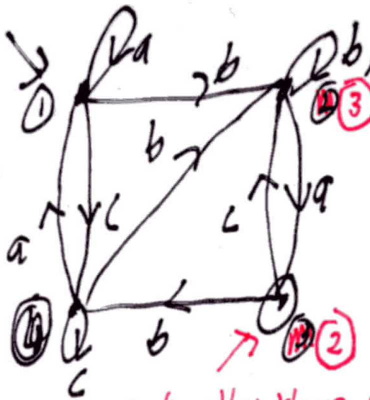
Top Hat #9366

KWR Wed ofc Hrs Cancelled (away)

Junxiang Huang 1-2pm

KWR Thu 1-3pm.

N =



Regexp Matrix T of N:

	1	2	3	4
1	a	∅	b	c
2	∅	∅ (or ε)	c	b
3	∅	a	b+c	∅
4	a	∅	b	c

Only one acc. state other than S. Hence re-number it 2 and don't need to add any extra final state

OK to put ε not ∅ on main diagonal because it will be *ed and $\emptyset^* = \varepsilon^* = \varepsilon$

For $T(1,2) = \emptyset$, however, ε would be wrong.

Also $(c+\varepsilon)^* = c^*$, so $T(4,4) = c$ is fine

Elim. State 4:

In: (1,c] Out:
[2,b] [a,1)
∴ Update [b,3)

T(1,1), T(1,3), T(2,1), and T(2,3).

```
for k = n downto 3:
  for i = 1 to k:
    for j = 1 to k:
      T(i,j) += T(i,k)·T(k,k)*·T(k,j).
```

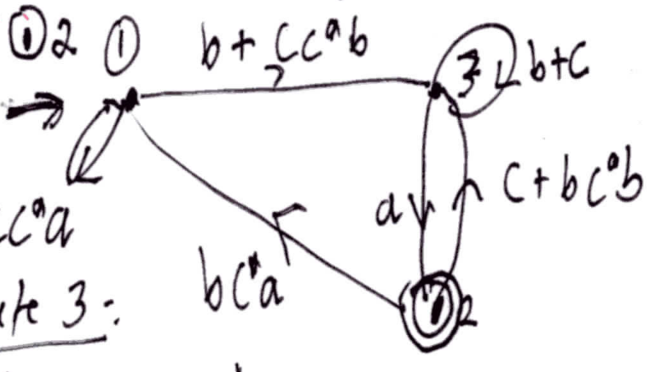
if T(i,k) = ∅ or if T(k,j) = ∅ can skip doing this.

$T(1,1)_{new} = T(1,1)_{old} + T(1,4)\,T(4,4)^*T(4,1)$
$= a + c \cdot c^* \cdot a$

$T(1,3)_{new} = T(1,3)_{old} + T(1,4)\,T(4,4)^*T(4,3)$
$= b + c \cdot c^* b = b + cc^*b$

$T(2,3)_{new} = T(2,3)_{old} + T(2,4)\,T(4,4)^*T(4,3)$
$= c + b \cdot c^* b = c + bc^*b$

$T(2,1)_{new} = T(2,1)_{old} + T(2,4)\,T(4,4)^*T(4,1)$
$= \emptyset + b \cdot c^* \cdot a = bc^*a$



New T Matrix

$T' =$

	1	2	3
1	a+cc*a	∅	b+cc*b
2	bc*a	∅, or ε	c+bc*b
3	∅	a	b+c

Eliminate State 3:

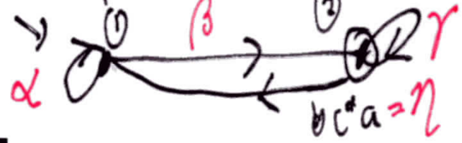
In [1] Out:
(2] [2)

∴ Update T(1,2), T(2,2) only.

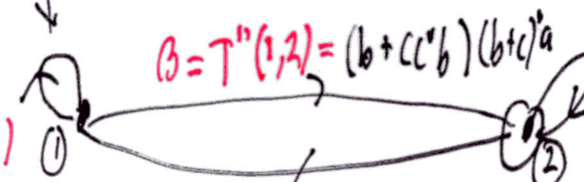
$T'(1,2)_{new} = T'(1,2)_{old} + T'(1,3)\,T'(3,3)^*\,T'(3,2)$
$= \emptyset + (b+cc^*b)(b+c)^*a$

$T'(2,2)_{new} = T'(2,2)_{old} + T'(2,3)\,T'(3,3)^*\,T'(3,2)$
or ε $\emptyset + (c+bc^*b)(b+c)^*a$

blah



$L(N) =$ [redacted] $L_{12} = T''(1,1) +$



$\alpha = T''(1,1) = a + cc^*a$

$\beta = T''(1,2) = (b + cc^*b)(b+c)^*a$

$\gamma = T''(2,2) = (c + bc^*b)(b+c)^*a$ You can stop here!

