Limbaje Formale si Automate

Instructor *Andrei PĂUN*

Semestrul II

Departmentul de Infomrmatica
Universitatea Bucuresti
Bucuresti, Romania

Structura cursului:

1. Cunostinte de baza
   multimi, multiseturi, secvente, functii, relatii, metode si tehnici de demonstratie, alfabet, cuvinte, limbaje

2. Automate finite
   automate finite nedeterministe (NFA), automate finite deterministe (DFA), transformarea de la NFA la DFA, proprietati pentru limbajele acceptate de automate,
   lema de pompare

3. Expresii regulate
   echivalenta dintre expresiile regulate si automatele finite
   (in definirea limbajelor)

4. Gramatici independente de context (CFG)
   simplificare si forme normale, algoritmi de parsare, lema de pompare si alte proprietati

5. Automate pushdown (PDA)
nondeterministic pushdown automata (NPDA),
deterministic pushdown automata (DPDA),
PDA vs. CFGs

6. Masini Turing (TM)
diverse tipuri de TM, TM universale,
decidabilitate si nedecidabilitate,
complexitate de timp si spatiu

Motivare pentru curs

Theory of Computing

(1) Putere de calcul

(2) Complexitate

(3) Software design

(4) Circuit design

. . . . . .

I. Cunostinte de baza

1. Multimi

1) O multime este o colectie de elemente luate dintr-un univers.

2) O multime este finita daca contine un numar finit de elemente, si este infinita in caz contrar.

3) Cardinalitatea unei multimi este numarul sau de elemente si se noteaza cu $\#A$ sau $|A|$ (pentru multimea A).

– **Cum specificam o multime:**

(1) Listam toate elementele sale:

{} (or $\emptyset$),

$A = \{2,\ 3,\ 5,\ 7\}$**,**

$B = \{3,\ 2,\ 2,\ 7,\ 5,\ 7\}$

(2) Precizam o proprietate: $\{x \mid P(x)\}$

$C = \{x \mid x$ **este numar prim**$\}$**,**

$D = \{x \mid x$ **este un cuvant in romana si** x **este si palindrom**$\}$**,**

$E = \{x \mid x$ **este un intreg divizibil cu 3 si mai mic decat 10** $\}$

(3) Definite recursiv:

Forma generala:

i) Anumite elemente initiale sau de baza sunt in A

ii) Daca $a,\ b \in A$**, atunci** $a \circ b \in A$

iii) Nici un alt element nu apartine lui A

Exemplu Definirea lui A:

i) $2 \in A$

ii) Daca $a,\ b \in A$**, atunci** $a + b \in A$

iii) Nici un alt element nu apartine lui A

(4) Inchiderea la operatii

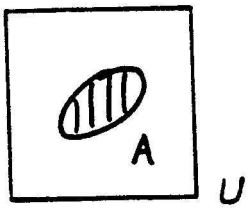
- **Spunem ca o multime** A **este inchisa la o operatie binara** $\circ$ **daca pentru toate** $a,\ b \in A$**,** $a \circ b \in A$
- **Fie** $B \supseteq A$ **si** B **este inchisa la** $\circ$**. Spunem ca** B **este o inchidere a lui** A **la operatia** $\circ$ **daca nu exista** B' **astfel incat** $A \subseteq B' \subset B$ **si** B' **este inchisa la** $\circ$**.**
- **Forma generala (pentru multimi definite prin inchidere)**
  A **este cea mai mica multime care contine anumite elemente initiale (de baza) si care este inchisa la operatia** $\circ$**.**

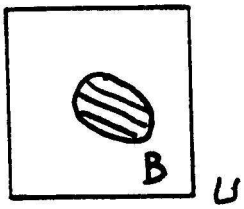
- **Exemplu**

  B este cea mai mica multime care il contine pe τ si este inchisa la "+".

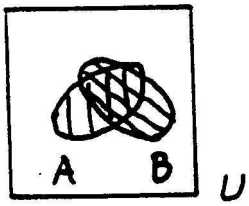
— **Operatii cu multimi**

Daca se dau A si B doua multimi:

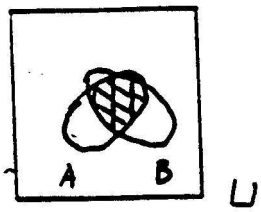




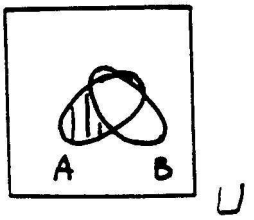
$A \cup B = \{x \mid x$ **este in** A **sau** x **este in** $B\}$



$A \cap B = \{x \mid x$ **este in** A **si** x **este si in** $B\}$



$A - B = \{x \mid x$ **este in** A **si** x **nu este in** $B\}$



$\cup_{i=1}^{n} A_i = \{x \mid x \textbf{ este in } A_i, \textbf{ pentru un } i,\ 1 \leq i \leq n\}$
$\cap_{i=1}^{n} A_i = \{x \mid x \textbf{ este in } A_i,\ 1 \leq i \leq n\}$

— Proprietati ale operatiilor cu multimi

Distributivitate

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

Indempotenta

$A \cup A = A;\ A \cap A = A$

Involutie

$\neg(\neg A) = A$

Comutativitate

$A \cap B = B \cap A;\ A \cup B = B \cup A$

Asociativitate

$A \cup (B \cup C) = (A \cup B) \cup C$
$A \cap (B \cap C) = (A \cap B) \cap C$

Regulile lui De Morgan

$\neg(A \cup B) = \neg A \cap \neg B$

$\neg(A \cap B) = \neg A \cup \neg B$

— **Multimea submultimilor**

Multimea tuturor submultimilor unei multimi A ("power set" of A) se noteaza cu 2^A.

Exemplu:

$A = \{2, 3, 5\}$

$2^A = \{\emptyset, \{2\}, \{3\}, \{5\}, \{2, 3\}, \{2, 5\}, \{3, 5\}, \{2, 3, 5\}\}$

— **Cateva multimi speciale:**

$\emptyset$: multimea vida;

$\mathcal{Z}$: multimea intregilor;

$\mathcal{N}$: multimea intregilor pozitivi;

$\mathcal{N}_0$: multimea intregilor care nu sunt negativi;

$\mathcal{R}$: multimea numerelor reale.

. . . . . .

2. **Multiseturi**

   Un multiset este o colectie de elemente dintr-un univers oarecare, colectie in care repetitiile nu sunt ignorate.

3. **Secvente**

   O secventa de elemente dintr-un univers oarecare este o lista de elemente care sunt ordonate (fiecare element are o pozitie).

4. Functii

Definitie Daca se dau doua multimi A si B, o functie (partiala) de la A la B asociaza cu fiecare a din A (cel mult) un element b din B.

Notatii:

functie $f : A \to B$
asocierea $f(a) = b$
nedefinita $f(a) = undef$

Functii speciale:

functia identitate:
$f : A \to A$ **si**
$f(a) = a$ **pentru toti** $a \in A$

Functia caracteristica a lui A **pentru** B**:**
$B \subseteq A$
$f : A \to \{true,\ false\}$
$f(a) = true,\ if\ a \in B$
$f(a) = false,\ if\ a \in A - B$

Cateva proprietati ale functiilor $f : A \to B$

totala:
$f(a)$ **este definita pentru toate elementele** a **din** A

surjectiva:
pentru toate elementele b **din** B **exista un** a **din** A **astfel incat** $f(a) = b$

injectiva:
$a,\ a' \in A,\ a \neq a'$ **implica faptul ca** $f(a) \neq f(a')$

bijectiva:
in acelasi timp si surjectiva si injectiva

Notatia big-O

$f(n) = O(g(n))$ **daca exista** $c > 0$ **si intregul pozitiv** d **astfel incat pentru toate** $n \geq d$

$$f(n) \leq cg(n)$$

$f(n) = \Omega(g(n))$ **daca**

$$cf(n) \geq g(n)$$

$f(n) = \Theta(g(n))$ **daca**

$$f(n) = O(g(n)) \text{ si } f(n) = \Omega(g(n))$$

functie (partiala) m,n

$$f : A_1 \times A_2 \times \ldots \times A_m \to B_1 \times \ldots \times B_n$$

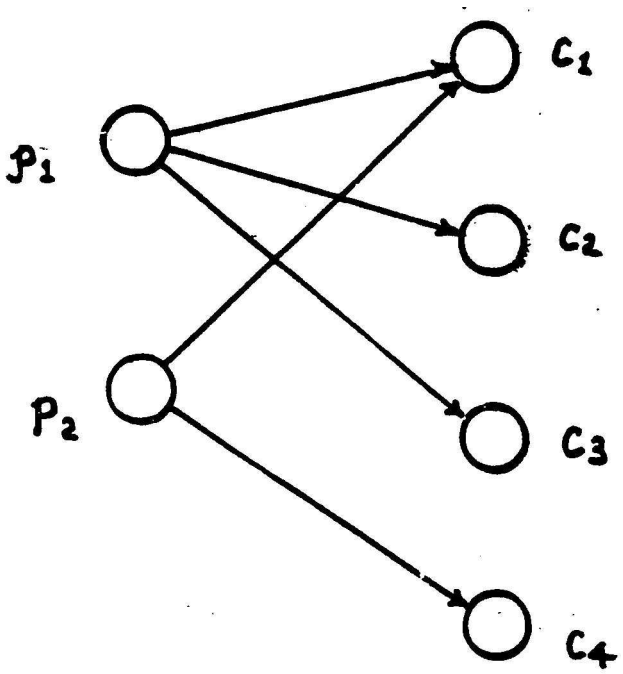
$$f(a_1, a_2, \ldots, a_m) = (b_1, \ldots, b_n)$$

sau $f(a_1, a_2, \ldots, a_m) = undef$

5. Relatii

O relatie n-ara R, $n \geq 1$, pentru multimile $A_1, \ldots, A_n$ este orice submultime R a produsului cartezian $A_1 \times \ldots \times A_n$.

Exemplu



$P = \{p_1, p_2\}, \ C = \{c_1, c_2, c_3, c_4\}$
$T \subseteq P \times C$
$T = \{(p_1, c_1), (p_1, c_2), (p_1, c_3), (p_2, c_1), (p_2, c_4)\}$

Compunerea relatiilor

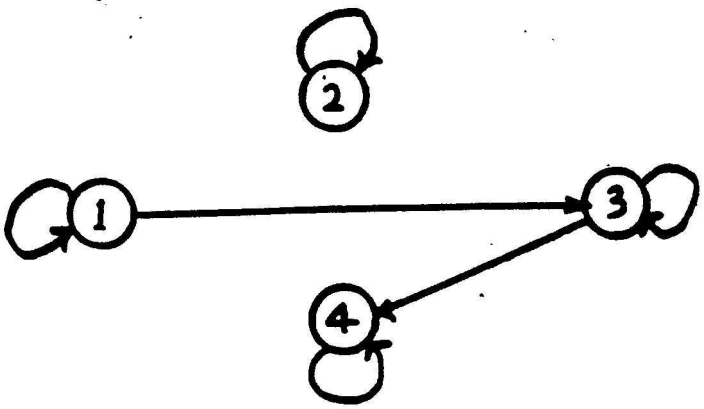
Fie $R \subseteq A \times B$ si $S \subseteq B \times C$ doua relatii. Atunci compunerea lui R cu S, notata prin $R \circ S$, este relatia $R \circ S \subseteq A \times C$, definita prin:
$R \circ S = \{(a, c) \mid (a, b)$ **este in** R **si** (b, c) **este in** S
pentru un b **din** $B\}$

Compunerea unei relatii $R : A \times A$ cu ea insasi

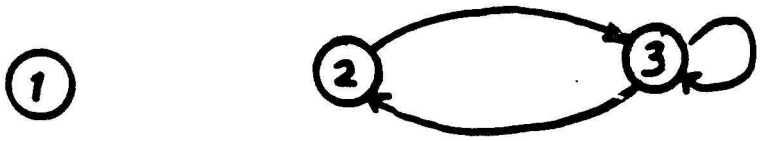
$R^0 \ : \ \{(a, a) \mid a$ **este in** $A\}$
$R^1 \ : \ R$
$R^i, \ _{i \geq 1} \ : \ R \circ R^{i-1}$
$R^+ = \cup_{i=1}^{\infty} R^i$ **inchiderea tranzitiva a lui** R
$R^* = \cup_{i=0}^{\infty} R^i$ **inchiderea reflexiva si tranzitiva a lui** R

Proprietatile relatiei $R \subseteq A \times A$

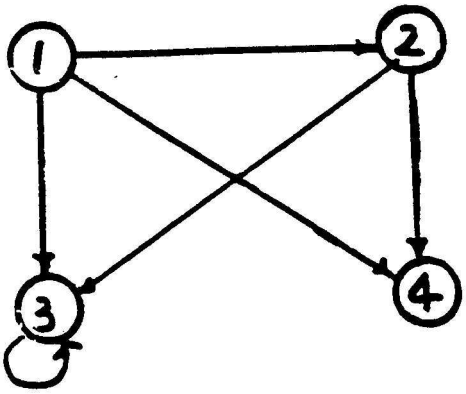
- **reflexiva:** aRa **pentru toti** a **din** A



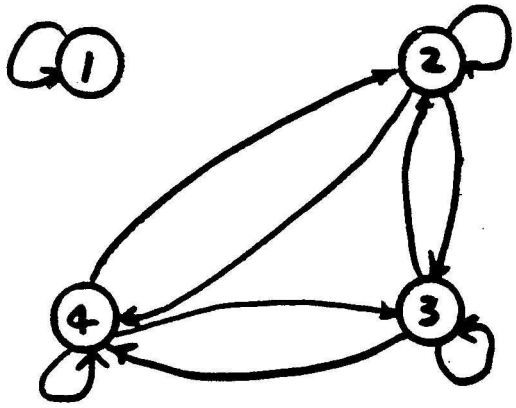
- **simetrica:** aRb **implica** bRa



- **antisimetrica:** $aRb,\ bRa$ **implica** $a = b$



- **tranzitiva:** $aRb,\ bRc$ **implica** aRc



△ **O relatie binara** R **peste** A **este o relatie de echivalenta daca este**

- **reflexiva,**
- **simetrica, si**
- **tranzitiva.**

△ **O relatie de echivalenta** R **peste** A **defineste submultimi disjuncte ale lui** A**, numite clase de echivalenta.**

$$[a]_R = \{b \mid b \text{ este in } A \text{ si } aRb\}$$

• **Fie R o relatie binara de echivalenta peste A si $B \subseteq A$.**
Spunem ca B este R-inchisa daca pentru toti a din B, aRb implica faptul ca b este in B.

Exemplu A : **multimea intregilor;**
$B = \{9, 4, 3, 2\}$**;**
R :**"este patratul lui" ("is the square of")**
Atunci B este R-inchisa.

• **R este o relatie n-ara peste A. $B \subseteq A$ este R-inchisa daca pentru toate $b_1, \ldots, b_{n-1}$ din B, avem ca daca $(b_1, \ldots, b_{n-1}, b_n)$ este in R atunci b_n este in B.**

Exemplu A : **multimea tuturor intregilor;**
$R = \{(a, b, c) \mid a \times b = c\}$**;**
$B = \{ia \mid i \geq 1\}$ **pentru un intreg a**
Atunci B este R-inchisa.

