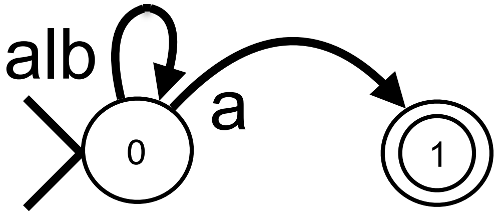
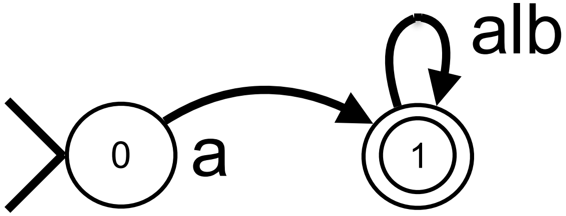
Regular Languages

cl

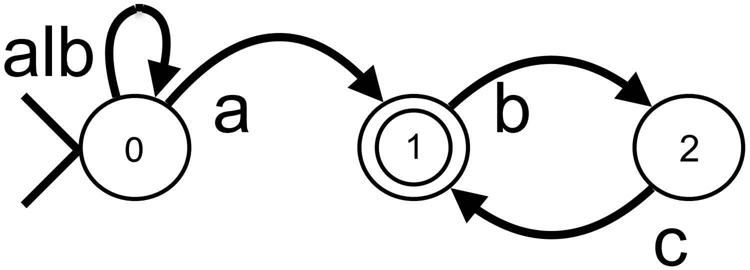
- The language accepted by a state
- Arden's Lemma
- NFA - DFA



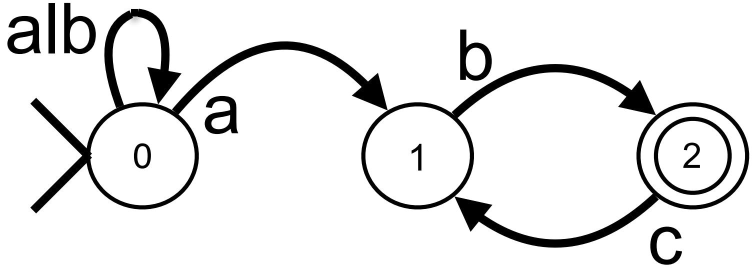
(a|b)*a



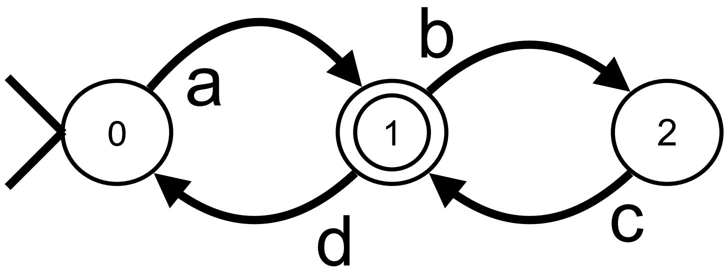
a(a|b)*



(a|b)*a(bc)*



(a|b)*ab(cb)*



a(da|bc)*

rules for regular languages

The following equations hold for any sets of strings **R,S,T**

- $\{\} \mid S =$
- $\{\}\, S =$
- $\varepsilon\, S =$
- $\varepsilon^* =$
- $\{\}^* =$
- $R\,(S \mid T) \;=$
- $S\,R \mid T\,R =$
- $S^*\,S \mid \varepsilon =$

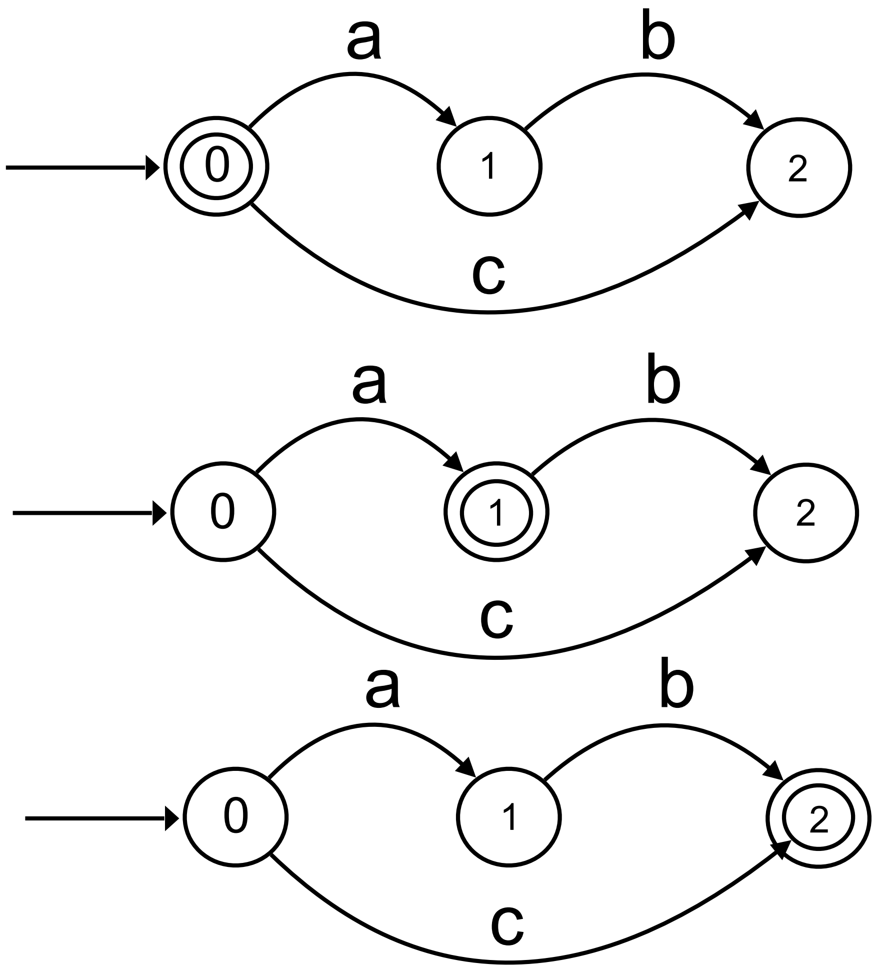
rules for regular languages

The following equations hold for any sets of strings **R,S,T**

- {} | S = {}|S = S
- {} S = S {} = {}
- ε S = S ε = S
- ε* = ε
- {}* = {}
- R (S | T) = R S | R T
- (S | T) R = S R | T R
- S* = S* S | ε = S S* | ε

Is there a regular expression for every FSM?

