The collection of languages associated with context-free grammars are called the Context-Free languages
- PDA: a class of machines recognizing the context-free languages A Context Free Grammar is a "machine" that creates a language. A language created by a CFG is called A Context Free Language

CONTEXT FREE LANGUAGE

Consider language $\{ 0^n1^n \mid n \geq 0 \}$, which is nonregular.

- Start variable S with "substitution rules": $S \rightarrow 0S1$ $S \rightarrow \varepsilon$
- Rules can yield string 0^k1^k by applying rule "$S \rightarrow 0S1$" k times, followed by rule "$S \rightarrow \varepsilon$" once.
- Derivation of string 0^31^3 $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000\varepsilon111 = 000111$

EXAMPLE

- Example of CFG Example: Language $\{ 0^n 1^n \mid n \geq 0 \}$ has CFG $G = (V, \Sigma, R, S)$
- Variables $V = \{S\}$
- Terminals $\Sigma = \{0, 1\}$
- Start variable S
- Rules R: $S \rightarrow 0S1$ $S \rightarrow \varepsilon$
- Combine rules with same left-hand side in Backus-Naur (or Backus Normal) Form (BNF):

$$S \rightarrow 0S1 \mid \varepsilon$$

The following is an example of CFG, which we call G1.

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

- A grammar consists of a collection of rules. Each rule appears as a line in the grammar, comprising a symbol and a string separated by an arrow.
- The symbol is called a variable.
- The string consists of variables and other symbols called terminals.
- One variable is designated as the start variable.
- It usually occurs on the LHS of the topmost rule.
- For example, grammar G1 contains 3 rules. G1's variables are A and B, where A is the start variable. Its terminals are 0, 1, and #.

$A \rightarrow 0A1$

$A \rightarrow B$

$A \rightarrow 0A1 \mid B$

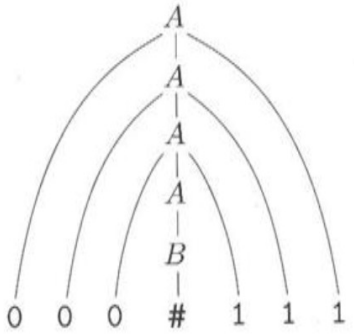
$B \rightarrow \#$

For example, grammar G1 generates the string 000#111. The sequence of substitutions to obtain a string is called a derivation.

A derivation of string 000#111 in grammar G1 is

$A \rightarrow 0\,A1 \rightarrow 00\,A11 \rightarrow 000\,A111 \rightarrow 000\,B111 \rightarrow 000\,\#\,111.$

Language defined by grammar G1 $\{0^n\#1^n \mid n \geq 0\}$



Parse tree for 000#111 in grammar G1

EXAMPLE

- Consider grammar G3 = ({S}, {a, b}, R, S). The set of rules, R, is
- S → aSb | SS | ε
- What are some of the strings generated by this grammar?
- Can you verify if the strings abab, aaabbb, and aababb, belong to the grammar.

EXAMPLE

- CFG G = (V, Σ, R, S) with

1. V = {S}

2. Σ = {0, 1}

3. Rules R: S → 0S | ε

Then L(G) = { 0^n | n ≥ 0 }.

For example, S derives 0^3

S ⇒ 0S ⇒ 00S ⇒ 000S ⇒ 000ε = 000

Note that → and ⇒ are different.

→ used in defining rules

⇒ used in derivation

EXAMPLE

- CFG G = (V, Σ, R, S) with

1. V = {S}
2. Σ = {0, 1}
3. Rules R: S → 0S | 1S | ε

Then L(G)=Σ*.

For example, S derives 0100

S ⇒ 0S ⇒ 01S ⇒ 010S ⇒ 0100S ⇒ 0100

EXAMPLE

Consider grammar G4 = (V, Σ ,R, <EXPR>)

V is {<EXPR> , <TERM> , <FACTOR>} and Σ is {a, +, x, (,)}.

