

Chapter-6

Regular Expressions

Regular Expression (RE)

A RE is a string that can be formed according to the following rules:

1. \emptyset is a RE.
2. ϵ is a RE.
3. Every element in Σ is a RE.
4. Given two REs α and β , $\alpha\beta$ is a RE.
5. Given two REs α and β , $\alpha \cup \beta$ is a RE.
6. Given a RE α , α^* is a RE.
7. Given a RE α , α^+ is a RE.
8. Given a RE α , (α) is a RE.

if $\Sigma = \{a,b\}$, the following strings are regular expressions:

$\emptyset, \epsilon, a, b, (a \cup b)^*, abba \cup \epsilon$.

Semantic interpretation function L for the language of regular expressions:

1. $L(\emptyset) = \emptyset$, the language that contains no strings.
2. $L(\epsilon) = \{\epsilon\}$, the language that contains empty string.
3. For any $c \in \Sigma$, $L(c) = \{c\}$, the language that contains single character string c .
4. For any regular expressions α and β , $L(\alpha\beta) = L(\alpha) L(\beta)$.
5. For any regular expressions α and β , $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$.
6. For any regular expression α , $L(\alpha^*) = (L(\alpha))^*$.
7. For any regular expression α , $L(\alpha^+) = L(\alpha\alpha^*) = L(\alpha) (L(\alpha))^*$.
8. For any regular expression α , $L((\alpha)) = L(\alpha)$.

Analysing Simple Regular Expressions

$$\begin{aligned} 1. L((a \cup b)^*b) &= L((a \cup b)^*)L(b) \\ &= (L((a \cup b)))^*L(b) \end{aligned}$$

$$= (L(a) \cup L(b))^* L(b)$$

$$= (\{a\} \cup \{b\})^* \{b\}$$

$$= \{a,b\}^* \{b\}$$

$(a \cup b)^* b$ is the set of all strings over the alphabet $\{a, b\}$ that end in b .

$$2. L((a \cup b)(a \cup b)a(a \cup b)^*)$$

$$= L(((a \cup b)(a \cup b)))L(a)L((a \cup b)^*)$$

$$= L((a \cup b)(a \cup b))\{a\}(L((a \cup b)))^*$$

$$= L((a \cup b))L((a \cup b))\{a\}\{a,b\}^*$$

$$= \{a, b\}\{a, b\}\{a\}\{a, b\}^*$$

- $((a \cup b)(a \cup b)a(a \cup b)^*)$ is

$\{xay : x \text{ and } y \text{ are strings of } a\text{'s and } b\text{'s and } |x| = 2\}$.

Finding RE for a given Language

1. Let $L = \{w \in \{a, b\}^* : |w| \text{ is even}\}$.

$$L = \{aa, ab, abba, aabb, ba, baabaa, \dots\}$$

$$RE = ((a \cup b)(a \cup b))^* \text{ or } (aa \cup ab \cup ba \cup bb)^*$$

2. Let $L = \{w \in \{a, b\}^* : w \text{ starting with string } abb\}$.

$$L = \{abb, abba, abbb, abbab, \dots\}$$

$$RE = abb(a \cup b)^*$$

3. Let $L = \{w \in \{a, b\}^* : w \text{ ending with string } abb\}$.

$$L = \{abb, aabb, babb, ababb, \dots\}$$

$$RE = (a \cup b)^* abb$$

4. $L = \{w \in \{0, 1\}^* : w \text{ have } 001 \text{ as a substring}\}$.

$$L = \{001, 1001, 000101, \dots\}$$

$$RE = (0 \cup 1)^* 001 (0 \cup 1)^*$$

5. $L = \{w \in \{0, 1\}^* : w \text{ does not have } 001 \text{ as a substring}\}$.

$$L = \{0, 1, 010, 110, 101, \dots\}$$

$$RE = (1 \cup 01)^* 0^*$$

6. $L = \{w \in \{a, b\}^* : w \text{ contains an odd number of } a\text{'s}\}.$

$$L = \{a, aaa, ababa, bbaaaaba, \dots\}$$

$$RE = b^*(ab^*ab^*)^* a b^* \quad \text{or} \quad b^*ab^*(ab^*ab^*)^*$$

7. $L = \{w \in \{a, b\}^* : \#a(w) \bmod 3 = 0\}.$

$$L = \{aaa, abbaba, baaaaaa, \dots\}$$

$$RE = (b^*ab^*ab^*a)^*b^*$$

8. Let $L = \{w \in \{a, b\}^* : \#a(w) \leq 3\}.$

$$L = \{a, aa, ba, aaab, bbbabb, \dots\}$$

$$RE = b^*(a \cup \epsilon)b^*(a \cup \epsilon)b^*(a \cup \epsilon)b^*$$

9. $L = \{w \in \{0, 1\}^* : w \text{ contains no consecutive } 0\text{'s}\}$

$$L = \{0, \epsilon, 1, 01, 10, 1010, 110, 101, \dots\}$$

$$RE = (0 \cup \epsilon)(1 \cup 10)^*$$

10. $L = \{w \in \{0, 1\}^* : w \text{ contains at least two } 0\text{'s}\}$

$$L = \{00, 1010, 1100, 0001, 1010, 100, 000, \dots\}$$

$$RE = (0 \cup 1)^*0(0 \cup 1)^*0(0 \cup 1)^*$$

11. $L = \{a^n b^m / n \geq 4 \text{ and } m \leq 3\}$

$$RE = (aaaa)a^*(\epsilon \cup b \cup bb \cup bbb)$$

12. $L = \{a^n b^m / n \leq 4 \text{ and } m \geq 2\}$

$$RE = (\epsilon \cup a \cup aa \cup aaa \cup aaaa)bb(b)^*$$

13. $L = \{a^{2n} b^{2m} / n \geq 0 \text{ and } m \geq 0\}$

$$RE = (aa)^*(bb)^*$$

14. $L = \{a^n b^m : (m+n) \text{ is even}\}$

$(m+n)$ is even when both a 's and b 's are even or both odd.

$$RE = (aa)^*(bb)^* \cup a(aa)^*b(bb)^*$$

Three operators of RE in precedence order(highest to lowest)

1. Kleene star
2. Concatenation
3. Union

Eg: $(a \cup bb^*a)$ is evaluated as $(a \cup (b(b^*)a))$

Kleene's Theorem

Theorem 1:

Any language that can be defined by a regular expression can be accepted by some finite state machine.

Theorem 2:

Any language that can be accepted by a finite state machine can be defined by some regular expressions.

Note: These two theorems are proved further.

Building an FSM from a RE

Theorem 1: For Every RE, there is an Equivalent FSM.

Proof: The proof is by construction.

We can show that given a RE α ,

we can construct an FSM M such that $L(\alpha) = L(M)$.

Steps:

1. If α is any $c \in \Sigma$, we construct simple FSM shown in Figure(1)

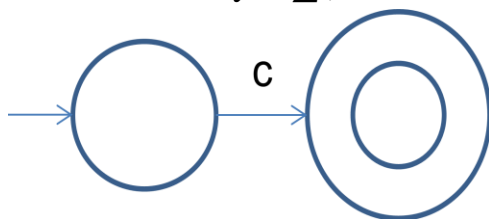


Figure (1)

2. If α is any \emptyset , we construct simple FSM shown in Figure(2).

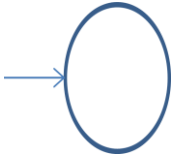


Figure (2)

3. If α is ε , we construct simple FSM shown in Figure(3).

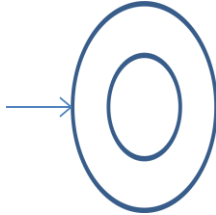


Figure (3)

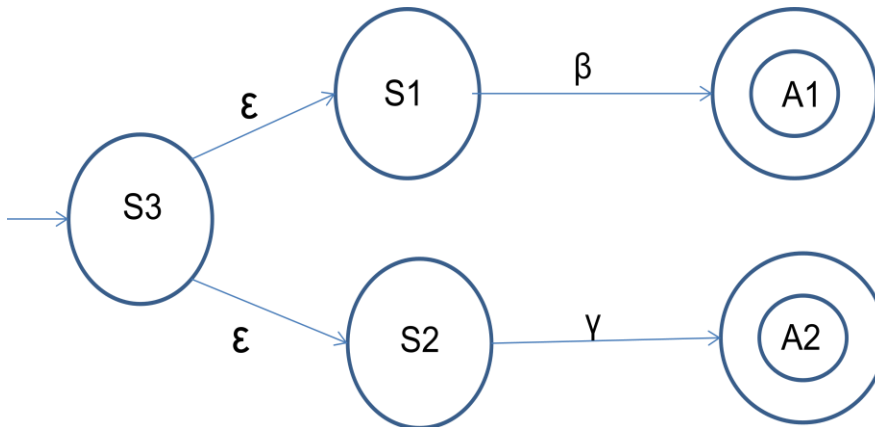
4. Let β and γ be regular expressions.

If $L(\beta)$ is regular, then FSM $M1 = (K1, \Sigma, \delta1, s1, A1)$.

If $L(\gamma)$ is regular, then FSM $M2 = (K2, \Sigma, \delta2, s2, A2)$.

If α is the RE $\beta \cup \gamma$, FSM $M3 = (K3, \Sigma, \delta3, s3, A3)$ and $L(M3) = L(\alpha) = L(\beta) \cup L(\gamma)$

$M3 = (\{S3\} \cup K1 \cup K2, \Sigma, \delta3, s3, A1 \cup A2)$, where $\delta3 = \delta1 \cup \delta2 \cup \{((S3, \varepsilon), S1), ((S3, \varepsilon), S2)\}$.



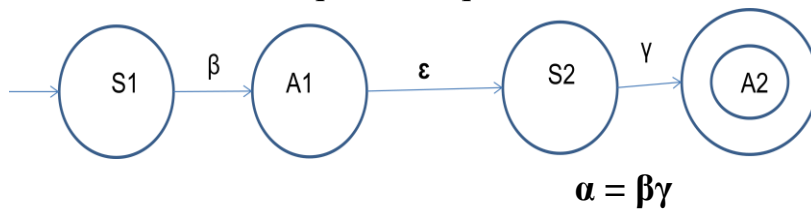
$$\alpha = \beta \cup \gamma$$

5. If α is the RE $\beta\gamma$, FSM $M3 = (K3, \Sigma, \delta3, s3, A3)$ and

$L(M3) = L(\alpha) = L(\beta)L(\gamma)$

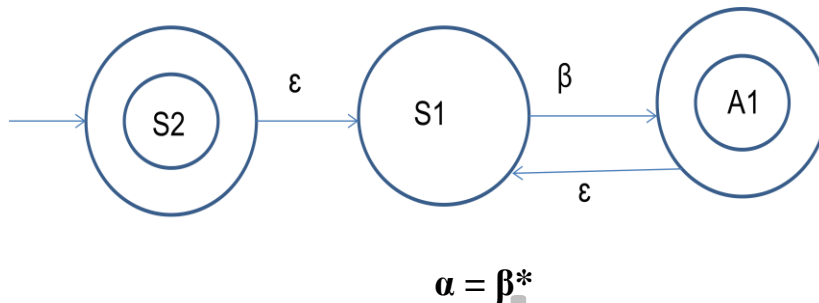
$M3 = (K1 \cup K2, \Sigma, \delta3, s1, A2)$, where

$\delta_3 = \delta_1 \cup \delta_2 \cup \{ ((q, \epsilon), S_2) : q \in A_1 \}$.