<u>Lema</u> Fie R o relatie de echivalenta peste A. Atunci pentru toti a, b din A avem sau $[a]_R = [b]_R$ sau $[a]_R \cap [b]_R = \emptyset$

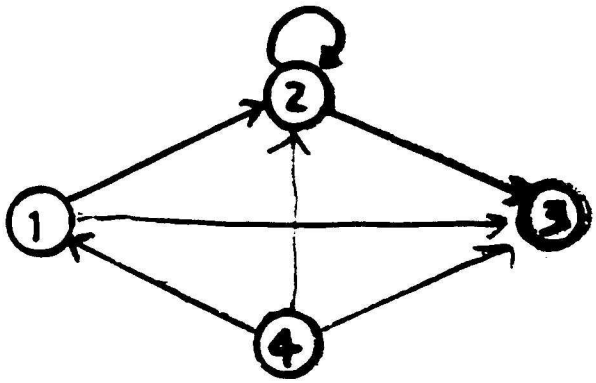
△ **O relatie binara R peste A este o ordine partiala daca este**

- **reflexiva,**
- **tranzitiva, si**
- **antisimetrica.**

△ **<u>Inchidere</u>**

– R^+ este inchiderea tranzitiva a lui R

– O reprezentare in digraf pentru o inchidere tranzitiva:



6. Cardinalitate

△ **Daca se dau doua multimi A si B, spunem ca ele au cardinalitate egala daca exista o bijectie $f : A \to B$.**

Exemplu $\#\mathcal{Z} = \#\mathcal{N}_0$

$$f(i) = -2i - 1, \ i < 0$$
$$f(i) = 2i, \ i \geq 0$$

△ **O multime este numarabila sau enumerabila daca este ori finita ori infinita, caz in care are aceeasi cardinalitate cu $\mathcal{N}$.**

7. Metode de demonstratie

— demonstratie directa

— demonstratie prin contradictie

— demonstratie prin inductie

△ **Demonstratie directa**

Enumeram toate cazurile pentru a arata ca proprietatea este adevarate.

Exemplu Orice numar par dintre 4 si 2^{32} este suma a doua numere prime.

△ **Demonstratie prin contradictie**

Stabilim validitatea propozitiei presupunand ca nu este adevarata si inferam o contradictie.

Exemplu Exista un numar infinit de numere impare.

△ Demonstratia prin inductie

– baza

– ipoteza inductiva

– pasul inductiv

– concluzie

Two forms:
(1) — Prove it holds for a basis value b,
— Hypothesize it holds for value n,
— Prove it holds for value $n+1$.

(2) — Prove it holds for basis values,
— Hypothesize it holds for all values less than n and greater than basis value,
— Prove it holds for value n.

Example (proof by induction)

(i) no two lines are parallel;

(ii) no three lines have a common intersection point.

Prove that the number of regions in an arrangement of n lines is

$$1 + n(n+1)/2$$

8. Proof Techniques

— pigeonhole principle

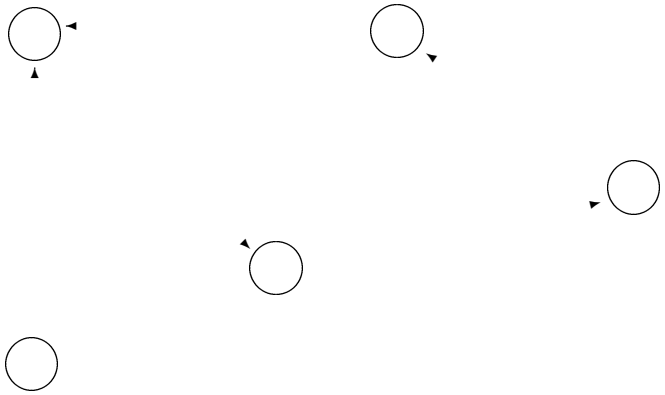
— counting

— diagonalisation

The Pigeonhole Principle
Let $n \geq 1$ be an integer, let there be n pigeonholes, and there be $n+1$ items of mail to be placed in the pigeonholes. Then, however the the items are placed, at least one pigeonhole will contain at least two items.

The Pigeonhole Principle(Extended version)
Let $q_1, \ldots, q_t \geq 1$. There exists a positive integer $N(q_1, \ldots, q_t)$. If $n \geq N(q_1, \ldots, q_t)$, then however n items are placed into t pigeonholes, there exists i, $1 \leq i \leq t$ such that the i-th pigeonhole contains at least $q_1 + 1$ items.

Example **Let $G = (V, E)$ be a digraph with $\#V = n \geq 1$. Then G has a cycle if and only if G has a path of length at least n.**



9. Alphabets, Words, and Languages

△ **An alphabet is a finite, nonempty set of elements.**
The elements of the alphabet are called symbols or letters.

△ **A word over an alphabet Σ is a finite sequence of symbols from Σ.**

△ **A language is a set of words.**

Examples:
(1) English alphabet, words and language.

(2) Alphabet: $\{a,\ b\}$
words: $\lambda,\ a,\ b,\ ab,\ ba,\ \ldots\ldots$
language: $\{\lambda,\ ab,\ aabb,\ aaabbb,\ \ldots\ldots\}$
i.e. $\{a^i b^i \mid i \geq 0\}$

(3) Alphabet : Fortran reserved words and identifiers
words : a Fortran program
Language : the set of all Fortran programs.

△ **Empty word is a word over every alphabet.**

$$\lambda$$

△ **The length of a word is the number of symbols in the given word.**

Example: $|\lambda| = 0$

$|abbc| = 4$

△ **The a-length of a word x is the number of times a occurs in x.**

Example: $|01101|_0 = 2$

$|cbcbc|_a = 0$

△ **The <u>catenation</u> of two words x and y, denoted by xy, is the word obtained by appending the word y to x.**

<u>Example</u> $x = abc$
$y = defg$
$xy = abcdefg$
$yx = defgabc$

$$x\lambda = \lambda x = x$$
$$\lambda\lambda = \lambda$$

△ **Catenation is <u>associative</u>:**

$$(xy)z = x(yz)$$

△ $|xy| = |x| + |y|$

△ $x^0 = \lambda$ **(Power of a word)**
$x^i = xx^{i-1},\ i \geq 1$

<u>Example</u> $(abc)^0 = \lambda$
$(abc)^3 = abcabcabc$

△ **Prefix: x is a prefix of y if $\exists z$ such that**

$$xz = y$$

Example
- ***not* is a prefix of *notice***
- **λ is a prefix of any word**
- **Any word is a prefix of itself**

△ **Suffix: x is a suffix of y if $\exists z$ such that**

$$zx = y$$

Example
- ***allow* is a suffix of *swallow***
- **λ is a suffix of any word**
- **01011 is a suffix of 01011**

△ **Subword: x is a subword of y if $\exists w, z$ s.t.**

$$wxz = y$$

Example The subwords of *beat* are:

$$\lambda,\ b,\ e,\ a,\ t,\ be,\ ea,\ at,\ bea,\ eat,\ beat$$

△ **Proper prefix (suffix, subword):**
x is a proper prefix (suffix, subword) of y if x is a prefix (suffix, subword) of y and $x \neq \lambda$ and $x \neq y$.

△ **Reversal:**

(i) $x^R = x$, **if** $x = \lambda$;

(ii) $x^R = y^R a$, **if** $x = ay$ **for** $a \in \Sigma$, y **is a word**

Lemma If $x = a_1 \dots a_n$ **for some** $n \geq 0$, **then**

$$x^R = a_n \dots a_1.$$

Proof: By induction on $|x|$.

Basis: $|x| = 0$. $\lambda^R = \lambda$ **by (i).**

I.H.: Assume Lemma holds for all x **with** $|x| = n$.

I.S.: Consider a word x **with** $|x| = n + 1$.

Then $x = a_1 \dots a_{n+1} = a_1 u$, **where**

$$u = a_2 \dots a_{n+1}.$$

Then $x^R = u^R a_1$.

But $|u| = n$, **by I.H.,** $u^R = a_{n+1} \dots a_2$, **so we get**

$$x^R = a_{n+1} \dots a_1. \; qed$$

△ **Universal Language over** Σ: Σ^*

$\Sigma = \{0,\ 1\}$, $\Sigma^* = \{\lambda,\ 0,\ 1,\ 00,\ 01,\ 10,\ 11,\ 000,\ \dots\}$

△ Mappings (Morphism and Substitution)

Definition Let Σ and Δ be two alphabets. Then a mapping $h : \Sigma^* \to \Delta^*$ is said to be a morphism if

(i) $h(\lambda) = \lambda$**;**

(ii) $h(xy) = h(x)h(y)$**, for all** $x,\ y$ **in** Σ^***.**

Note: (ii) implies that we need only define the images of letters to define the images of words.

Example:

$\Sigma = \{0,\ 1\}, \quad \Delta = \{a,\ b,\ c,\ d\},$

$h(0) = baac, \quad h(1) = \lambda.$

Then $h(1001) = h(1)h(0)h(0)h(1) = \lambda baacbaac\lambda = baacbaac$

A morphism h is said to be λ-free if, for all $x \neq \lambda, \quad h(x) \neq \lambda.$

Definition **Let Σ and Δ be two alphabets. A mapping $\sigma : \Sigma^* \to 2^{\Delta^*}$ is said to be a substitution if**

i) $\sigma(\lambda) = \{\lambda\}$**;**

ii) $\sigma(xy) = \sigma(x)\sigma(y)$**, for all** $x,\ y \in \Sigma^*$**.**

Example:

$\Sigma_1 = \{a,\ b\}, \quad \Delta_1 = \{a,\ b,\ c\},$

a substitution σ_1**:**

$\sigma_1(a) = \{ab,\ ac,\ b\};$

$\sigma_1(b) = \{b,\ ba\}.$

Then $\sigma_1(ba) =$

Example:

$\Sigma_2 = \{0,\ 1,\ 2\}, \quad \Delta_2 = \{a,\ b\},$

$\sigma_2(0) = \{a^i \mid i \geq 1\}$

$\sigma_2(1) = \{b^i \mid i \geq 1\}$

$\sigma_2(2) = \{\lambda\}$

Then $\sigma_2(1020) =$

△ **Given an alphabet** Σ, Σ^* **is defined as follows:**

(i) $\lambda \in \Sigma^*$**;**

(ii) if $x \in \Sigma^*$**, then** $ax \in \Sigma^*$ **for all** $a \in \Sigma$**.**

△ **A language** L **over an alphabet** Σ **is a subset of** Σ^***, i.e.,** $L \subseteq \Sigma^*$**.**

Example:

ϕ **is a language over every alphabet .**

$\{\lambda\}$ **is a language over every alphabet.**

$\{a^i b^i \mid i \geq 0\}$ **is a language over** $\{a,\ b\}$

$\{a^i b^i c^i \mid i \geq 0\}$ **is a language over** $\{a,\ b,\ c\}$

△ Catenation of Languages

$L_1 \circ L_2 = \{x_1 \circ x_2 \mid x_1 \textbf{ in } L_1,\ x_2 \textbf{ in } L_2\}$,
usually written as L_1L_2.

Examples

$L_1 = \{a,\ ab\}; \quad L_2 = \{bc,\ c\}$

$L_1L_2 = \{abc,\ ac,\ abbc\}$

(Note that $L_1 \times L_2 = \{(a, bc), (a, c), (ab, bc), (ab, c)\}$**)**

$\phi L_1 =$

$\{\lambda\}L_1 =$

$\{\lambda\}\phi =$

$L_A = \{a,\ \lambda\}; \quad L_B = \{ab,\ b\}$

$L_AL_B =$

△ **Powers of Languages:**

$L^0 = \{\lambda\}$

$L^{n+1} = L^n \circ L, \ n \geq 1$

△ **Plus**

$L^+ = \cup_{i=1}^{\infty} L^i$

△ **Star**

$L^* = \cup_{i=0}^{\infty} L^i \qquad\qquad L^0 \cup L^1 \cup L^2 \cup \ldots\ldots$

Examples

$L = \{ab, \ b\}$**. Then**

$L^0 = \{\lambda\}$

$L^1 = L = \{ab, \ b\}$

$L^2 = \{abab, \ bab, \ abb, \ bb\}$

$\phi^0 = \{\lambda\} \neq \phi$

$\{\lambda\}^0 = \{\lambda\}$

$L^* = \{\lambda, \ ab, \ b, \ abab, \ bab, \ abb, \ bb, \ \ldots\ldots\}$

(Note $ababbabab \in L^4$**)**

△ Reversal of Languages

$$L^R = \{x^R \mid x \textbf{ is in } L\}$$

Examples

$$L_F = \{aab,\ abb,\ abbb\};$$
$$L_F^R = \{baa,\ bba,\ bbba\}$$
$$L_I = \{a^m b^n \mid m > n > 0\}$$
$$L_I^R = \{b^n a^m \mid m > n > 0\}$$

— **properties of reversal**

$$\begin{aligned}(A \cup B)^R &= A^R \cup B^R \\ (A \cap B)^R &= A^R \cap B^R \\ (AB)^R &= B^R A^R \\ (A^+)^R &= (A^R)^+ \\ (A^*)^R &= (A^R)^*\end{aligned}$$

△ Complementation of Languages

Given a language L over Σ,

$$\overline{L} = \Sigma^* - L$$

$\overline{\Sigma^*} = \phi$ $\qquad$ $\overline{\Sigma^*}\ ?\ \{\lambda\}$ $\qquad$ $(\overline{A})^R = \overline{A^R}$

$L = \{a^n b^n \mid n \geq 0\}$ **over** $\Sigma = \{a,\ b\}$

$\overline{L} =$

△ **Alternative definition of L^+ and L^***

$$L^+ = \{w \text{ in } \Sigma^* \mid w = w_1 w_2 \dots w_n, \text{ for } w_1, w_2, \dots, w_n \in L \text{ and } n \geq 1\}$$

$$L^* = \{w \text{ in } \Sigma^* \mid w = w_1 w_2 \dots w_n, \text{ for } w_1, w_2, \dots, w_n \in L \text{ and } n \geq 0\}$$

△ **Given a word x in Σ^*, and a language $L \subseteq \Sigma^*$ to test if x is in L^* we use $L^* = \cup_{i=0}^{\infty} L^i$ and test if x is in $L^0,\ L^1,\ \dots$.**

△ Property Summaries

— properties of catenation (of languages)

$$\begin{aligned}
\textbf{Associativity} &\text{ — } L_1(L_2L_3) = (L_1L_2)L_3 \\
\textbf{Identity} &\text{ — } L\{\lambda\} = \{\lambda\}L = L \\
\textbf{Zero} &\text{ — } L\phi = \phi L = \phi \\
\textbf{Distributivity} &\text{ — } L_1(L_2 \cup L_3) = L_1L_2 \cup L_1L_3 \\
\textbf{w.r. } \cup &\text{ — } (L_1 \cup L_2)L_3 = L_1L_3 \cup L_2L_3
\end{aligned}$$

Distributivity does not hold with respect to $\cap$

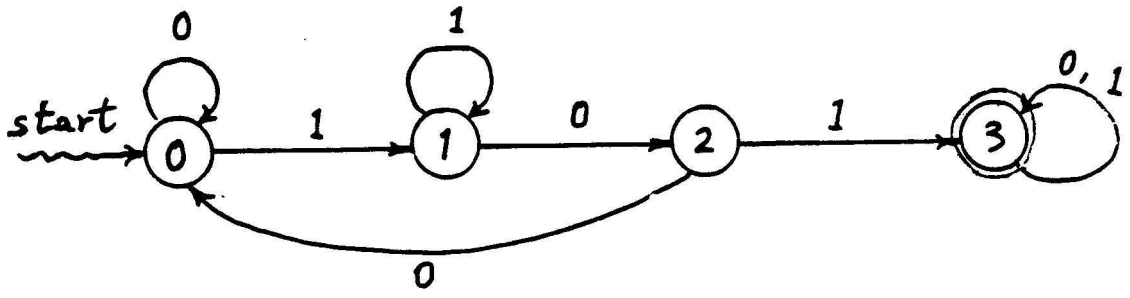
— properties of Plus and Star

$$\begin{aligned}
L^* &= L^+ \cup \{\lambda\} \quad (\textbf{ But } L^+ = L^* - \{\lambda\}?) \\
L^+ &= LL^* \\
(L^+)^+ &= L^+, \quad (L^*)^* = L^* \\
\phi^+ &= \phi, \qquad \phi^* = \{\lambda\}, \quad \phi^0 = \{\lambda\} \\
\{\lambda\}^+ &= \{\lambda\}, \quad \{\lambda\}^* = \{\lambda\}, \quad \phi^2 = \phi \\
L^+L &= LL^+, \quad L^*L = LL^* \\
A \subseteq B &\Rightarrow A^+ \subseteq B^+ \textbf{ and } A^* \subseteq B^*
\end{aligned}$$

Chapter 2. FINITE AUTOMATA

Example Design a "sequential lock". The lock has 1-bit sequential input. Initially the lock is closed. If the lock is closed it will open when the last three input signals are "1", "0", "1", and then remains open.