The rules are

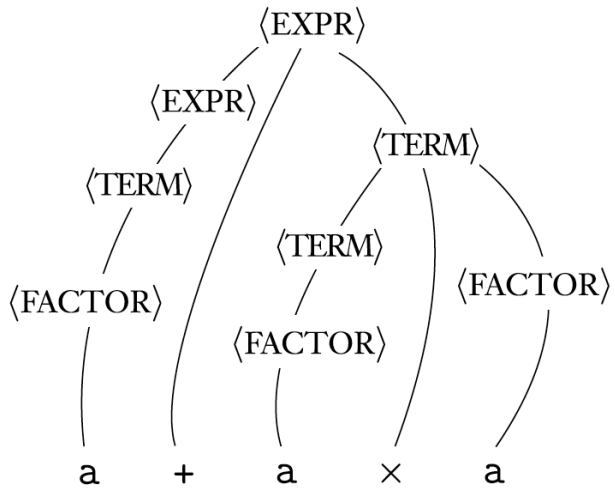
<EXPR> → <EXPR > + <TERM> | <TERM>

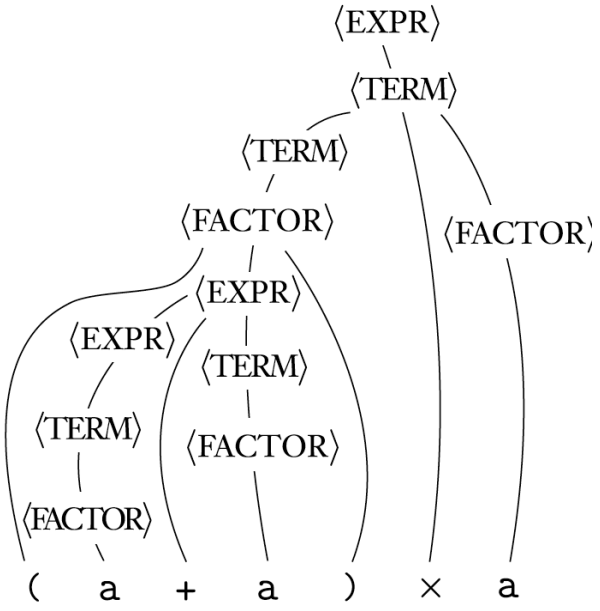
<TERM> → <TERM> * <FACTOR> | <FACTOR>

<FACTOR> → (<EXPR >) | a

The two strings a + a * a and (a + a) * a can be generated with grammar G4.

PARSE TREE


⟨EXPR⟩
⟨EXPR⟩
⟨TERM⟩
⟨FACTOR⟩
a
+
⟨TERM⟩
⟨TERM⟩
⟨FACTOR⟩
a
×
⟨FACTOR⟩
a


⟨EXPR⟩
⟨TERM⟩
⟨TERM⟩
⟨FACTOR⟩
⟨EXPR⟩
⟨EXPR⟩
⟨TERM⟩
⟨TERM⟩
⟨FACTOR⟩
⟨FACTOR⟩
⟨FACTOR⟩
(
a
+
a
)
×
a

Consider grammar G4 = (V, Σ ,R, <EXPR>)

V is {<EXPR> , <TERM> , <FACTOR>} and Σ is {a, +, x, (,)}.

The rules are

<EXPR> → <EXPR > + <TERM> | <TERM>

<TERM> → <TERM> * <FACTOR> | <FACTOR>

<FACTOR> → (<EXPR >) | a

<EXPR> → <EXPR> + <TERM>

→ <TERM> + <TERM>

→ <FACTOR> + <TERM>

→ a + <TERM>

→ a + <FACTOR>

→ a + (<EXPR>)

→ a + (<TERM>)

→ a + (<TERM> * <FACTOR>)

→ a + (<FACTOR> * <FACTOR>)

→ a + (a * <FACTOR>) = a + (a * a)

TRY YOURSELF!

