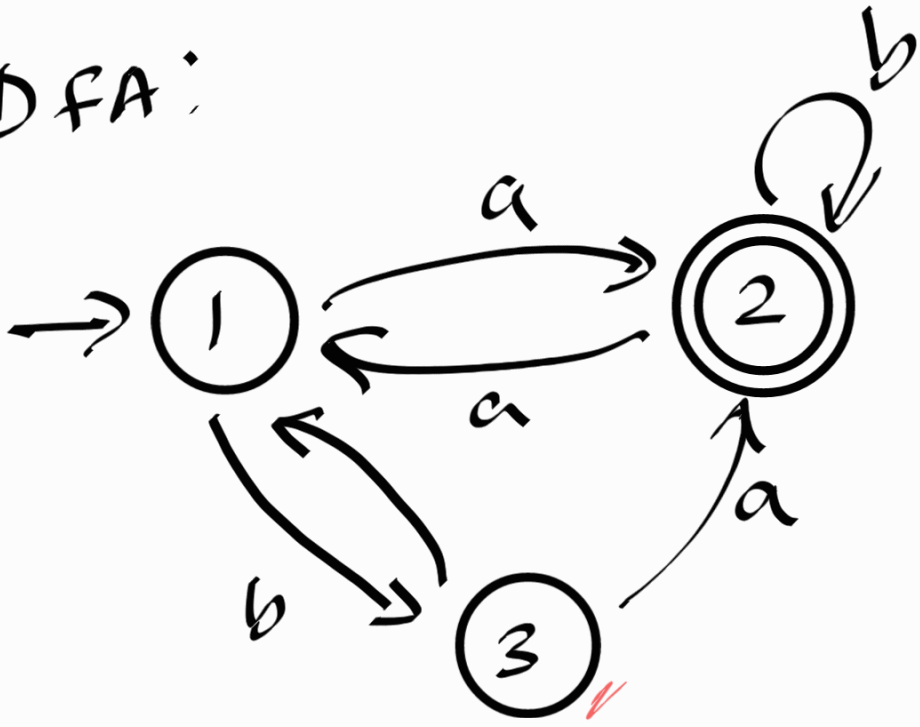
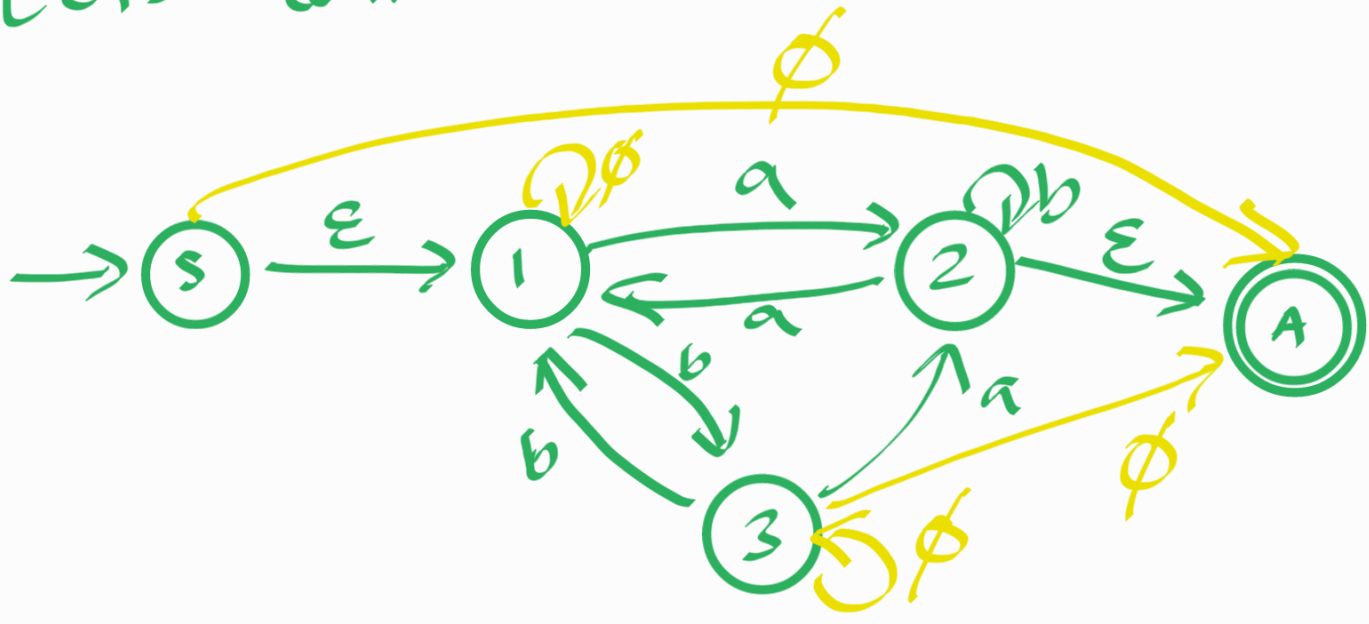
02/10/2025

Another example of DFA ⇒ GNFA ⇓ regular expression

DFA:

