CS375: Logic and Theory of Computing

Fuhua (Frank) Cheng

Department of Computer Science

University of Kentucky

Table of Contents:

Table of Contents (conti):

- **Weeks 12-13: Propositional Logic (Chapter 6), Predicate Logic (Chapter 7), Computational Logic (Chapter 9), Algebraic Structures (Chapter 10)**

6. Regular Languages & Finite Automata

- Finite Automata

algorithm

Can a machine recognize a regular language?

Yes

Deterministic Finite Automaton (DFA)

A finite digraph over an alphabet A (vertices are called states).

Each state emits one labeled edge for each letter of *A*. One state is defined as the start state and several states may be final states.

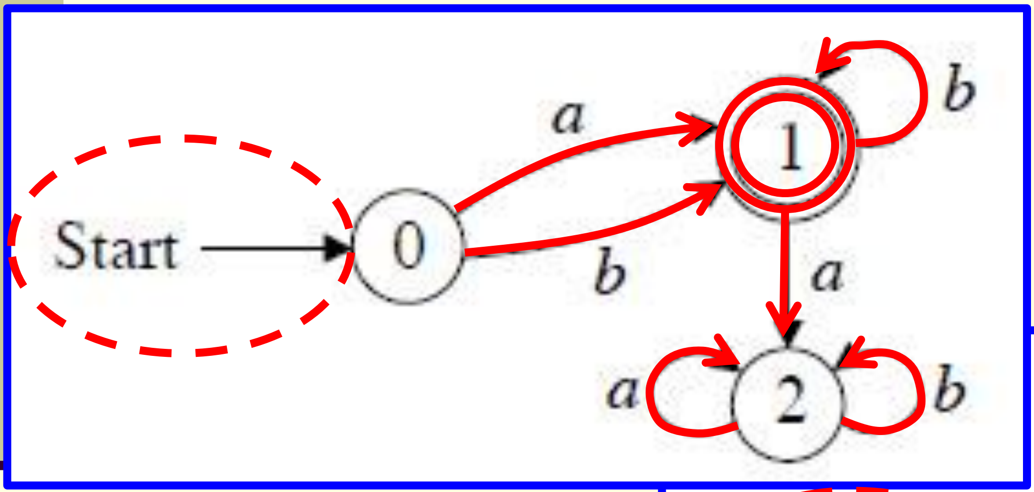
indicated by double circles

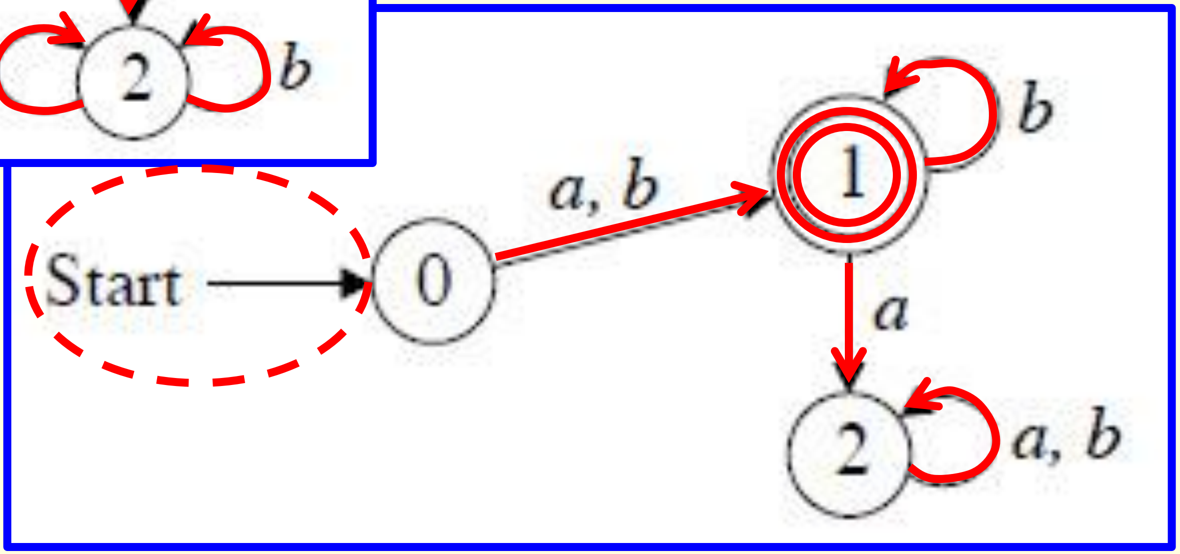
6. Regular Languages & Finite Automata

- Finite Automata

Example.

Either one is acceptable





A={a, b}

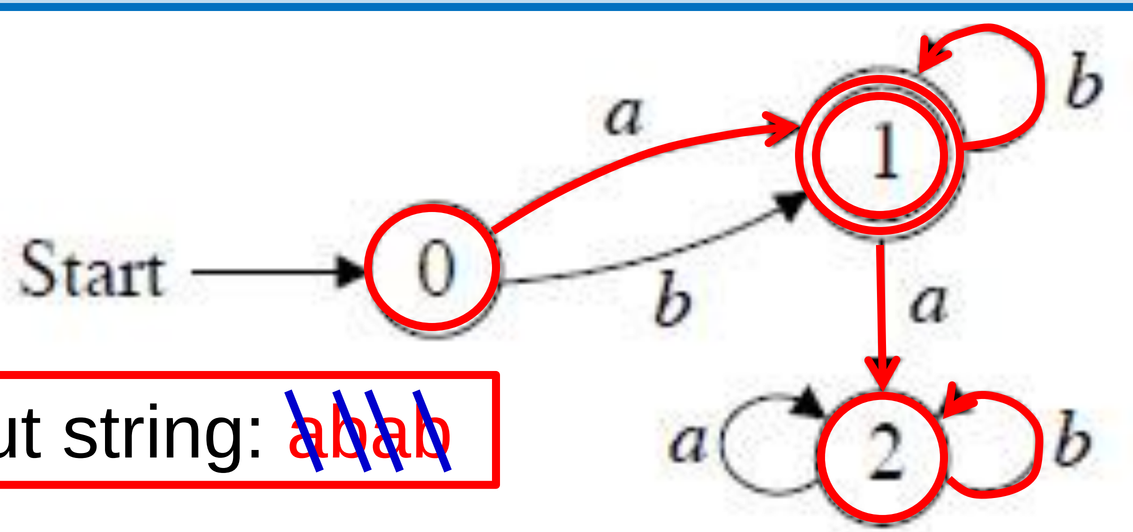
6. Regular Languages & Finite Automata

- Finite Automata

The execution of DFA for input string $w \in A^*$ begins at the start state and follows a path whose edges concatenate to w.

The DFA accepts w if the path ends in a final state. Otherwise the DFA rejects w.

The language of a DFA is the set of accepted strings.

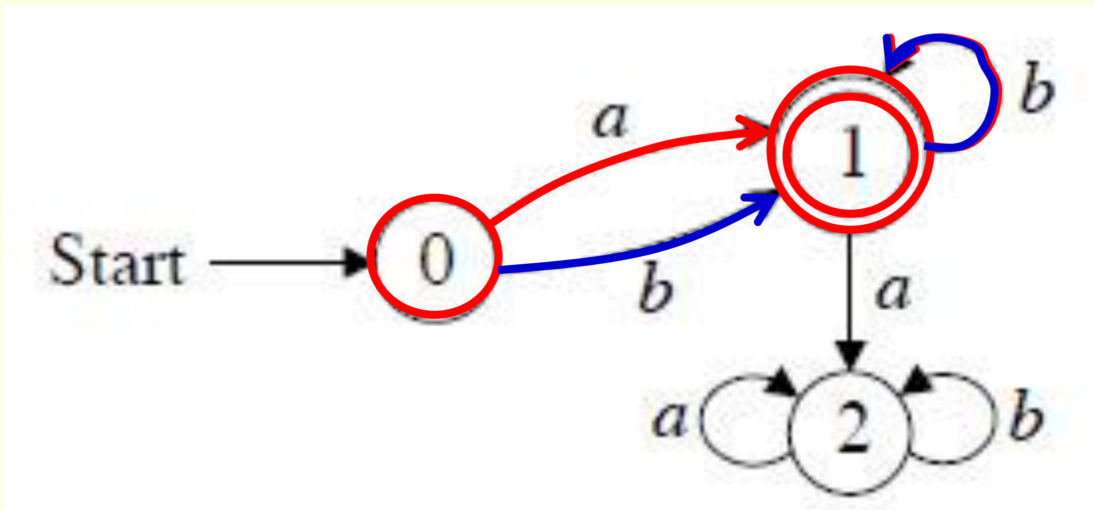


Input string: ~~abab~~

an empty string will enter the start state but the empty set will not.

6. Regular Languages & Finite Automata

- Finite Automata



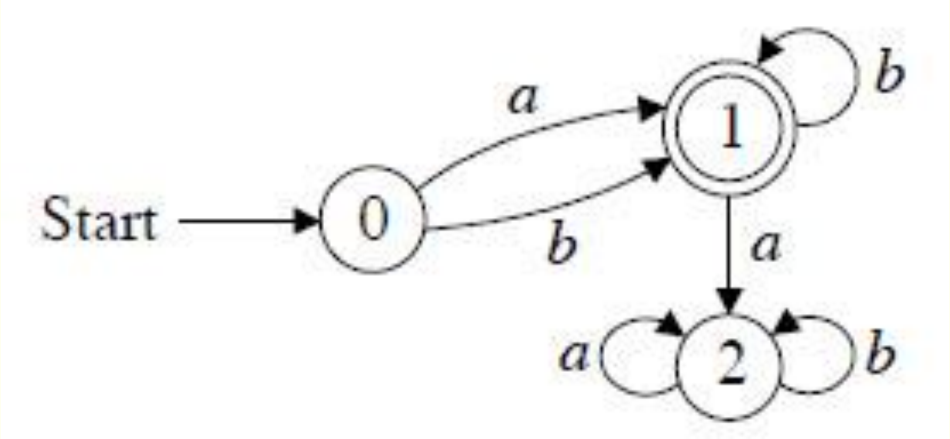
Example. The example DFA accepts the strings

$a, b, ab, bb, abb, bbb, \ldots, ab^n, bb^n, \ldots$

The language of the DFA is $\{ ab^n, bb^m \mid n \in N, m \in N \}$

6. Regular Languages & Finite Automata

– Finite Automata



Example. The example DFA accepts the strings

$a, b, ab, bb, abb, bbb, \ldots, ab^n, bb^n, \ldots$

The regular expression of the language of the DFA is

$\boxed{(a + b)b^*}$ Why?

a+b (a+b)b $(a+b)b^2$ $(a+b)b^n$

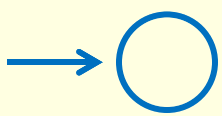
6. Regular Languages & Finite Automata

- Finite Automata

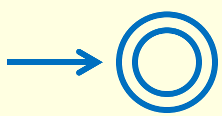
over the same alphabet

Theorem (Kleene) The class of regular languages is exactly the same as the class of languages accepted by DFAs.

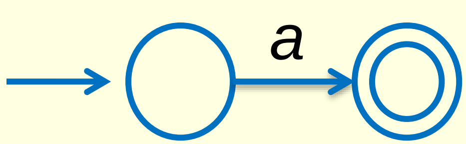
Proof. Need three lemmas or by induction.



Language accepted by this DFA is Φ



Language accepted by this DFA is $\{\Lambda\}$

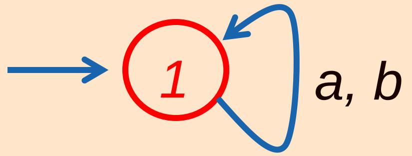


Language accepted by this DFA is $\{a\}$

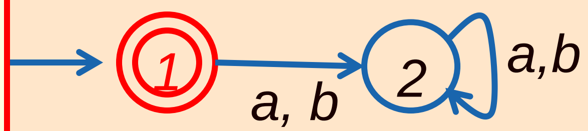
6. Regular Languages & Finite Automata

- Finite Automata

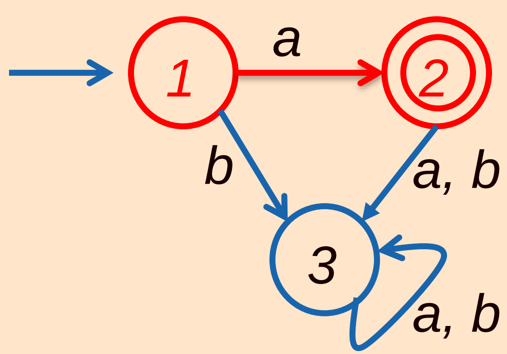
Specifically, say A = {a, b}, then



Φ is its language



{Λ} is its language



{a} is its language

- Finite Automata

Theorem (Kleene) The class of regular languages is exactly the same as the class of languages accepted by DFAs.

Proof. (conti.)

Inductive step: prove that if L_1 and L_2 are accepted by DFAs, then $\boxed{L_1 \cup L_2}$, $\boxed{L_1 L_2}$ and $\boxed{L_1^*}$ are accepted by DFAs.

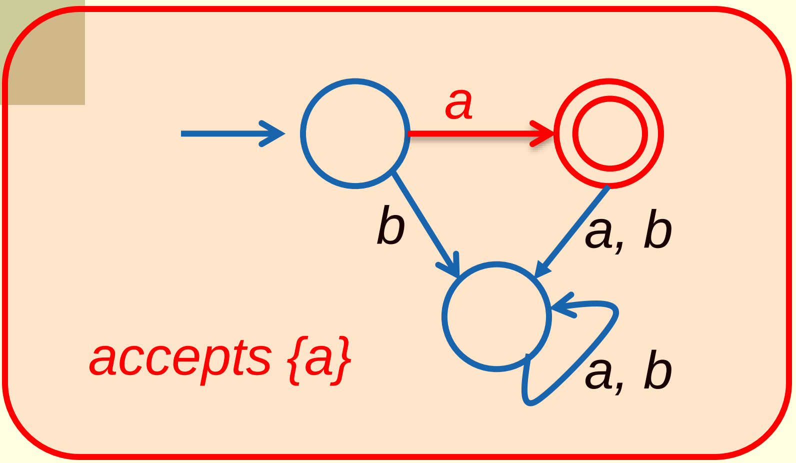
Since any regular language is obtained from $\{\Lambda\}$ and $\{a\}$ for any symbol *a* in the alphabet *A* by using union, concatenation and Kleene star operations, that

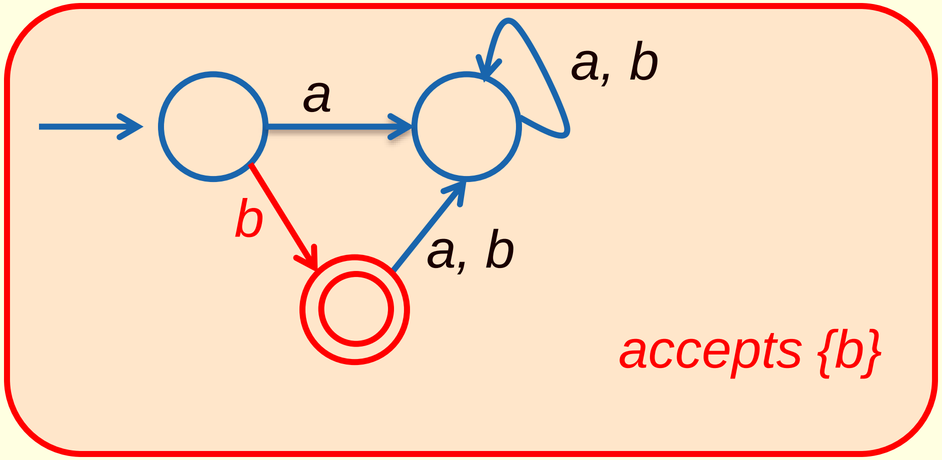
together with the basis step would prove the theorem.

