# Formal Language Theory

Lecture Tue Feb 24



$A \cdot B = \{ x \cdot y : x \in A \ \& \ y \in B \}$. ($\emptyset$ if FALSE)

There is nondeterminism in how the string w might be broken.

$\alpha \cdot \beta = \{ w : w$ can be broken as $w =: x \cdot y$ such that $x \in A$ & $y \in B \}$.

⚠ $A \cdot B \neq B \cdot A$ in general (as with matrices)

$A^2 =_{def} A \cdot A = \{ x \cdot y : x \in A$ & $y \in A \}$ NOT $\{ x \cdot x : x \in A \}$

$x^2 = xx$

$A \cdot A = A \cdot A = \{ w : w$ can be broken as $w =: xy$ st $x \in A$ & $y \in A \}$.

$A \cdot (B \cdot C) = (A \cdot B) \cdot C$

$$A^* = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \cdots = \bigcup_{i=0}^{\infty} A^i$$

Rule: For any language A, even $A = \emptyset$, $A^0 = \{\varepsilon\}$. Like $a^0 = 1$ in math.

Rule: For all languages A, $A \cdot \emptyset = \emptyset$ and $A \cdot \{\varepsilon\} = A$ (char) $C^0 = \varepsilon$

$\emptyset \cdot A = \emptyset$ and $\{\varepsilon\} \cdot A = A$

For all languages A, B, and C:

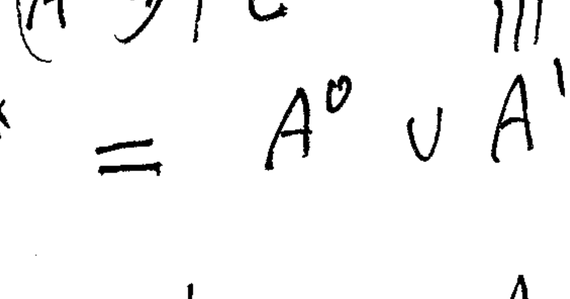
Distributive Law: $A \cdot (B \cup C) = A \cdot B \cup A \cdot C$ and $(A \cup B) \cdot C = A \cdot C \cup B \cdot C$

With regexps $\alpha, \beta, \gamma$ $\alpha \cdot (\beta + \gamma) = \alpha\beta + \alpha\gamma$ $(\alpha + \beta) \cdot \gamma = \alpha\gamma + \beta\gamma$

With regexps r, t, u $r(t + u) = rt + ru$ Main Difference: $r + r = r$ $A \cup A = A$.

An example of a totally new kind of rule: $(\varepsilon + r)^* = r^*$ $(\{\varepsilon\} \cup A)^* = A^*$

Example




$L_{11} = \{ x \in \{0,1\}^* : \#0(x)$ is even$\}$ $L_{11} = (L_{11}^{\text{once}})^*$

$L_{12} = \{ x \in \{0,1\}^* : \#0(x)$ is odd$\}$.

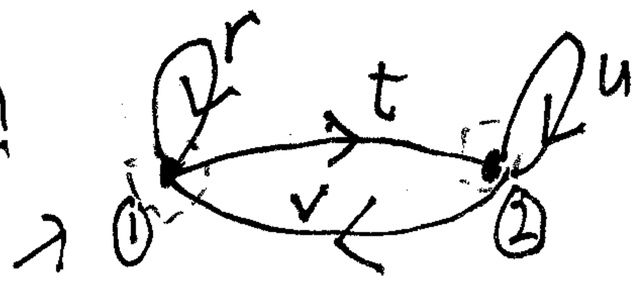
$\{\varepsilon\}^* = \{\varepsilon\}$ $\emptyset^* = \{\varepsilon\}$

Rule: For any state p, $L_{pp} = r^*$ for some regexp r.

