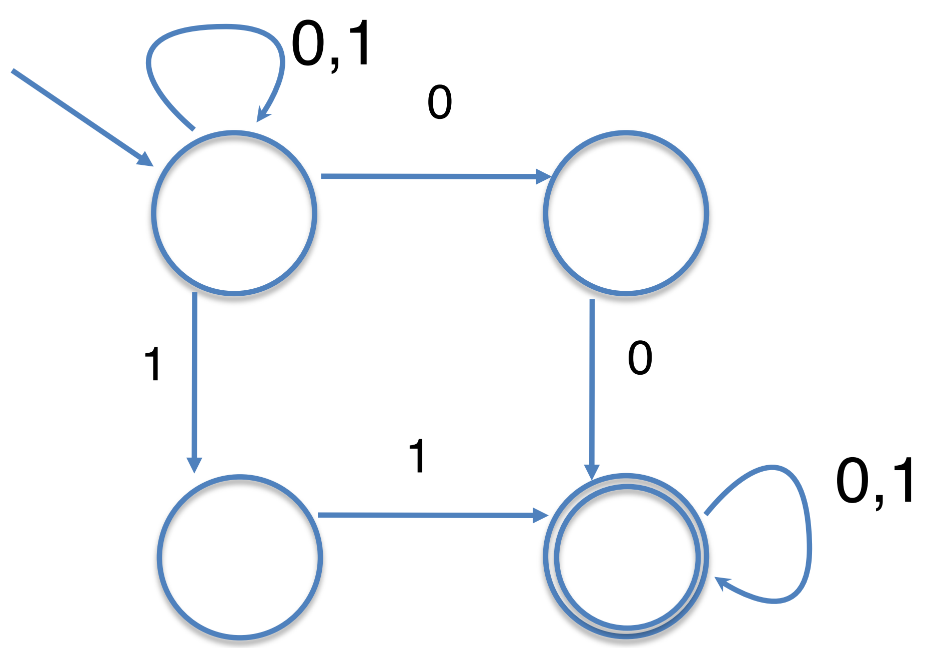
NFA/DFA, Relation to Regular Languages

Lecture 7

NFA recap

- Last lecture, we saw these objects called NFAs…



- Like DFA, but with a weird transition function: choices!
- DFA is a special case of NFA (how?)

NFA recap

- Last lecture, we saw these objects called NFAs…

3 models for (Regular) Languages:

Regular Expression

DFA

NFA

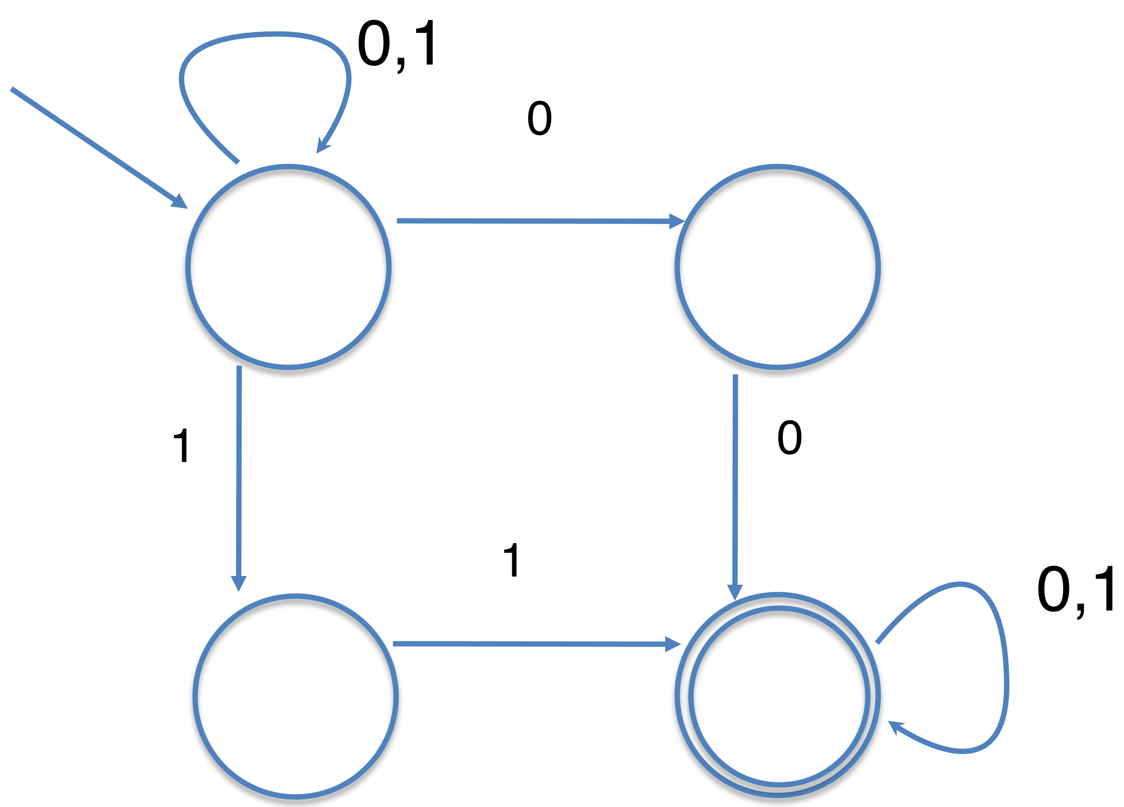
NFA recap

Kleene's Theorem

Regular Expression	=	DFA	=	NFA

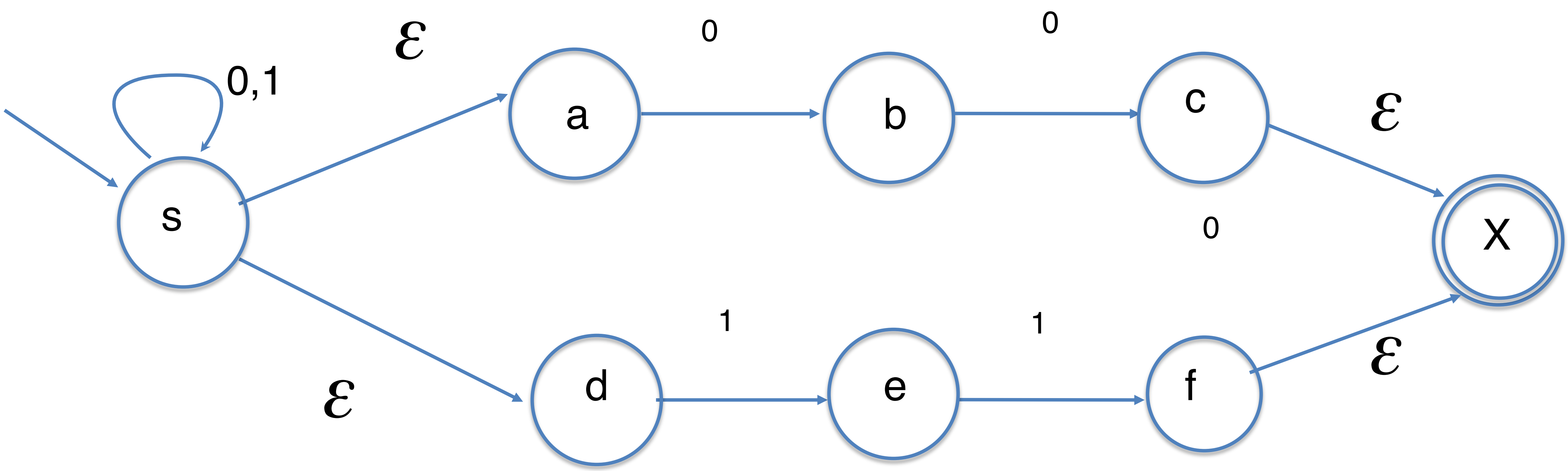
NFA+ε: Formally

- I want to be able to change my state without consuming input



NFA+ε: Formally

- I want to be able to change my state without consuming input



- On input 10001?

NFA+ε: Formally

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

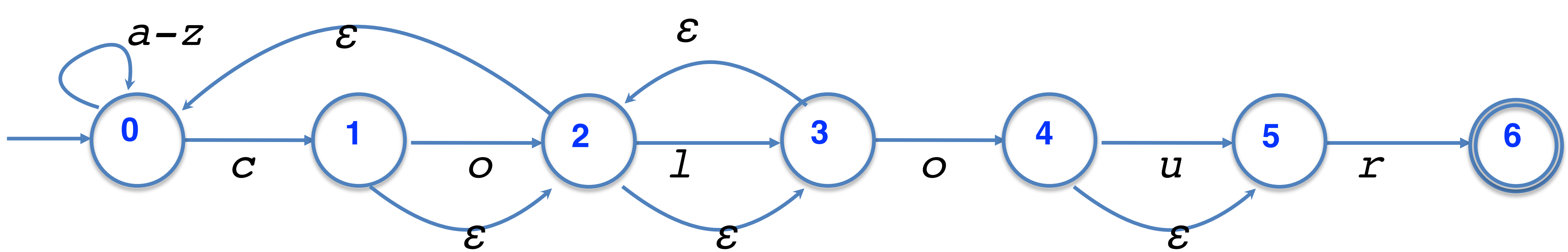
Σ: alphabet Q: state space s: start state A: set of accepting states

$$\delta : Q \times \{\Sigma \cup \varepsilon\} \rightarrow P(Q)$$

We say $q \overset{w}{\rightsquigarrow}_N p$

$$L(N) =$$

e.g., $\delta(\mathbf{1}, o) = \{\mathbf{2}\}$, $\delta(\mathbf{1}, x) = \emptyset$, $\delta(\mathbf{1}, \varepsilon) = \{\mathbf{2}\}$.



NFA+ε: Formally

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

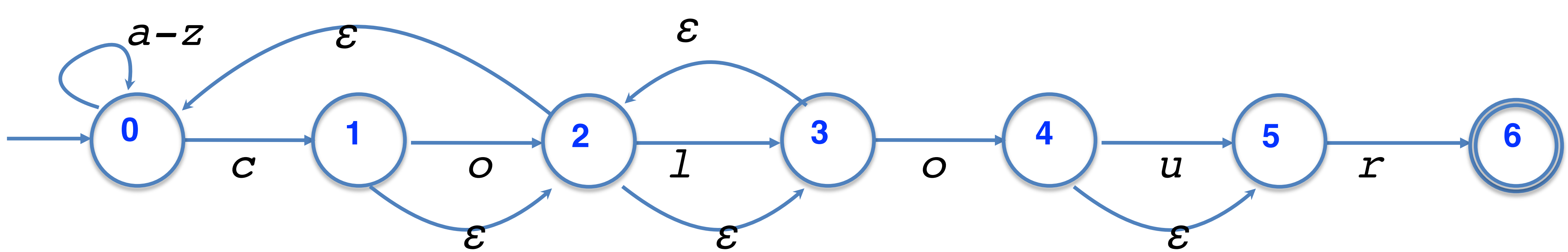
Σ: alphabet Q: state space s: start state A: set of accepting states

$$\delta : Q \times \{\Sigma \cup \varepsilon\} \rightarrow \mathrm{P}(Q)$$

We say $q \overset{w}{\rightsquigarrow}_N p$ if $\exists\, a_1,\ldots, a_t \in \Sigma \cup \{\varepsilon\}$ and $q_1,\ldots,q_{t+1} \in Q$, such that $w = a_1 \ldots a_t$, $q_1 = q$, $q_{t+1} = p$, and $\forall i \in [\,1, t\,]$, $q_{i+1} \in \delta(q_i, a_i)$

$$L(N) = \{\, w \mid s \overset{w}{\rightsquigarrow}_N p \text{ for some } p \in A \,\}$$

e.g., $\delta(\mathbf{1}, o) = \{\mathbf{2}\}$, $\delta(\mathbf{1}, x) = \emptyset$, $\delta(\mathbf{1}, \varepsilon) = \{\mathbf{2}\}$.

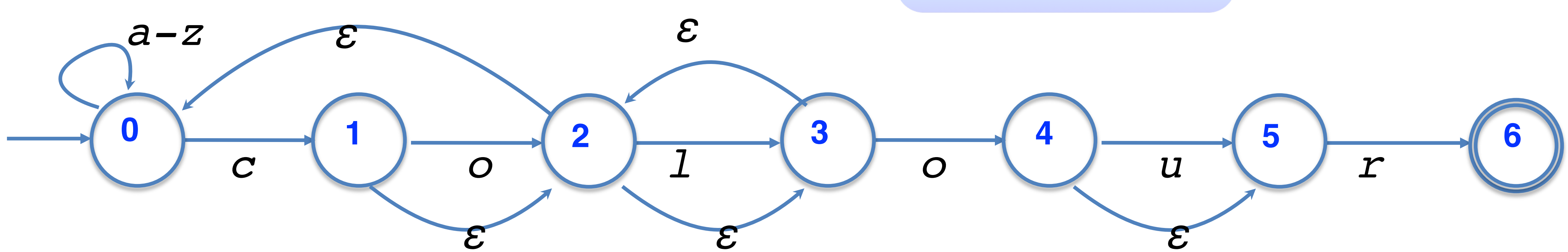


NFA+ε: Formally

We define the ε-reach of a state p:

e.g., $\delta(1,o) = \{2\}$, $\delta(1,x)=\emptyset$, $\delta(1,\varepsilon)=\{2\}$.

ε-reach({1}) = { 1, 2, 3, 0 }



NFA+ε: Formally

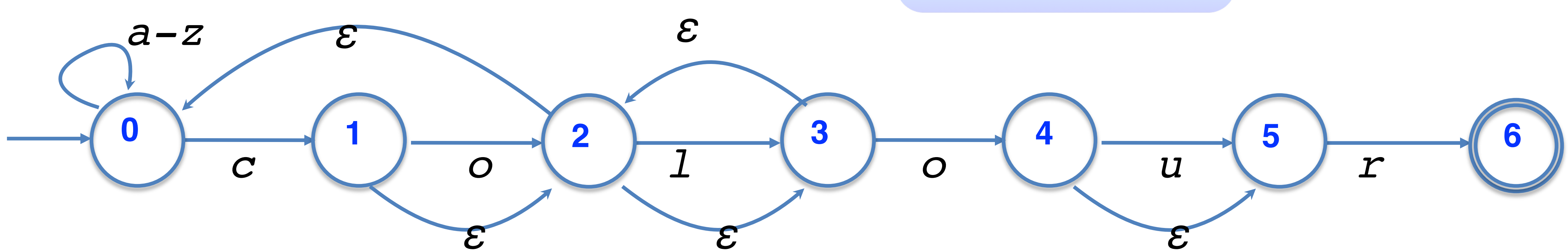
We define the ε-reach of a state p:

- p itself
- any state q such that $r \overset{\varepsilon}{\rightsquigarrow}_N q$ for some r in the ε-reach of p

Means that there is a sequence of ε-transitions from p to q

e.g., $\delta(\mathbf{1},o) = \{\mathbf{2}\}$, $\delta(\mathbf{1},x)=\varnothing$, $\delta(\mathbf{1},\varepsilon)=\{\mathbf{2}\}$.