- CFG G = (V, Σ, R, S) with V = {S}, Σ = {0, 1}, Rules R: S → 0S | 1S | 1
- CFG G = (V, Σ, R, S) with V = {S, Z}, Σ = {0, 1}, Rules R: S → 0S1 | Z Z → 0Z | ε
- PALINDROME = { w ∈ Σ* | w = wR }, where Σ = {a, b}. CFG G = (V, Σ, R, S) with V = {S}, Σ = {a, b}, Rules R: S → aSa | bSb | a | b | ε

TRY YOURSELF!

- CFG G = (V, Σ, R, S) with V = {S}, Σ = {0, 1}, Rules R: S → 0S | 1S | 1. Derive 011
- CFG G = (V, Σ, R, S) with V = {S, Z}, Σ = {0, 1}, Rules R: S → 0S1 | Z

Z → 0Z | ε. L(G) = $\{0^i1^j \mid i>=j\}$

- PALINDROME = { w ∈ Σ* | w = wR }, where Σ = {a, b}. CFG G = (V, Σ, R, S) with V = {S}, Σ = {a, b}, Rules R: S → aSa | bSb | a | b | ε
- A→0A, A→1B|1, B→0A|1B|1, Derive the string 10011011
- S→0S1|1S0|SS|ε, derive the string 00011011.

CFG FOR SIMPLE ARITHMETIC EXPRESSIONS

- CFG G = (V, Σ, R, S) with

1. V = {S}
2. Σ = { +, −, ×, /, (,), 0, 1, 2, ..., 9 }
3. Rules R: S → S + S | S − S | S × S | S/S | (S) | −S | 0 | 1 | ⋯ | 9

L(G) is a set of valid arithmetic expressions over single-digit integers.

S derives string 2 × (3 + 4)

CFG FOR SIMPLE ARITHMETIC EXPRESSIONS

- CFG G = (V, Σ, R, S) with

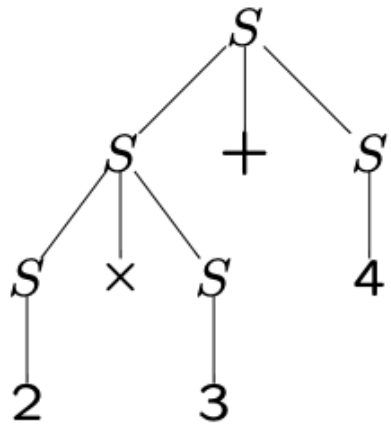
1. V = {S}
2. Σ = { +, −, ×, /, (,), 0, 1, 2, ..., 9 }
3. Rules R: S → S + S | S − S | S × S | S/S | (S) | −S | 0 | 1 | ⋯ | 9

L(G) is a set of valid arithmetic expressions over single-digit integers.

S derives string 2 × (3 + 4)

S ⇒ S × S ⇒ S × (S) ⇒ S × (S + S) ⇒ 2 × (S + S) ⇒ 2 × (3 + S) ⇒ 2 × (3 + 4)

DERIVATION TREE



- CFG

$S \rightarrow S + S \mid S - S \mid S \times S \mid S/S \mid (S) \mid -S \mid 0 \mid 1 \mid \cdots \mid 9$

- Can generate string $2 \times 3+4$ using derivation

$S \Rightarrow S + S \Rightarrow S \times S + S \Rightarrow 2 \times S + S \Rightarrow 2 \times 3 + S \Rightarrow 2 \times 3+4$

- Leftmost derivation: leftmost variable replaced in each step.
- Corresponding derivation (or parse) tree

DESIGNING CFG

For example, to get a grammar for the language $\{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$, first construct the grammar

$S_1 \rightarrow 0\, S_1 1 \mid \varepsilon$

for the language $\{0^n 1^n \mid n \geq 0\}$ and the grammar

$S_2 \rightarrow 1\, S_2 0 \mid \varepsilon$

for the language $\{1^n 0^n \mid n \geq 0\}$ and then add the rule

$S \rightarrow S_1 \mid S_2$ to give the grammar

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow 0\, S_1\, 1 \mid \varepsilon$

$S_2 \rightarrow 1\, S_2\, 0 \mid \varepsilon$

EXAMPLE

- Generate CFG for given language $L=\{a^n b^n / n \geq 0\}$
- Let $L=\{a^n b^{2n} / n \geq 1\}$.Find CFG
- $L=\{ a^n b a^n / n \geq 1\}$. Find CFG.
- $L=\{wcw^r$ /w is element of (a,b)*}. Find the CFG.

