

I3307 - Theory of Computation

* **Alphabet:** (Σ) a finite set of symbols
 Binary alphabet is $\{0, 1\}$
 English alphabet is $\{a, b, \dots, z\}$

* **String:** finite sequence of symbols from an alphabet.

Note: the empty string λ contains no symbols.

• length of a string is its length as a sequence. $|\text{abbabb}| = 5$

• Σ^* : set of all possible strings in an alphabet. $|\lambda| = 0$

• **Concatenation** (\circ): $u = a_1 a_2 \dots a_n$
 $v = b_1 b_2 \dots b_m$ $u \circ v = uv = a_1 a_2 \dots a_n b_1 b_2 \dots b_m$

• $\lambda u = u \lambda = u$

• in general, $uv \neq vu$

• $|uv| = |u| + |v|$

• **Reverse**: $u = a_1 a_2 \dots a_n$ $u^R = a_n a_{n-1} \dots a_1$

• $(uv)^R = v^R u^R$

• **Power**: $u^n = \underbrace{u u u \dots u}_{n \text{ times}}$

• $u^0 = \lambda$

• u palindrome $\Leftrightarrow u^n$ palindrome

* **Language**: set of strings defined over an alphabet.

• **union**: $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$ inclusive-or: can be either

• **intersection**: $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ and } w \in L_2\}$ or both.

• **Complement**: $(\bar{L}) = \{w \mid w \notin L\}$ unless specified, consider the complement with respect to Σ^* .

• **Concatenation**: $L_1 \circ L_2 = \{w = xy \mid x \in L_1, y \in L_2\}$

• **Reverse**: $L_1^R = \{w^R \mid w \in L\}$

• **Power**: $L^n = \underbrace{L \circ L \circ L \dots \circ L}_{n \text{ times}}$ • $L^n = L \circ L^{n-1}$
 $L^0 = \lambda$

Note: just like strings, we can denote L^k as L^k for simplicity.

• **Kleene Closure**: $L^* = \bigcup_{i \in \mathbb{N}} L^i = \{\lambda, \underbrace{\dots, \dots, \dots}_{L^1}, \dots, \underbrace{\dots, \dots, \dots}_{L^2}, \dots\}$

• **Positive Closure**: $L^+ = L^* \setminus \{\lambda\}$ L^+ is closed under concatenation.
 $L^* = L^* L^*$ (L^*, \circ) is closed $\leftarrow x \in L^*, y \in L^* \Rightarrow xy = xy \in L^*$

Reminder:
 to prove something false: counter example

to prove it true: Method 1:
 taking general values
 (let $x \in \Sigma^*, y \in \Sigma^*$)

Method 2: Induction

{ - prove true for base case
 - assume true for K^i
 - prove true for K^{i+1}

* **Regular Expressions**: pattern that matches a set of strings.
It uses (*) the Kleene star that indicates the pattern repeats 0 or more times.

ex: $ab(a^*)$ matches ab, aba, abaaa, etc...

$L(r)$ is the language of the set of strings matching 'r'.

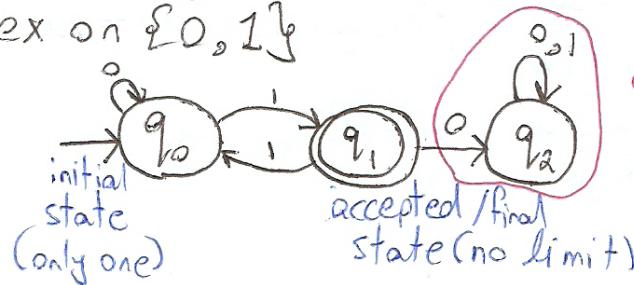
$$\cdot L(r_1 + r_2) = L(r_1) \cup L(r_2) \quad \text{ex: } (a+b) \text{ matches } a \text{ or } b.$$

$$\cdot L(r_1 r_2) = L(r_1) L(r_2) = \{xy \mid x \in L(r_1) \text{ and } y \in L(r_2)\}$$

$$\cdot L(r^*) = \{x_1 x_2 x_3 \dots x_n \mid x_i \in L(r)\}$$

Notes: $L((a+b)^*)$ = any string in $\{a, b\}^*$ $\neq L(a^* + b^*)$ = any string in $\{a\}^*$ or $\{b\}^*$

* **Automata**: ex on $\{0, 1\}$



dead end: a state that can never send us to an accepted state. Not adding a transition is equivalent of that transition being a dead end.