— state (transition) diagram



— state (or transition) table

Present state	Present symbol 0	1
0	0	1
1	2	1
2	0	3
3	3	3

state set : $\{0,\ 1,\ 2,\ 3\}$
input alphabet : $\{0,\ 1\}$
Transition function : $\delta(0,0) = 0,\ \delta(0,1) = 1,\ \ldots$
Start state : 0
Final state set : $\{3\}$

Deterministic Finite Automata (DFA)

$M = (Q,\ \Sigma,\ \delta,\ s,\ F)$ **where**

Q **is a finite nonempty set of states**

Σ **is the input alphabet**

$\delta : Q \times \Sigma \to Q$ **transition function**

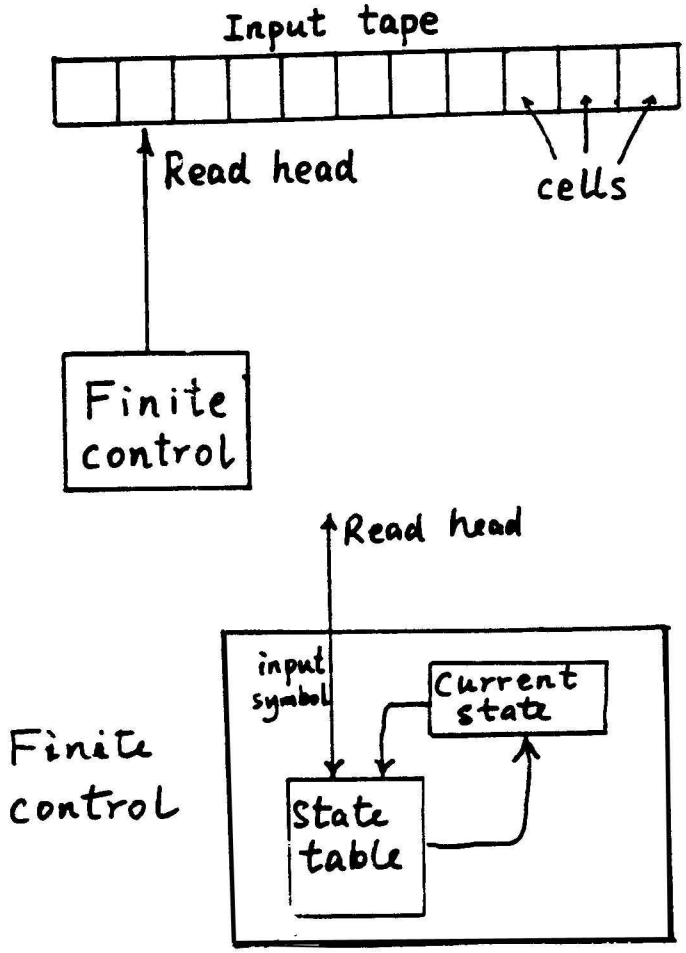
s **start state**

$F \subseteq Q$ **final state set**

A computer is a finite state system (i.e. FA) which has millions of states.

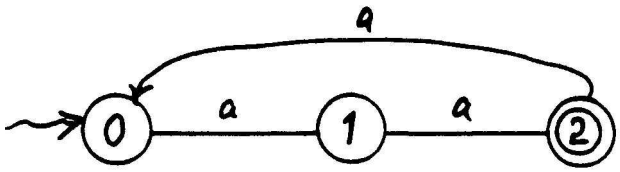
There are many examples of <u>FINITE STATE SYSTEMS</u>. A finite automaton is an <u>ABSTRACTION</u> of them.

<u>View a DFA as a machine</u>



Specifying δ

1)



State diagram
(Transition diagram)

Start state

Final state

2)

Present state	Present symbol a
0	1
1	2
2	0

Configurations

a word in $Q\Sigma^*$

$$px$$

where p **is the present state, and**
x **is the remaining input**

Example:

$0aa \quad \ldots \quad 1a \quad \ldots \quad 2$

(start configuration) (final configuration)

Moves of a DFA

$0aa \vdash 1a$ $\qquad$ $px \vdash qy$

$1a \vdash 2$ $\qquad$ **if** $x = ay$ **and** $\delta(p, a) = q$

Configuration sequence

$0aa \vdash 1a \vdash 2$

$\vdash^+$ **and** $\vdash^*$

$\vdash$ **is a binary relation over** $Q\Sigma^*$**.**

$\vdash^+$: **transitive closure of** $\vdash$**.**

$\vdash^*$: **reflexive transitive closure of** $\vdash$**.**

$$0aa \vdash^+ 2$$
$$0aa \vdash^* 2$$
$$0aa \vdash^* 0aa$$
$$0aa \vdash^2 2$$

$$px \vdash^k qy$$

if $px \underbrace{\vdash p_{i_1}x_{i_1} \vdash p_{i_2}x_{i_2} \vdash \ldots}_{k \textbf{ steps}} \vdash qy$

Accepting Configuration Sequence

$0aa \vdash 1a \vdash 2$

$\vdash$ can also be viewed as a function

$$\vdash : Q\Sigma^* \to Q\Sigma^*,$$

since the next configuration is determined uniquely for a given configuration.

The DFA stops when:

(i) we have no more input,

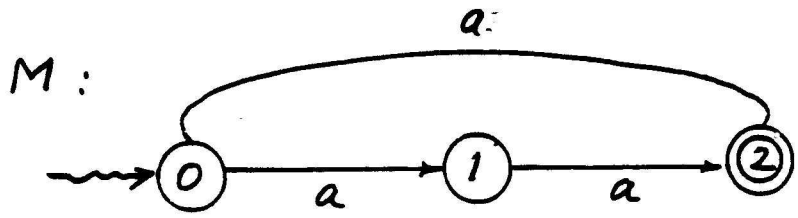
or (ii) the next configuration is undefined.

A word x is said to be accepted by a DFA M if $sx \vdash^* f,\ f \in F$.

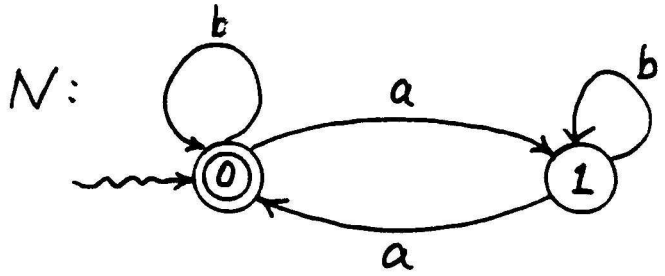
The language of a DFA M, $L(M)$, is defined as:

$$L(M) = \{x \mid sx \vdash^* f, \text{ for some } f \in F\}$$

Examples



$L(M) =$



$L(N) =$

DFA membership problem

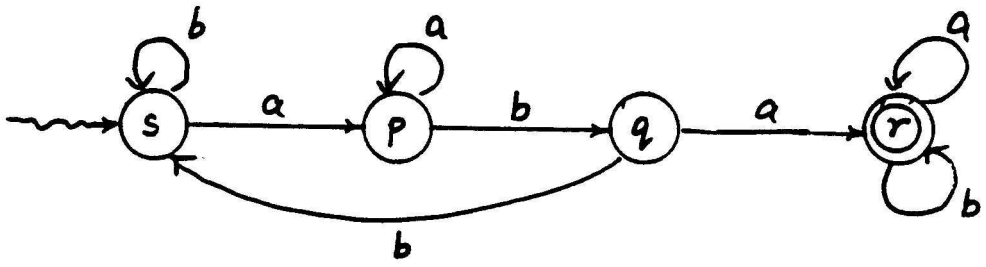
DFA MEMBERSHIP

INSTANCE: A DFA, $M = (Q, \Sigma, \delta, s, F)$
and a word $x \in \Sigma^*$.

QUESTION: Is x **in** $L(M)$ **?**

Run the DFA M **with input** x**.**
In at most $|x|$ **steps it accepts, rejects or aborts.**

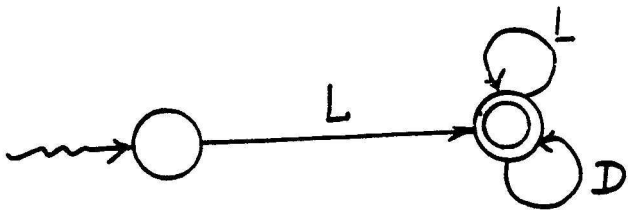
Examples



Checking for words that contain *aba* **as subword.**

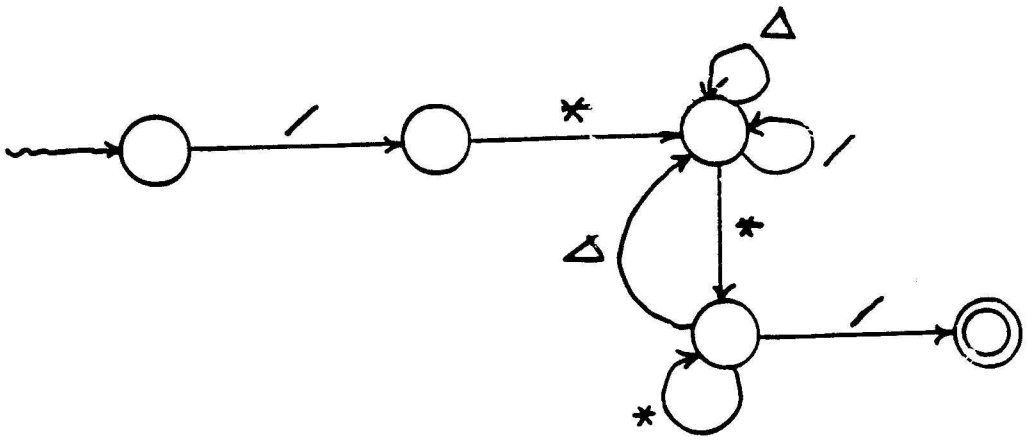
Check: *ababba*
abbaabbbab

Let L denote any letter of English alphabet and D any decimal digit; the form of PASCAL IDENTIFIERS can be specified by



Recognizing comments that may go over several lines.

/*.....*/

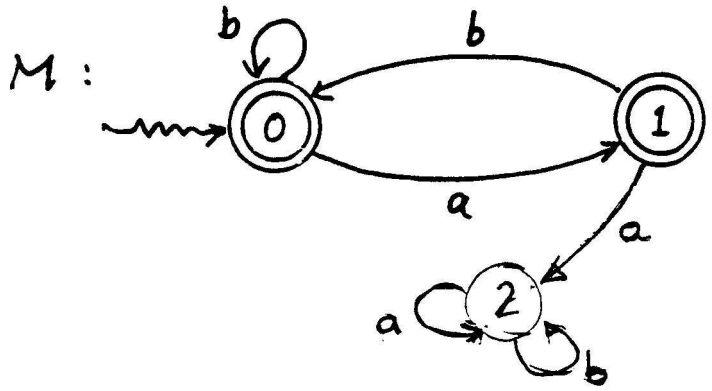


Δ: symbols other than " $*$ " and "/"

A DFA which has a total δ is said to be complete; if δ is nontotal it is incomplete.

Theorem. Every incomplete DFA M can be "completed" by adding one new state ("sink") to give DFA M' such that $L(M') = L(M)$.

Example:



$L(M)$ is the set of all words that do not contain two consecutive a's.

$\triangle$ **Two DFA M_1 and M_2 are equivalent if $L(M_1) = L(M_2)$.**

$\triangle$ **The collection of languages accepted by DFA's is denoted by**

$$\mathcal{L}_{DFA}.$$

It is called the family of DFA languages and it is defined as:

$$\mathcal{L}_{DFA} = \{L \mid L = L(M) \text{ for some DFA } M\}$$

$\triangle$
$K = \{a^i b^i \mid i \geq 1\}$ **is not accepted by any DFA.**

Proof: Use contradiction and Pigeonhole principle.

Assume $K = L(M)$, for some DFA

$$M = (Q, \{a, b\}, \delta, s, F).$$

Let $n = \#Q$. Consider the accepting configuration sequence for $a^n b^n$,

$$s_0 a^n b^n \vdash s_1 a^{n-1} b^n \vdash \ldots \vdash s_n b^n \vdash \ldots \vdash s_{2n}$$

where $s_0 = s$ and $s_{2n} \in F$. Now $n + 1$ states appear during the reading of a^n, but there are only n distinct states in Q. By Pigeonhole principle at least one state must appear at least twice during the reading of a's.

Assume $s_i = s_j, 0 \leq i < j \leq n$.
Then

$$s_0 a^{n-(j-i)} b^n \vdash \ldots \vdash s_i a^{n-j} b^n$$
$$s_j a^{n-j} b^n \vdash \ldots \vdash s_n b^n \vdash \ldots \vdash$$
$$\vdash s_{2n}$$

Therefore $a^{n-(j-i)} b^n \in K$.

This is a contradiction.

$\triangle$ $L_i = \{a^i b^i\}, i \geq 1.$

For any $i \geq 1$, is L_i a DFA language?

$\triangle$ $K_j = \{a^i b^i : 0 \leq i \leq j\}, j \geq 1.$

For any $j \geq 1$, is K_j a DFA language ?

Nondeterministic Finite Automata (NFA)

$M = (Q, \Sigma, \delta, s, F)$

same as a DFA except
$\delta \subseteq Q \times \Sigma \times Q$.

δ is a finite transition relation.