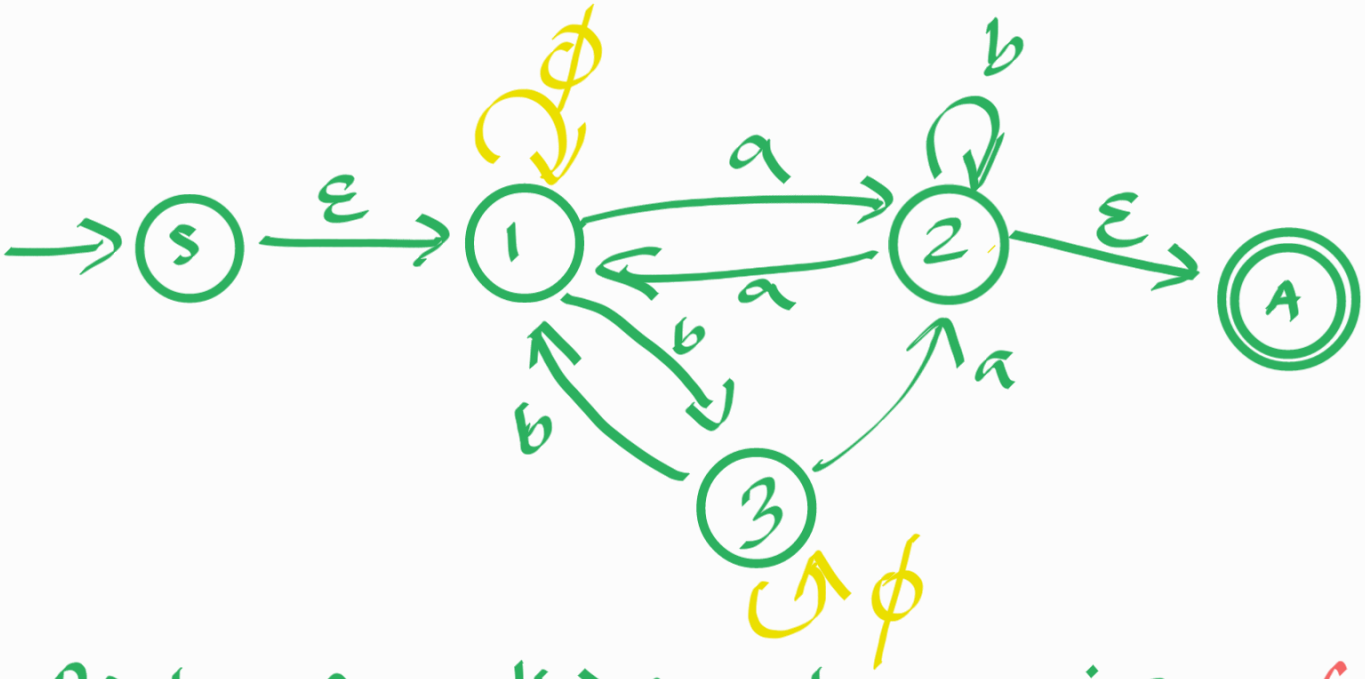


Let's convert to a GNFA



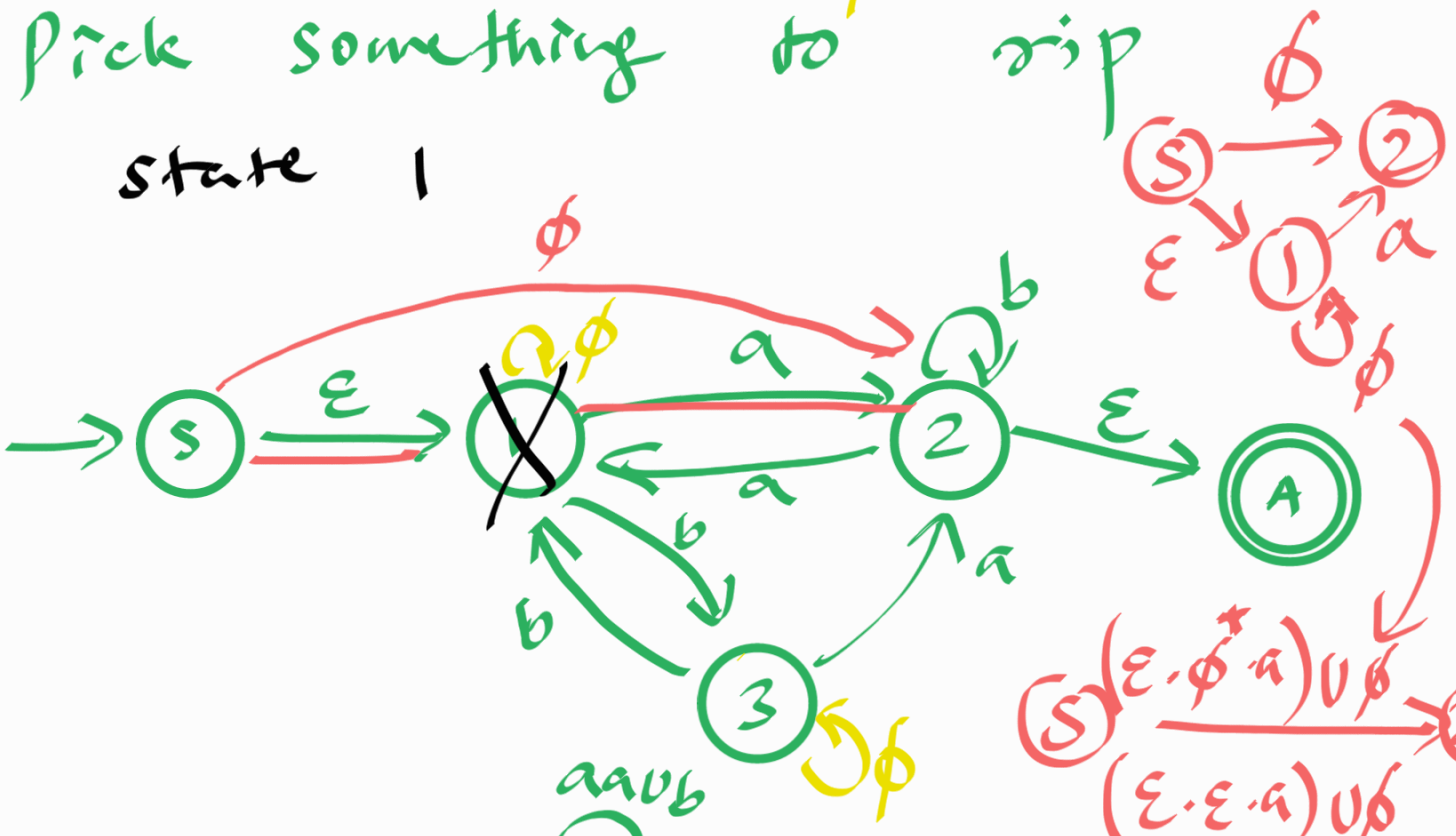
But we do not want to clutter the figure so I wont be adding