Deterministic Final Automata (DFA):

only one transition can leave a state for the same symbol.

defined by $(Q, \Sigma, \delta, q_0, F)$

↕ finite set of states ↕ input alphabet ↕ initial state ↗ set of final states
 δ ↗ transition function

ex: $\delta \mid 0 \ 1$



Regular Language: accepted by a certain DFA.

• L is regular $\Leftrightarrow \bar{L}$ is regular.

• L_1 and L_2 are regular \Rightarrow so is $(L_1 \cup L_2)$ and $(L_1 \cap L_2)$

Grammars: (V, Σ, R, S)

set of variables → set of terminals → rules → start variable

ex: $S \rightarrow xS$ } abbreviated
 $S \rightarrow y$
 $S \rightarrow yT$
 $T \rightarrow z$

A grammar is regular if the language defined by it is regular.

$$S \Rightarrow xS \Rightarrow xxS \Rightarrow \dots$$

at any point we can also replace S with y or yT which is yZ then?

$$S \Rightarrow xS \Rightarrow xy \quad \{x, xy, xyz, \dots, xyz\}$$

Language defined by this grammar.

Nondeterministic Final Automata (NFA) : multiple paths of the same symbol can leave each state.
 [DFA is a type of NFA]

defined by $(Q, \Sigma, \Delta, q_0, F)$

set of states
alphabet

transition function
initial state

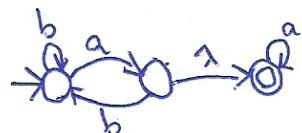
set of final states

$\Delta = Q \times \Sigma \rightarrow P(Q)$

powerset of Q :

$\{\emptyset, \{q_1\}, \{q_2\}, \{q_1, q_2\}, \dots\}$

NFAs can also have λ -transitions:



When traversing, taking a λ transition is optional

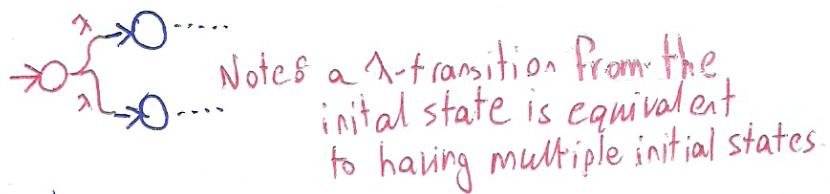
\Rightarrow there are multiple ways to traverse an NFA with the same input string.

A string only has to match one path to a final state to be accepted by an NFA.

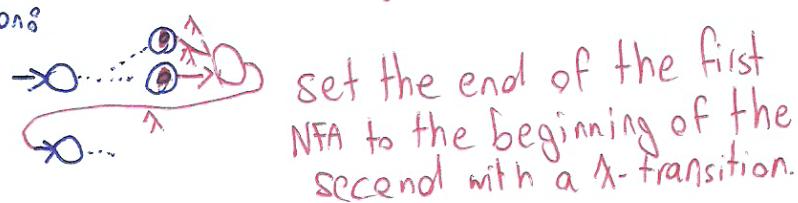
- Every NFA has an equivalent (\approx) DFA.
- Every NFA \approx to an NFA with one single accepting states



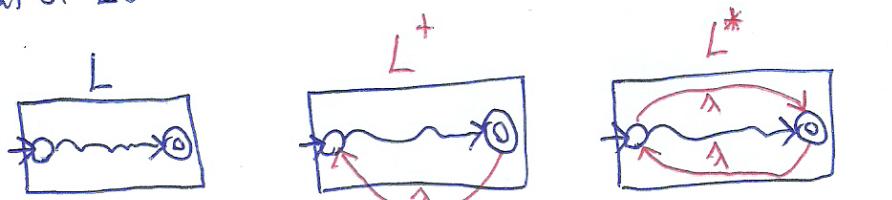
- Union with NFAs



- NFA concatenations



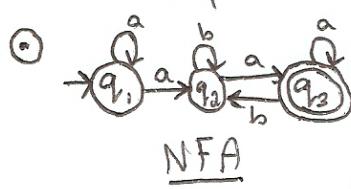
- NFA of L^* from that of L :



