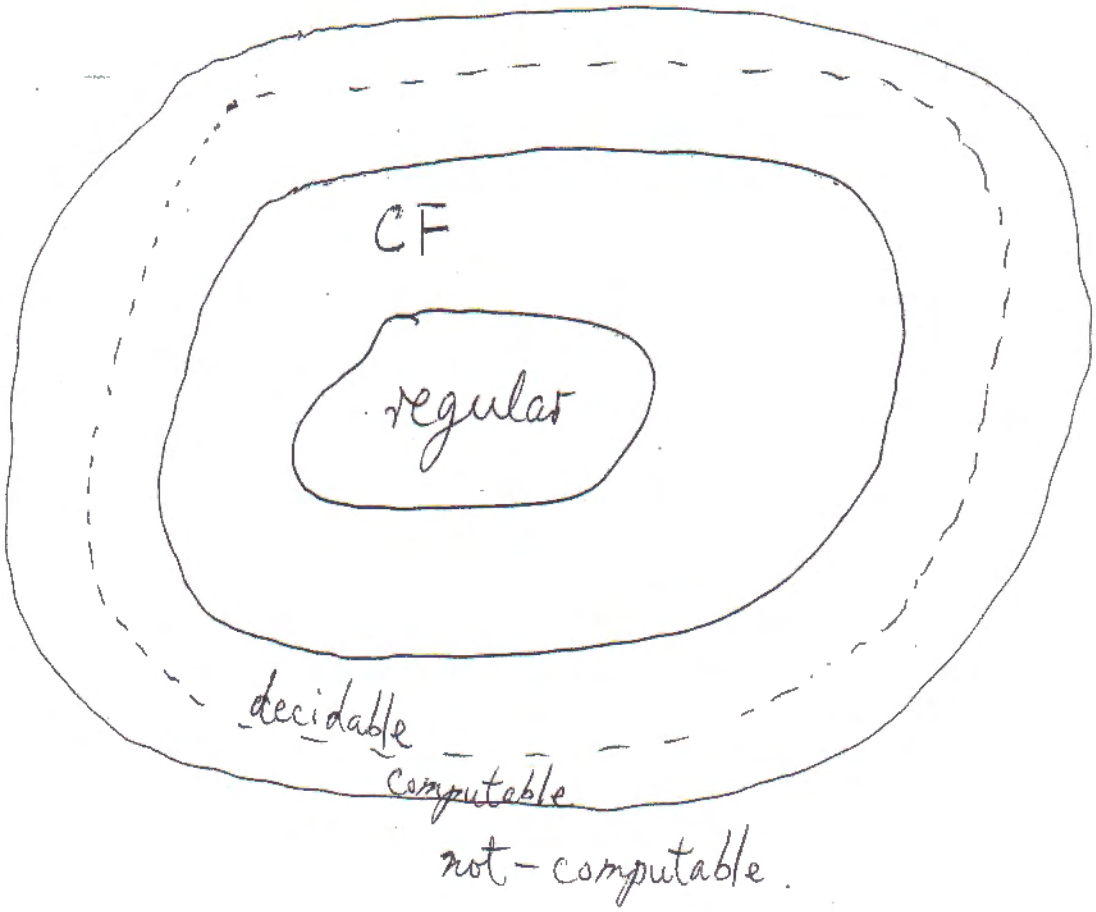
Computer Science

– compute to solve problems.

Classification of problem difficulty



– problems with "yes"/"no" answer.

Models of Computation

- finite representation of a solution.

1. machine / automata
   - recognizer
2. grammar
3. recursive function theory (Mathematics)

- one machine : recognizes all "yes" input
- one grammar : generates all "yes" input
- correctly answer the question
  - in a finite amount of time
  - for each input

problems	machine	grammar
regular	FA	regular
CF	PDA	context free
computable	TM	unrestricted

Notations.

- symbols e.g., digits, letters (lower case)

- string/word
  - sequence of symbols
  - constants e.g., 01101, abcba
  - variable e.g., w, x, y, z

  - length : # of symbols
    e.g., $|abcba| = 5$
  - ε : empty string, $|\varepsilon| = 0$.
    - always have input
      $\varepsilon \equiv$ input of length 0.

- concatenation operator
  the concatenation of strings w and x is wx.
  identity element for concatenation : ε.
  $$\varepsilon w = w\varepsilon = w$$

- set of strings
  - upper case letters e.g., A, B, L

Alphabet, Σ

- a finite set of symbols.
- e.g., $\Sigma = \{0, 1\}$
- Empty set $= \phi = \{\ \}$
- $\{\varepsilon\} \neq \phi$
- Σ^* = set of all strings over Σ

  e.g., $\Sigma = \{0, 1\}$.

  $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$
- $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.
- L, language : a set of strings

  e.g., $L = \{00, 000, 100, 0100, \ldots\}$

  - the set of all input strings that has "yes" answer.

  language $\equiv$ problem.

Regular languages.

e.g., string-search problems in text-editors.

Problem. $\Sigma = \{0, 1\}$, $\Sigma^* = \{\epsilon, 0, 1, 00, 01, \ldots\}$
L = set of all strings ending in 00.

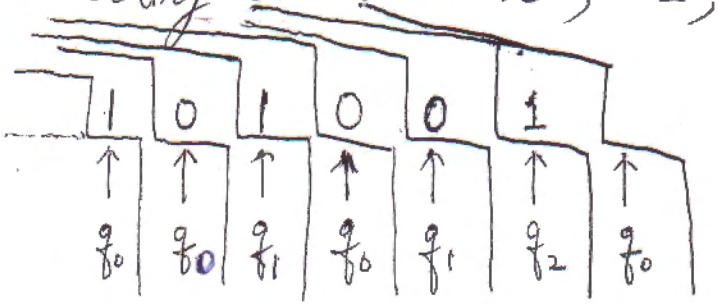
$101100 \in L$
$101 \notin L$

Solution: finite automata (FA)
- deterministic, one-way

idea:
- after process each symbol, in
- one of a finite # of possibilities

e.g., # of ending 0's : none, 1, 2

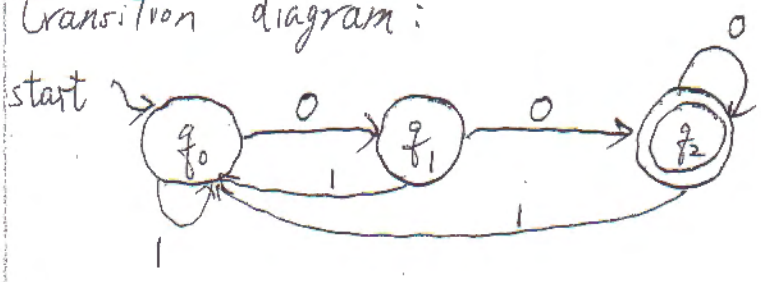
e.g.,



state	# of ending 0's
q_0	0
q_1	1
q_2	2

Finite representation of the solution:

transition diagram:



table; δ:

symbol / state	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_0

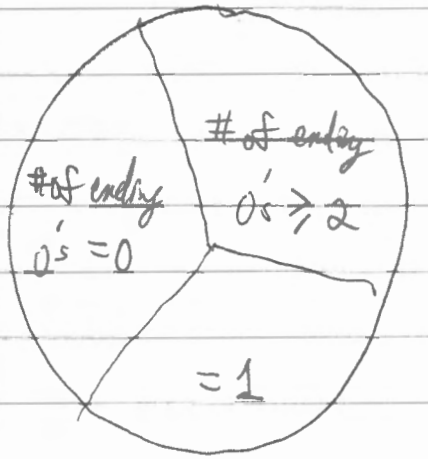
new:

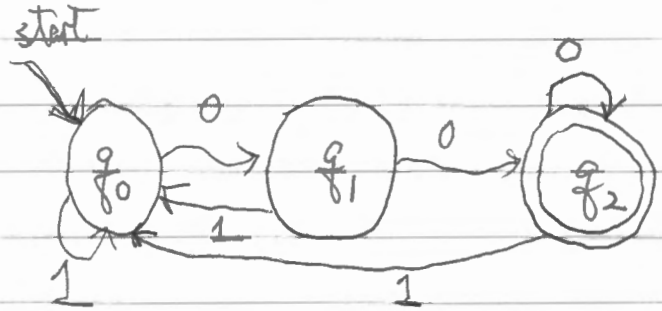
symbol / state	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_0
$*q_2$	q_2	q_0

Formal solution: $FA, M = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_2\})$

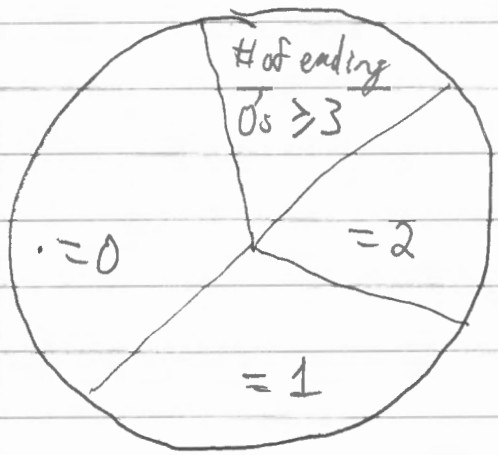
Formal: $FA, M = (Q, \Sigma, \delta, q_0, F)$.

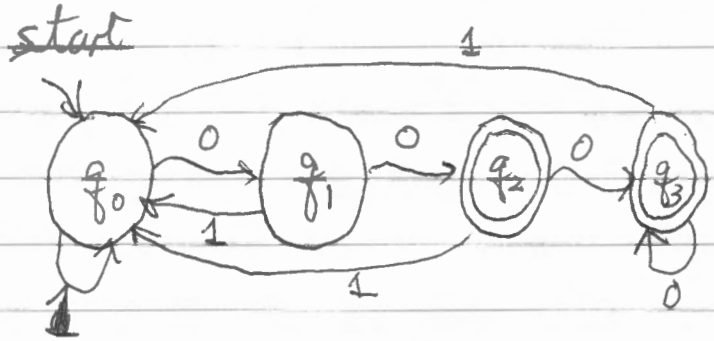
all possible input:





An alternative answer





Deterministic, one-way finite automata
— DFA, FA, $M = (Q, \Sigma, \delta, q_0, F)$

- finite set of states, Q
- starts — in initial state : q_0
- process symbols : from left to right
- deterministic :
  each state, each symbol
  — exactly one transition
- ends after whole input processed, in state q
  — accept if $q \in F$
  — reject if $q \notin F$

— extended transition function :

$$\hat{\delta}(q, \varepsilon) = q$$
$$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$$

— $L(M) = \{ x \mid \hat{\delta}(q_0, x) \in F \}$.
$= \{ x \mid \delta(q_0, x) \in F \}$ for simplicity

Def. A language is regular if there exists a FA accepts the set.