ε-reach({**1**}) = { **1**, **2**, **3**, **0** }



Get rid of nothing

Can modify any NFA N, to get an NFA N_{new} without ε-moves

$$N_{new} = (\Sigma, Q_{new}, \delta_{new}, s_{new}, A_{new})$$

$$Q_{new} = Q$$

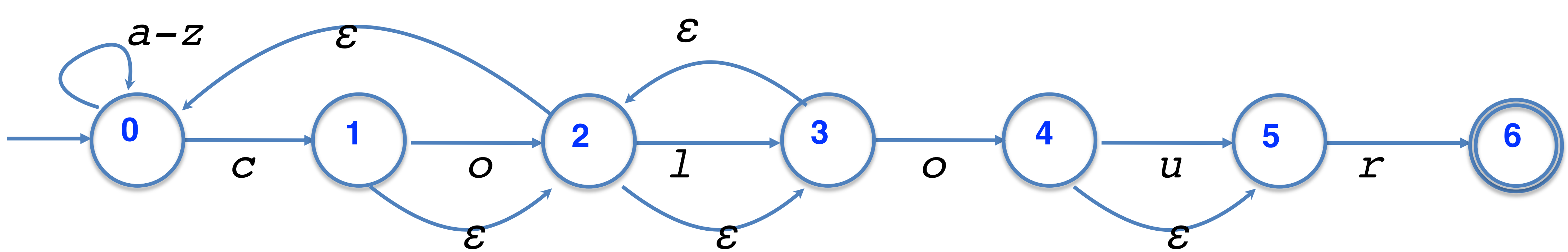
$$s_{new} = s$$

A_{new}={q| **ε-reach(q)** includes a state in A}

$$\{\, p \mid q \overset{a}{\rightsquigarrow}_N p \,\}$$

$$\delta_{new}(q, a) = \cup_{p \in \varepsilon-\text{reach}(q)} \delta(p, a)$$

e.g.: $\delta_{new}(1, o) =$? $\{0,2,3,4,5\}$



Get rid of nothing

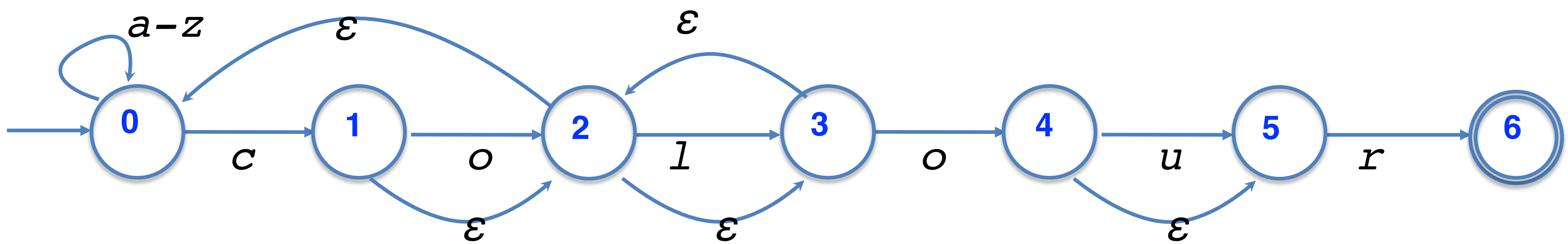
Can modify any NFA N, to get an NFA N_{new} without ε-moves

$$N_{new} = (\Sigma, Q_{new}, \delta_{new}, s_{new}, A_{new})$$

$$Q_{new} = Q$$

$$s_{new} = s$$

$$A_{new} = \{ \qquad \}$$



Get rid of nothing

Can modify any NFA N, to get an NFA N_{new} without ε-moves

$$N_{\text{new}} = (\Sigma, Q_{\text{new}}, \delta_{\text{new}}, s_{\text{new}}, A_{\text{new}})$$

$$Q_{\text{new}} = Q$$

$$s_{\text{new}} = s$$

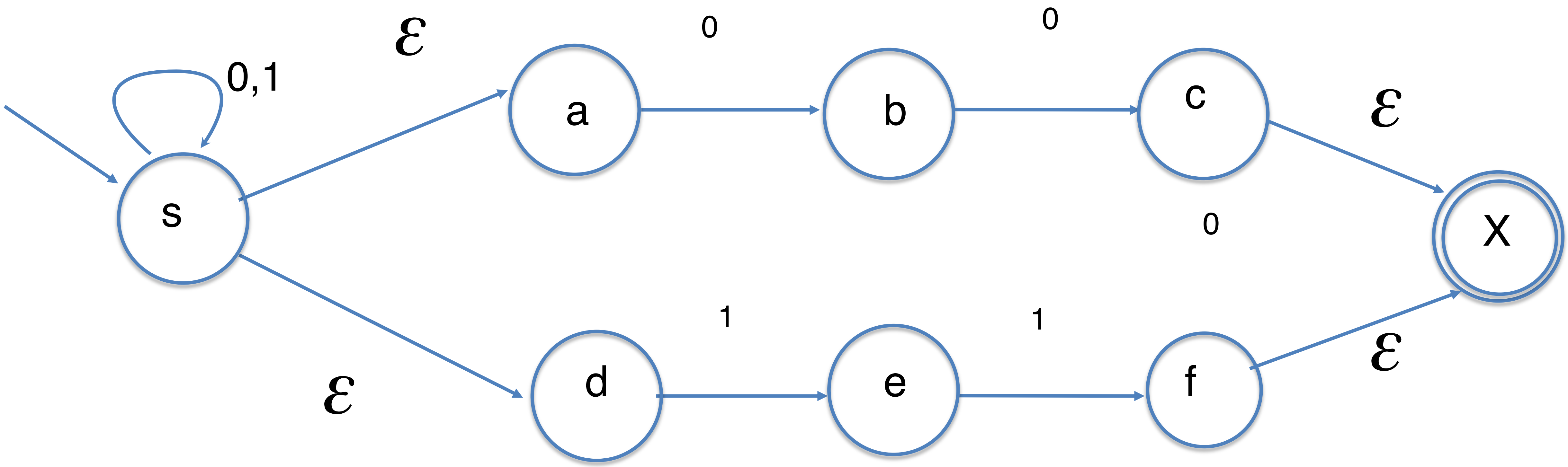
A_{new}={q| **ε-reach(q)** includes a state in A}

$\{ p \mid q \overset{a}{\rightsquigarrow}_N p \}$

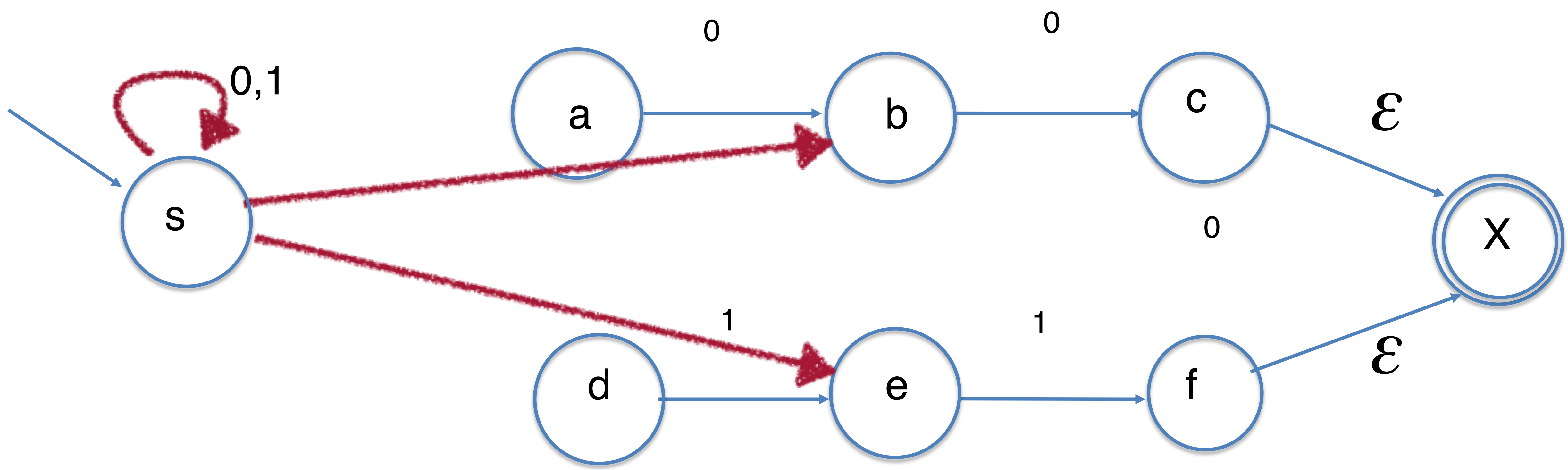
$$\delta_{\text{new}}(q, a) = \cup_{p \in \varepsilon-\text{reach}(q)} \delta(p, a)$$

Theorem: $L(N) = L(N_{\text{new}})$

NFA+ε: Formally

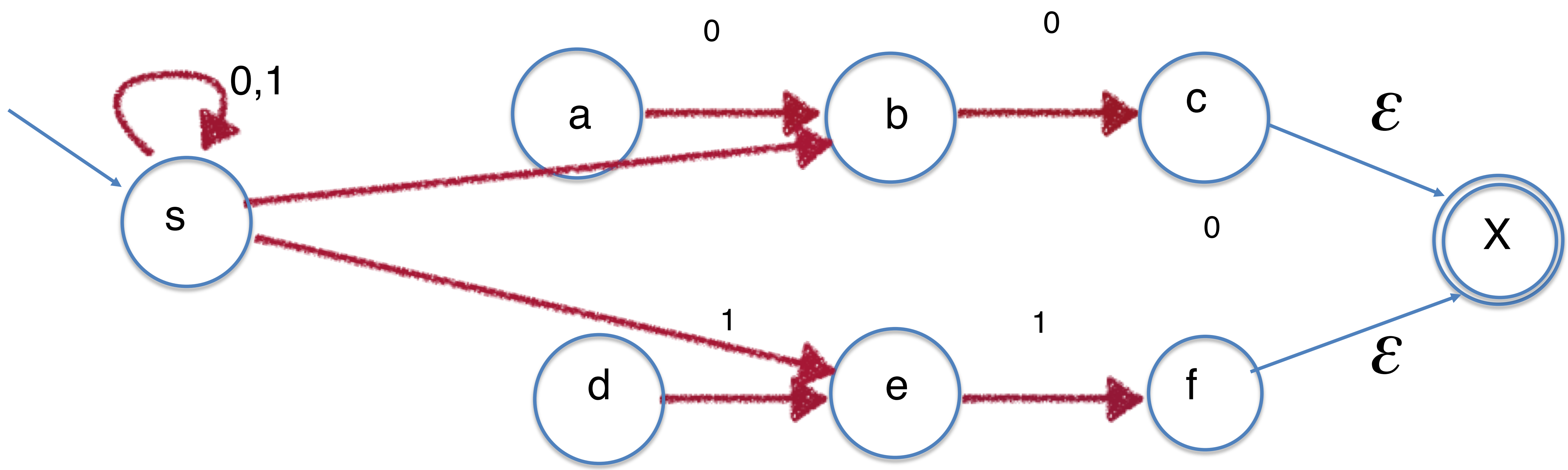


NFA-ε



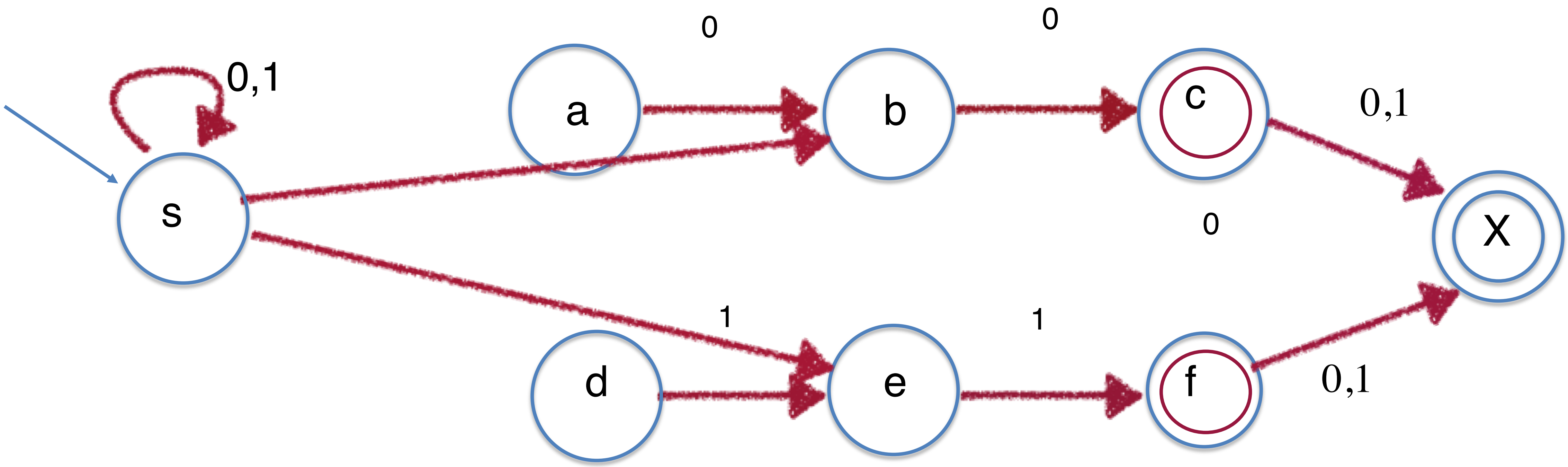
$$\delta_{\text{new}}(q,a) = \cup_{p \in \varepsilon-\text{reach}(q)} \delta(p,a)$$

NFA-ε



$$\delta_{\text{new}}(q,a) = \cup_{p \in \varepsilon-\text{reach}(q)} \delta(p,a)$$