$L_{11}^{\text{once}} = (1 + 01^*0)$ $\therefore L_{11} = (1 + 01^*0)^*$. Is equiv to $1^*(01^*01^*)^*$

~~$L_{12}^{\text{once}}$~~ $L_{12} = L_{11}^{\text{once}} \cdot 01^* = (1 + 01^*0)^*01^* = L_{12}^{\text{first}} \cdot L_{22} = 1^*0 \cdot (L_{22}^{\text{once}})^*$

The Abstract 2-State GNFA




$L_{11} = (r + tu^*v)^*$ $L_{12} = L_{11}tu^* = (r + tu^*v)^*tu^*$

$L_{22} = (u + vr^*t)^*$ $L_{21} = L_{22}vr^* = (u + vr^*t)^*vr^*$.

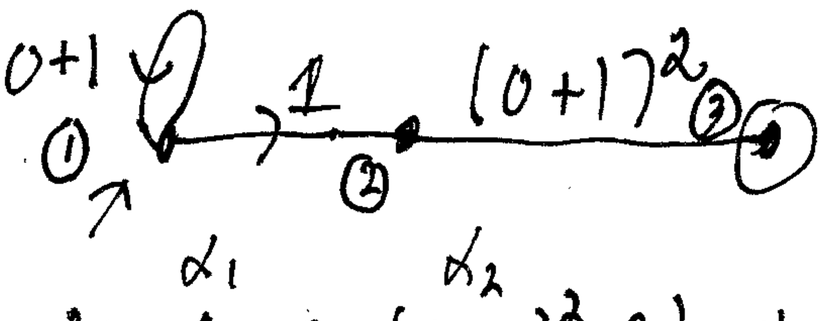
Defn: A Generalized NFA (GNFA) is a 5-tuple $N=(Q,\Sigma,\delta,s,F)$ which is like an NFA except $\delta \subseteq Q \times \underline{\text{Regexps}(\Sigma)} \times Q$. $(p,\alpha,q)$ where $\alpha$ is a regexp.

In fact, an NFA is just the case $\delta \subseteq Q \times \text{BasicRegexps}(\Sigma) \times Q$ since $(p,\varepsilon,q)$ involve the basic $(p,c,q)$ regexps $\varepsilon$ and $c$ while $(p,\emptyset,q)$ means "no instruction".

Defn: A computation path from a state $p$ to a state $q$ is a sequence $(p=q_0, \alpha_1, q_1, \alpha_2, q_2, \alpha_3, q_3, \ldots, q_{m-1}, \alpha_m, q_m=q)$ such that for all $i$, $1 \le i \le m$, $(q_{i-1}, \alpha_i, q_i)$ is an instruction in $\delta$.
(This computation path of)
N can process a string $w \in \Sigma^*$ from $p$ to $q$ if $w$ can be broken into $m$ substrings $w =: u_1 \cdot u_2 \cdot u_3 \cdots u_m$ such that for each $i$, $1 \le i \le m$ $u_i \in L(\alpha_i)$ ($u_i$ matches $\alpha_i$)

Finally, $\underline{\underline{L_{pq}}} = \{w : N \text{ can process } w \text{ from } p \text{ to } q\}$.

An old example in this new form:




$(0+1)^* 1 (0+1)^2 =$
$(0+1)^* 1 (00+01+10+11)$

$w = 1101$

To witness this defn, if I try path $= (1, \underline{1}, 2, (0+1)^2, 3)$ then I fail.
$w = u_1 \cdot u_2$ with $u_1 = 1$ and $u_2 = 101 \notin L((0+1)^2)$.

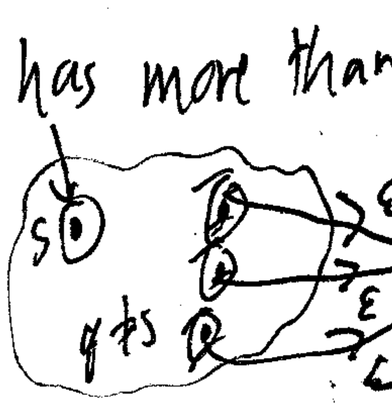
$$L(N) = \bigcup_{q \in F} L_{sq} = \text{just } L_{13}.$$

Good path: $(1, 0+1, 1, \underline{1}, 2, (0+1)^2, 3)$
Matched with $w = \underline{1} \cdot \underline{1} \cdot \underline{01}$ $\quad u_1 = 1 \quad u_2 = 1 \quad u_3 = 01$