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

Problem. Given a number, is 1 the remainder when divided by 3?

(i) leading 0's allowed.

(ii) no leading 0's.

DFA :

- 3 possibilities :

1. remainder is 0 : q_0
2. " 1 : q_1
3. " 2 : q_2

- Any binary #, one more digit on its right:

either $2*\#$ or $2*\# + 1$.

- original remainder, r.

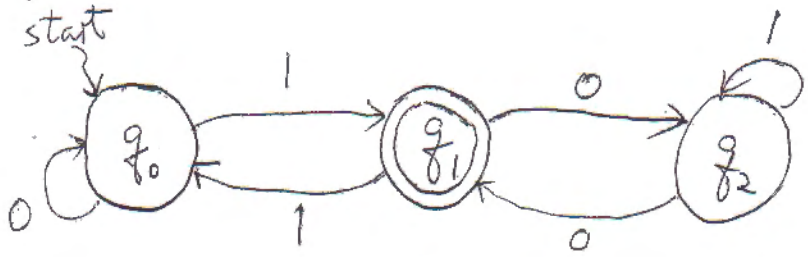
$$x = 3q + r \quad , \quad r = \{0, 1, 2\}.$$

$$x0 : \; 2x = 2(3q + r) = 3(2q) + 2r$$

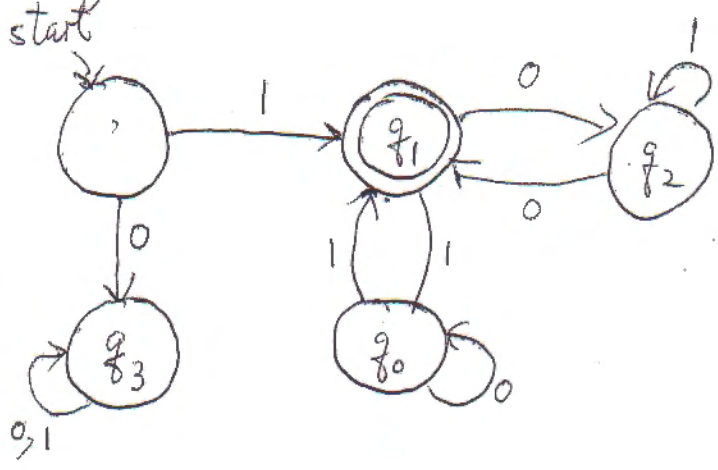
$$x1 : \; 2x+1 = 3(2q) + 2r + 1$$

r \ input	0	1
0	0	1
1	2	0
2	1	2

(i) leading 0's allowed :



(ii) no leading 0's :

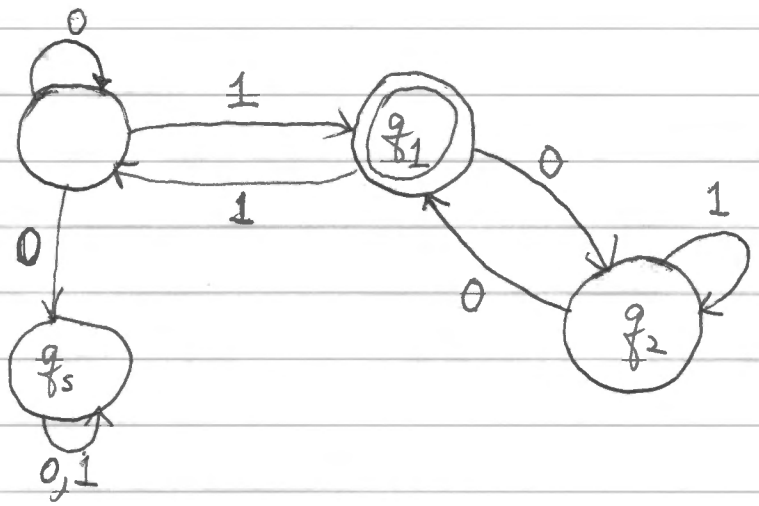


— states renumbering/renaming.

DFA: leading 0's illegal

Why need a separate state for q_0.

if not,



- need a 0 transition back to itself
  to allow/accept non-leading 0's strings that have an intermediate remainder of 0.

e.g. 1 1 1 1
1 1 0 1

remainder 0

Problems:

1. not a DFA (a NFA)
   - not a problem if it is correct

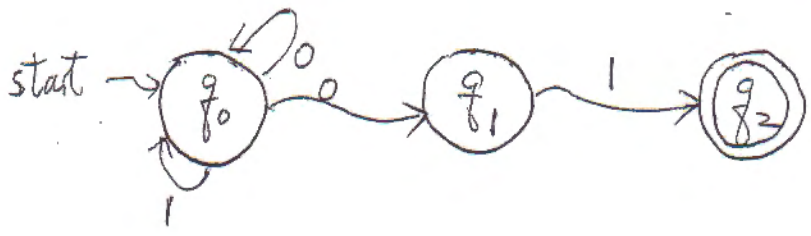
2. incorrect: it accepts leading 0's strings

e.g. 0 0 1 1 | 1
0 1 1 0 0 0 | 1

NFA

nondeterministic
- a symbol : any # of transitions
  0, 1, 2, 3, ...
- ability to guess.
- accepts if any sequence leads to a final state

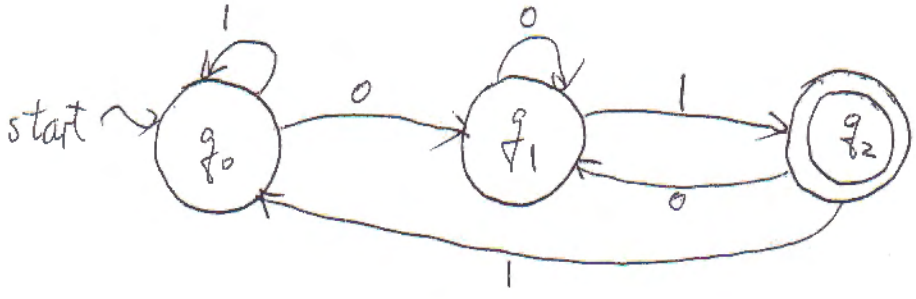
Problem. Strings that ends in 01 , $\Sigma = \{0, 1\}$.



- q_0 : 0 ~ guess

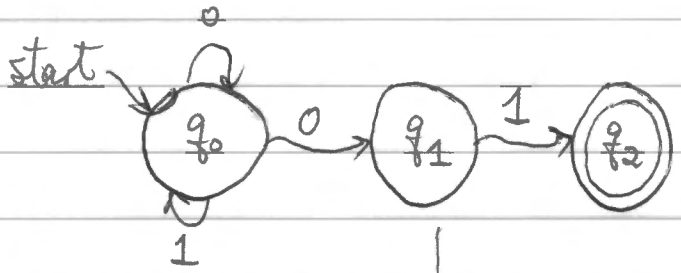
DFA : 3 cases (00, 01, 10, 11)

1. stat/start over : q_0 $\varepsilon \equiv 1 \equiv 11$
2. see the ending 0 : q_1 $0 \equiv 00 \equiv 10$
3. see 01 : q_2 01

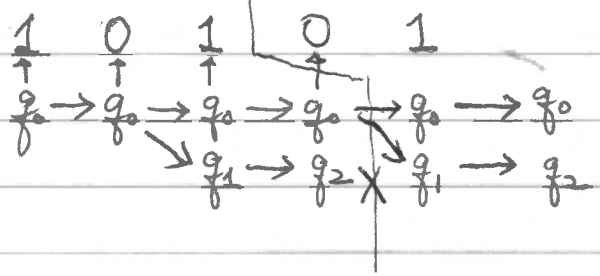


NFA

strings that ends in 01



e.g.



NFA → DFA

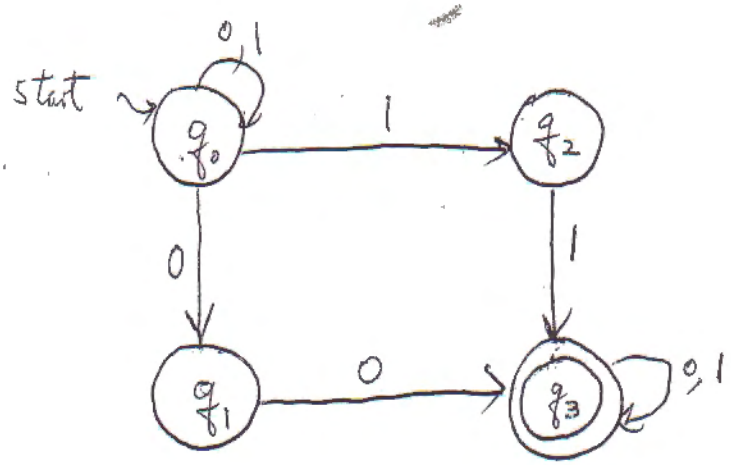
- each column of alternative states ≡ a single state in DFA.

NFA.

e.g. Problem. $L = \{$ 2 consecutive 0's or 2 consecutive 1's $\}$.

NFA: guess when condition starts to happen.

e.g., 0: guess — start of the 2 consecutive 0's.
~ or not.