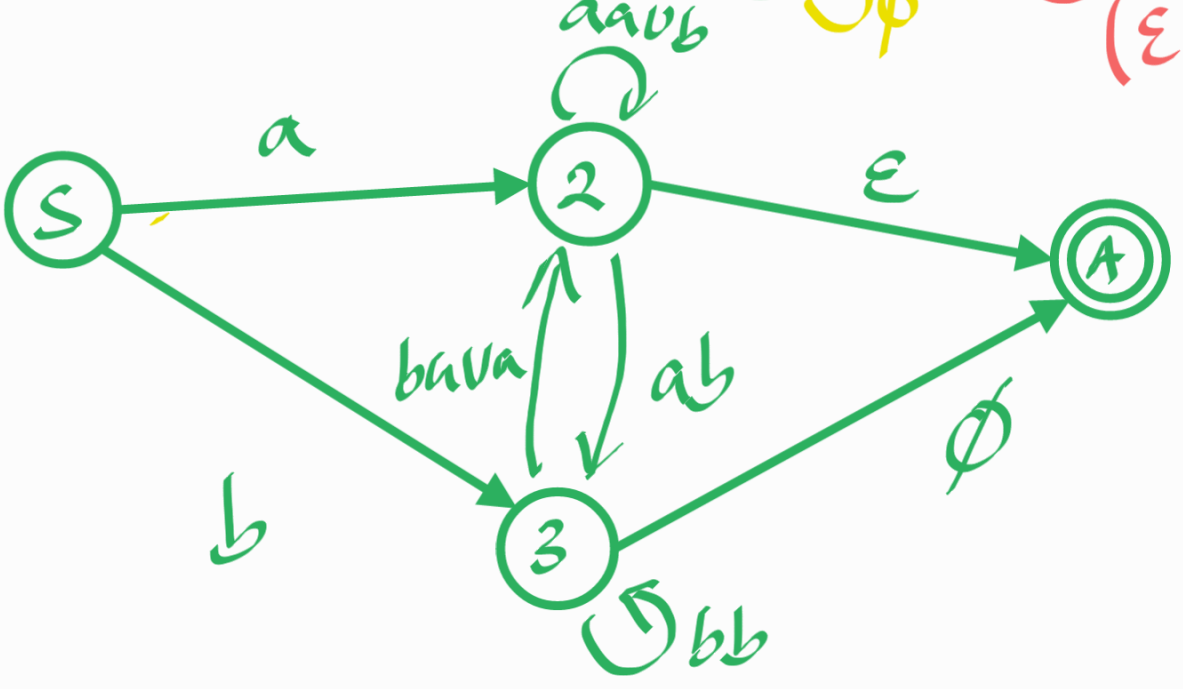
these ∅ transitions



Pick something to rip

state 1





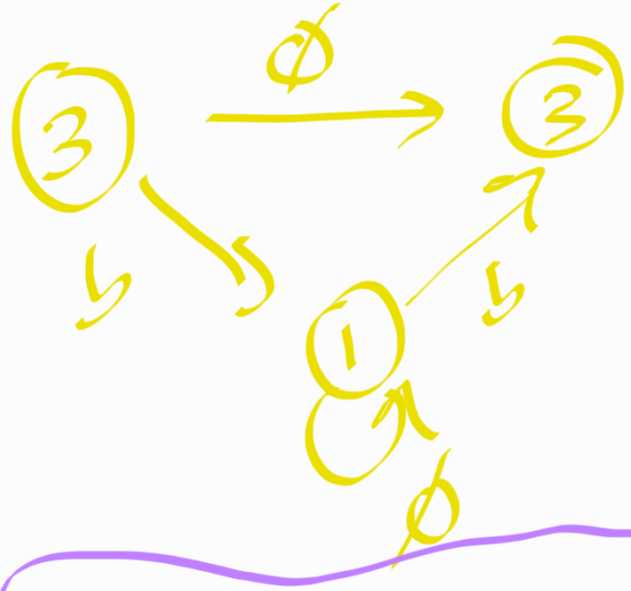
Explanation on ripping ①

$$\Rightarrow \textcircled{3} \xrightarrow{(\varepsilon \cdot \phi^* \cdot a) \cup \phi} \textcircled{2}$$