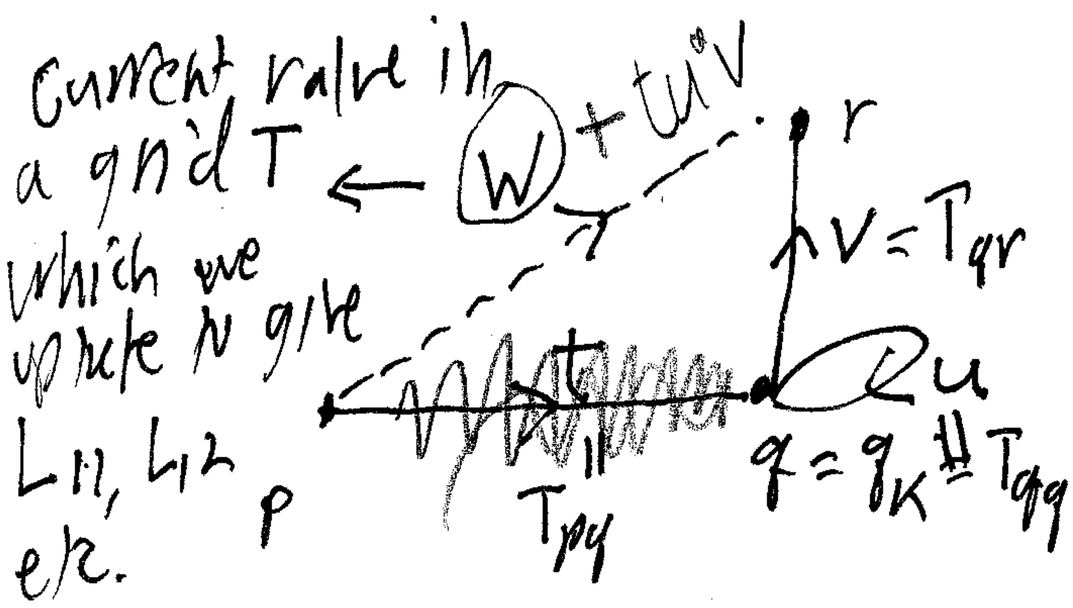
Theorem: Given any DFA or NFA or GNFA $N$, we can compute a regexp $\alpha$ such that $L(N) = L(\alpha)$.
Text target $s \xrightarrow{\alpha} f$

Proof: We have already proved this for $K \le 2$, where $N = (Q,\Sigma,\delta,s,F)$ and $K = |Q|$.

Given $N$ with $K \ge 2$, if $N$ has more than one accepting state $q \ne s$, then add a new accepting state $f$ and $\varepsilon$-arcs from all $q \in F$ to $f$



Hence we may suppose $N$ is in well-grounded form. (maybe old $K$ became $K+1$)
Number $S=1$, and the final states $2$, $\{1,2\}$, or just $1$ if $F=\{S\}$.
Thus states $3,\dots,K$ are non-accepting. We will <u>eliminate</u> them one-by-one.




To eliminate the highest-numbered state $q$, we <u>bypass</u> all arcs $(p,t,q)$ into $q$.
To <u>bypass</u> $(p,t,q)$, for all <u>outgoing</u> arcs $(q,v,r)$, we <u>update</u> the direct path from $p$ to $r$ by $T_{pr} \mathrel{+}= tu^*v$. Then we can delete $(p,t,q)$.