CONTEXT FREE LANGUAGE (CFL)

- The CFG is given as S → aSb/ab. Find the CFL.

CONTEXT FREE LANGUAGE (CFL)

- The CFG is given as S → aSb/ab. Find the CFL.
- Let G=({S},{a,b},{S → aSb, S → ab},{S})
- S → ab
- S →aSb
  - S → aabb
- S → aSb
  - S → aaSbb
  - S → aaabbb
- L(G) = {ab,aabb,aaabbb,….} Therefore,L(G)={$a^n b^n / n \geq 1$}

EXAMPLE

- Find the CFL for S → a B/bA, A → a/aS/bAA, B → b/bS/aBB

TRY YOURSELF!

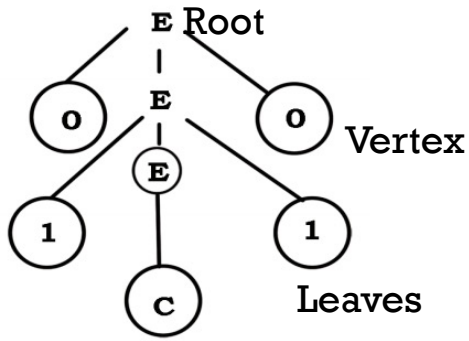
- Find L(G) S → aSa/bSb/ε
- Find L(G) S → aSa/bSa/ε
- Find L(G) S → aSa/bSb/a/b
- Find L(G) S → SS/bS/a
- Find L(G) for S → aS/bS/a
- Find L(G) S → aSa/bSb/ε

TRY YOURSELF!

- Consider the grammar G=({A,S},{a,b},P,S) where p consist of S → aAS/a A → SbA/SS/ba Draw the derivation for string"aabbaa"
- Draw the derivation for the string abb where S → aAB A → bBb B → A/ε

PARSE OR DERIVATION TREE

- Parse tree is an ordered rooted tree that graphically presents the semantic information of string derived from CFG.
- Any one production or derivation derived from tree format.
- It is the graph form of representation.
- RULE
  - Start with 'S'.→ Root
  - The final answer should be terminal. →Leaves
  - The derivation should be applied from left to right.
  - The intermediate derivation should be terminal or nonterminal. → vertex

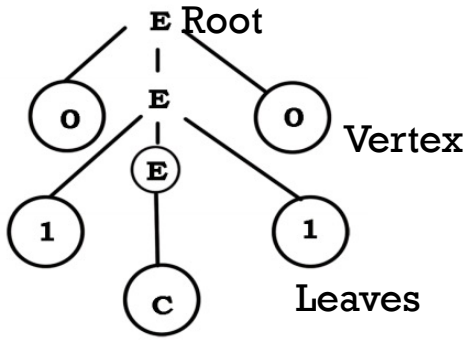


- W = 01C10 , E → 0E0 E →1E1 E → C
- DERIVATION E => 0E0

  => 01E10 ∴ E => 1E1

  => 01C10 ∴ E => C

  => E => 01C10
- PARSE TREE (OR) DERIVATION TREE



TYPES OF DERIVATION

- LEFT MOST DERIVATION (LMD): If at each step-in derivation, a production is applied to the left most variable (or) left most non-terminal then the derivation method is called left most derivation.
- Example w = id+id*id E → E+E , E → E*E, E → id

E → E+E

E → id+E ∴ E=>id

E → id+E*E ∴ E=>E*E

E → id+id*E ∴ E=>id

E → id+id* id

- RIGHT MOST DERIVATION (RMD): A derivation in which the right most variable is replaced at each step then, the derivation method is called right most derivation.
- Example