6. Regular Languages & Finite Automata

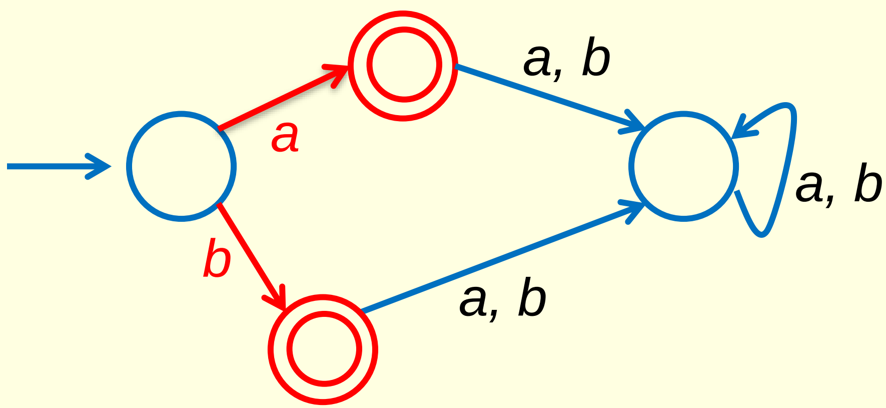
- Finite Automata

For instance, if L_1 = {a} and L_2 = {b}





then



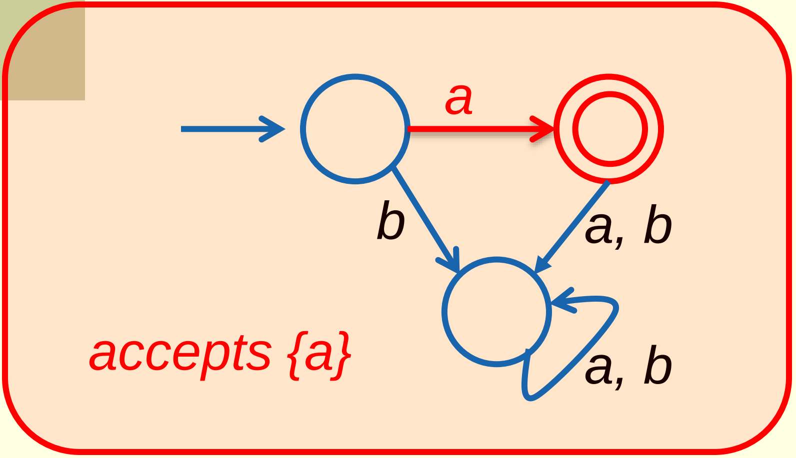
$\boxed{L_1 \cup L_2}$

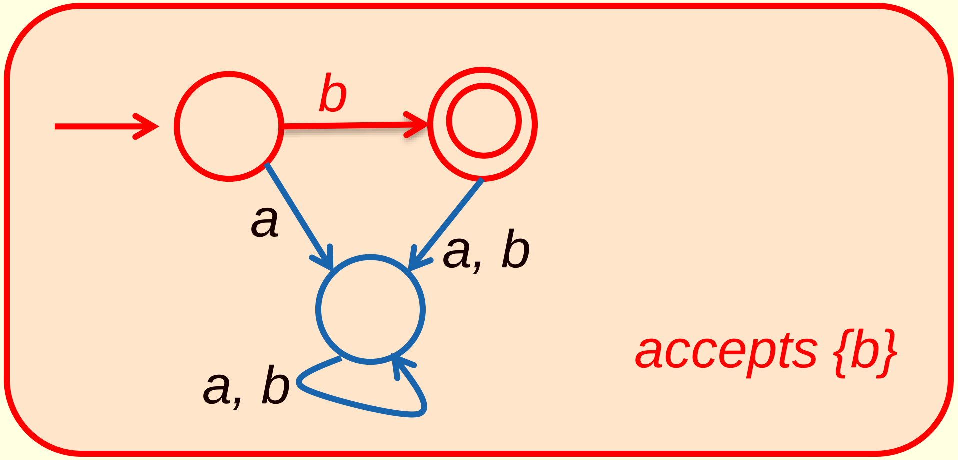
accepts {a, b}

6. Regular Languages & Finite Automata

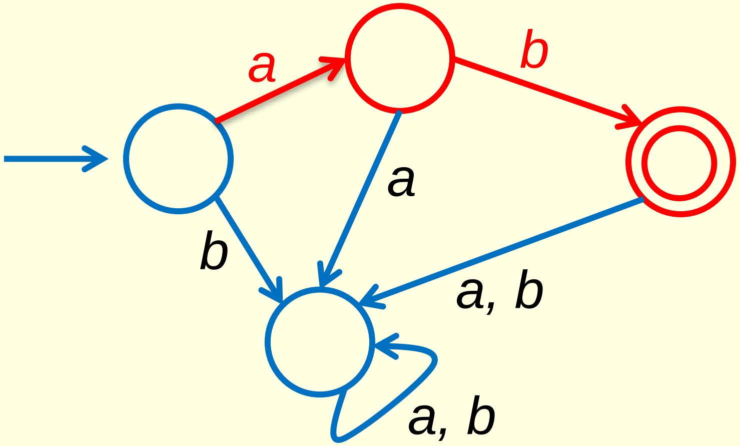
- Finite Automata

For instance, if L_1 = {a} and L_2 = {b}





then



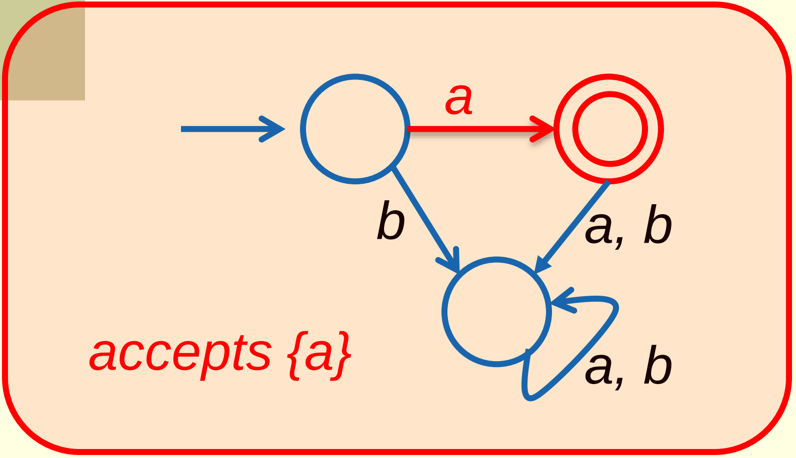
$\boxed{L_1 L_2}$

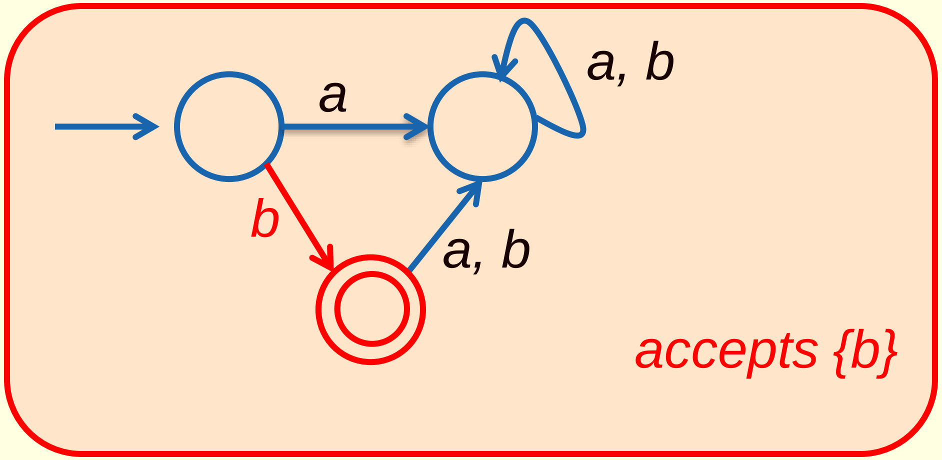
accepts {ab}

6. Regular Languages & Finite Automata

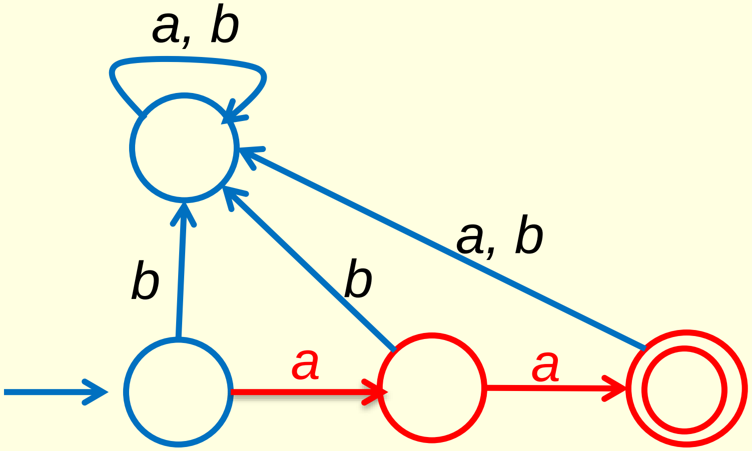
- Finite Automata

For instance, if L_1 = {a} and L_2 = {b}





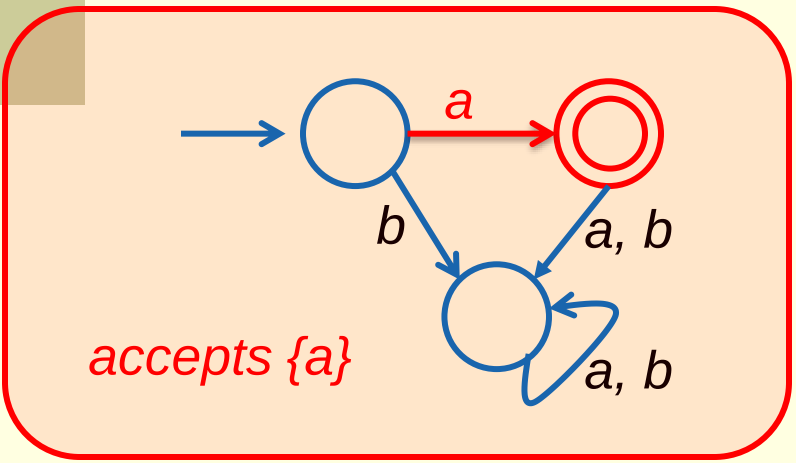
then

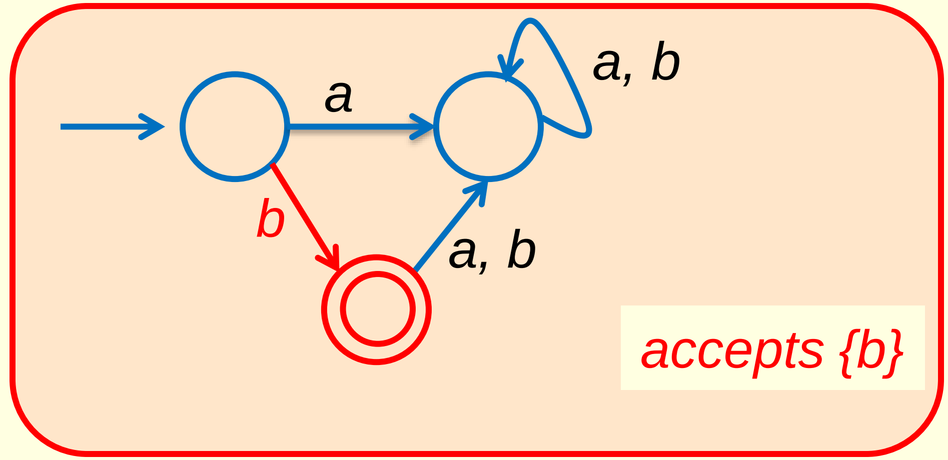


6. Regular Languages & Finite Automata

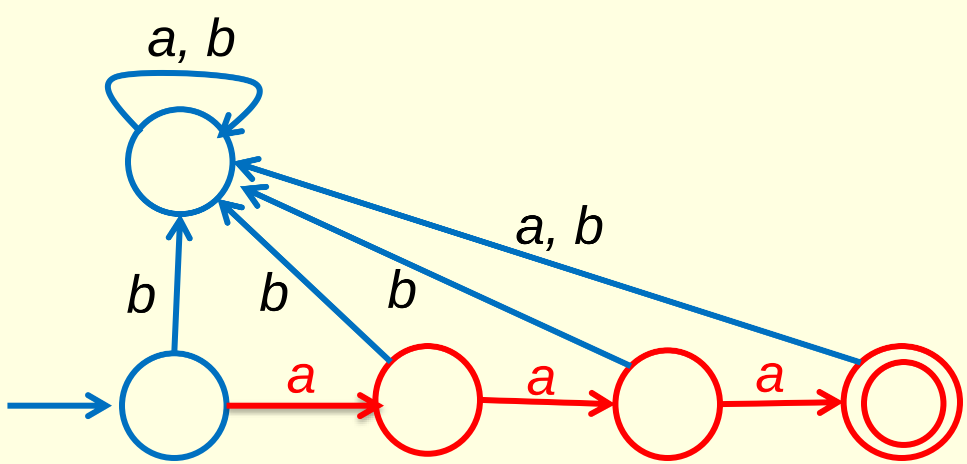
- Finite Automata

For instance, if L_1 = {a} and L_2 = {b}





then

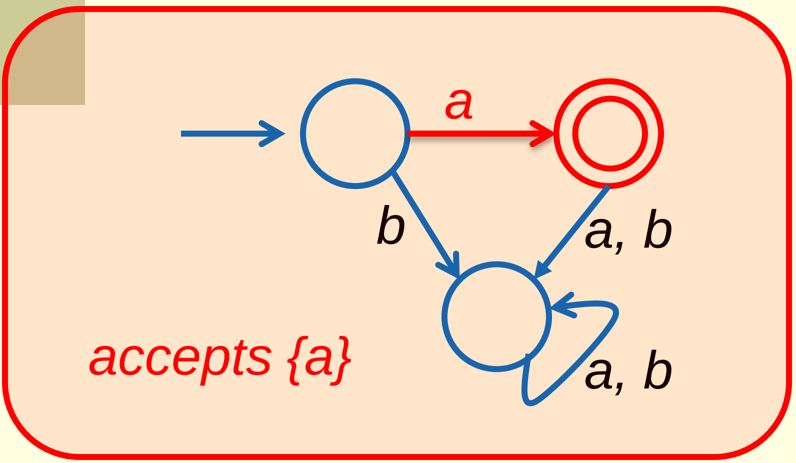


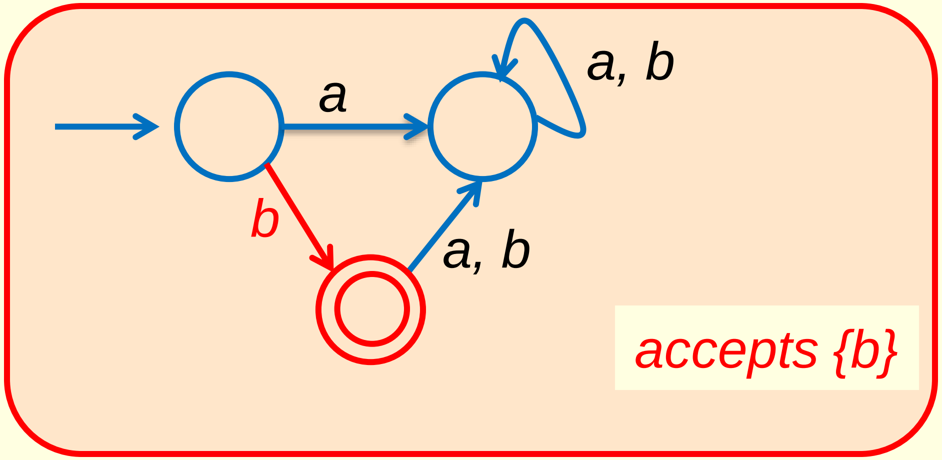
accepts {aaa}

6. Regular Languages & Finite Automata

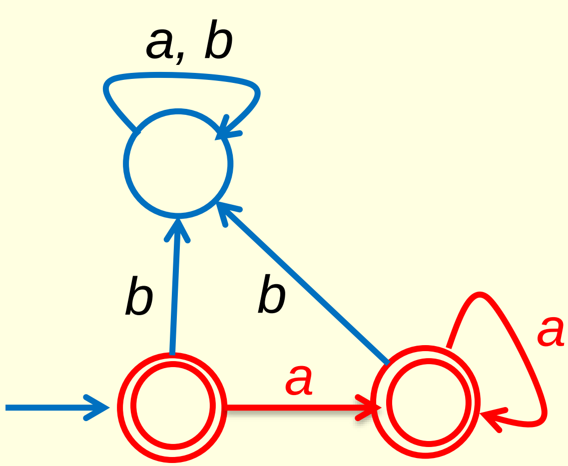
- Finite Automata

For instance, if L_1 = {a} and L_2 = {b}





then



$\boxed{L_1^*}$

accepts {a}*

6. Regular Languages & Finite Automata

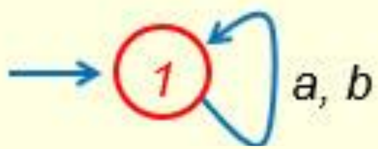
- Finite Automata

Example. Find a DFA for each language over the alphabet {a,b}.

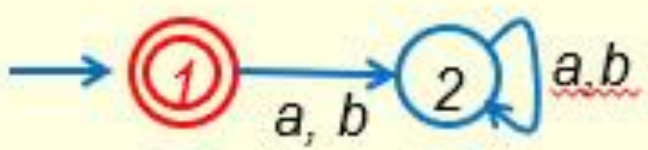
(a) ∅. (b) {Λ}. (c) $\{ (ab)^{n} \mid n \in \mathbf{N}\}$, which has regular expression (ab)*.

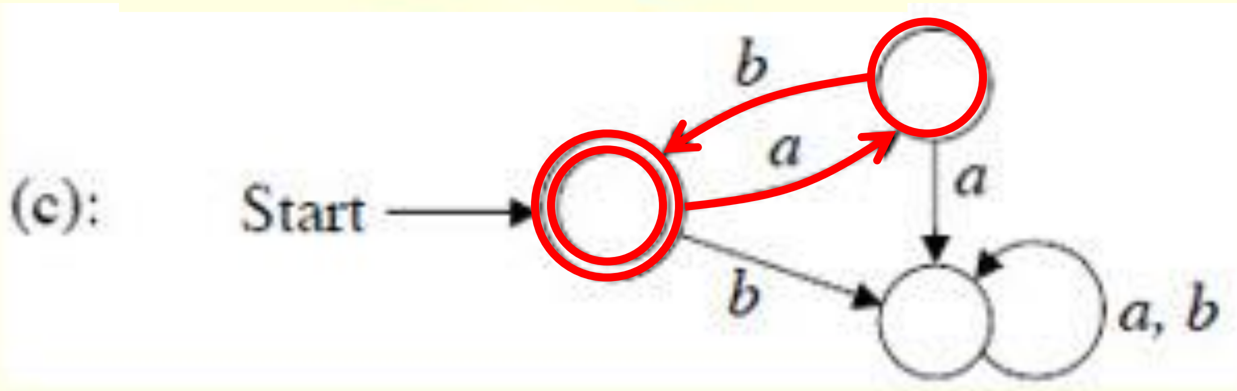
Solution:

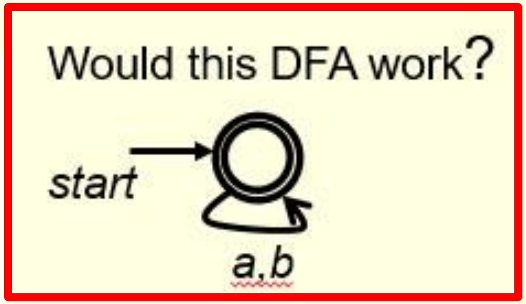
(a)



(b)

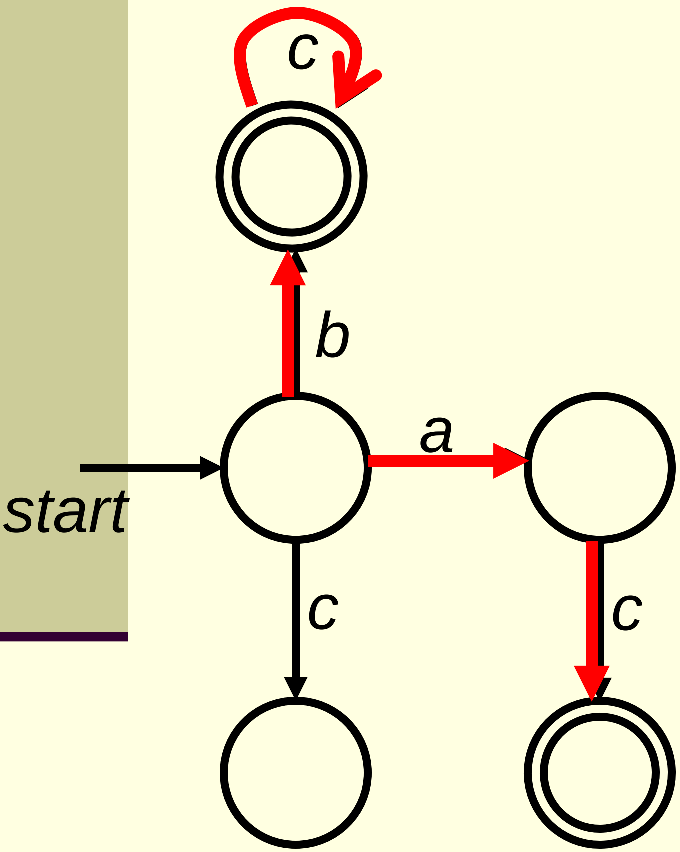




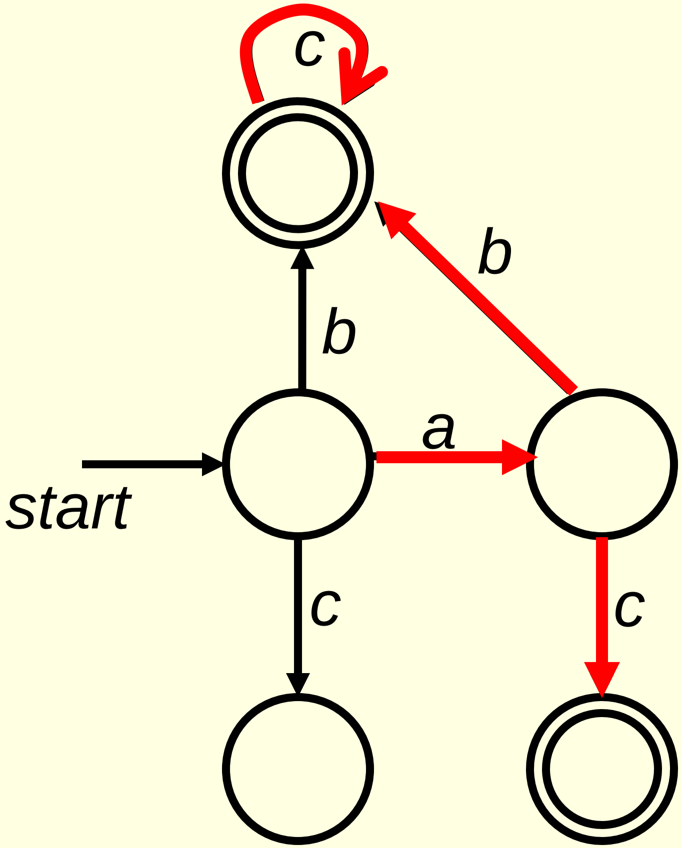


No, it accepts {a, b}*

A DFA that recognizes
bc + ac*

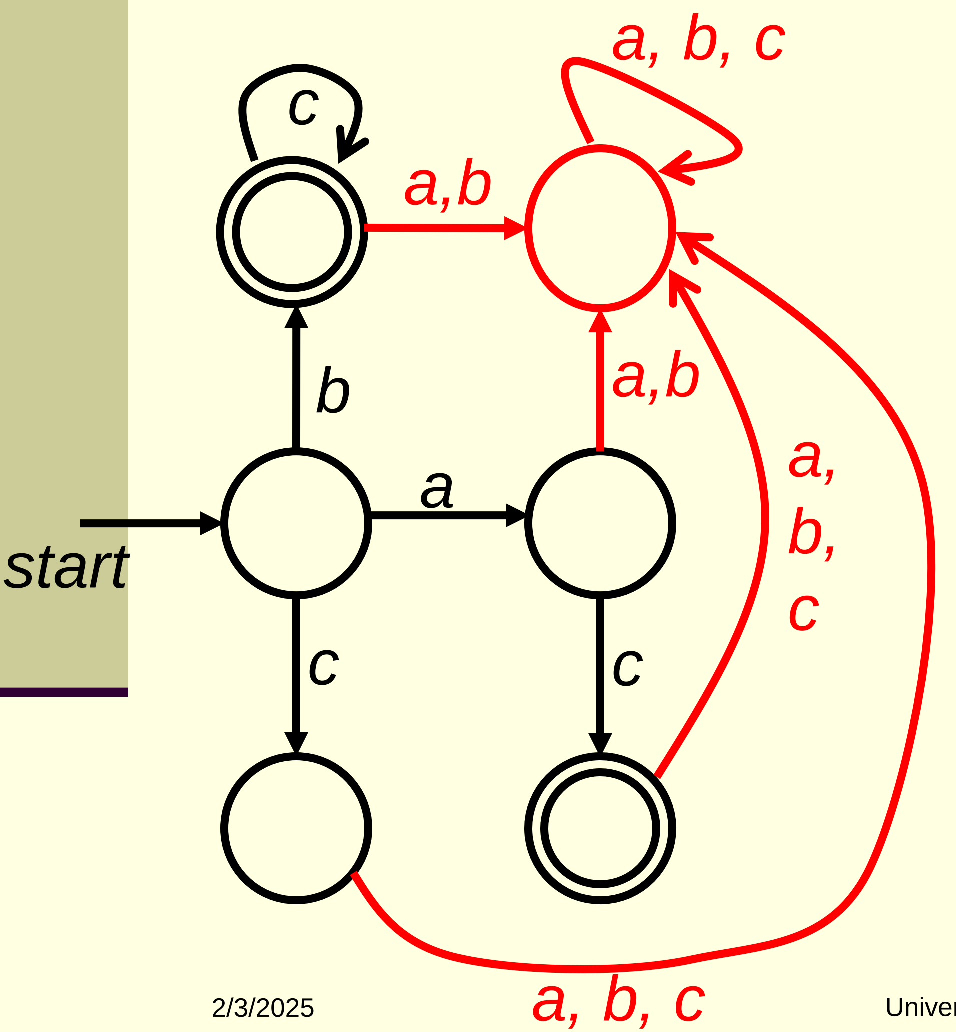


A DFA that recognizes
bc + abc* + ac*

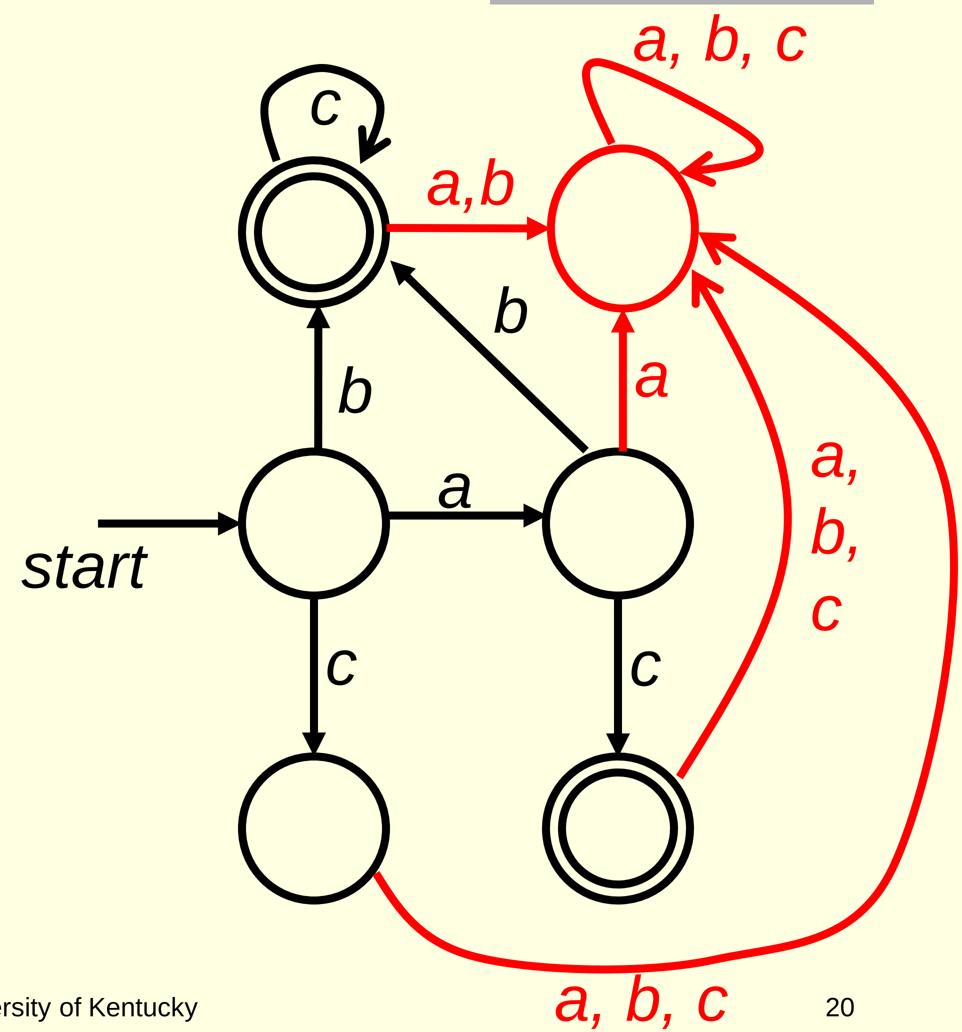


Making them deterministic:

A DFA that recognizes
bc + ac*

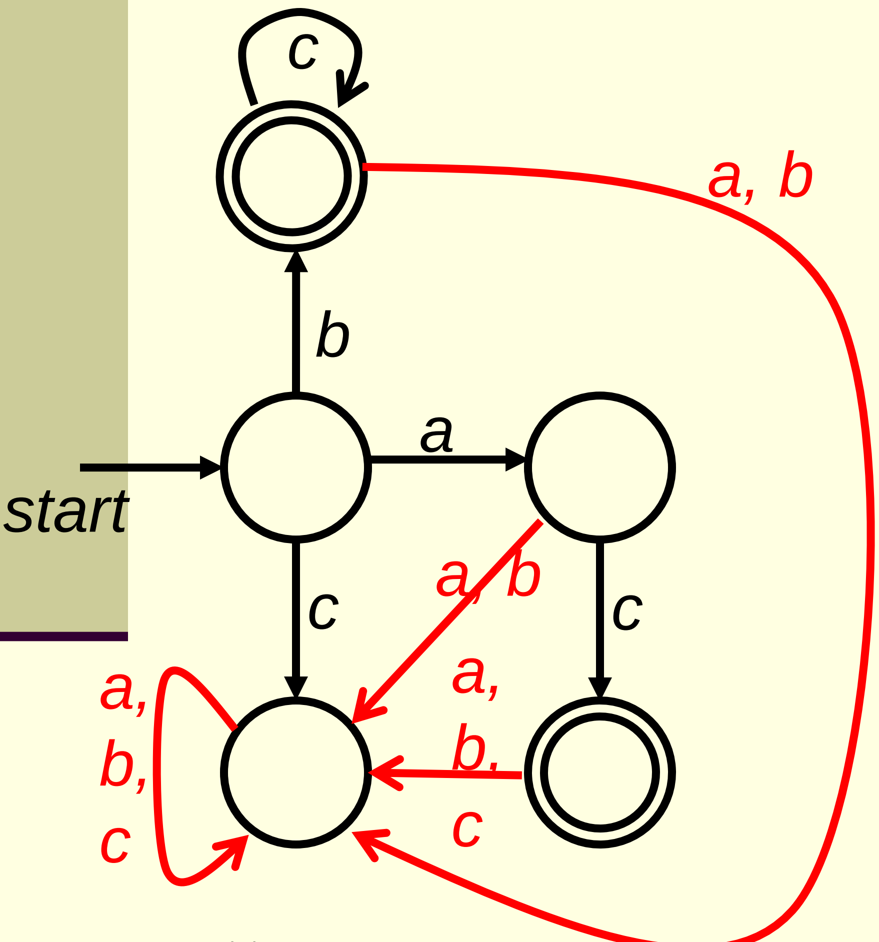


A DFA that recognizes
bc + abc* + ac*

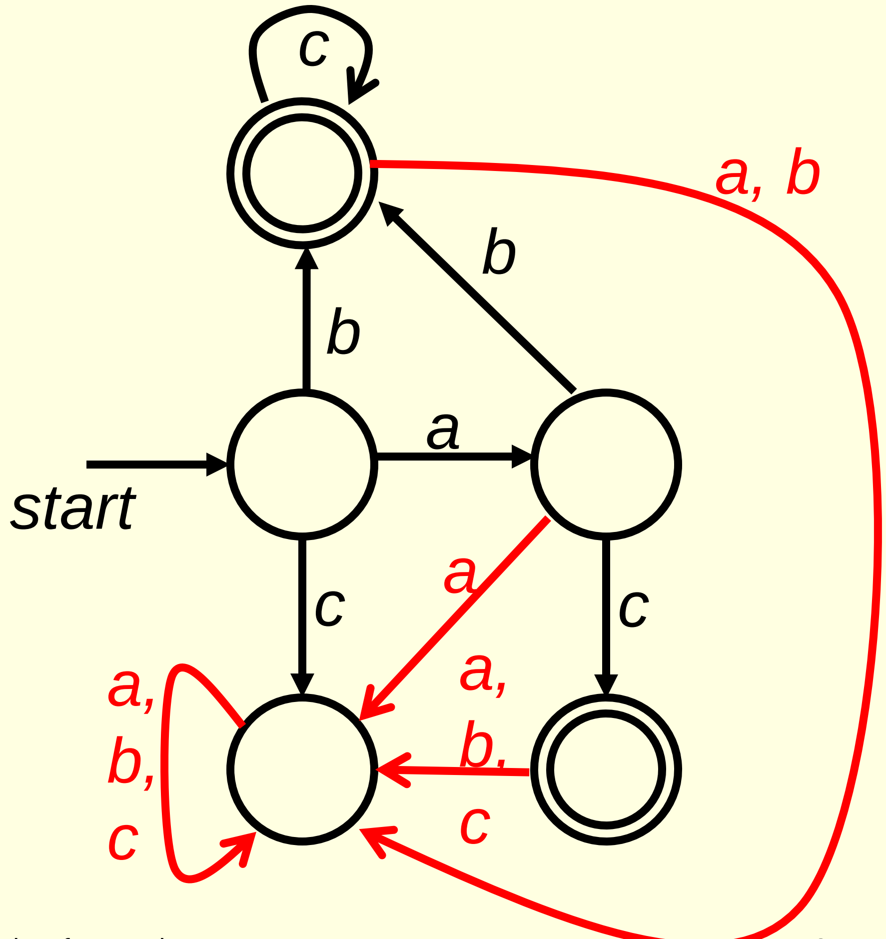


Making them deterministic:

A DFA that recognizes
bc + ac*



A DFA that recognizes
bc + abc* + ac*



To make each of these FA's a DFA, you either create a new state or use a non-final state as the sink of all the remaining edges of the FA, or both.

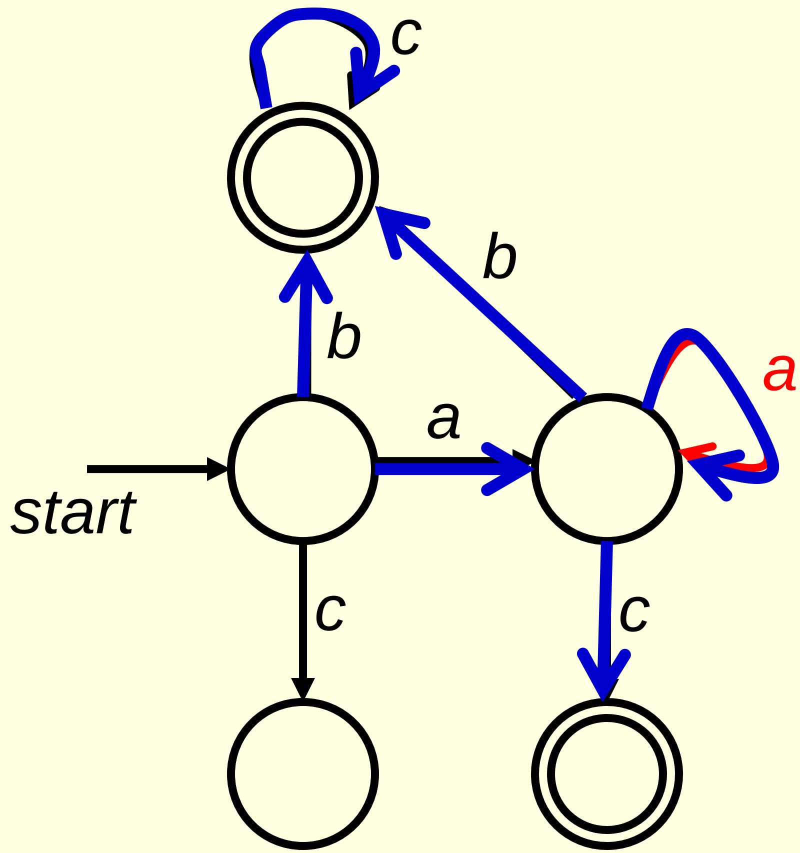
That state should not have outgoing edges, or edges that would eventually lead to final state(s).

Would the following DFA recognize *a*bc* + ac* only ?

bc*

a*bc*

ac

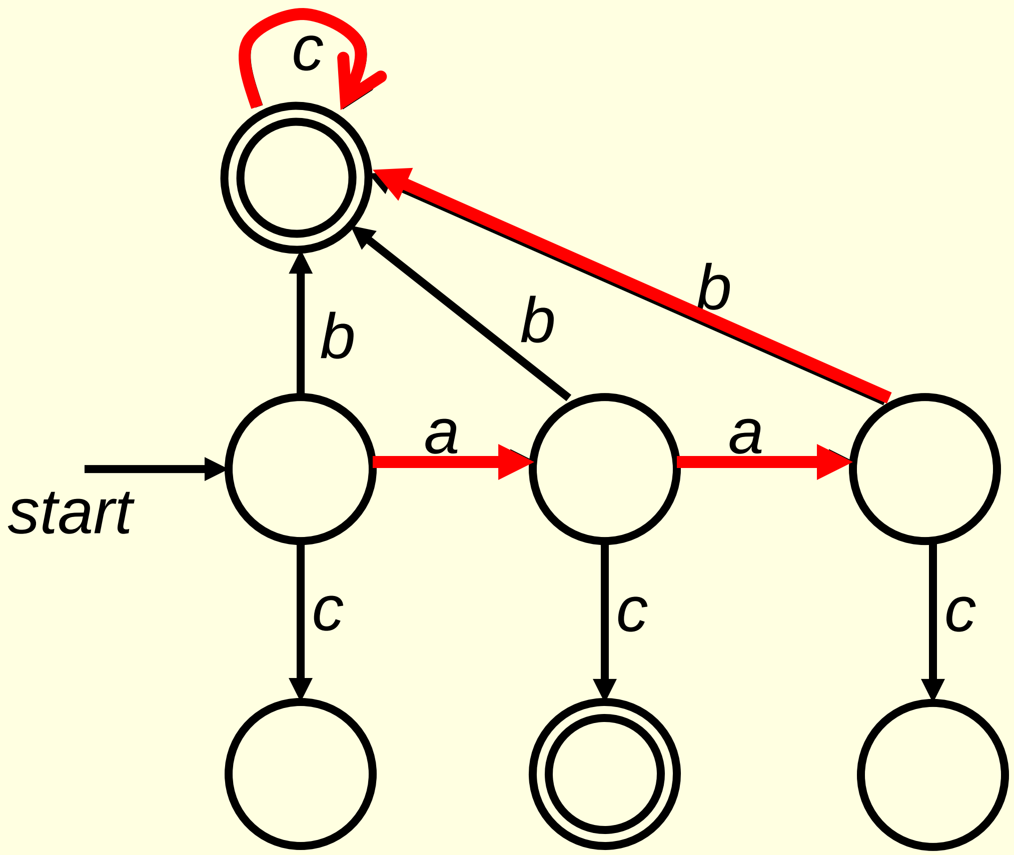


In addition to bc*, a*bc* and ac, does it recognize anything else?