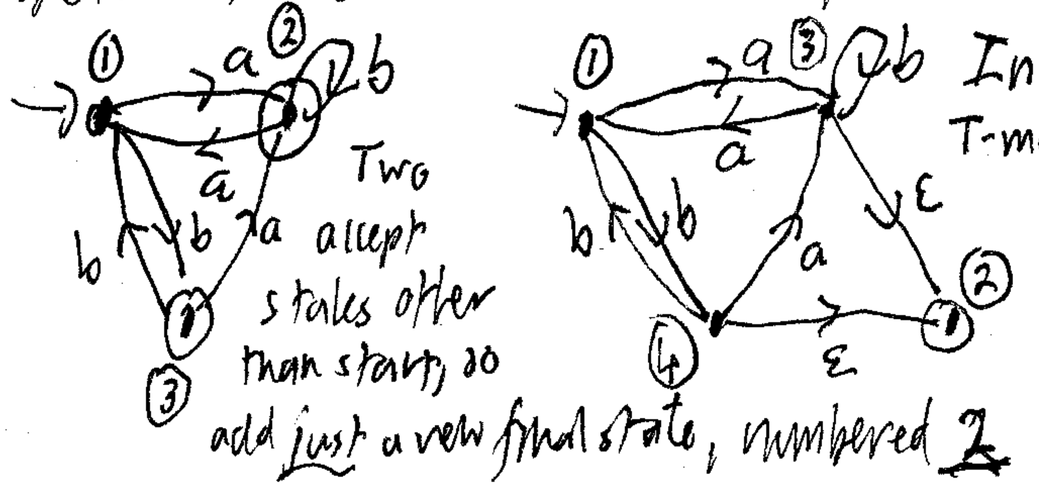
$T_{pr}=W$ when all incoming arcs to $q$ have been bypassed, then we <u>delete</u> $q$. New $T_{pr} = W + tu^*v$.

```
for (int j = K; j ≥ 3; j--) {  // eliminate state j  ie state qj
    for (int i=1; i ≤ j-1; i++) {
        if (there is an instruction (i, t, j) in δ) {
            for (int h = 1; h ≤ j-1; h++) {
                if (there is an instruction (j, v, h)) {
                    T_{i,h} += T_{ij} T_{jj}* T_{jh}.
                }
            }
        }
    }
```

Read off final answer from $K=2$ case after outer loop exits.

(that is, new $T_{ih}$ = old $T_{ih} \cup T_{ij} T_{jj}^* T_{jh}$)

**Extra**: You can regard $T$ as a "matrix of regular expressions" if you think "better in code" than "in pictures." E.g. for my version of the text's example on p76:




Initial T-matrix:

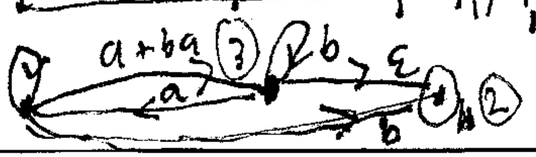
| | 1 | (2) | 3 | 4 |
|---|---|---|---|---|
| 1 | ∅ | ∅ | a | b |
| (2) | ∅ | ∅ | ∅ | ∅ |
| 3 | a | ε | b | ∅ |
| 4 | b | ε | a | ∅ |

| | 1 | (2) | 3 |
|---|---|---|---|
| 1 | bb | b | a+ba |
| (2) | ∅ | ∅ | ∅ |
| 3 | a | ε | b |

$b + (a+ba)b^*\cdot\varepsilon$

| | |
|---|---|
| $bb + (a+ba)b^*a$ | ↓ |
| ∅ | ∅ |

$T_{11}^* T_{12}$ = final answer.



Example (text w/o adding new start state, still remembering new acc state ④ & 2)

Elim state 4: Incoming: Just (①, b, 4)

Outgoing: (4, b, ①) ∴ Update (1,1) (1, —, 1)

(4, a, ③) Update (1, —, 3)

(4, ε, 2) T(4,4) = ∅ but T(4,4)* = ∅* = ε.

Bypass (1, b, 4) back to 1:

New T(1,1) = old T(1,1) ∪ T(1,4) T(4,4)* T(4,1).

= ∅ ∪ b · ε · b = bb

or = ε ∪ b · ε · b = ε + bb

You can also initialize T(1,1) to ε, likewise T(i,i) for all i.

⚠ ★ Both answers will work because the ultimate expression will have T(1,1)* and (bb)* = (ε + bb)* as a component term.

Bypass (1,b,4) to 3

New T(1,3) = old T(1,3) ∪ T(1,4) T(4,4)* T(4,3)

= a ∪ b · ε · a = a + ba

Bypass (1,b,4) to ②.