L_0
L_1
L_2

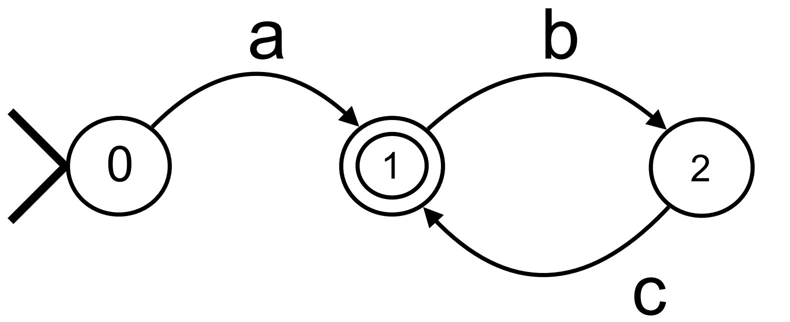


Let L_i be the language accepted if **i** is the accepting state

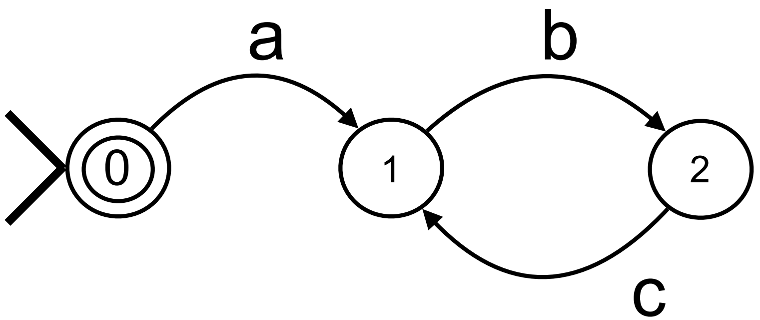
$L_0 = \varepsilon$
$L_1 = L_0\, a$
$L_2 = L_1\, b \mid L_0\, c$

$L_2 = L_0\, a\, b \mid \varepsilon\, c$
$L_2 = \varepsilon\, a\, b \mid \varepsilon\, c$
$L_2 = a\, b \mid c$

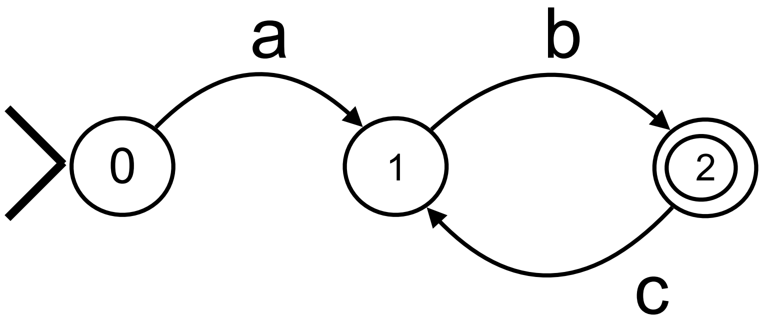
Is there a regular expression for every FSM?



$L_1 =$

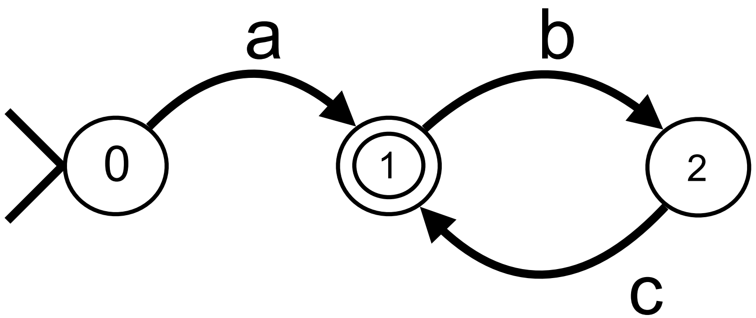


$L_0 = \varepsilon$

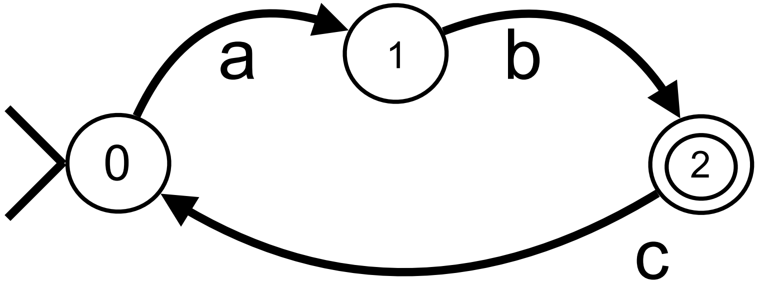


$L_2 =$

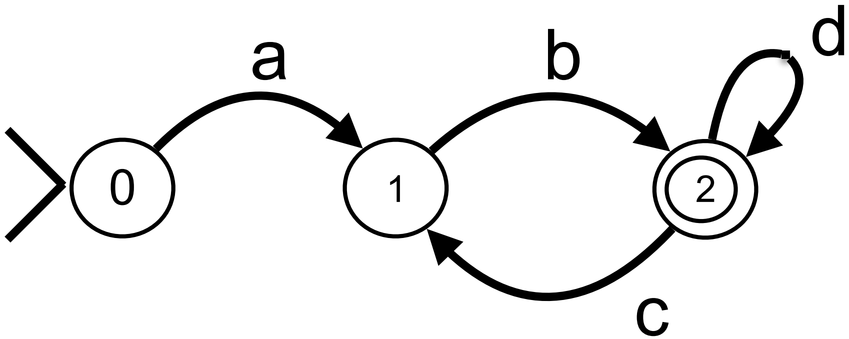
Is there a regular expression for every FSM?



?

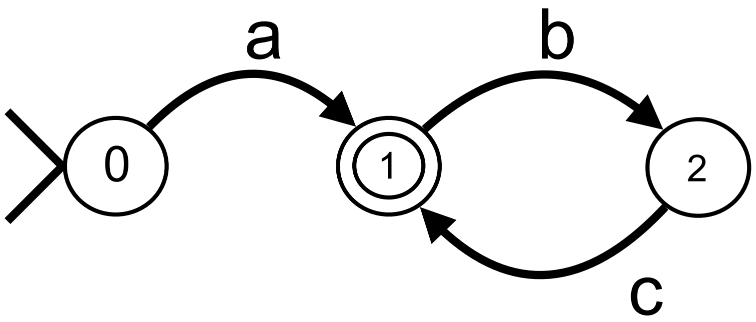


?

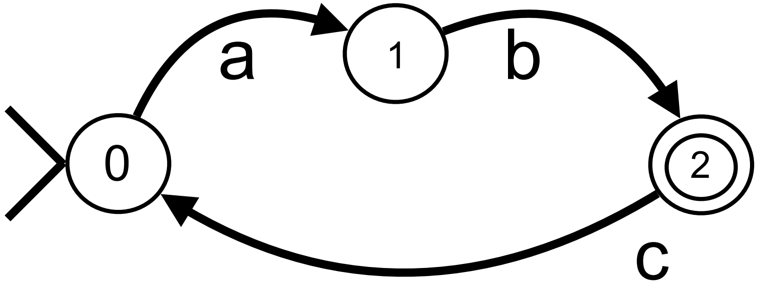


?