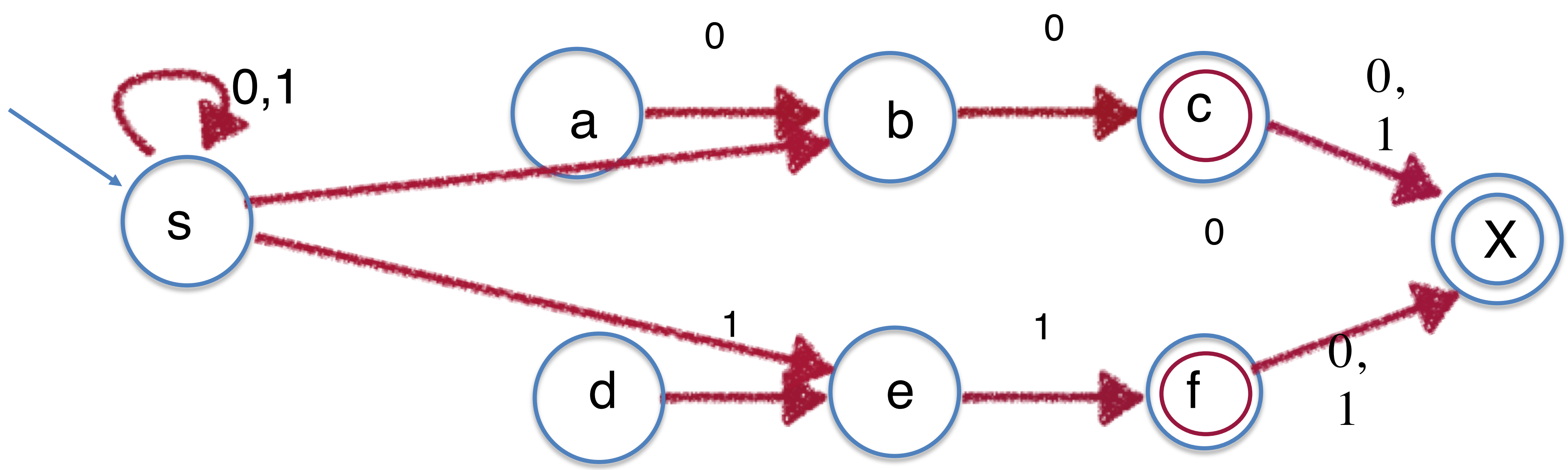
NFA-ε

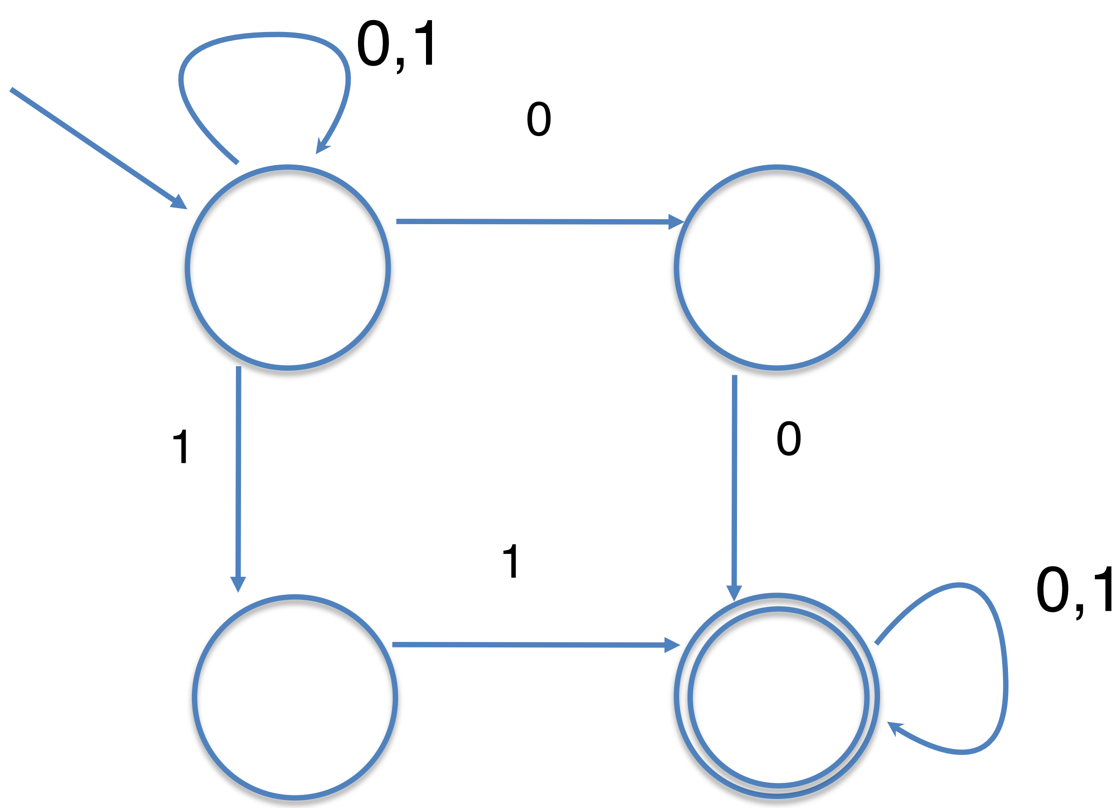


$$\delta_{\text{new}}(q, a) = \cup_{p \in \varepsilon-\text{reach}(q)} \delta(p, a)$$

NFA-ε

- Same NFA!



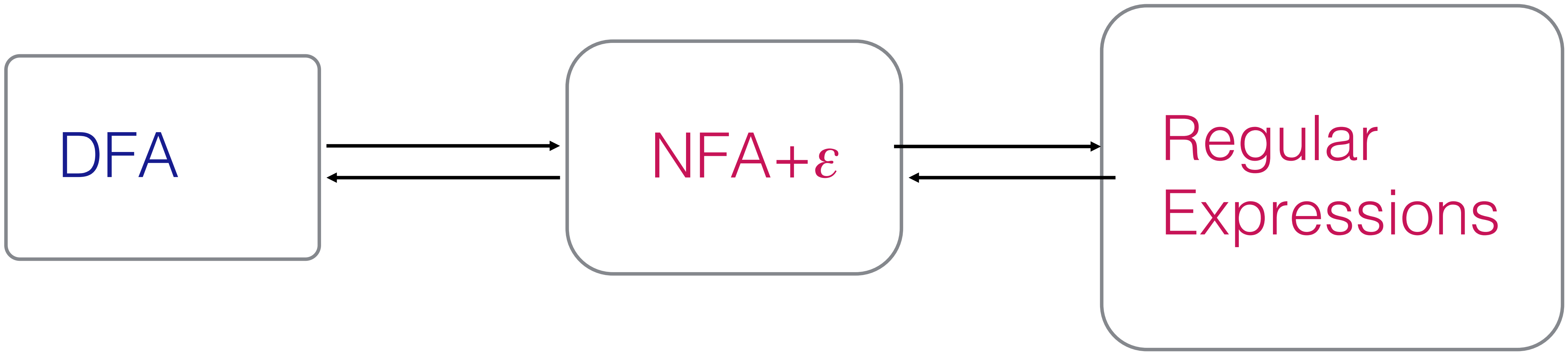


Kleene's theorem

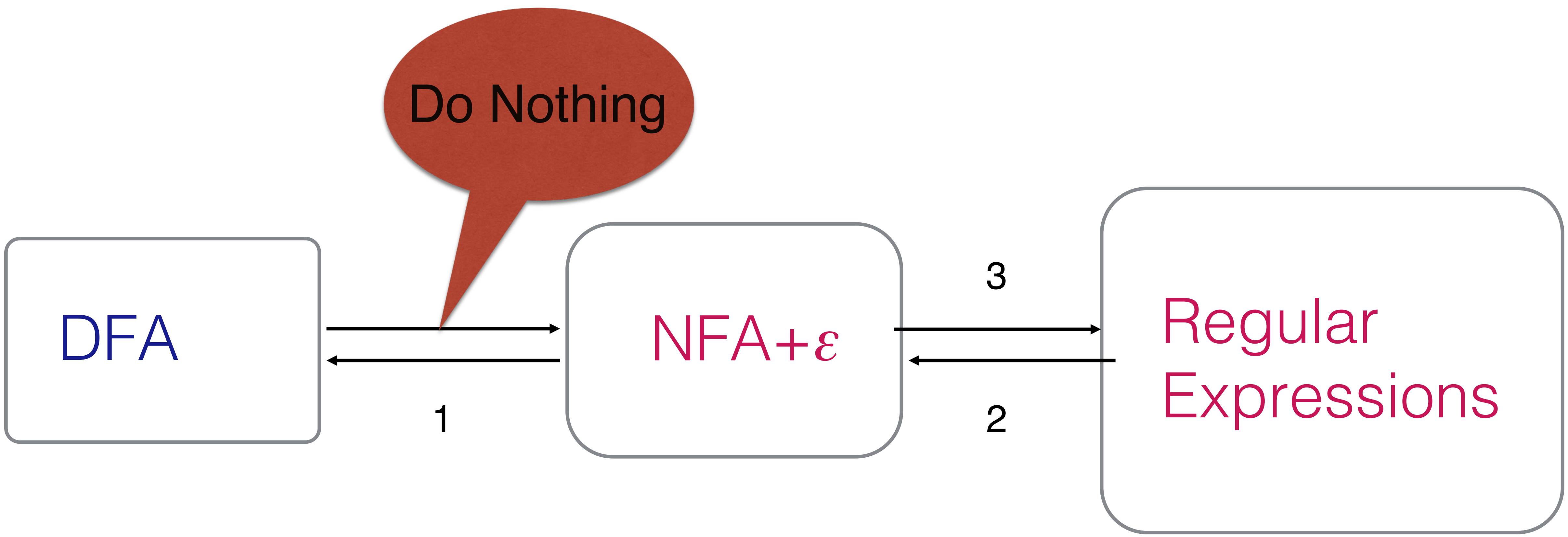
Regular Expression = DFA = NFA

Theorem: A language L can be described by a regular expression if and only if L is the language accepted by a DFA.

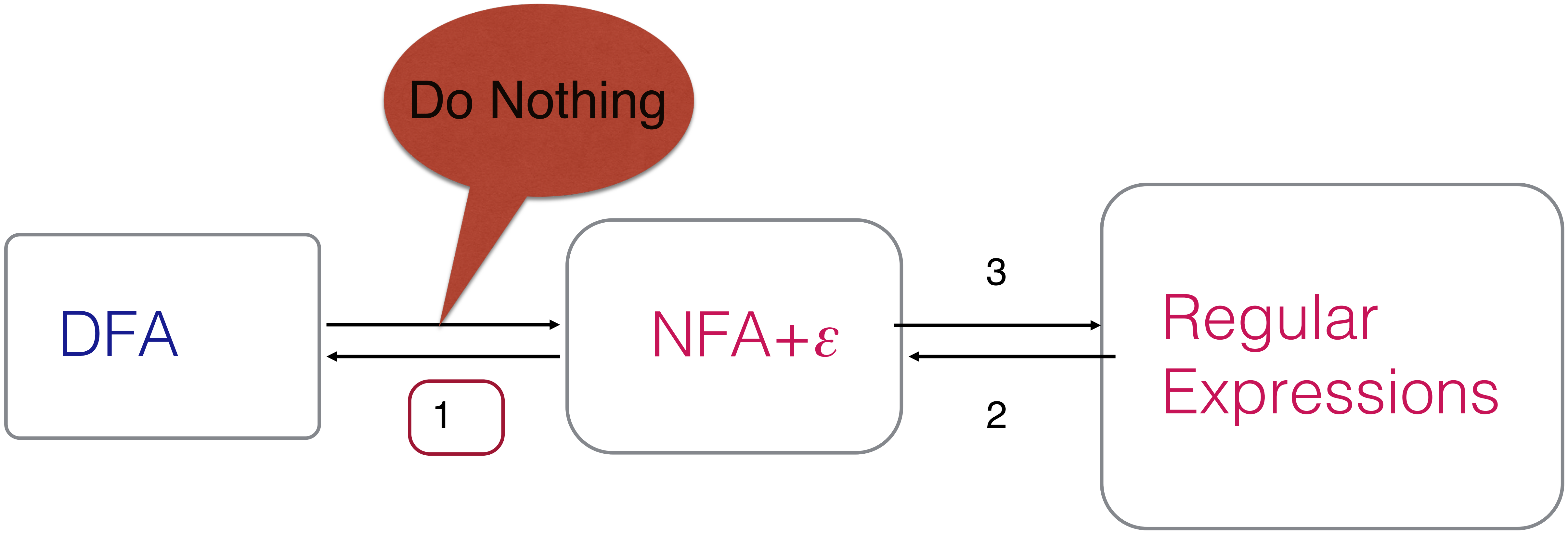
Kleene’s theorem



Kleene’s theorem



Kleene's theorem

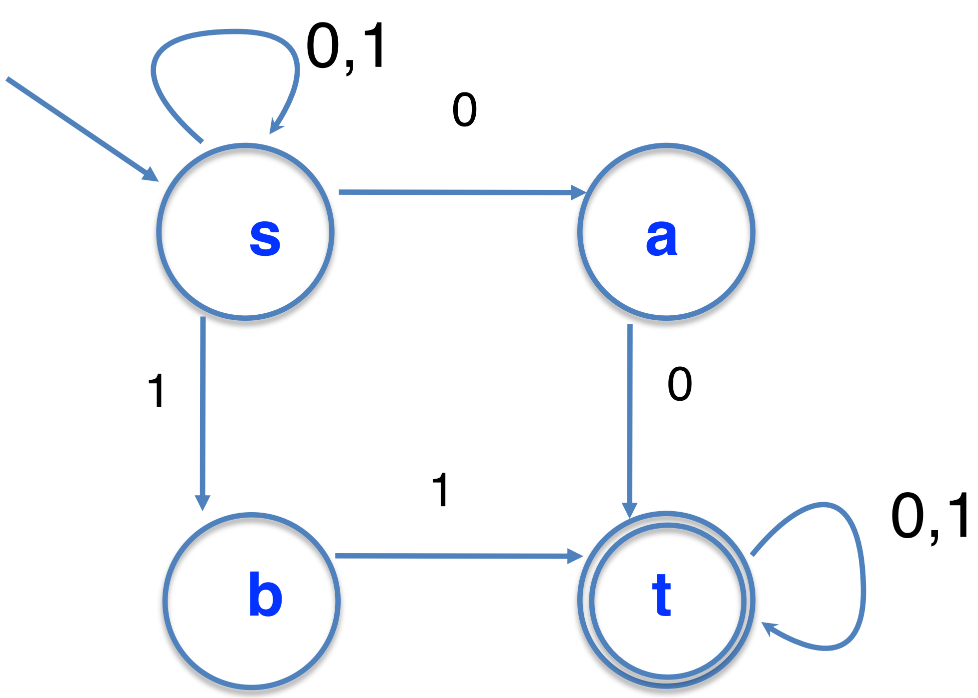


DFA from NFA (aka the subset construction)