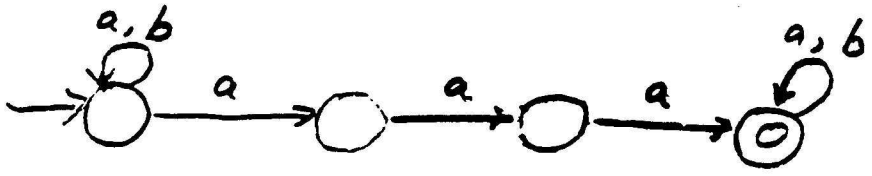
In a DFA
δ is a transition function:
$\delta : Q \times \Sigma \to Q$

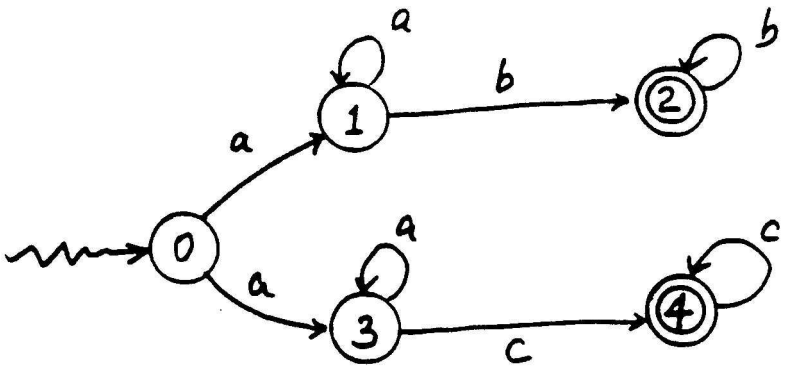
It can be viewed as a relation
$\delta : Q \times \Sigma \times Q$

In a NFA, δ can be be viewed as a function:
$\delta : Q \times \Sigma \to 2^Q$

Examples:

NFA for words in $\{a, b\}^*$ that contain three consecutive a's.





Both $(0, a, 1)$ and $(0, a, 3)$ are in δ.

We define acceptance by existence of a computation that leads to a final state.

Conversely, we define rejection by the nonexistence of any computation that leads to a final state.

The language of an NFA $M = (Q, \Sigma, \delta, s, F)$ is defined by

$L(M) = \{x \mid sx \vdash^* f$**, for some** f **in** F $\}$**.**

The family of NFA languages $\mathcal{L}_{NFA}$

is defined by:

$\mathcal{L}_{NFA} = \{L \mid L = L(M)$, **for some NFA** $M\ \}$.

Two NFAs M_1, and M_2 are equivalent if $L(M_1) = L(M_2)$.

Why NFA?

(i) easy to construct;
(ii) useful theoretically;
(iii) are of same power as DFA.

Note:

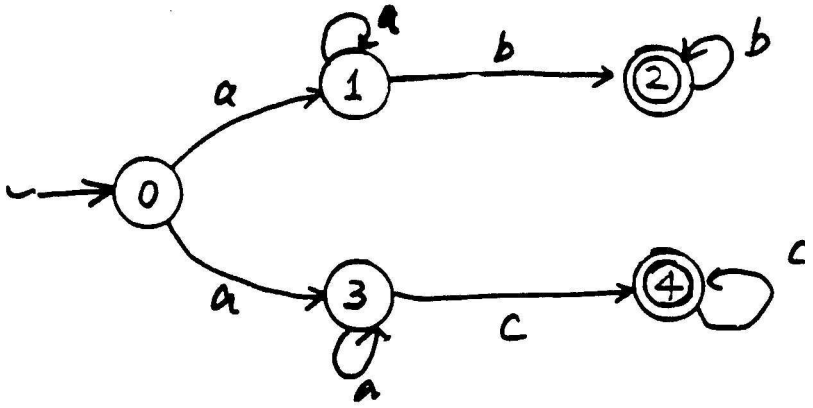
configurations are defined in the same way
Transition (move)

$$px \vdash qy$$

if $x = ay$, for some $a \in \Sigma$, and $(p, a, q) \in \delta$.

Transforming NFA to DFA

Consider the NFA M_1 again



There are only limited number of choices. For example:

$0\underline{a}ab \vdash 1ab \vdash 1b \vdash 2$

$0aab \vdash 3ab \vdash 3b$

$\{0\}aab \vdash \{1,3\}ab \vdash \{1,3\}b \vdash \{2\}$

Why limited number of choices?

The state set is finite.

We summarize the choices at each step by combining all configuration sequences into one "super-conf. sequence".

$\{0\}aab \vdash \{1,3\}ab \vdash \{1,3\}b \vdash \{2\}$.

We now have a set of all possible states at each step. From this point of view the computation of the NFA on an input word is deterministic.

A super-configuration has the form

$$Kx$$

where $K \subseteq Q$ **and** $x \in \Sigma^*$**.**

Note that $\emptyset x$ **is a super-conf., it means that the NFA cannot be in any state at that point, i.e., an abort has occured.**

We say that

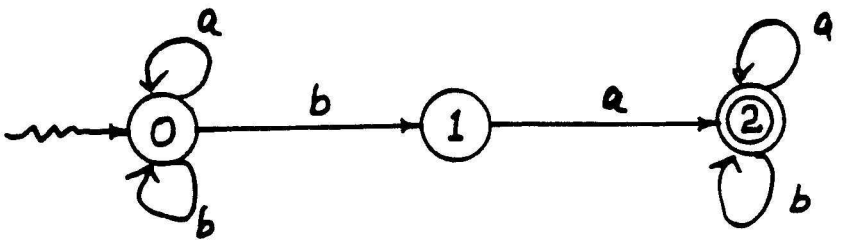
$$Kx \vdash Ny$$

if $x = ay$**, for some** $a \in \Sigma$**, and**

$N = \{q \mid (p, a, q) \in \delta$**, for some** $p \in K\}$

More examples on super-configurations

M: $L(M)$ is the set of all words that have "ba" as a subword.



The super-configuration sequence on input word "$abbaa$" is as follows:

$\{0\}abbaa \vdash \{0\}bbaa \vdash \{0,1\}baa \vdash \{0,1\}aa$
$\vdash \{0,2\}a \vdash \{0,2\}$

Notice that given a set $K \subseteq Q$ and an input symbol $a \in \Sigma$, the set $N \subseteq Q$ s.t. $Ka \vdash N$ is uniquely determined.

Lemma (2.3.1) (Determinism Lemma) Let $M = (Q, \Sigma, \delta, s, F)$ be an NFA. Then for all words x in Σ^* and for all $K \subseteq Q$.

$Kx \vdash^* N$ **and** $Kx \vdash^* P$

implies

$P = N$**.**

Lemma (2.3.2) Let $M = (Q, \Sigma, \delta, s, F)$ be an NFA. Then for all words x in Σ^* and for all q in Q,

$qx \vdash^* p$

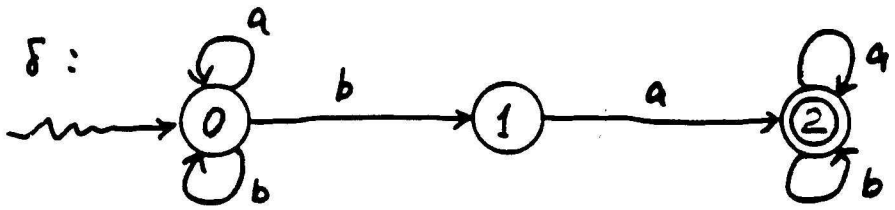
iff $\{q\}x \vdash^* P$, for some P with p in P.

Example (Transformation of an NFA to a DFA)

$M = (Q, \Sigma, \delta, s, F)$ **where**

$Q = 0, 1, 2, \qquad \Sigma = a, b$

$s = 0, \qquad F = \{2\}$



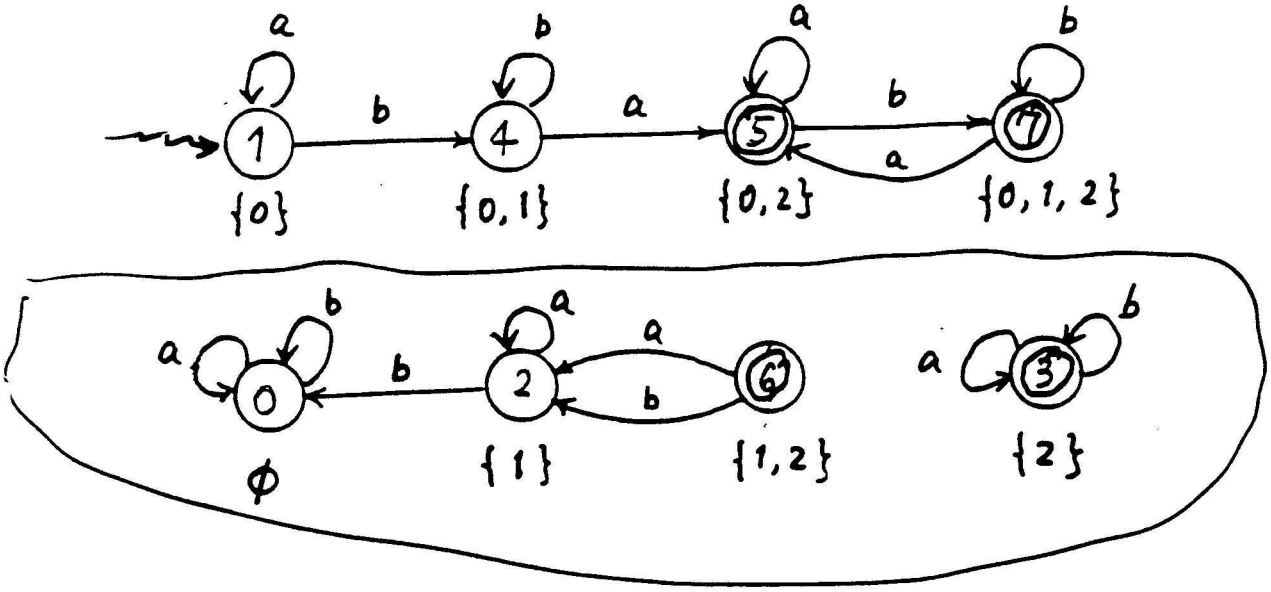
$M' = (Q', \Sigma, \delta', s', F')$ **where**

$Q' = 2^Q = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0,1\}, \{0,2\}, \{1,2\}, \{0,1,2\}\}$

δ':

	current state	a	b
0	ϕ	ϕ	ϕ
1	$\{0\}$	$\{0\}$	$\{0,1\}$
2	$\{1\}$	$\{2\}$	ϕ
3	$\{2\}$	$\{2\}$	$\{2\}$
4	$\{0,1\}$	$\{0,2\}$	$\{0,1\}$
5	$\{0,2\}$	$\{0,2\}$	$\{0,1,2\}$
6	$\{1,2\}$	$\{2\}$	$\{2\}$
7	$\{0,1,2\}$	$\{0,2\}$	$\{0,1,2\}$

$\delta'(P, a) = \{q \mid (p, a, q) \in \delta \text{ and } p \in P\}$



$s' = \{0\}$

$F' = \{$

Algorithm NFA to DFA

—The Subset Construction

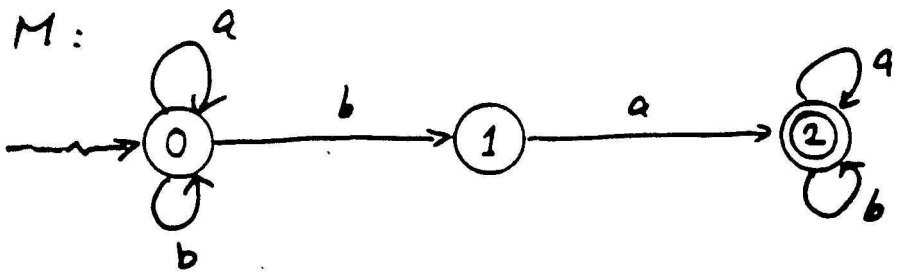
On entry: An NFA $M = (Q, \Sigma, \delta, s, F)$.

On exit: A DFA $M' = (Q', \Sigma, \delta', s', F')$
satisfying $L(M) = L(M')$**.**

begin Let $Q' = 2^Q, s' = \{s\}$ **and**
$F' = \{K \mid K \in Q',$ **and** $K \cap F \neq \emptyset\}$
We define $\delta' : Q' \times \Sigma \to Q'$ **by**
For all $K \in Q'$ **and for all** $a \in \Sigma$**,**
$\delta'(K, a) = N$**, if** $Ka \vdash N$ **in** M**.**

end of Algorithm

if $N = \{q \mid (p, a, q) \in \delta$ **and** $p \in K\}$



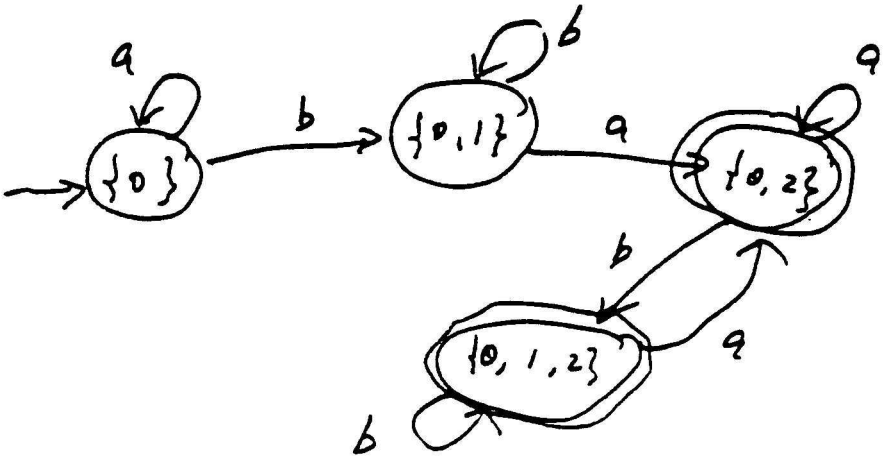
$s' = \{0\}$

input symbol / current state	a	b
{0}	{0}	{0, 1}
{0, 1}	{0, 2}	{0, 1}
{0, 2}	{0, 2}	{0, 1, 2}
{0, 1, 2}	{0, 2}	{0, 1, 2}

	a	b
0	{0}	{0, 1}
1	{2}	ϕ
2	{2}	{2}

Algorithm NFA to DFA 2

—The Iterative Subset Construction



Theorem **Given an NFA $M = (Q, \Sigma, \delta, s, F)$, then the DFA $M' = (Q', \Sigma', \delta', s', F')$ obtained by either subset construction satisfies $L(M') = L(M)$.**

Proof:

By Lemma 2.3.2, for all $x \in \Sigma^*$ in M

$sx \vdash^* p$, **iff** $\{s\}x \vdash^* P$ **for some** P **with** $p \in P$

By the construction of M',
$\{s\}x \vdash^* P$ **in** M **iff**
$\{s\}x \vdash^* P$ **in** M'**.**

$$
\begin{aligned}
x \in L(M) &\Leftrightarrow sx \vdash^* f, \text{ for some } f \in F \\
&\Leftrightarrow \{s\}x \vdash^* P,\ f \in P, \text{ in } M \\
&\Leftrightarrow \{s\}x \vdash^* P, \text{ in } M' \text{ and } P \cap F \neq \emptyset \\
&\Leftrightarrow s'x \vdash^* P,\ P \in F \\
&\Leftrightarrow x \in L(M')
\end{aligned}
$$

Theorem

Every NFA Language is a DFA language and conversely.
($\mathcal{L}_{NFA} = \mathcal{L}_{DFA}$)

Example

Every finite language is accepted by a DFA.