$$= \varepsilon \cdot \phi^* \cdot a$$

$$= \varepsilon \cdot \varepsilon \cdot a$$

$$= a$$



$$\textcircled{3} \xrightarrow{(b \cdot \phi^* \cdot b) \cup \phi} \textcircled{3}$$

$$= (b \cdot \varepsilon \cdot b) \cup \phi$$

$$= bb$$

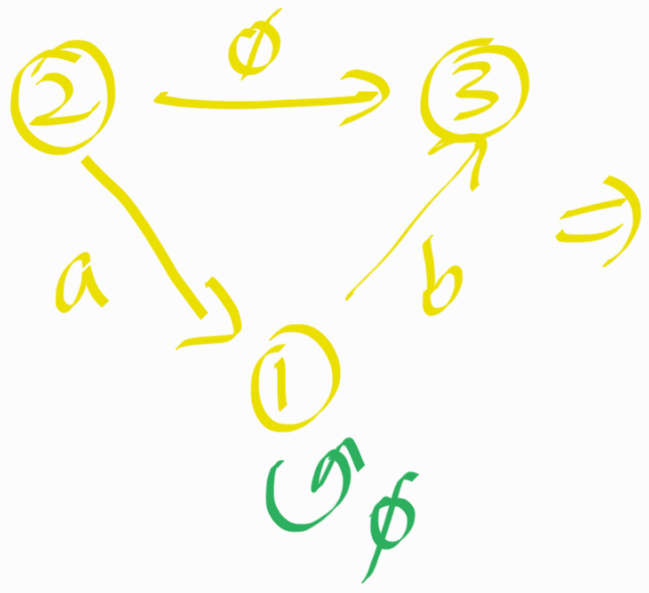
Remember $\phi^* = \{\varepsilon\}$

$$\Rightarrow \textcircled{3} \xrightarrow{(b \cdot \phi^* \cdot a) \cup a} \textcircled{2}$$

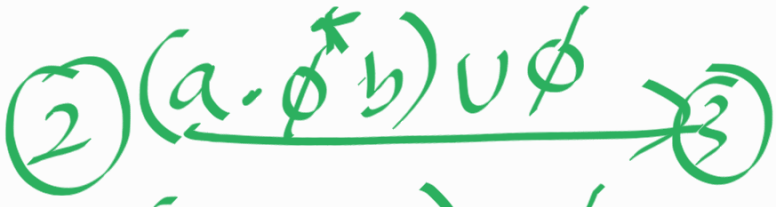
$$= (b \cdot \varepsilon \cdot a) \cup a$$

$$= (ba \cup a)$$

we need to consider

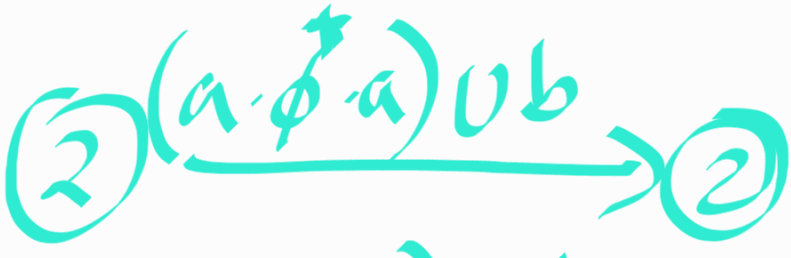


$\Rightarrow$



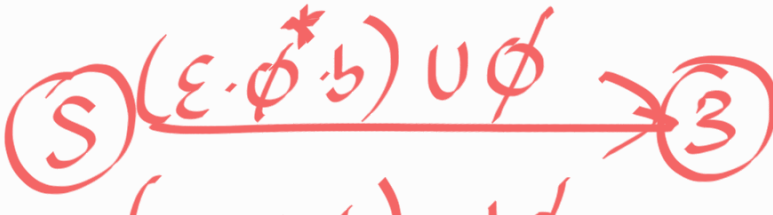
$= (a \cdot \varepsilon \cdot b) \cup \emptyset$