6. If α is the regular expression β^* , FSM $M_2 = (K_2, \Sigma, \delta_2, s_2, A_2)$ such that $L(M_2) = L(\alpha) = L(\beta)^*$.

$M_2 = (\{S_2\} \cup K_1, \Sigma, \delta_2, S_2, \{S_2\} \cup A_1)$, where
 $\delta_2 = \delta_1 \cup \{ ((S_2, \epsilon), S_1) \} \cup \{ ((q, \epsilon), S_1) : q \in A_1 \}$.



Algorithm to construct FSM, given a regular expression α

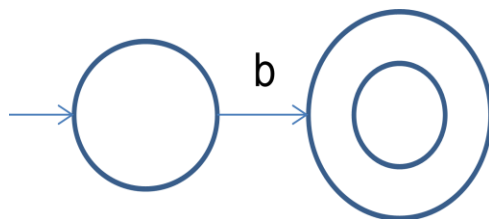
regextofsm(α : regular expression) =

Beginning with the primitive subexpressions of α and working outwards until an FSM for an of α has been built do:

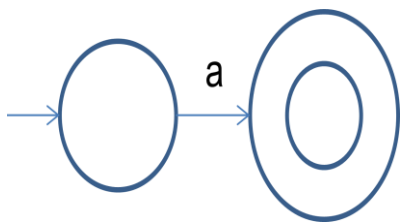
Construct an FSM as described in previous theorem.

Building an FSM from a Regular Expression

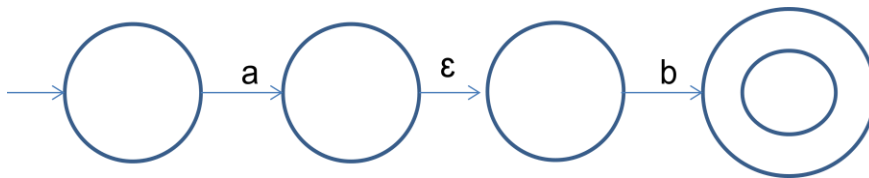
1. Consider the regular expression $(b \cup ab)^*$.



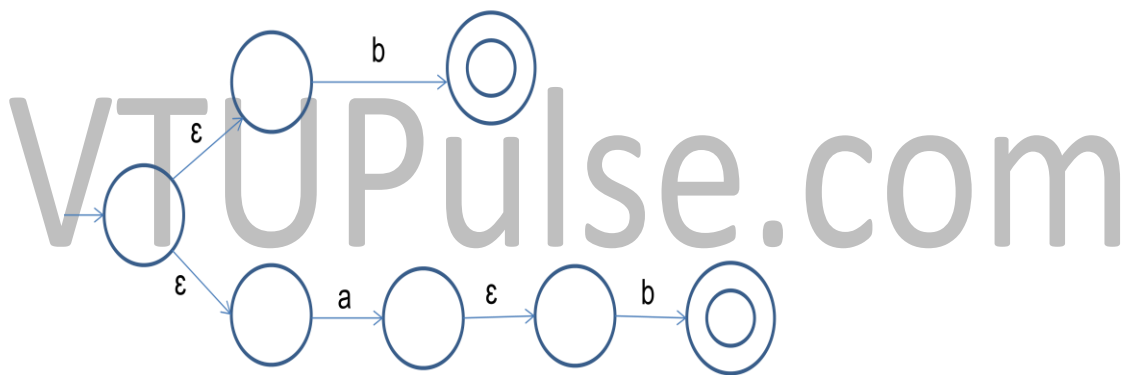
An FSM for b



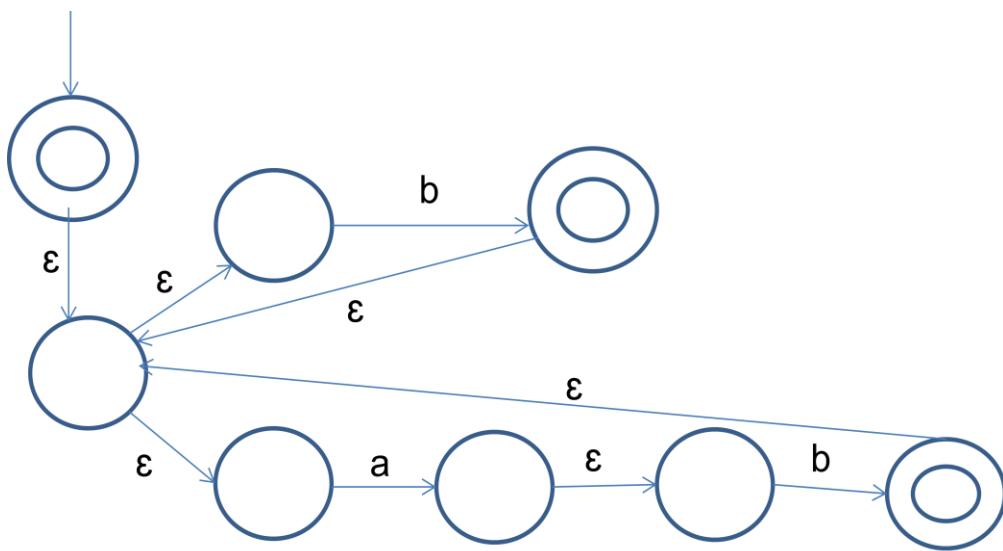
An FSM for a



An FSM for ab

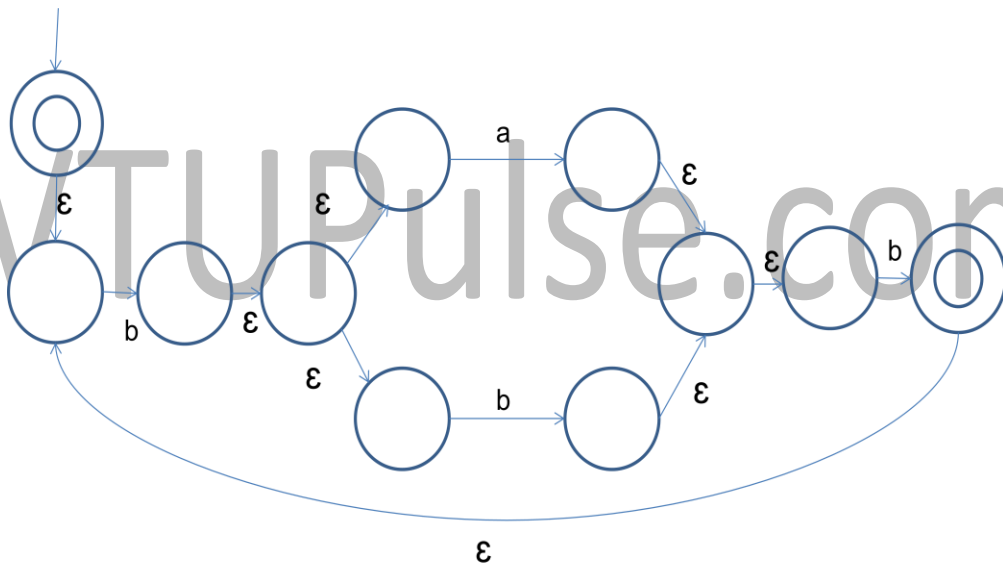


An FSM for (b U ab)

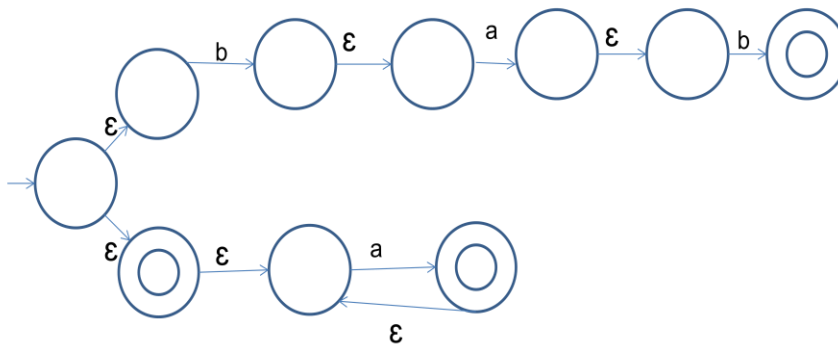


An FSM for $(b \cup ab)^*$

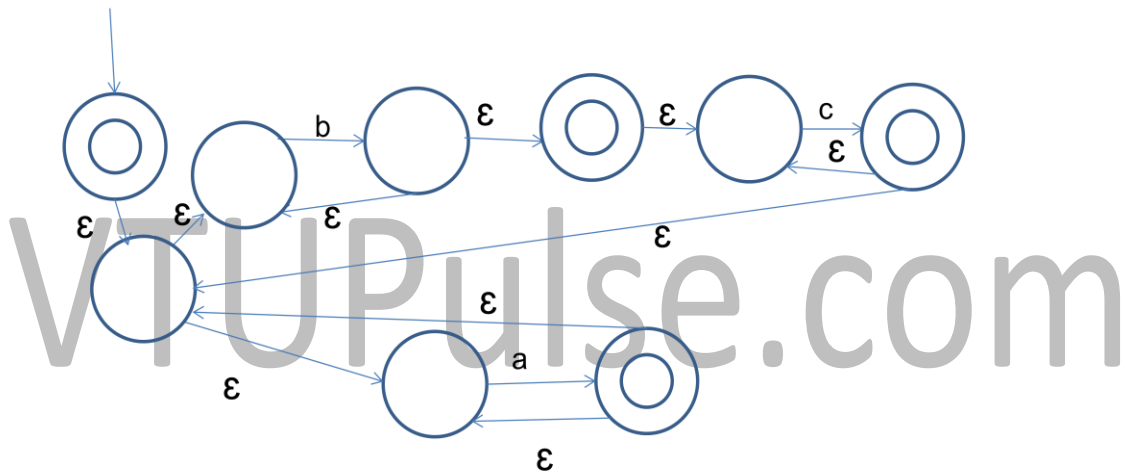
2. Construct FSM for the RE $(b(a \cup b)b)^*$