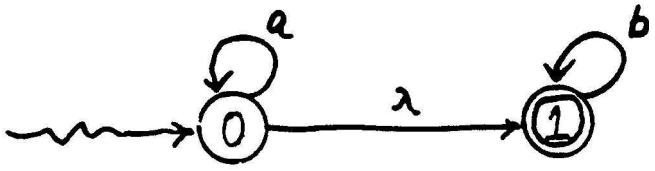
λ–NFA

It is useful to loosen the definition of NFA even more by allowing the read head to remain over the same symbol of the input and read nothing.

Example

$L_1 = \{a^i b^j \mid i \geq 0,\ j \geq 0\}$

M:



$$L(M) = L_1$$

$$\begin{aligned}
\delta: \quad & (0, a, 0) \\
& \underline{(0, \lambda, 1)} \qquad \lambda - transition \\
& (1, b, 1) \\
0aab \quad & \vdash \ 0ab \ \vdash \ 0b \ \vdash \ 1b \ \vdash \ 1 \\
0bb \quad & \vdash \ 1bb \ \vdash \ 1b \ \vdash \ 1 \\
0a \quad & \vdash \ 0 \ \vdash \ 1
\end{aligned}$$

Formally, a λ-NFA $M = (Q, \Sigma, \delta, s, F)$ where $Q,\ \Sigma,\ s,\ F$ are as before, but δ is a finite transition relation for which

$$\delta \subseteq Q \times (\Sigma \cup \{\lambda\}) \times Q$$

Configurations are as before.
$\vdash$ is defined by

$$px \vdash qy$$

if either $x = ay$ for $a \in \Sigma$ and $(p, a, q) \in \delta$
or $x = y$ and $(p, \lambda, q) \in \delta$

Example

Given FA M_1 and M_2, construct a FA M_3 such that
$L(M_3) = L(M_1) \cup L(M_2)$

Transforming λ-NFA to NFA

Two steps:
Step I: $\lambda-$ completion

Step II: $\lambda-$ transition removal

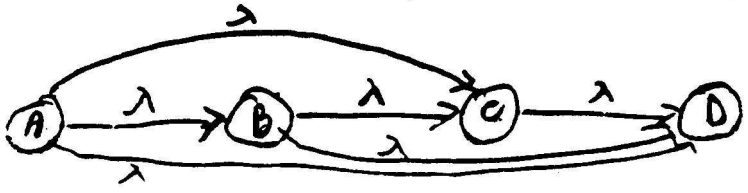
(I). λ-Completion

Given a λ-NFA $M = (Q, \Sigma, \delta, s, F)$
perform the following process:

> **For all $p, q, r \in Q$:**
> **whenever $(p, \lambda, q), (q, \lambda, r)$ are in δ**
> **add (p, λ, r) to δ**
> **until no new transitions are added to δ**
> **and let this be δ'.**

Let the new $\lambda-$NFA be
$M' = (Q, \Sigma, \delta', s, F')$
where $F' = F \cup \{p \mid (p, \lambda, f) \in \delta$ and $f \in F\}$
and $\delta' = \delta \cup \{(p, \lambda, q) \mid p \vdash^+ q\}$

Example:



Claim 1: For any $p,\ q \in Q$**,**

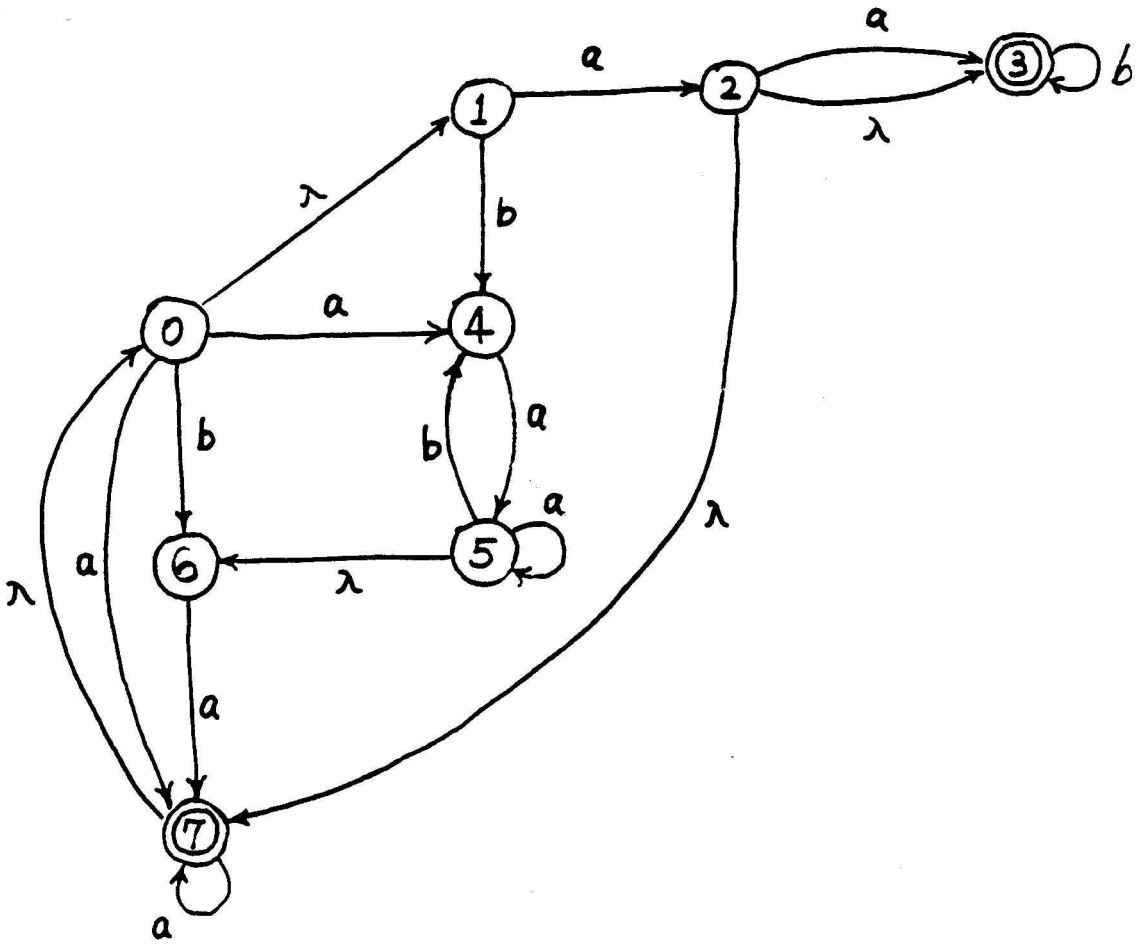
$$p \vdash_M^+ q \text{ if and only if } p \vdash_{M'} q$$

Claim 2: For any $p,\ q \in Q,\ x \in \Sigma^*$**,**

$$px \vdash_M^* q \text{ if and only if } px \vdash_{M'}^* q$$

Theorem: $L(M') = L(M)$

Example:



(II) λ–Transition Removal

Given a λ-completed λ-NFA

$$M = (Q,\ \Sigma,\ \delta,\ s,\ F),$$

perform the following process:

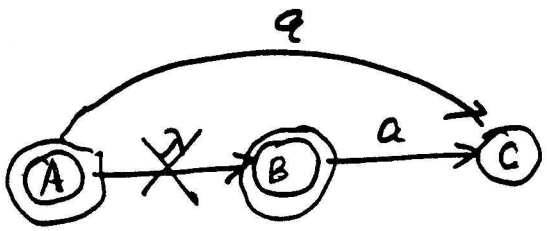
(0) $\delta' = \delta$;

(i) **For all** $p, q, r \in Q$,
if (p, λ, q) **and** (q, a, r) **in** δ
then add (p, a, r) **to** δ';

(ii) **Delete all λ-transitions from** δ'.

Now we got $M' = (Q,\ \Sigma,\ \delta',\ s,\ F)$
where $\delta' = (\delta \cup \{(p, a, r) \mid (p, \lambda, q), (q, a, r) \in \delta\})$
$-\{(p, \lambda, q) \mid p, q \in Q\}$

Example



Claim **Whenever**

$$sx \vdash_M^* f$$

for some $f \in F$**, we have**

$$sx \vdash_{M'}^* f$$

and vice versa.

Claim $L(M') = L(M)$

Theorem

$$\mathcal{L}_{\lambda-NFA} = \mathcal{L}_{NFA}$$

Properties of FA Languages

I. Closure Properties

Union: The union of two DFA languages is a DFA language.

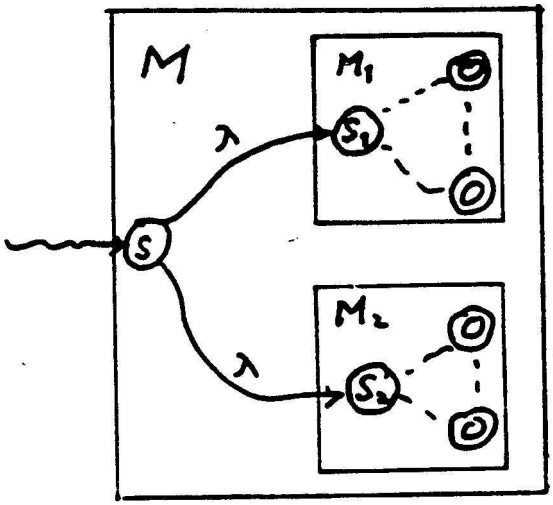
Proof: Given $L_1 = L(M_1)$ and $L_2 = L(M_2)$, $L_1,\ L_2$ are generic DFA languages.

$$M_1 = (Q_1,\ \Sigma_1,\ \delta_1,\ s_1,\ F_1) \text{ and}$$

$$M_2 = (Q_2,\ \Sigma_2,\ \delta_2,\ s_2,\ F_2).$$

Construct a λ-NFA $M = (Q,\ \Sigma,\ \delta,\ s,\ F)$ where

$$\begin{aligned} Q &= Q_1 \cup Q_2 \cup \{s\} \\ \Sigma &= \Sigma_1 \cup \Sigma_2 \\ \delta &= \delta_1 \cup \delta_2 \cup \{(s, \lambda, s_1), (s, \lambda, s_2)\} \\ F &= F_1 \cup F_2 \end{aligned}$$



Now we claim that $L(M) = L_1 \cup L_2$

(1) Let x be a generic word in $L(M)$.
$x \in L(M)$ **implies**
$sx \vdash_M^+ f$ **for some** $f \in F_1$ **or** $f \in F_2$.
If $f \in F_1$, **then** $sx \vdash_M s_1x \vdash_M^* f$ **and then** $s_1x \vdash_{M_1}^* f$. **We know that** $x \in L(M_1)$.
If $f \in F_2$, **then** $sx \vdash_M s_2x \vdash_M^* f$. **Since** $s_2x \vdash_{M_2}^* f$, **we know that** $x \in L(M_2)$, **therefore** $x \in L(M_1) \cup L(M_2)$, **i.e.,** $x \in L_1 \cup L_2$. **So** $L(M) \subseteq L_1 \cup L_2$.

(2) Let x be a generic word in $L_1 \cup L_2$.
Without losing of generality we can assume that $x \in L_1$, **i.e.,** $x \in L(M_1)$.
Then $s_1x \vdash_{M_1}^* f$, **for some** $f \in F_1$.
Then $sx \vdash_M s_1x \vdash_M^* f$. **Therefore** $x \in L(M)$.
So, $L_1 \cup L_2 \subseteq L(M)$.

(3) By the results of (1) and (2), we conclude $L(M) = L_1 \cup L_2$.

Since every λ-NFA language is a DFA language, $L(M)$ is a DFA language.

2. **Complementation:**

   The complement of a DFA language $L \subseteq \Sigma^*$ is a DFA language.
   Proof: There is a complete DFA $M = (Q,\ \Sigma,\ \delta,\ s,\ F)$ **with** $L = L(M)$**. Define** $\overline{M} = (Q,\ \Sigma,\ \delta,\ s,\ Q - F)$

3. **Intersection:**

$$\begin{array}{ccccc} & & L_1 \text{ and } L_2 & & \\ L & = & L_1 \cap L_2 & = & \overline{\overline{L_1} \cup \overline{L_2}} \end{array}$$

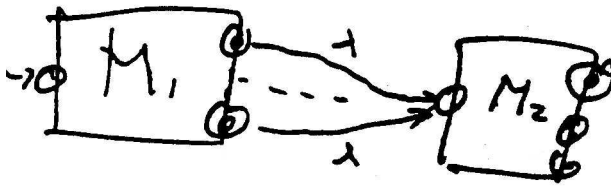
4. Catenation:

The catenation of two DFA languages is a DFA language:

$$M_1 = (Q_1,\ \Sigma_1,\ \delta_1,\ s_1,\ F_1);$$
$$M_2 = (Q_2,\ \Sigma_2,\ \delta_2,\ s_2,\ F_2).$$

$$L(M) \ = \ L(M_1)L(M_2)$$

$$M = (Q_1 \cup Q_2,\ \Sigma_1 \cup \Sigma_2,\ \delta,\ s_1,\ F_2),$$
$$\delta = \delta_1 \cup \delta_2 \cup \{(f, \lambda, s_2) \mid f \in F_1\}$$



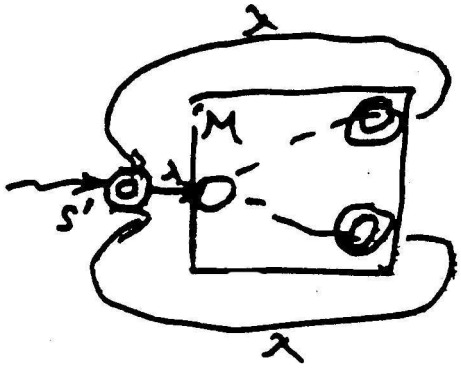
$M = (Q,\ \Sigma,\ \delta,\ s,\ F)$ **where**

$$\begin{aligned} Q &= Q_1 \cup Q_2 \\ \Sigma &= \Sigma_1 \cup \Sigma_2 \\ \delta &= \delta_1 \cup \delta_2 \cup \{(f, \lambda, s_2) \mid f \in F_1\} \\ s &= s_1 \\ F &= F_2 \end{aligned}$$

5. Star:

The star of a DFA language is a DFA language

$L = L(M)$ **where** $M = (Q,\ \Sigma,\ \delta,\ s,\ F)$

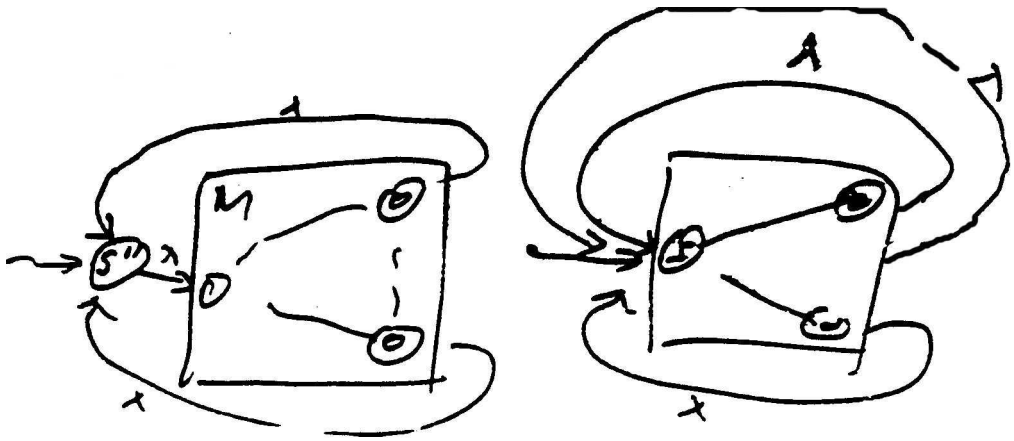


$L^* = L(M')$

$M' = (Q',\ \Sigma',\ \delta',\ s',\ F')$ **where**

$$
\begin{aligned}
Q' &= Q \cup \{s'\} \\
\Sigma' &= \Sigma \\
\delta' &= \delta \cup \{(s', \lambda, s)\} \cup \{(f, \lambda, s') \mid f \in F\} \\
F' &= F \cup \{s'\} \quad F' = \{s'\}
\end{aligned}
$$

6. Plus:



II. Decidability Properties

A decision problem is a problem each instance of which is either false or true.