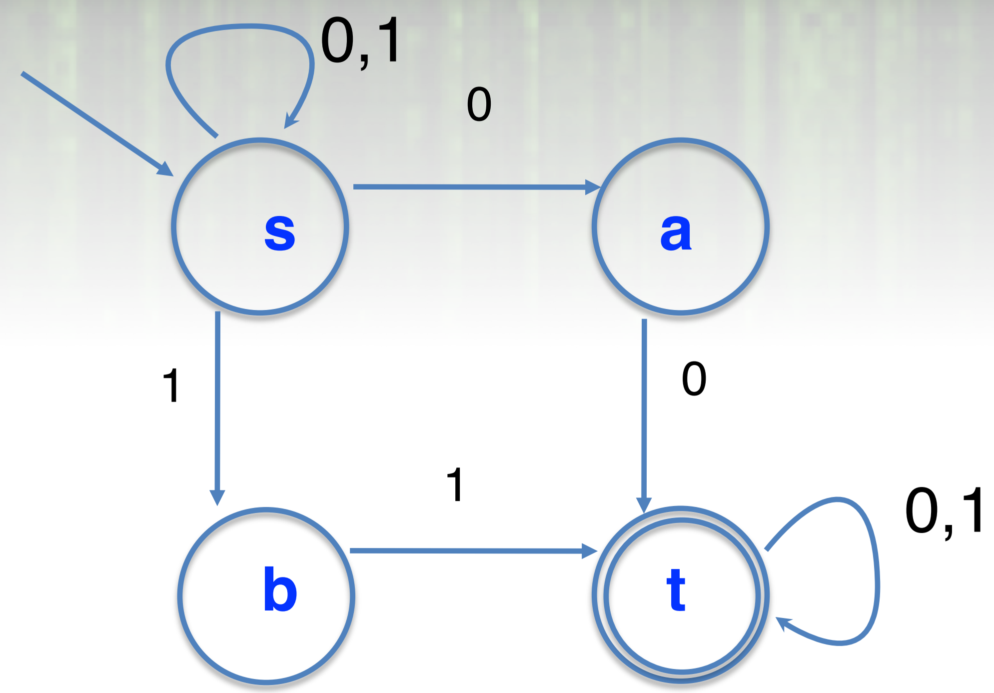
NFA: $N = (\Sigma, Q, \delta, s, A)$

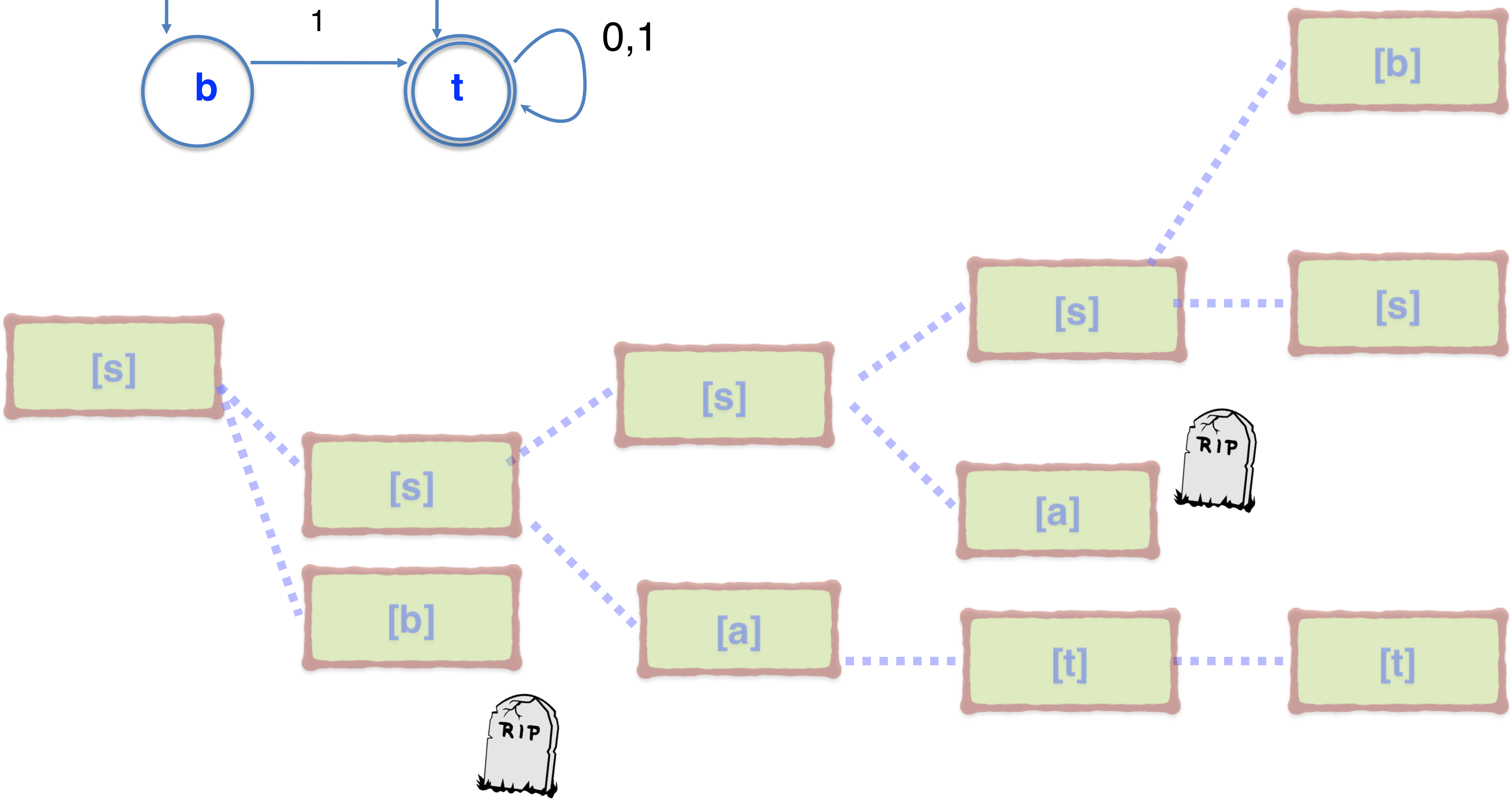
$\delta : Q \times \Sigma \rightarrow \mathsf{P}(Q)$

assume no ε-moves



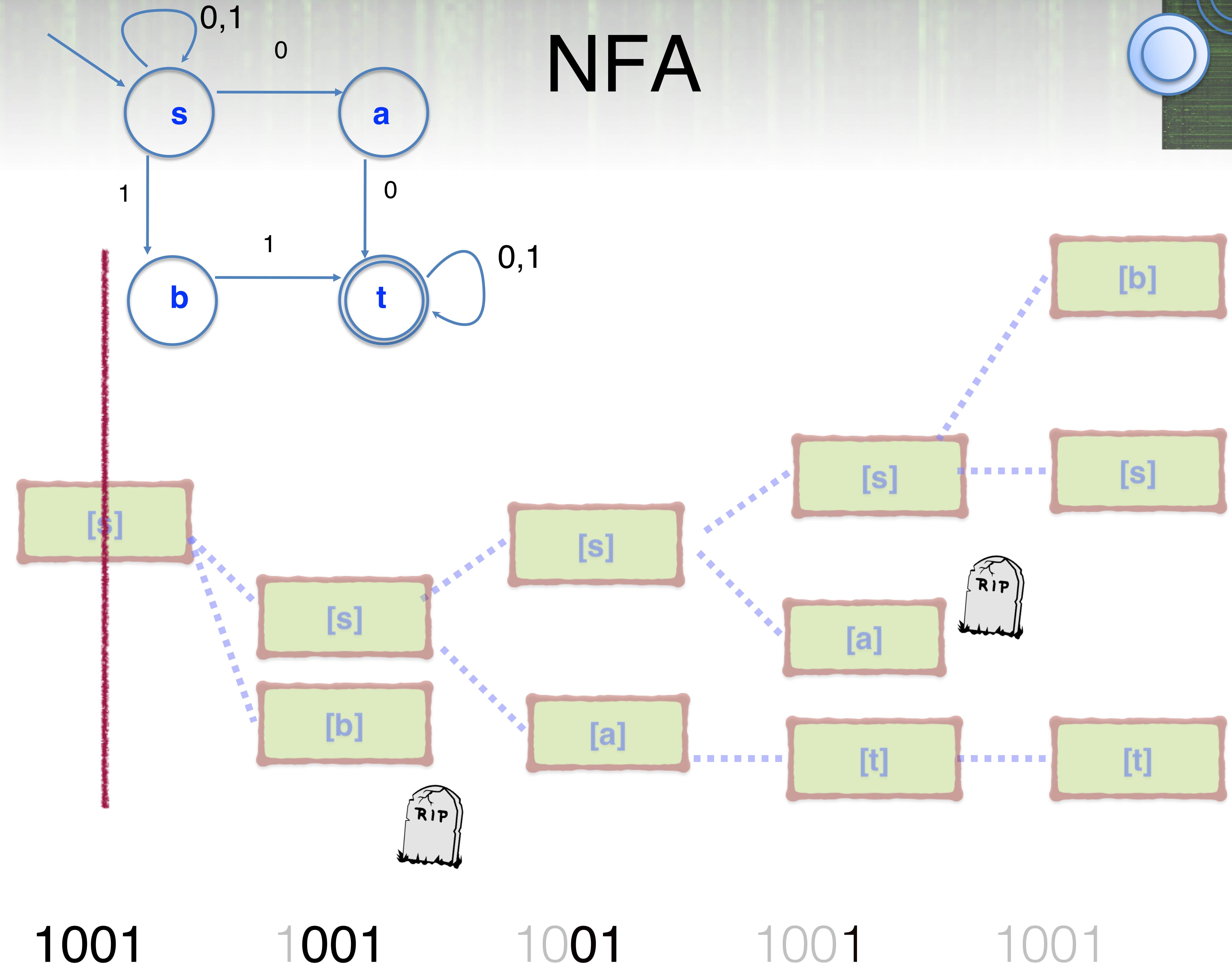
NFA



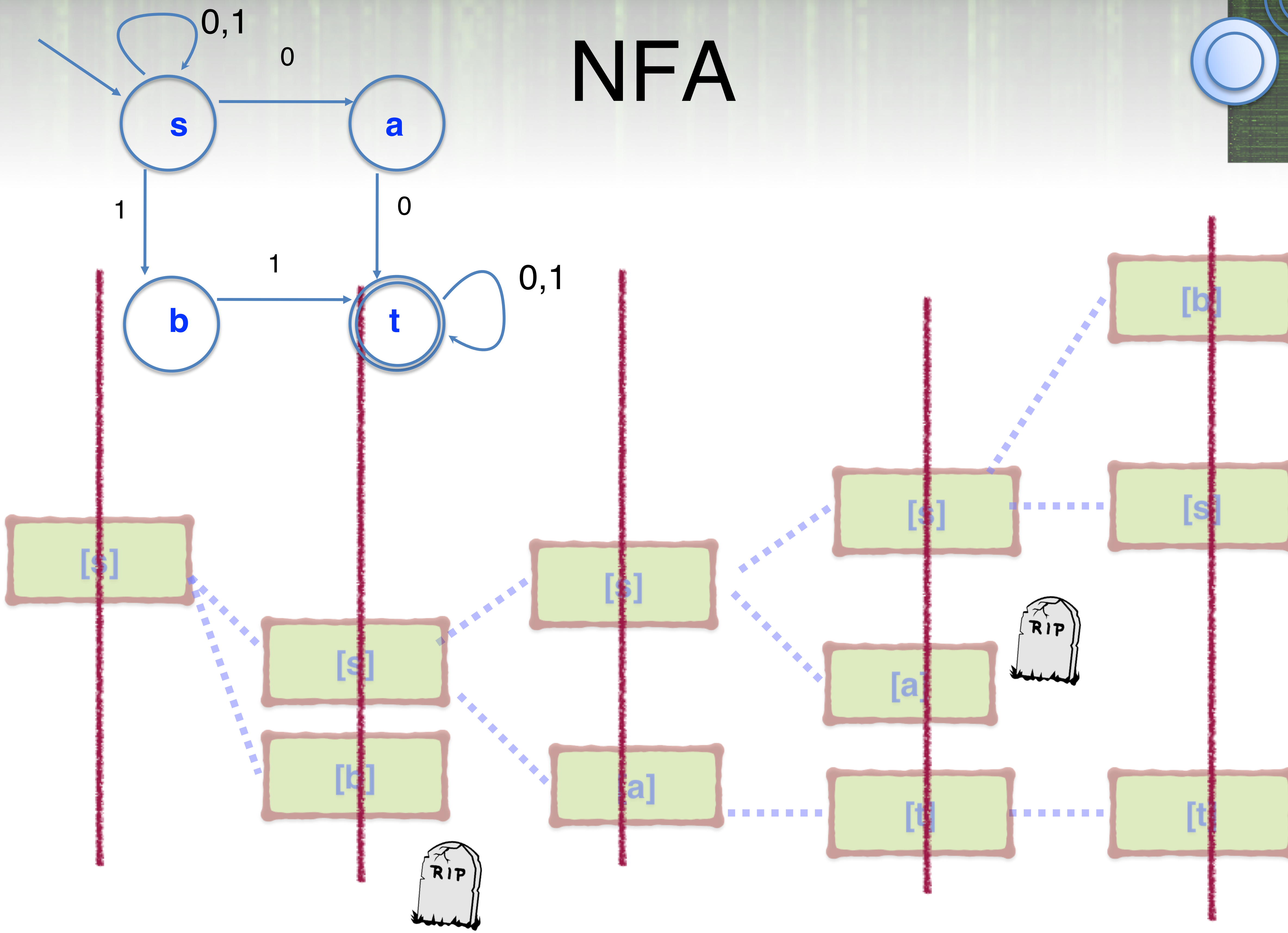


1001 1001 1001 1001 1001

NFA



NFA



1001 1001 1001 1001 1001

NFA to DFA

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

$\delta : Q \times \Sigma \rightarrow \mathrm{P}(Q)$

assume no ε-moves

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

$Q' = 2^Q = \mathrm{P}(Q)$

$s' = \{s\}$

Deterministic state is now a set of (non-deterministic) states

$A' = \{\text{all subsets P of Q s.t. } P \cap A \neq \emptyset\}$

Theorem : $L(N) = L(M_N)$

$\delta' : \mathrm{P}(Q) \times \Sigma \rightarrow \mathrm{P}(Q)$

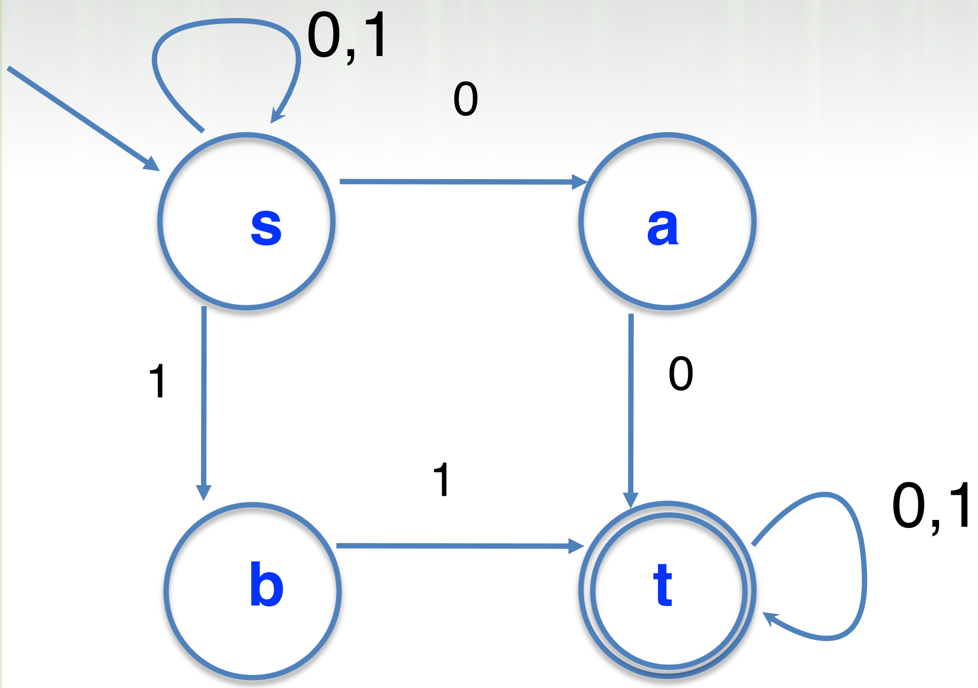
$\delta'(P, a) = \bigcup_{q \in P} \delta(q, a)$

NFA to DFA

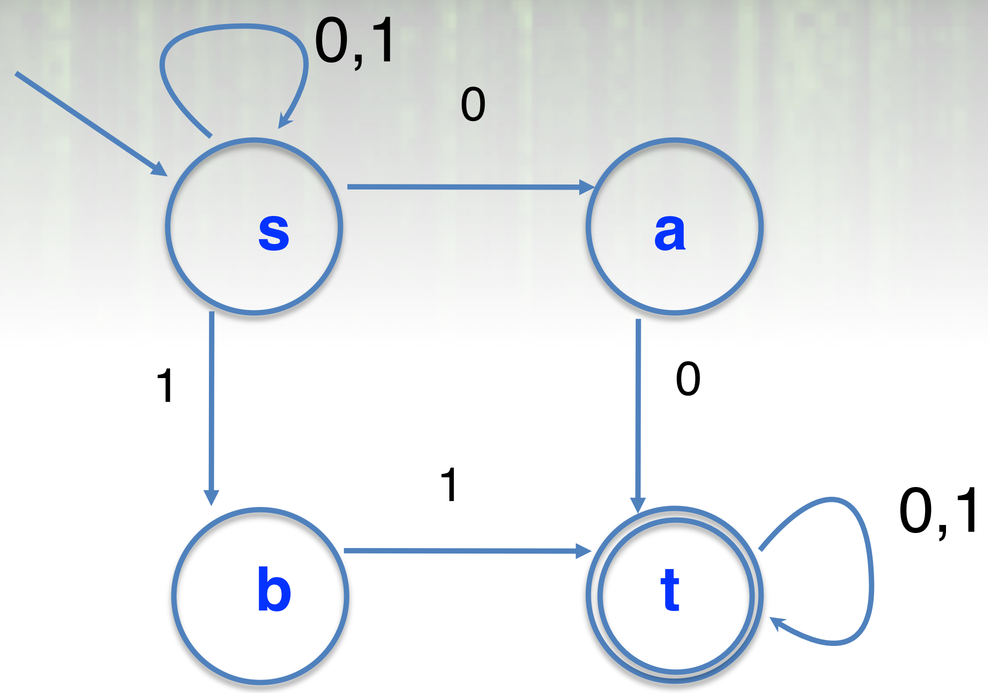
- There are too many states in this DFA, more than
- necessary.

- Construct the DFA incrementally instead,
- by performing BFS on the DFA graph.