3. Construct FSM for the RE **bab U a***

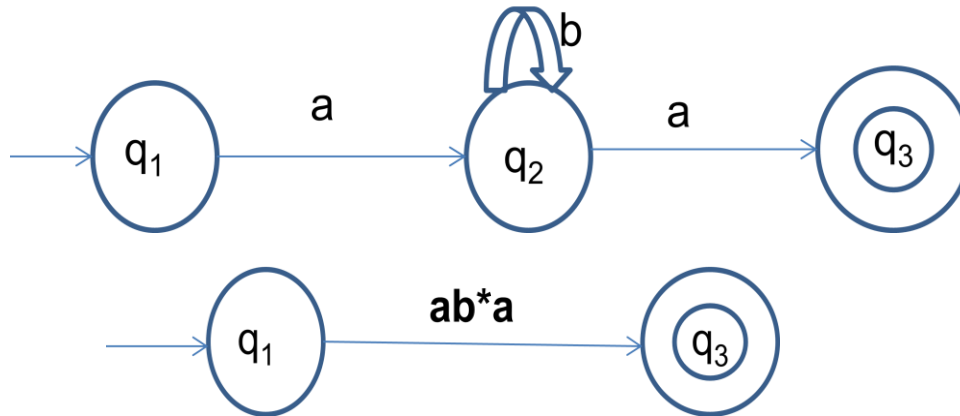


FSM for RE = $(a^* \cup b^*c^*)^*$



Building a Regular Expression from an FSM

Building an Equivalent Machine M



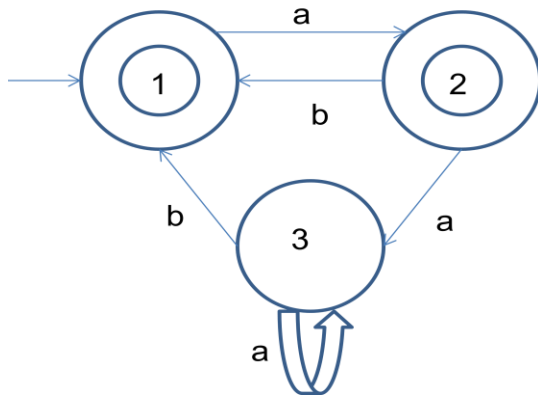
Algorithm for FSM to RE(heuristic)

fsmtoregexheuristic(M: FSM) =

1. Remove from M-any unreachable states.
2. No accepting states then return the RE \emptyset .
3. If the start state of M is has incoming transitions into it, create a new start state s.
4. If there is more than one accepting state of M or one accepting state with outgoing transitions from it, create a new accepting state.
5. M has only one state, So $L(M) = \{ \epsilon \}$ and return RE ϵ .
6. Until only the start state and the accepting state remain do:
 - 6.1. Select some state rip of M.
 - 6.2. Remove rip from M.
 - 6.3. Modify the transitions. The labels on the rewritten transitions may be any regular expression.
7. Return the regular expression that labels from the start state to the accepting state.

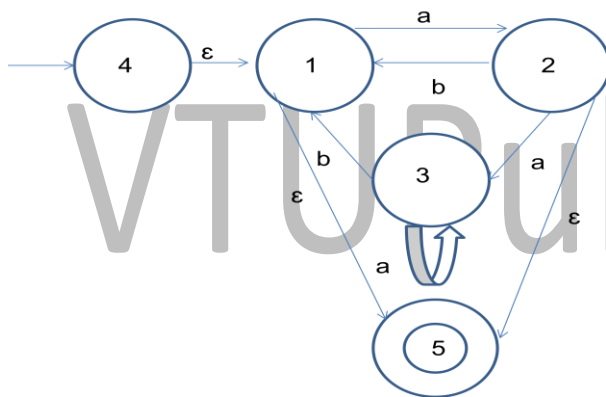
Example 1 for building a RE from FSM

Let M be:

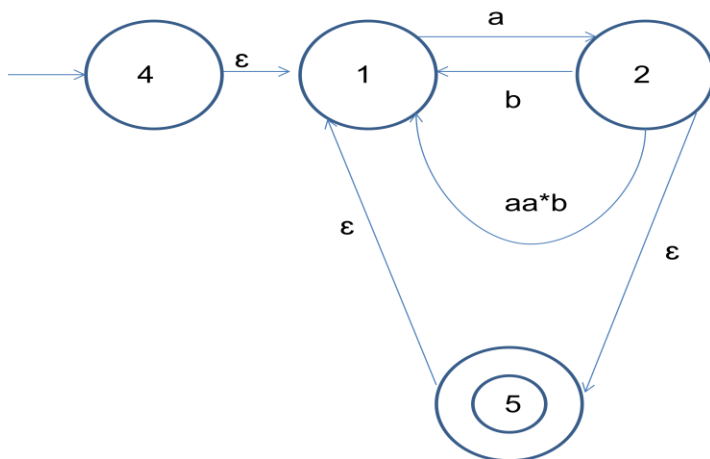


Step 1: Create a new start state and a new accepting state and link them to M

After adding new start state 4 and accepting state 5



Step 2: let rip be state 3



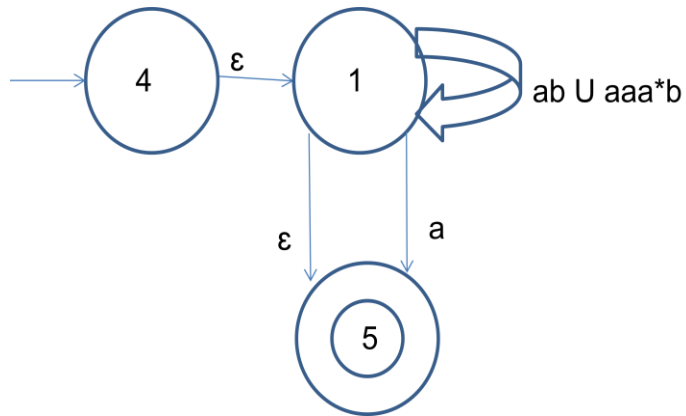
After removing rip state 3

1-2-1:ab U aaa*b

1-2-5:a

Step 3: Let rip be state 2

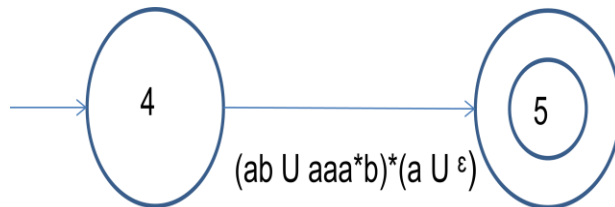
After removing rip state 2



4-1-5: (ab U aaa*b)*(a U ε)

Step 4: Let rip be state 1

After removing rip state 1



RE = (ab U aaa*b)*(a U ε)

Theorem 2 :For Every FSM ,there is an equivalent regular expression

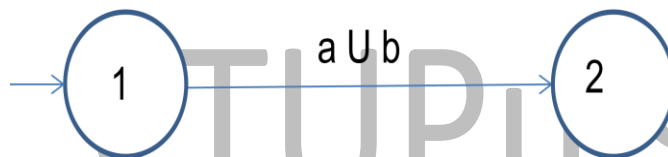
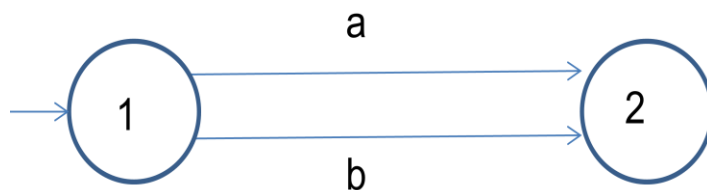
Statement : Every regular language can be defined with a regular expression.

Proof : By Construction

Let FSM $M = (K, \Sigma, \delta, S, A)$, construct a regular expression α such that

$$L(M) = L(\alpha)$$

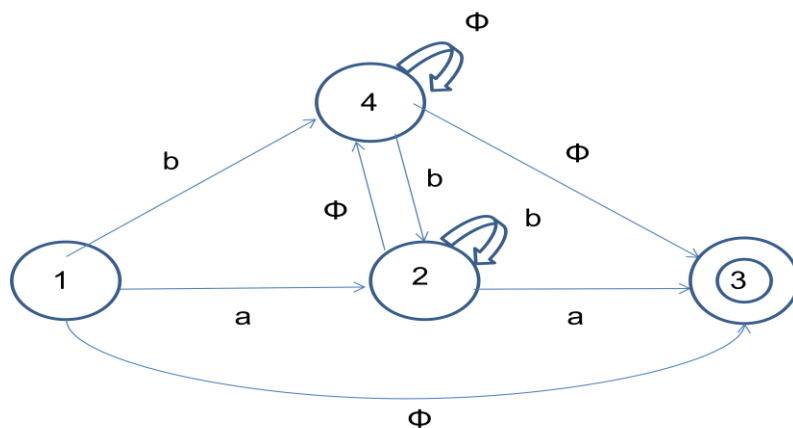
Collapsing Multiple Transitions



$\{C_1, C_2, C_3, \dots, C_n\}$ - Multiple Transition

Delete and replace by $\{C_1 \cup C_2 \cup C_3, \dots, C_n\}$

If any of the transitions are missing, add them without changing $L(M)$ by labeling all of the new transitions with the RE \emptyset .



Select a state rip and remove it and modify the transitions as shown below.

Consider any states p and q. once we remove rip, how can M get from p to q?

Let $R(p,q)$ be RE that labels the transition in M from P to Q. Then the new machine M' will be removing rip, so $R'(p,q)$

$$R'(p,q) = R(p,q) \cup R(p,rip)R(rip,rip)^*R(rip,q)$$

Ripping States out one at a time

$$R'(1,3) = R(1,3) \cup R(1,rip)R(rip,rip)^*R(rip,3)$$

$$= R(1,3) \cup R(1,2)R(2,2)^*R(2,3)$$

$$= \emptyset \cup ab^*a$$

$$= ab^*a$$

Algorithm to build RE that describes $L(M)$ from any FSM $M = (K, \Sigma, \delta, S, A)$

Two Sub Routines:

1. **standardize** : To convert M to the required form
2. **buildregex** : Construct the required RE from
modified machine M

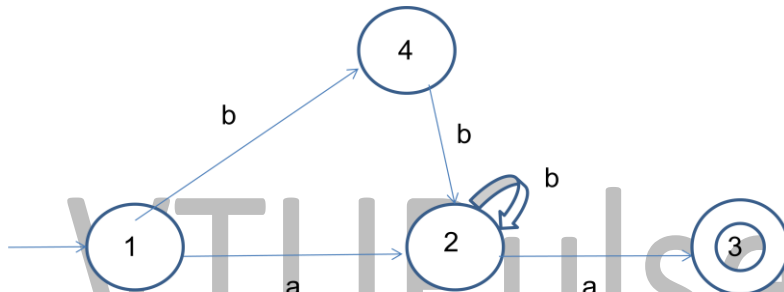
1.Standardize (M:FSM)

- i. Remove unreachable states from M
- ii. Modify start state
- iii. Modify accepting states
- iv. If there is more than one transition between states p and q, collapse them to single transition
- v. If there is no transition between p and q and $p \notin A$, $q \notin S$, then create a transition between p and q labeled Φ