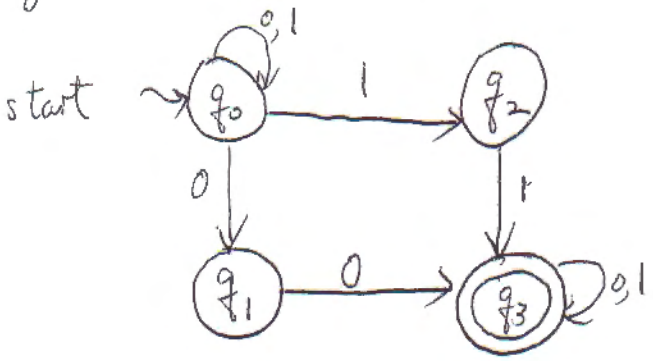
alternative reasoning:

- starts with any string
- consecutive 0 or 1
- ends with any string

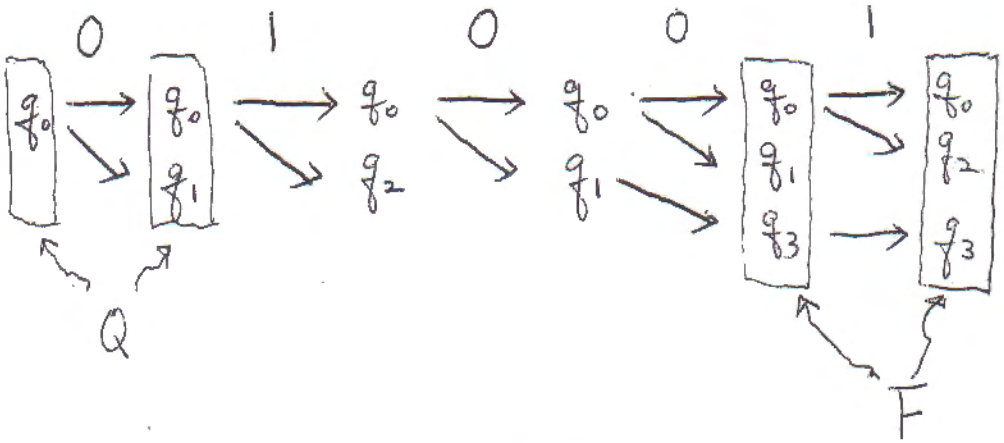
DFA $\equiv$ NFA

- DFA $\subseteq$ NFA
- NFA $\subseteq$ DFA

e.g., NFA for $L = \{$ 2 consecutive 0's or 1's $\}$.



e.g.



Potential states in "converted" DFA:

- $[q_0]$, $[q_1]$, $[q_2]$, $[q_3]$
- $[q_0, q_1]$, $[q_0, q_2]$, $[q_0, q_3]$, $[q_1, q_2]$, $[q_1, q_3]$, $[q_2, q_3]$
- $[q_0, q_1, q_2]$, $[q_0, q_1, q_3]$, $[q_0, q_2, q_3]$, $[q_1, q_2, q_3]$
- $[q_0, q_1, q_2, q_3]$
- $\emptyset$

"lazy evaluation":

$\delta([q_0], 0) = [q_0, q_1]$ $\quad$ $\delta([q_0], 1) = [q_0, q_2]$

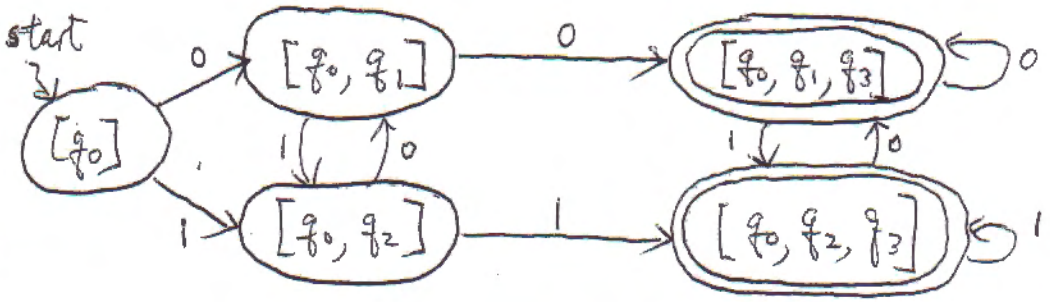
$\delta([q_0, q_1], 0) = [q_0, q_1, q_3]$ $\quad$ $\delta([q_0, q_1], 1) = [q_0, q_2]$

$\delta([q_0, q_2], 0) = [q_0, q_1]$ $\quad$ $\delta([q_0, q_2], 1) = [q_0, q_2, q_3]$

$\delta([q_0, q_1, q_3], 0) = [q_0, q_1, q_3]$ $\quad$ $\delta([q_0, q_1, q_3], 1) = [q_0, q_2, q_3]$

$\delta([q_0, q_2, q_3], 0) = [q_0, q_1, q_3]$ $\quad$ $\delta([q_0, q_2, q_3], 1) = [q_0, q_2, q_3]$

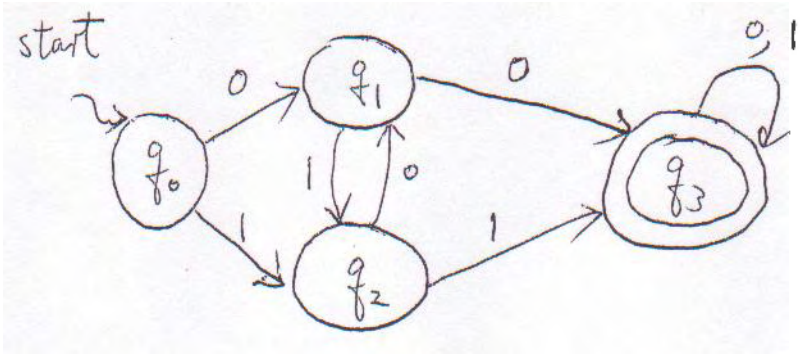
DFA:



$[q_0]$: no 2 consecutive 0's nor 1's, no last symbol
$[q_0, q_1]$: " , last symbol is 0
$[q_0, q_2]$: " " is 1
$[q_0, q_1, q_3]$: have consecutive 0's or 1's, " is 0
$[q_0, q_2, q_3]$: " " is 1.

- not the best DFA (min. # of states)
- > 1 DFA ≡ same language.

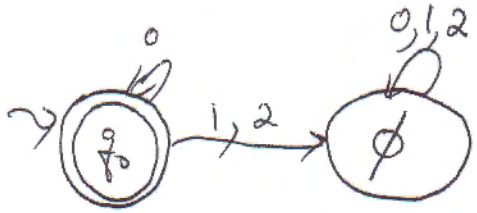
Best DFA for $L = \{$ 2 consecutive 0's or 1's $\}$.



$L = \{$ words with any # of 0's $\}$, $\Sigma = \{0, 1, 2\}$ and no other types of symbols.

NFA :

DFA : $\delta([q_0], 0) = [q_0]$, $\delta([q_0], 1) = \emptyset$, $\delta([q_0], 2) = \emptyset$.

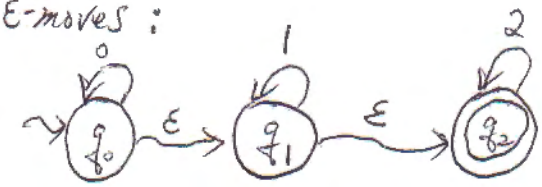


$\delta(\emptyset, 0) = \emptyset$, $\delta(\emptyset, 1) = \emptyset$, $\delta(\emptyset, 2) = \emptyset$

Problem. $L = \{$ words with any # of 0's, followed by any # of 1's, followed by any # of 2's $\}$

0^* :

NFA with ε-moves :



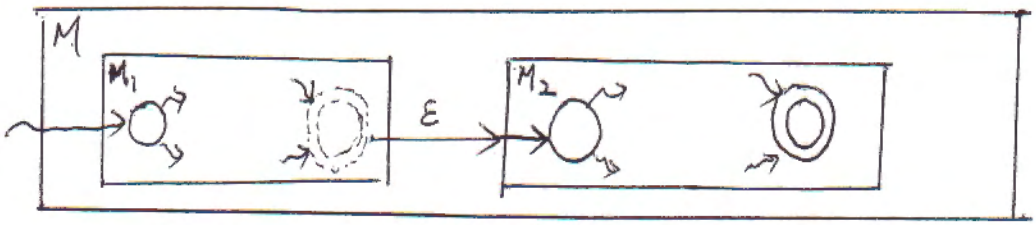
$L(M) = \{ w \mid \hat{\delta}(q_0, w)$ contains a state in $F \}$

where

$\hat{\delta}(q, w) = \{$ all states reachable from q on w using all possible transitions including ε-transitions $\}$

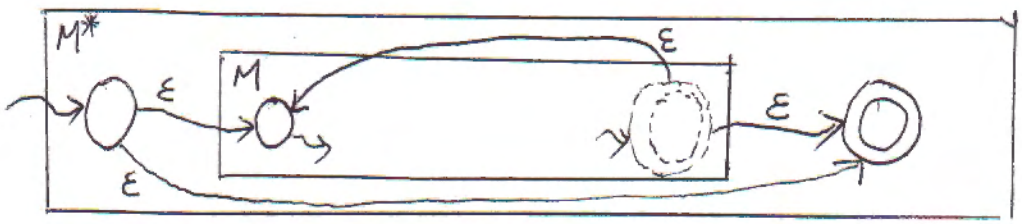
2. "sequence" of problems

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



3. any # of times (Kleene closure)

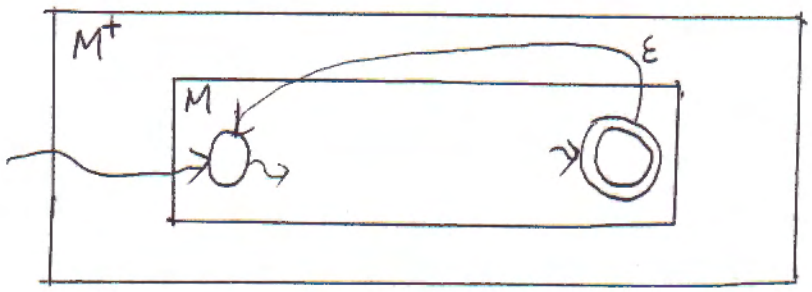
$$L(M^*) = [L(M)]^*$$



---

e.g. 4. at least 1 times (positive closure)

$$L(M^+) = [L(M)]^+$$



L = { 0's followed by 1's followed by 2's }.

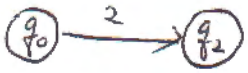
NFA without ϵ-moves: 3 states:

q_0 : working on the 0's.

q_1 : working on the 1's.

q_2 : " 2's.