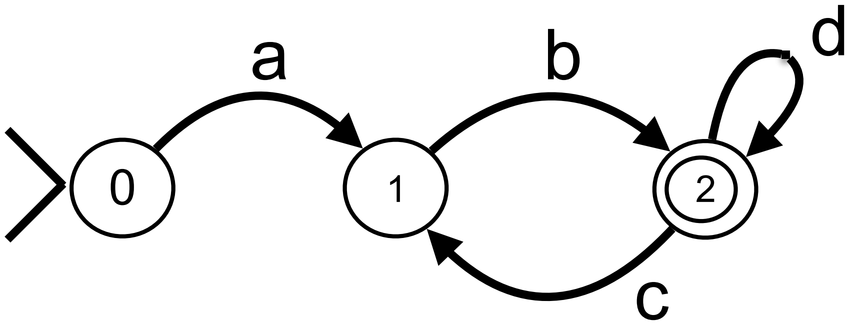
Is there a regular expression for every FSM?



a(bc)*

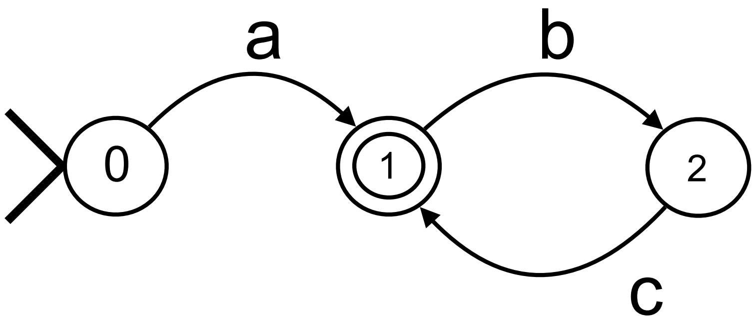


ab(cab)*

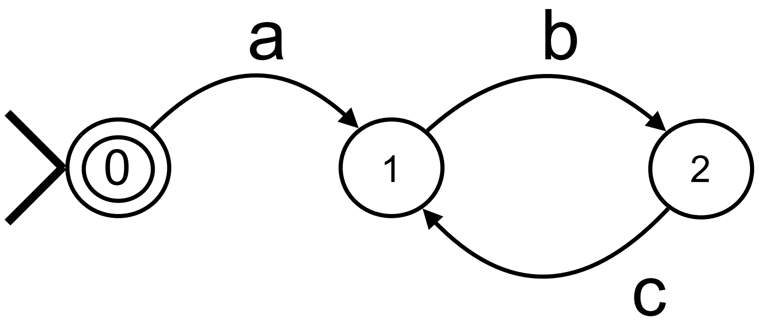


ab(cb | d)*

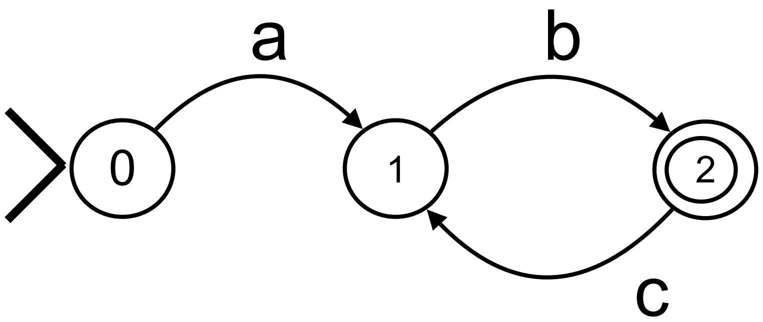
Is there a regular expression for every FSM?



$$L_1 = L_0\, a \mid L_2\, c$$

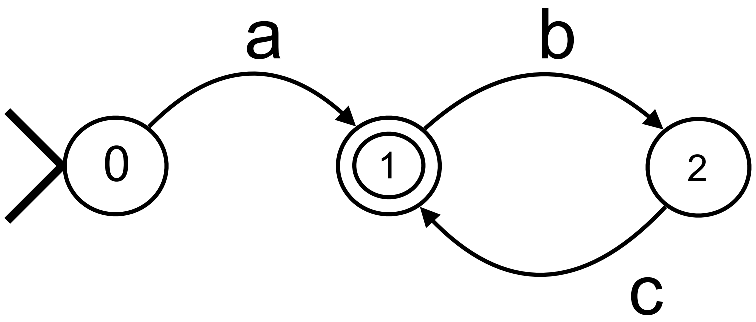


$$L_0 = \varepsilon$$

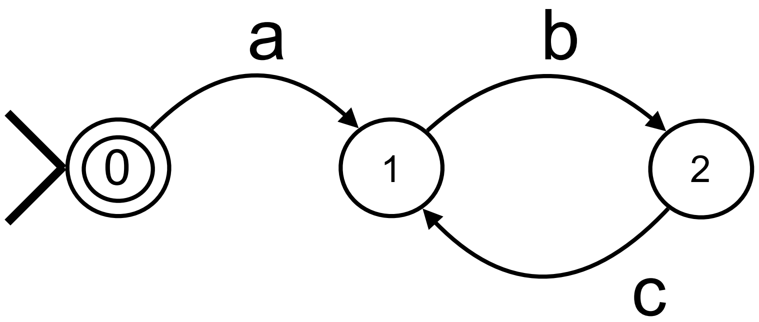


$$L_2 = L_1\, b$$

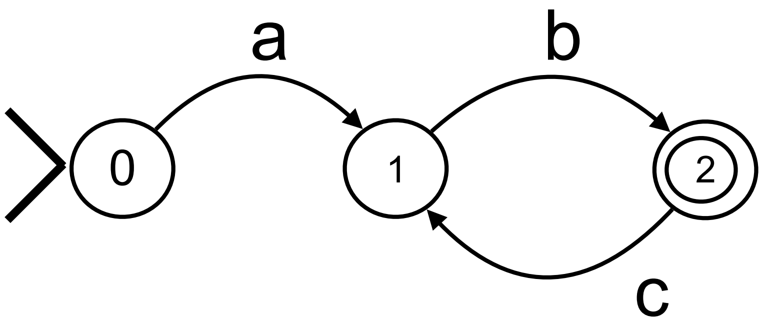
Is there a regular expression for every FSM?