2.buildregex(M:FSM)

- i. If M has no accepting states then return RE Φ
- ii. If M has only one accepting state ,return RE ϵ
- iii. until only the start state and the accepting state remain do:
 - a. Select some state rip of M
 - b. Find $R'(p,q) = R(p,q) \cup R(p,rip).R(rip,rip)^*.R(rip,q)$
 - c. Remove rip on d all transitions into ad out of it
- iv. Return the RE that labels from start state to the accepting state

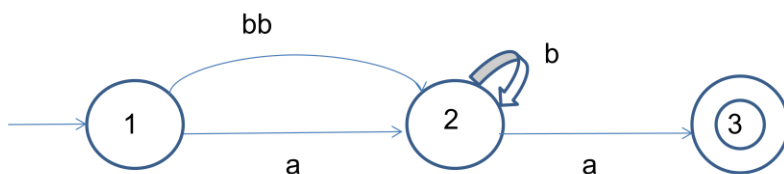
Example 2: Build RE from FSM



Step 1: let RIP be state 4

1-4-2 : bb

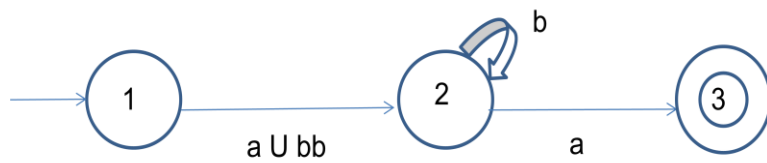
After removing rip state 4



Step 2: Collapse multiple transitions from state 1 to state 2

1-2: a U bb

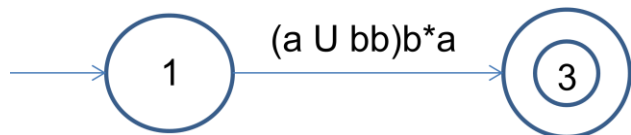
After collapsing multiple transitions from state 1 to state 2



Step 3: let rip be state 2

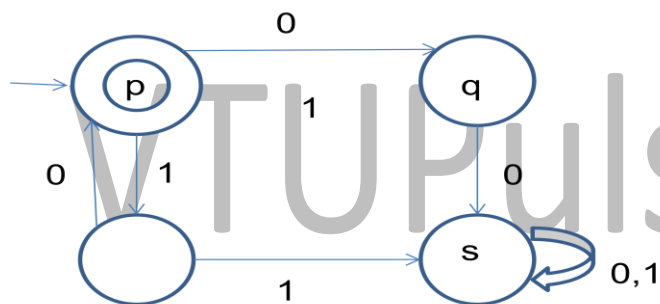
1-3: $(a \cup bb)b^*a$

After removing rip state 2



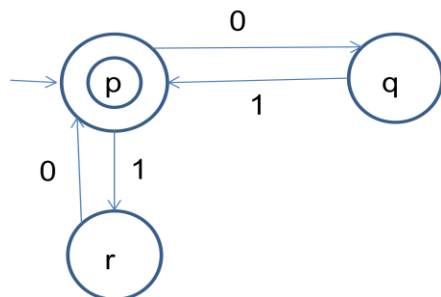
RE = $(a \cup bb)b^*a$

Example 3: Build RE From FSM



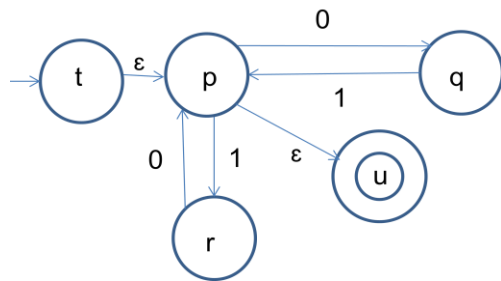
Step 1: Remove state s as it is dead state

After removing state s



Step 2: Add new start state t and new accepting state u

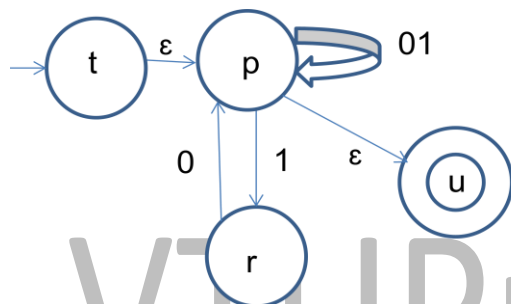
After adding t and u



Step 3: Let rip be state q

p-q-p: 01

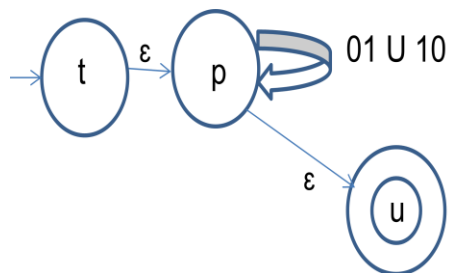
After removing rip state q



Step 4: Let rip be state r

p-r-p: 10

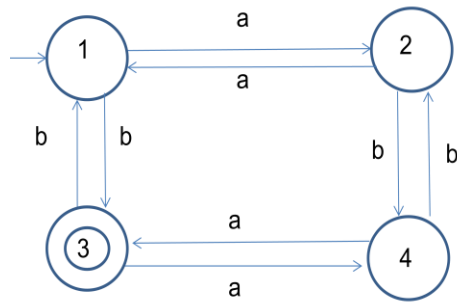
After removing rip state r



RE = (01 U 10)*

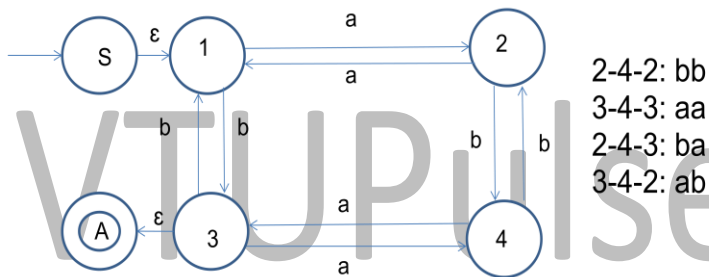
Example 4: A simple FSM with no simple RE

$L = \{w \in \{a,b\}^* : w \text{ contains an even no of a's and an odd number of b's}\}$



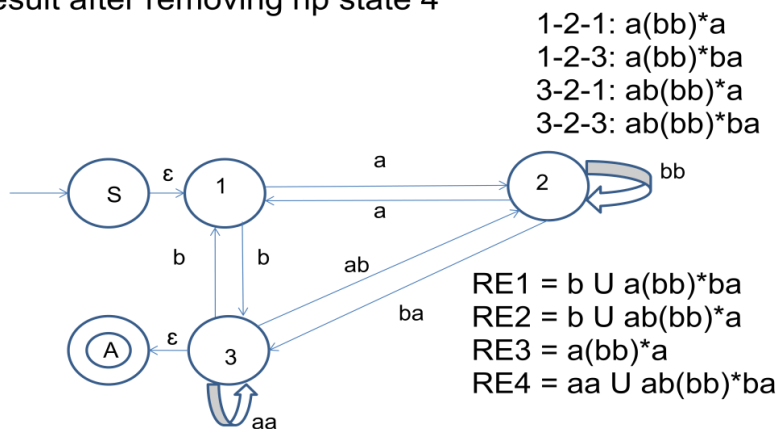
[3] even a's odd b's

Step 1: Add new start state S and new accepting state A.



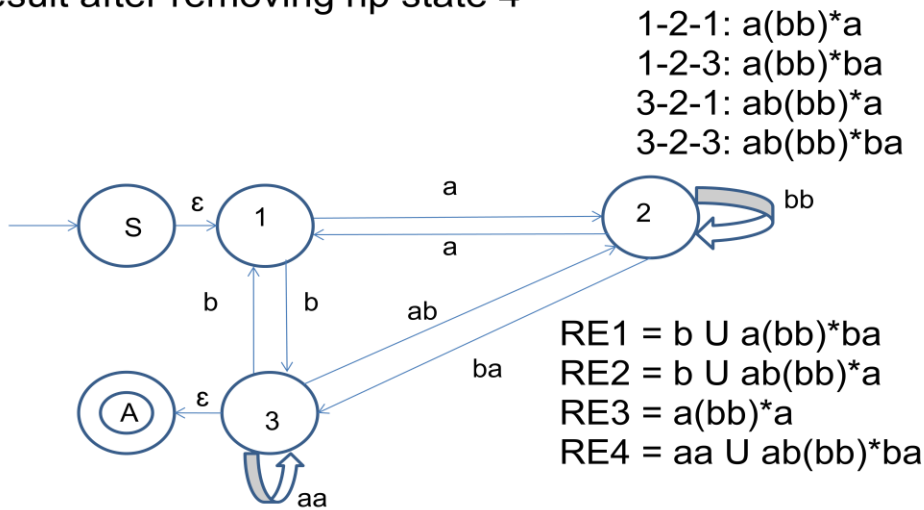
Step 2: let rip be state 4

Result after removing rip state 4



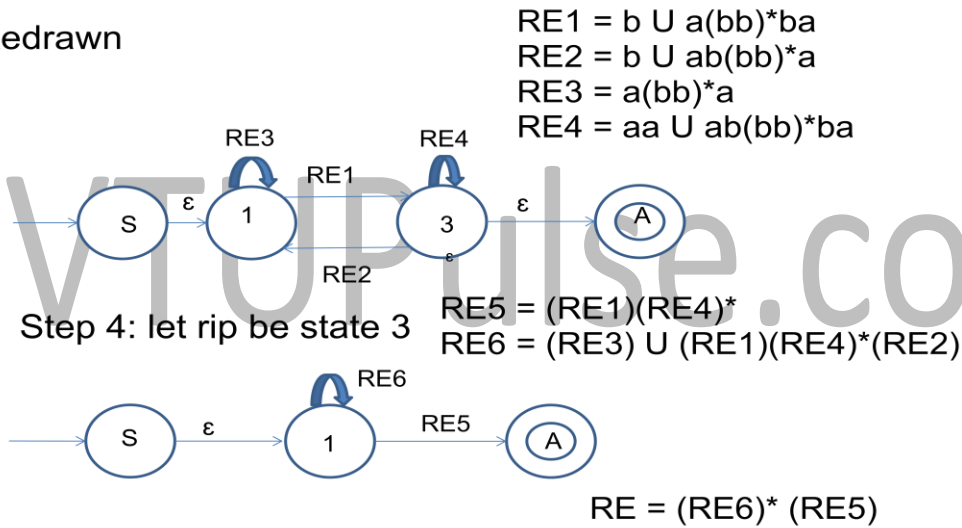
Step 3: let rip be state 2

Result after removing rip state 4



Step 3: let rip be state 2

Redrawn

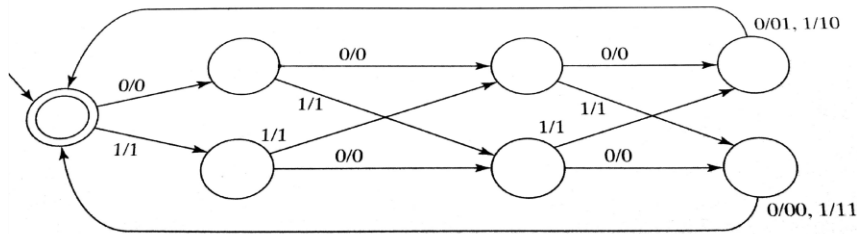


Last Step: let rip be state 1



$$\begin{aligned} RE &= (RE6)^*(RE5) \\ &= ((RE3) \cup (RE1)(RE4)^*(RE2))^*((RE1)(RE4)^*) \\ &= ((a(bb)^*a) \cup (b \cup a(bb)^*ba)(aa \cup ab(bb)^*ba)^*(b \cup ab(bb)^*a))^*((b \cup a(bb)^*ba)((aa \cup ab(bb)^*ba)^*)) \end{aligned}$$