$= ab$



$= (a \cdot \varepsilon \cdot a) \cup b$

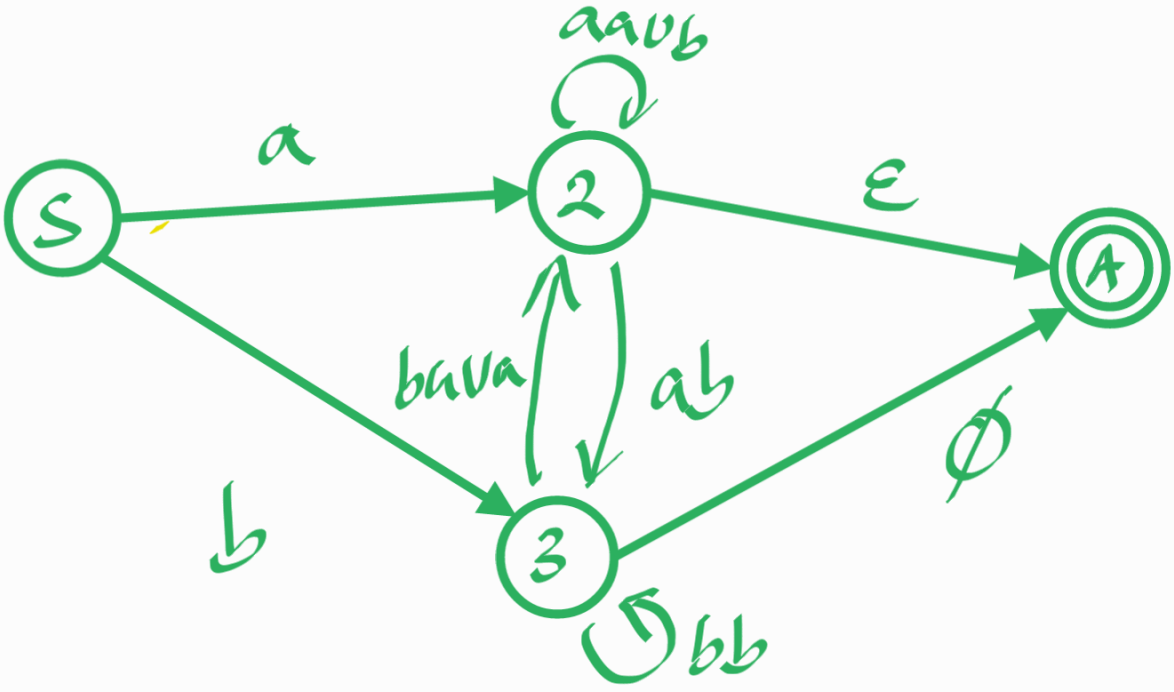
$= aa \cup b$



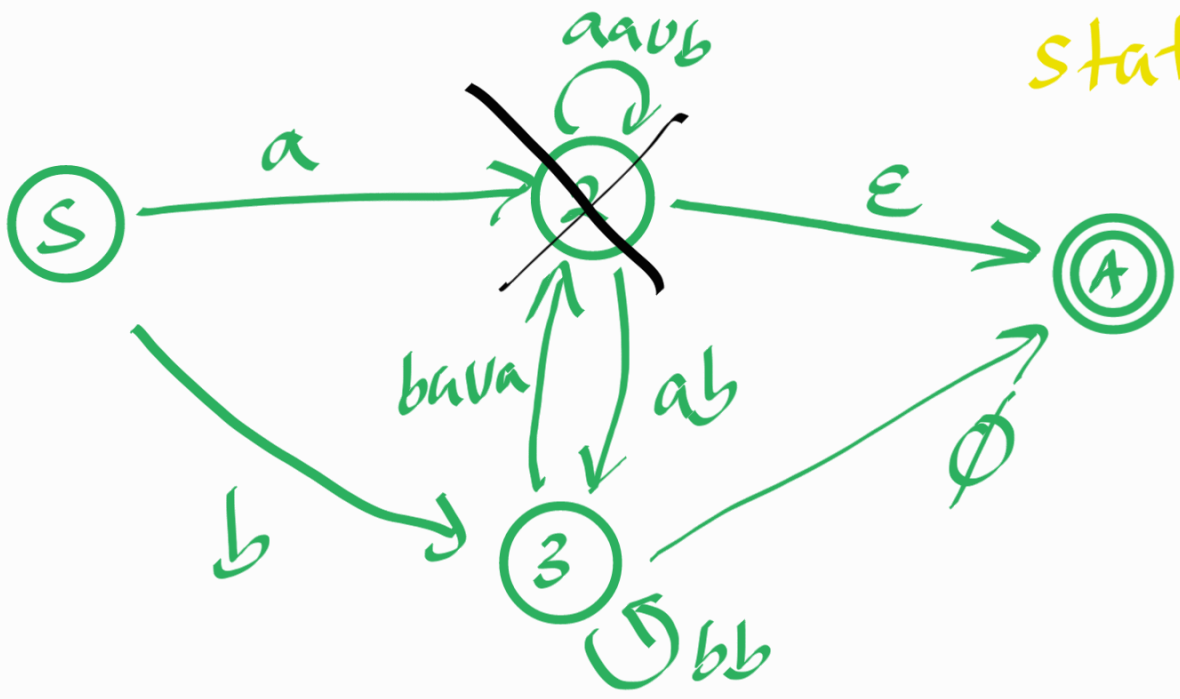
$= (\varepsilon \cdot \varepsilon b) \cup \emptyset$