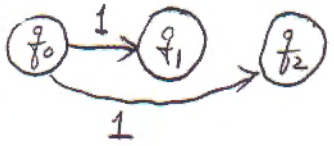
At q_0: (i) read 2 $\xrightarrow{\text{no more 0's, 1's}}$ q_2



(ii) read 1, no more 0's:

guess: (a) some more 1's $\rightarrow q_1$

(b) no more 1's $\rightarrow q_2$

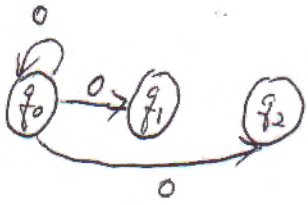


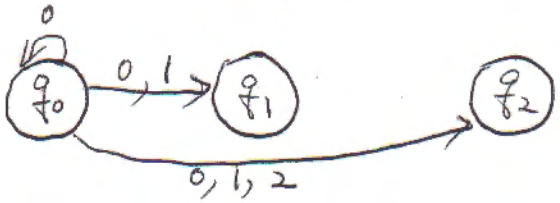
(iii) read 0:

guess: (a) some more 0's $\rightarrow q_0$

(b) no more 0's, have 1's $\rightarrow q_1$

(c) no more 0's and 1's $\rightarrow q_2$





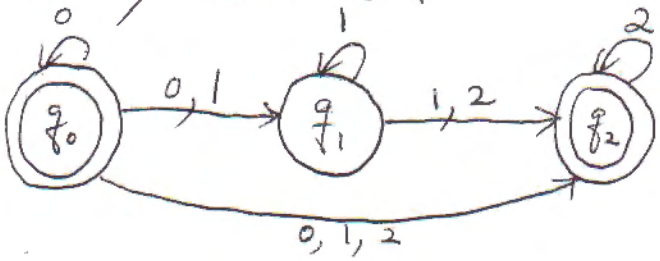
At q_1: (i) read 2: no more 0's, 1's $\rightarrow q_2$

(ii) read 1: guess: (a) some more 1's $\rightarrow q_1$

(b) no more 1's $\rightarrow q_2$

(iii) cannot read 0: 0's after 1's. $\notin L$.

At q_2: can only read 2's.



$F = \{q_0, q_2\}$. , q_0 : $\epsilon \in L$.

NFA with ε-moves $\equiv$ NFA without ε-moves.

- NFA without ε-moves $\subseteq$ NFA with ε-moves.

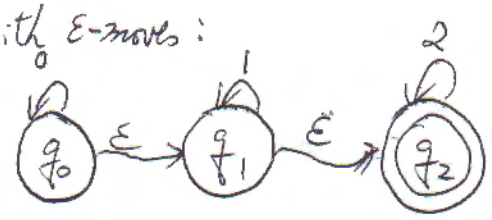
- NFA with ε-moves. $\subseteq$ NFA without ε-moves.

NFA with ε-moves, $M = (Q, \Sigma, \delta, q_0, F)$
NFA without ε-moves, $M' = (Q, \Sigma, \hat{\delta}, q_0, F')$

$$F' = \begin{cases} F \cup \{q_0\} & \text{if } \varepsilon\text{-closure}(q_0) \text{ contains a state of } F \\ F & \text{otherwise} \end{cases}$$

Def. ε-closure(q) = { states reachable from state q using no transition or ε-transitions only }

e.g., NFA with ε-moves:



ε-closure(q_2) = { q_2 }
ε-closure(q_1) = { q_1, q_2 }
ε-closure(q_0) = { q_0, q_1, q_2 }

Def. ε-closure(P) = $\bigcup_{q \in P} \varepsilon$-closure($q$)

where P is a set of states.

Def. $\hat{\delta}$: (1) $\hat{\delta}(q, \varepsilon) = \varepsilon\text{-closure}(q)$

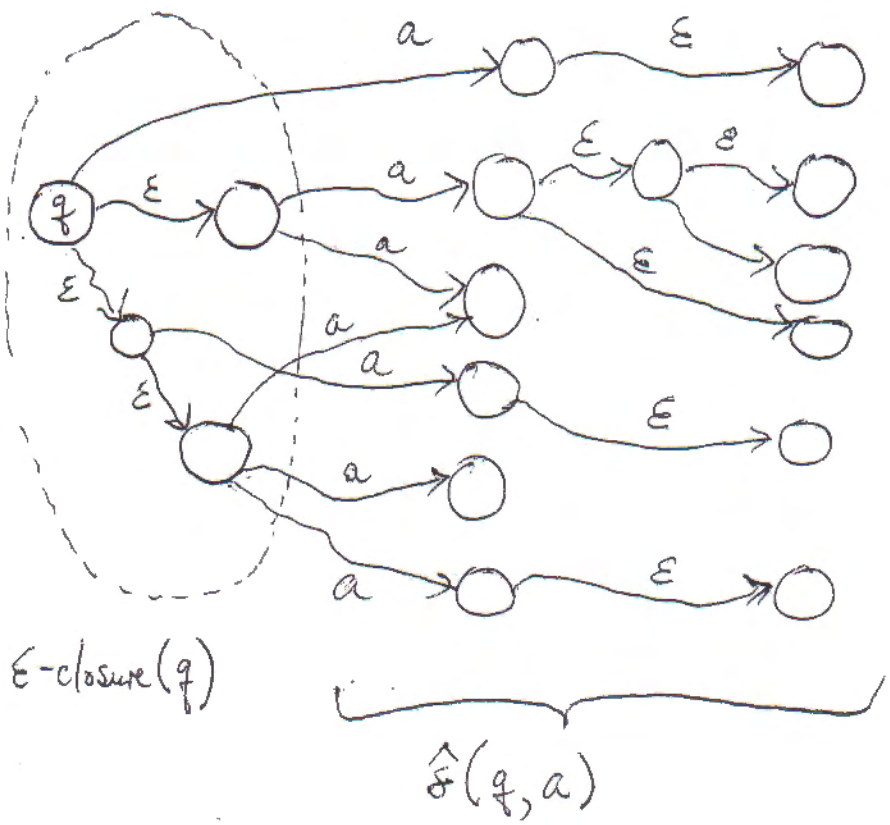
(2) $\forall w \in \Sigma^*, a \in \Sigma, \quad \hat{\delta}(q, wa) = \varepsilon\text{-closure}\left(\delta(\hat{\delta}(q,w), a)\right)$

(3) $\delta(R, a) = \bigcup_{q \in R} \delta(q, a)$

(4) $\hat{\delta}(R, w) = \bigcup_{q \in R} \hat{\delta}(q, w)$.

$\delta(q,a) = \{$ states reachable from q by arcs labelled $a\}$.

$$\begin{aligned}\hat{\delta}(q,a) &= \hat{\delta}(q, \varepsilon a) \\ &= \varepsilon\text{-closure}\left(\delta(\hat{\delta}(q,\varepsilon), a)\right) \\ &= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(q), a)\right) \\ &= \{\text{states reachable from } q \text{ by paths labelled } a\}.\end{aligned}$$



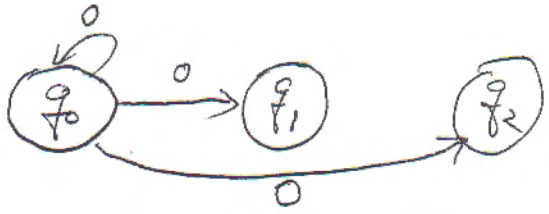
$\hat{\delta}(q,a) \neq \delta(q,a)$, $\hat{\delta}(q,\varepsilon) \neq \delta(q,\varepsilon)$

e.g.,

$\varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$.

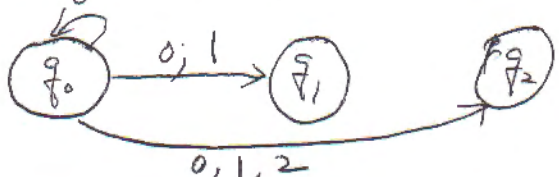
$$\begin{aligned}\hat{\delta}(q_0, 0) &= \varepsilon\text{-closure}\big(\delta(\varepsilon\text{-closure}(q_0), 0)\big) \\ &= \varepsilon\text{-closure}\big(\delta(\{q_0, q_1, q_2\}, 0)\big) \\ &= \varepsilon\text{-closure}(q_0) \\ &= \{q_0, q_1, q_2\}.\end{aligned}$$

⇒ NFA without ε-moves :

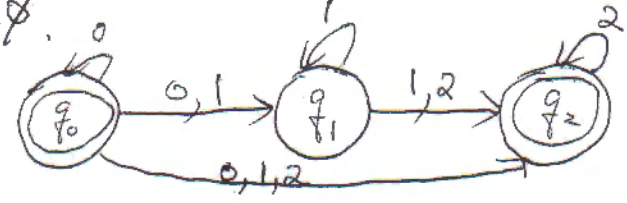


$$\begin{aligned}\hat{\delta}(q_0, 1) &= \varepsilon\text{-closure}\big(\delta(\varepsilon\text{-closure}(q_0), 1)\big) \\ &= \varepsilon\text{-closure}\big(\delta(\{q_0, q_1, q_2\}, 1)\big) \\ &= \varepsilon\text{-closure}(q_1) \\ &= \{q_1, q_2\}.\end{aligned}$$

$\hat{\delta}(q_0, 2) = \varepsilon\text{-closure}\big(\delta(\{q_0, q_1, q_2\}, 2)\big) = \varepsilon\text{-closure}(q_2) = \{q_2\}$.



$$\begin{cases}\hat{\delta}(q_1, 0) = \varepsilon\text{-closure}\big(\delta(\varepsilon\text{-closure}(q_1), 0)\big) \\ \qquad = \varepsilon\text{-closure}\big(\delta(\{q_1, q_2\}, 0)\big) = \varepsilon\text{-closure}(\emptyset) = \emptyset \\ \hat{\delta}(q_1, 1) = \varepsilon\text{-closure}\big(\delta(\{q_1, q_2\}, 1)\big) = \varepsilon\text{-closure}(q_1) = \{q_1, q_2\}. \\ \hat{\delta}(q_1, 2) = \varepsilon\text{-closure}\big(\delta(\{q_1, q_2\}, 2)\big) = \varepsilon\text{-closure}(q_2) = \{q_2\} \\ \hat{\delta}(q_2, 2) = \varepsilon\text{-closure}\big(\delta(\varepsilon\text{-closure}(q_2), 2)\big) = \varepsilon\text{-closure}\big(\delta(\{q_2\}, 2)\big) = \{q_2\} \\ \hat{\delta}(q_2, 0) = \hat{\delta}(q_2, 1) = \emptyset.\end{cases}$$



Regular Expressions.

Def. The regular expressions over an alphabet, Σ, and the sets (languages) that they represent are defined recursively as:

(1) $\emptyset$ is a regular expression : represents $\emptyset$
(2) ε " : " $\{\varepsilon\}$
(3) For each $a \in \Sigma$, a " : " $\{a\}$
(4) if r represents R
s " S,
then $r+s$: " $R \cup S$
rs : " $R \cdot S$
r^* : " R^*

– order of precedence : $*$ > concatenation > $+$

e.g., $(0+1)^*011$
$0^*1^*2^*$
$0^+1^+2^+ \equiv 00^*11^*22^*$

FA $\equiv$ regular expressions.