A decision algorithm is an algorithm whose result for each possible input is either false or true.

A decision problem is decidable if there exists a decision algorithm for it. Otherwise it is undecidable.

1. DFA membership

INSTANCE: A DFA $M = (Q,\ \Sigma,\ \delta,\ s,\ F)$ and a word $x \in \Sigma^*$.

QUESTION: Is $x \in L(M)$?

Theorem: DFA membership is decidable.

Proof: Compute for the given M and x the terminating state q such that $sx \vdash^* q$. If $q \in F$ then answer "True" else answer "false".

2. DFA Emptiness

INSTANCE: A DFA $M = (Q,\ \Sigma,\ \delta,\ s,\ F)$.
QUESTION: Is $L(M) = \emptyset$?

Theorem DFA emptiness is decidable.

Proof: $L(M) = \emptyset$ iff there is no path in the state diagram of M from s to a final state. If $F = \emptyset$, then this holds immediately.
Otherwise, enumerate the states that can be reached from s. (Mark s. Now mark all states reachable by one transition from one of the marked states. Repeat this until no newly marked state is introduced.
The marked states are the reachable states).

3. DFA Universality

INSTANCE: A DFA $M = (Q,\ \Sigma,\ \delta,\ s,\ F)$.
QUESTION: Is $L(M) = \Sigma^*$?

Theorem DFA universality is decidable.
Proof:

4. DFA Containment

INSTANCE: Two DFA
$$M_1 = (Q_1,\ \Sigma_1,\ delta_1,\ s_1,\ F_1) \text{ and}$$
$$M_2 = (Q_2,\ \Sigma_2,\ delta_2,\ s_2,\ F_2).$$
QUESTION: Is $L(M_1) \subseteq L(M_2)$?

Theorem DFA containment is decidable.
Proof:

$L(M_1) \cap \overline{L(M_2)} = \emptyset$ **means that** $L(M_1) \subseteq L(M_2)$

5. DFA Equivalence

INSTANCE: Two DFA
$M_1 = (Q_1,\ \Sigma_1,\ delta_1,\ s_1,\ F_1)$ **and**
$M_2 = (Q_2,\ \Sigma_2,\ delta_2,\ s_2,\ F_2)$**.**
QUESTION: Is $L(M_1) = L(M_2)$ **?**

Theorem DFA equivalence is decidable.

$$L(M_1) \subseteq L(M_2) \quad \textbf{?} \quad \textbf{yes}$$
$$L(M_2) \subseteq L(M_1) \quad \textbf{?} \quad \textbf{yes}$$

Pumping Lemma and Non-DFA Language.

The DFA Pumping Lemma

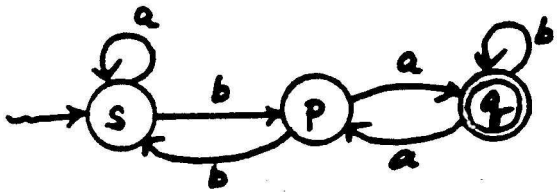
Let $M = (Q,\ \Sigma,\ \delta,\ s,\ F)$ be a DFA and let $p = \#Q$. For all words x in $L(M)$ such that $|x| \geq p$, x can be decomposed into uvw, for some u, v, and w in Σ^* such that

(i) $|uv| \leq p$;

(ii) $|v| \geq 1$; **and**

(iii) **for all** $i \geq 0,\ uv^iw$ **is in** $L(M)$.

Example $M = (Q, \Sigma, \delta, s, F)$ **where**
$Q = \{s,\ p,\ q\}, \quad \Sigma = \{a,\ b\},$
$F = \{q\}. \qquad \#Q = 3$



Let x be an arbitrary word of length $\geq \#Q$ in $L(M)$, e.g. $x = baaa$. The accepting conf. sequence for x is:
$\underline{s}baaa \vdash paaa \vdash \underline{q}aa \vdash pa \vdash \underline{q}$.
So, $u = b,\ v = aa,\ w = a$ and $b(aa)^ia \in L(M)$ for all $i \geq 0$.

Proof of DFA Pumping Lemma:

Let x be in $L(M)$ with $|x| \geq p$.
Then M has an accepting configuration sequence

$$q_0 a_1 a_2 \ldots a_r \vdash q_1 a_2 \ldots a_r \vdash \ldots \vdash q_{r-1} a_r \vdash q_r$$

where $q_0 = s,\ q_r \in F$.
Consider the first p transitions. $s,\ q_1,\ \ldots,\ q_p$ cannot all be distinct since there are only p distinct states (by pigeonhole principle).
This means that $q_i = q_j$ for some $0 \leq i < j \leq p$.
Let

$$u = a_1 \ldots a_i,$$

$$v = a_{i+1} \ldots a_j,$$

$$w = a_{j+1} \ldots a_r.$$

So, we have $q_0 u \vdash^* q_i$
$q_i v \vdash^* q_j \quad (q_i = q_j)$
$q_j w \vdash^* q_r$**.**
Since $q_i = q_j$, $q_i v^k \vdash^* q_j$ for any $k \geq 0$

(i) $|uv| \leq p,$ **since** $j \leq p$

(ii) $|v| \geq 1,$ **since** $i < j$

(iii) $uv^k w \in L(M)$ **for all** $k \geq 0,$ **since**
$q_0 uv^k w \vdash^* q_i v^k w \vdash^* q_j w \vdash q_r \in F$

qed.

Re-state Pumping Lemma
If L **is a DFA Language**
then
($L = L(M)$ **for some** M **of** p **states;)**
for every word x **of length** $\geq p$ **in** L**,**
there exists one decomposition $x = uvw$
which satisfies

(i) $|uv| \leq p,$

(ii) $|v| \geq 1,$

(iii) $uv^k w \in L(M)$ **for all** $k \geq 0.$

Comments: The DFA pumping lemma shows a property of DFA languages. It can be used possitively; but it is mainly used negatively to show that some languages are not in $\mathcal{L}_{DFA}$. The DFA pumping lemma gives a necessary condition for DFA languages. The condition is not sufficient.

Review of logic:

if A then $B \Leftrightarrow$ if $\neg B$ then $\neg A$

if A then $B \not\Leftrightarrow$ if B then A

if A then $B \not\Leftrightarrow$ if $\neg A$ then $\neg B$

Example

if it is sugar then it is sweet.

if it is not sweet then it is not sugar.

if it is sweet then it is sugar.

So, pumping lemma can be used to show that a language is not a DFA language, but cannot be used to show that a language is a DFA language.

Non-DFA Language

$L = \{a^i b^i \mid i \geq 0\}$ is not a DFA Language

Proof (I):

Assume L is a DFA language. Then $L = L(M)$ for some DFA $M = (Q,\ \Sigma,\ \delta,\ s,\ F)$ with $|Q| = p$.
Consider $x = a^t b^t,\ |x| \geq p$.
By DFA pumping lemma, there is a decomposition

$$x = uvw$$

which satisfies (i), (ii), and (iii).
Consider all the possible decompositions

$$\begin{aligned}
&\textbf{Case 1}: && v = a^k,\ k \geq 1.\ k \leq t \\
& && \textbf{But } uv^0w = a^{t-k}b^t \notin L. \\
&\textbf{Case 2}: && v = b^k,\ k \geq 1. \\
& && uv^0w = a^t b^{t-k} \notin L. \\
&\textbf{Case 3}: && v = a^k b^l,\ k, l \geq 1. \\
& && \textbf{Then } uv^2w = a^t b^l a^k b^t \notin L.
\end{aligned}$$

All the possible decompositions fail.
Therefore, L is not a DFA language.

Proof (II):

. . . . . .

$|Q| = p$.

Consider $x = a^p b^p$ in L. Obviously, $|x| \geq p$.

By DFA pumping lemma, there is a decomposition

$$x = uvw$$

that satisfies (i), (ii), and (iii).

The only decompositions that satisfy (i) and (ii) are the following:

$$v = a^k \text{ for } p \geq k \geq 1.$$

But $uv^0w = a^{p-k}b^p \notin L$.

So, there is no such decomposition.

L is not a DFA language.

(i) $|uv| \leq p$;

(ii) $|v| \geq 1$;

Notes on proving that a language is not a DFA language by pumping lemma

1) **Assume L is a DFA language.**
   Then we have a constant p.
2) **Choose a word in** L **of length** $\geq p$.
3) **Consider all the possible decompositions of** x. **If none of them satisfy (i), (ii), and (iii) at the same time, then conclude that such decomposition does not exist.**

So, L **is not a DFA language.**

$L = \{a^{2^n} \mid n \geq 0\}$ **is not a DFA language.**

Proof:

(1) Assume $L = L(M)$ **for some** $M = (Q, \Sigma, \delta, s, F)$ **with** $|Q| = p$**.**
(2) Consider $x = a^{2^{p+1}},\ |x| \geq p.$
(3) By DFA P.L., there is a decomposition

$$x = uvw$$

satisfying (i), (ii), and (iii).
The only decompositions of x **which satisfy (i) and (ii) are the following:**

$$v = a^k, \quad p \geq k \geq 1.$$

Notice that $uw = a^{2^{p+1}-k}$**.**
$2^{p+1} > 2^{p+1} - k \geq 2^{p+1} - p > 2^{p+1} - 2^p = 2^p$
i.e., $2^{p+1} > 2^{p+1} - k > 2^p$.

$$|a^{2^{p+1}}| > |uw| > |a^{2^p}|.$$

So, $uw \notin L$**. It is contrary to (iii), so** L **is not a DFA language.**

6. DFA Finiteness

INSTANCE: A DFA $M = (Q, \Sigma, \delta, s, F)$
QUESTION: Is $L(M)$ finite?
It is decidable.

Theorem $L(M)$ is finite iff
M accepts no words x that satisfy

$$|Q| \leq |x| < 2|Q|$$

Proof: If: Assume M accepts no words x with $|Q| \leq |x| < 2|Q|$ and $L(M)$ is infinite. Since $L(M)$ is infinite, it must accept some word x with $|x| \geq 2|Q|$. By P.L., $x = uvw$ with $|Q| \geq |v| \geq 1$, and $x' = uw$ is in $L(M)$. Notice that $|x| > |x'| \geq |x| - |Q|$. If $|x'| \geq 2|Q|$, then repeat this process.
Otherwise $2|Q| > |x'| \geq |Q|$. (i.e., $2|Q| - |Q|$).
This is a contradiction.

Only if: By P.L., $x = uvw,\ |v| \geq 1$.
Since $uv^iw \in L(M)$ for all $i \geq 0$, $L(M)$ is infinite.

Theorem **The family of DFA languages is closed under morphisms.**

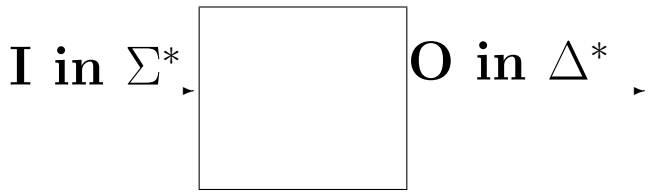
Proof:

Let $L \subseteq \Sigma^*$ be a DFA language. Then there is a DFA $M = (Q, \Sigma, \delta, s, F)$ such that $L(M) = L$.
Let $f : \Sigma^* \to \Delta^*$ be a morphism.
We are going to show that
$f(L) = \{f(x) \mid x \in L\}$ is a DFA language.
Construct $M' = (Q, \Delta, \delta', s, F)$ by defining $(p, f(a), q)$ is in δ' iff $\delta(p, a) = q$.

M' is a lazy FA. So, L(M') is a DFA language.
It is not difficult to show that $L(M') = f(L)$.
Therefore, $f(L)$ is a DFA language. qed

Finite Transducers and Finite Transductions (Translations)

In automata and language theory, a machine with input and output is called a transducer.



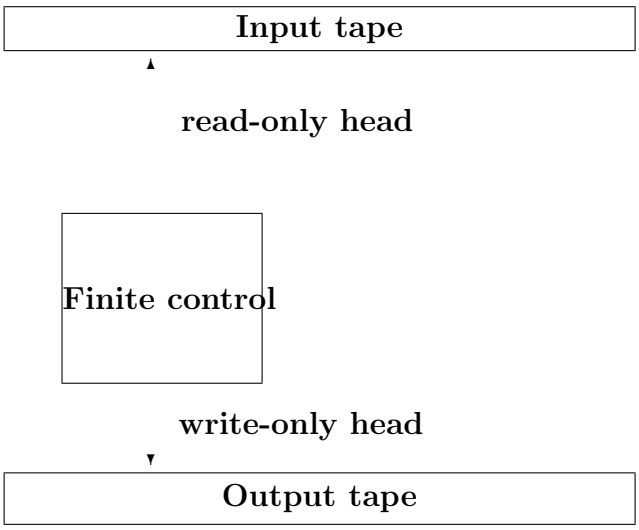
A transducer is single-valued if it produces at most one output word for each input word. It is multi-valued otherwise.

A transducer defines
— a function $f : \Sigma^* \to \Delta^*$, if it is single-valued,
—a relation $R \subseteq \Sigma^* \times \Delta^*$, otherwise.

A function or a relation that is defined by a transducer is called a transduction (translation).

Specifically, a finite automaton with output is called a finite transducer.

FA(DFA, NFA, or λ-NFA)
+ output (output alphabet, output tape)
= FT



Formally, a finite transducer (FT) M is a six-tuple $(Q, \Sigma, \Delta, \delta, s, F)$ where
Δ is an output alphabet;
$\delta \subseteq Q \times (\Sigma \cup \{\lambda\}) \times Q \times \Delta^*$ is a finite transition relation;
the others are the same as in a FA.

A configuration vpu is a word in $\Delta^* Q \Sigma^*$.

v — **output till now;**
p — **current state;**
u — **remaining input.**

We write

$$vpau \vdash vzqu$$

if (p, a, q, z) is in δ.

$\vdash^i$, $\vdash^+$, $\vdash^*$ **are defined as usual.**

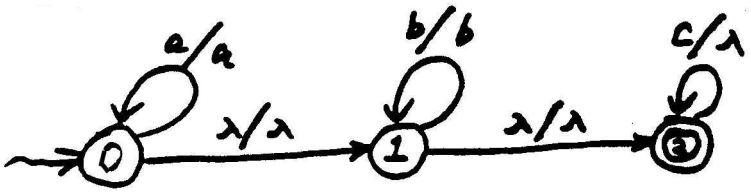
Given $x \in \Sigma^*$, we say that y is an output for x w.r.t. M if

$$sx \vdash^* yf$$

for some $f \in F$.