$$L_1 = L_0\, a \mid L_2\, c$$
$$= a \mid L_1\, bc$$

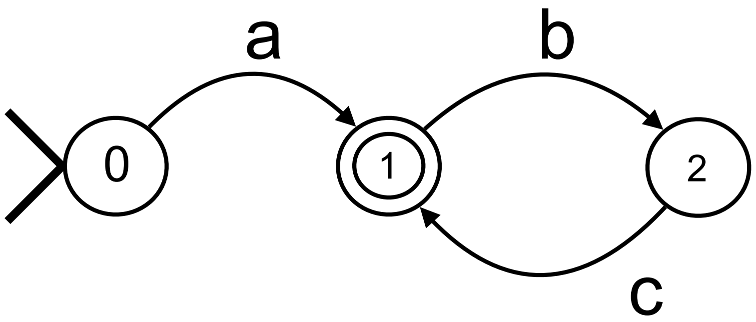


$$L_0 = \varepsilon$$

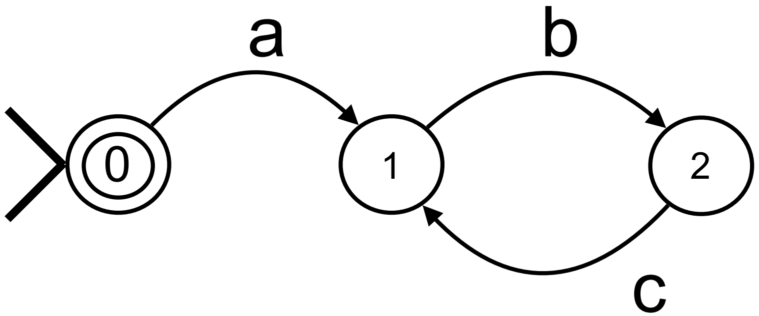


$$L_2 = L_1\, b$$

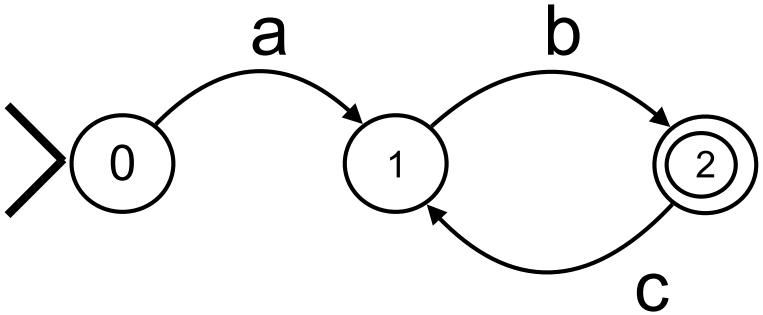
Is there a regular expression for every FSM?



$$L_1 = L_0\, a \mid L_2\, c$$
$$= a \mid L_1\, bc$$
$$L_1 = a(bc)^*$$



$$L_0 = \varepsilon$$



$$L_2 = L_1\, b$$

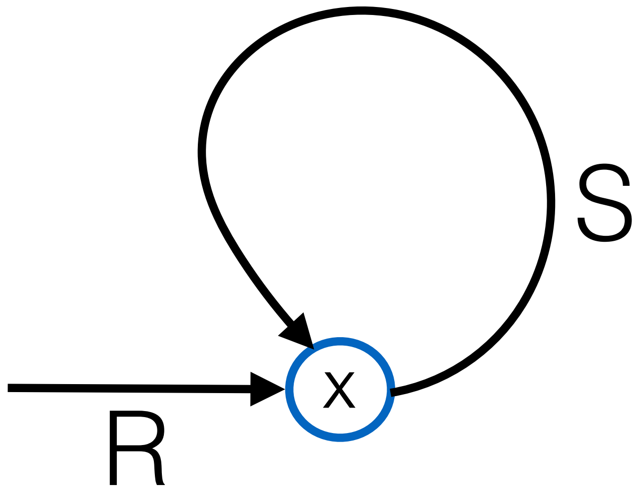
Arden's Lemma

If R and S are regular expressions then the equation

$X = R \mid X\, S$

has a solution $X = R\, S^*$

If $\varepsilon \notin L(S)$ then this solution is unique.

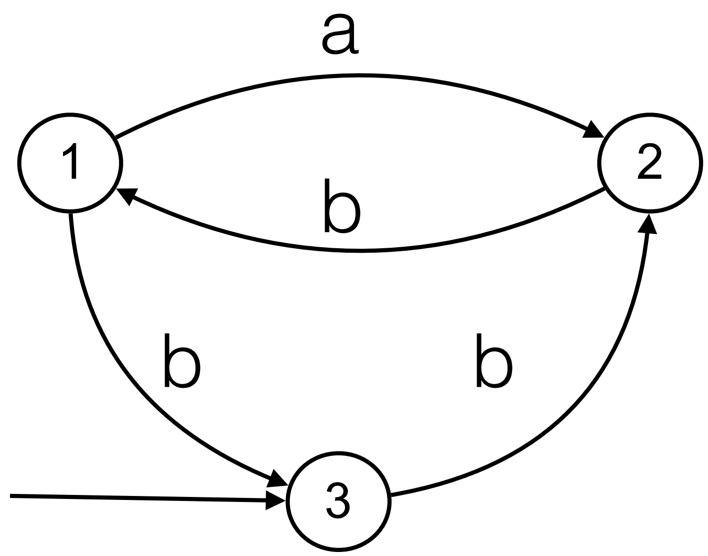


Is there a regular expression for every FSM?

$$L_1 = L_2\, b$$

$$L_2 = L_3\, b \mid L_1\, a$$

$$L_3 = \varepsilon \mid L_1\, b$$

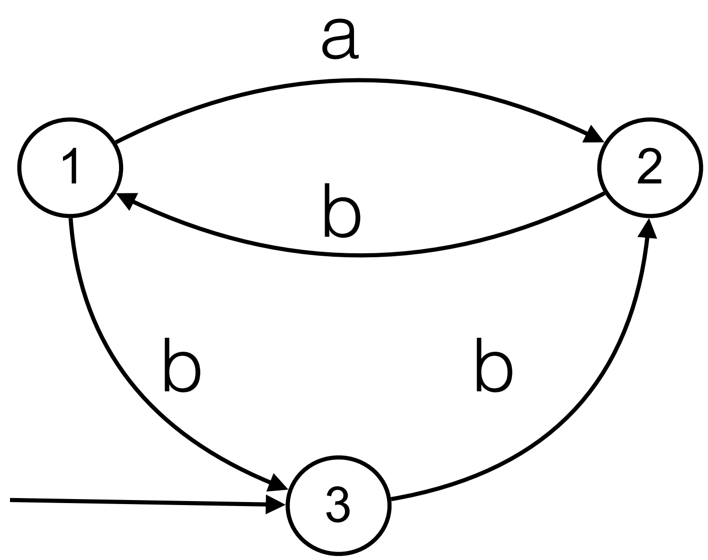


Is there a regular expression for every FSM?

$$L_1 = L_2\, b$$

$$L_2 = L_3\, b \mid L_1\, a$$

$$L_3 = \varepsilon \mid L_1\, b$$
$$= \varepsilon \mid L_2\, b\, b$$



$$L_2 = (\varepsilon \mid L_2\, b\, b)\, b \mid L_2\, b\, a$$
$$= b \mid L_2\, b\, b\, b \mid L_2\, b\, a$$
$$= b \mid L_2\, (b\, b\, b \mid b\, a)$$

Arden's Lemma

If R and S are regular expressions then the equation

$$X = R \mid X\, S$$

has a solution $X = R\, S^*$

If $\varepsilon \notin L(S)$ then this solution is unique.