E → E+E

E → E+E*E ∴ E=>E*E

E → E+E*id ∴ E=>id

E → E+id*id => id+id*id

EXAMPLE

- Draw derivation for the string abbabba For thee CFG given G where production is S → bA/aB A → a/aS/Baa B → b/bS/Abb
  - s → aB
  - s → aBB
  - s → abSB
  - s → abSbs
  - s → abbAbs
  - s → abbAbbA
  - s → abbabbA
  - s → abbabba

EXAMPLE

Draw derivation tree for the string abaaba for the CFG given by G where p is S → aSa/bSb/b/a/ε

Solution:

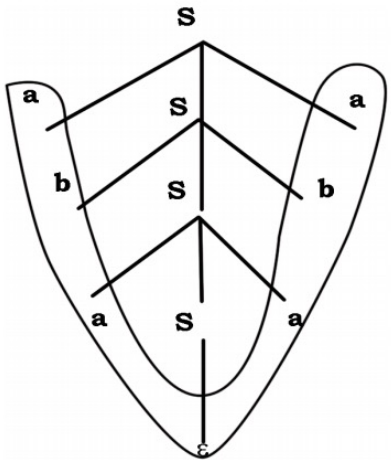
S → aSa

S → abSba

S → abaSaba

S → abaεaba

S → abaaba



TRY YOURSELF!

- Draw derivation tree for the string aabbaabba For thee CFG given G where production is S → bA/aB A → a/aS/bAA B →b/bS/aBB
- Draw the derivation tree for the string abb where S → aAB A → bBb B → A/ε.
- Draw the derivation tree for the given graph CFG G=(v,t,p,s) where p={S → aSb/ ε} and the input string :aabb

EXAMPLE

- Draw derivation for the string abbabba For thee CFG given G where production is S → bA/aB A → a/aS/Baa B → b/bS/Abb/BB
  - s → aB
  - s → aBB
  - s → abSB
  - s → abSbS
  - s → abbAbS
  - s → abbAbbA
  - s → abbabbA
  - s → abbabba

TRY YOURSELF!

- Consider the grammar G=({A,S},{a,b},P,S) where p consist of S → aAS/a A → SbA/SS/ba Draw the derivation for string"aabbaa"
- Draw the derivation for the string abb where S → aAB A → bBb B → A/ε

TYPES OF DERIVATION

- LEFT MOST DERIVATION (LMD): If at each step-in derivation, a production is applied to the left most variable (or) left most non-terminal then the derivation method is called left most derivation.
- Example w = id+id*id E → E+E , E → E*E, E → id

E → E+E

E → id+E ∴ E=>id

E → id+E*E ∴ E=>E*E

E → id+id*E ∴ E=>id

E → id+id* id

- RIGHT MOST DERIVATION (RMD): A derivation in which the right most variable is replaced at each step then, the derivation method is called right most derivation.
- Example

E → E+E

E → E+E*E ∴ E=>E*E

E → E+E*id ∴ E=>id

E → E+id*id => id+id*id

TRY YOURSELF!

- Draw derivation tree for the string aabbaabba For thee CFG given G where production is S → bA/aB A → a/aS/bAA B →b/bS/aBB
- Draw the derivation tree for the string abb where S → aAB A → bBb B → A/ε.
- Draw the derivation tree for the given graph CFG G=(v,t,p,s) where p={S → aSb/ ε} and the input string :aabb

AMBIGUITY

- CFG G is ambiguous if string w ∈ L(G) having different parse trees (or equivalently, different leftmost derivations).
- A string is derived ambiguously in a CFG if it has two or more different leftmost derivations
- A grammar that produces more than one parse tree (or) derivation tree for some string then the grammar is said to be an ambiguous grammar.
- An ambiguous grammar produces more than one LMD (or) more than 1 RMD then, the given grammar is said to be an ambiguous grammar.
- Leftmost derivation : At every step in the derivation the leftmost variable is replaced
- A grammar is ambiguous if it generates *some* string ambiguously
- Some context free languages are inherently ambiguous, that is, every grammar for the language is ambiguous