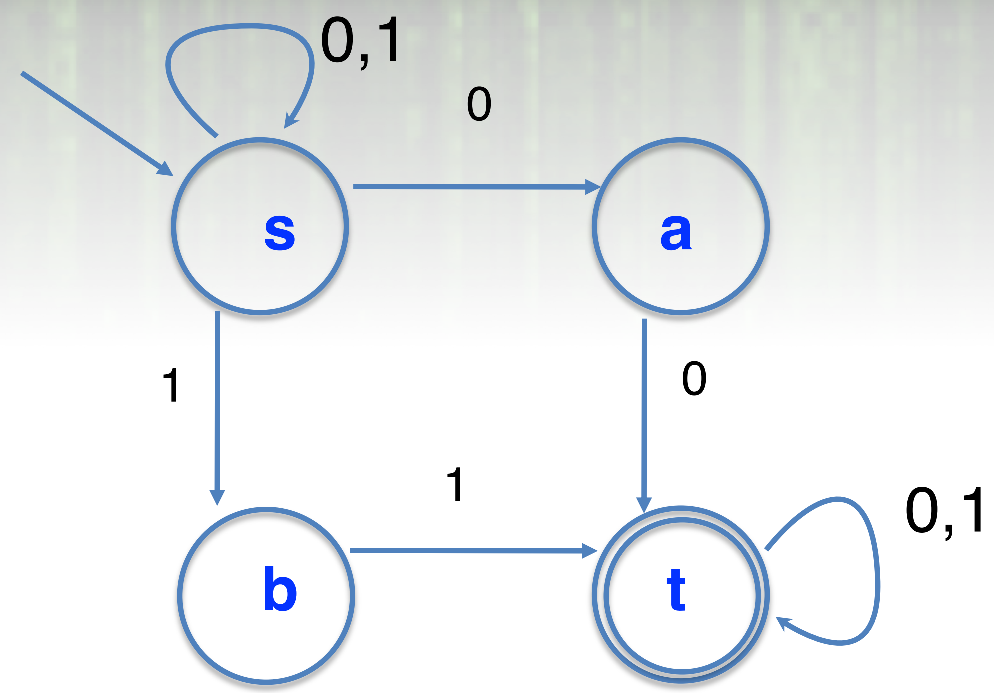
- Prepare a table as follows

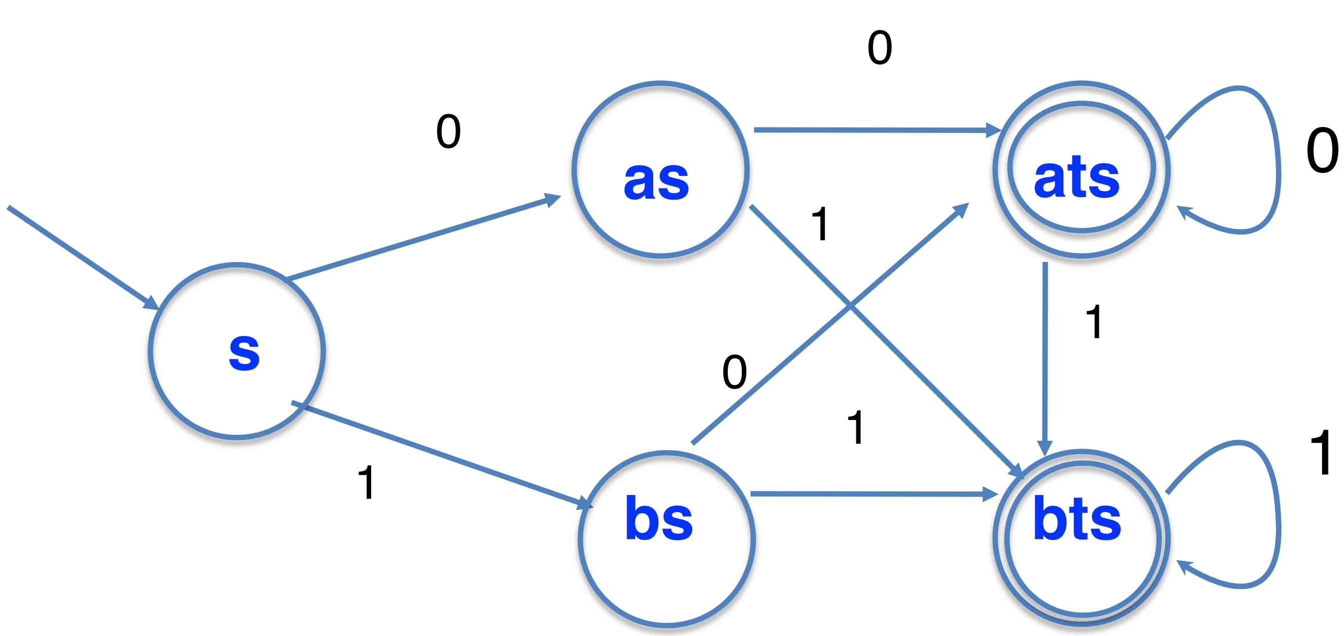


P	ε	$\delta'(P,0)$	$\delta'(P,1)$	$q' \in A'$
s				
as				
bs				
ats				
bts				



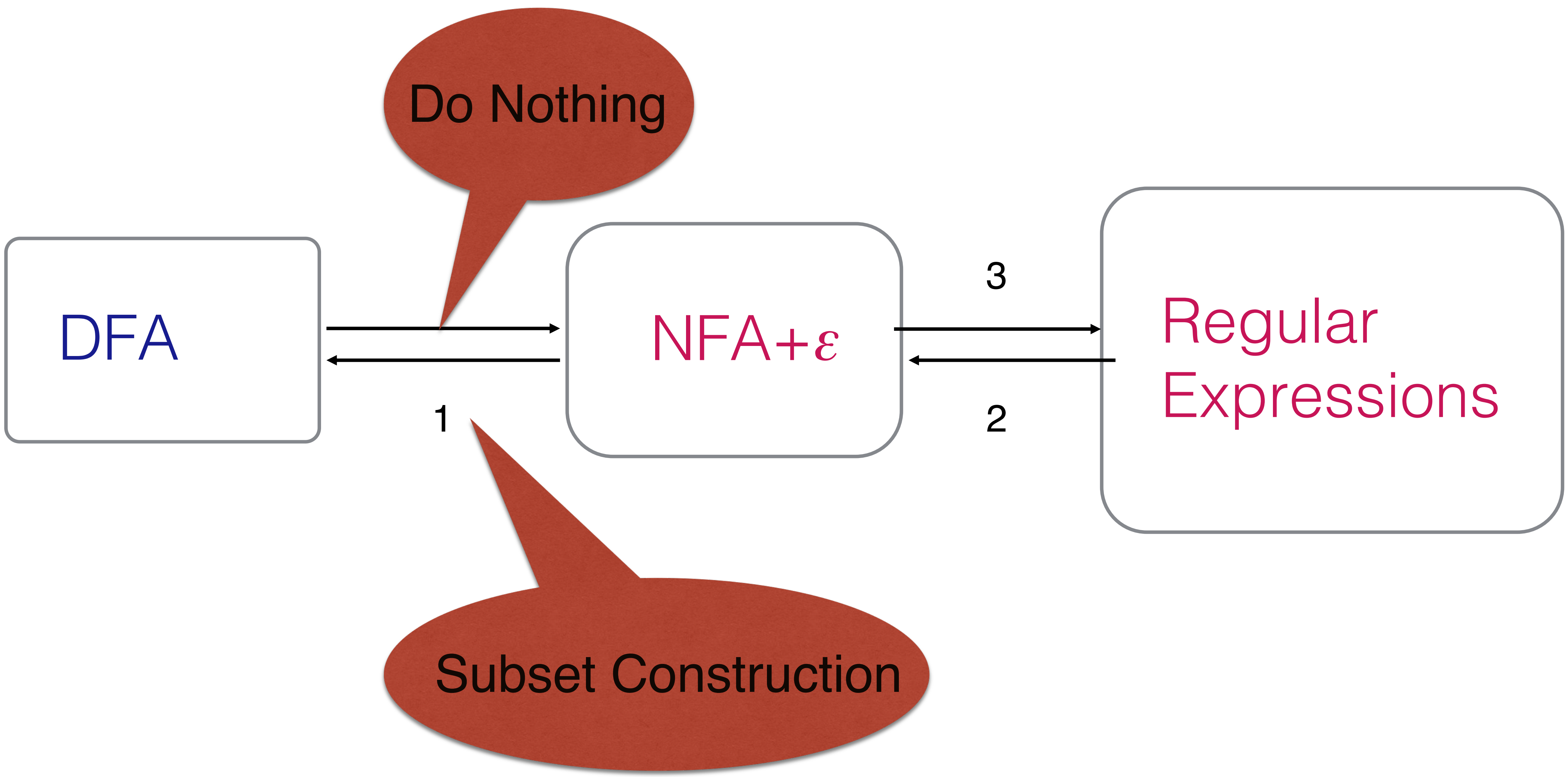
P	ε	$\delta'(P,0)$	$\delta'(P,1)$	$q' \in A'$
s	s	as	bs	No
as	as	ats	bs	No
bs	bs	as	bts	No
ats	ats	ats	bts	Yes
bts	bts	ats	bts	Yes



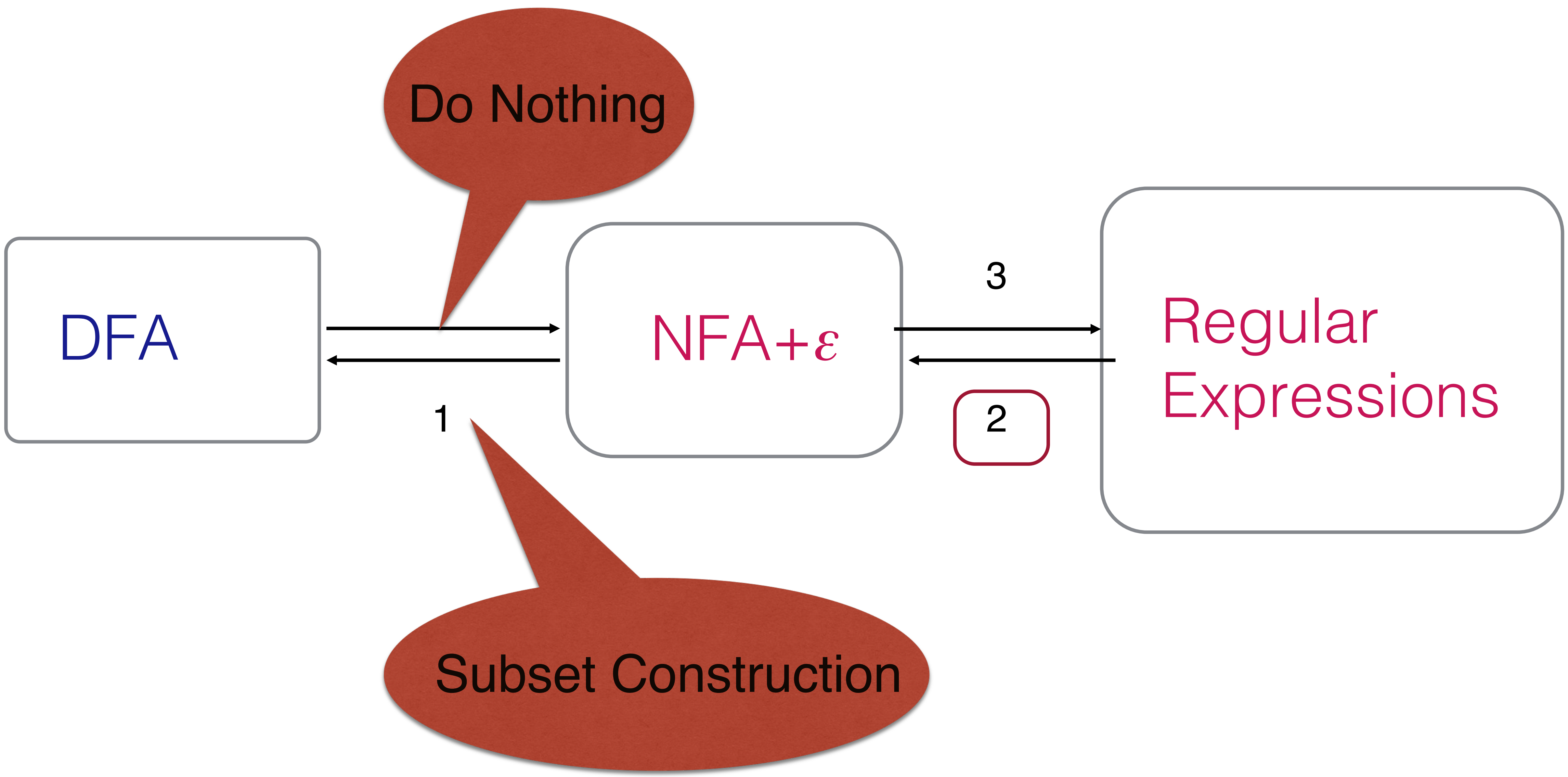


P	ε	$\delta'(P,0)$	$\delta'(P,1)$	$q' \in A'$
s	s	as	bs	No
as	as	ats	bs	No
bs	bs	as	bts	No
ats	ats	ats	bts	Yes
bts	bts	ats	bts	Yes

Kleene's theorem



Kleene's theorem



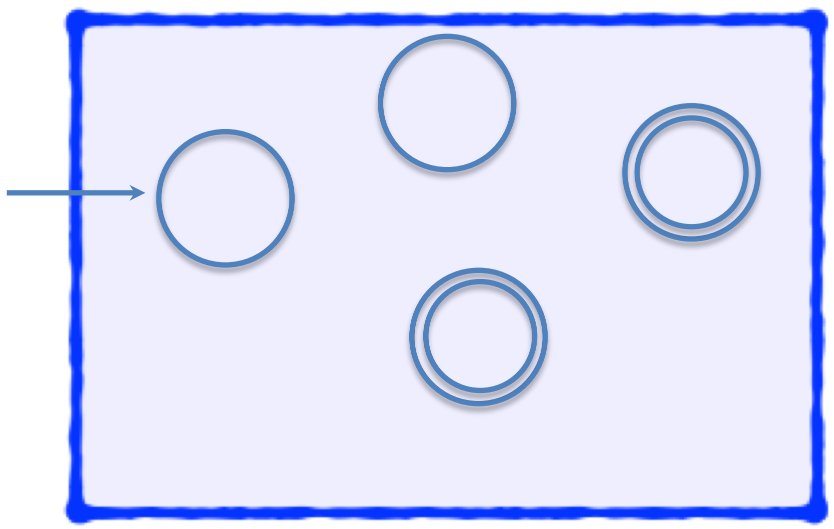
NFAs from Regular Languages

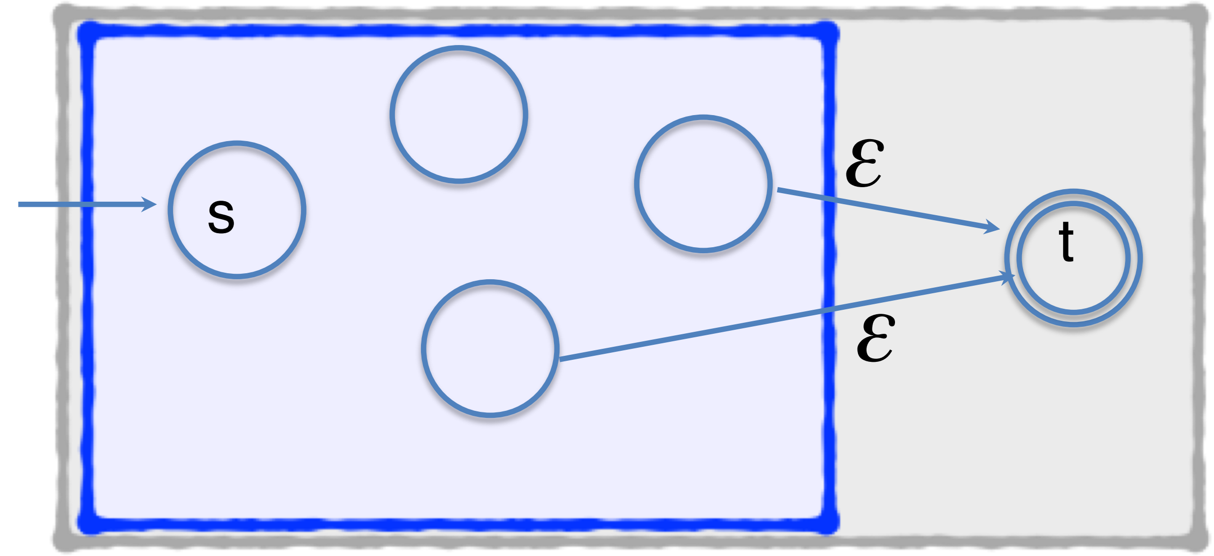
Theorem (Thompsons Algorithm): Every regular language is accepted by an NFA.