Let $M(x)$ be the set of all outputs for x, then $M(x) = \{y \mid sx \vdash^* yf$ for some $f \in F\}$.
The transduction or translation defined by M is $T(M) = \{(x, y) \mid x \in \Sigma^*$ and $y \in M(x)\}$

Example M**:**



$0abbc \vdash a0bbc \vdash a1bbc \vdash ab1bc$
$\vdash abb1c \vdash abb2c \vdash abb2$

i.e. $0abbc \vdash^* abb2$

Since $2 \in F$**, we say** $abb \in M(abbc)$**,**
and $(abbc, abb) \in T(M)$**.**

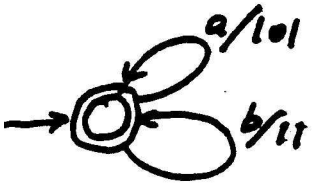
We can see that

$$T(M) = \{(a^i b^j c^k, a^i b^j) \mid i, j, k \geq 0\}$$

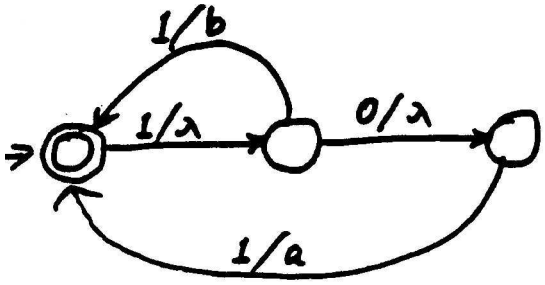
Example

Let $h(a) = 101$**,** $h(b) = 11$**.**

Encoding:



Decoding:



REGULAR EXPRESSIONS

— The Second Model for Defining Languages

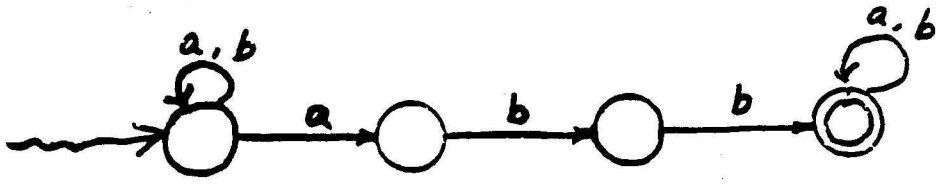
Example:

Consider the language of all the words that consist of a's and b's and have abb as a subword.

We can formally define this language by the following:

(1) $L = \{x \mid x \in \{a, b\}^*$ **and** x **has** abb **as a subword}**;

(2) $L = L(M)$ **where** M **is an NFA given by the following diagram:**



Both of the above definitions are lengthy. It can also be expressed by

$$L([a \cup b]^* abb[a \cup b]^*)$$

Definition Let Σ be an alphabet.
A regular expression over Σ is defined recursively:

Basis: (1) $\emptyset$,
(2) λ,
(3) a, where $a \in \Sigma$
are R.E.'s over Σ

Induction Step:
If E_1 and E_2 are R.E.'s over Σ, then
(4) $[E_1 \cup E_2]$,
(5) $[E_1 \cdot E_2]$,
(6) E_1^*
are R.E.'s over Σ

We usually omit $\cdot$.

The set of all regular expressions over Σ is denoted by

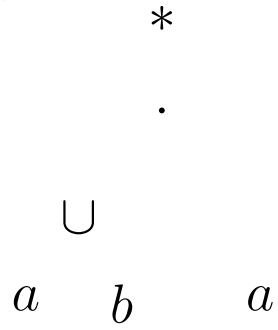
$$\mathcal{R}_\Sigma$$

$[[a+b]a]^*$ **is the same as** $[[a \cup b]a]^*$

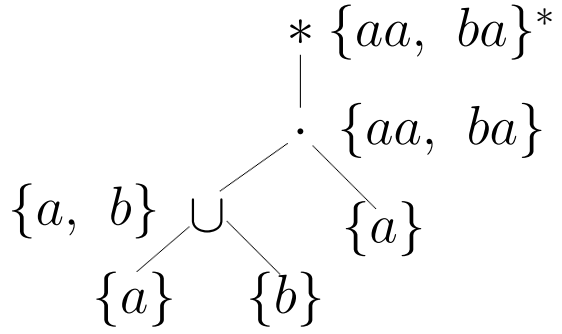
Example:

$\underbrace{[\underbrace{[\overbrace{a}^{E_1} \cup \overbrace{b}^{E_2}]}_{E_3} \underbrace{a}_{E_4}]^*}_{E_5}$ **is a R.E. over** $\{a,\ b\}$

We display the "parsing" by an expression tree:



What language does an R.E. define?



Definition **Given a regular expression E, the language $L(E)$ denoted by E is defined as follows:**

Basis: **(1) if $E = \emptyset$, then $\emptyset$;**

(2) if $E = \lambda$, then $\{\lambda\}$;

(3) if $E = a$, then $\{a\}$;

Induction Step:

(4) if $E = [E_1 \cup E_2]$, then $L(E_1) \cup L(E_2)$;

(5) if $E = [E_1 E_2]$, then $L(E_1)L(E_2)$;

(6) if $E = E_1^*$, then $(L(E_1))^*$.

Properties of R.E.'s

$\cup$: $E_1 \cup E_2 \equiv E_2 \cup E_1$;

$[E_1 \cup E_2] \cup E_3 \equiv E_1 \cup [E_2 \cup E_3]$;

So, $[E_1 \cup E_2] \cup E_3 \Rightarrow E_1 \cup E_2 \cup E_3$.

$\cdot$: $[E_1 E_2]E_3 \equiv E_1[E_2 E_3]$;

So, $[E_1 E_2]E_3 \Rightarrow E_1 E_2 E_3$.

. . . . . .

Definition **A language L is regular iff there is a regular expression E such that $L = L(E)$.**

The family of (all) regular languages is denoted by $\mathcal{L}_{REG}$.

Example $E = [b^*ab^*a]^*b^*$.
$L_{even} = \{x \mid x$ **in** $\{a,b\}^*$ **and** x **contains an even number of** $a's\}$.

Claim: $L(E) = L_{even}$

Proof:

(i) $L(E) \subseteq L_{even}$**, since every word in** $L(E)$ **contains an even number of** a**'s.**

(ii)Let $x \in L_{even}$**. Then** x **can be written as** $b^{i_0}ab^{i_1}a \ldots\ldots ab^{i_{2n}},\ i_0, i_1, \ldots, i_{2n} \geq 0$.
So, $x = (b^{i_0}ab^{i_1}a)(b^{i_2}ab^{i_3}a) \ldots\ldots (b^{i_{2n-2}}ab^{i_{2n-1}}a)b^{i_{2n}}$.
$x \in L([b^*ab^*a]^*)L(b^*) = L(E)$. $q.e.d.$

Examples $\Sigma = \{a,\ b\}$**.**

$L_1 = \{x \mid x = au,\ u \in \Sigma^*\}$**.**

$L_2 = \{x \mid |x|_a \equiv 0 \ \textbf{mod}\ 3\}$**.**

$[a[aa]]^*$

$L_3 = \{x \mid x$ **has 2 or 3** $a's$ **with the last two appearances nonconsecutive** $\}$

$L_4 = \{x \mid x = a^n b^n,\ n \geq 1\}$

Examples

What are the languages denoted by the following R.E.'s ?

$$E_1 = a^*ba^*$$

$$E_2 = [a \cup ab]^*$$

$$E_3 = a[a \cup b]^*a$$

$$E_4 = [aa \cup bb \cup ba \cup ab]^*$$

How many languages over Σ do R.E.'s define?

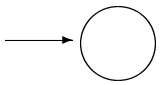
$$\Sigma = \{a,\ b\}$$

(1) Infinitely many ?

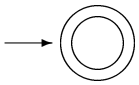
(2) Countable ?

Regular Expression into Finite Automata

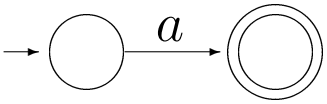
Let E be a regular expression over Σ. Then we can construct a λ-NFA M such that $L(M) = L(E)$, using the following rules:
(i) $E = \emptyset$.
Construct M such that $L(M) = \emptyset$



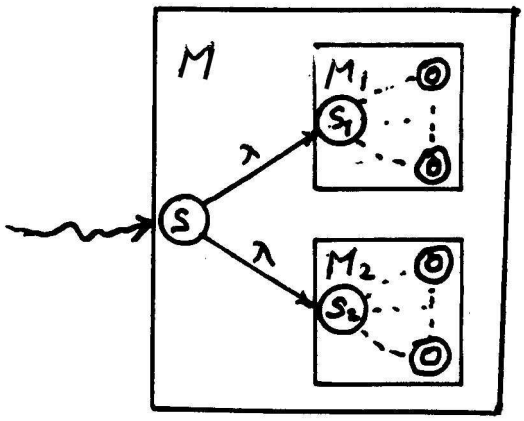
(ii) $E = \lambda$.
Construct M such that $L(M) = \{\lambda\}$



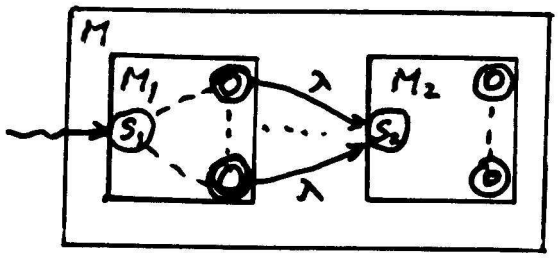
(iii) $E = a$, $a \in \Sigma$.
Construct M such that $L(M) = \{a\}$



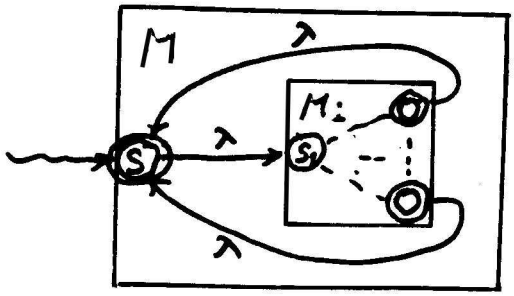
(iv) $E = [E_1 \cup E_2]$.
Construct M **such that** $L(M) = L(M_1) \cup L(M_2)$.



(v) $E = [E_1 E_2]$.
Construct M **such that** $L(M) = L(M_1)L(M_2)$.



(vi) $E = E_1^*$.
Construct M **such that** $L(M) = L(M_1)^*$.



Example

$E = [c^*[a \cup [bc^*]]^*]$

Construct a FA M such that $L(M) = L(E)$.

$\triangle$ $a, \quad b, \quad c$ **by (iii)**

$\triangle$ c^* **by (vi)**

$\triangle$ $[bc^*]$ **by (v)**

$\triangle$ $[a \cup bc^*]$ **by (iv)**

$\triangle$ $[a \cup bc^*]^*$ **by (vi)**

$\triangle$ $[c^*[a \cup bc^*]^*]$ **by (v)**

Theorem For E, an arbitrary regular expression over Σ, the λ-NFA, M, constructed as above satisfies $L(M) = L(E)$.

Proof: Let $Op(E)$ be the total number of $\cup,\ \cdot,$ and * operations in E. We prove this theorem by induction on $Op(E)$.

Basis: $Op(E) = 0$. Then $E = \emptyset,\ \lambda,$ or $a \in \Sigma$. Then clearly we have $L(M) = L(E)$.

Induction Hypothesis:
Assume the claim holds for all E with $Op(E) \leq k$, for some $k \geq 0$.

Induction Step:
Consider an arbitrary regular expression E with $Op(E) = k+1$. Since $k+1 \geq 1,\ E$ contains at least one operator $\cup,\ \cdot,$ or *.

Case I: $E = E_1 \cup E_2$. Then $Op(E_1) \leq k$ and $Op(E_2) \leq k$. So, $L(M_1) = L(E_1)$ and $L(M_2) = L(E_2)$ by I.H.. We know the construction of $M = M_1 \cup M_2$ satisfies $L(M) = L(M_1) \cup L(M_2)$, and $L(E) = L(E_1) \cup L(E_2)$

Therefore, $L(M) = L(E)$**.**

Case II: $E = E_1 E_2$

. . . . . .

Case III: $E = E_1^*$

. . . . . .

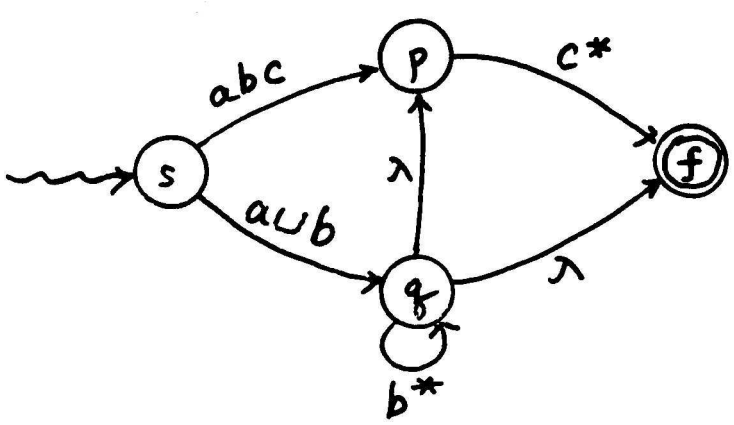
In each of the three cases, we have shown that $L(M) = L(E)$**. Therefore this holds for all regular expressions by the principle of induction.** $q.e.d.$

Finite Automata into Regular Expressions

To prove that every DFA language is regular we introduce an extension of finite automata.