$L_2 = b \mid L_2\,(b\,b\,b \mid b\,a)$

$L_2 = b\,(b\,b\,b \mid b\,a)^*$

Language

Σ: a finite alphabet

A **language** L is a set of finite strings

$$L \subseteq \Sigma^*$$

where the **strings** in Σ^* are of
finite sequences of tokens from Σ
the string $< x_0, \ldots, x_{n-1} >$ has length n
strings include the empty string
$\varepsilon = <>$ of length 0

Finite Automata

finite alphabet $a, b \in \boldsymbol{\Sigma}$; $\boldsymbol{\Sigma}^+ = \boldsymbol{\Sigma} \cup \{\boldsymbol{\varepsilon}\}$

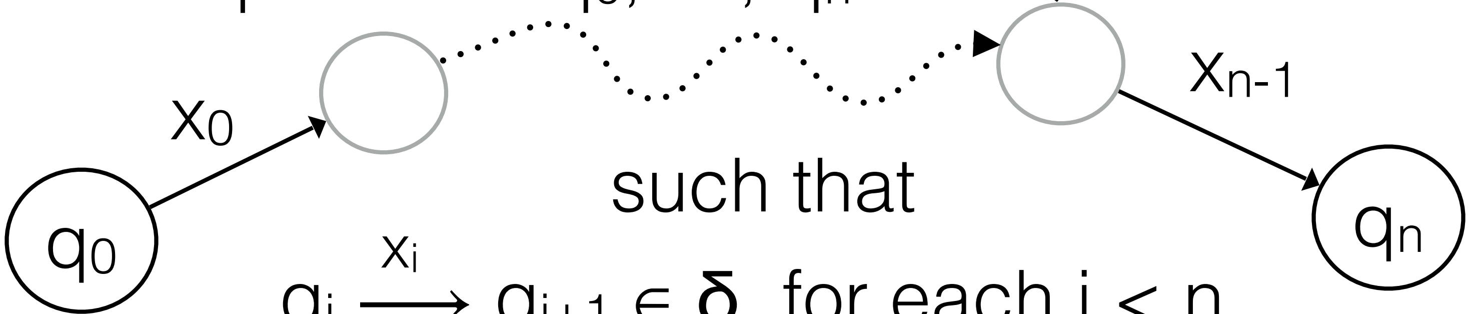
finite set of states $A, B \in \mathbf{Q}$

start states $\mathbf{S} \subseteq \mathbf{Q}$ and final states $\mathbf{F} \subseteq \mathbf{Q}$

labelled transitions $A \xrightarrow{a} B \in \boldsymbol{\delta} \subseteq \mathbf{Q} \times \boldsymbol{\Sigma}^+ \times \mathbf{Q}$

A **trace** $q_0 \overset{s}{\rightsquigarrow} q_n$ for $\mathbf{s} \in \boldsymbol{\Sigma}^*$ in M is a

sequence $< q_0, \ldots, q_n > \in \mathbf{Q}^*$ of states



such that

$q_i \xrightarrow{x_i} q_{i+1} \in \boldsymbol{\delta}$, for each $i < n$,

and **s** is the concatenation of the x_i (with $\boldsymbol{\varepsilon}$ = "")

Finite Automaton, M

finite alphabet a, b $\in$ **Σ** ; Σ^{+} = **Σ** $\cup$ {ε}

finite set of states A, B $\in$ **Q**

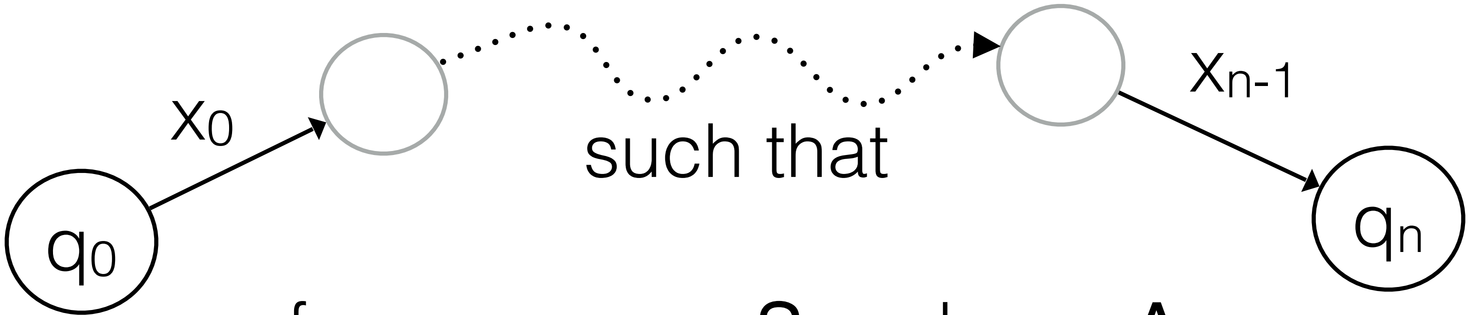
start states **S** $\subseteq$ **Q** and final states **F** $\subseteq$ **Q**

labelled transitions $A \xrightarrow{a} B \in \mathbf{Q} \times \mathbf{\Sigma} \cup \{\varepsilon\} \times \mathbf{Q}$

The **language accepted by M** is the set of strings **s** for which

there is a trace $q_0 \overset{s}{\rightsquigarrow} q_n$

such that



for some $q_0 \in$ **S** and $q_n \in$ **A**

Finite Automata

finite alphabet a, b $\in$ **Σ** ; **Σ**$^+$ = **Σ** $\cup$ {**ε**}

finite set of states A, B $\in$ **Q**

start states **S** $\subseteq$ **Q** and final states **F** $\subseteq$ **Q**

labelled transitions $A \xrightarrow{a} B \in \mathbf{Q} \times \mathbf{\Sigma} \cup \{\boldsymbol{\varepsilon}\} \times \mathbf{Q}$

DFA

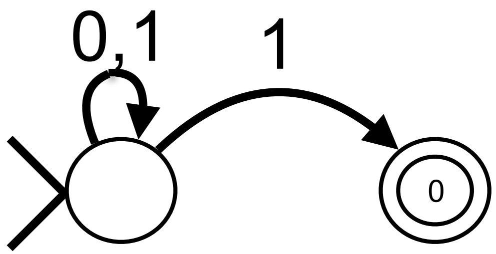
- a single start state
- exactly one **a**-labelled transition from each state for each symbol **a** $\in$ **Σ**
- no **ε**-transitions.

NFA

- no restrictions

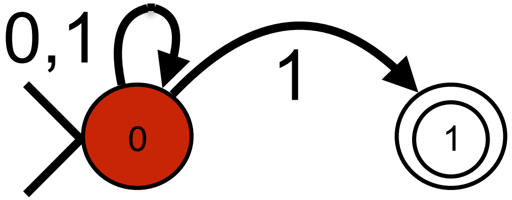
Are there any languages recognised by some NFA, but by no DFA?