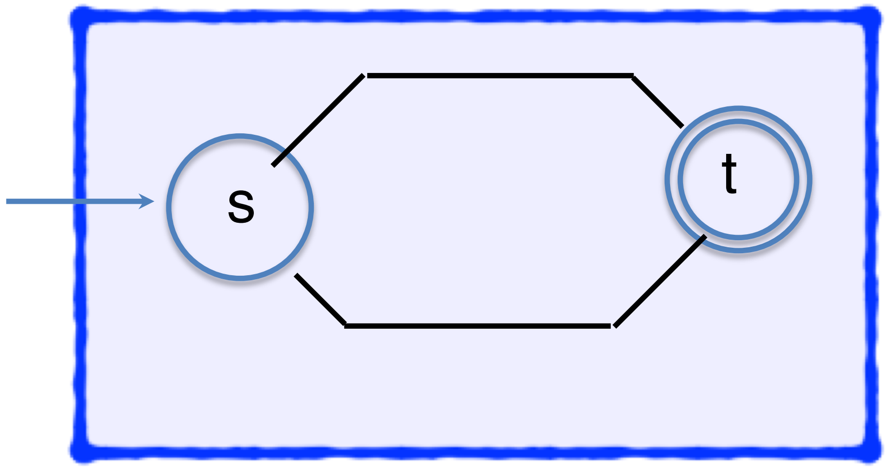
We will show how to get from regular expressions to NFA+ε, *but in a particular way. One accepting state only!*

Single Final State Form

Can compile a given NFA so that there is only one final state
(and there is no transition out of that state)







NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Atomic expressions (Base cases)	
$\emptyset$	$L(\emptyset) = \emptyset$
w for w $\in \Sigma^*$	$L(w) = \{w\}$

Inductively defined expressions	
(r_1+r_2)	$L(r_1+r_2) = L(r_1) \cup L(r_2)$
$(r_1 r_2)$	$L(r_1 r_2) = L(r_1)L(r_2)$
(r^*)	$L(r^*) = L(r)^*$

NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Base Case 1: L=Ø

What is a NFA for L?

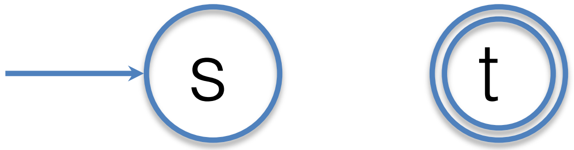
NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Base Case 1: L=Ø

What is a NFA for L?



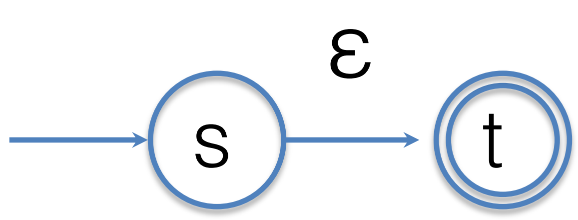
NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Base Case 2: L={ε}

What is a NFA for L?



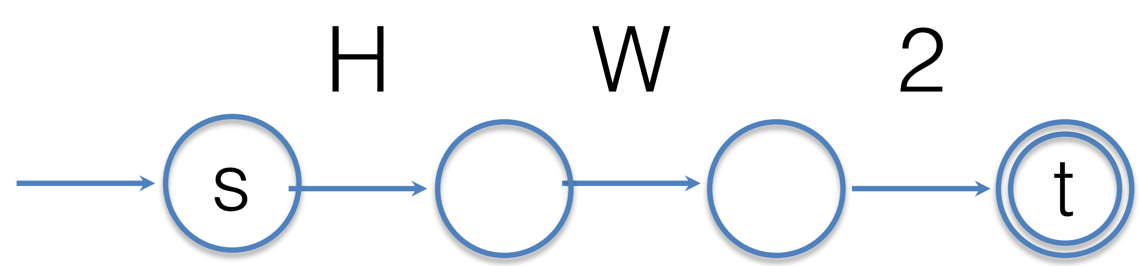
NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Base Case 3: L={a}, some string in Σ* (e.g. HW2)

What is a NFA for L?



NFAs from Regular Languages

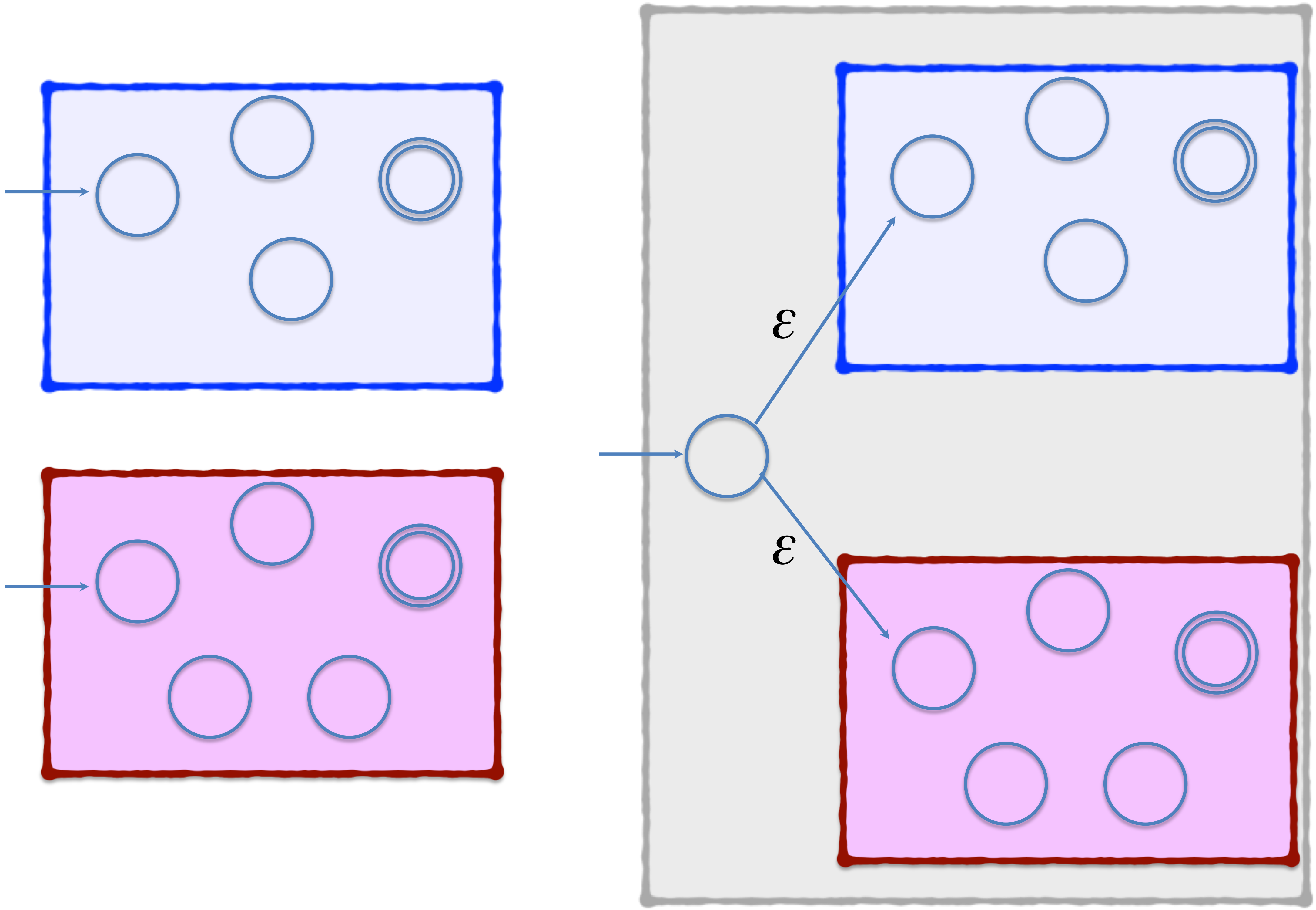
Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 1: $L = A \cup B$

What is a NFA for L?

Closure Under Union



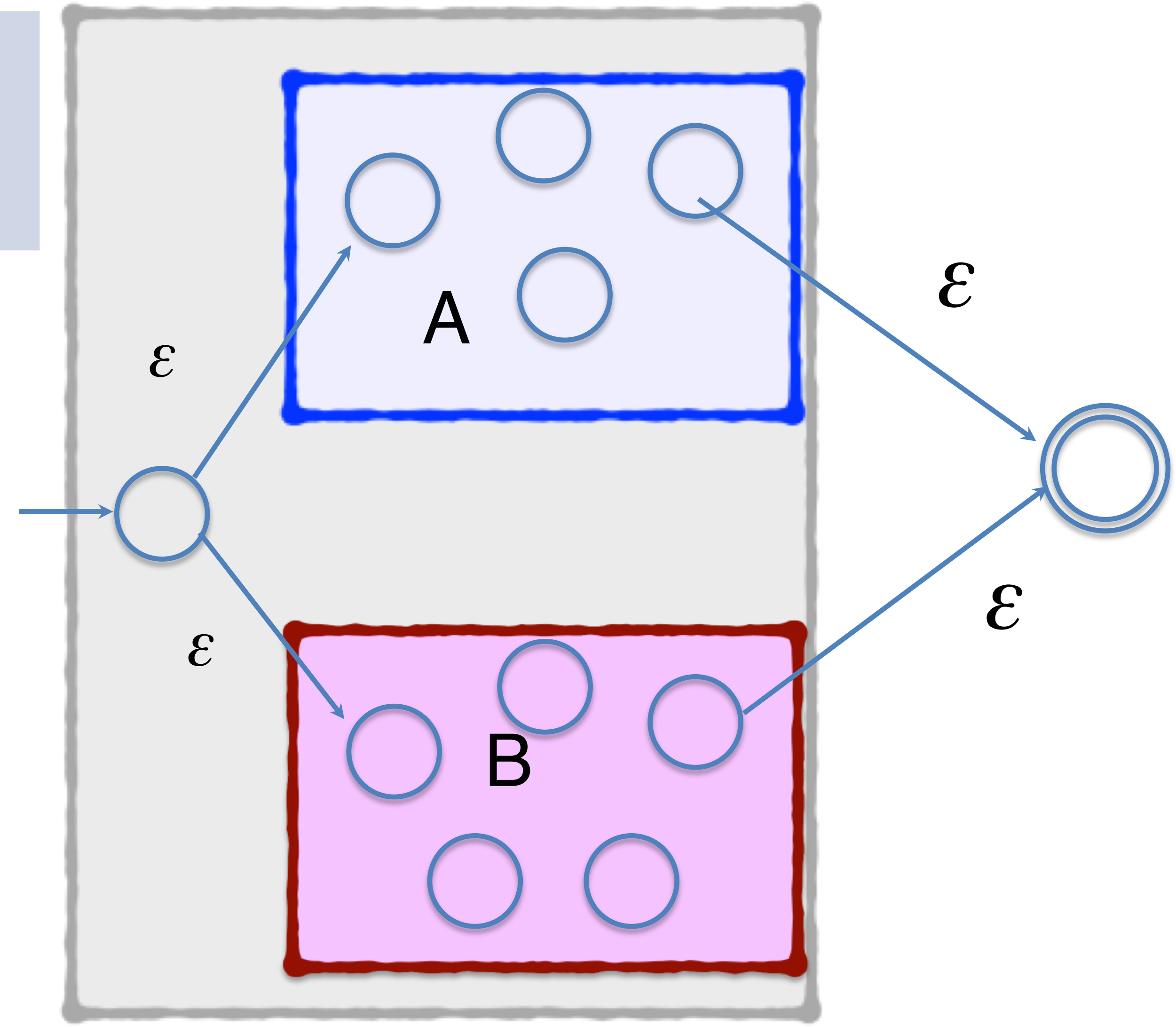
NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 1: L=A $\cup$ B

What is a NFA for L?



NFAs from Regular Languages

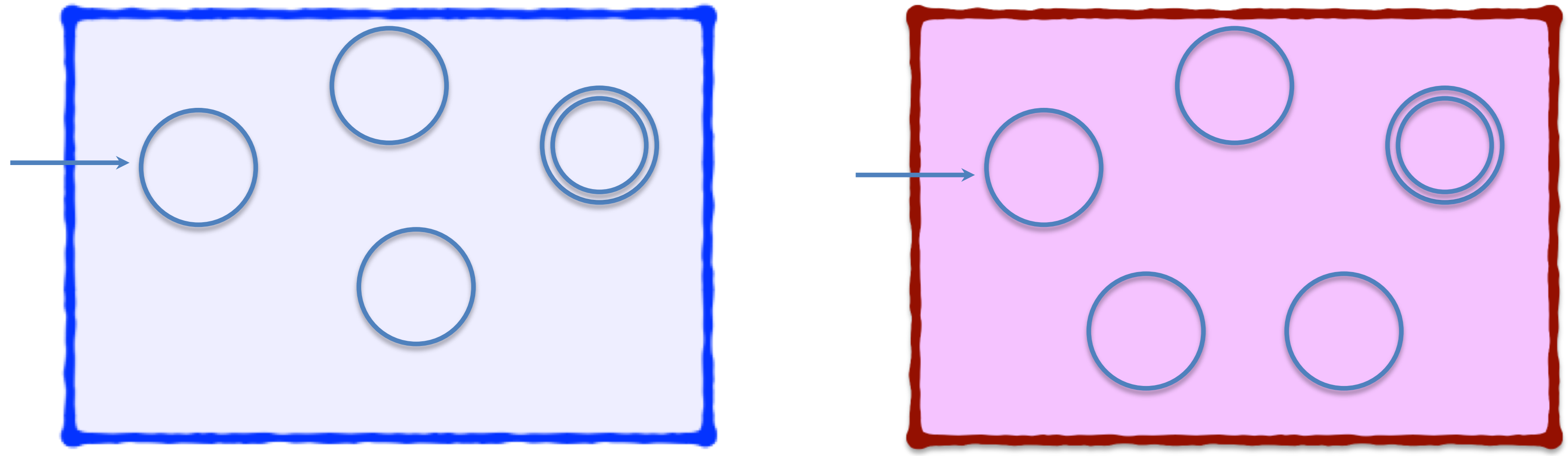
Theorem : Every regular language is accepted by an NFA.

