Lecture 5

Tuesday, 9 February, 2021 10:56

Last time:

$\exists$ DFA M
$L = L(M)$

- If L is automatic then there is an NFA N so that ~~that~~ $L = L(N)$

  Pf. M can be interpreted as N

- If L is regular then there is an NFA N so that $L = L(N)$

  $\exists$ regex r
  $L = L(r)$

  Pf. Apply Thompson's alg

DFA $\longrightarrow$ NFA $\longleftarrow$ regex

today $\longleftarrow$ $\longrightarrow$

In conclusion: auto $\longleftrightarrow$ reg

NFA $\to$ DFA: subset construction
(power set)

Idea: in prod construction: simulated being in two machines at once.

$\to$ here: build a DFA that simulates nondeterminism in the NFA by keeping track of all possible states we can be in

every subset of Q represents a possible set of states we can be in simultaneously.

Given NFA $N = (Q, \Sigma, \delta, s, A)$

Given NFA $N = (Q, \Sigma, \delta, s, A)$

build DFA: $M = (Q', \Sigma, \delta', s', A')$

$Q' = P(Q)$ ← power set.

$s' = \cancel{\{s\}}$ $\varepsilon reach(s)$

$A' = \{ T \subseteq Q \mid T \cap A \neq \phi \}$

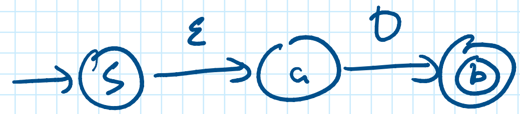
$\{ T \in P(Q) = Q' \}$

NFA acceptance:

$\delta^*(s, w) \cap A \neq \phi$

$$\delta'(T, a) = \bigcup_{q \in T} \delta^*(q, a)$$

NFA states



$\bigcup_{q \in \{s\}} \delta(q, 0) \neq$

$\delta'(\{s\}, 0) = \{b\}$

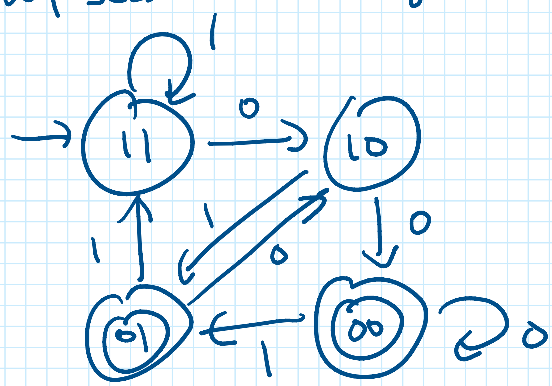
in NFA:

$$\delta^*(q, w) = \begin{cases} \varepsilon reach(q) & \text{if } w = \varepsilon \\ \varepsilon reach\left(\bigcup_{r \in \varepsilon reach(q)} \delta(r, a) \right) & \text{if } w = ax \end{cases}$$

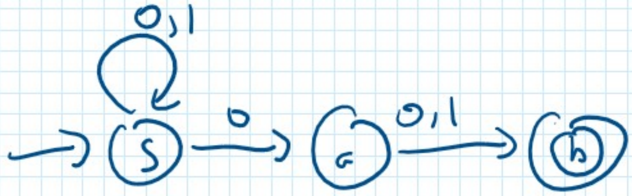
$\varepsilon reach(q)$
= set of all states reachable from q by following ε-transitions