Yes, such as *aac, aaac, …*

So, how should such a DFA be designed?

First, a DFA that recognizes $bc^* + abc^* + a^2bc^* + ac$

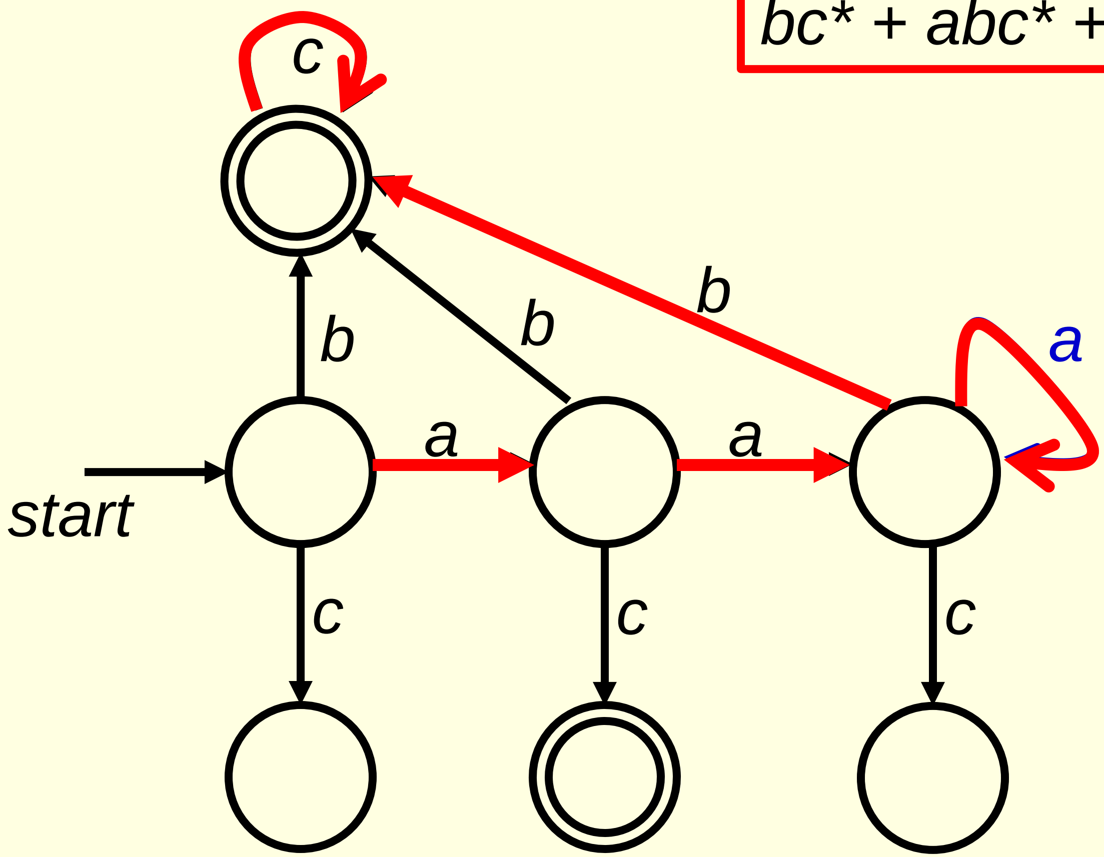


Then, a DFA that recognizes

$a^*bc^* + ac$

$bc^* + abc^* + a^2bc^* + a^nbc^*$

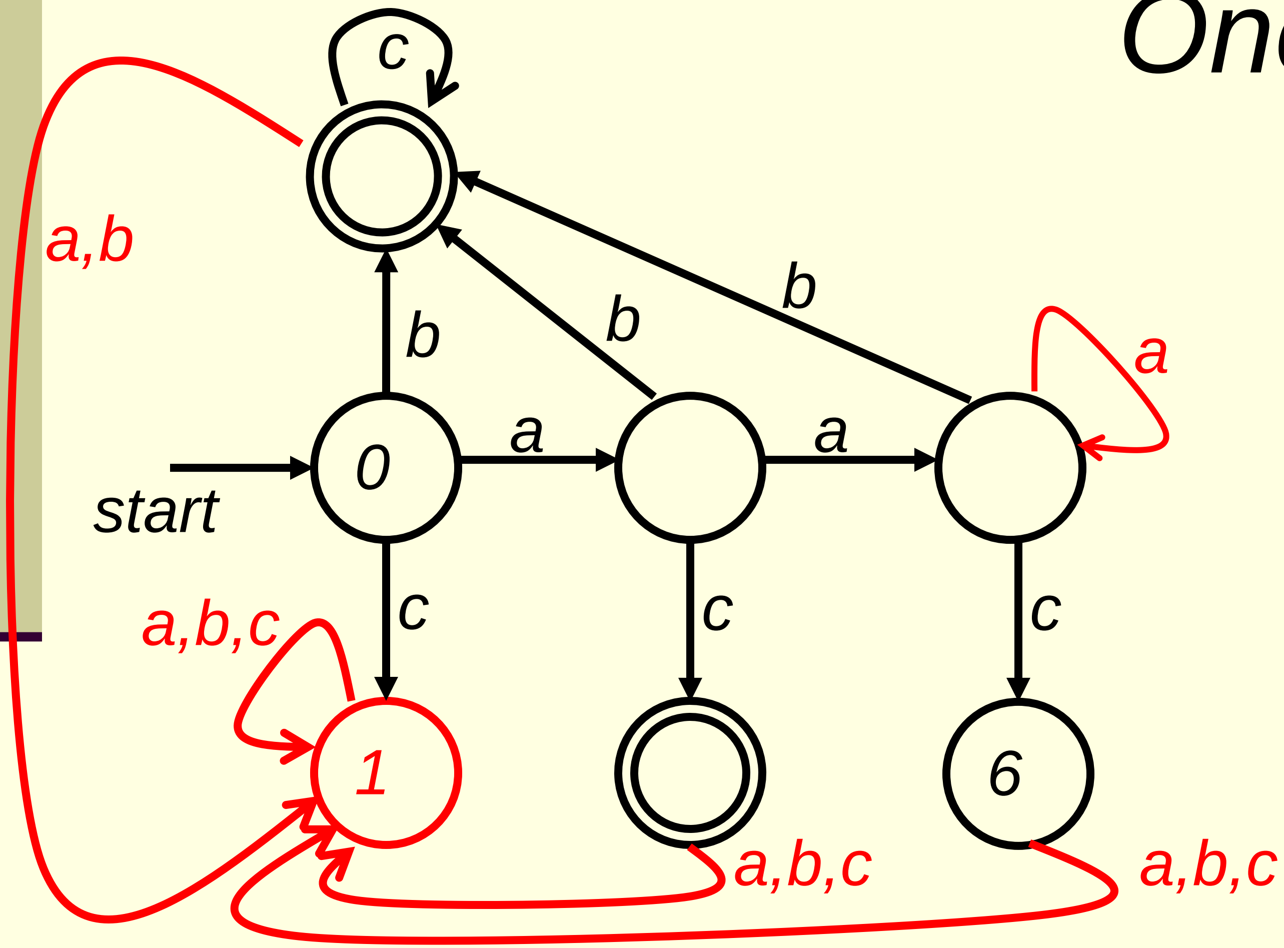
$n \geq 3$



How to make this FA a real DFA?

A DFA that recognizes a*bc* + ac

One option:



a real DFA now

Can we use state 0 as the sink state, or state 6?

6. Regular Languages & Finite Automata

- Finite Automata

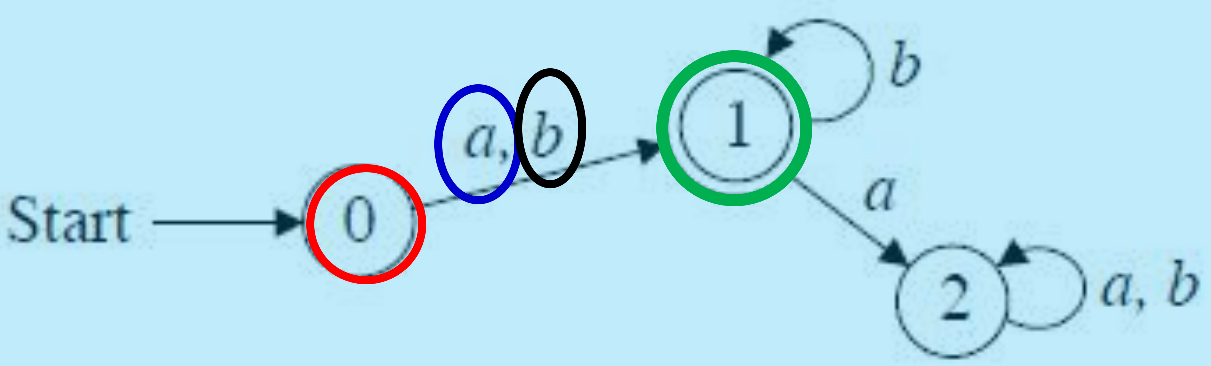
Table Representation of a DFA

DFA over A can be represented by a transition function

$$T : States \times A \rightarrow States,$$

where $T(i, a)$ is the state reached from state i along the edge labeled a, and we mark the start and final states.

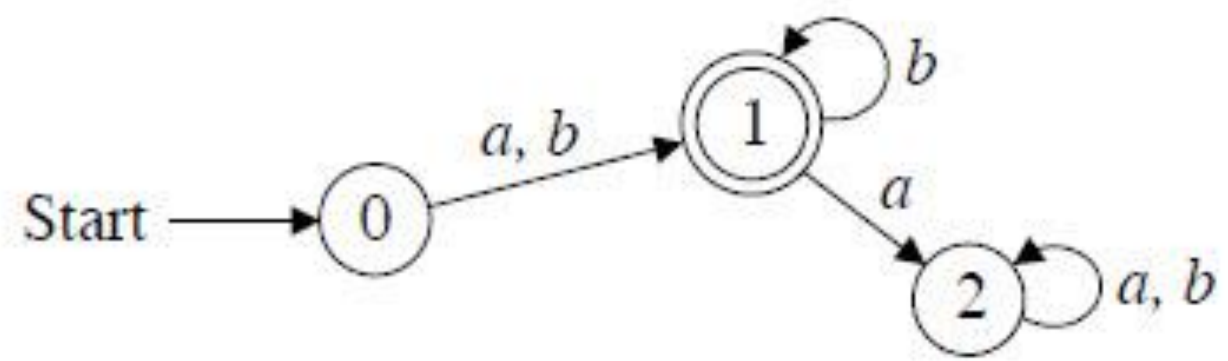
Example:



	T	a	b
start	0	1	1
final	1	2	1
	2	2	2

6. Regular Languages & Finite Automata

- Finite Automata



	T	a	b
start	0	1	1
final	1	2	1
	2	2	2

Note: *T* can be extended to *T : States × A* → States*

by $T(i, \Lambda) = i$, $T(i, aw) = T(T(i, a), w)$ $\quad a \in A$, $w \in A^*$

Question: $T(0, bba) = ?$

or

$T(0, bba) = T(1, ba) = T(1, a) = T(2, \Lambda) = 2.$

6. Regular Languag … ata

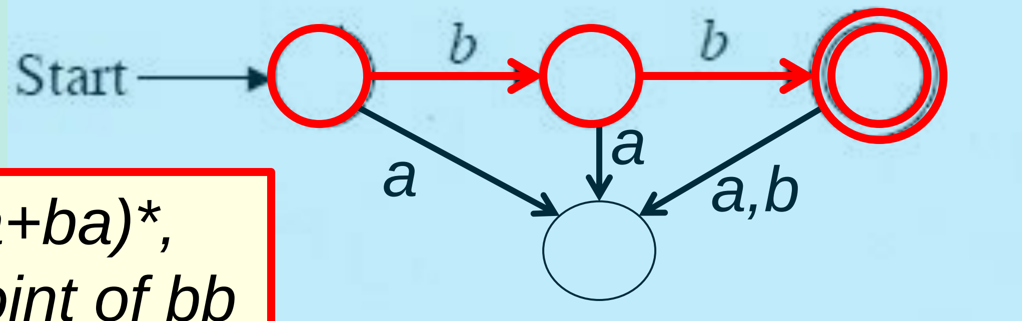
- Finite Automata

Start with the part that has a fixed length

Example. Find a DFA to recognize $(a + ba)^*bb(a + ab)^*$.

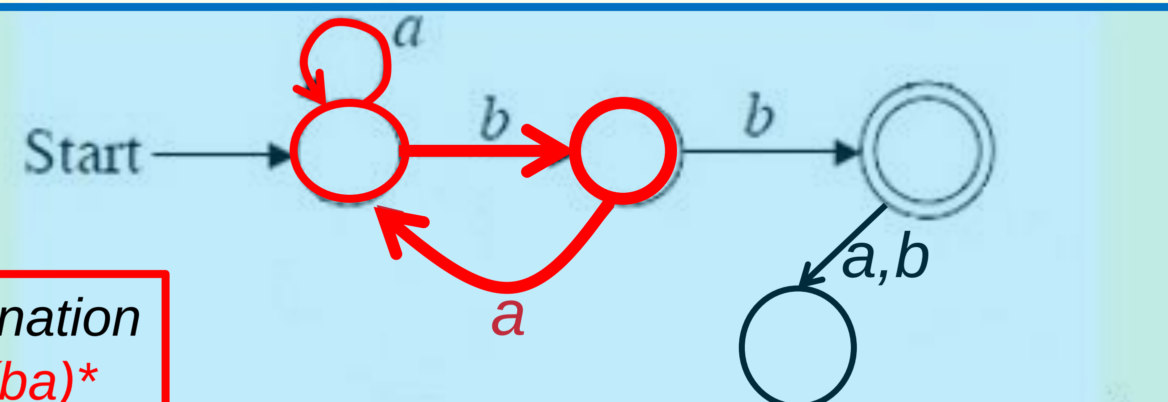
A solution:

{bb}



After excuting (a+ba)*, return to start point of bb

$\{(a+ba)^* bb\}$



any combination of a* and (ba)*

6. Regular Languages & Finite Automata

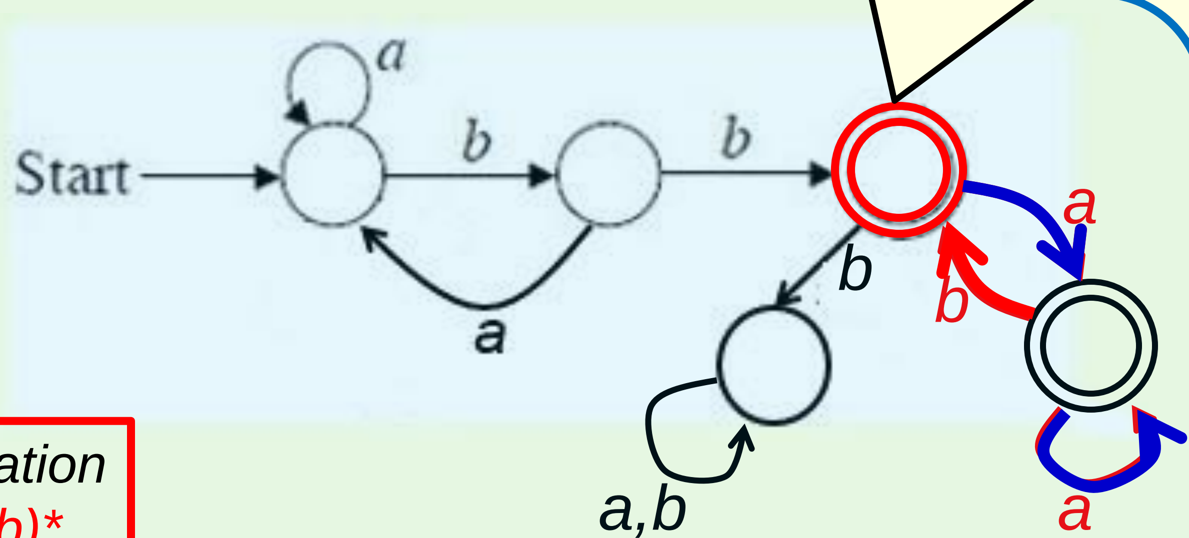
– Finite Automata

Example (conti). Find a DFA to recognize *(a + ba)*bb(a + ab)**.

A solution:

This must be a final state

$$\{(a+ba)^*\underline{bb}(a+ab)^*\}$$



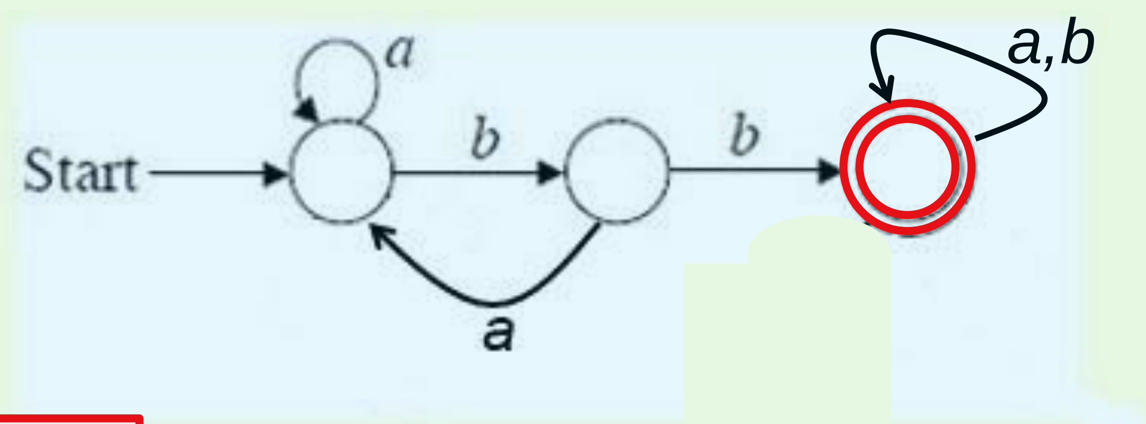
any combination of a and (ab)**

Would the following approach work?

Example (conti). Find a DFA to recognize *(a + ba)*bb(a + ab)*.*

Is this a solution?

$$\{(a+ba)^{*}\underline{bb}(a+ab)^{*}\}$$



any combination of a and (ab)**

The answer is NO

Why not?

*Would accept (ba)**

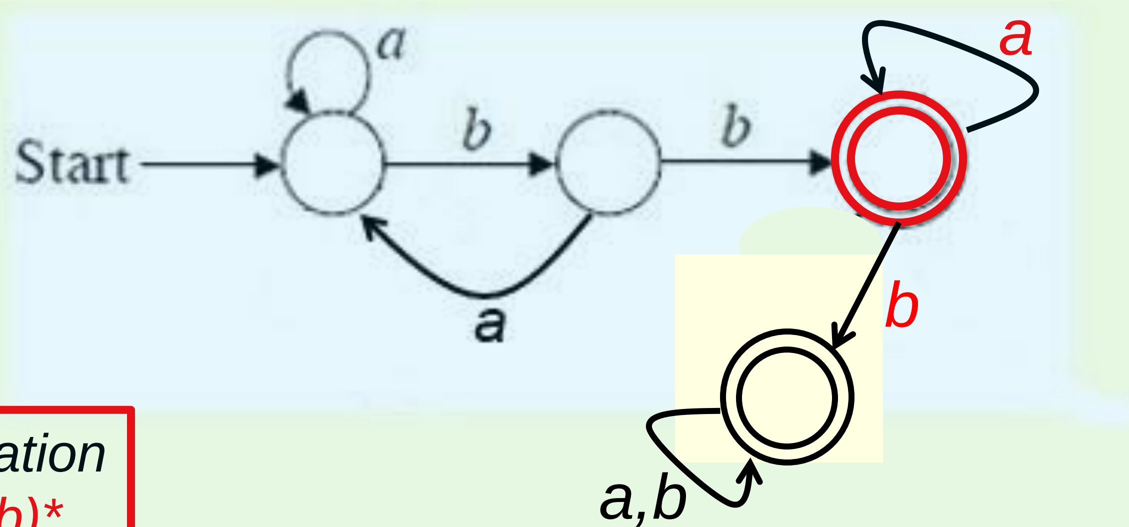
Would the following approach work?

Example (conti). Find a DFA to recognize *(a + ba)*bb(a + ab)*.*

Is this a solution?

$$\{(a+ba)^{*}\,bb\,(a+ab)^{*}\}$$

any combination of a and (ab)**



The answer is NO

Why not?

*Would accept (ba)**

6. Regular Languages & Finite Automata

- Finite Automata

Nondeterministic Finite Automata (NFA)

An NFA over an alphabet *A* is similar to a DFA except that

Λ-edges are allowed,

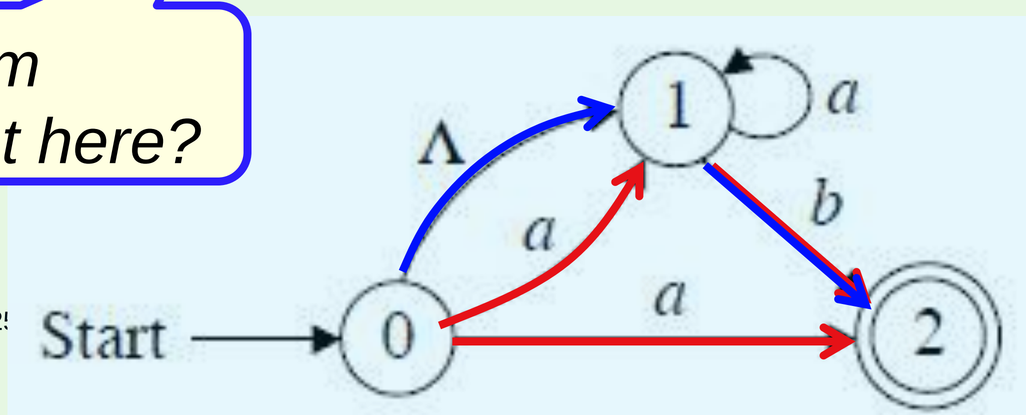
there is no requirement to emit edges from a state, and

multiple edges with the same letter can be emitted from a state.

Example. The following NFA recognizes the language of

*a + aa*b + a*b.*

Is any item redundant here?



a ?