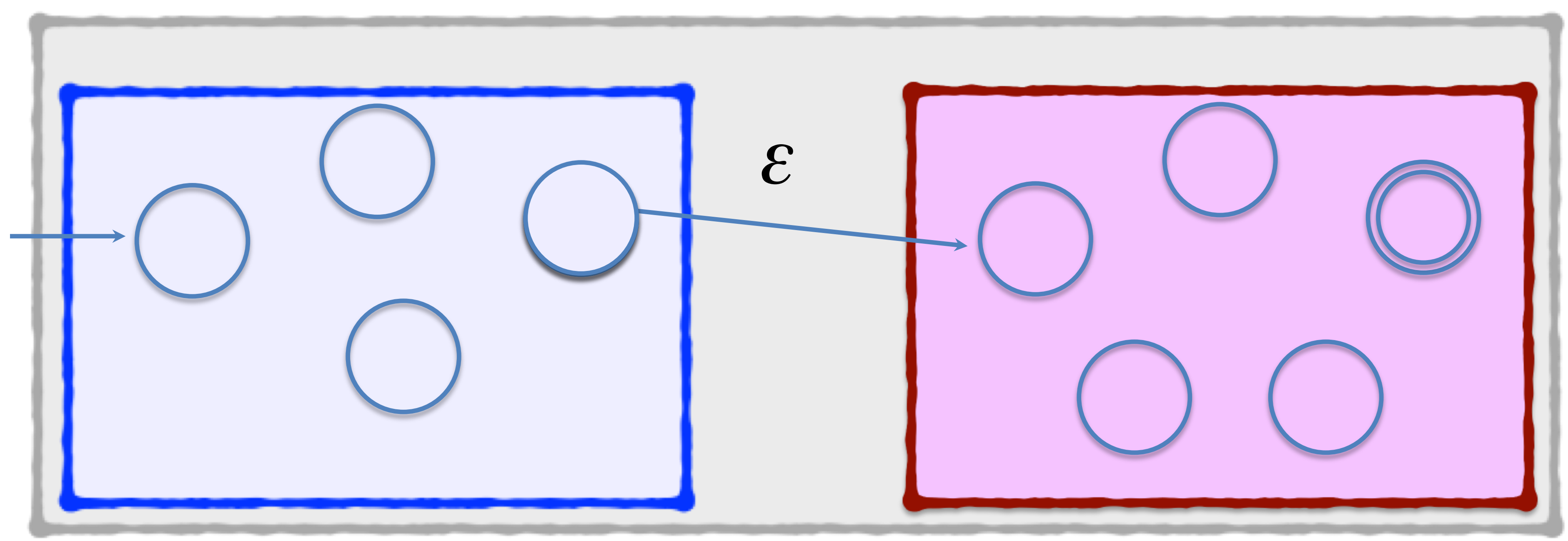
Proof: Recall definition or Regular Language.

Inductive case 2: L=AB

What is a NFA for L?

Closure Under Concatenation





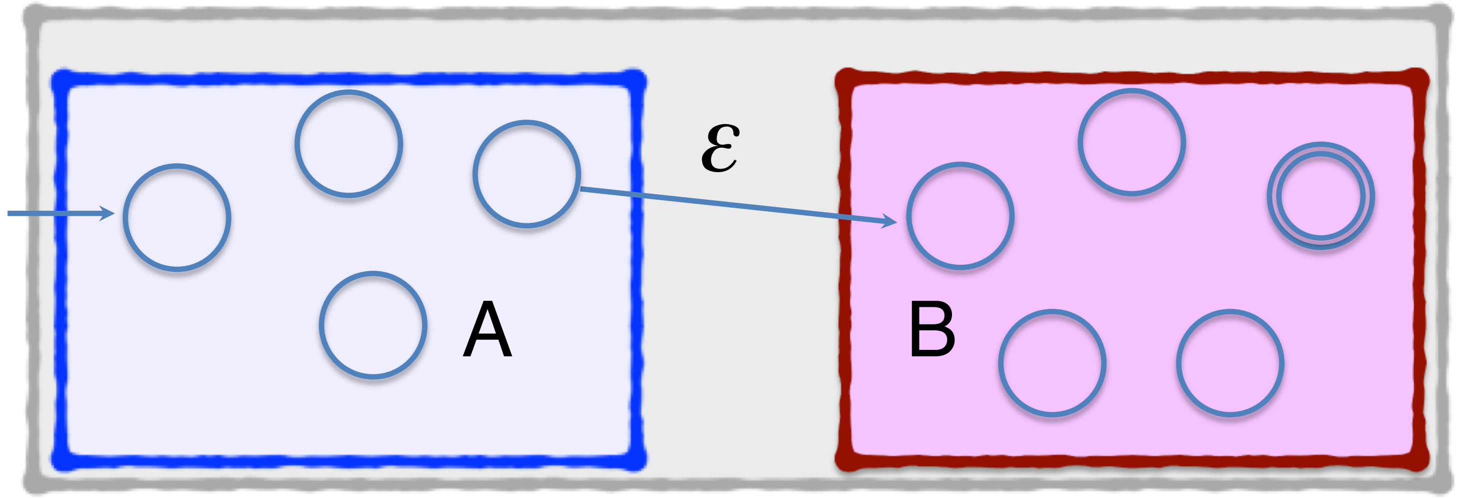
NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 2: L=AB

What is a NFA for L?



NFAs from Regular Languages

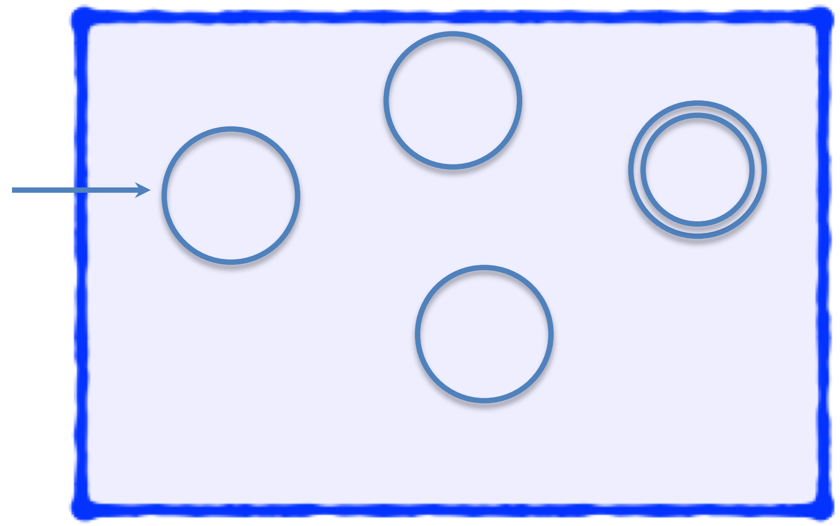
Theorem : Every regular language is accepted by an NFA.

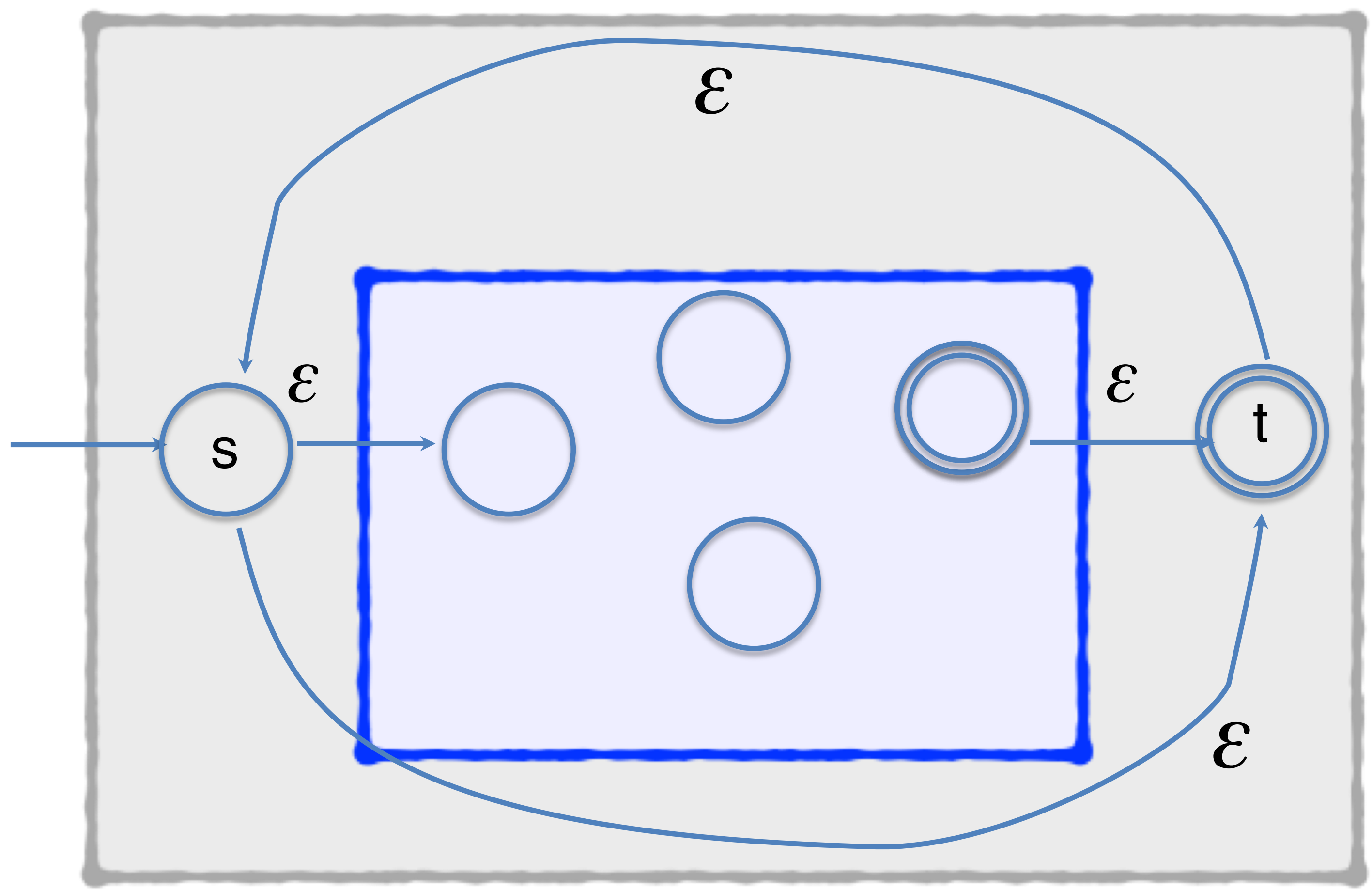
Proof: Recall definition or Regular Language.

Inductive case 3: L=A*

What is a NFA for L?

Closure Under Kleene Star





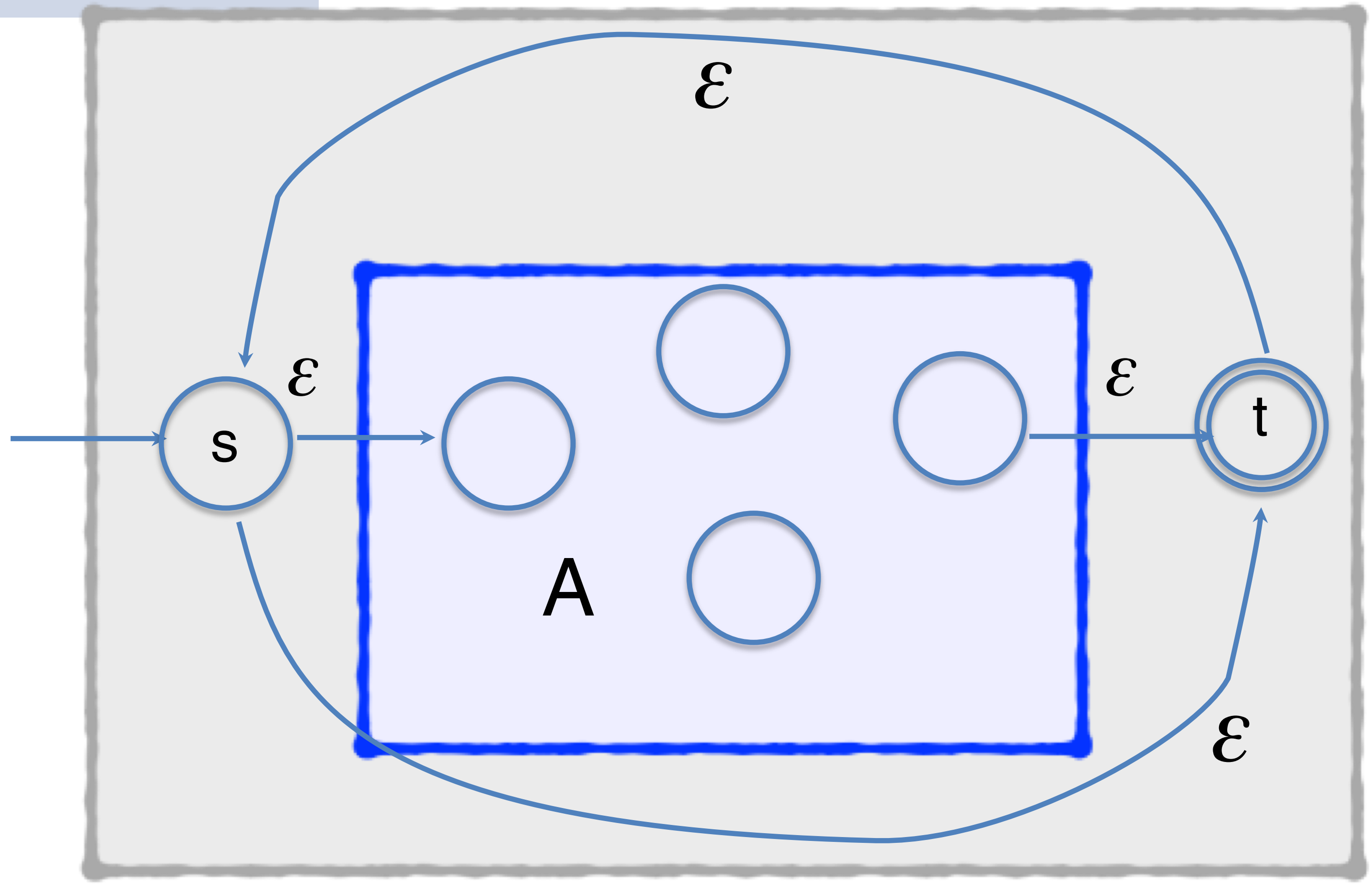
NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 3: L=A*

What is a NFA for L?



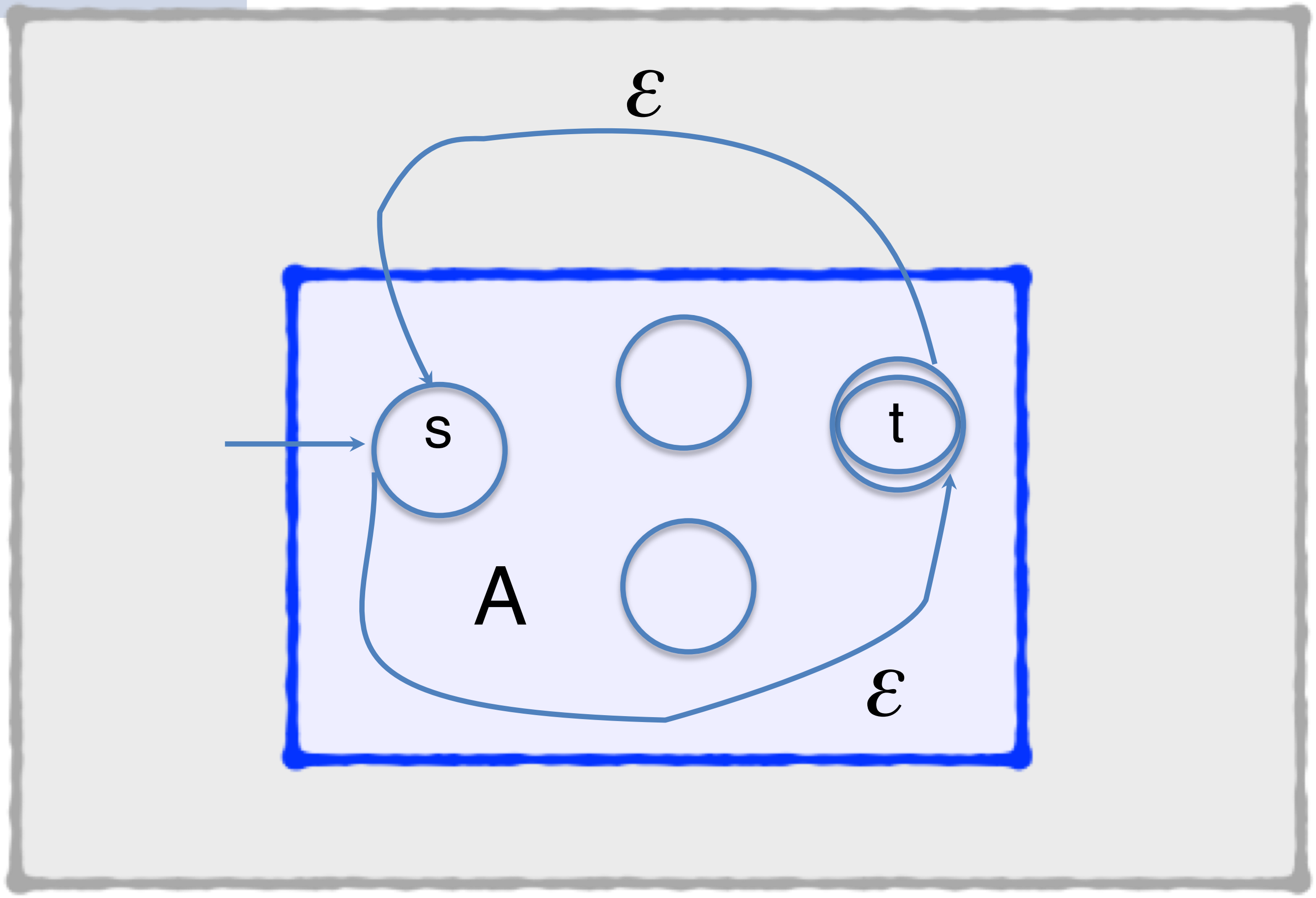
NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 3: L=A*

Why not?