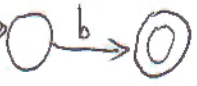
(i) regular expression $\subseteq$ FA.
NFA with ε-moves.

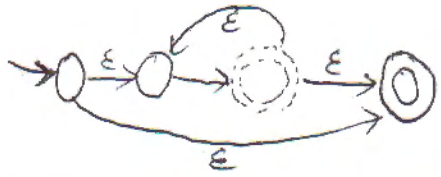
regular expression $\subseteq$ FA

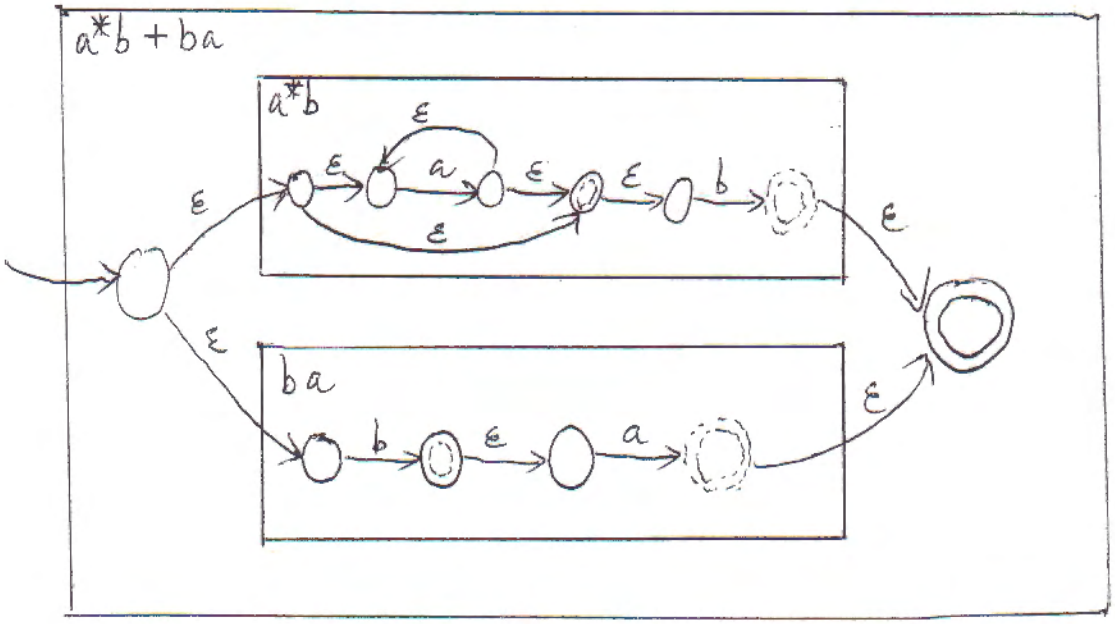
e.g., $\Sigma = \{a, b\}$

$r = a^* b + ba$

a: 

b: 

a^* 



(ii) DFA $\subseteq$ regular expressions.

DFA, $M = (\{q_1, \dots, q_n\}, \Sigma, \delta, q_1, F)$

Define $R_{ij}^k = \{ x \mid \delta(q_i, x) = q_j$ without going through any state numbered higher than $k \}$

$R_{ij}^0, R_{ij}^1, R_{ij}^2, \dots, R_{ij}^n = \{ x \mid \delta(q_i, x) = q_j \}$

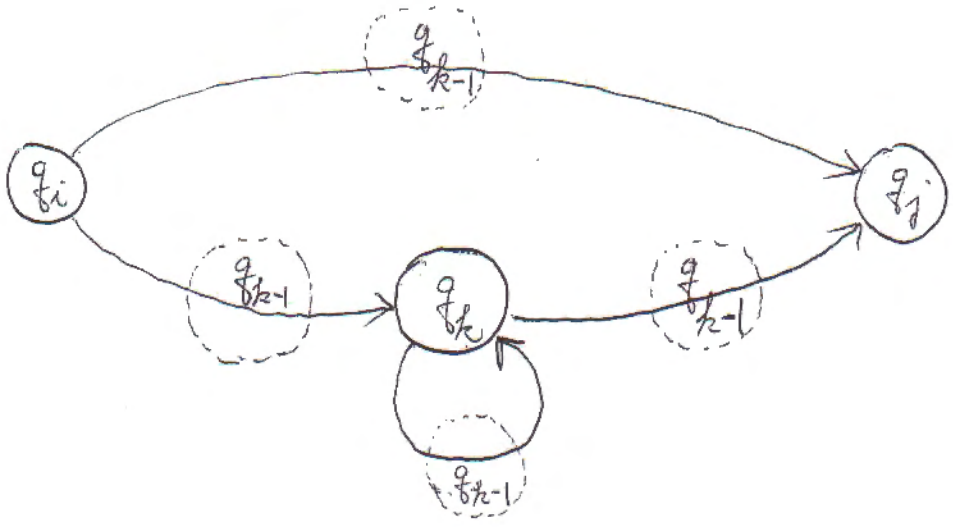
- $L(M) = \bigcup_{q_j \in F} R_{1j}^n$

- if r_{ij}^k is the regular expression for R_{ij}^k, and $F = \{ q_{j_1}, q_{j_2}, \dots, q_{j_p} \}$, then $L(M) = r_{1j_1}^n + r_{1j_2}^n + \dots + r_{1j_p}^n$

By dynamic programming,

$$R_{ij}^0 = \begin{cases} \{a \mid \delta(q_i, a) = q_j\} & \text{if } i \neq j \\ \{a \mid \delta(q_i, a) = q_j\} \cup \{\varepsilon\} & \text{if } i = j. \end{cases}$$

$$R_{ij}^k = R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1} \cup R_{ij}^{k-1}$$



Necessary and Sufficient Condition.

e.g.,	Given	condition	classification
	quadrilaterals	{ all sides equal and all angles equal (90°)	square

- can prove both positive and negative results.

Sufficient Condition. (not necessary)

e.g.,	Given	condition	classification
	quadrilaterals	- all sides equal (rhombus) - all angles equal (rectangle)	parallelogram
	person	- born in US. - immigrate legally	US citizen

- can prove only positive results.

Necessary Condition (not sufficient)

Given	condition	classification
quadrilaterals	- all sides equal - all angles equal	square.

- cannot prove positive results
- only can prove negative results, i.e.,
  ~~e.g.~~, if condition fails, not a square

Pumping Lemma: necessary condition. $\forall$ = for all

$\forall$ regular language, language is pumpable.

- To prove a language not regular, use pumping lemma.
  - show language NOT pumpable!

To prove condition fails (or not satisfied true)

Condition has	proof (condition fails)
$\forall$: for all cases/situations, something is true	– show one counter-example i.e. show one case is false.
e.g.) all sides are equal	–only need to show one side is different.
$\exists$: there exist one case, something is true	– prove all cases, something is false.
e.g., Given a number x, condition. $\exists$ a number $y \rightarrow 1<y<x$, y a factor of x. –classify x to be a composite number.	– prove not a composite number –need to prove all y, $1<y<x$, y is not a factor of x.

Pumping Lemma.

$\forall$ regular L, $\exists n$

$\forall z$, $z \in L$ and $|z| \geq n$.

$\exists u, v, w \rightarrow z = uvw$, $|uv| \leq n$, $|v| \geq 1$

$\forall i$, $uv^i w \in L$.

- necessary condition ($\because$ all regular L has pumping ability)
- but not sufficient

even if L can be pumped $\not\Rightarrow$ L is regular.

$\because$ not sufficient,

cannot use pumping lemma to prove L is regular.

$\because$ necessary,

can use pumping lemma to prove L is not regular.

Proof technique: proof-by-contradiction:

Assume L is regular,

by pumping lemma, L can be pumped.

But we prove that it is not possible

$\Rightarrow\Leftarrow$

$\Rightarrow$ assumption is wrong.

$\therefore$ L is not regular.

Pumping lemma: show L not regular.

Assume L regular

$\forall n$

$\exists z \nrightarrow z \in L$ and $|z| \geq n$.

$\forall u,v,w \nrightarrow z = uvw$, $|uv| \leq n$, $|v| \geq 1$,

$\exists i$, $uv^i w \notin L$.

$\Rightarrow\Leftarrow$

$\therefore L$ is not a regular language.

e.g., $L = \{a^k b^k \mid k \geq 1\}$ not regular.

Proof: Assume L regular, and n the constant.

Consider $z = a^n b^n$, $z \in L$, $|z| \geq n$.

[$\because |uv| \leq n$, both u, v are all a's.

$\forall u,v,w \nrightarrow z = uvw$, $|uv| \leq n$ and $|v| \geq 1$

$$z = \underbrace{u}_{a^r} \underbrace{v}_{a^s} \underbrace{w}_{a^{n-r-s}b^n} \qquad r+s \leq n,\ s \geq 1.$$

Let $i = 0$,

$$uv^0 w = a^r a^{n-r-s} b^n = a^{n-s} b^n \notin L \quad \because s \geq 1.$$

$\Rightarrow\Leftarrow$

$\therefore L$ is not regular.

— same proof for $L = \{$strings with an equal # of a's and b's$\}$

$L = \{ a^k b^k \mid k \geq 1 \}$ not regular

(i) $z = a^n b^n$
⋮

(ii) $z = a^{\frac{n}{2}} b^{\frac{n}{2}}$, $z \in L$, $|z| \geq n$.

$\forall u, v, w \rightarrow z = uvw$, $|uv| \leq n$, $|v| \geq 1$.
all possible decomposition patterns:

(a) v : only a's.
Let $i = 2$, $uv^2w \notin L$ ∵ # of a's > # of b's.

(b) v : only b's.
Let $i = 0$, $uw \notin L$ ∵ # of a's > # of b's.

(c) v : has both a's & b's.
Let $i = 2$, $uv^2w \notin L$ ∵ symbols out of order.

$\underbrace{a \cdots a}_{u} \underbrace{a \cdots a b \cdots b}_{v} \underbrace{a \cdots a b \cdots b}_{v} \underbrace{b \cdots b}_{w}$

$\Rightarrow\Leftarrow$

∴ L is not regular.

(iii) $z = a^k b^k$ where $k \geq \frac{n}{2}$.
e.g. $z = a^{\frac{n}{2}} b^{\frac{n}{2}}, a^{\frac{n}{2}+1} b^{\frac{n}{2}+1}, \ldots, a^n b^n, a^{n+1} b^{n+1}, \ldots$

- need proof (ii)

$L = \{$ equal #'s of a's & b's $\}$ not regular.