$L = \{ w \mid \text{second to last symbol in } w \text{ is } 0 \}$

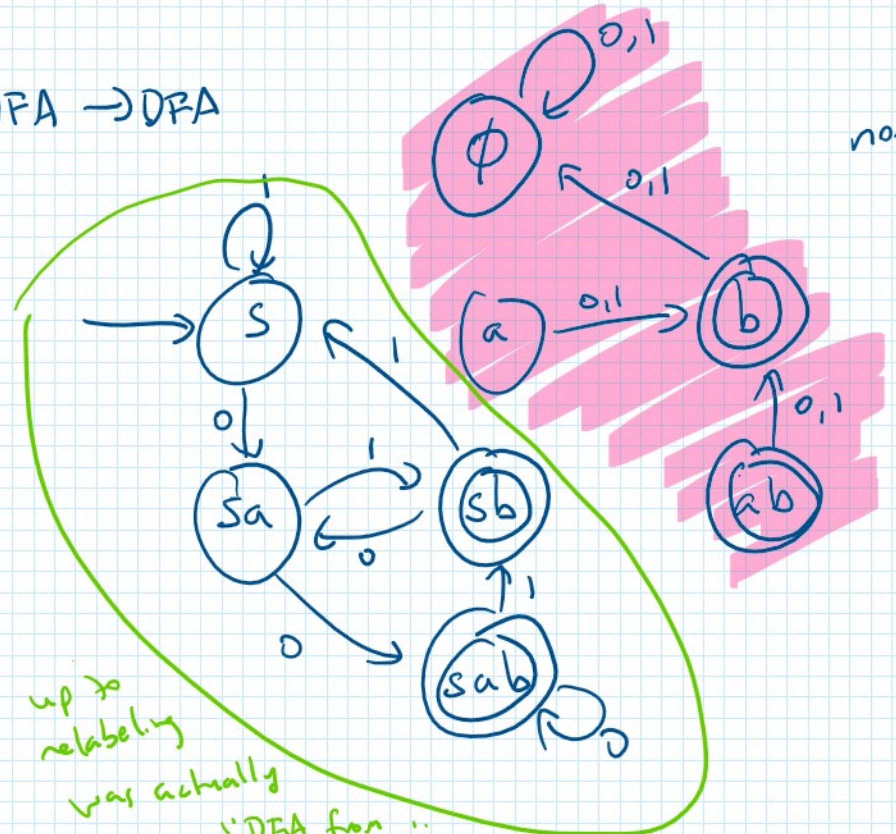
DFA by hand:



NFA:



NFA → DFA

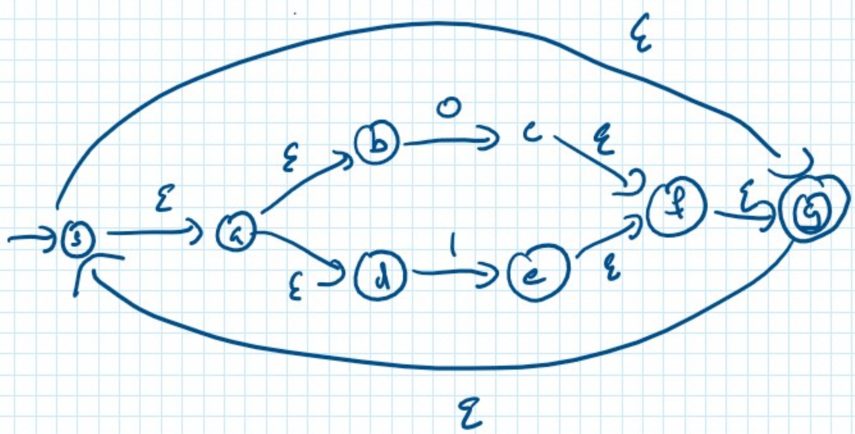


not reachable from the start!

up to relabeling was actually the "DFA from scratch"

!! $|P(Q)| = 2^{|Q|}$ if $|Q|$ is large?

$(0+1)^* \longrightarrow$ Thompson



Incremental Subset Construction

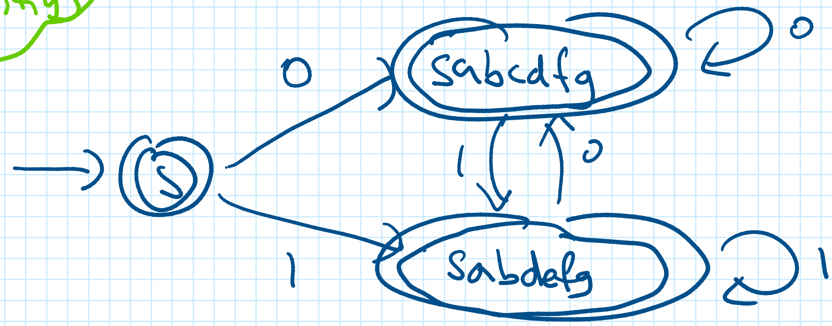
- build the part of the DFA I need.

in P(Q)

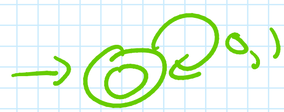
based on ε reach(q)

q	ε-reach(q)	δ′(q,0)	δ′(q,1)	A′?
s	s a b d g	s a b c d f g	s a b d e f g	✓
sabcdfg		sabcdfg	sabdefg	✓
sabdefg		sabcdfg	sabdefg	✓

{s, a, b, c, d, f, g}



simpler DFA:



NFA → regex (state elimination)

(informally) GNFAs / expression NFAs / expression automata

like NFAs but transitions can be arbitrary regexes,

exactly one transition between each pair of states

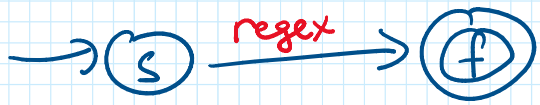
(missing transitions are ϕ)

require: start state have no incoming transitions ?