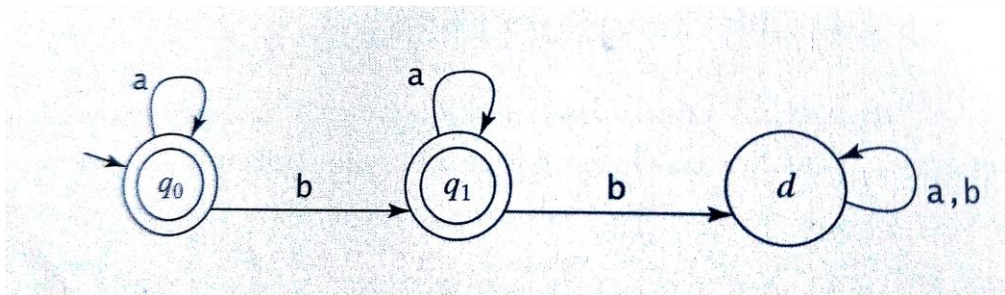
Example 5: Using fsm to regex heuristic construct a RE for the following FSM (Example 5.3 from textbook)



RE = (0000 U 0001 U 1100 U 1101 U 0010 U 1110 U
1100 U 0100 U 0011 U 1111 U 1101 U 0101)

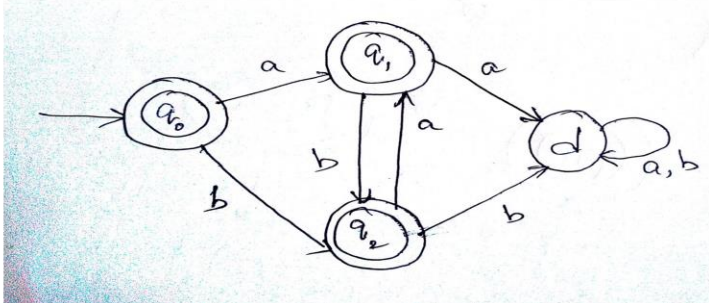
Writing Regular Expressions

- Let $L = \{w \in \{a,b\}^* : \text{there is no more than one } b\}$
 $L = \{\epsilon, b, a, aa, ab, ba, aba, baa, abaa, aabaa, \dots\}$
RE = $a^*(b \cup \epsilon)a^*$



Writing Regular Expressions

- Let $L = \{w \in \{a,b\}^* : \text{No two consecutive letters are same}\}$
RE = $(b \cup \epsilon)(ab)^*(a \cup \epsilon)$ or $(a \cup \epsilon)(ba)^*(b \cup \epsilon)$
 $L = \{\epsilon, a, b, ab, ba, aba, bab, ababa, baba, \dots\}$



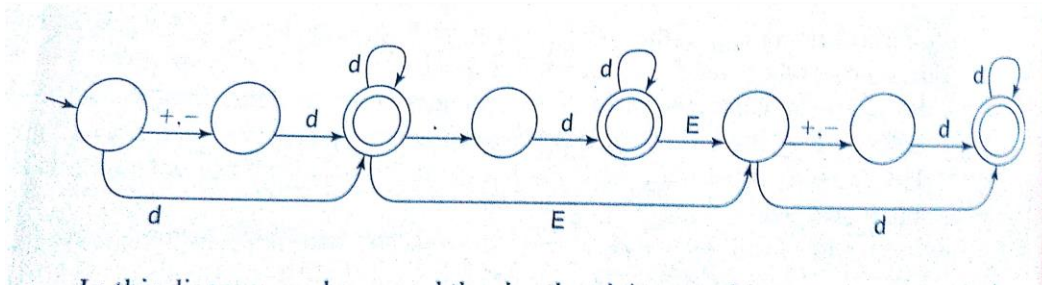
Writing Regular Expressions

- Floating point Numbers

D stands for $(0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)$

RE = $(\epsilon \cup + \cup -)D^+(\epsilon \cup .D^+)(\epsilon \cup (E(\epsilon \cup + \cup -)D^+)$

L = { 24.06, +24.97E-05,-----}



Building DFSM

- It is possible to construct a DFSM directly from a set of patterns
- Suppose we are given a set K of n keywords and a text string s.
- Find the occurrences of s in keywords K
- K can be defined by RE

$(\Sigma^*(K_1 \cup K_2 \cup \dots \cup K_n)\Sigma^*)^+$

- Accept any string in which at least one keyword occurs

Algorithm- buildkeywordFSM

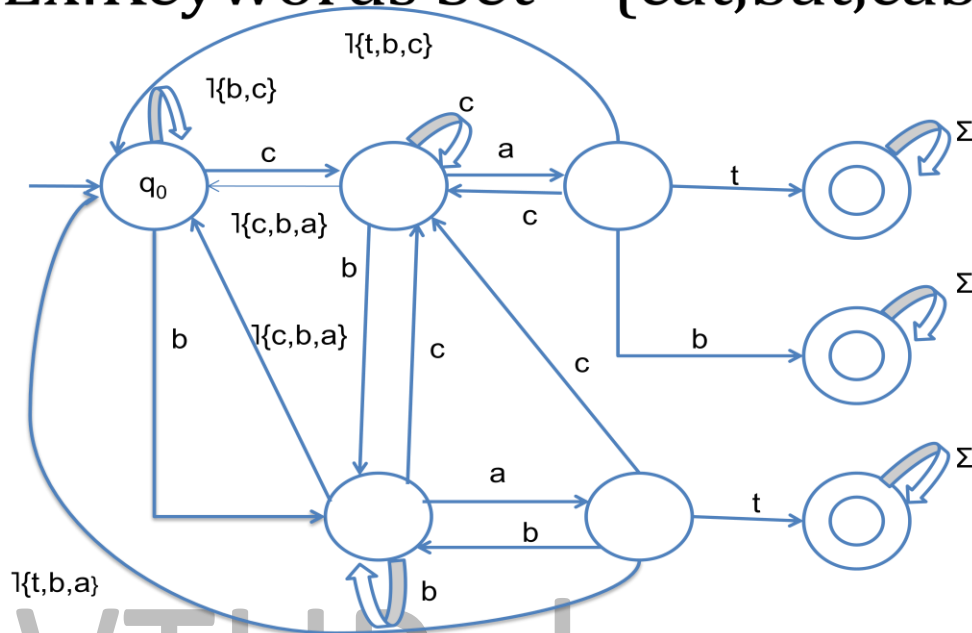
- To build dfsM that accepts any string with atleast one of the specified keywords

Buildkeyword(K:Set of keywords)

- Create a start state q_0
- For each element k of K do
Create a branch corresponding to k

- Create a set of transitions that describe what to do when a branch dies
- Make the states at the end of each branch accepting

Ex: Keywords Set = {cat,bat,cab}



Applications Of Regular Expressions

- Many Programming languages and scripting systems provide support for regular expression matching
- Re's are used in emails to find spam messages
- Meaningful words in protein sequences are called motifs
- Used in lexical analysis
- To Find Patterns in Web
- To Create Legal passwords
- Regular expressions are useful in a wide variety of text processing tasks,

- More generally string processing, where the data need not be textual.
- Common applications include data validation, data scraping (especially web scraping), data wrangling, simple parsing, the production of syntax highlighting systems, and many other tasks.

RE for Decimal Numbers

RE = -? ([0-9]+(\.[0-9]*)? | \.[0-9]+)

- $(\alpha)?$ means the RE α can occur 0 or 1 time.
- $(\alpha)^*$ means the RE α can repeat 0 or more times.
- $(\alpha)^+$ means the RE α can repeat 1 or more times.

24.23, -24.23, .12, 12. ----- are some examples

Requirements for legal password

- A password must begin with a letter
- A password may contain only letters numbers and a underscore character
- A password must contain atleast 4 characters and no more than 8 characters

$((a-z) \cup (A-Z))$

$((a-z) \cup (A-Z) \cup (0-9) \cup _)$

$((a-z) \cup (A-Z) \cup (0-9) \cup _)$

$((a-z) \cup (A-Z) \cup (0-9) \cup _)$

$((a-z) \cup (A-Z) \cup (0-9) \cup _ \cup \epsilon)$

$((a-z) \cup (A-Z) \cup (0-9) \cup _ \cup \epsilon)$

$((a-z) \cup (A-Z) \cup (0-9) \cup _ \cup \epsilon)$

$((a-z) \cup (A-Z) \cup (0-9) \cup _ \cup \epsilon)$

Very lengthy regular expression

Different notation for writing RE

- α means that the pattern α must occur exactly once.
- α^* means that the pattern may occur any number of times(including zero).
- α^+ means that the pattern α must occur atleast once.
- $\alpha\{n,m\}$ means that the pattern must occur **atleast n times** but not more than **m times**
- $\alpha\{n\}$ means that the pattern must occur **n times exactly**
- So RE of a legal password is :

$$\text{RE} = ((\mathbf{a-z}) \cup (\mathbf{A-Z}))((\mathbf{a-z}) \cup (\mathbf{A-Z}) \cup (\mathbf{0-9}) \cup _)\{3,7\}$$

Examples: RNSIT_17,Bangalor, VTU_2017 etc

- RE for an ip address is :

$$\text{RE} = ((\mathbf{0-9})\{1,3\}(\mathbf{.}(0-9)\{1,3\}))\{3\})$$

Examples: 121.123.123.123

118.102.248.226

10.1.23.45

Manipulating and Simplifying Regular Expressions

Let α , β , γ represent regular expressions and we have the following identities.