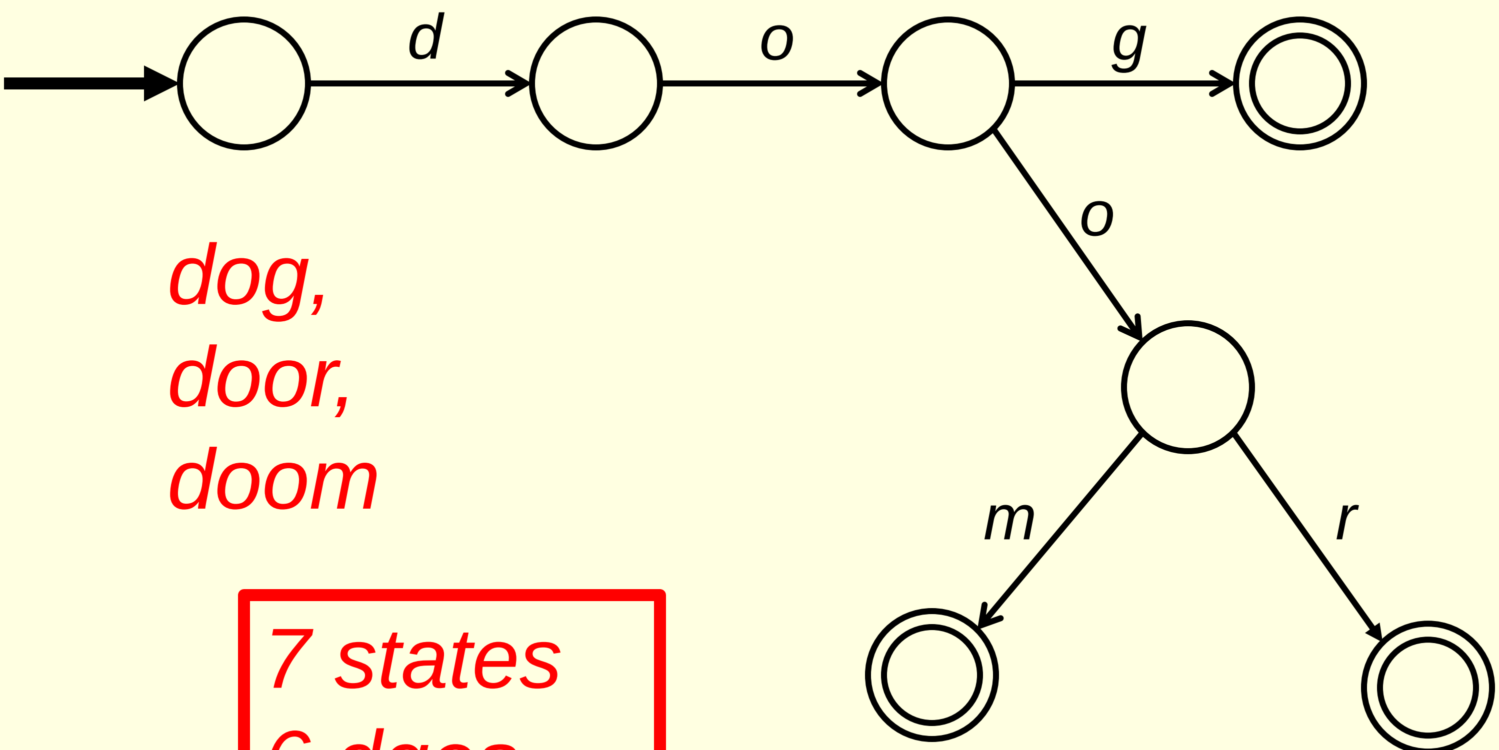
*aa*b: ab, aab ?*

*a*b: b, ab ?*

Intuitive examples

NFA

A = { d, g, m, o, r}



dog,
door,
doom

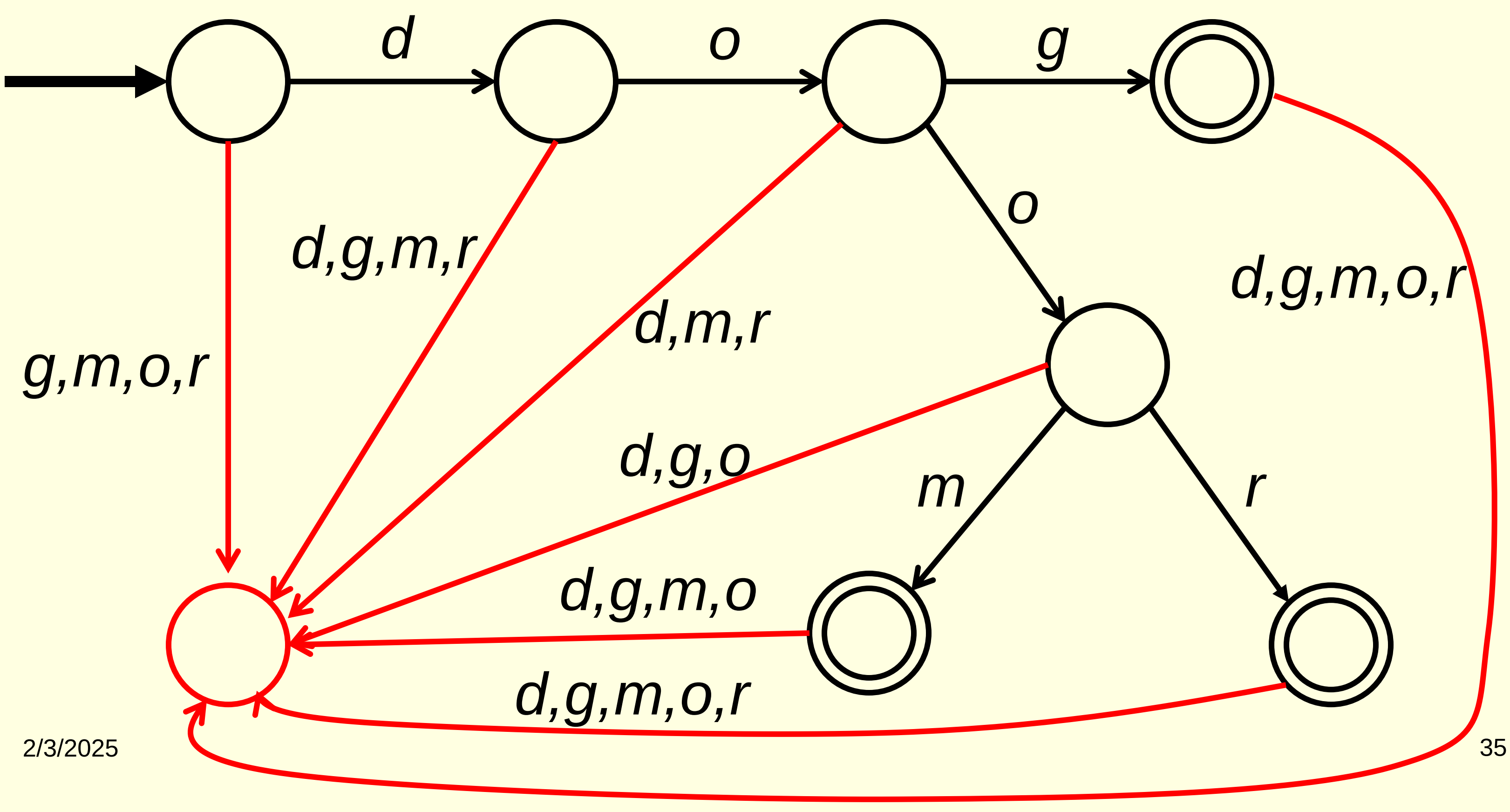
7 states
6 dges

Intuitive examples

8 states
34 edges

DFA

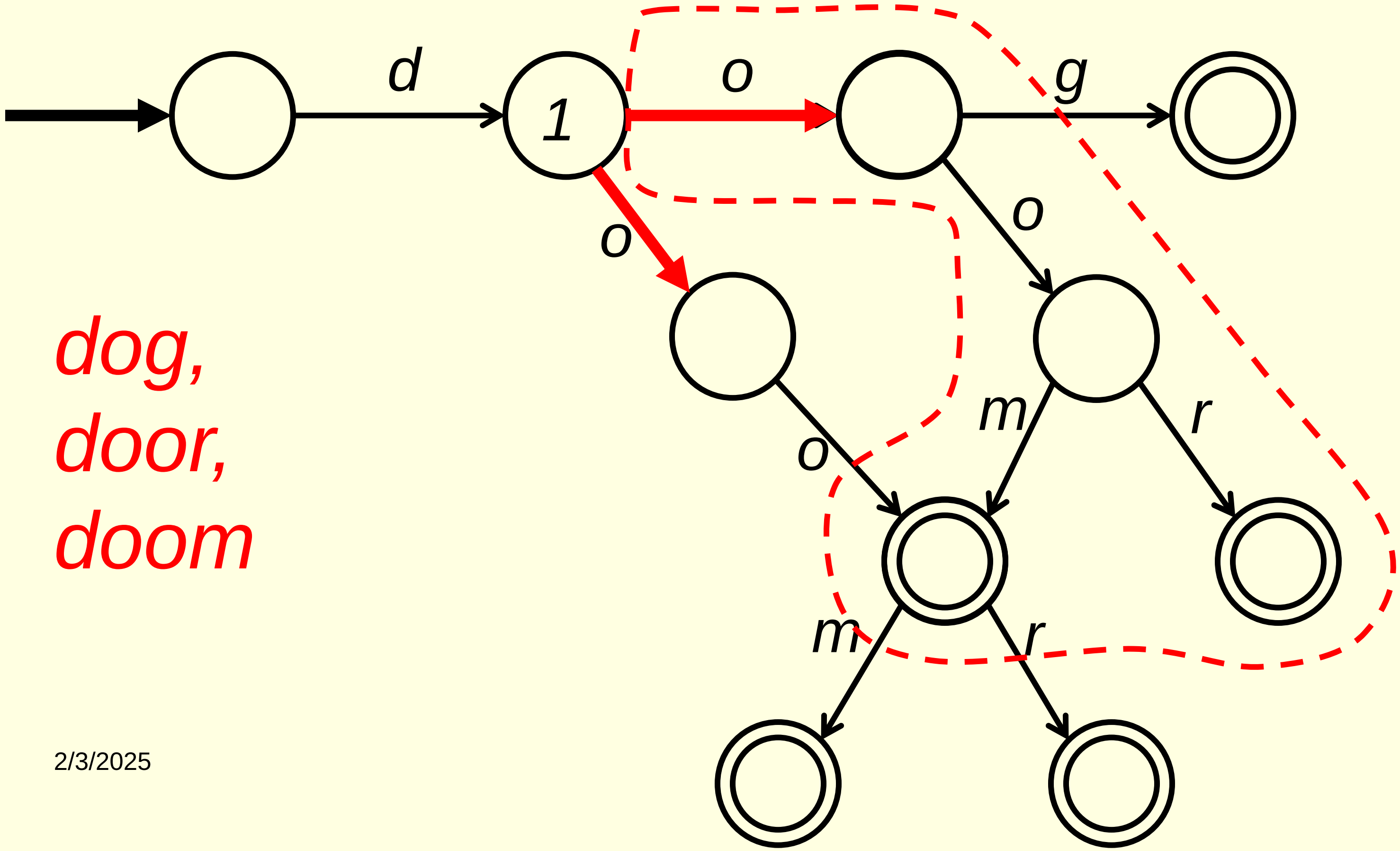
A = { d, g, m, o, r}



Intuitive examples

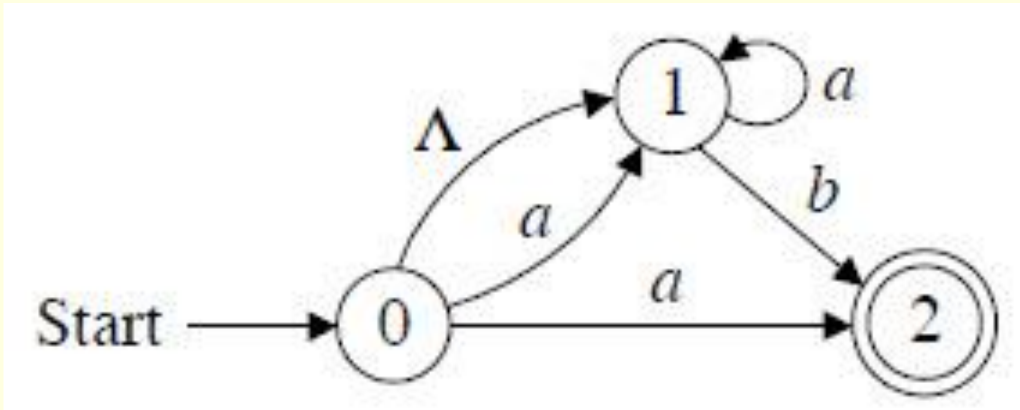
A = { d, g, m, o, r}

Actually, the NFA can also be defined as follow:



dog,
door,
doom

Example. NFA for the language of $a + aa^*b + a^*b$.

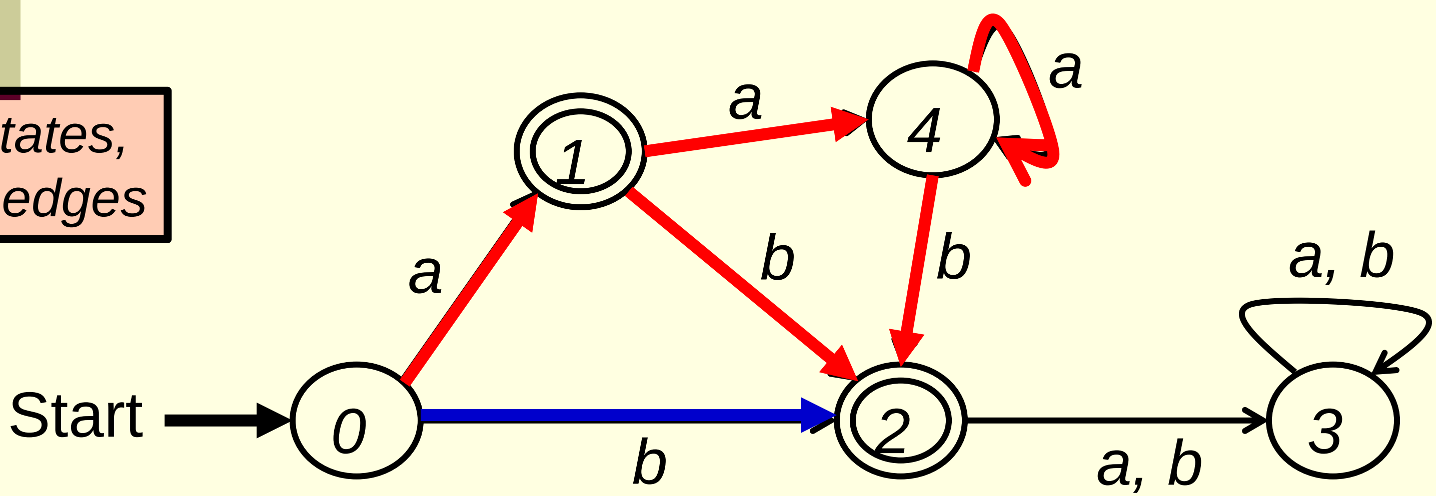


3 states,
5 edges

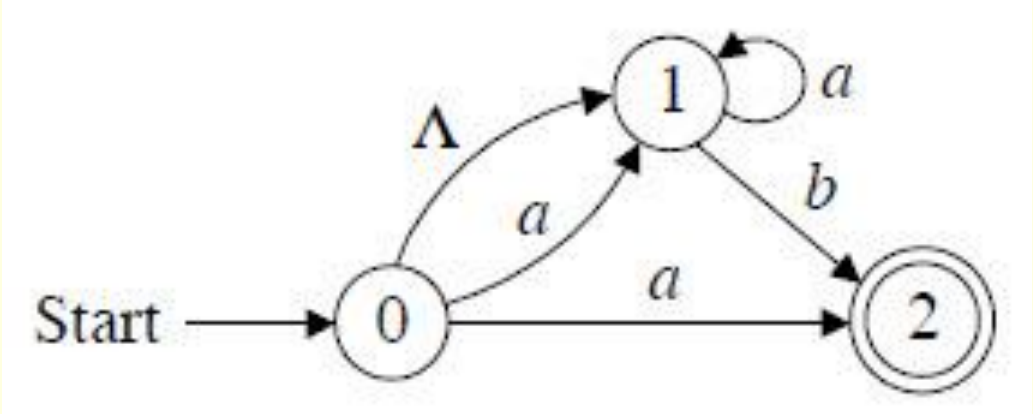
DFA for the language of $a + aa^*b + a^*b$.

a, ab, b,
aab, aaab,
…

5 states,
10 edges

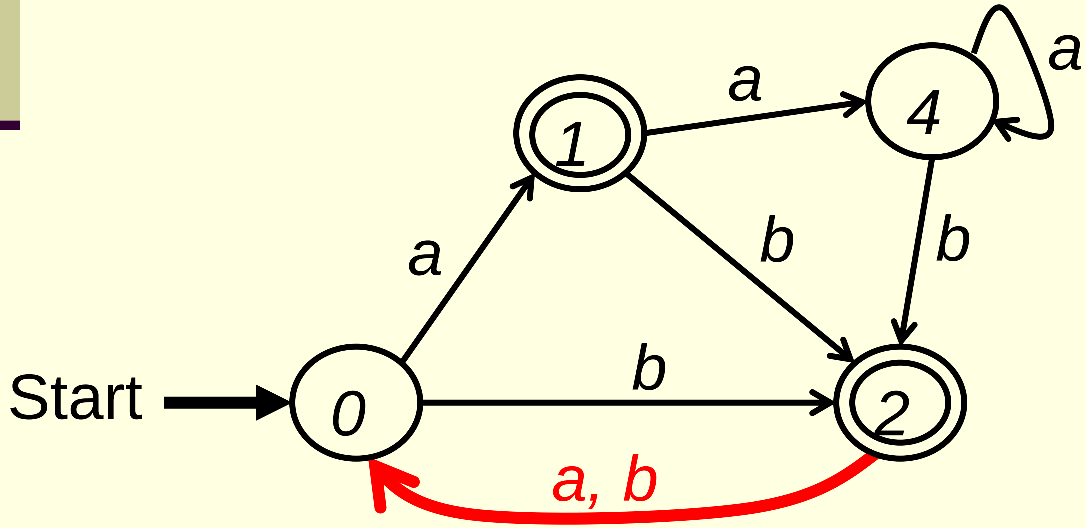


Example. NFA for the language of $a + aa^*b + a^*b$.



Would the following DFA (use state 0 to replace state 3) work?

a, b, ab, aab, aaab, …



No, b/c it accepts bbb, baa, and abbb.

DFA and NFA are equivalent concept.

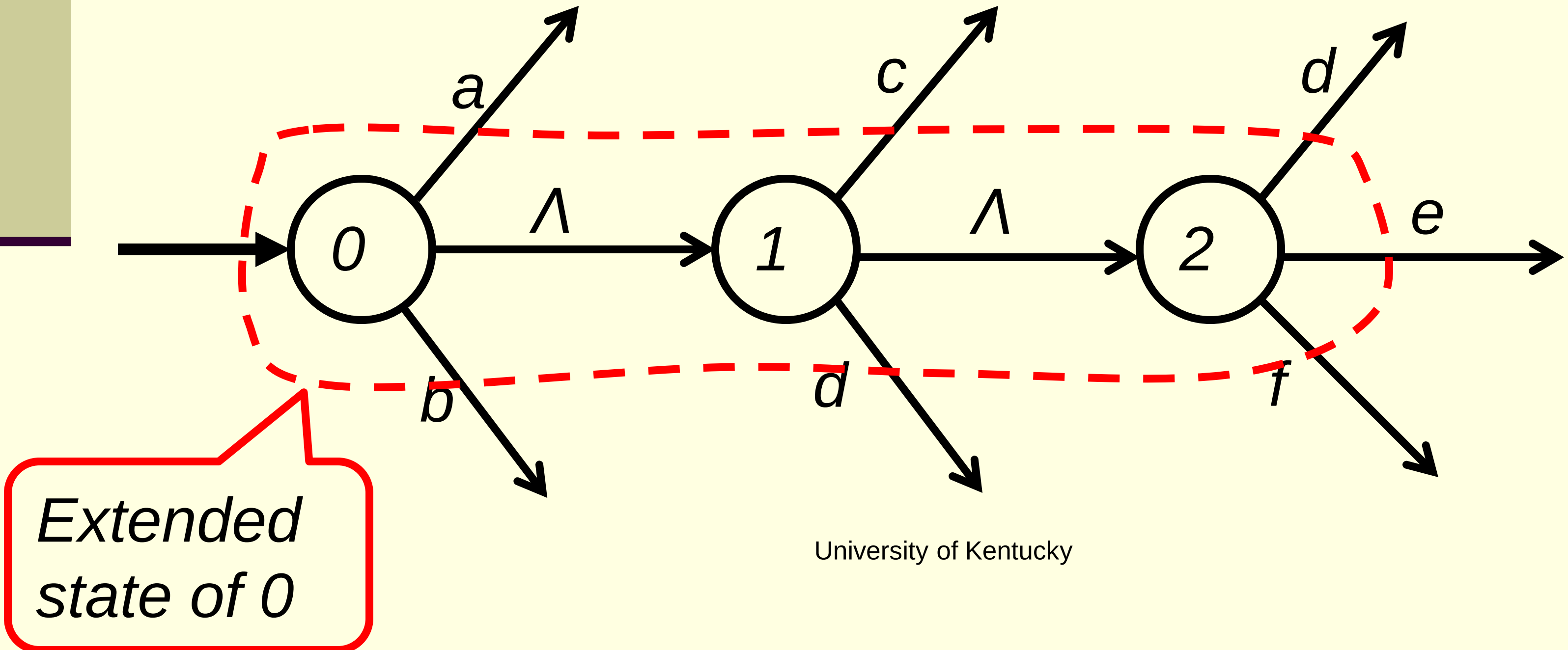
A DFA is also an NFA.

But actually for every NFA there's a corresponding DFA that accepts the same language, but if the NFA has n states, the DFA could have $O(2^n)$ states.

So we work with NFAs because they're usually a lot smaller than DFAs.

A few points about NFA's.

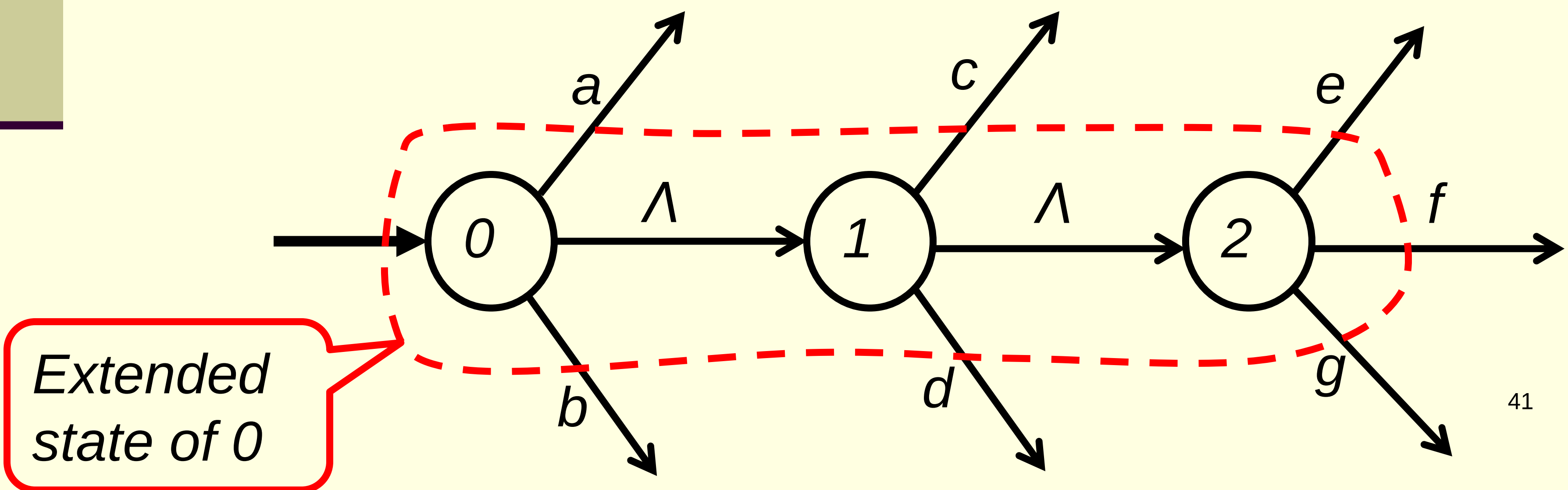
The existence of Λ-edges implicitly creates the concept of an extended state (a multiple-node state) of a given state.



Edges a, b, c, d, e, f, g are edges of the extended state of 0.

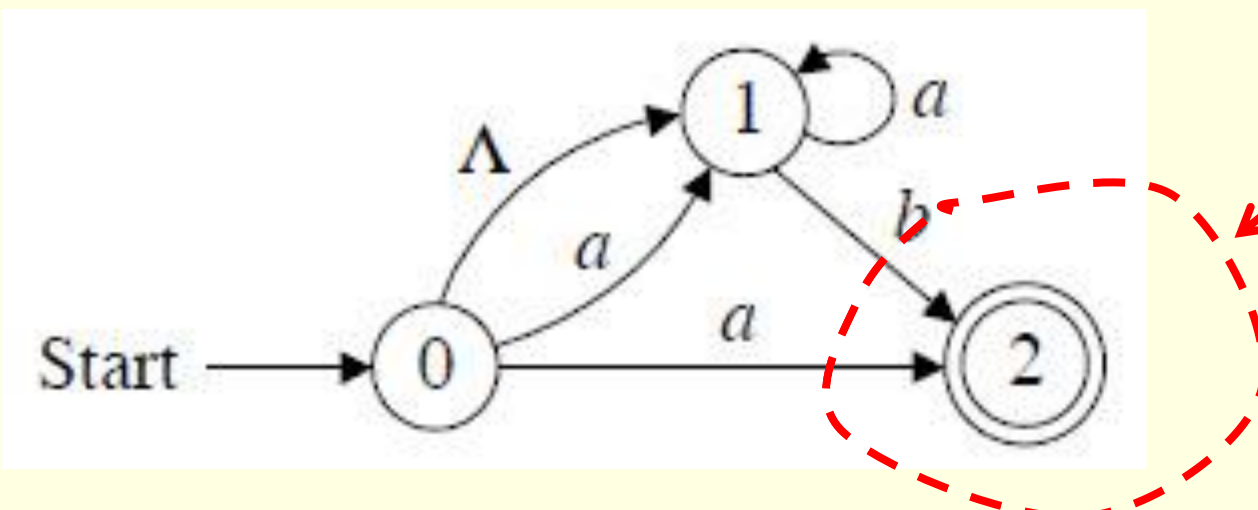
Each edge of the extended state of 0 can be used by state 0 through some Λ-edges.

So essentially state 0 has 7 edges to use even though there are only 2 edges emitted directly from state 0.



For an NFA, only edges really needed for its function have to be designed.

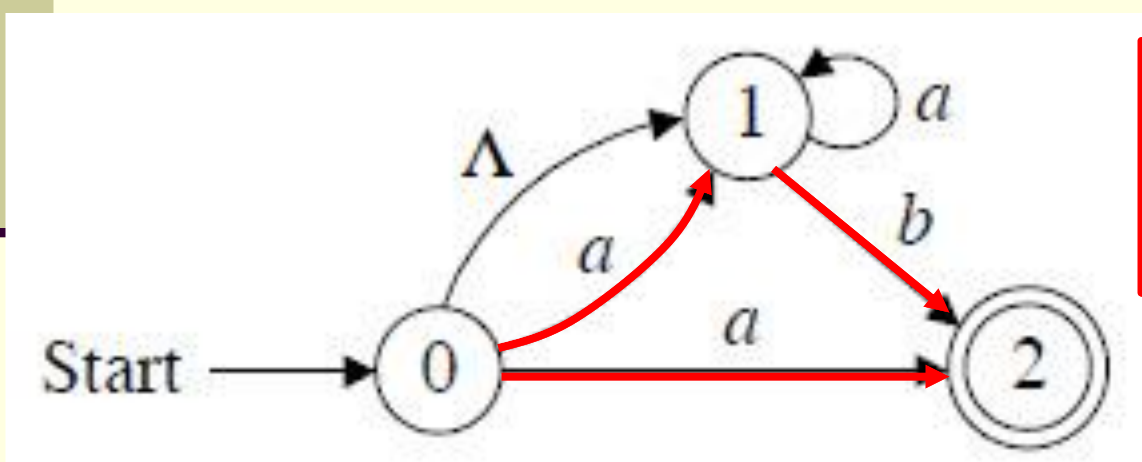
NFA for the language of *a + aa*b + a*b.*



There is no need to design any edges for state 2.

Why do we need “multiple edges with the same label”?

NFA for the language of *a* + *aa*b* + *a*b.*

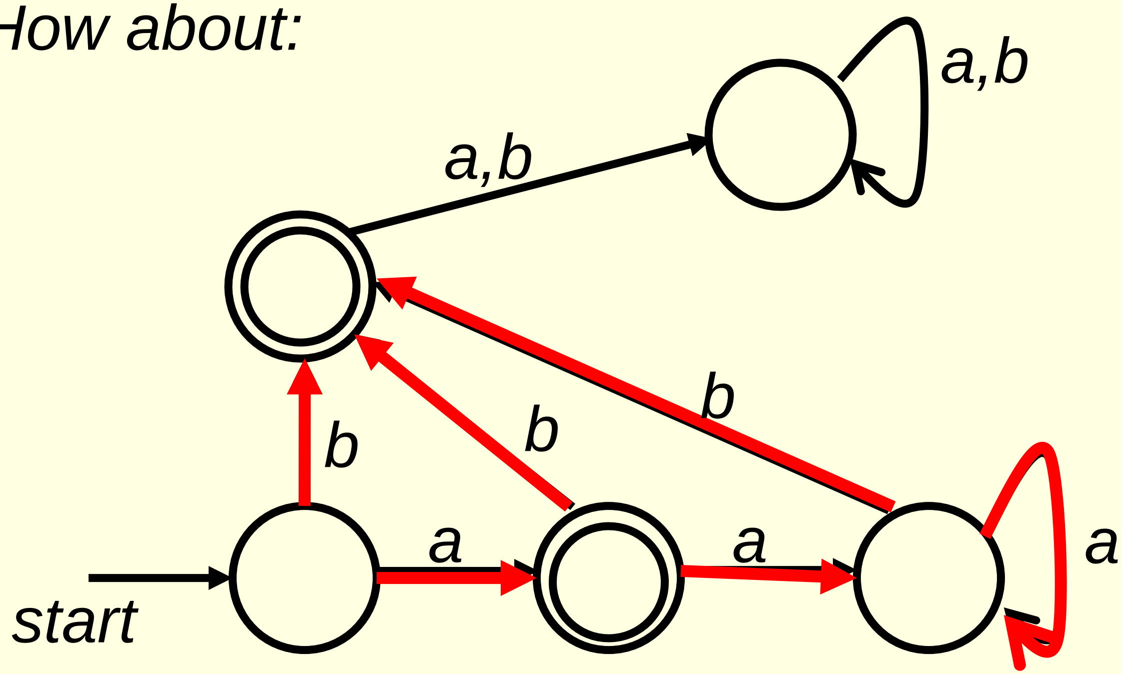


A letter can be used as the lead symbol for disjoint paths

Note that there are two ways to accept ‘ab’

Question: can you think of a DFA that would recognize *a+ aa*b + a*b* ?

How about:

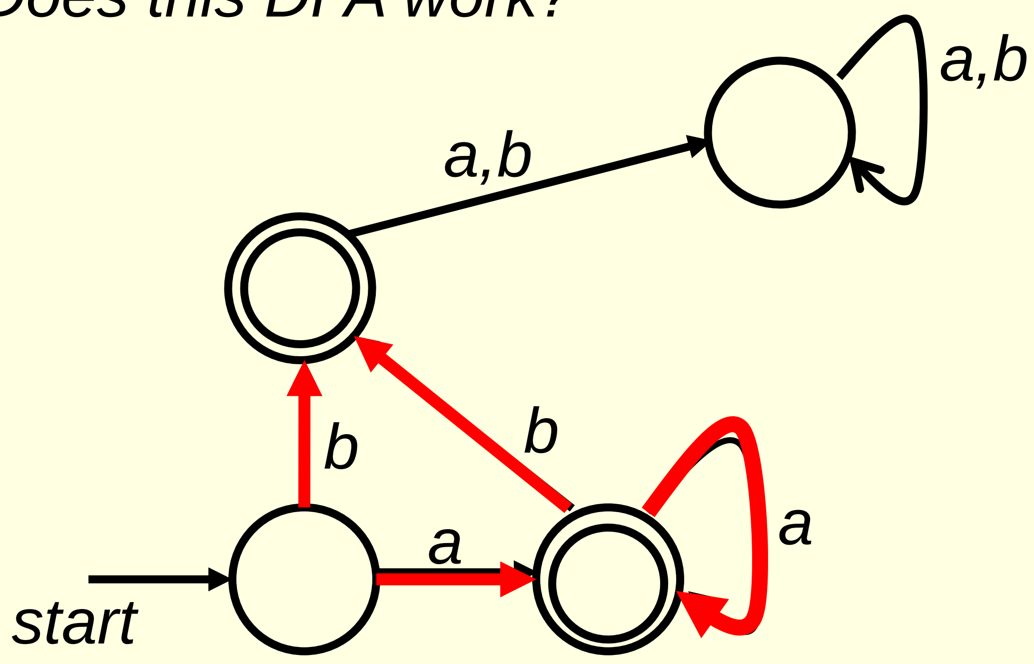


What is the difference between this DFA and the one on slide 42?

a   b   ab   aab   aaab

Question: can you think of a DFA that would recognize *a+ aa*b + a*b* ?