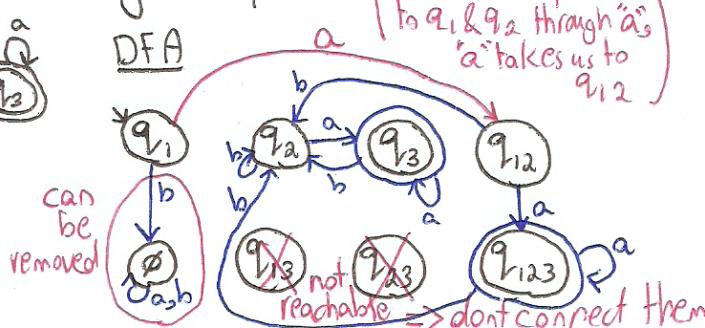
- NFA to DFA conversion:

explained using example:

(as q_1 has a path to q_1 & q_2 through 'a', 'a' takes us to q_{12})



DFA



- For q_{12} and "b":
 $q_1 + b \rightarrow \emptyset$
 $q_2 + b \rightarrow q_2$ } which is q_2
- "a":
 $q_1 + a \rightarrow q_1 \& q_{12}$
 $q_2 + a \rightarrow q_{12}$ } $\rightarrow q_{123}$

• REGEX \rightarrow NFA:

$$\emptyset : \rightarrow \textcircled{0}$$

$$\wedge : \rightarrow \textcircled{0}$$

$$a : \rightarrow \textcircled{0} \xrightarrow{a} \textcircled{0}$$

using these
rules, we can
deduce

$$ab : \rightarrow \textcircled{0} \xrightarrow{a} \textcircled{0} \xrightarrow{b} \textcircled{0}$$

$$atb : \rightarrow \textcircled{0} \xrightarrow{a} \textcircled{0}$$

$$\rightarrow \textcircled{0} \xrightarrow{b} \textcircled{0}$$

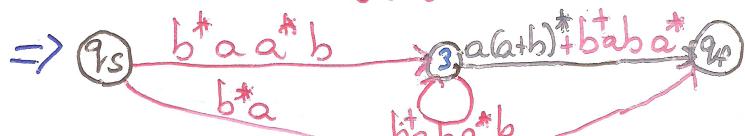
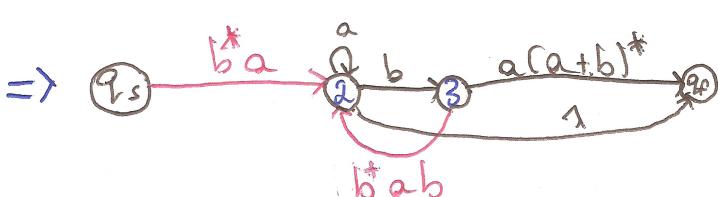
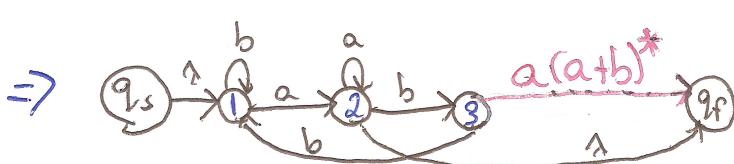
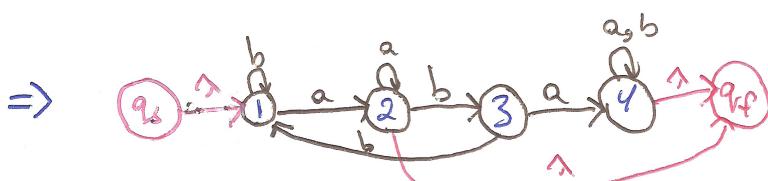
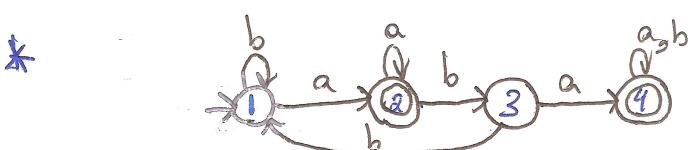
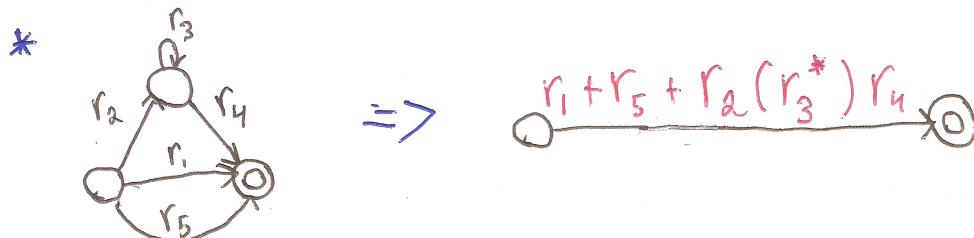
$$a^* : \rightarrow \textcircled{0} \xrightarrow{a} \textcircled{0} \xrightarrow{a} \textcircled{0}$$