Proof: Assume L is regular. Let n be the p.l.'s constant.

(i) Let $z = a^n b^n$, $z \in L$, $|z| \geq n$

$\forall u, v, w \Rightarrow z = uvw$, $|uv| \leq n$, $|v| \geq 1$.

v must be purely a's.

Let $i = 2$, $uv^2w \notin L$ $\because$ #of a's > #of b's $\Rightarrow \neq$.

$\Rightarrow\Leftarrow$

$\therefore L$ is not regular.

(ii) Let $z = a^{\frac{n}{2}} b^{\frac{n}{2}}$, $z \in L$, $|z| \geq n$.

(a) v : purely a's.

(b) v : purely b's.

(c) v : has both a's & b's.

(1) unequal # of a's & b's.

(2) equal # of a's & b's.

$\forall i \geq 0$, $uv^iw \in L$ $\because$ increase/decrease equal #of a's & b's.

- $\exists$ decompositions : pumpable!

$\therefore$ proof fails.

(iii) $z = a^k b^k$ where $k \geq \frac{n}{2}$.

- same problem as (ii)

e.g., $L = \{0^{k^2} \mid k \text{ is an integer}, k \geq 1\}$ not regular.

Proof: Assume L regular, n the constant.

Let $z = 0^{n^2}$, $z \in L$, $|z| \geq n$.

$\forall u, v, w \Rightarrow z = uvw$, $|uv| \leq n$, $|v| \geq 1$,

— v are 0's and $1 \leq |v| \leq n$.

Let $i = 2$, $uv^2w = uvvw$

$|uv^2w| > n^2$ ($\because |v| \geq 1$)

$|uv^2w| \leq n^2 + n$ ($\because |v| \leq n$)

$< n^2 + 2n + 1 = (n+1)^2$.

$\therefore \quad n^2 < |uv^2w| < (n+1)^2$

$\Rightarrow \quad uv^2w \notin L$

$\Rightarrow\Leftarrow$

Hence, L is not regular.

e.g., $L = \{0^k 1^m 2^m \mid k \geq 1, m \geq 1\}$ not regular.

Proof: Assume L regular, n the constant.

1. wrong word: $z = 0^n 1^n 2^n$, $z \in L$, $|z| \geq n$.

$\exists u, v, w \Rightarrow |uv| \leq n$, $|v| > 1$:

$u = \varepsilon$, $v = 0$, $w = 0^{n-1} 1^n 2^n$

$\forall i$, $i \geq 0$, $uv^i w = 0^i 0^{n-1} 1^n 2^n \in L$, $n > 1$

2. Let $z = 0\,1^n 2^n$, $z \in L$, $|z| \geq n$.

$\forall u, v, w \Rightarrow z = uvw$, $|uv| \leq n$, $|v| \geq 1$:

(i) v contains a single type of symbol.

(a) v contains 0's only:

$u = \varepsilon$, $v = 0$, $w = 1^n 2^n$

~~For~~ for $i = 0$, $uv^0 w = 1^n 2^n \notin L$.

(b) v contains 1's only: $v = 1^l$, $1 \leq l \leq n$.

for $i = 2$, $uv^2 w = 0\,1^{n+l} 2^n \notin L$.

(c) v contains 2's only: impossible $\because |uv| \leq n$.

(ii) v contains more than 1 type of symbols:

only case: $v = 01^l$ ($v = 1^l 2^l$, $v = 01^n 2^l$: impossible)

for $i = 2$, $uv^2 w$: out of order 0's and 1's.

ie, 1's before 0's.

$\therefore uv^2 w \notin L$.

$\Rightarrow\Leftarrow$

Hence L is not regular.

Pigeon hole principle.

If there are $n+1$ pigeons but only n pigeon holes,
at least one pigeon hole must have $\geq$ two pigeons.

Pumping lemma for regular languages.

A regular language solved by a FA with n states,
given a string of length $\geq n$,
at least one state must repeat.

Pumping Lemma.

idea: DFA, n states

string z, $|z| \geq n$,

after $\delta(q_0, z)$, at least repeat one state.

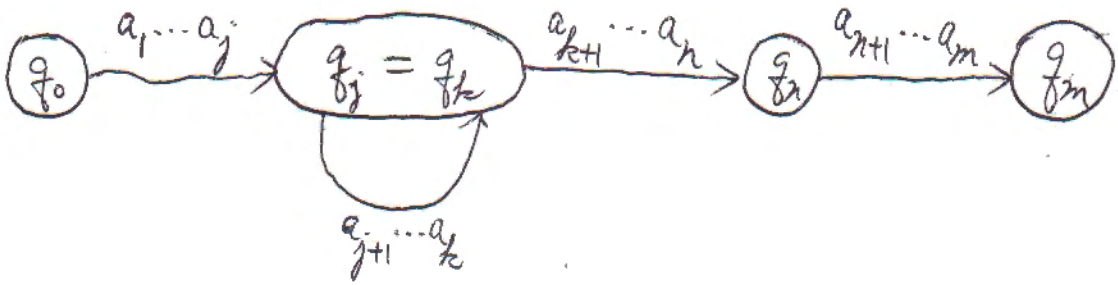
Formally. regular L, DFA, $M = (Q, \Sigma, \delta, q_0, F)$.

$|Q| = n$, $w = a_1 a_2 \cdots a_m$, $m \geq n$.

Let $\delta(q_0, a_1 a_2 \cdots a_i) = q_i$, $i = 1, 2, \ldots, m$.

- $q_0, q_1, q_2, \ldots, q_n$ cannot be all different/distinct.

$\therefore \exists j, k$, $0 \leq j < k \leq n \rightarrow q_j = q_k$.



$|a_{j+1} \cdots a_k| \geq 1$ ($\because j < k$)

$|a_1 \cdots a_k| \leq n$ ($\because k \leq n$)

$z = a_1 a_2 \cdots a_m \in L(M)$

then $a_1 \cdots a_j \; (a_{j+1} \cdots a_k)^i \; a_{k+1} \cdots a_m \in L(M)$, $i \geq 0$.

$a_1 \cdots a_j = u$, $(a_{j+1} \cdots a_k)^i = v^i$, $a_{k+1} \cdots a_m = w$

$z = uvw$, $|z| \geq n$, $|v| \geq 1$, $|uv| \leq n$.

Closure Properties of regular languages.

Th. If L and M are regular, then
$L \cup M$, $L \cdot M$, and L^* are also regular.
– by def. of operations $+, \cdot, *$ for regular expressions.

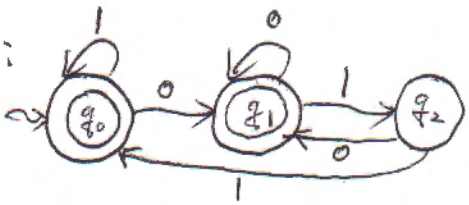
Th. If L is regular, then $\bar{L} = \Sigma^* - L$ is also regular.

Proof. DFA, $M = (Q, \Sigma, \delta, q_0, F) \rightarrow L = L(M)$.
DFA, $M' = (Q, \Sigma, \delta, q_0, Q-F)$ accepts $\bar{L}$.

e.g., $L(M) = \{$ strings that ends in $01\} = (0+1)^*01$

DFA, M:

DFA, M':



$L(M') = \{$ strings that do not end in $01\}$.
– shows a language regular.

eg., shows a language not regular.

problem. L_{eq} = { strings with an equal number of 0's and 1's }
- Not regular by pumping lemma.

$\overline{L_{eq}}$ = { strings with an unequal number of 0's and 1's }.

- pumping lemma does not work
  - if $z = uvw$ has both 0's and 1's
    $\exists\ u, v, w \nrightarrow v = 01$ (an equal # of 0's and 1's).
  - increase/decrease an equal # of 0's and 1's.
  - if z have an unequal # of 0's and 1's,
    - if z only have one kind of symbols, either 0's or 1's, so does uv^iw
      then $\exists u, v, w \nrightarrow v = 0$ or 1, and $uv^iw \in L$.
- use closure property for complement and proof by contradiction:

if $\overline{L_{eq}}$ is regular,

then $\overline{\overline{L_{eq}}} = L_{eq}$ is regular.

But since L_{eq} is not regular, $\Rightarrow\Leftarrow$

$\therefore$ $\overline{L_{eq}}$ is not regular.

Th. If L and M are regular, then $L \cap M$ is also regular.

Proof: i) By DeMorgan's Laws,

$$L \cap M = \overline{\bar{L} \cup \bar{M}}$$

ii) DFA, $M_1 = (Q, \Sigma, \delta_1, q_1, F_1) \Rightarrow L(M_1) = L$

DFA, $M_2 = (Q, \Sigma, \delta_2, q_2, F_2) \Rightarrow L(M_2) = M$

product DFA runs M_1, M_2 in parallel:

$$M' = (Q_1 \times Q_2, \Sigma, \delta, [q_1, q_2], F_1 \times F_2)$$

where

$$\delta([P_1, P_2], a) = [\delta_1(P_1, a), \delta_2(P_2, a)]$$

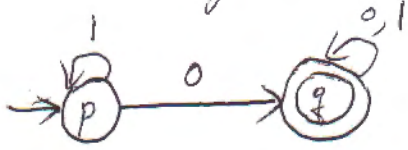
$P_1 \in Q_1$, $P_2 \in Q_2$, $a \in \Sigma$.

$$L(M') = L(M_1) \cap L(M_2).$$

— useful to solve problems with "and" conditions.

e.g., Problem. $L = \{$ strings that have both a 0 and a 1 $\}$.

DFA, M_1:



$L(M_1) = \{$ strings that have a 0 $\}$.

DFA, M_2:

$L(M_2) = \{$ strings that have a 1 $\}$.

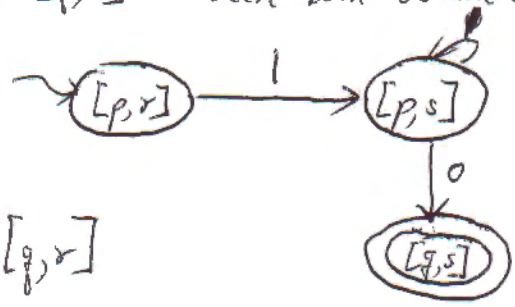
product DFA: $[p,r]$ = start state, seen neither 0 nor 1.

$\delta([p,r],1) = [\delta(p,1), \delta(r,1)] = [p,s]$.

$[p,s]$: seen only 1's.