1. Identities involving union
2. Identities involving concatenation
3. Identities involving Kleene Star

Identities involving Union

- Union is Commutative

$$\alpha \cup \beta = \beta \cup \alpha$$

- Union is Associative

$$(\alpha \cup \beta) \cup \gamma = \alpha \cup (\beta \cup \gamma)$$

- Φ is the identity for union

$$\alpha \cup \Phi = \Phi \cup \alpha = \alpha$$

- union is idempotent

$$\alpha \cup \alpha = \alpha$$

- For any 2 sets A and B, if $B \subseteq A$, then $A \cup B = A$

$$a^* \cup aa = a^*, \text{ since } L(aa) \subseteq L(a^*).$$

Identities involving concatenation

- Concatenation is associative

$$(\alpha\beta)\gamma = \alpha(\beta\gamma)$$

- ε is the identity for concatenation

$$\alpha\varepsilon = \varepsilon\alpha = \alpha$$

- Φ is a zero for concatenation.

$$\alpha\Phi = \Phi\alpha = \Phi$$

- Concatenation distributes over union

$$(\alpha \cup \beta)\gamma = (\alpha\gamma) \cup (\beta\gamma)$$

$$\gamma(\alpha \cup \beta) = (\gamma\alpha) \cup (\gamma\beta)$$

Identities involving Kleene Star

- $\Phi^* = \varepsilon$

- $\varepsilon^* = \varepsilon$

- $(\alpha^*)^* = \alpha^*$

- $\alpha^*\alpha^* = \alpha^*$

- If $\alpha^* \subseteq \beta^*$ then $\alpha^* \beta^* = \beta^*$
- Similarly If $\beta^* \subseteq \alpha^*$ then $\alpha^* \beta^* = \alpha^*$

$a^*(a \cup b)^* = (a \cup b)^*$, since $L(a^*) \subseteq L((a \cup b)^*)$.

- $(\alpha \cup \beta)^* = (\alpha^* \beta^*)^*$
- If $L(\beta) \subseteq L(\alpha)$ then $(\alpha \cup \beta)^* = \alpha^*$

$(a \cup \varepsilon)^* = a^*$, since $\{\varepsilon\} \subseteq L(a^*)$.

Simplification of Regular Expressions

$$\begin{aligned}
 1. ((a^* \cup \Phi)^* \cup aa) &= (a^*)^* \cup aa && // L(\Phi) \subseteq L(a^*) \\
 &= a^* \cup aa && // (\alpha^*)^* = \alpha^* \\
 &= a^* && // L(aa) \subseteq L(a^*)
 \end{aligned}$$

$$\begin{aligned}
 2. (b \cup bb)^* b^* &= b^* b^* && // L(bb) \subseteq L(b^*) \\
 &= b^* && // \alpha^* \alpha^* = \alpha^*
 \end{aligned}$$

$$\begin{aligned}
 3. ((a \cup b)^* b^* \cup ab)^* &= ((a \cup b)^* \cup ab)^* && // L(b^*) \subseteq L(a \cup b)^* \\
 &= (a \cup b)^* && // L(a^*) \subseteq L(a \cup b)^*
 \end{aligned}$$

$$4. ((a \cup b)^* (a \cup \varepsilon) b^* = (a \cup b)^* // L((a \cup \varepsilon) b^*) \subseteq L(a \cup b)^*$$

$$\begin{aligned}
 5. (\Phi^* \cup b) b^* &= (\varepsilon \cup b) b^* && // \Phi^* = \varepsilon \\
 &= b^* && // L(\varepsilon \cup b) \subseteq L(b^*)
 \end{aligned}$$

$$\begin{aligned}
 6. (a \cup b)^* a^* \cup b &= (a \cup b)^* \cup b && // L(a^*) \subseteq L((a \cup b)^*) \\
 &= (a \cup b)^* && // L(b) \subseteq L((a \cup b)^*)
 \end{aligned}$$

$$7. ((a \cup b)^+)^* = (a \cup b)^*$$

Chapter-7

Regular Grammars

Regular grammars sometimes called as right linear grammars.

A regular grammar G is a quadruple (V, Σ, R, S)

- V is the rule alphabet which contains nonterminals and terminals.
- Σ (the set of terminals) is a subset of V
- R (the set of rules) is a finite set of rules of the form
$$X \rightarrow Y$$
- S (the start symbol) is a nonterminal.

All rules in R must:

- Left-hand side should be a single nonterminal.
- Right-hand side is ϵ or a single terminal or a single terminal followed by a single nonterminal.

Legal Rules

$S \rightarrow a$

$S \rightarrow \epsilon$

$T \rightarrow aS$

Not legal rules

$S \rightarrow aSa$

$S \rightarrow TT$

$aSa \rightarrow T$

$S \rightarrow T$

- The language generated by a grammar $G = (V, \Sigma, R, S)$ denoted by $L(G)$ is the set of all strings w in Σ^* such that it is possible to start with S .
- Apply some finite set of rules in R , and derive w .
- Start symbol of any grammar G will be the symbol on the left-hand side of the first rule in R_G

Example of Regular Grammar

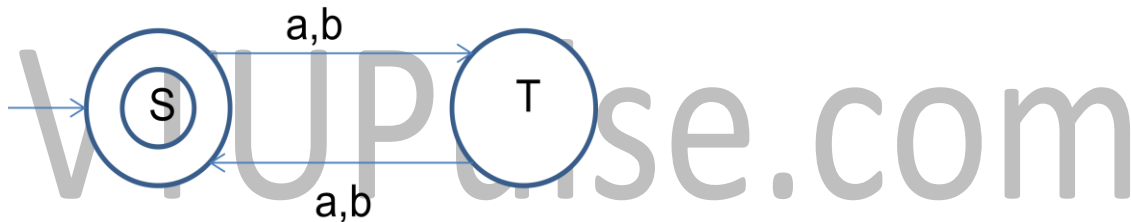
Example 1: Even Length strings

Let $L = \{w \in \{a, b\}^* : |w| \text{ is even}\}$.

The following regular expression defines L :

$((aa) \cup (ab) \cup (ba) \cup (bb))^*$ or $((a \cup b)(a \cup b))^*$

DFSM accepting L



Regular Grammar G defining L

$S \rightarrow \epsilon$

$S \rightarrow aT$

$S \rightarrow bT$

$T \rightarrow aS$

$T \rightarrow bS$

Derivation of string using Rules

Derivation of string “abab”

$S \Rightarrow aT$

$\Rightarrow abT$

$\Rightarrow abaS$

$\Rightarrow ababS$

$\Rightarrow abab$

Regular Grammars and Regular Languages

THEOREM

Regular Grammars Define Exactly the Regular Languages

Statement:

The class of languages that can be defined with regular grammars is exactly the regular languages.

Proof: Regular grammar \rightarrow FSM

FSM \rightarrow Regular grammar

The following algorithm constructs an FSM M from a regular grammar $G = (V, \Sigma, R, S)$ and assures that

$L(M) = L(G)$:

Algorithm-Grammar to FSM

grammartofsm (G: regular grammar) =

1. Create in M a separate state for each nonterminal in V .
2. Make the state corresponding to S the start state.
3. If there are any rules in R of the form $X \rightarrow w$, for some $w \in \Sigma$, then create an additional state labeled $\#$.
4. For each rule of the form $X \rightarrow wY$,

- add a transition from X to Y labeled w.
5. For each rule of the form $X \rightarrow w$, add a transition from X to # labeled w.
 6. For each rule of the form $X \rightarrow \epsilon$, mark state X as accepting.
 7. Mark state # as accepting.
 8. If M is incomplete then M requires a dead state.
Add a new state D. For every (q, i) pair for which no transition has already been defined, create a transition from q to D labeled i. For every i in Σ , create a transition from D to D labeled i.

Example 2: Grammar \rightarrow FSM

Strings that end with aaaa

Let $L = \{w \in \{a, b\}^*: w \text{ end with the pattern aaaa}\}$.

RE = $(a \cup b)^*aaaa$

Regular Grammar G

$S \rightarrow aS$

$S \rightarrow bS$

$S \rightarrow aB$

$B \rightarrow aC$

$C \rightarrow aD$

$D \rightarrow a$

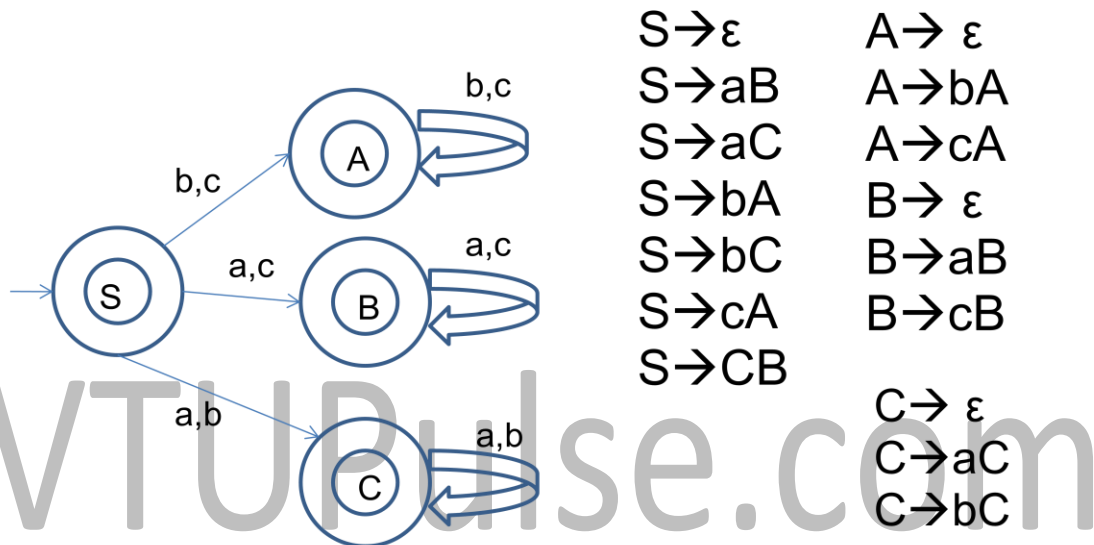
Example 3: The Missing Letter Language

Let $\Sigma = \{a, b, c\}$.

$L_{\text{Missing}} = \{ w : \text{there is a symbol } a \in \Sigma \text{ not appearing in } w \}$.

Grammar G generating L_{Missing}

FSM for Missing Letter Language



Example 4 :Strings that start with abb.

Let $L = \{w \in \{a, b\}^* : w \text{ starting with string } abb\}$.

RE = $abb(a \cup b)^*$

Regular Grammar G

$S \rightarrow aB$

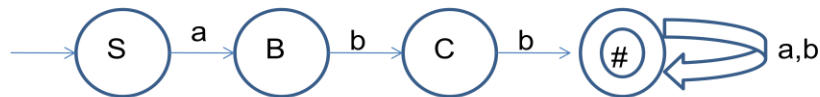
$B \rightarrow bC$

$C \rightarrow bT$

$T \rightarrow aT$

$T \rightarrow bT$

$T \rightarrow \epsilon$



Example 5 :Strings that end with abb.

Let $L = \{w \in \{a, b\}^* : w \text{ ending with string } abb\}$.