Try a simple example
binary strings
that end in 1

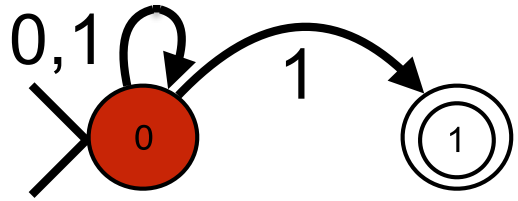


Each state X lights up when we've seen a string
with a trace from some start set to X

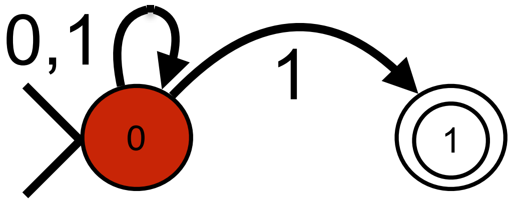
ε



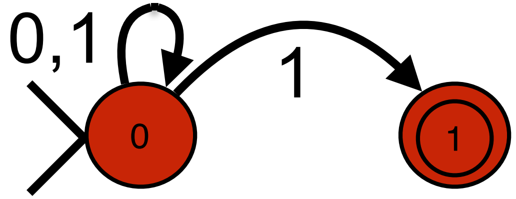
10



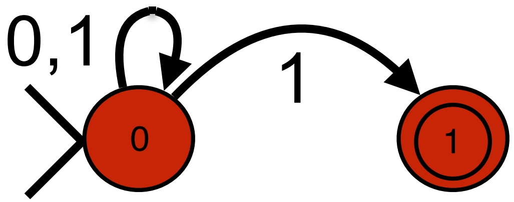
0



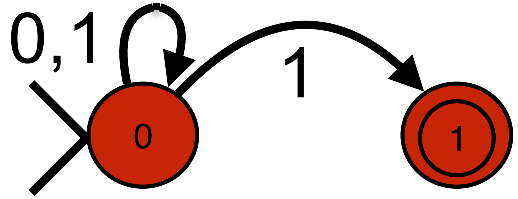
01



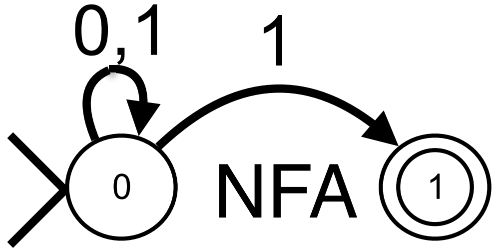
1

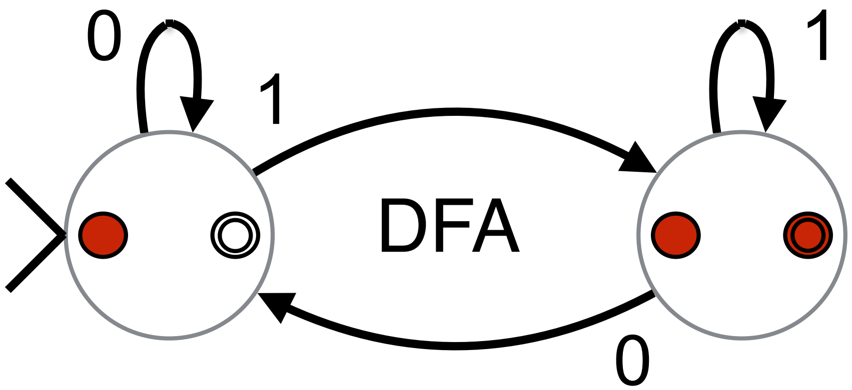


11



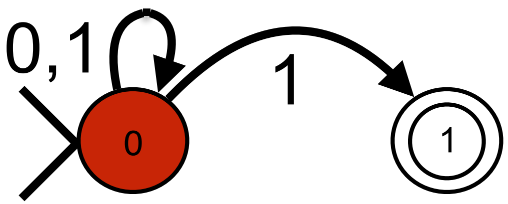
Each state X lights up when we've seen
a string with a trace from some start set to X



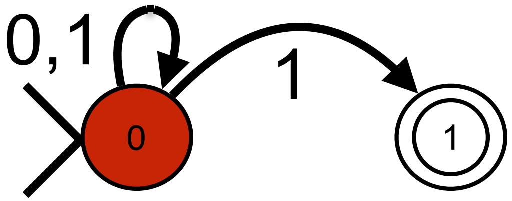


The states of the DFA
are sets of
states of the NFA.
subset construction

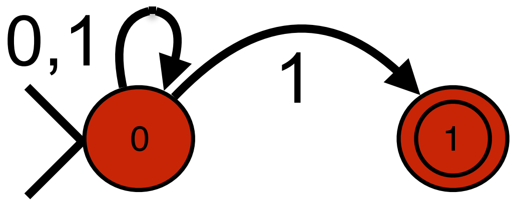
ε



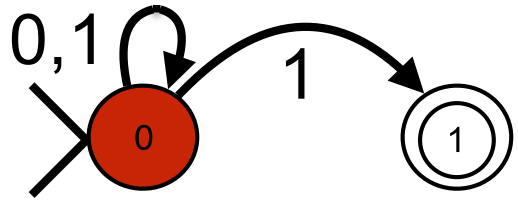
0



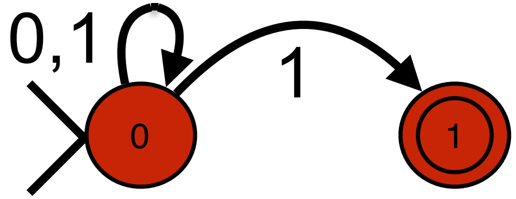
1



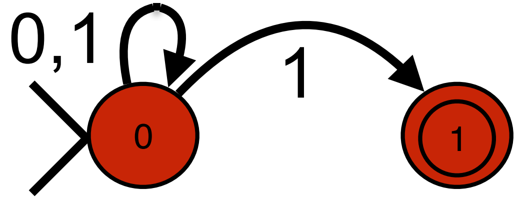
10



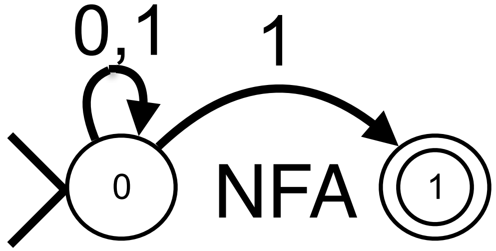
01

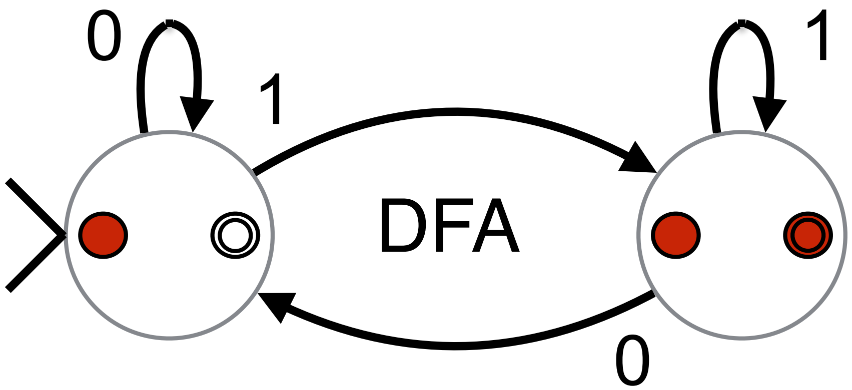


11



Each state X lights up when we've seen
a string with a trace from some start set to X





The states of the DFA **M**
are sets of
states of the NFA $\mathcal{P}$(**M**).
subset construction

The **start state** of $\mathcal{P}$(**M**) is the set of start states of **M**.