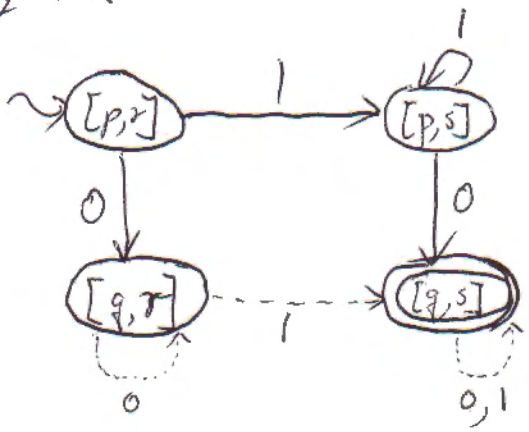
$\delta([p,s],1) = [\delta(p,1), \delta(s,1)] = [p,s]$ still seen only 1's.

$\delta([p,s],0) = [\delta(p,0), \delta(s,0)] = [q,s]$ seen both 0's and 1's.



$\delta([p,r],0) = [\delta(p,0), \delta(r,0)] = [q,r]$

$[q,r]$: seen only 0's.



Grammars (unrestricted)

$G = (V, T, P, S)$

V = a finite set of variables/non-terminals
T = a finite set of terminals , $V \cap T = \emptyset$
S = start symbol , $S \in V$
P = a finite set of production rules :
$\alpha \to \beta$ $\quad \alpha, \beta \in (V \cup T)^*$, $\alpha \neq \varepsilon$.

Def. If $\alpha \to \beta \in P$, then $\gamma \alpha \delta \underset{G}{\Rightarrow} \gamma \beta \delta$; $\alpha, \beta \in (V \cup T)^*$
— "directly" derives

Def. If $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_2$, then α_1 derives α_2 in grammar G

Def. If $S \overset{*}{\Rightarrow} \alpha$, then α is called a sentential form.

Def. If $S \overset{*}{\Rightarrow} w$, $w \in T^*$, then G generates w.

Def. $L(G) = \{ w \mid S \overset{*}{\Rightarrow} w \}$

Def. $G_1 \equiv G_2$ if $L(G_1) = L(G_2)$.

Regular Grammar

$G = (V, T, P, S)$

(i) left-linear grammar

P: $A \to w$ $\quad w \in T^*$, $A \in V$

$A \to Bw$ $\quad B \in V$

(ii) right-linear grammar

P: $A \to w$

$A \to wB$

e.g., $L = \{$strings ending in 00$\}$

DFA:

regular expression: $(0+1)^*00$

left-linear grammar: $G = (\{S, A\}, \{0, 1\}, P, S)$

P: $S \to A00 \mid 00$, $A \to A0 \mid A1 \mid 0 \mid 1$

right-linear grammar: $G = (\{S\}, \{0, 1\}, P, S)$

P: $S \to 0S \mid 1S \mid 00$

Th. If L has a regular grammar, then L is a regular set.

(a) right-linear grammar

Given $G = (V, T, P, S)$,

construct a NFA with ε-moves, M,

that simulates derivations in G:

$M = (Q, T, \delta, [S], \{[\varepsilon]\})$ where

$Q = \{ [\alpha] \mid \alpha$ is S or a suffix of some RHS of a production in $P\}$

δ: (i) if $A \to \alpha$,
then $\delta([A], \varepsilon) = \{[\alpha]\}$.

(ii) if $a \in T$,
then $\delta([a\alpha], a) = \{[\alpha]\}$

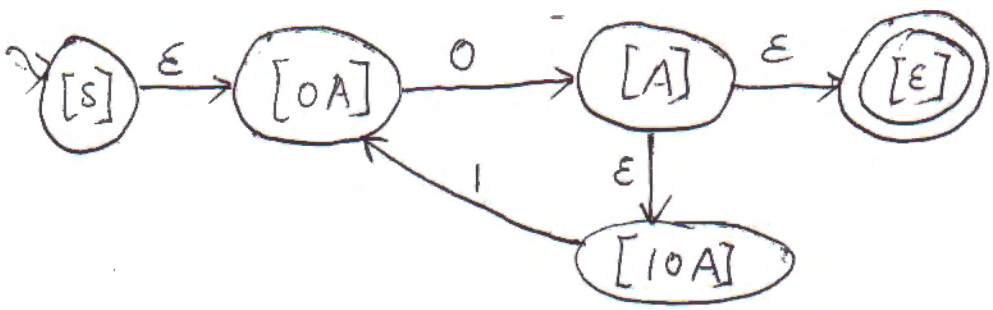
$\because$ right-linear, $\alpha \in T^* \cup T^*V$.

e.g., $L = 0(10)^*$

right-linear grammar: $S \to 0A$, $A \to 10A \mid \varepsilon$.

Q: $[S]$, $[0A]$, $[A]$, $[\varepsilon]$, $[10A]$

NFA with ε-moves:

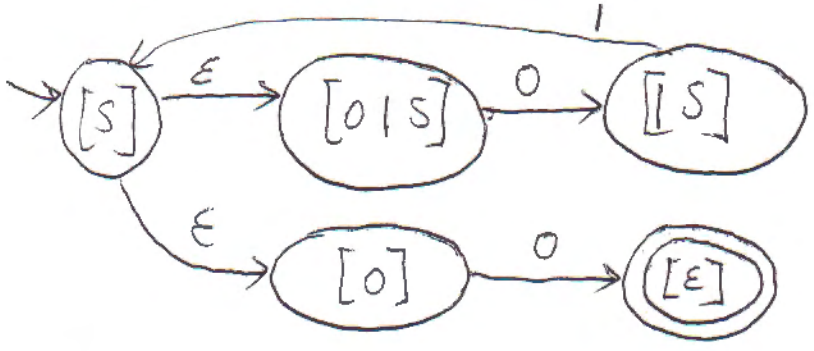


(b) left-linear grammar.

e.g., $S \to S10 \mid 0$

(1) reverse productions: $S \to 01S \mid 0$

- grammar for L^R
- right-linear
- NFA with ε-moves for L^R:

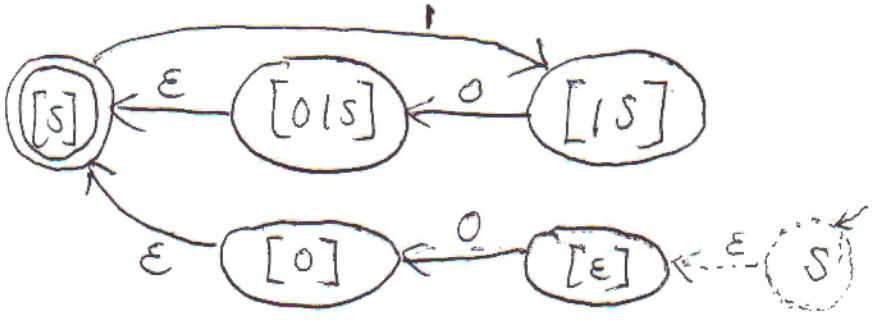


- L^R is regular.

(2): regular sets are closed under reversal.
∴ L is regular.

- reverse edges.
- start state ⇒ final state.
- ε-moves from new start state to all final states.

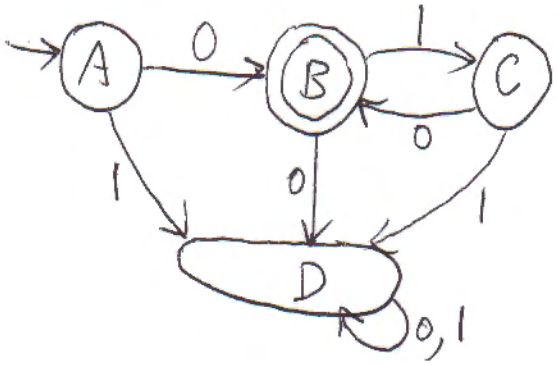
e.g.,



Th. If L is regular, then L is generated by a regular grammar.

(i) right-linear grammar.

e.g., given DFA for $L = 0(10)^*$:



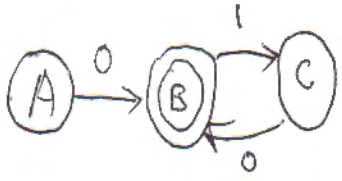
right-linear grammar: $A \equiv S$.

$$A \to 0B \mid 0 \mid 1D$$
$$B \to 1C \mid 0D$$
$$C \to 0B \mid 0 \mid 1D$$
$$D \to 0D \mid 1D$$

— D is useless

$$\therefore \quad A \to 0B \mid 0$$
$$B \to 1C$$
$$C \to 0B \mid 0$$

$\equiv$



Given DFA, $M = (Q, \Sigma, \delta, q_0, F)$

(i) right-linear grammar

$G = (Q, \Sigma, P, q_0)$ where P:

if $\delta(p, a) = q$, then $p \to aq$

if $\delta(p, a) \in F$, add: $p \to a$.

(a) if $q_0 \notin F$, then $L(M) = L(G)$.

(b) if $q_0 \in F$, $\Rightarrow \varepsilon \in L(M)$.

— add a new start symbol, S,
and $S \to q_0 \mid \varepsilon$.

(ii) construct left-linear grammar.

- given DFA for L
- NFA with ε-moves for L^R:
  reverse edges, start, final states.
- convert to DFA for L^R
- construct right-linear grammar for L^R
- reverse the RHS of all productions
  — left-linear grammar for L.

Context-free grammars.

$G = (V, T, P, S)$

P: $A \to \alpha$, $A \in V$, $\alpha \in (V \cup T)^*$

e.g., $L = \{a^n b^n \mid n \geq 1\}$

$G = (\{S\}, \{a, b\}, P, S)$, P: $S \to aSb \mid ab$

e.g., $L = \{a^{2n} b^{3n} \mid n \geq 1\}$

$G = (\{S\}, \{a, b\}, P, S)$, P: $S \to aaSbbb \mid aabbb$

e.g., $L = \{wcw^R \mid w \in (0+1)^*\}$

$G = (\{S\}, \{0, 1, c\}, P, S)$, P: $S \to c \mid 0S0 \mid 1S1$

e.g., $L = \{ww^R \mid w \in (0+1)^*\}$

$G = (\{S\}, \{0, 1\}, P, S)$, P: $S \to \varepsilon \mid 0S0 \mid 1S1$

- derivation trees
- leftmost/rightmost derivation
- ambiguous , inherently ambiguous.

Def. A symbol X is useful if
$S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$ exists.
where $\alpha, \beta \in (V \cup T)^*$ and $w \in T^*$.
Otherwise, X is useless.