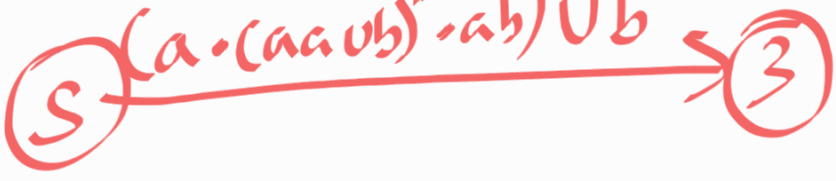
$= b \cup \emptyset$

$= b$



Now let's rip state 2

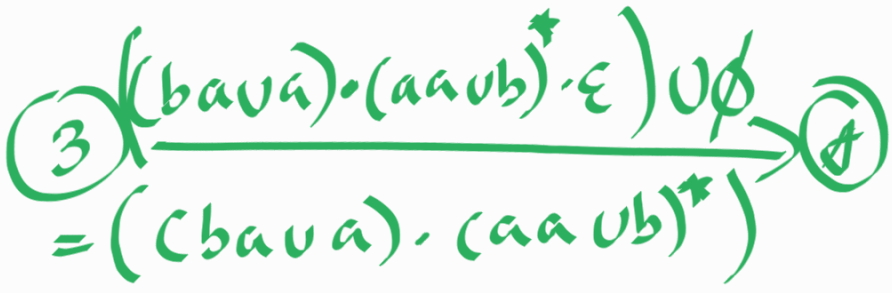


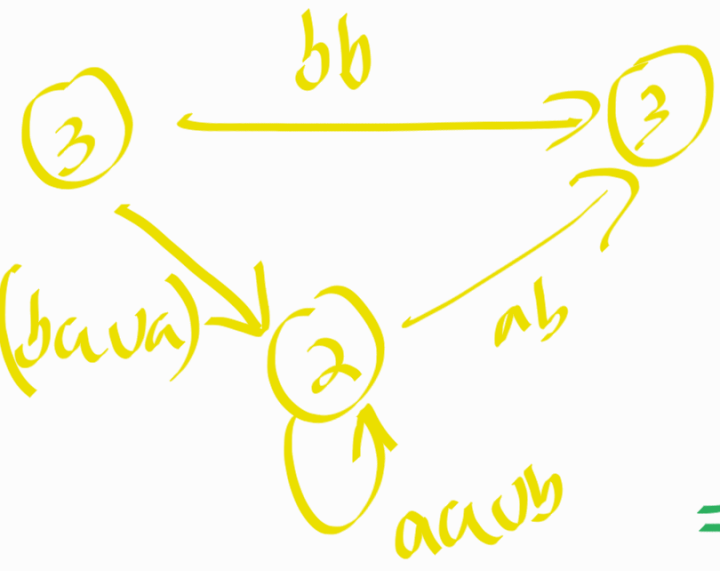


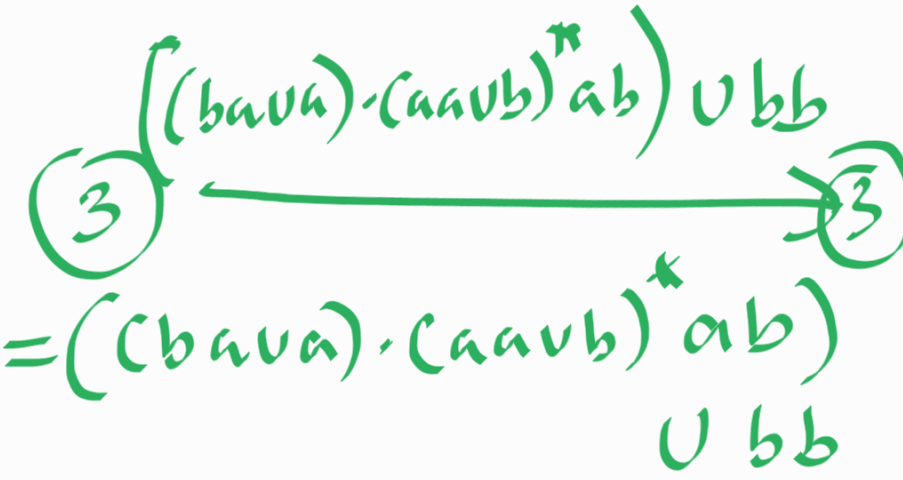
$= a \cdot (aa \cup b)^* \cdot \varepsilon$