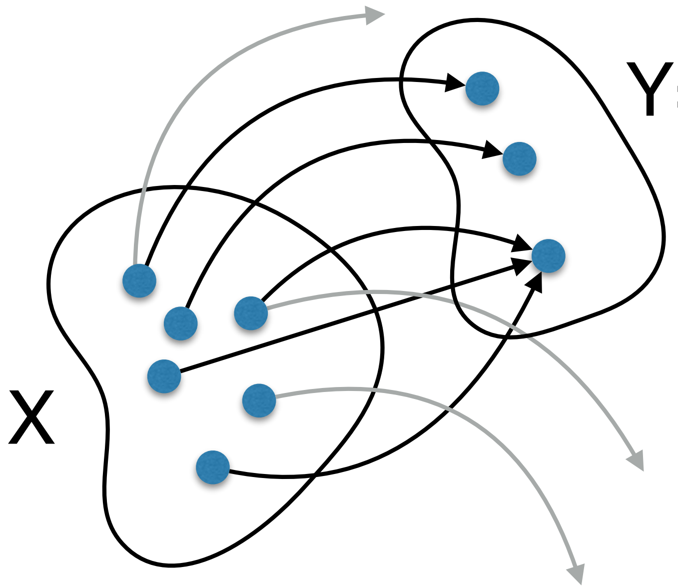
X ⊆ M is an **accepting state** of $\mathcal{P}$(**M**)
iff there is an accepting state A of **M** with A∈ **X**

In general not all subsets of **M**
are reachable from the start state
we can ignore those any are not reachable.

In this example, we have no ε-transitions.

The **start state** of $\mathcal{P}$(M) is the set **S** of start states of **M**.

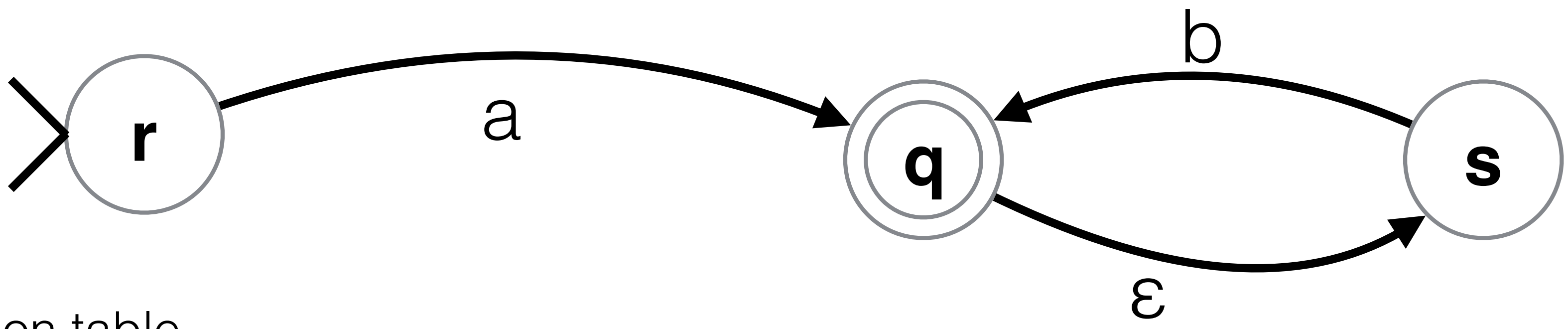
For any reachable **X** ⊆ **M**, for each a ∈ Σ,
the a-labelled transition from **X** leads to
the set **Y** of states reachable in **M**
from a state X in **X** by an a-transition.



$\mathbf{Y} = \{\, Y \mid \text{for some } X \in \mathbf{X},\ X \xrightarrow{a} Y \,\}$

$\mathbf{X} \xrightarrow{a} \mathbf{Y}$ in $\mathcal{P}$(M)
and **Y** is reachable.

What if we do have an ε-transition?



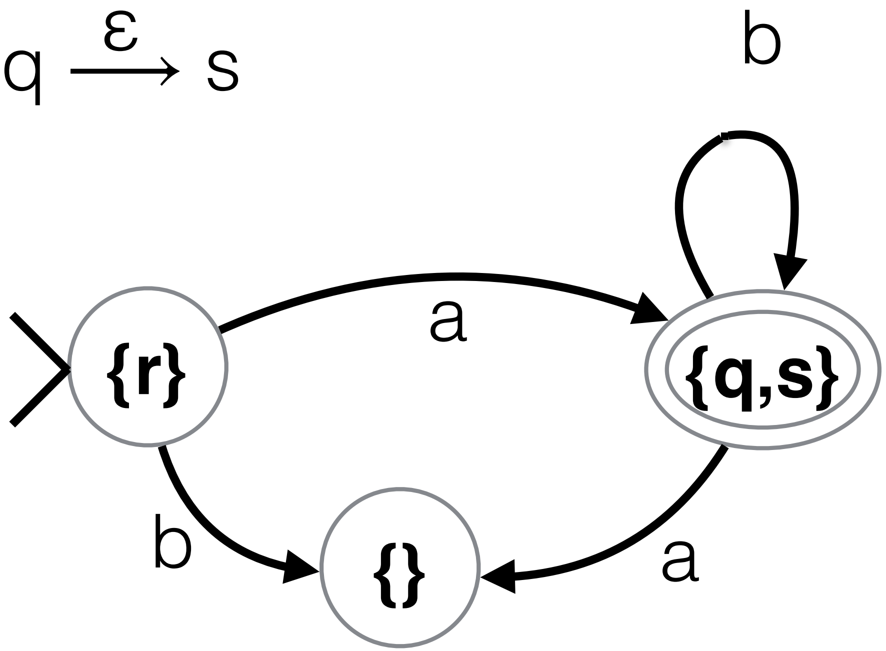
transition table

M	a	b	ε
r	q		
q			s
s		q	

We have two traces for “a”.

$$r \xrightarrow{a} q \quad \text{and } r \xrightarrow{a} q \xrightarrow{\varepsilon} s$$

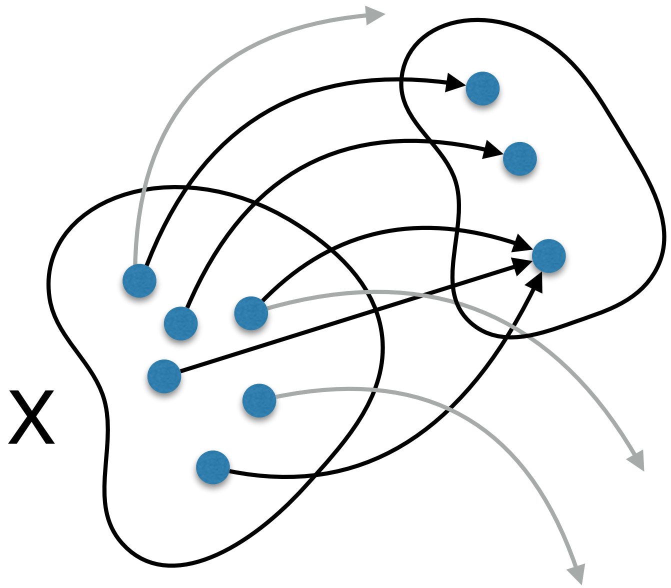
$\mathcal{P}$(M)	a	b
{r}	{q,s}	{}
{q,s}	{}	{q,s}



What if we do have some ε-transitions?