Th. Every non-empty CFL is generated by a CFG with no useless symbols.

Steps 1. remove symbols that do not derive terminal strings: $B \overset{*}{\not\Rightarrow} w$.
2. remove symbols that cannot be reached from S: $S \overset{*}{\not\Rightarrow} \alpha B \beta$.

e.g., $S \to AB \mid a$, $A \to a$.
1. B derives no terminal string, eliminate B
$\therefore S \to a$, $A \to a$.
2. A cannot be derived from S, remove A.
$\therefore S \to a$.

Steps reversed: 2. All variables can be derived from S, keep all.
$\therefore S \to AB \mid a$, $A \to a$.
1. B derives no terminal string, remove B.
$\therefore S \to a$, $A \to a$.

Problem: – although A derives terminal string,
– but it can only be derived from S in conjunction with B, a variable that does not derive terminal strings.

Lemma. Given a CFG, $G=(V, T, P, S)$ with $L(G)\neq\emptyset$, we can ~~effectively~~ find an equivalent CFG, $G'=(V', T, P', S)$ $\ni$ for each $A\in V'$, $\exists\ w\in T^*$, for which $A \overset{*}{\Rightarrow} w$.

Alg.

$oldV \leftarrow \emptyset$
$newV \leftarrow \{ A \mid A\to w,\ w\in T^* \}$
while $oldV \neq newV$ **do**
  $oldV \leftarrow newV$
  $newV \leftarrow oldV \cup \{ A \mid A\to\alpha,\ \alpha\in(T\cup oldV)^* \}$
end
$V' \leftarrow newV$.

e.g., $S\to AB \mid a \mid CD$, $A\to a$, $C\to SD$, $D\to b$.

$oldV$	$\emptyset$	$\Rightarrow$	$\{S, A, D\}$	$\Rightarrow$ $\{S, A, C, D\}$
$newV$	$\{S, A, D\}$		$\{S, A, D, C\}$	$\{S, A, C, D\}$

– B cannot derive terminal string, remove B

$\therefore\ S\to a \mid CD$, $A\to a$, $C\to SD$, $D\to b$.

Lemma. Given a CFG, $G=(V,T,P,S)$, we can effectively find an equivalent CFG, $G'=(V',T',P',S)$ s.t. for each $X\in V'$, $\exists\ \alpha,\beta\in(V'\cup T')^*$ for which $S\overset{*}{\Rightarrow}\alpha X\beta$.

Alg. $V'\leftarrow\{S\}$, $T'\leftarrow\emptyset$

repeat

if $A\in V'$ and $A\to\alpha_1|\alpha_2|\cdots|\alpha_n$ then

$V'\leftarrow V'\cup\{\text{all variables in } \alpha_1,\alpha_2,\ldots,\alpha_n\}$

$T'\leftarrow T'\cup\{\text{all terminals in } \alpha_1,\alpha_2,\ldots,\alpha_n\}$

end

until V' is unchanged.

e.g., $S\to a|CD$, $A\to a$, $C\to SD$, $D\to b$.

V'	$\{S\}$ $\Rightarrow$	$\{S,C,D\}$ $\Rightarrow$	$\{S,C,D\}$
T'	$\emptyset$	$\{a\}$	$\{a,b\}$.

A cannot be reached from S, remove A.

i.e. $S\to a|CD$, $C\to SD$, $D\to b$.

Def. For variable A, if $A \overset{*}{\Rightarrow} \varepsilon$, then A is nullable.

Th. If $L = L(G)$ for some CFG, $G = (V, T, P, S)$, then $L - \{\varepsilon\} = L(G')$, G' a CFG with no useless symbols, and ε-productions.

Alg. 1. Determine nullable symbols of G.
2. remove ε-productions:
idea: e.g., $S \to CD$, $C \to \varepsilon$, $D \to \varepsilon$, $C \to \cdots$, $D \to \cdots$, ...

$\Downarrow$

$S \to CD \mid C \mid D \mid (\varepsilon)$ → remove

3. produce G' with no useless symbols.

1. Determine nullable symbols.
Alg. - if $A \to \varepsilon$, then A is nullable
- if $B \to \alpha$, and all symbols of α are nullable, then B is nullable.
- repeat until no more nullable symbols.

e.g., $S \to ABC \mid a \mid CD$, $A \to a$, $B \to b \mid CS$, $C \to c \mid \varepsilon$, $D \to d \mid \varepsilon$

$\therefore$ $C \to \varepsilon$, $D \to \varepsilon$, C, D are nullable
$S \to CD$, S is nullable
$B \to CS$, B is nullable

nullable symbols: S, B, C, D.

2. remove ε-productions, create G'' for $L-\{\varepsilon\}$.

$G'' = (V, T, P'', S)$

$\forall A \in V$, $w \in T^*$, $A \overset{*}{\underset{G''}{\Rightarrow}} w$ iff $A \overset{*}{\underset{G}{\Rightarrow}} w$ and $w \neq \varepsilon$.

Rules P': If $A \to X_1 X_2 \cdots X_n$ is in P,
then add $A \to \alpha_1 \alpha_2 \cdots \alpha_n$ to P'' where

(1) if X_i is not nullable, then $\alpha_i = X_i$.

(2) if X_i is nullable, then $\alpha_i = X_i$ or ε

(3) not all α_i's are ε.

e.g., $S \to ABC \mid a \mid CD$, $A \to a$, $B \to b \mid CS$, $C \to c \mid \varepsilon$, $D \to d \mid \varepsilon$

nullable symbols: S, B, C, D.

$S \to ABC$: $S \to ABC \mid AB \mid AC \mid A$

$S \to a$

$S \to CD$: $S \to CD \mid C \mid D \mid \varepsilon$

$A \to a$, $B \to b$

$B \to CS$: $B \to CS \mid C \mid S$

$C \to c$, $D \to d$.

$C \to \varepsilon$, $D \to \varepsilon$ eliminated.

Def. if $A, B \in V$, unit productions are productions of the form $A \to B$.
All other productions, including $A \to a$ and $B \to \varepsilon$, are non-unit productions.

Th. Every CFL without ε is defined by a grammar with no useless symbols, ε-productions, and unit productions.

Alg. 1. remove ε-productions.
2. remove unit productions.
3. remove useless symbols.

2. Given a CFG $G = (V, T, P, S)$, G has no ε-productions.
$L = L(G)$ is a CFL without ε.

Construct P' from P:
- include all non-unit productions of P.
- for $A, B \in V$, if $A \overset{*}{\Rightarrow}_G B$, add $A \to \alpha$ if $B \to \alpha$ is a non-unit production of P.
- to test whether $A \overset{*}{\Rightarrow}_G B$, sufficient to consider only those unit-production sequences (∵ no ε-productions) that do not repeat variables.

- only unit-production sequences:
∵ $A \Rightarrow CB \Rightarrow B$ (∵ no $\varepsilon \to \varepsilon$).

- no repeat variables:
∵ if $A \Rightarrow E \Rightarrow C \Rightarrow D \Rightarrow E \Rightarrow B$
then $A \Rightarrow E \Rightarrow B$
- finite # of sequences.

eg, P: $S \to ABC \mid AB \mid AC \mid A$, $S \to a$, $S \to CD \mid C \mid D$.
$A \to a$, $B \to b$, $B \to CS \mid C \mid S$, $C \to c$, $D \to d$.

P': $S' \to ABC \mid AB \mid AC \mid a \mid CD$
$A \to a$, $B \to b \mid CS$, $C \to c$, $D \to d$.

$S \Rightarrow A$: $S \to a$
$S \Rightarrow C$: $S \to c$
$S \Rightarrow D$: $S \to d$.
$B \Rightarrow C$: $B \to c$
$B \Rightarrow S$: $B \to ABC \mid AB \mid AC \mid a \mid CD$
$B \Rightarrow S \Rightarrow A$: $B \to a$ already have
$B \Rightarrow S \Rightarrow C$: $B \to c$ "
$B \Rightarrow S \Rightarrow D$: $B \to d$.

Chomsky Normal Form. (CNF)

Any CFL without ε is generated by a grammar in CNF.

- generate Greibach Normal Form (GNF)
- prove pumping lemma.

GNF : prove CFG $\equiv$ PDA

Given : any CFG for $L - \{\varepsilon\}$

Alg. : 1. remove ε-productions
2. remove unit productions
3. remove useless symbols
4. convert to CNF
   (introduce variables to embedded terminals,
   reduce # of variables to 2)

Chomsky Normal Form (CNF).

- generate Greibach Normal Form (GNF)
- prove pumping lemma.

GNF : prove CFG $\equiv$ PDA.

Th. Any CFL without ε is generated by a grammar in which all productions are of the form:
$A \to BC$ or $A \to a$, where $A, B, C \in V$, $a \in T$.

Alg. Given CFG, G : no useless symbols, ε-productions, unit-productions
$L(G)$: no ε.

- if a single symbol on the right, must be a terminal
  $A \to a$, $a \in T$.
- if $A \to X_1 X_2 \cdots X_m$, $m \geq 2$
  - if X_i is a terminal, a,
    introduce C_a, $C_a \to a$
    replace X_i by C_a.
- Now either $A \to a$ or $A \to B_1 B_2 \cdots B_m$, $m \geq 2$: B_i are variables.
  - if $A \to B_1 B_2 \cdots B_m$, $m \geq 3$:
    create new variables $D_1, D_2, \ldots, D_{m-2}$
    replace $A \to B_1 B_2 \cdots B_m$ by
    $\{A \to B_1 D_1, D_1 \to B_2 D_2, \ldots, D_{m-3} \to B_{m-2} D_{m-2}, D_{m-2} \to B_{m-1} B_m\}$.

e.g.,

$$S \to bA \mid aB$$
$$A \to bAA \mid aS \mid a$$
$$B \to aBB \mid bS \mid b.$$

$$S \to C_bA \mid C_aB, \quad C_a \to a, \quad C_b \to b.$$
$$A \to C_bAA \mid C_aS \mid a$$
$$B \to C_aBB \mid C_bS \mid b.$$

$$S \to C_bA \mid C_aB, \quad C_a \to a, \quad C_b \to b.$$

$$A \to C_bD_1, \quad D_1 \to AA$$
$$A \to C_aS \mid a$$

$$B \to C_aD_2, \quad D_2 \to BB$$
$$B \to C_bS \mid b$$

Greibach Normal Form (GNF).

Every CFL without ε can be generated by a grammar in which every production is of the form $A \to a\alpha$ where $A \in V$, $a \in T$, $\alpha \in V^*$.