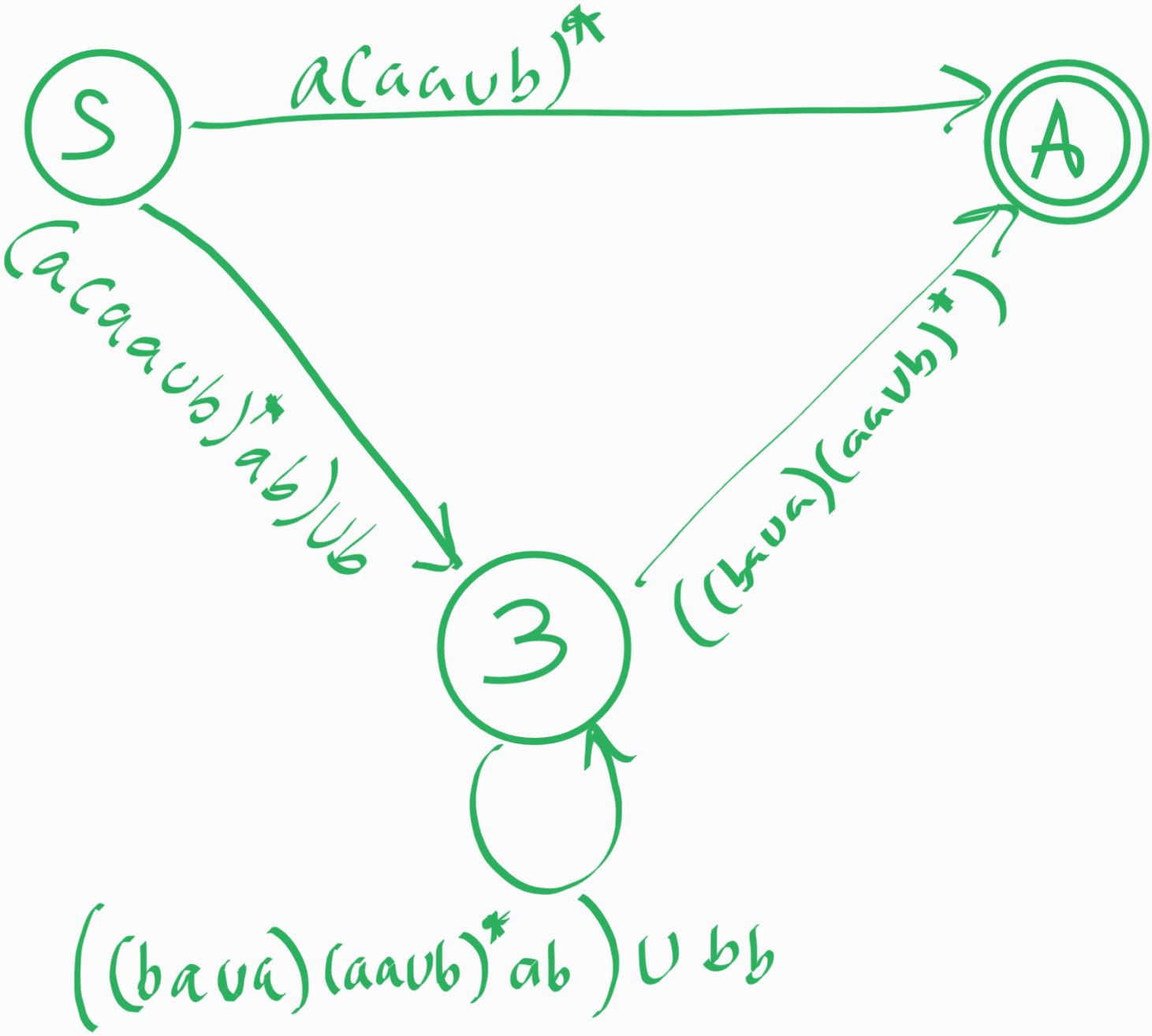
$= a\,(aa \cup b)^*$

3
4
2
ϕ
(bavua)
ε
aavb

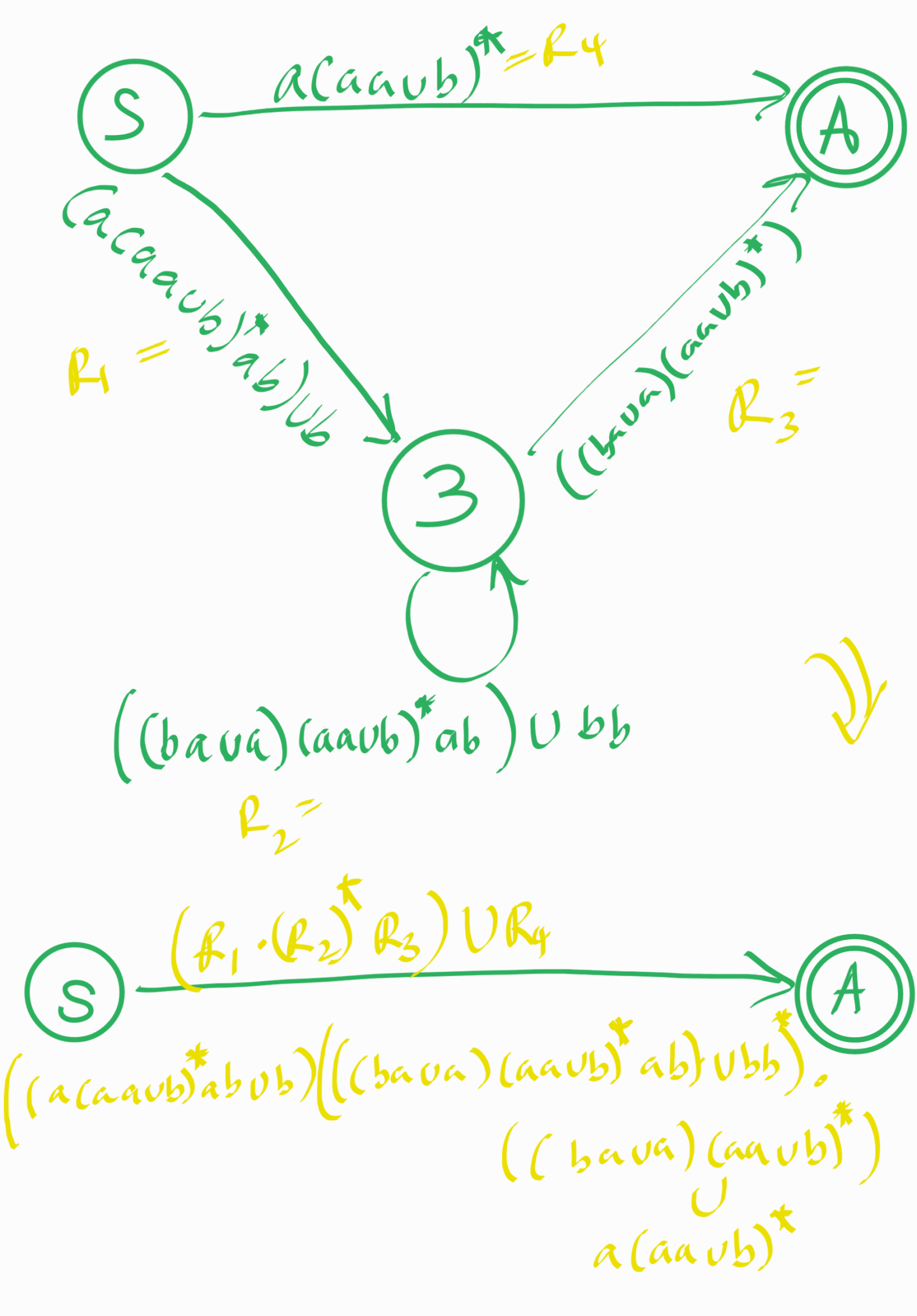

3
4
((bavua)·(aavb)*·ε)∪ϕ
=((bavua)·(aavb)*)


3
3
2
bb
(bavua)
ab
aavb


3
3
((bavua)·(aavb)*ab)∪bb
=((bavua)·(aavb)*ab)
∪bb



After this we remove 3


S
A
3
a(aaub)* = R4
(a(aaub)*ab)Ub
R1 =
((bavba)(aaub)*)
R3 =
((bavba)(aaub)*ab)Ubb
R2 =
S
A
(R1·(R2)*R3)UR4
(a(aaub)*abub)((bavba)(aaub)*ab)Ubb)*.
((bavba)(aaub)*)
U
a(aaub)*

Whole lot of proofs

just to say:

NFA = DFA = regular expressions

Now, let's see whether there are languages we cannot recognize by a NFA / DFA.