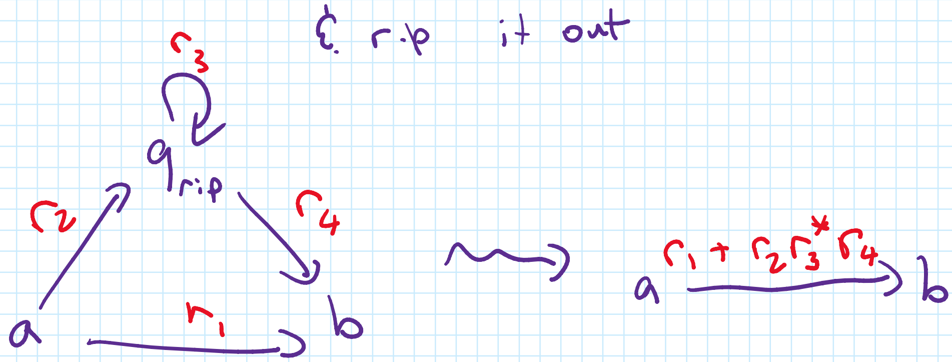
require: start state have no incoming transitions
only one accept state w/ no outgoing transitions } different

→ enforce by adding new start/accept state if necessary.

if two states, then these are start/accept

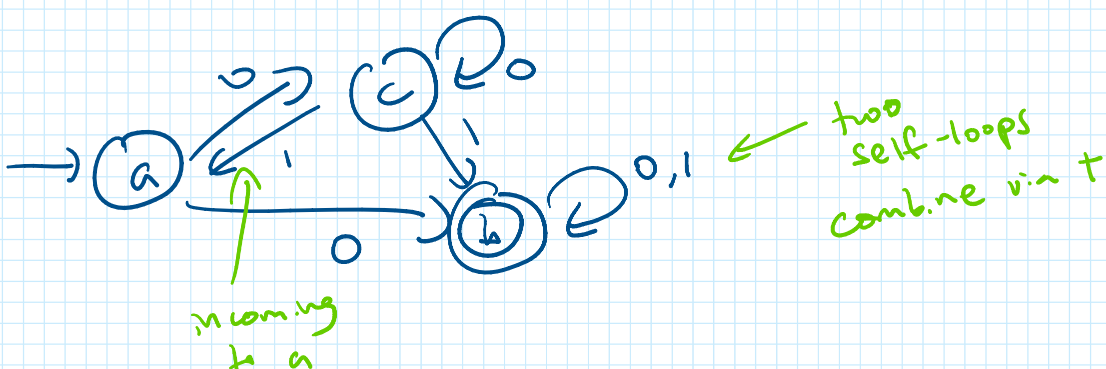


else, pick q_{rip} other than start/accept & rip it out

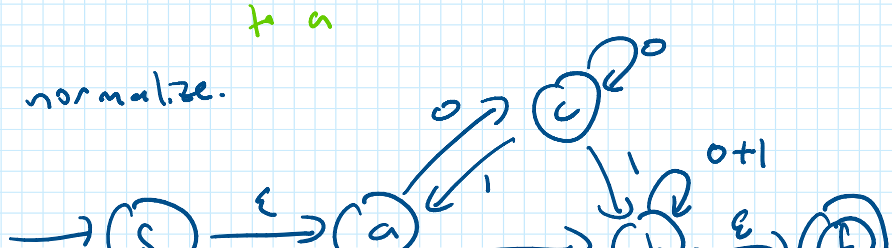


if we do this repeatedly, end up w/ just start/accept. done.

ex.



1. normalize.



2. rip out c.

before

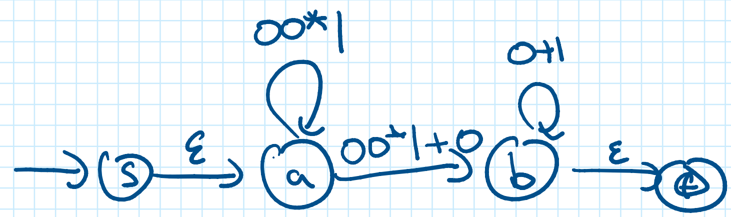
$$\begin{tikzcd} a \arrow[r, "0"] \arrow[rr, bend right, "\cancel{\emptyset}"'] & c \arrow[loop above, "0"] \arrow[r, "1"] & a \end{tikzcd}$$

$$\begin{tikzcd} a \arrow[r, "0"] \arrow[rr, bend right, "0"'] & c \arrow[loop above, "0"] \arrow[r, "1"] & b \end{tikzcd}$$