Does this DFA work?



a   b   ab   aab   aaab

6. Regular Languages & Finite Automata

- Finite Automata

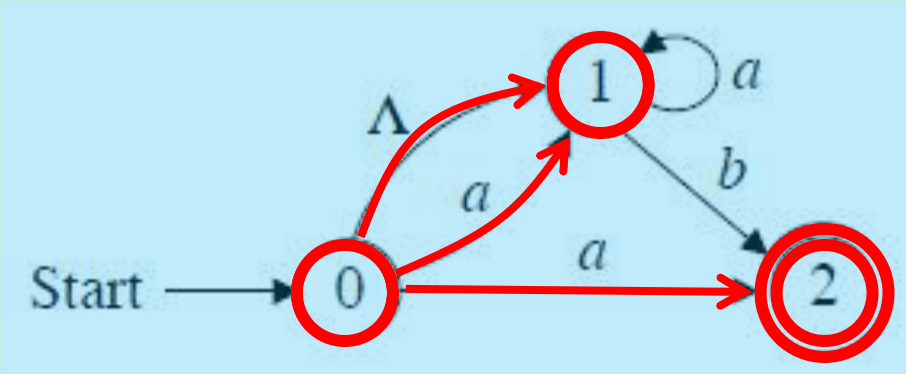
Table representation of NFA

An NFA over *A* can be represented by a function

$$T : \underline{States \times A \cup \{\Lambda\}} \rightarrow \underline{power(States)},$$

where $T(i, a)$ is the set of states reached from state i along the edge(s) labeled a, and we mark the start and final states.

Example:



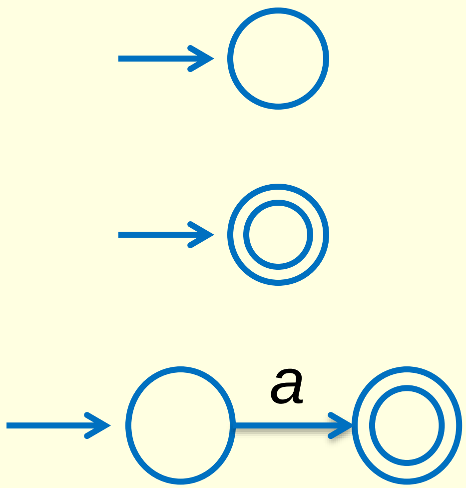
	T	a	b	Λ
start	0	{1, 2}	∅	{1}
	1	{1}	{2}	∅
final	2	∅	∅	∅

6. Regular Languages & Finite Automata

- Finite Automata

Theorem (Rabin and Scott) The class of regular languages is exactly the same as the class of languages accepted by NFAs.

Proof. First, show the three basis cases:



Language accepted by this NFA is Φ

Language accepted by this NFA is $\{\Lambda\}$

Language accepted by this NFA is $\{a\}$

6. Regular Languages & Finite Automata

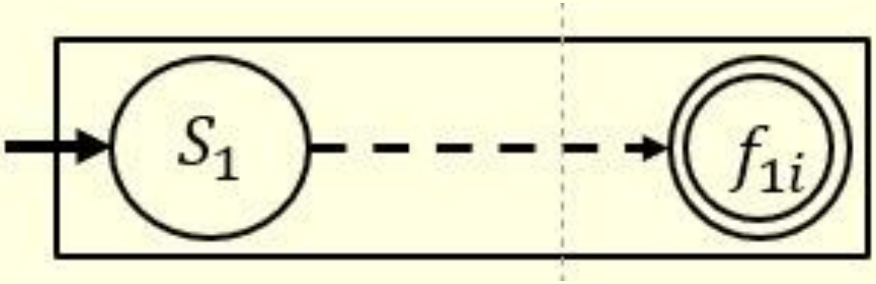
- Finite Automata

Theorem (Rabin and Scott) The class of regular languages is exactly the same as the class of languages accepted by NFAs.

Proof. (conti.)

Inductive step: prove that if L_1 and L_2 are accepted by NFAs, then $L_1 \cup L_2$, $L_1 L_2$ and L_1^* are accepted by NFAs.

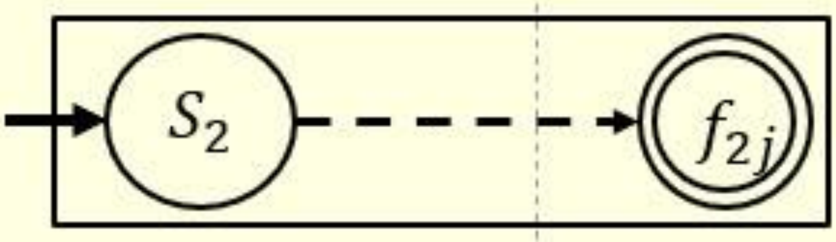
Let the following NFA be the NFA for L_1 where f_{1i} are final states of this NFA, $i = 1, 2, \dots, N_1$.



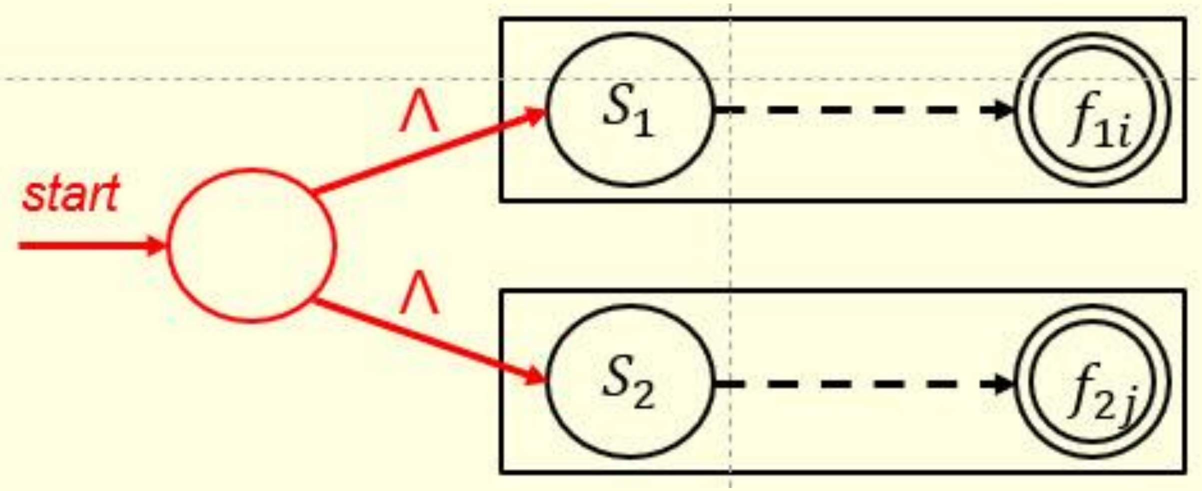
6. Regular Languages & Finite Automata

- Finite Automata

Let the following NFA be the NFA for L_2 where f_{2j} are final states of this NFA, $j = 1, 2, \dots, N_2$.



Then the following NFA is an NFA for $L_1 \cup L_2$:

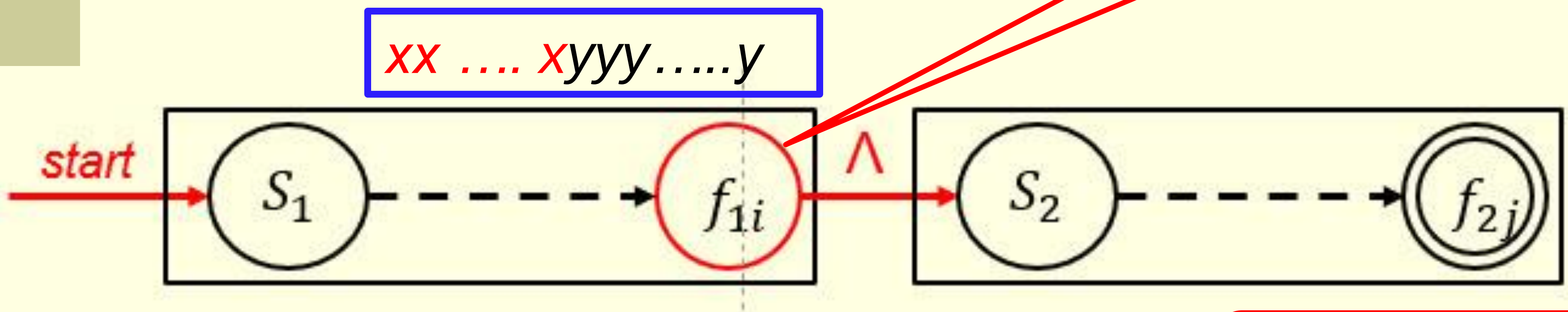


6. Regular Languages & Finite Automata

- Finite Automata

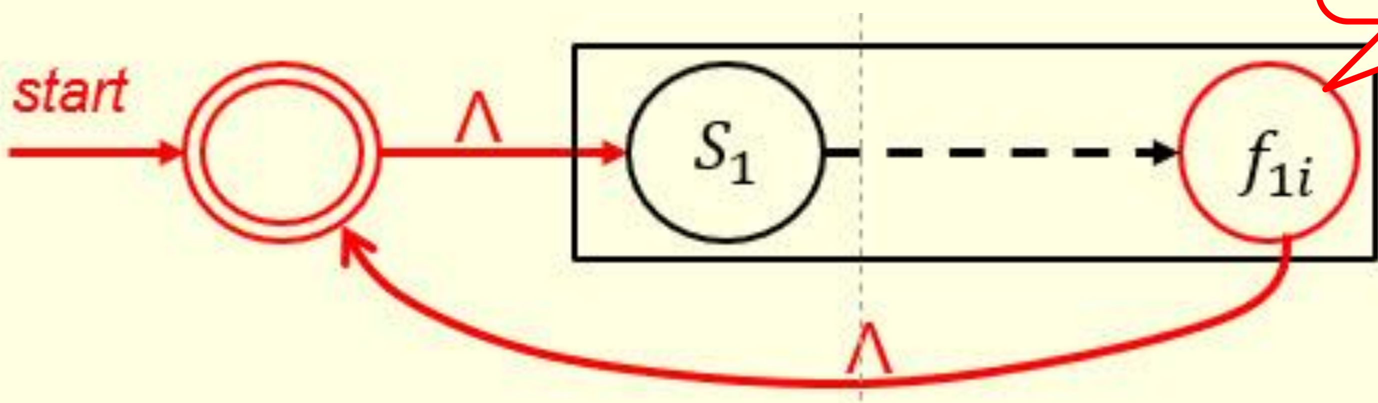
Each f_{1i} is connected to S_2

the following NFA is an NFA for L_1L_2:



And the following NFA is an NFA for L_1^* :

Each f_{1i} is connected to S_2



EOP

Rabin and Scott's theorem is important

Regular languages

(So we get this equivalence relation automatically. We don't really need Kleene's Theorm)

(Rabin and Scott Theorm)

DFAs

NFAs

(proved next)

Joke for today:

I am 83 years old.

I was in the McDonald's drive-thru this morning. The young lady behind me leaned on her horn **and** started mouthing some ugly things b/c I was taking too long to place my order.

So when I got to the 1st window, I paid for her order along **with** my own. The cashier must have told her what I'd done, b/c as we moved up, she leaned out her window **and** waved to me **and** she began mouthing "Thank you, thank you" probably feeling embarrassed that I had repaid her rudeness with kindness.

When I got to the 2nd window, I showed the server both receipts and I took her food too.

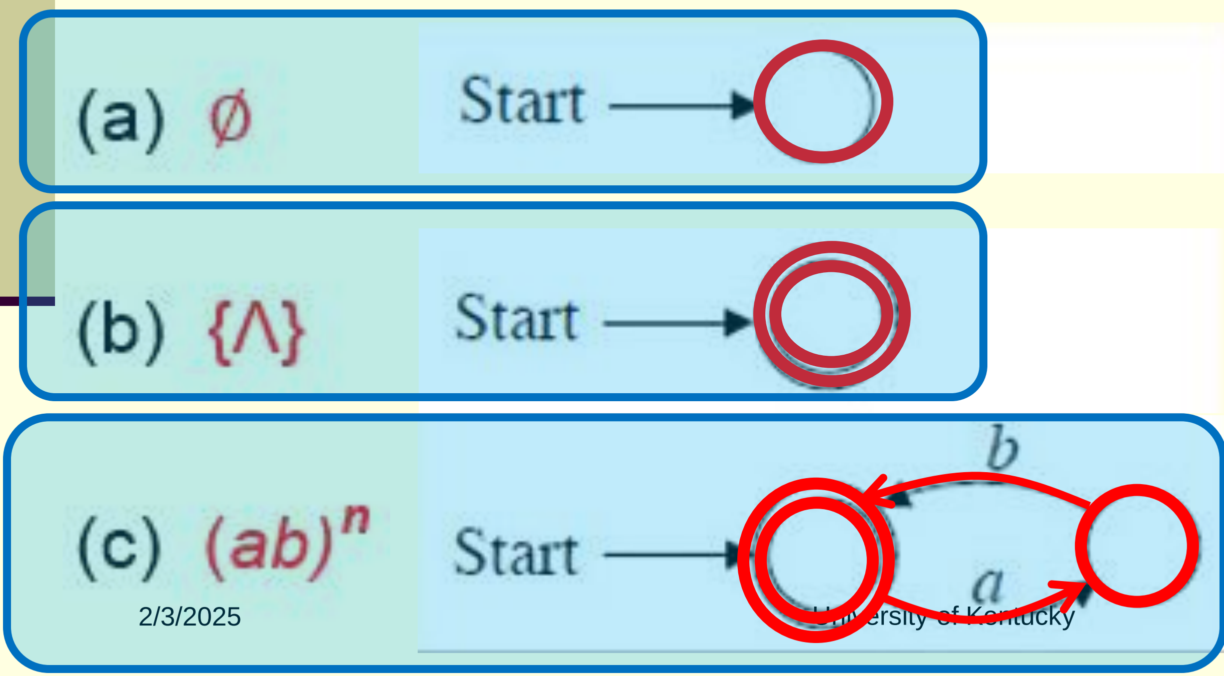
Now she has to go back to the end of the queue **and start** all over again.

Don't blow your horn at old people, we have been around for a lon---g time.

6. Regular Languages & Finite Automata

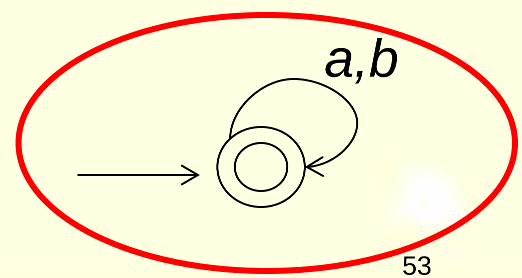
- Finite Automata

Questions. Find an NFA for each of the languages over *{a, b}*.



= ?

$\{a^*, b^*, (ab)^*\}$



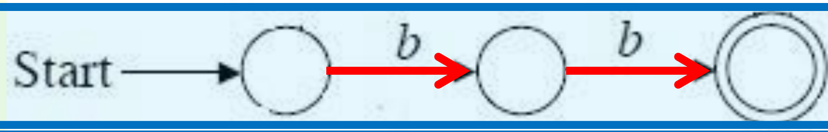
6. Regular Languages & Finite Automata

– Finite Automata

Example. Find an NFA to recognize *(a + ba)*bb(a + ab)*.*

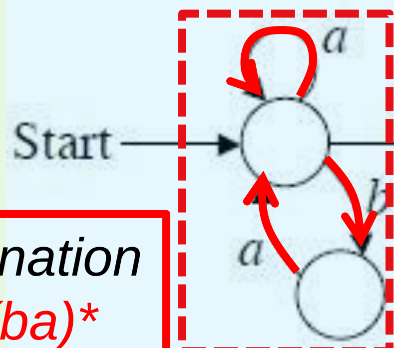
A solution:

$\{bb\}$



$\{(a+ba)^*bb\}$

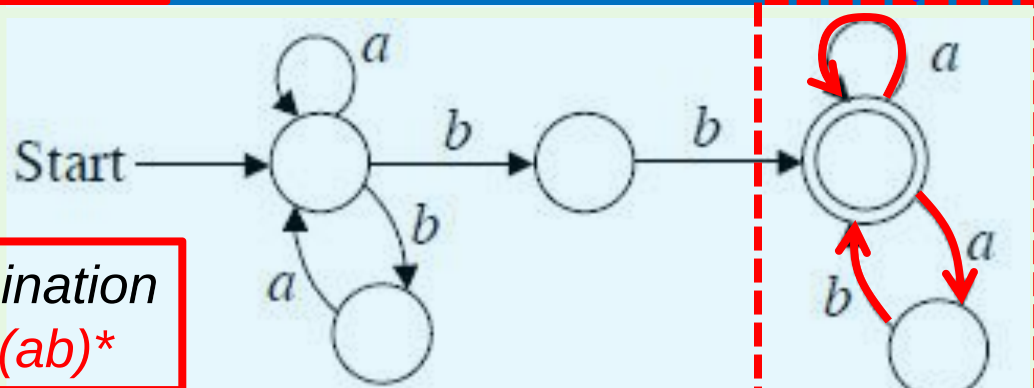
any combination of a and (ba)**



OK for an NFA, not for a DFA

$\{(a+ba)^*bb(a+ab)^*\}$

any combination of a and (ab)**

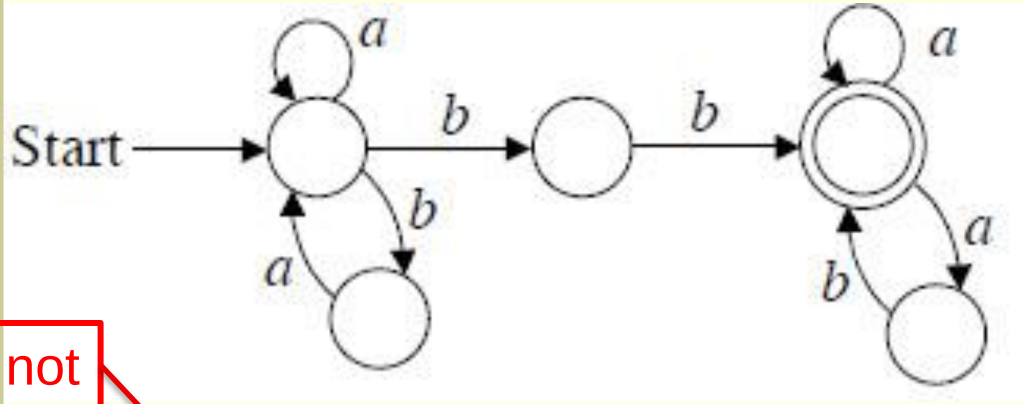


6. Regular Languages & Finite Automata

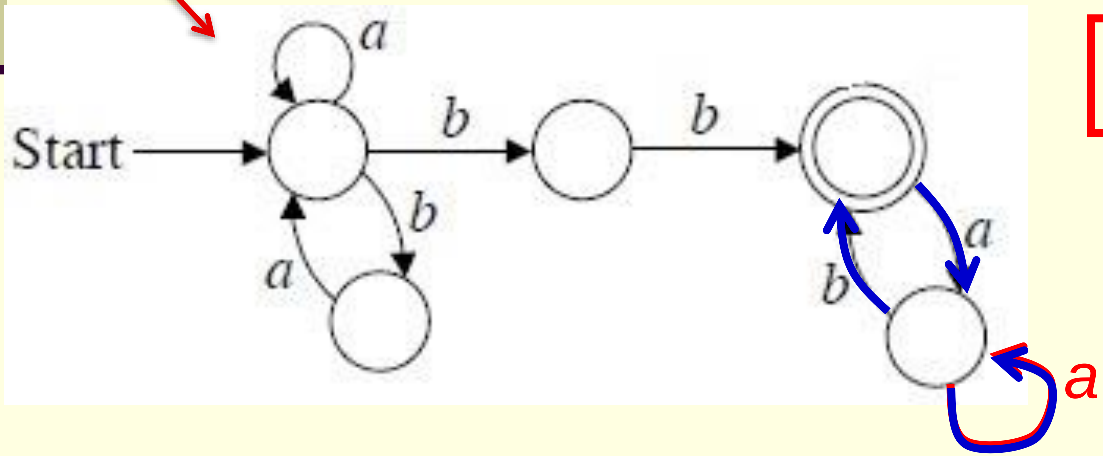
- Finite Automata

Example (conti). Find an NFA to recognize $(a + ba)^*bb(a + ab)^*$.

A solution:



But not



$$\{(a+ba)^* bb(aa^*b)^*\}$$

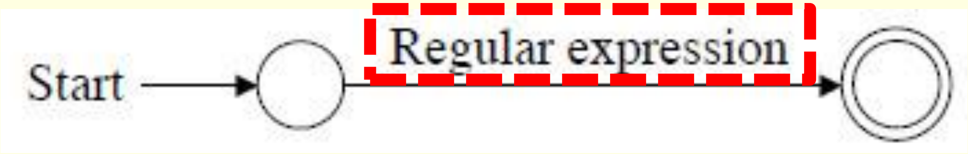
6. Regular Languages & Finite Automata

- Finite Automata

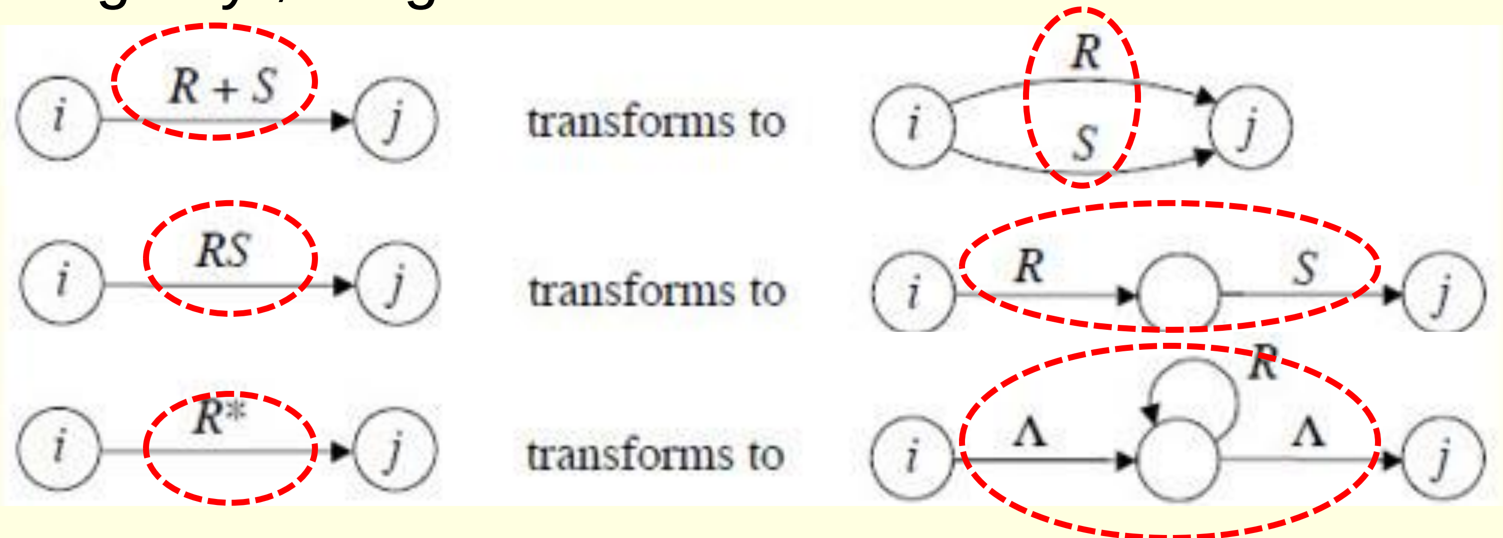
(DFA or NFA)

Algorithm: Transform a *Regular Expression (RE)* to a *Finite Automaton*

(1) Placing the RE on the edge between a start and final state:



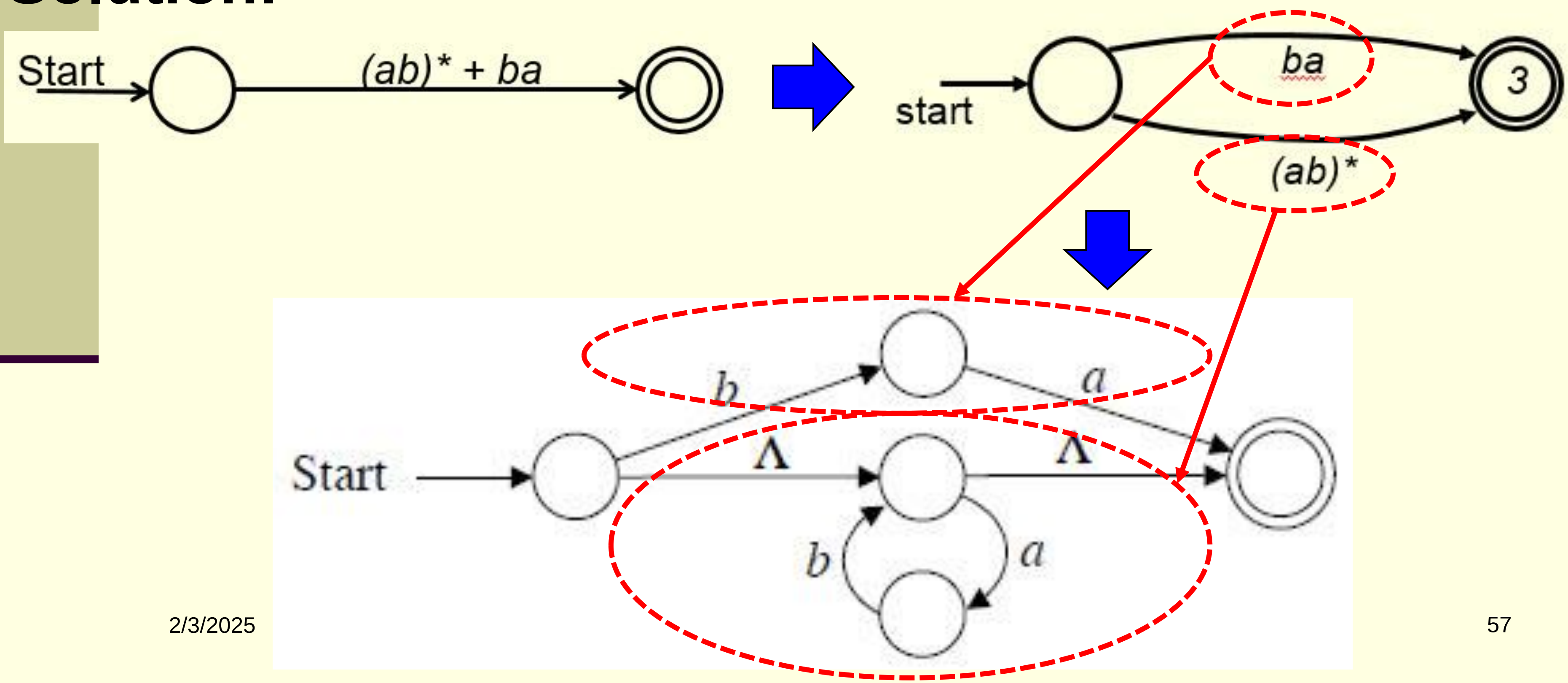
(2) Apply the following rules to obtain a finite automaton after erasing any ∅-edges.