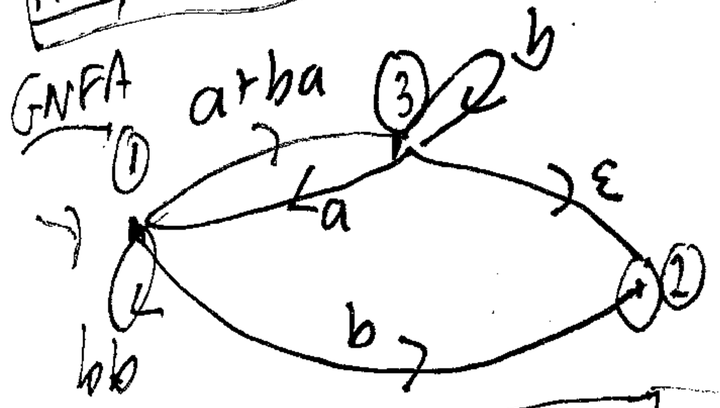
new T(1,2) = old T(1,2) ∪ T(1,4) T(4,4)* T(4,2)

= ∅ ∪ b · ε · ε = b.

Now elim ④

We CANNOT replace this ∅ by ε because dest state 2 ≠ origin 1.

GNFA




Elim 3: Incoming: only ① (①, —, 3)

outgoing: (3, —, ①) ∴ Update T(1,1) again

: (3, ε, ②) Update T(1,2).

Bypass (1, a+ba, 3) back to 1:

New T(1,1) = old T(1,1) ∪ T(1,3) T(3,3)* T(3,1).

bb ∪ (a+ba) · b* · a = bb + (a+ba) b*a

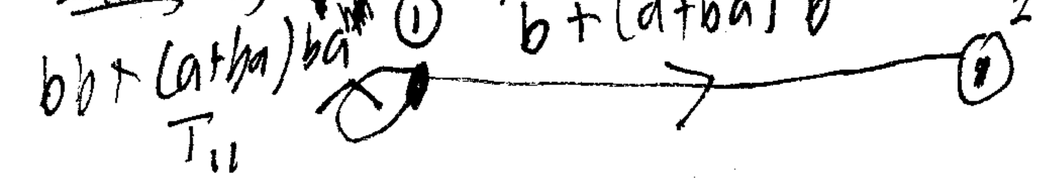
or = ε + bb + (a+ba) b*a

Bypass (1, a+ba, 3) to 2:

New T(1,2) = old T(1,2) ∪ T(1,3) T(3,3)* T(3,2)

= b ∪ (a+ba) b* · ε = b + (a+ba) b*

Elim 3, redraw:




$L_{11} < T_{11}^* = (r^* \text{ above}) = (bb + (a+ba)\,b^*a)^*$ (putting ε + here makes no difference)

$L(M) = L_{12} = L_{11} \cdot T_{12} = (bb + (a+ba)\,b^*a)^* \cdot (b + (a+ba)\,b^*)$

The last $3\frac{1}{2}$ lectures have proved a theorem first obtained by Stephen Kleene in the mid-1950s:

**Theorem**: For any language $L \subseteq \Sigma^*$ (any $\Sigma$):

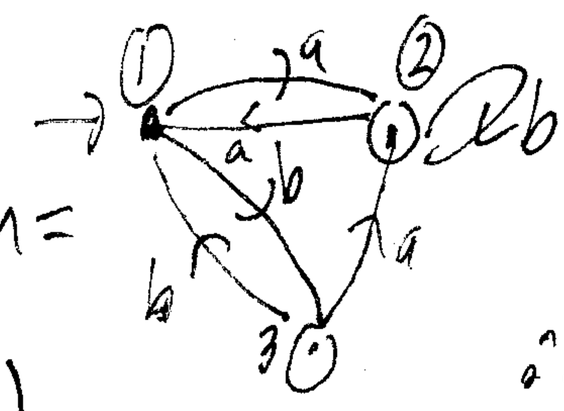
(a) there is a DFA $M$ s.t. $L = L(M)$
$\Leftrightarrow$ (b) there is an NFA $N$ s.t. $L = L(N)$ (GNFA ditto)
$\Leftrightarrow$ (c) there is a regular expression $r$ s.t. $L = L(r)$

} L is a regular language if any holds.