RE = $(a \cup b)^*abb$

Regular Grammar G

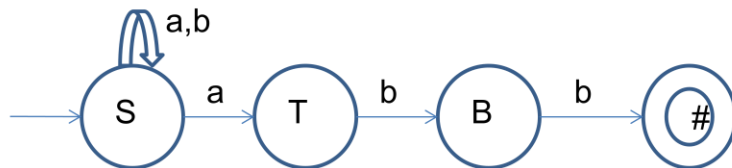
$S \rightarrow aS$

$S \rightarrow bS$

$S \rightarrow aT$

$T \rightarrow bB$

$B \rightarrow b$



Example 6: Strings that contain substring 001.

Let $L = \{w \in \{0, 1\}^* : w \text{ containing the substring } 001\}$.

$$RE = (0 \cup 1)^* 001 (0 \cup 1)^*$$

Regular Grammar G

$S \rightarrow 0S$

$S \rightarrow 1S$

$S \rightarrow 0T$

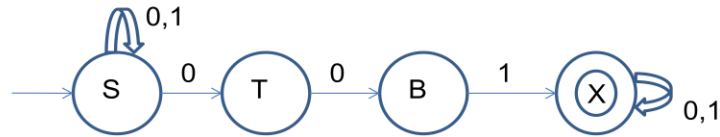
$T \rightarrow 0P$

$P \rightarrow 1X$

$X \rightarrow 0X$

$X \rightarrow 1X$

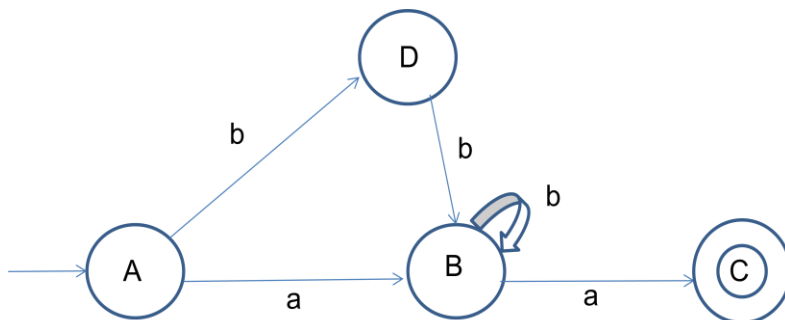
$X \rightarrow \epsilon$



Algorithm FSM to Grammar

1. Make M deterministic (to get rid of ϵ -transitions).
2. Create a nonterminal for each state in the new M.
3. The start state becomes the starting nonterminal.
4. For each transition $\delta(T, a) = U$, make a rule of the form $T \rightarrow aU$.
5. For each accepting state T, make a rule of the form $T \rightarrow \epsilon$.

Example 7: Build grammar from FSM



$$RE = (a \cup bb)b^*a$$

Grammar

$$A \rightarrow aB$$

$$A \rightarrow bD$$

$$B \rightarrow bB$$

$$B \rightarrow aC$$

$$D \rightarrow bB$$

$$C \rightarrow \epsilon$$

Derivation of string “aba”

$$A \Rightarrow aB$$

$$\Rightarrow abB$$

$$\Rightarrow abaC$$

$$\Rightarrow aba$$

Derivation of string “bba”

$$A \Rightarrow bB$$

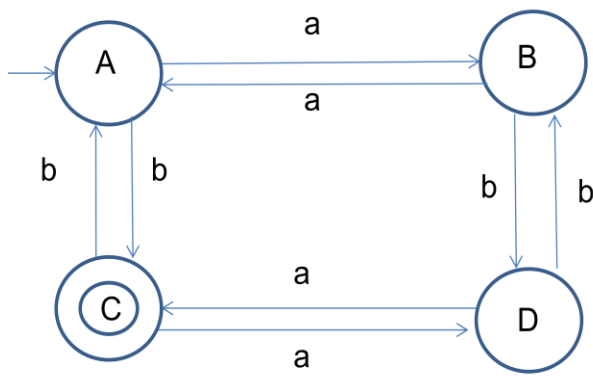
$$\Rightarrow bbB$$

$$\Rightarrow bbaC$$

$$\Rightarrow bba$$

Example 8: A simple FSM with no simple RE

$L = \{w \in \{a,b\}^* : w \text{ contains an even no of a's and an odd number of b's}\}$



Grammar

$A \rightarrow aB$

$A \rightarrow bC$

$B \rightarrow aA$

$B \rightarrow bD$

$C \rightarrow bA$

$C \rightarrow aD$

$D \rightarrow bB$

$D \rightarrow aC$

$C \rightarrow \epsilon$

Derivation of string “ababb”

$A \Rightarrow aB$

$\Rightarrow abD$

$\Rightarrow abaC$

$\Rightarrow ababA$

$\Rightarrow ababbC$

$\Rightarrow ababb$

RE, RG and FSM for given Language

Let $L = \{ w \in \{a, b\}^* : \text{every } a \text{ in } w \text{ is immediately followed by at least one } b. \}$

$L = \{ b, ab, abab, abb, \dots \}$

RE = $(ab \cup b)^*$

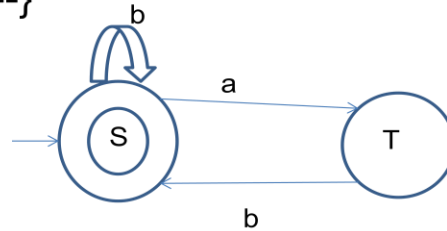
Regular Grammar

$S \rightarrow aT$

$S \rightarrow bS$

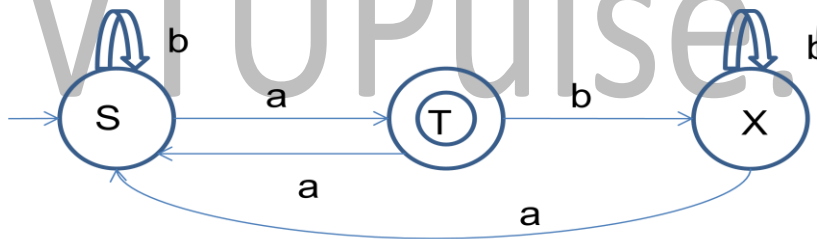
$S \rightarrow \epsilon$

$T \rightarrow bS$



Satisfying Multiple Criteria

Let $L = \{ w \in \{a, b\}^* : w \text{ contain an odd number of } a\text{'s and } w \text{ ends in } a \}$.



$S \rightarrow bS$

$S \rightarrow aT$

$T \rightarrow \epsilon$

$T \rightarrow aS$

$T \rightarrow bX$

$X \rightarrow aS$

$X \rightarrow bX$

Conclusion on Regular Grammars

- Regular grammars define exactly the regular languages.
- But regular grammars are often used in practice as FSMs and REs are easier to work.
- But as we move further there will no longer exist a technique like regular expressions.
- So we discuss about context-free languages and context-free-grammars are very important to define the languages of push-down automata.

Chapter-8

Regular and Nonregular Languages

- The language a^*b^* is regular.
 - The language $A^nB^n = \{a^n b^n : n \geq 0\}$ is not regular.
 - The language $\{w \in \{a,b\}^* : \text{every } a \text{ is immediately followed by } b\}$ is regular.
 - The language $\{w \in \{a,b\}^* : \text{every } a \text{ has a matching } b \text{ somewhere and no } b \text{ matches more than one } a\}$ is not regular.
- Given a new language L , how can we know whether or not it is regular?

Theorem 1: The Regular languages are countably infinite

Statement:

There are countably infinite number of regular languages.

Proof:

- We can enumerate all the legal DFMSs with input alphabet Σ .
- Every regular language is accepted by at least one of them.
- So there cannot be more regular languages than there are DFMSs.

- But the number of regular languages is infinite

because it includes the following infinite set of

languages:

$\{a\}, \{aa\}, \{aaa\}, \{aaaa\}, \{aaaaa\}, \{aaaaaa\} \dots$

- Thus there are at most a countably infinite number of regular languages.

Theorem 2 : The finite Languages

Statement: Every finite language is regular.

Proof:

- If L is the empty set, then it is defined by the R.E \emptyset and so is regular.

- If it is any finite language composed of the strings s_1, s_2, \dots, s_n for some positive integer n , then it is defined by

the R.E: $s_1 \cup s_2 \cup \dots \cup s_n$

- So it too is regular

- ❖ Regular expressions are most useful when the elements of L match one or more patterns.
- ❖ FSMs are most useful when the elements of L share some simple structural properties.

Examples:

- $L_1 = \{w \in \{0-9\}^*: w \text{ is the social security number of the current US president}\}.$

L_1 is clearly finite and thus regular. There exists a simple FSM to accept it.

- $L_2 = \{1 \text{ if Santa Claus exists and } 0 \text{ otherwise}\}.$
- $L_3 = \{1 \text{ if God exists and } 0 \text{ otherwise}\}.$

L_2 and L_3 are perhaps a little less clear.

So either the simple FSM that accepts $\{0\}$ or the simple FSM that accepts $\{1\}$ and nothing else accepts L_2 and L_3 .

- $L_4 = \{1 \text{ if there were people in north America more than } 10000 \text{ years ago and } 0 \text{ otherwise}\}.$
- $L_5 = \{1 \text{ if there were people in north America more than } 15000 \text{ years ago and } 0 \text{ otherwise}\}.$

L_4 is clear. It is the set $\{1\}$.

L_5 is also finite and thus regular.

- $L_6 = \{w \in \{0-9\}^*: w \text{ is the decimal representation, without leading } 0\text{'s, of a prime Fermat number}\}$

- Fermat numbers are defined by

$$F_n = 2^{2^n} + 1, n \geq 0.$$

- The first five elements of F are $\{3, 5, 17, 257, 65537\}$.
- All of them are prime. It appears likely that no other Fermat numbers are prime. If that is true, then L_6 is finite and thus regular.

- If it turns out that the set of Fermat numbers is infinite, then it is almost surely not regular.

Four techniques for showing that a language L (finite or infinite) is regular:

1. Exhibit a R.E for L .
2. Exhibit an FSM for L .
3. Show that the number of equivalence of \approx_L is finite.
4. Exhibit a regular grammar for L .

Closure Properties of Regular Languages

The Regular languages are closed under

- Union
- Concatenation
- Kleene star
- Complement
- Intersection
- Difference
- Reverse
- Letter substitution

Closure under Union, Concatenation and Kleene star

Theorem: The regular languages are closed under union, concatenation and Kleene star.

Proof: By the same constructions that were used in the proof of Kleene's theorem.

Closure under Complement

Theorem:

The regular languages are closed under complement.

Proof:

- If L_1 is regular, then there exists a DFMSM $M_1=(K,\Sigma,\delta,s,A)$ that accepts it.
- The DFMSM $M_2=(K, \Sigma,\delta,s,K-A)$, namely M_1 with accepting and nonaccepting states swapped, accepts $\neg(L(M_1))$ because it rejects all strings that M_1 accepts and rejects all strings that M_1 accepts.