Example. Use the algorithm to construct a finite automaton for

(ab) + ba.*

Solution:



6. Regular Languages & Finite Automata

- Finite Automata

Quiz. Use the algorithm to construct a finite automaton for

*(a+ba)*bb(a+ab)**

and compare your result with the NFA obtained on slide 51.

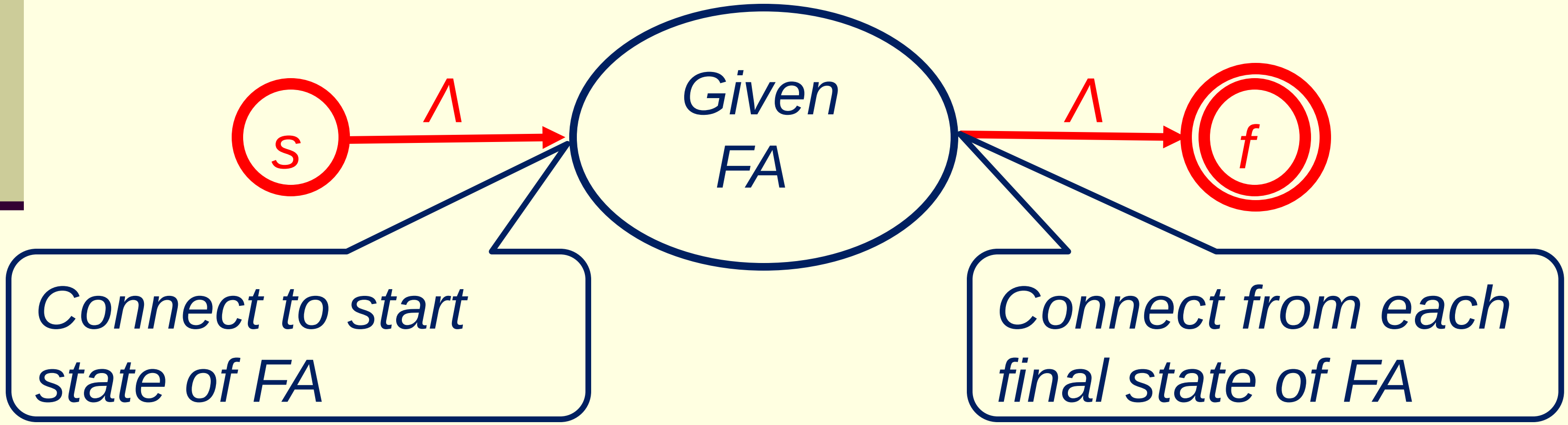
Do this at home.
We will discuss this question Thursday

6. Regular Languages & Finite Automata

- Finite Automata

Algorithm: Transform a Finite Automaton to a Regular Expression

Connect a new start state *s* to the start state of the FA and connect each final state of the FA to a new final state *f* as shown in the figure.

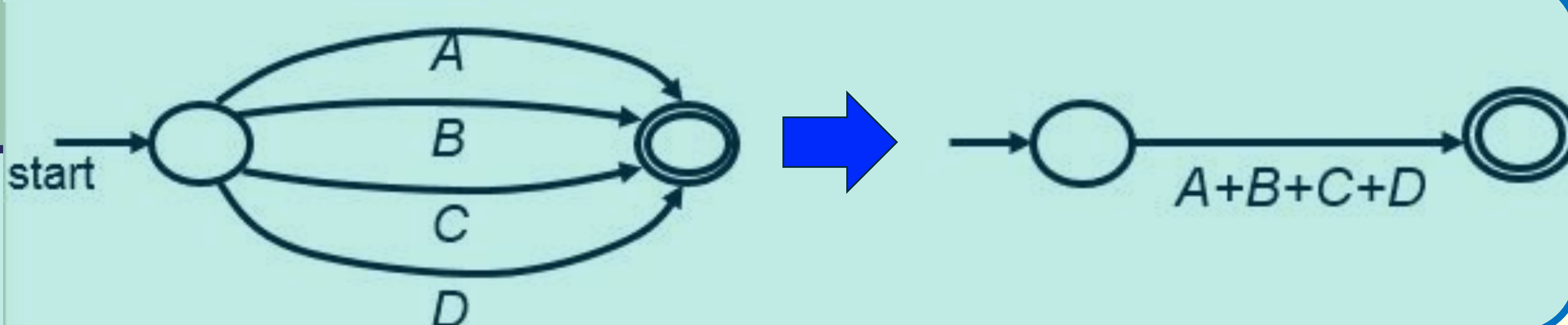


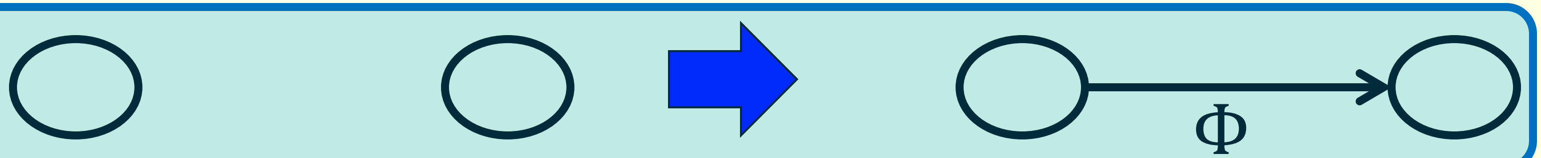
6. Regular Languages & Finite Automata

- Finite Automata

If needed, combine all multiple edges between the same two nodes into one edge with label the sum of the labels on the multiple edges.

If there is no edge between two states, assume there is an ∅-edge.





6. Regular Languages & Finite Automata

- Finite Automata

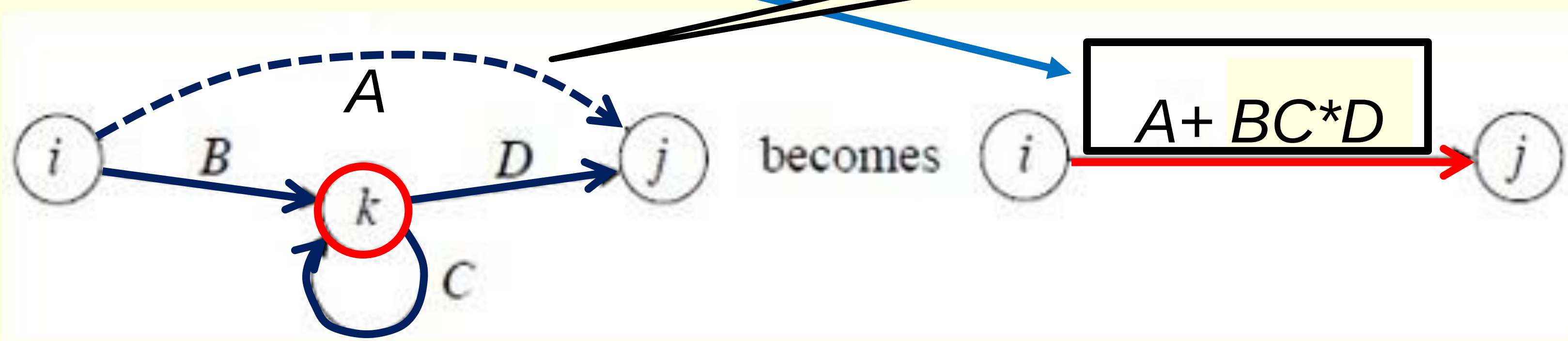
Now eliminate each state k of the FA by constructing a new edge (i, j) for **each pair** of edges (i, k) and (k, j) where $i \neq k$ and $j \neq k$.

New label $new(i, j)$ is defined as follows:

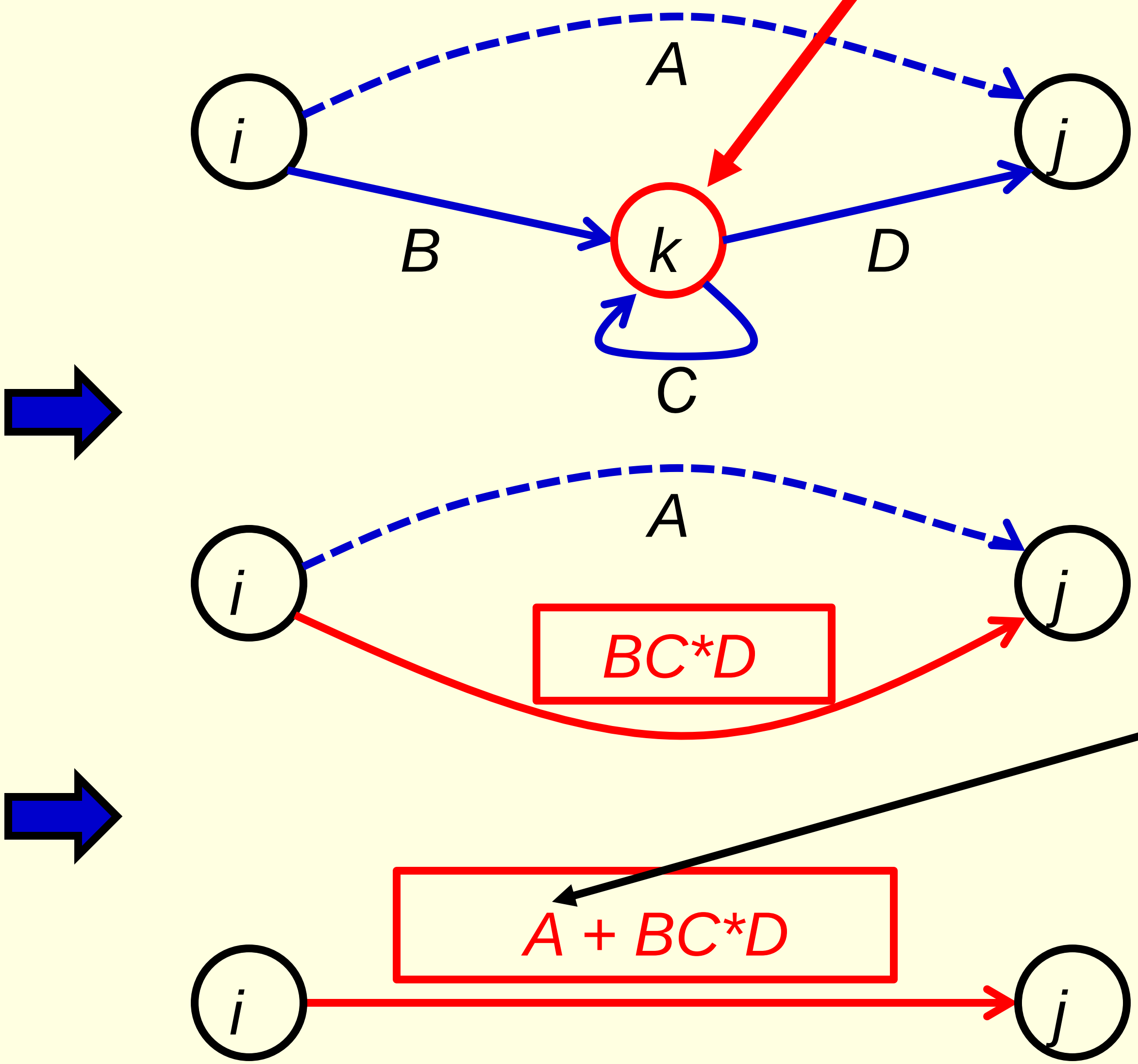
$$new(i, j) = old(i, j) + old(i, k)\ old(k, k)^*\ old(k, j)$$

Example:

Remove edge A as well



Think of the process of eliminating state k as a two-step procedure:

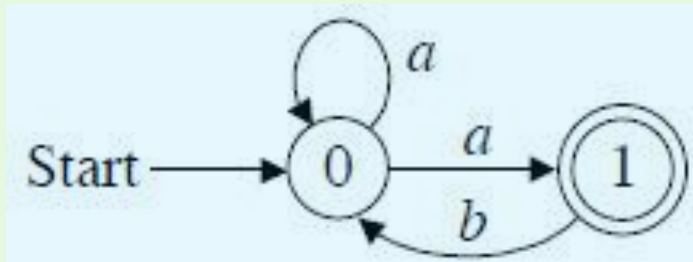


Put Φ here if there is not a direct edge between i and j originally

6. Regular Languages & Finite Automata

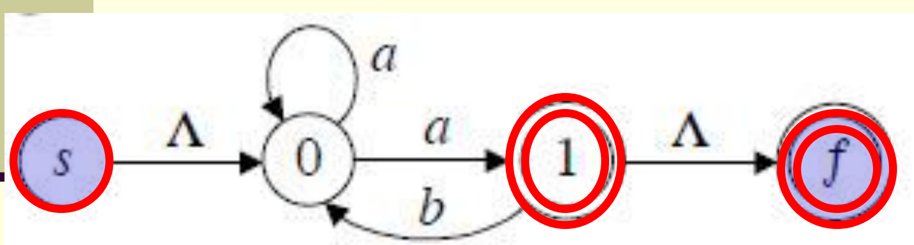
- Finite Automata

Example. Transform the following NFA to a regular expression.



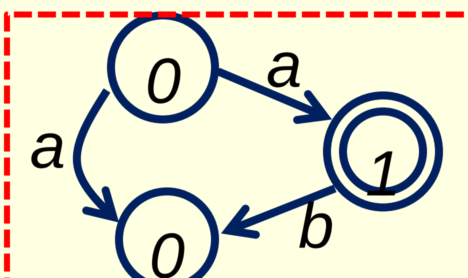
*a*a(ba*a)**

Solution I (eliminate state 1 first):



State 1 is a state between state 0 and state f

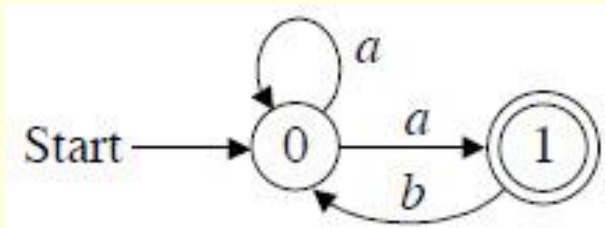
State 1 is also a state between state 0 and state 0



6. Regular Languages & Finite Automata

- Finite Automata

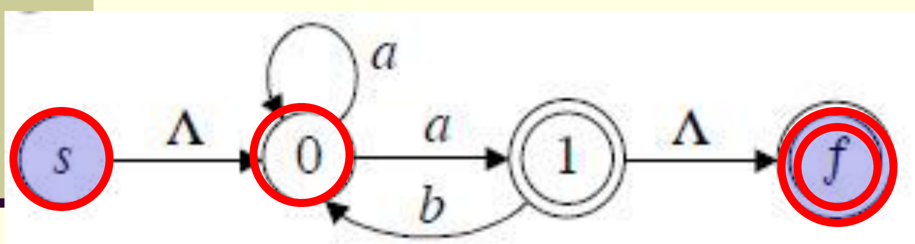
Example. Transform the following NFA into a regular expression.

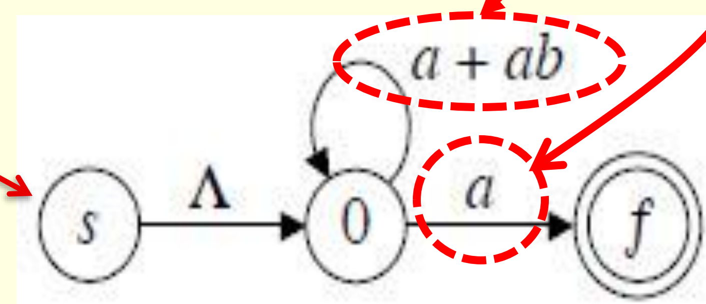


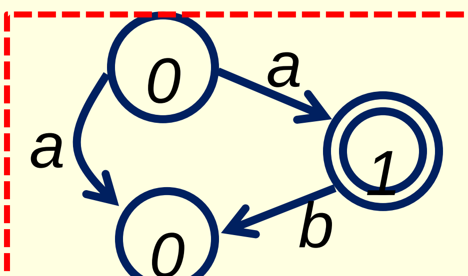
Solution I (eliminate state 1 first):

$$\text{new}(0, f) = \varnothing + a\varnothing^*\Lambda = a$$

$$\text{new}(0,0) = a + a\varnothing^* b = a + ab$$



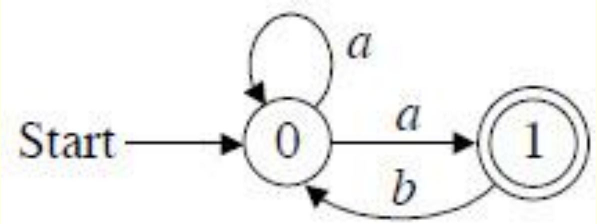




6. Regular Languages & Finite Automata

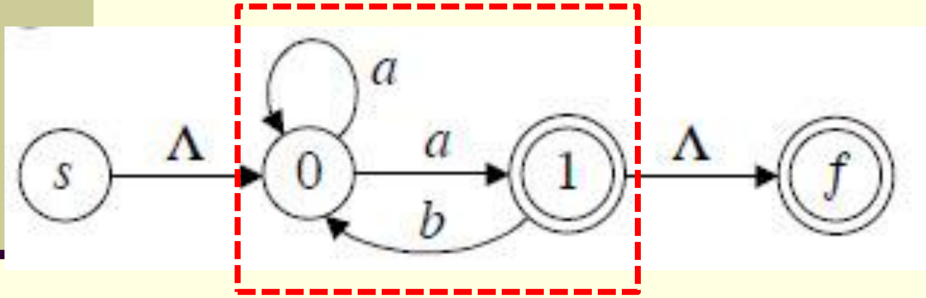
- Finite Automata

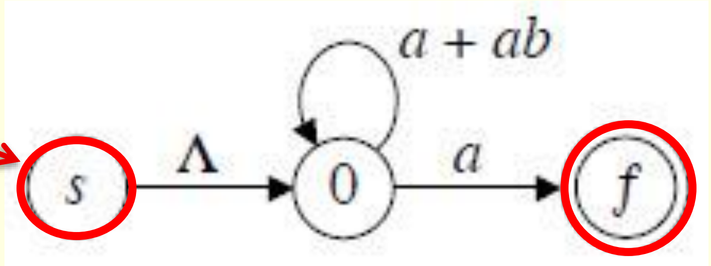
Example. Transform the following NFA into a regular expression.

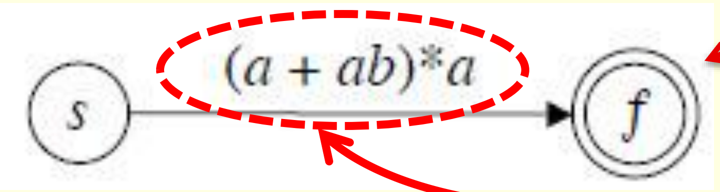


Solution I (eliminate state 1 first):

Then eliminate state 0

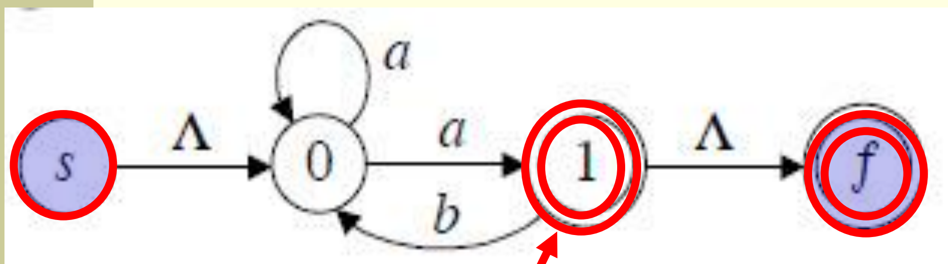






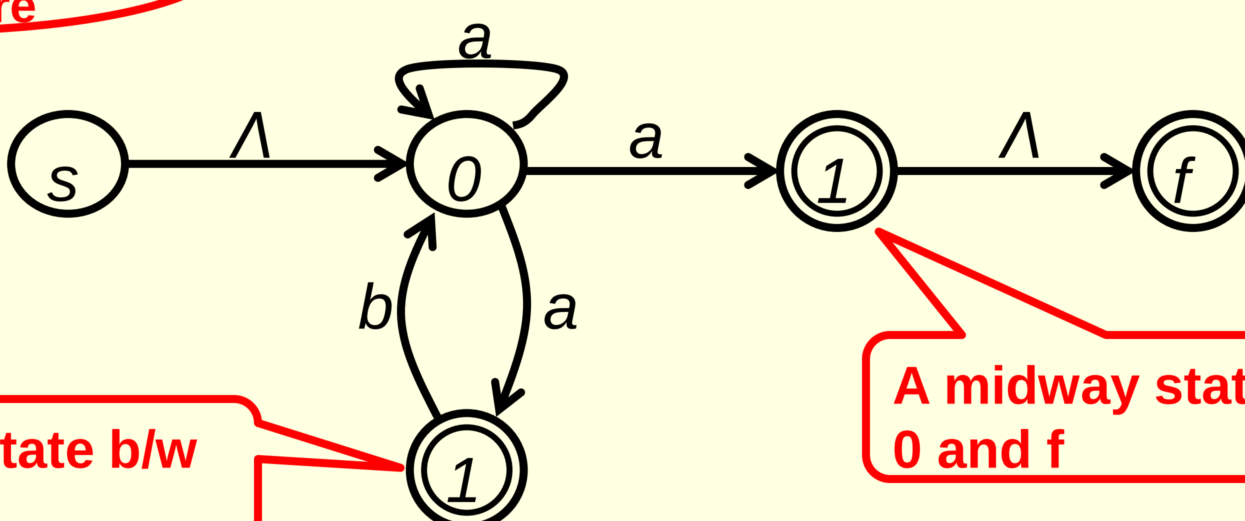
$$\text{new}(s, f) = \varnothing + \Lambda(a+ab)^* a = (a+ab)^* a$$

Or, eliminate state 1 first, the following way:



You can think of the given NFA as an NFA of the following form

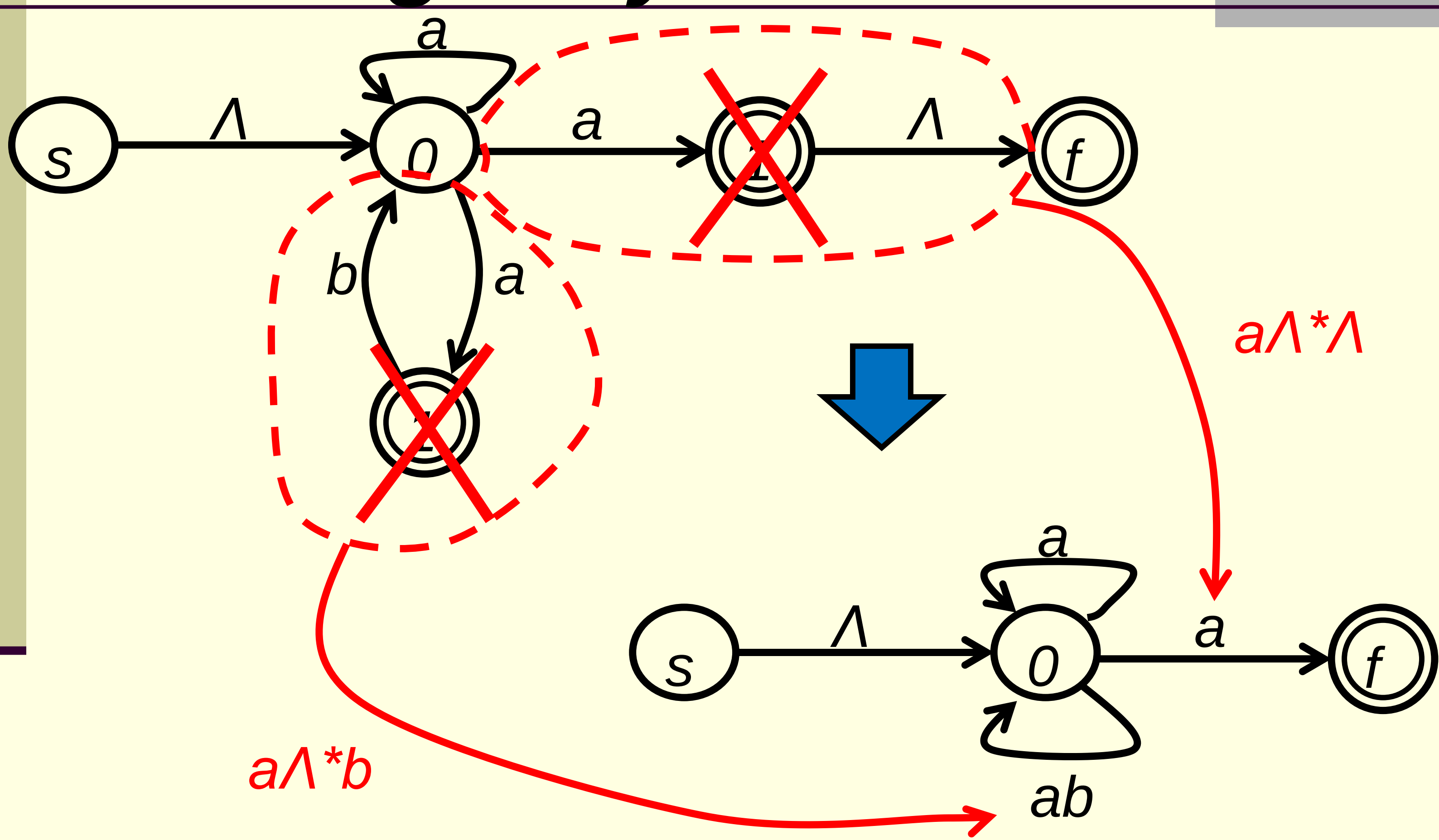
State 1 plays 2 roles here



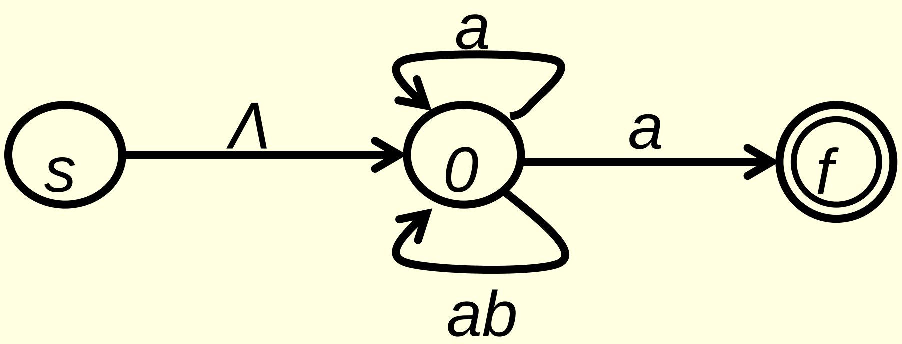
A midway state b/w 0 and f

A midway state b/w 0 and 0

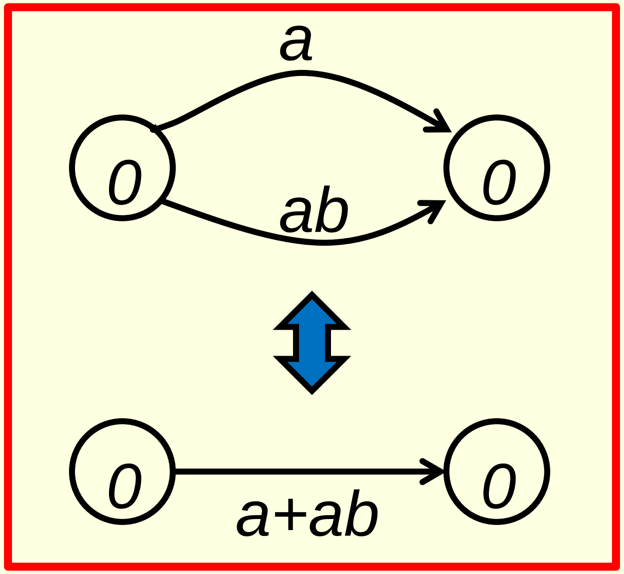
Or, eliminate state 1 first, the following way:



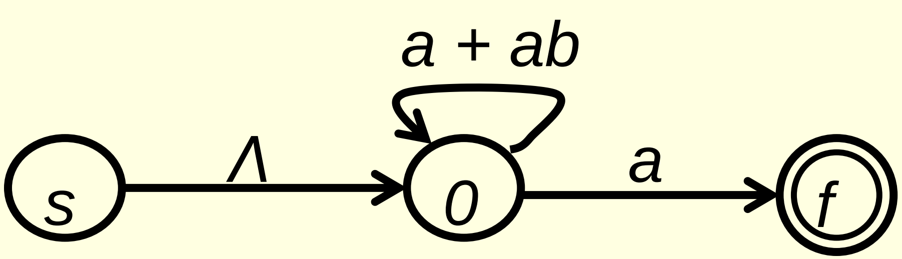
Or, eliminate state 1 first, the following way:



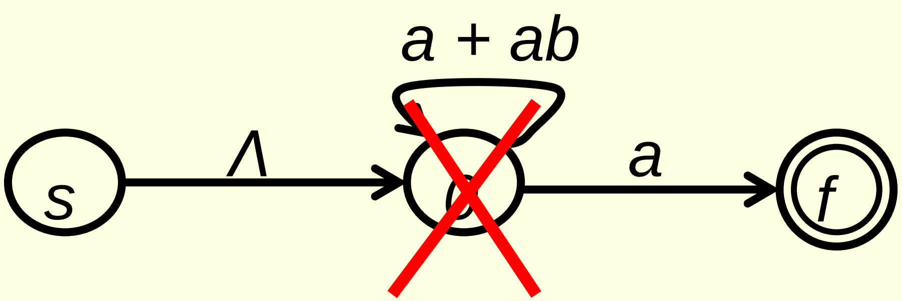
Why?