1.4 Non-Regular languages

$A = \{ w \mid w \text{ has more 0's than 1's} \}$

$B = \{ a^n b^n \mid n \geq 0 \}$

$C = \{ w \mid w \text{ has equal number of 0's and 1's} \}$

$D = \{ w \mid w \text{ has an equal number of 01 and 10 as substrings} \}$

In A machine needs to keep track of # of 0's or 1's. Depending on the string this value could be different. Hence, a machine with limited memory cannot do this.

However, just because a language appears to require unbounded number of memory does not mean that the language is not regular.

$$C = \{ w \mid w \text{ has an equal \# of 0's and 1's} \}$$

$$D = \{ w \mid w \text{ has equal \# of occurrences of 01 and 10} \}$$

Sometimes our intuition is misleading

Let's look at how we can show a particular language is not regular.

Pumping Lemma

- Basically this theorem states that regular languages have a special property.
- If we can show that a language does not have this property, then the language is not regular.

* The basic idea of this property is that all strings in a regular language can be "pumped" if they are at least as long as a certain special value, called the Pumping length.

Pumping Lemma

If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$
2. $|y| > 0$, and
3. $|xy| \leq p$

* $|s|$ represents the length of the string
* y^i means i copies of string y.

Proof idea:

Pick p to be # of states in the DFA that recognizes the regular language A.

Then pick any string s in A of length at least p.

* If no such strings exist, then the claim is vacuously true.

Since the size of s is larger than p, then in its execution path to an accepting state, we must contain a repeated state.

$s = xyz$

Since this path leads to a accepting state, this should accept xy, xyz, xy^2z, xy^3z, ..., xy^iz.

This is the basic idea.

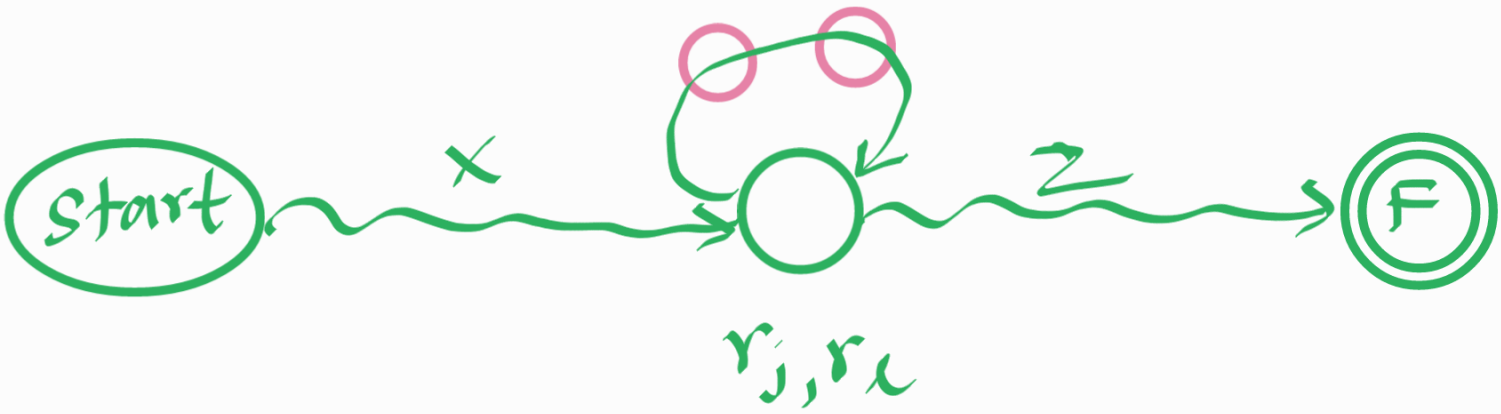
A condensed proof

Let $M = (Q, \Sigma, \delta, q_0, f)$ be a DFA that accepts the regular language A.

Let p be the # of states in M. i.e., $|Q| = p$

- Let $S = s_1 s_2 \ldots s_n$ $S \in A$, $n \geq p$
- Let $r_1, r_2, r_3 \ldots r_{n+1}$ be the sequence of states that M enters while processing S, i.e., $r_{i+1} = \delta(r_i, s_i)$, Note that $n+1 \geq p+1$
- By the pigeon hole principle, among the first p+1 states, two must be the same.

- Let say first be r_j and second be r_l



- Let $x = s_1 s_2 \ldots s_{j-1}$

$y = s_j \ldots s_{l-1}$

$z = s_l \ldots s_n$

- clearly x takes M from r_1 to r_j

y takes M from r_j to r_l

z takes M from r_l to r_{n+1}

- M must also accept xy^iz for $i \geq 0$

- As $j \neq \ell$, we have $|y| > 0$

- As $\ell \leq p+1$, $|xy| \leq p$