• NFA \rightarrow REGEX

NFA \rightarrow DFA \rightarrow GNFA \rightarrow REGEX

where GNFA are DFAs with regex transitions

Better explained with examples:



4	<ul style="list-style-type: none"> $3 \xrightarrow{a(a+b)^*} q_f$
1	<ul style="list-style-type: none"> $q_s \xrightarrow{b^*a} 2$ $3 \xrightarrow{b^*a} 2$
2	<ul style="list-style-type: none"> $q_s \xrightarrow{b^*a^*b} 3$ $q_s \xrightarrow{b^*a^*} q_f$ $3 \xrightarrow{b^*a^*b} 3$
	<p>this path already exists</p>

The Pumping Lemma

The lemma is a general technique to prove that a language is not regular. It makes use of the following rule:

If L is regular $\Rightarrow \exists$ a "pumping length" p / $\forall w \in L$, if $|w| \geq p$

$\Rightarrow w = xyz$ where

$$\textcircled{a} |xy| \leq p$$

$$\textcircled{b} |y| > 0$$

$$\textcircled{c} xy^iz \in L \quad \forall i \geq 0$$

to prove languages not regular, assume they are and use \textcircled{a} & \textcircled{b} to disprove \textcircled{c}

Ex: $L_1 = \{a^n b^n \mid n \geq 0\}$ obviously not regular as there is no way for an NFA to count "n" to compare the amounts of a and b .

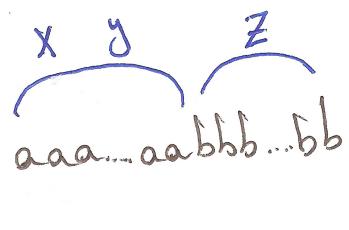
Assume that L is regular

$\Rightarrow \exists$ a pumping length p for L

$$\text{let } w = a^p b^p$$

$$w \in L \quad \& \quad |w| = p + p = 2p \geq p$$

then $w = xyz$ with $\begin{cases} |xy| \leq p \\ |y| > 0 \end{cases}$



$$\Rightarrow y = a^i \quad 1 \leq i \leq p$$

$$\text{but now } xy^2z = a^{p+i} b^p \in L$$

usually use xy^2z or xz (which has y^0) to disprove it.

Ex2: $L_2 = \{a^{n^2} \mid n \in \mathbb{N}^*\}$ take $w = a^{p^2} = xyz \Rightarrow y = a^k \quad 1 \leq k \leq p$
 $\therefore xy^2z = a^{p^2+k}$ but $p^2+k \leq (p+1)^2 \therefore xy^2z \notin L_2$
 since the next hop of a^{p^2} is $a^{(p+1)^2}$ and xy^2z falls in between the 2 \Rightarrow it doesn't belong to L_2 .