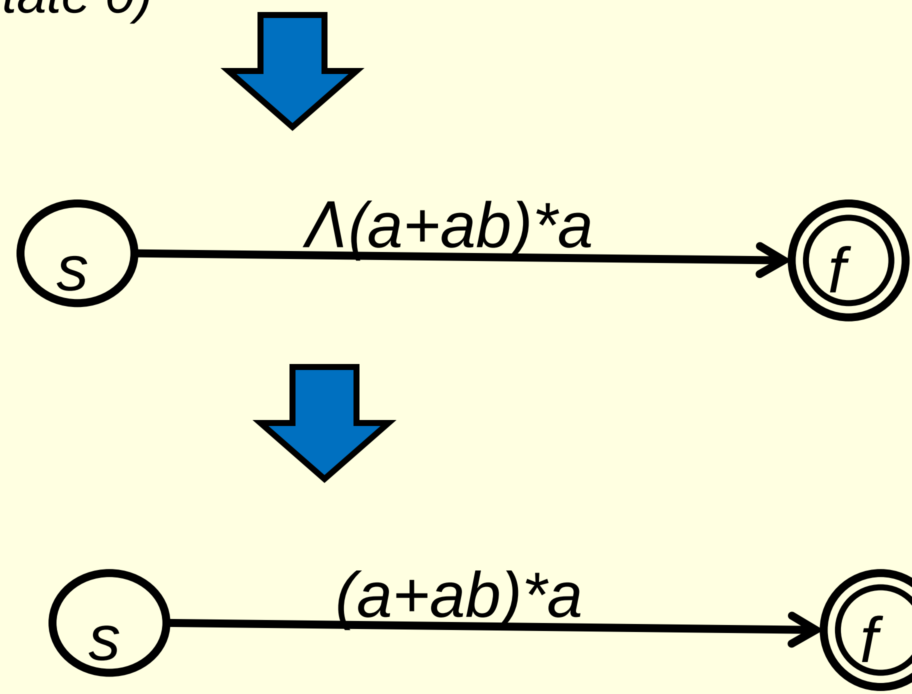
Then



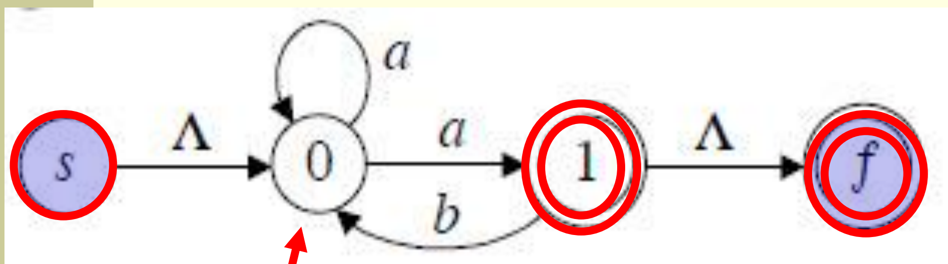
Or, eliminate state 1 first, the following way:



(Now eliminate state 0)



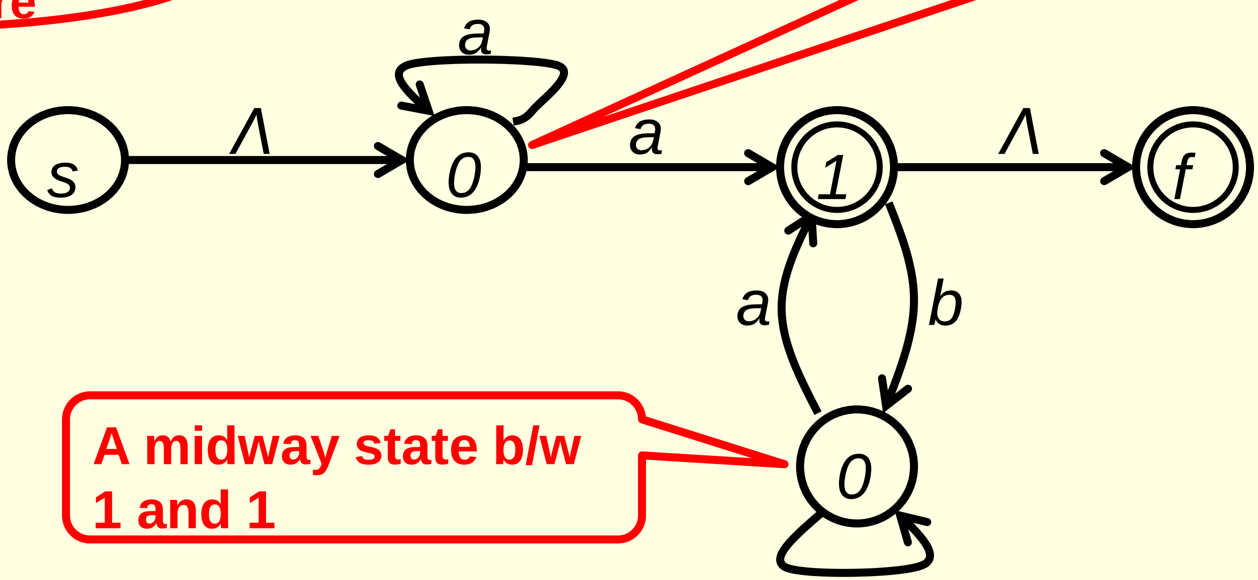
Question: *while we eliminate state 1 first, can we also do it the following way?*



State 0 plays 2 roles here

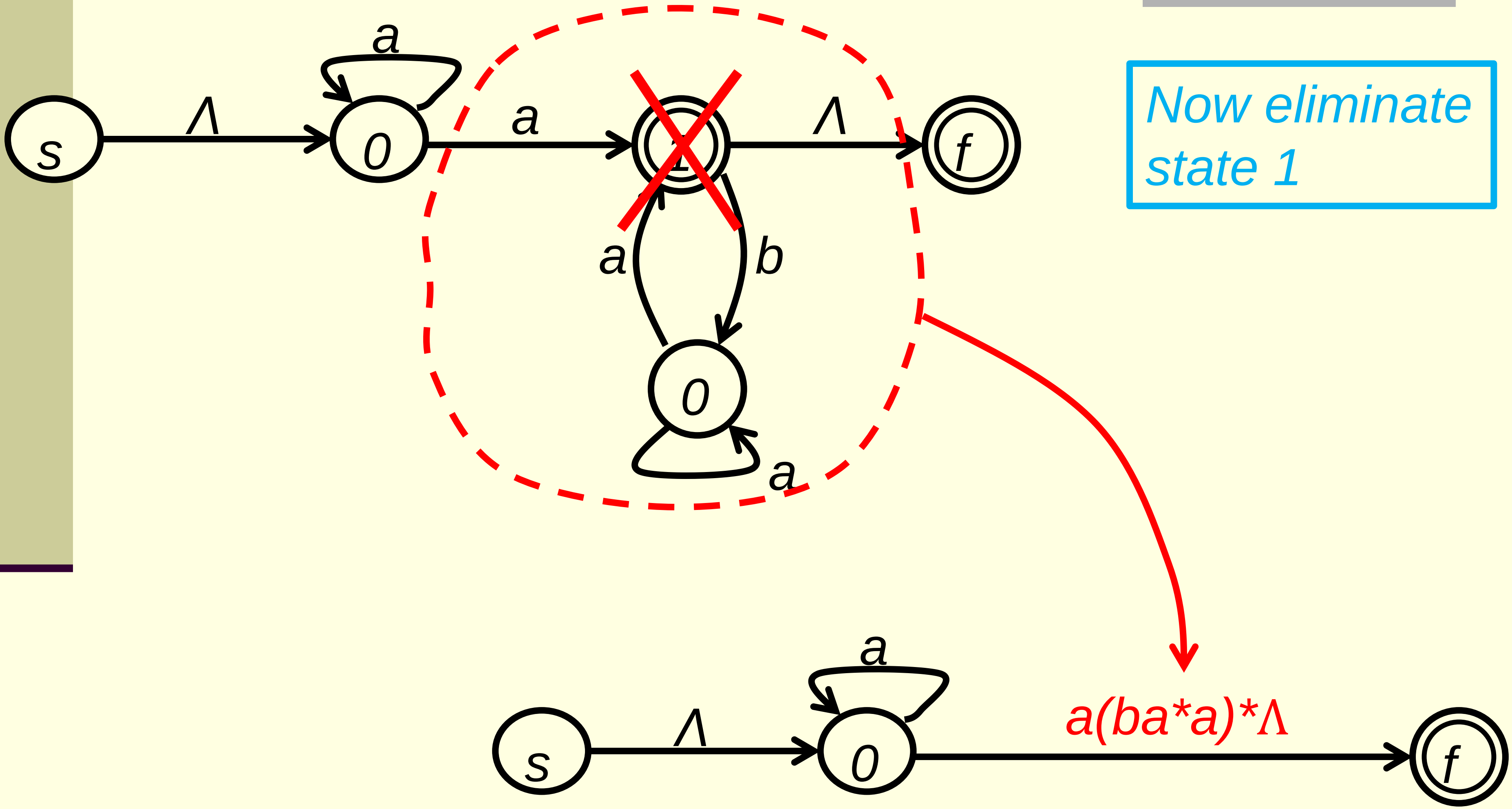
Think of the given NFA as an NFA of the following form

A midway state b/w s and 1

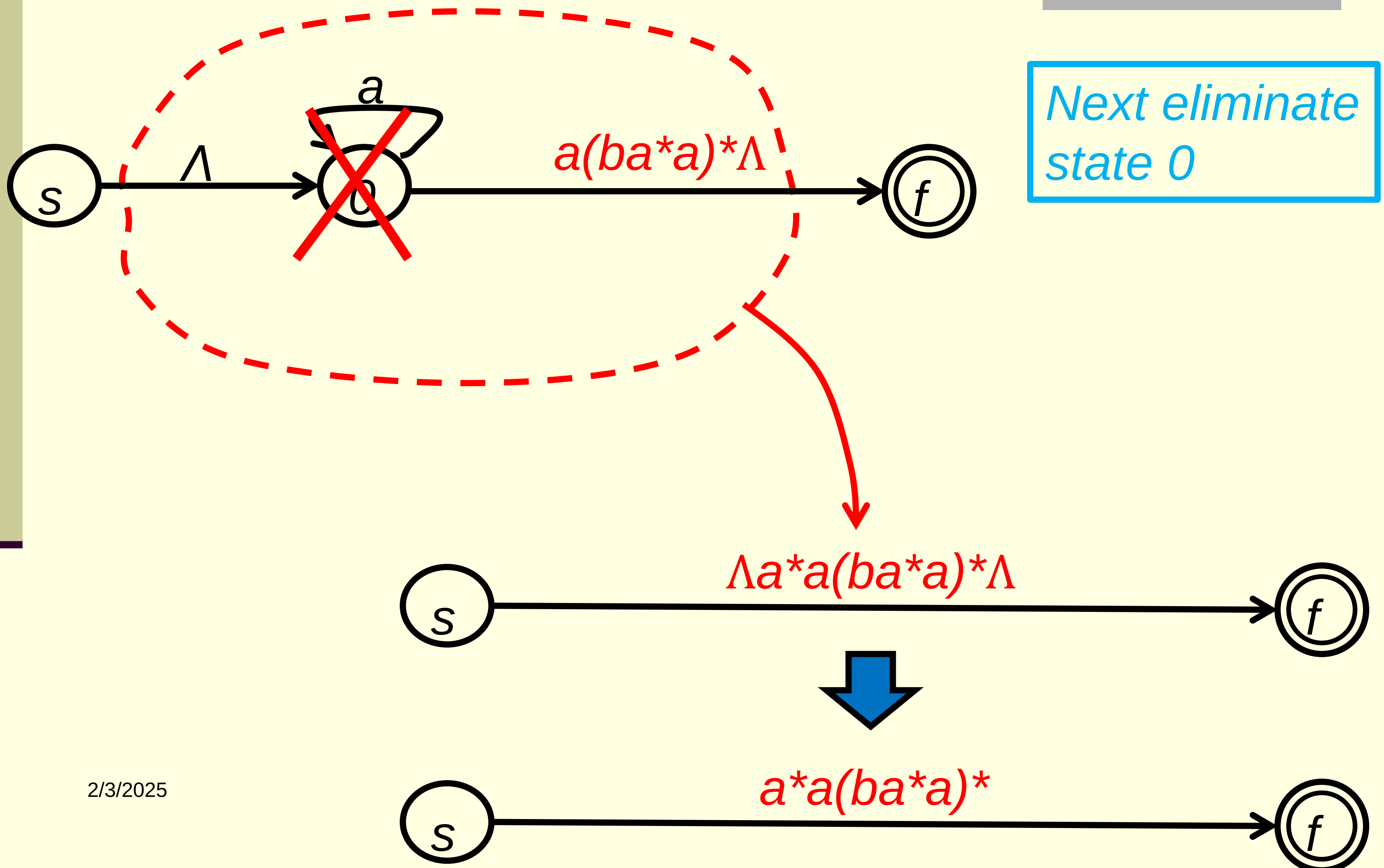


A midway state b/w 1 and 1

Question: *while we eliminate state 1 first, can we also do it the following way?*



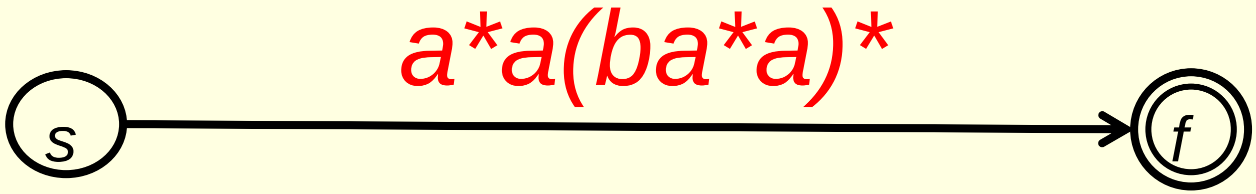
Question: *while we eliminate state 1 first, can we also do it the following way?*



Question: *while we eliminate state 1 first, can we also do it the following way?*

What is this?

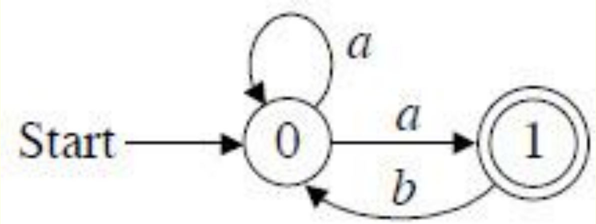
This is the regular expression we would get if we eliminate state 0 first



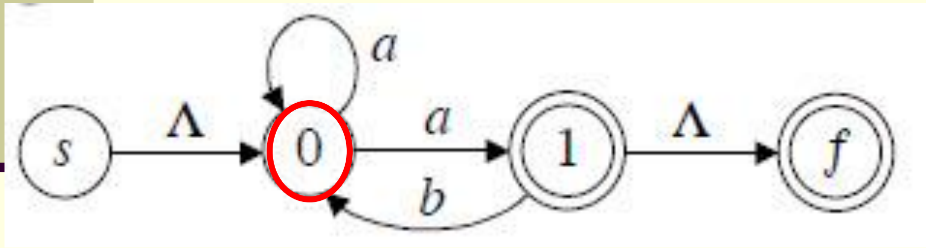
6. Regular Languages & Finite Automata

- Finite Automata

Example. Transform the following NFA into a regular expression.

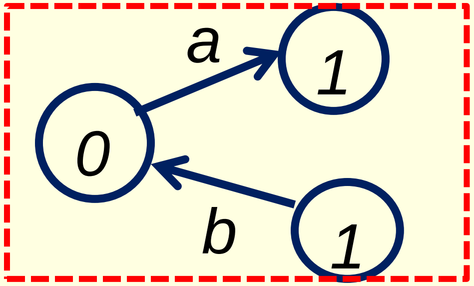


Solution II (eliminate state 0 first):



State 0 is a midway state b/w state s and state 1

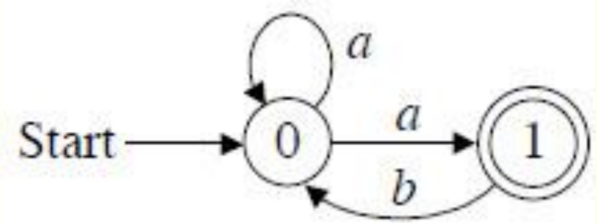
State 0 is also a midway state b/w state 1 and state 1



6. Regular Languages & Finite Automata

- Finite Automata

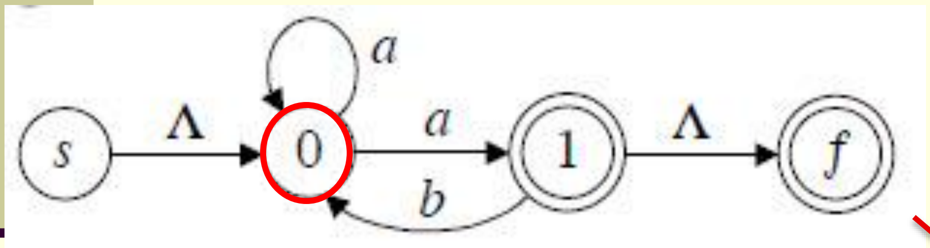
Example. Transform the following NFA into a regular expression.

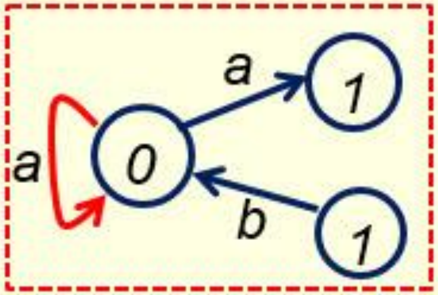


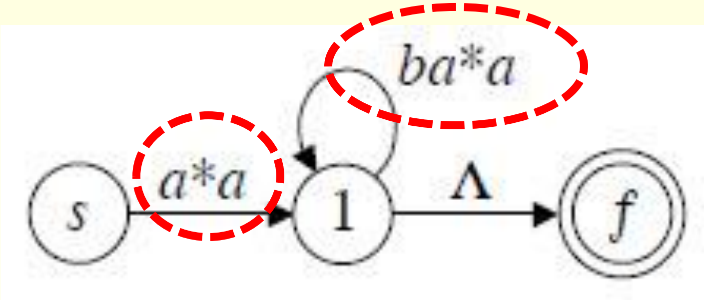
Solution II (eliminate state 0 first):

$$\text{new}(s,1) = \emptyset + \Lambda a^* a = a^* a$$

$$\text{new}(1,1) = \emptyset + b a^* a = b a^* a$$



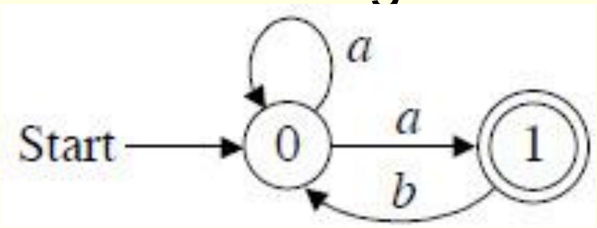




6. Regular Languages & Finite Automata

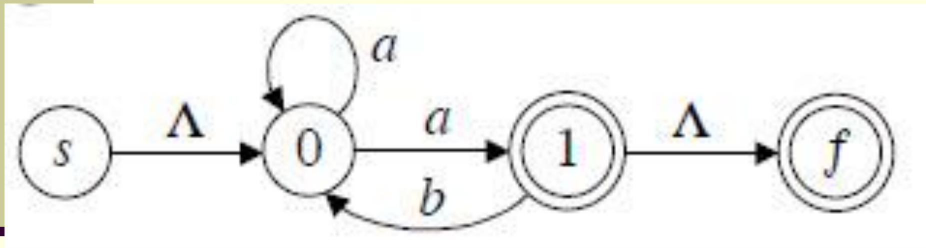
- Finite Automata

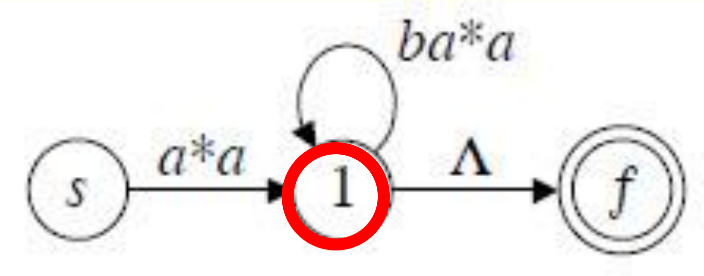
Example. Transform the following NFA into a regular expression.

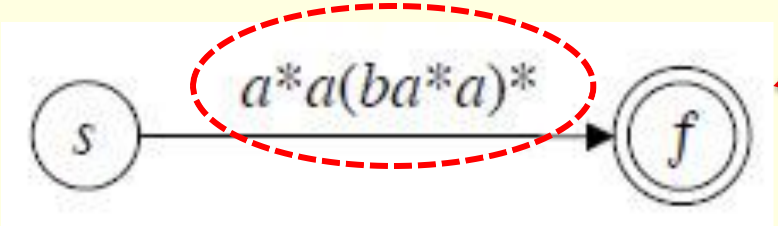


Solution II (eliminate state 0 first):

Then eliminate state 1

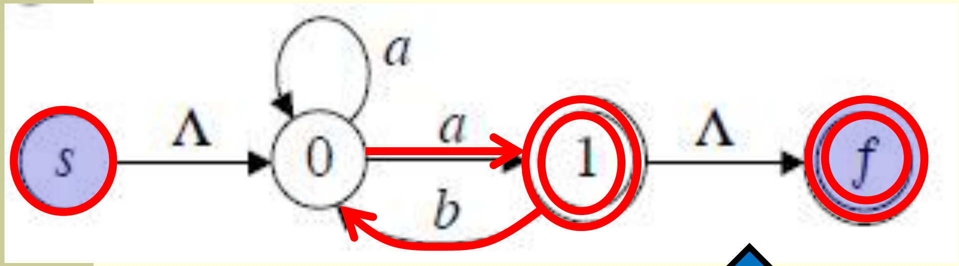




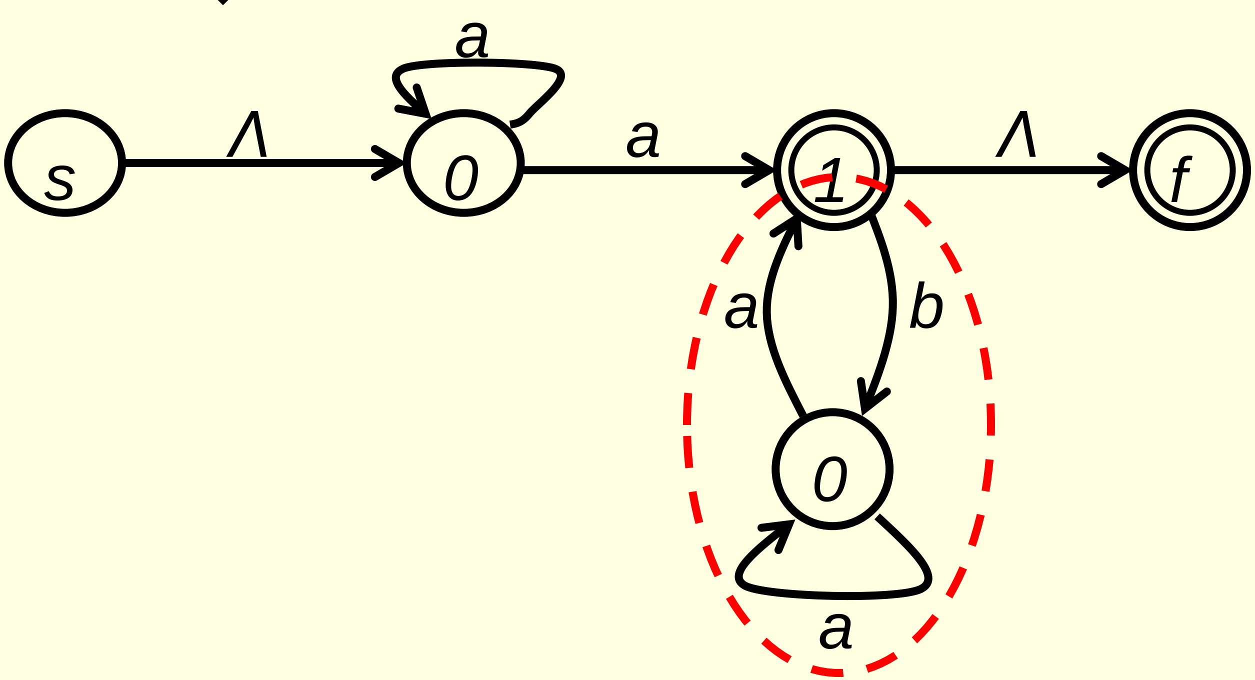


$$\text{new}(s, f) = \varnothing + a^* a (ba^* a)^* \Lambda = a^* a (ba^* a)^*$$

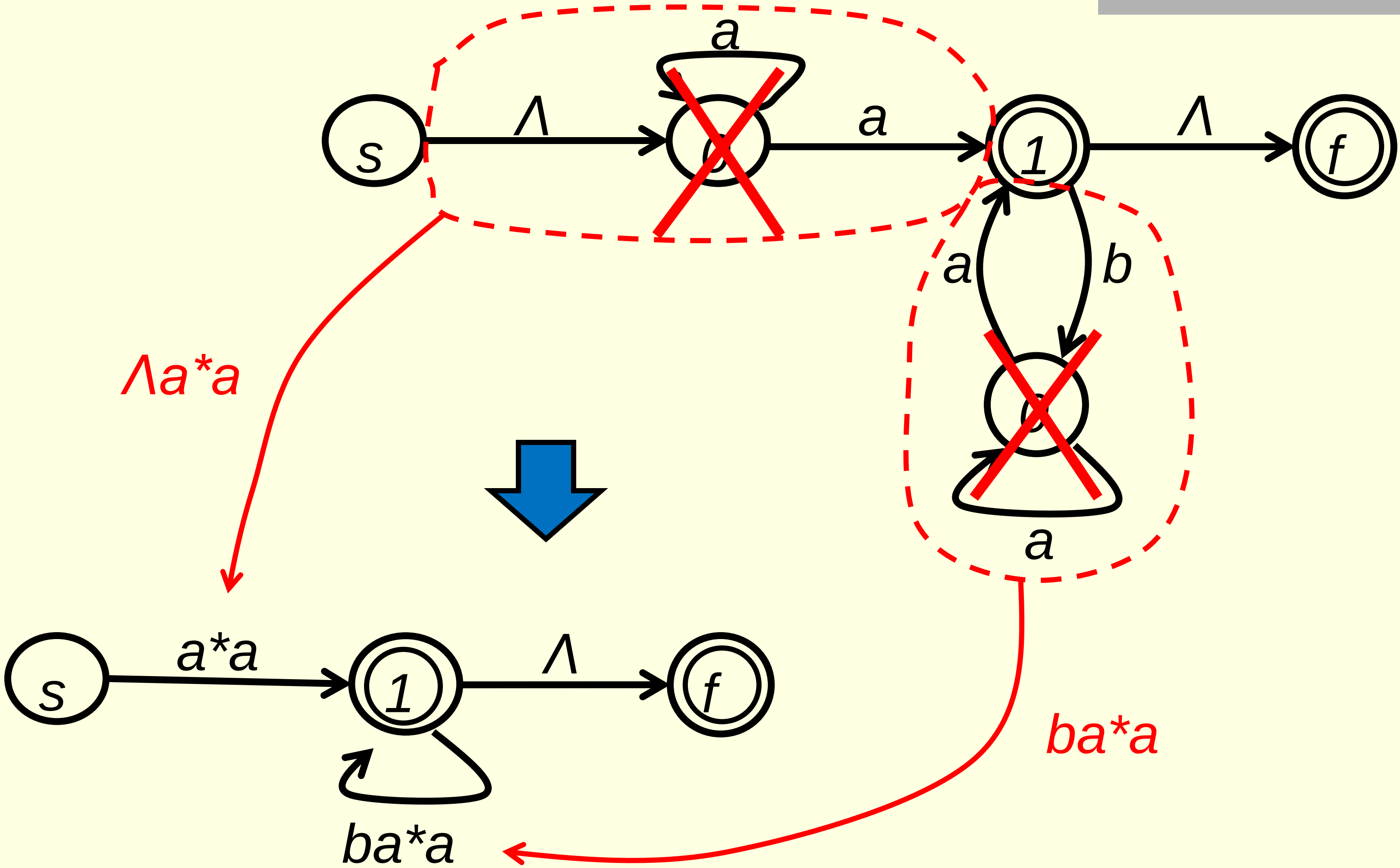
Or, eliminate state 0 first, the following way:



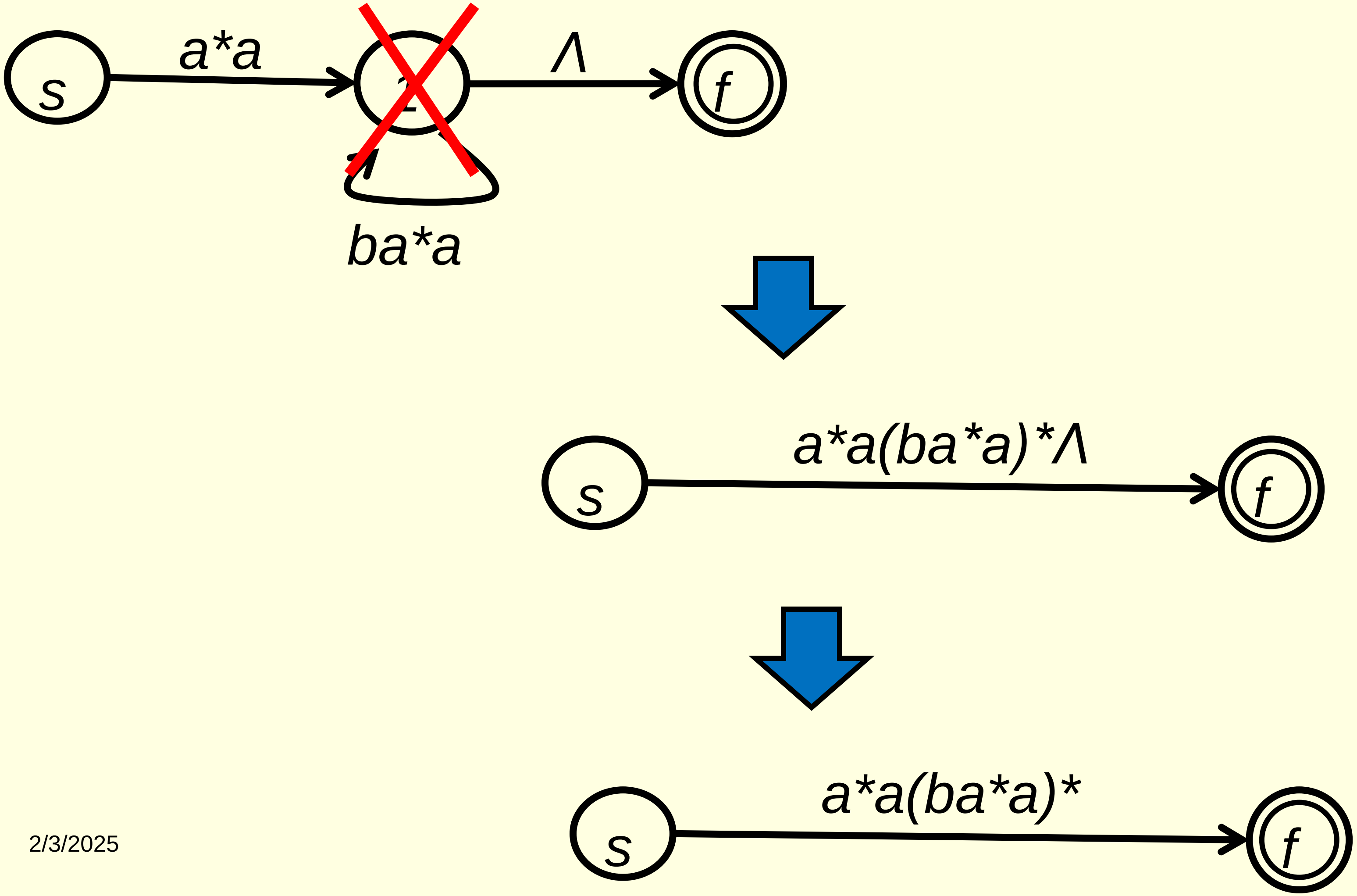
You can think of the given NFA as an NFA of the following form



Or, eliminate state 0 first, the following way:



Or, eliminate state 0 first, the following way:



6. Regular Languages & Finite Automata

- Finite Automata

Note. The two regular expressions obtained in the previous example are equal, i.e., $\boxed{a^*a(ba^*a)^* = (a+ab)^*a.}$

Proof I.

$$(a+ab)^*a \qquad \boxed{(R+S)^* = R^*(SR^*)^*}$$

$$= [\underline{a^*}\,\underline{((ab)a^*)^*}]\,\underline{a} \qquad \boxed{\cdot \text{ is associative}}$$

$$= \underline{a^*}\,[\,\underline{(\boxed{(ab)a^*})^*}\,\underline{a}\,] \qquad \boxed{\cdot \text{ is associative}}$$

$$= a^*\,[\,(\underline{\boxed{a}}\,\underline{\boxed{(ba^*)}})^*\,\underline{a}\,] \qquad \boxed{(RS)^*R = R(SR)^*}$$

$$= a^*\,[\,\underline{a}\,(\underline{(ba^*)}\,\underline{a})^*\,] \qquad \boxed{\cdot \text{ is associative}}$$

$$= a^*a(ba^*a)^*$$

6. Regular Languages & Finite Automata

- Finite Automata

Note. The two regular expressions obtained in the previous example are equal, i.e., $a^*a(ba^*a)^* = (a + ab)^*a.$

Proof II.

$a^*\,a\,(\,\underline{ba^*a}\,)^* = a^*[\underline{a}(\underline{(\,ba^*\,)}\,\underline{a})^*]$ · is associative

$= a^*[(\,\underline{a}\,\underline{(ba^*)}\,)^*\underline{a}\,]$ $\boxed{R(SR)^* = (RS)^*R}$

$= a^*[\,(\,\underline{(ab)}a^*\,)^*\,a]$ · is associative

$= \boxed{[\underline{a^*}\,(\underline{(ab)}\,\underline{a^*})^*\,]}\,a$ · is associative

$= (\underline{a} + \underline{ab})^*a$ $\boxed{R^*(SR^*)^* = (R + S)^*}$

QED.

6. Regular Languages & F

- Finite Automata

any combinations of
a^* and $(ab)^*$

Note. The two regular expressions obtained in the previous example are equal, i.e., $a^*a(ba^*a)^* = (a + ab)^*a$.

Intuitive Proof.

$LHS = a^*\, a\, (\, ba^*a\,)^* = a^*a(ba^*a)(ba^*a)(ba^*a) \cdots (ba^*a)$

$RHS = (a + ab)^*a = a^*(ab)^*a^*(ab)^* \cdots a^*(ab)^*a$

$LHS \subseteq RHS$ *Why?*

$a^*a(ba^*a)(ba^*a)(ba^*a) \;\epsilon\; LHS$

$= a^*(ab)a^*(ab)a^*(ab)a^*a$

$= a^*(ab)a^*(ab)a^*(ab)a^*(ab)^0a \;\epsilon\; RHS$

Hence, $LHS \subseteq RHS$

6. Regular Languages & Finite Automata

- Finite Automata

Note. The two regular expressions obtained in the previous example are equal, i.e., $a^*a(ba^*a)^* = (a + ab)^*a.$

Intuitive Proof (conti).

$LHS = a^*\,a\,(\,ba^*a\,)^* = \underline{a^*a(ba^*a)(ba^*a)(ba^*a) \cdots (ba^*a)}$

$RHS = (a + ab)^*a = a^*(ab)^*a^*(ab)^* \cdots a^*(ab)^*a$

$RHS \subseteq LHS$ *Why?*

$a^*(ab)^2a^*(ab)^2a^*(ab)^2a \;\epsilon\; RHS$

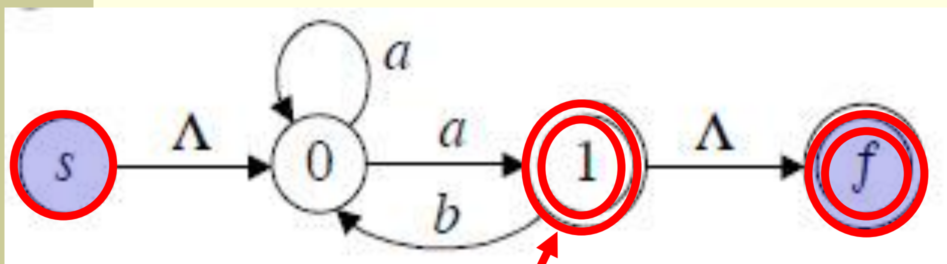
$= \underline{a^*}\,\underline{(ab)}\underline{(ab)}\underline{a^*(ab)}\underline{(ab)}\underline{a^*(ab)}\underline{(ab)a}$

$= a^*a(ba)(ba^*a)(ba)(ba^*a)(ba)(ba)$

$= \underline{a^*a(ba^0a)(ba^*a)(ba^0a)(ba^*a)(ba^0a)(ba^0a)} \;\epsilon\; LHS$

Hence, $RHS \subseteq LHS$

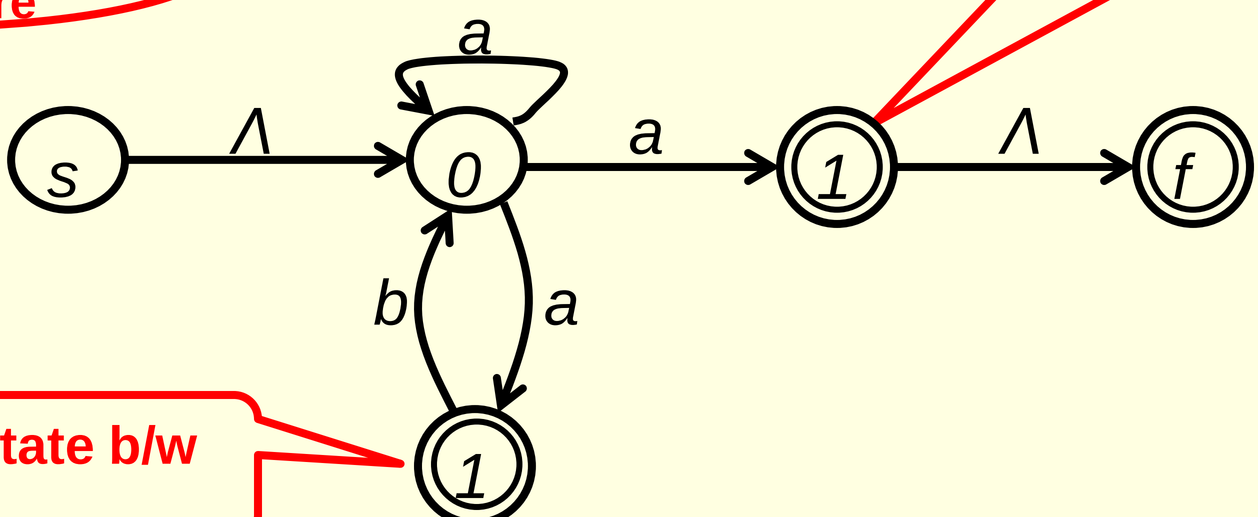
Question: *while we eliminate state 0 first, can we also do it the following way?*



Think of the given NFA as an NFA of the following form

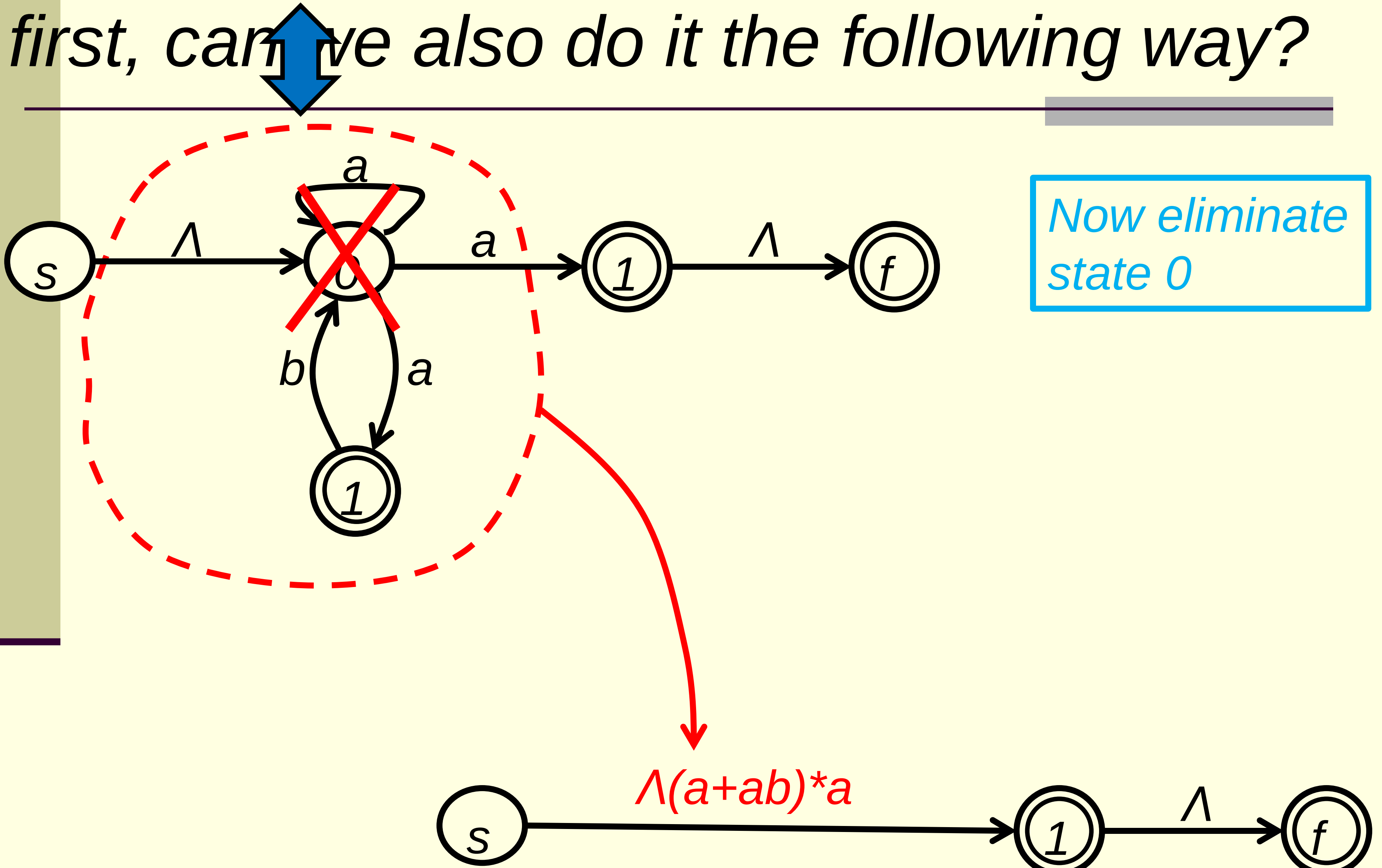
State 1 plays 2 roles here

A midway state b/w 0 and 1



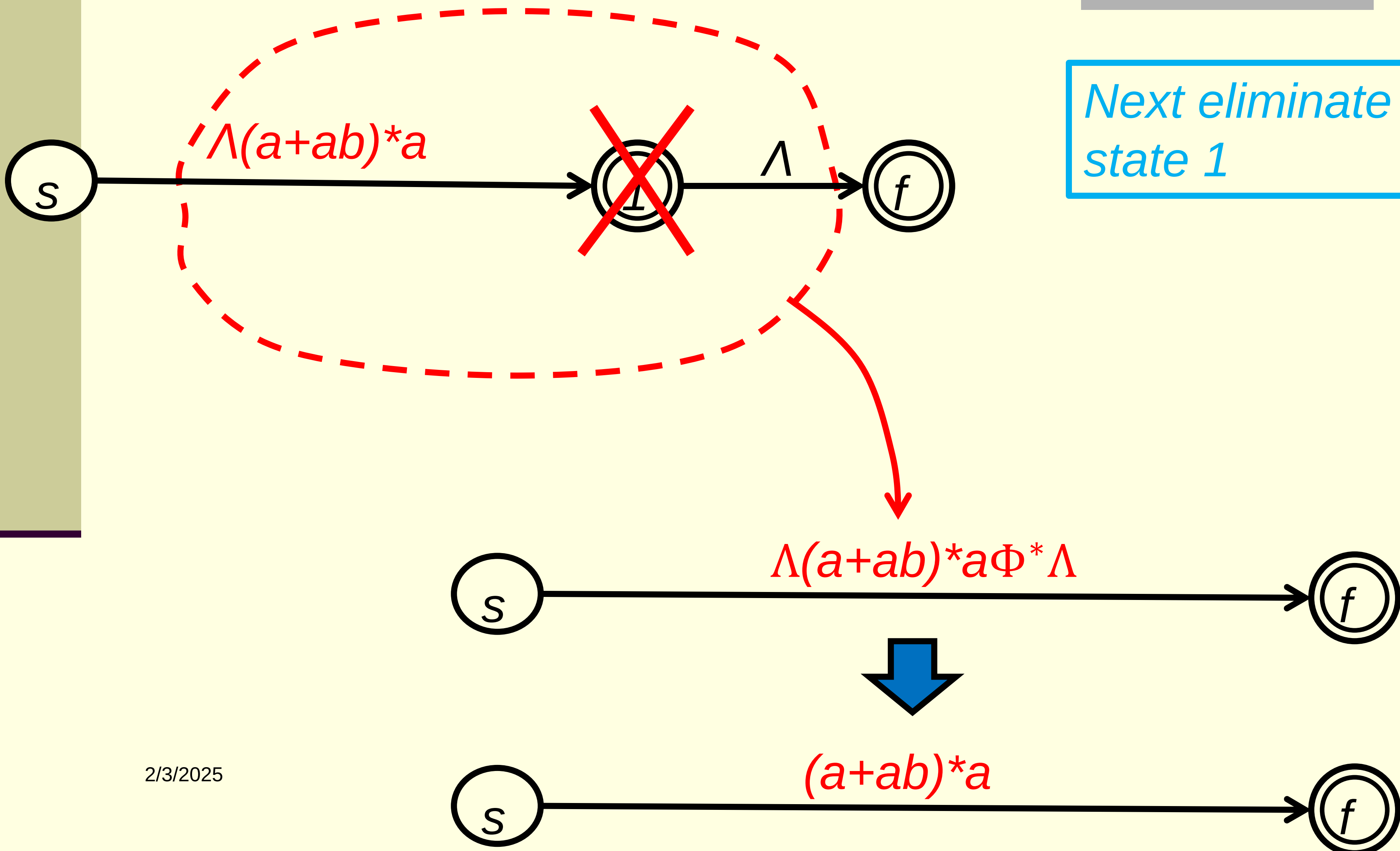
A midway state b/w 0 and 0

Question: *while we eliminate state 1 first, can we also do it the following way?*



Now eliminate state 0

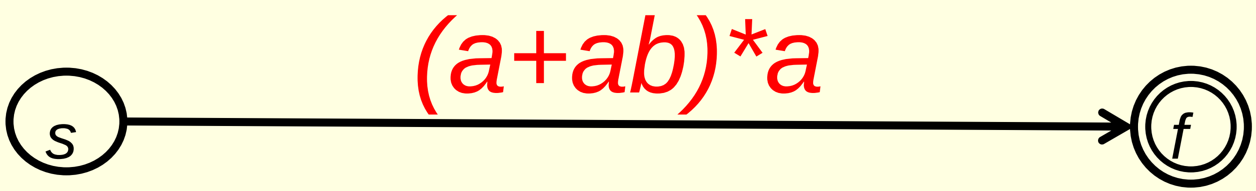
Question: *while we eliminate state 1 first, can we also do it the following way?*



Question: *while we eliminate state 1 first, can we also do it the following way?*

What is this?

This is the regular expression we would get if we eliminate state 1 first



End of Regular Languages and Finite Automata II

Closure Properties:

- $(R + S)^* = (R^* + S^*)^* = (R^*S^*)^* = (R^*S)^*R^* = R^*(SR^*)^*$.

Proof:
Obviously LHS ⊆ RHS.

On the other hand, if $w \in L\big((R^* + S^*)^*\big) = L(R^* + S^*)^*$ then there exists $n \in N$ such that $w \in L(R^* + S^*)^n$, or there exist $w_i \in L(R^* + S^*)$ for *i=1, 2, … n* such that $w = w_1 w_2 \cdots w_n$.

But $L(R^* + S^*) = L(R^*) \cup L(S^*)$ and

$L(R)^* = L(R)^0 \cup L(R)^1 \cup L(R)^2 \cup$ ...

$L(S)^* = L(S)^0 \cup L(S)^1 \cup L(S)^2 \cup$ ...

Hence, each $w_i \in L(R)^{i_j}$ or $L(S)^{i_k}$ for some i_j or i_k. But then each $w_i \in L(R + S)^*$. Consequently, w ∈ $L(R + S)^*$.

So, RHS ⊆ LHS.

Closure Properties:

- $(R + S)^* = \underline{(R^* + S^*)^* = (R^*S^*)^*} = (R^*S)^*R^* = R^*(SR^*)^*$.

Proof:
It is easy to see this by definition.

Note that $L\big((R^* + S^*)^*\big) = L(R^* + S^*)^* = (L(R^*) \cup L(S^*))^*$
$= ((L(R)^0 + L(R)^1 + L(R)^2 + \cdots) \cup (L(S)^0 + L(S)^1 + L(S)^2 + \cdots))^*$ **(1)**

On the other hand, $L\big((R^*S^*)^*\big) = L(R^*S^*)^* = (L(R^*)L(S^*))^* = (L(R)^*L(S)^*)^*$
$= ((L(R)^0 + L(R)^1 + L(R)^2 + \cdots)(L(S)^0 + L(S)^1 + L(S)^2 + \cdots))^*$ **(2)**

It is easy to see that Eq. (1) and Eq. (2) are equal ty definition.

So LHS = RHS.

Closure Properties:

- *(R + S)* = (R* + S*)* = (R*S*)* = (R*S)*R* = R*(SR*)*.*

Proof:

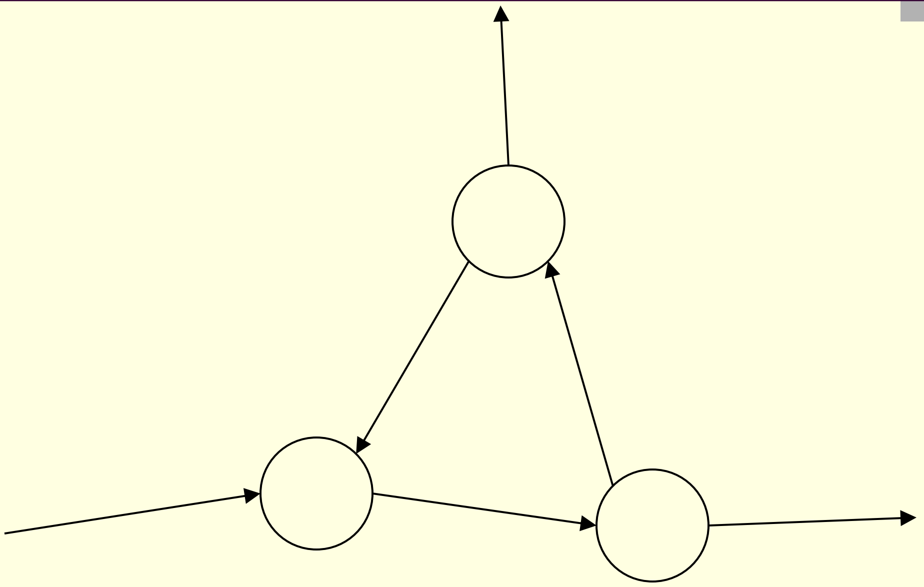
First note that $L\left((R^*S)^*\right) \subseteq L(R^*S^*)^*$.

On the other hand,

$(R^*S^*)^* = (R^*S^*)^0 + (R^*S^*)^1 + (R^*S^*)^2 + \cdots$ **(1)**

For each $(R^*S^*)^i$ in (1), if the all the exponents of S are not zero, it is covered by $(R^*S)^*$. However, if the exponents of S are all zero, it is not. But in such case, it is covered by *(R*S)*R** *(with the exponent of R*S being zero). Hence, we have* $L(R^*S^*)^* \subseteq L((R{*}S){*}R{*})$.

So LHS = RHS.



while (i=0; i<=n; i++) { }

i += i;

Closure Properties:

- $R(SR)^* = (RS)^*R$

- **Proof:**

LHS $= R((SR)^0+(SR)^1+(SR)^2+ \ldots + (SR)^n + \ldots)$

$= R(\Lambda+(SR)^1+(SR)^2+ \ldots + (SR)^n + \ldots)$

$= R + R(SR) + R(SR)^2+ \ldots + R(SR)^n + \ldots$

$= R + (RS)R + (RS)(RS)R + \ldots + \underbrace{(RS)(RS)\cdots(RS)}_{n\ times}R + \ldots$

$=(\Lambda + (RS) + (RS)^2 + \ldots + (RS)^n + \ldots)R$

$=(RS)^*R$

= RHS