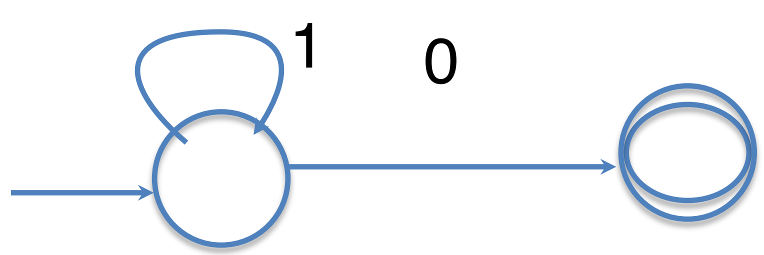


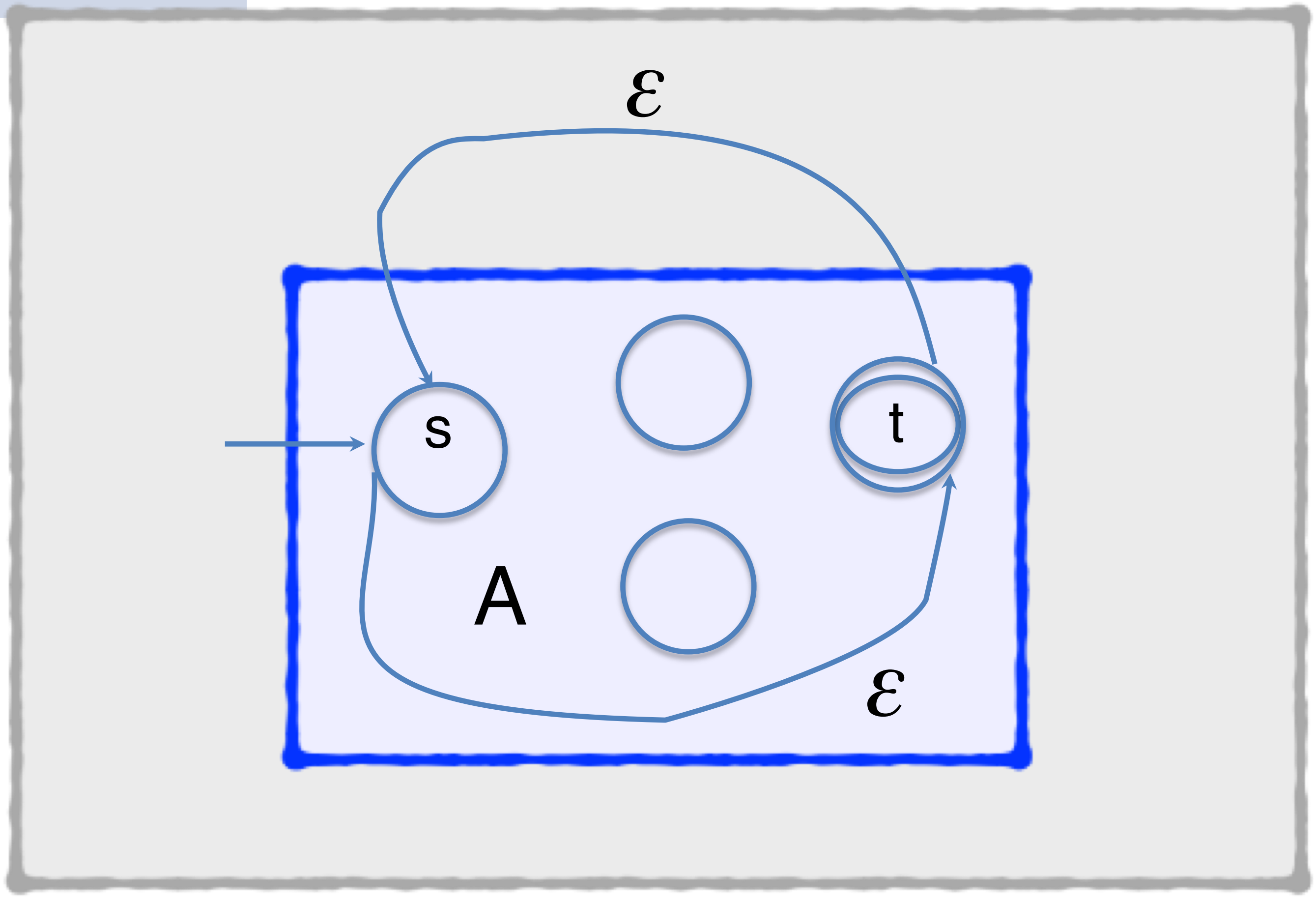
NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 3: L=A*



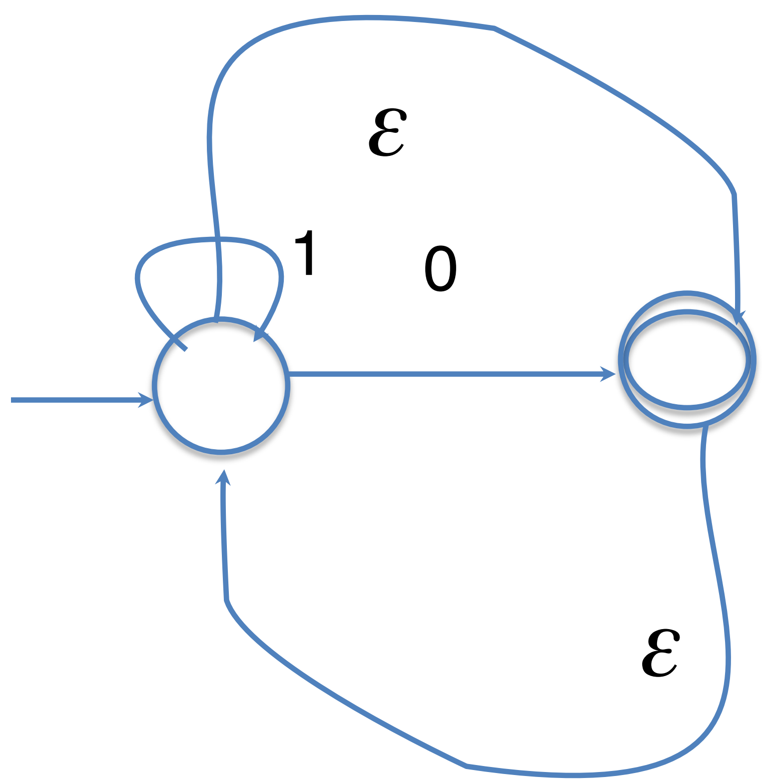


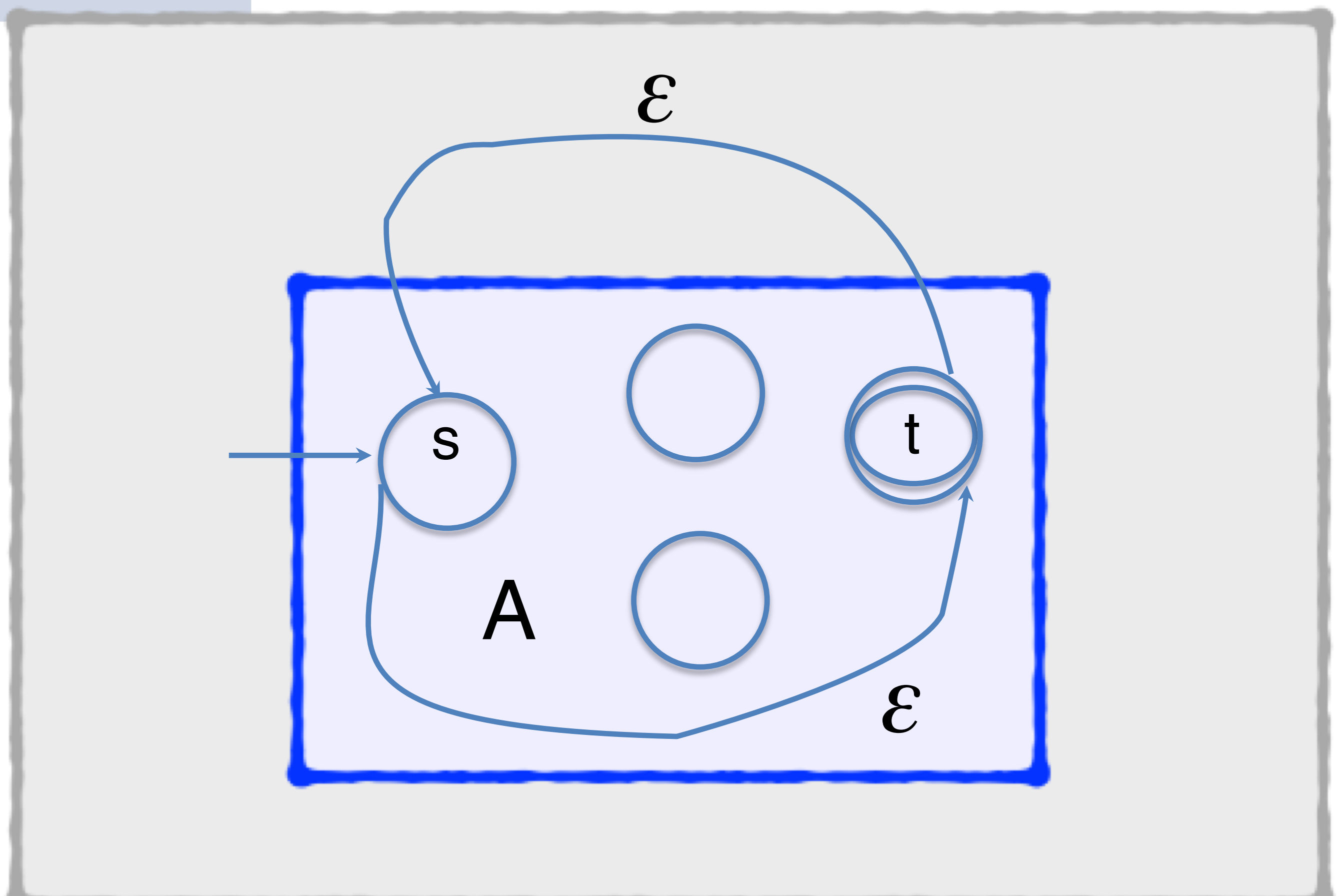
NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 3: L=A*





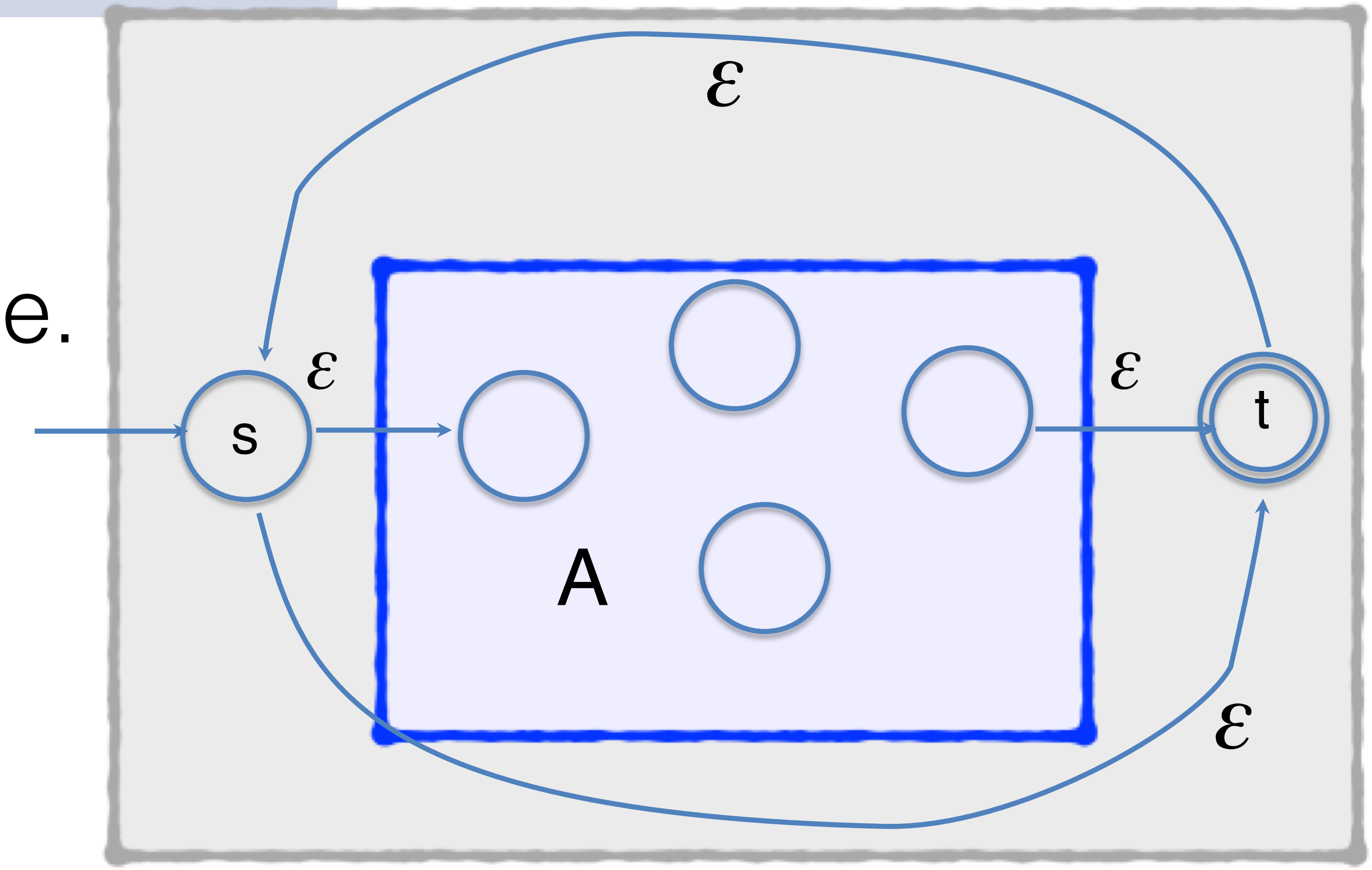
NFAs from Regular Languages

Theorem : Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 3: L=A*

I need the new start state.



NFAs & Regular Languages

Example : L given by regular expression $(10+1)^*$

NFAs & Regular Languages

Example : L given by regular expression $(10+1)^*$