**When** is a language $L$ (non-) regular?

Related: **Why** is a regular $L$ regular?

Let's revisit the original DFA

$M =$ 


Let $L = L(M)$

Note that $a$ and $ba$ both get processed from 1 to 2.

∴ For any suffix string $z$, $az$ and $baz$ get processed the same route.

∴ $(\forall z \in \Sigma^*)\ az \in L(M) \Leftrightarrow baz \in L(M)$.

If $z = \varepsilon$, or, if $z = ab$, they're both in $L(M)$
if $z = a$ they're both out: $aa, baa \notin L(M)$.

If $x$ and $y$ are any two strings such that for some $z \in \Sigma^*$

$xz \in L \wedge yz \notin L$

or $xz \notin L \wedge yz \in L$

$xz \in L \not\Leftrightarrow yz \in L$

i.e. $xz \in L$ XOR $yz \in L$, then $x$ and $y$ CANNOT be processed to the same state

i.e. $L(xz) \neq L(yz)$ I.e. they must be processed to different states

∴ Hence, $M$ must have at least 2 states.

Suppose $S$ is a set of strings such that for any distinct $x, y \in S$ there exists $z$ s.t. $L(xz) \neq L(yz)$. [s.t. $L(M) = L$] different states

Then for any $x, y \in S$ $(x \neq y)$, any DFA $M_\wedge$ must process $x, y$ to $\wedge$.

∴ If $S$ has $K$ strings, then any DFA $M_\wedge$ must have at least $K$ states.

∴ If $S$ has $\infty$-many strings, then any DFA $M$ s.t. $L = L(M)$ must have at least $\infty$-many states. Hence $L$ has no finite DFA, so $L$ is not regular.

∴ Theorem (Half of the Myhill-Nerode Thm): Suppose $L$ is a language and $S$ is a set of strings s.t. • $S$ is infinite, and • for all $x, y \in S$ $(x \neq y)$ $(\exists z \in \Sigma^*)$ $L(xz) \neq L(yz)$.

Then $L$ is not regular.

Example: $L = \{a^n b^n : n \geq 0\}$. To prove $L$ is not regular:

Take $S = a^* = \{a^n : n \geq 0\}$. Clearly $S$ is infinite

Let any $x, y \in S$, $x \neq y$, be given. Then there are numbers $m, n \geq 0$, $m \neq n$, s.t. $x = a^m$ and $y = a^n$.

Take $z = b^m$.

Then $xz = a^m b^m \in L$, but $yz = a^n b^m \notin L$ since $m \neq n$.

∴ $L(xz) \neq L(yz)$, and since $x, y \in S$ are arbitrary, $S$ is an infinite PD set for $L$, ∴ $L$ is not regular ☒

Extra: The other half of the theorem: If $L$ is not regular, then there is an infinite set $S$ such that $(\forall x, y \in S, x \neq y)(\exists z \in \Sigma^*)$ $L(xz) \neq L(yz)$ (i.e. such that $S$ is PD for $L$.) This means that in principle every nonregular language has some kind of Myhill-Nerode proof.

The contrapositive of this is: If every PD set $S$ for $L$ is finite, then $L$ is regular. We can prove this – which completes both halves of the theorem – without going into quite as much detail as the text in problem 1.52. Consider the following unbounded "allocation" process:

For string $x = \varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$ in order, (having initialized $S = \emptyset$) {
  If $(\exists y < x)(\forall z \in \Sigma^*)$ $L(yz) = L(xz)$ then define State$(x)$ = State$(y)$.
  Else allocate State$(x)$ as a new state and put $S := S \cup \{x\}$; // Invariant: $S$ is PD for $L$
  If $(x = 1^n)$ then for all $w$ of length $n-1$ and $c \in \Sigma$, define delta$(w, c)$ = State$(wc)$;
}

Note: $S$ always gets $\varepsilon$ right away

By hypothesis, $S$ stays finite, and so $(S, \Sigma, \text{delta}, \text{State}(\varepsilon), F)$ is a DFA $M$ s.t. $L(M) = L$, where $F = \{\text{State}(x) : x \in L\}$!