Steps:

1. Given an arbitrary NDFSM M_1 , construct an equivalent DFMSM M' using the algorithm `ndfsmtoDFSM`.
2. If M_1 is already deterministic, $M' = M_1$.
3. M' must be stated completely, so if needed add dead state and all transitions to it.
4. Begin building M_2 by setting it equal to M' .
5. Swap accepting and nonaccepting states. So

$$M_2=(K, \Sigma,\delta,s,K-A)$$

Example:

- Let $L = \{w \in \{0,1\}^* : w \text{ is the string ending with } 01\}$
 $RE = (0 \cup 1)^*01$
- The complement of $L(M)$ is the DFMSM that will accept strings that do not end with 01.

Closure under Intersection

Theorem:

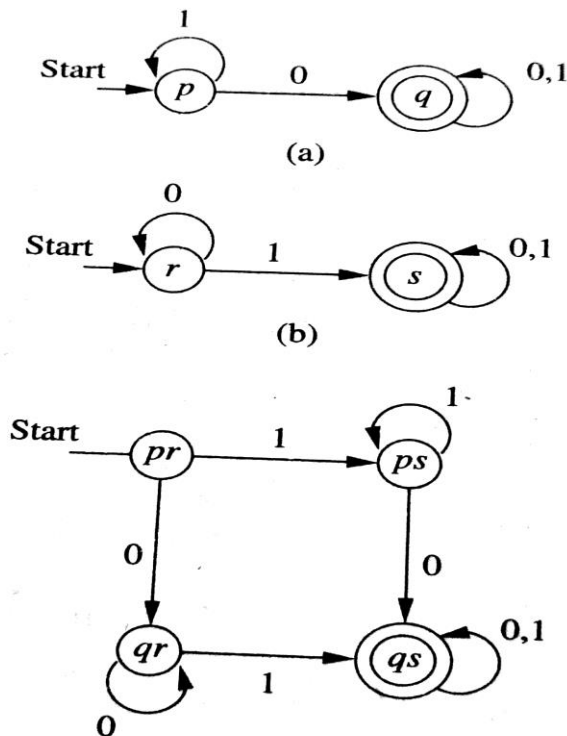
The regular languages are closed under intersection.

Proof:

- Note that

$$L(M_1) \cap L(M_2) = \neg (\neg L(M_1) \cup \neg L(M_2)).$$

- We have already shown that the regular languages are closed under both complement and union.
- Thus they are closed under intersection.
- Example:



- Fig (a) is DFSM L1 which accepts strings that have 0.
- Fig(b) is DFSM L2 which accepts strings that have 1.

- Fig(c) is Intersection or product construction which accepts that have both 0 and 1.

The Divide and Conquer Approach

- Let $L = \{w \in \{a,b\}^* : w \text{ contains an even number of } a\text{'s and an odd number of } b\text{'s and all } a\text{'s come in runs of three} \}$.

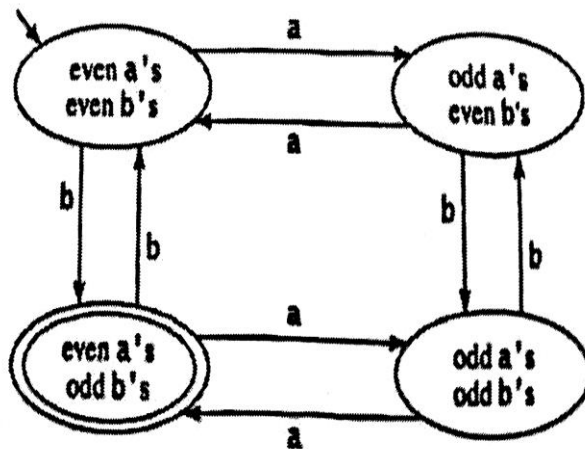
- L is regular because it is the intersection of two regular languages,

$$L = L_1 \cap L_2, \text{ where}$$

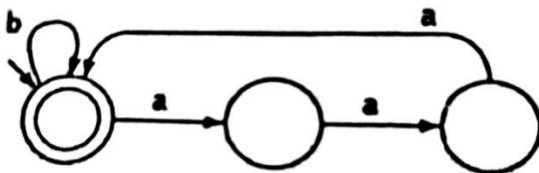
- $L_1 = \{w \in \{a,b\}^* : w \text{ contains an even number of } a\text{'s and an odd number of } b\text{'s}\}$, and

$$L_2 = \{w \in \{a,b\}^* : \text{all } a\text{'s come in runs of three}\}.$$

- L_1 is regular as we have an FSM accepting L_1



- $L_2 = \{w \in \{a,b\}^* : \text{all } a\text{'s come in runs of three}\}$.
- L_2 is regular as we have an FSM accepting L_2



$L = \{w \in \{a,b\}^* : w \text{ contains an even number of } a\text{'s and an odd number of } b\text{'s and all } a\text{'s come in runs of three} \}$.

L is regular because it is the intersection of two regular languages, $L = L_1 \cap L_2$

Closure under Set difference

Theorem:

The regular languages are closed under set difference.

Proof:

$$L(M_1) - L(M_2) = L(M_1) \cap \neg L(M_2)$$

- Regular languages are closed under both complement and intersection is shown.
- Thus regular languages are closed under set difference.

Closure under Reverse

Theorem:

The regular languages are closed under reverse.

Proof:

- $L^R = \{w \in \Sigma^* : w = x^R \text{ for some } x \in L\}$.

Example:

1. Let $L = \{001, 10, 111\}$ then $L^R = \{100, 01, 111\}$
2. Let L be defined by RE $(0 \cup 1)0^*$ then L^R is $0^*(0 \cup 1)$

$\text{reverse}(L) = \{x \in \Sigma^* : x = w^R \text{ for some } w \in L\}$.

By construction.

- Let $M = (K, \Sigma, \delta, s, A)$ be any FSM that accepts L .
- Initially, let M' be M .

- Reverse the direction of every transition in M' .
- Construct a new state q . Make it the start state of M' .
- Create an ϵ -transition from q to every state that was an accepting state in M .
- M' has a single accepting state, the start state of M .

Closure under letter substitution or Homomorphism

- The regular languages are closed under letter substitution.
- Consider any 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) = { $w \in \Sigma_2^* : \exists y \in L_1 (w = y \text{ except that every character } c \text{ of } y \text{ has been replaced by } \text{sub}(c))$ }.

• Example 1

Consider $\Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{0, 1\}$

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

$$\text{sub}(a) = 0, \text{sub}(b) = 11$$

$$\text{letsub}(a^n b^n : n \geq 0) = \{ 0^n 1^{2n} : n \geq 0 \}$$

• Example 2

Consider $\Sigma_1 = \{0, 1, 2\}$ and $\Sigma_2 = \{a, b\}$

Let **h** be any function from Σ_1 to Σ_2^* .

$$h(0) = a, h(1) = ab, h(2) = ba$$

$$h(0120) = h(0)h(1)h(2)h(0)$$

$$= aabbaa$$

$$h(01^*2) = h(0)h(1)^*h(2)$$

$$= a(ab)^*ba$$

Long Strings Force Repeated States

Theorem: Let $M=(K,\Sigma,\delta,s,A)$ be any DFSA. If M accepts any string of length $|K|$ or greater, then that string will force M to visit some state more than once.

Proof:

- M must start in one of its states.
- Each time it reads an input character, it visits some state. So, in processing a string of length n , M creates a total of $n+1$ state visits.
- If $n+1 > |K|$, then, by the pigeonhole principle, some state must get more than one visit.
- So, if $n \geq |K|$, then M must visit at least one state more than once.

The Pumping Theorem for Regular Languages

Theorem: If L is regular language, then:

$\exists k \geq 1 (\forall \text{ strings } w \in L, \text{ where } |w| \geq k (\exists x, y, z (w = xyz,$

$$|xy| \leq k,$$

$$y \neq \epsilon, \text{ and}$$

$$\forall q \geq 0 (xy^qz \in L))).$$

Proof:

- If L is regular then it is accepted by some DFSA $M=(K,\Sigma,\delta,s,A)$.

Let k be $|K|$

- Let w be any string in L of length k or greater.
- By previous theorem to accept w , M must traverse some loop at least once.

- We can carve w up and assign the name y to the first substring to drive M through a loop.
- Then x is the part of w that precedes y and z is the part of w that follows y .
- We show that each of the last three conditions must then hold:
- $|xy| \leq k$

M must not traverse thru a loop.

It can read $k - 1$ characters without revisiting any states.

But k th character will take M to a state visited before.

- $y \neq \epsilon$

Since M is deterministic, there are no loops traversed by ϵ .

- $\forall q \geq 0 (xy^qz \in L)$

y can be pumped out once and the resulting string must be in L .

Steps to prove Language is not regular by contradiction method.

1. Assume L is regular.
2. Apply pumping theorem for the given language.
3. Choose a string w , where $w \in L$ and $|w| \geq k$.
4. Split w into xyz such that $|xy| \leq k$ and $y \neq \epsilon$.
5. Choose a value for q such that xy^qz is not in L .
6. Our assumption is wrong and hence the given language is not regular.

Problems on Pumping theorem (Showing that the language is not regular)

1. Show that A^nB^n is not Regular

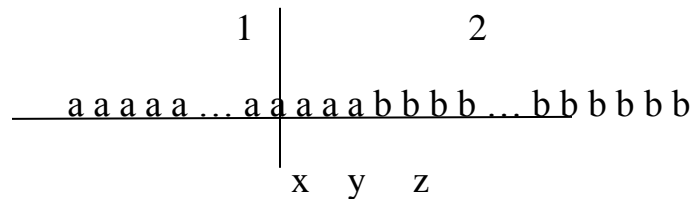
Let L be $A^nB^n = \{ a^n b^n : n \geq 0 \}$.

Proof by contradiction.

Assume the given language is regular.

Apply pumping theorem and split the string w into xyz

Choose w to be $a^k b^k$ (We get to choose any w).



We show that there is no x, y, z with the required properties:

$k \leq |xy|$,
 $y \neq \epsilon$
 $\forall q \geq 0$ (xy^qz is in L y must be in region 1.

So $y = a^p$ Since $|xy| \leq k$ for some p Let $q = 2$, producing: $a^{k+p} b^k \notin L$, since it has more a 's than b 's.

2. $\{a^i b^j : i, j \geq 0 \text{ and } i - j = 5\}$.

- Not regular.
- L consists of all strings of the form $a^i b^j$ where the number of a 's is five more than the number of b 's.
- We can show that L is not regular by pumping.
- Let $w = a^{k+5} b^k$.
- Since $|xy| \leq k$, y must equal a^p for some $p > 0$.

- We can pump y out once, which will generate the string $a^{k+5-p}b^k$, which is not in L because the number of a 's is less than 5 more than the number of b 's.

VTUPulse.com

VTUPulse.com