- LMD:

E => id+E ∴ E => id

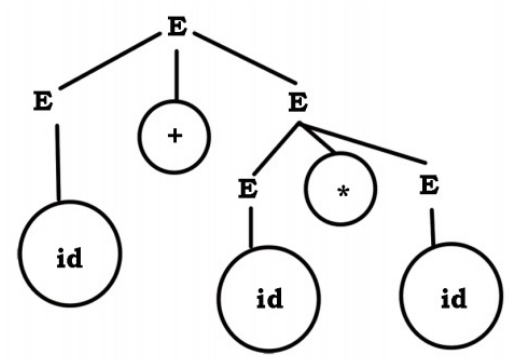
E => id+E*E

E => id+id*E

E => id+id*id

E=> id+id*id

PARSE TREE



Rules:
E → E+E ,E → E*E ,E → (E) ,E → id
w= id+id*id

- RMD:

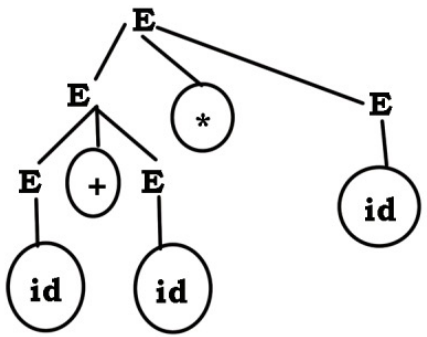
E => E*E

E => E+E*E

E => E+E*id

E => E+id*id

E => id+id*id



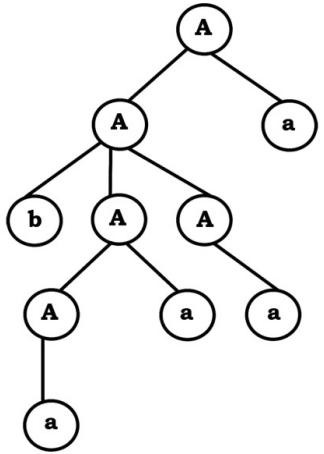
Therefore, the above grammar is ambiguous.

Show that CFG having productions A → a/Aa/bAA/AAb/AbA is ambiguous. The string can be baaaa

Show that CFG having productions A → a/Aa/bAA/AAb/AbA is ambiguous.

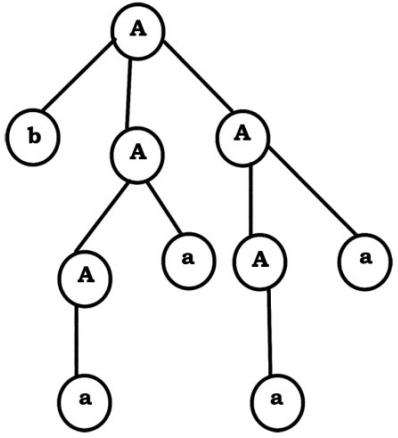
Parse tree 1 A → Aa => bAAa => bAaaa

A → baaaa



- Parse tree 2 A → bAA => bAaA => bAaAa

  A → baaaa



- Therefore, the above grammar is ambiguous.

TRY YOURSELF!

- Prove that given CFG is ambiguous, S →0B/1A A → 0/0S/1AA B → 1/1S/0BB. The string can be randomly chosen. For reference I have chosen "0011010"

- Show that grammar is S =>aSbS/ bSaS/ ε ambiguous.

RESOLVING AMBIGUITY

- Designing unambiguous grammars is tricky and requires planning from the start.
- It's hard to start with an ambiguous grammar and to manually convert it into an unambiguous one.
- Often, must throw the whole thing out and start over.
- We have just seen that this grammar is ambiguous:

  $E \rightarrow E+E|E*E|id$

  If we take a string id+id*id or id+id+id we get two parse trees.
- Goals:
  - Eliminate the ambiguity from the grammar.
  - Make the only parse trees for the grammar the ones corresponding to operator precedence.