after

$a \xrightarrow{00^*1 + \cancel{\emptyset}} a$ ignore $+\emptyset$

$a \xrightarrow{00^*1+0} b$



3. rip out a

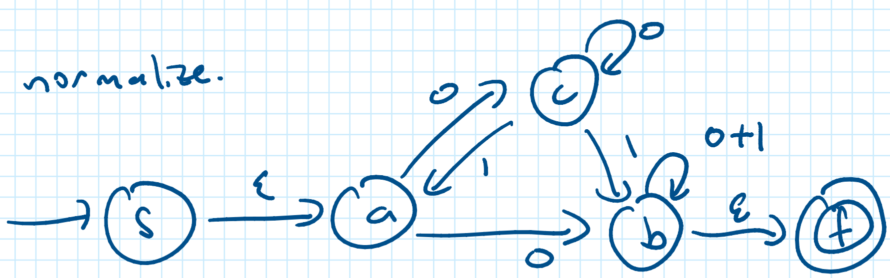
$$\begin{tikzcd} S \arrow[r, "\varepsilon"] \arrow[rr, bend right, "\emptyset"'] & a \arrow[loop above, "00^*1"] \arrow[r] & b \end{tikzcd} \longrightarrow \quad S \xrightarrow{\varepsilon(00^*1)^*(00^*1+0)} b$$

4. rip out b

$$\begin{tikzcd} S \arrow[r, "(00^*1)^*(00^*1+0)"] & b \arrow[loop below, "0+1"] \arrow[r, "\varepsilon"] & f \end{tikzcd} \longrightarrow \quad S \xrightarrow{(00^*1)^*\,(00^*1+0)(0+1)^*} f$$

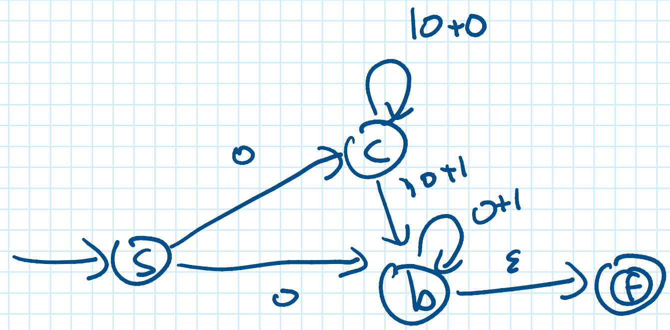
⚠ ripping out states in different orders may give different but equivalent regexes.

1. normalize.



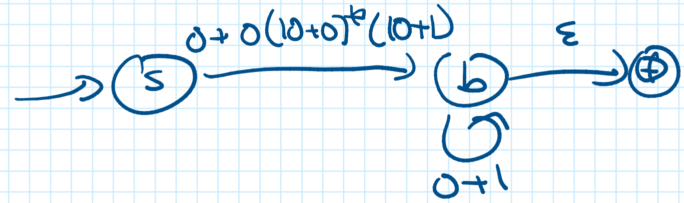
2. rip out a

before	after
$s \xrightarrow{\varepsilon} a \xrightarrow{0} c$	$s \xrightarrow{0} c$
$s \xrightarrow{\varepsilon} a \xrightarrow{0} b$	$s \xrightarrow{0} b$
$c \xrightarrow{1} a \xrightarrow{0} b$, $c \xrightarrow{1} b$	$c \xrightarrow{10+1} b$
$c \xrightarrow{1} a \xrightarrow{0} c$, $c \xrightarrow{0} c$	$c \xrightarrow{10+0} c$

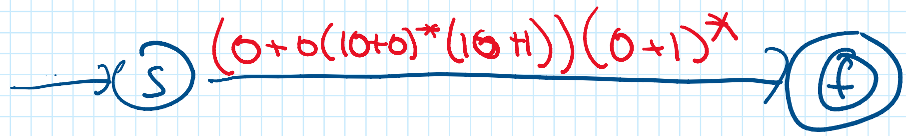


3. rip out c

before	after
$s \xrightarrow{0} c \xrightarrow{10+1} b$, loop $10+0$ on c, $s \xrightarrow{0} b$	$s \xrightarrow{0+0(10+0)^*(10+1)} b$



4. rip out b



$$(00^*1)^*(00^*1+0)(0+1)^* \cong (0+0(10+0)^*(10+1))(0+1)^*$$

$$\cong 0(0+1)^*$$

In general, state elimination gives larger regexes than optimal.

useful (?) rule $$\underset{r_1}{A} + \underset{r_2}{B}\,\underset{r_3^*}{B^*}\,\underset{r_4}{A} = B^* A$$