Definition **An extended finite automaton (EFA), M, is a quintuple $(Q,\ \Sigma,\ \delta,\ s,\ f)$ where**
$Q,\ \Sigma,\ s$ **are as in λ-NFA,**
f **is the only final state, $f \neq s$,**
$\delta : Q \times Q \to R_\Sigma$ **is a total extended transition function.**

Example of an EFA:



$\delta(p, s) = \emptyset$
$\delta(s, f) = \emptyset$
. . . . . .
One final state $f \neq s$

△ **A <u>configuration</u> is in** $Q\Sigma^*$

△ **<u>Move</u>**

$$px \vdash qy \text{ if}$$

(i) $x = wy,\ w \in \Sigma^*,$

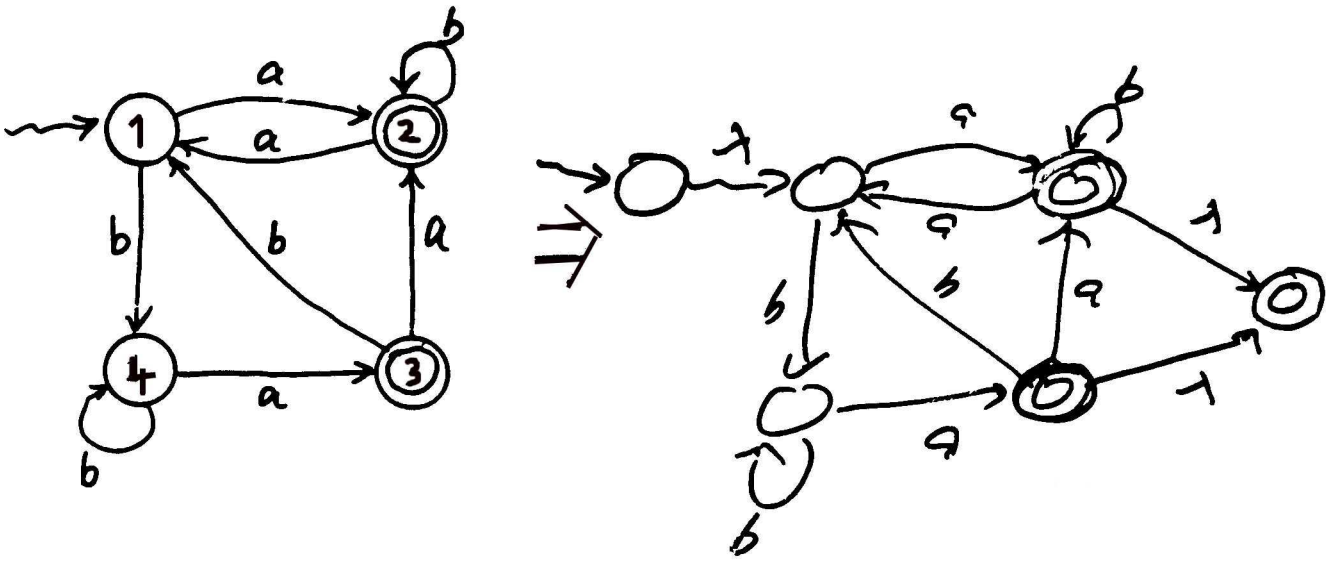
(ii) $\delta(p, q) = E,$ **and**

(iii) $w \in L(E).$

△ $\underline{\vdash^*}$, $\underline{\vdash^+}$ **are defined similarly as before.**

<u>Lemma</u> If M **is a DFA, Then there is an EFA** M' **with** $L(M') = L(M)$**.**

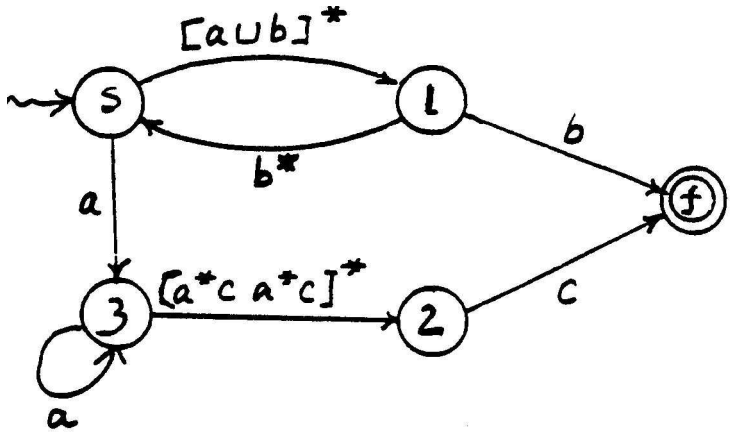
<u>Example</u> DFA into EFA.



Example:

An extended finite Automaton (EFA).

M:



Check if the following words are in $L(M)$

(1) $bbabab$

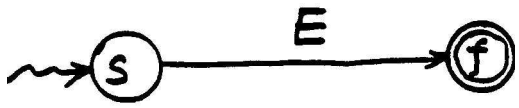
(2) $aabbc$

(3) $acccc$

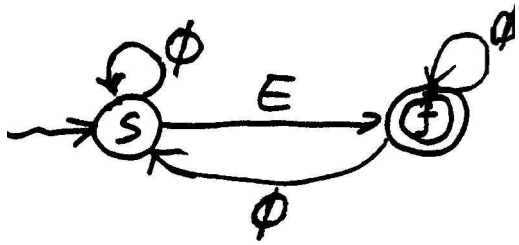
(4) $aaaaac$

State Elimination Technique

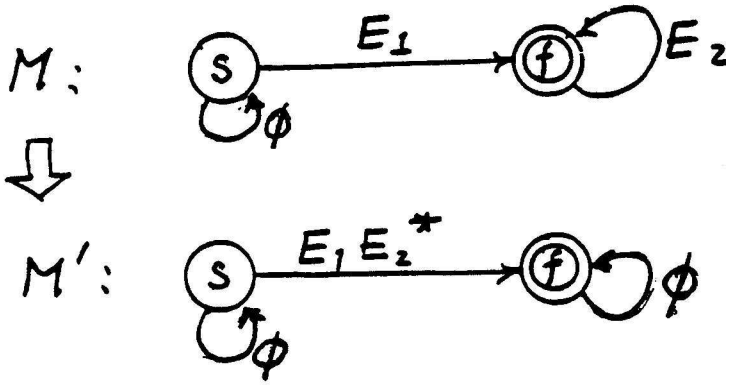
Goal of the technique:



i.e.:



(1) EFA has 2 states

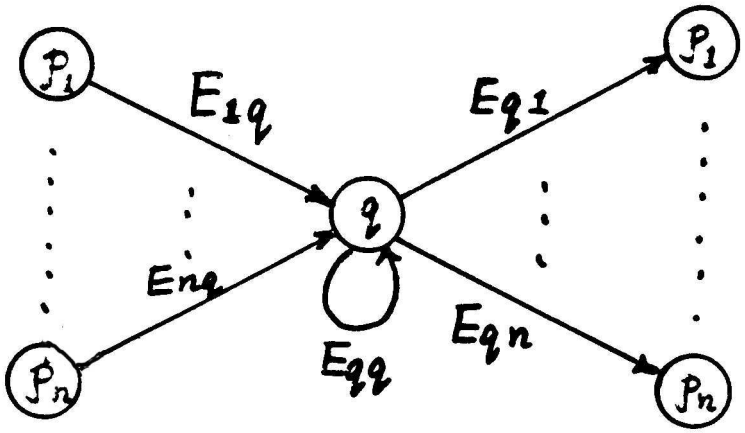


Example

(2) EFA M has $k+1$ states, $k \geq 2$.

Then eliminate a state from M to form M':

M :

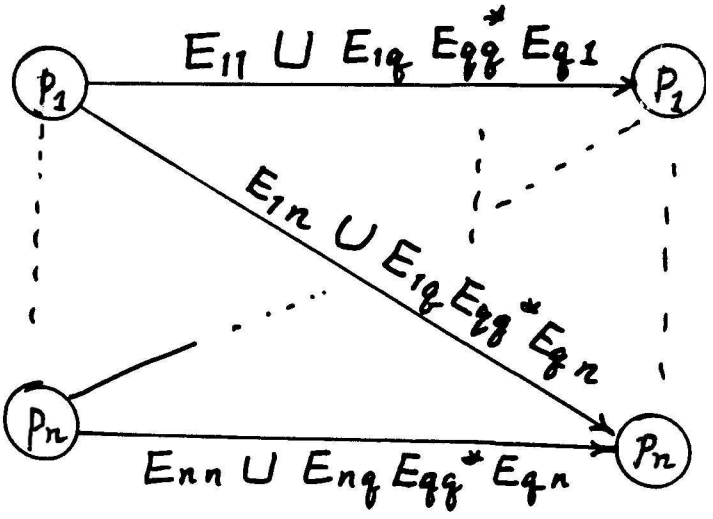


Note: $q \in Q - \{s,\ f\}$

Consider all transitions $(p_i,\ E_{iq},\ q)$

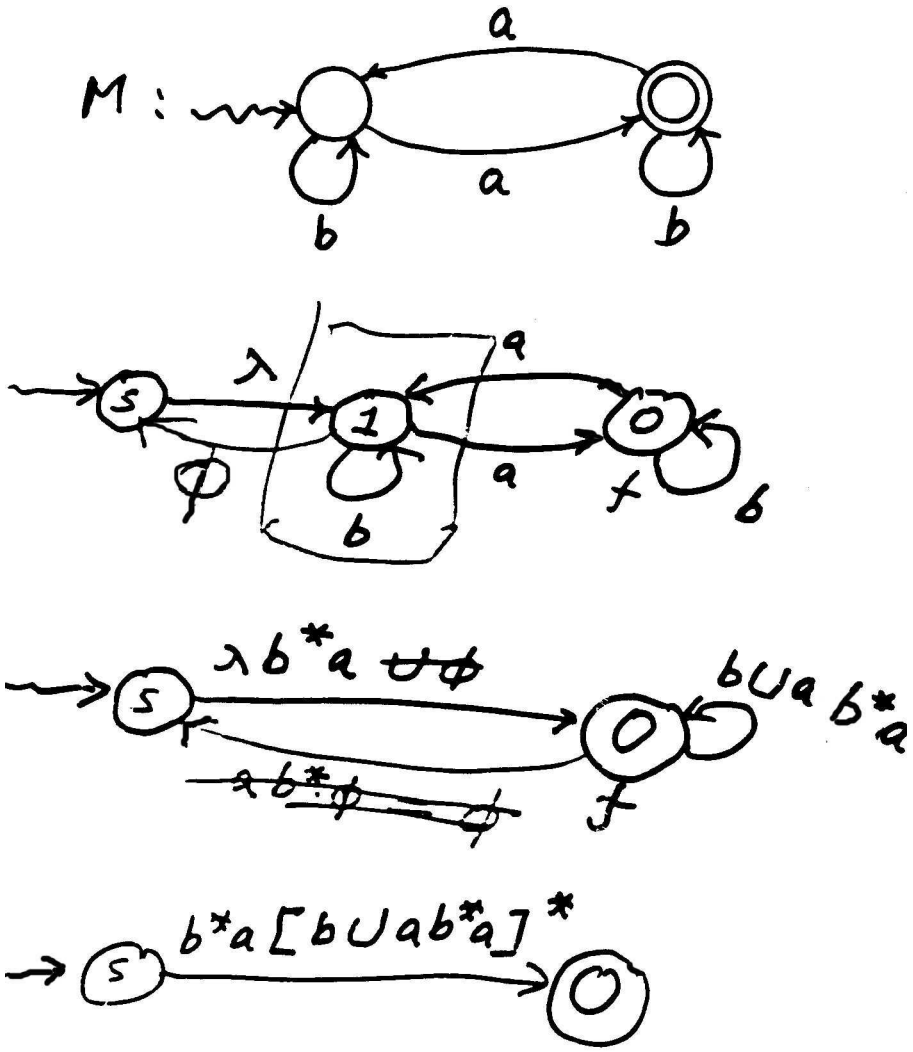
and $(q,\ E_{qi},\ p_i)$

M' :



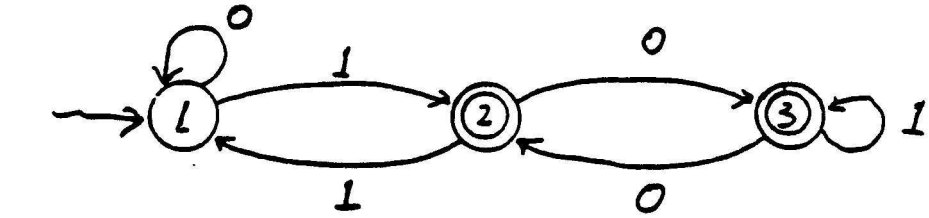
$$\delta'(p_i, p_j) = \delta(p_i, p_j) \cup \delta(p_i, q)(\delta(q, q))^* \delta(q, p_j)$$

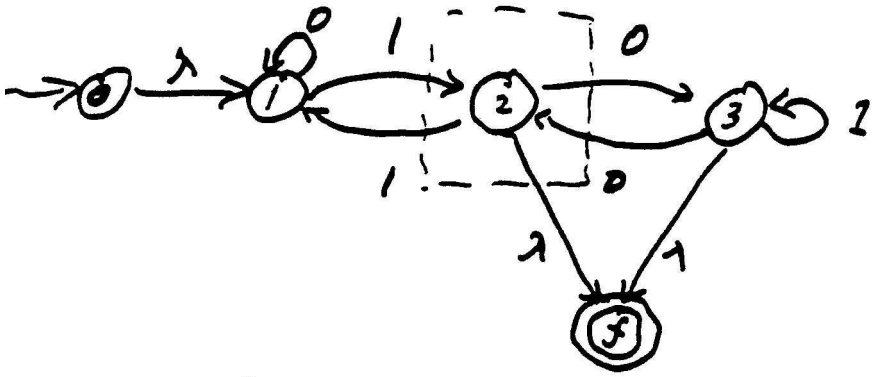
Example

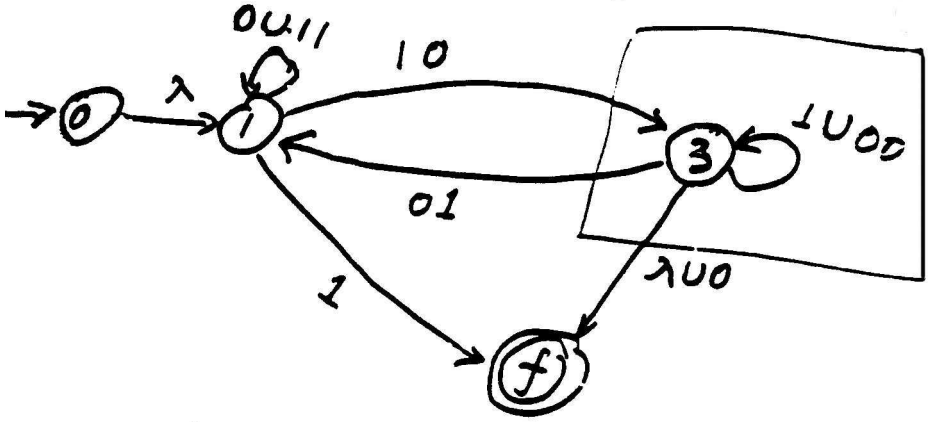


$b^*a[b \cup ab^*a]^*$

Example

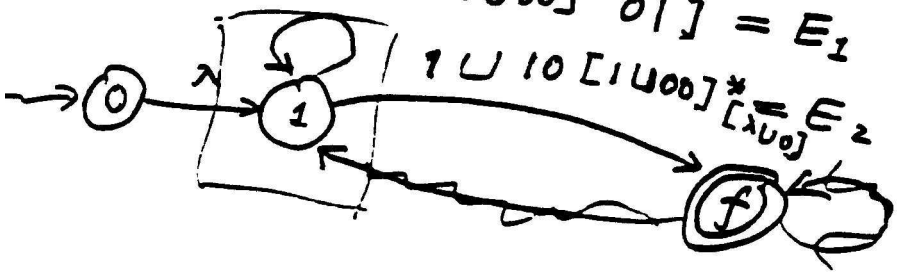


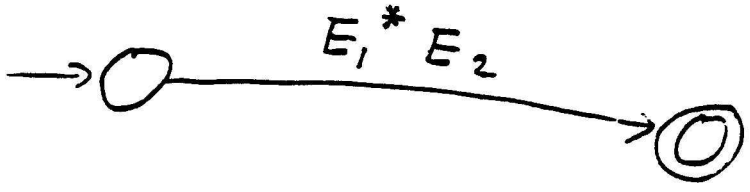




$0 \cup 11 \cup [10\,[1 \cup 00]^{*}\,01] = E_1$

$1 \cup 10\,[1 \cup 00]^{*}\,[\lambda \cup 0] = E_2$





Summary of the State Elimination Technique

(0) Change FA into EFA

(1) Add a new start state if the original one has incoming transitions.

(2) Add a new final state if there are more than one final states originally. Old final states become non-final states.

(3) Eliminate the states in $Q - \{s,\ f\}$ one by one.

(4) Eliminate the transition $\delta(f, f)$.