$\eta = T''(2,1) = T'(2,1) = bc^*a$ (No change)

$$\underline{\underline{L(N) = L_{12}}} = \alpha^* \cdot \beta \cdot (\gamma + \eta\alpha^*\beta)^*$$

or say: $= T''(1,1)^* \cdot T''(1,2) \cdot \left(T''(2,2) + T''(2,1) T''(1,1)^* T''(1,2) \right)^*$

My shortcut avoids the text's final steps on rather big regexps.

Note: $(T''(2,2) + \text{something})^*$ doesn't care whether $T''(2,2) = \text{blah}$ or $\underline{(\text{blah} + \epsilon)}$.

We have completed a cycle of proving $\underline{\text{Kleene's}}$ Theorem:

For every language A over an alphabet Σ

1. There is a regular expression r s.t. $A = L(r)$ ≡ Most often/historically A is a regular language
2. ⟺ There is a DFA M s.t. $A = L(M)$ ⟵ text's defn of regular
3. ⟺ There is an NFA N s.t. $A = L(N)$
4. ⟺ There is a GNFA N' s.t. $A = L(N')$.

Real point: All of these characterize the class REG of regular languages.

We proved ① ⟹ ③ ⟹ ②, ③ → ④, ② → ④ by "IS-A", ④ ⟹ ①

We can use these steps algorithmically — up to a point.

$\underline{\text{Theorem}}$: For every regular expression r_1, r_2 we can build a regexp r_3 such that $L(r_3) = L(r_1) \cap L(r_2)$. And r_4 s.t. $L(r_4) = L(r_1) \Delta L(r_2)$

Algorithmic Proof: Convert r_1 and r_2 into equivalent NFAs N_1 and N_2.

Can blow up ⟶ Convert N_1 and N_2 into equivalent DFAs M_1 and M_2

Use Cartesian Product to build a DFA M_3 s.t. $L(M_3) = L(M_1) \cap L(M_2)$ or M_4 s.t. $L(M_4) = L(M_1) \Delta L(M_2)$

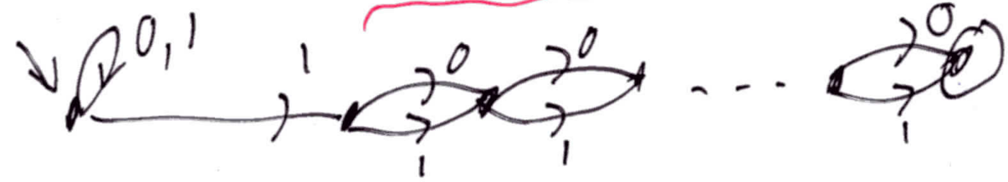
Convert M_3 into the regexp r_3. ☒ ...M_4 into r_4 ☒

An Example to Note: For every K, define

$L_K = \{ x \in \{0,1\}^* :$ the Kth bit from the end is a $1 \}$.

Regexp $r_K = (0+1)^* 1 \underbrace{(0+1)}_{K-1 \text{ of these}}{}^{K-1}$ $\quad 12 + \lceil \log_2(K) \rceil$ chars.

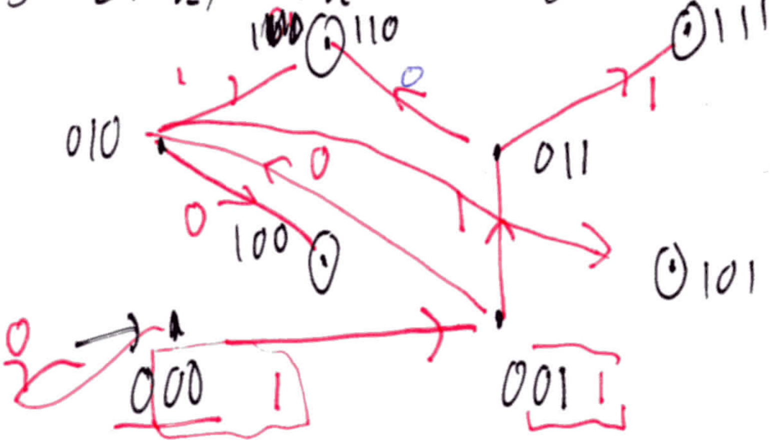
NFA $N_K =$



$K+1$ states

Fact: The smallest DFA M_K s.t. $L(M_K) = L_K$ ~~needs~~ has 2^K states!

Strategy: One state for every last K bits read.
Accept for K bits beginning with 1
For $K=3$, $2^3 = 8$ states



Extra How do we know that 2^K states are needed? A preview:
Consider the two states 110 and 111 at the top, which we got to upon reading $x = 110$ and $y = 111$, respectively. Now suppose the next two chars are $z = 00$. Then $xz = 11000$, which does not belong to L_3 because the third char from the right is a 0. But $yz = 11100$, which does belong to L_3. We can say $L_3(xz) \neq L_3(yz)$ for short, thinking of $L_3(w)$ as the Boolean function for "$w \overset{?}{\in} L_3$". This means the DFA needed different states to process x and y to, else it would have needed to give the same answer to xz and yz. Similar reasoning applies to any two states, taking $z = 00$, $z = 0$, or $z = \epsilon$ depending on where the states' binary labels differ. So the DFA needs all 8 states.