The **start state** of $\mathcal{P}$(M) is the set of states t such that s $\overset{\varepsilon}{\rightsquigarrow}$ t where s ∈ **S** is a start state of **M**.

For any reachable **X** ⊆ **M**, for each a ∈ Σ, the a-labelled transition from **X** leads to the set **Y** of states reachable in **M** from a state X in **X** by an a-**trace**.

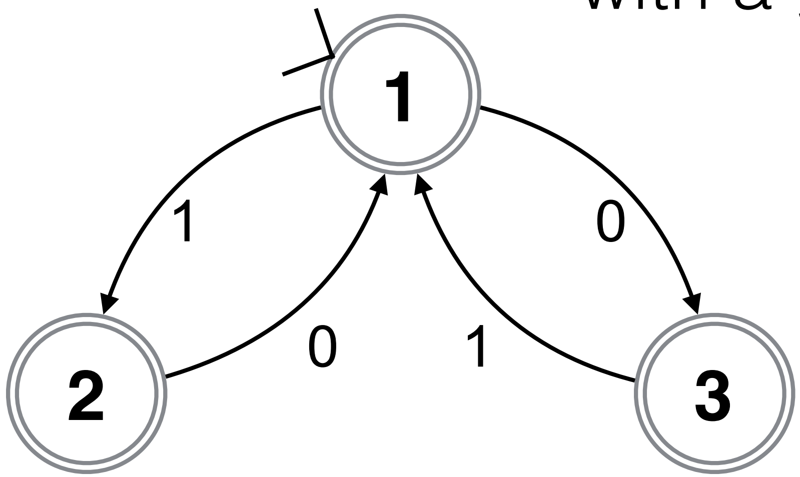


Y= { Y | **for some** X∈ **X**, X $\overset{a}{\rightsquigarrow}$ Y}

X $\xrightarrow{a}$ **Y** in $\mathcal{P}$(M)
and **Y** is reachable.

number of 1’s and
number of 0’s
are the same

A machine with **at most one** transition
with a given label from a given state



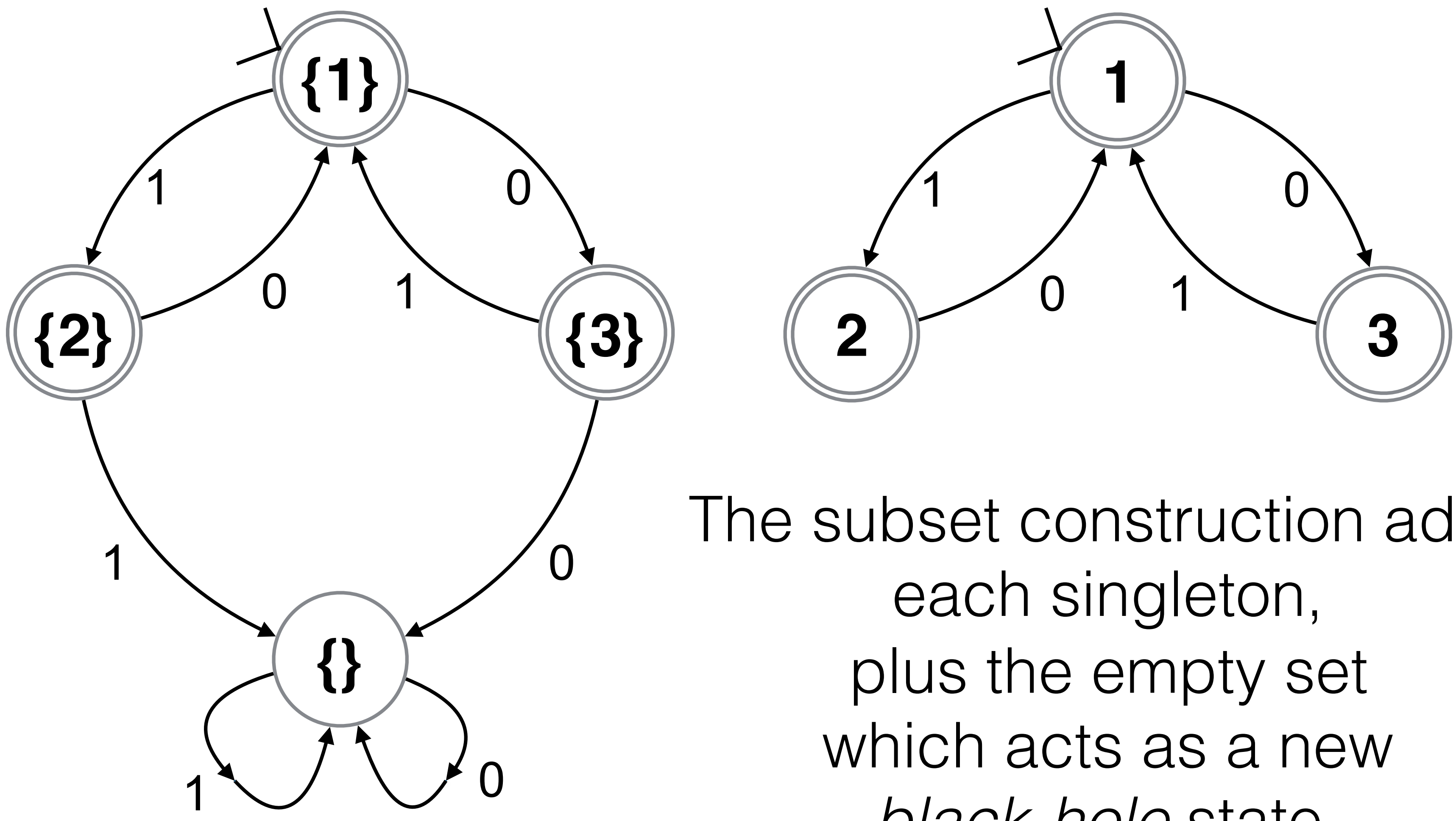
This machine is not a DFA
but it is equivalent to a DFA

number of 1’s is
one larger than
number of 0’s

number of 0’s is
one larger than
number of 1’s

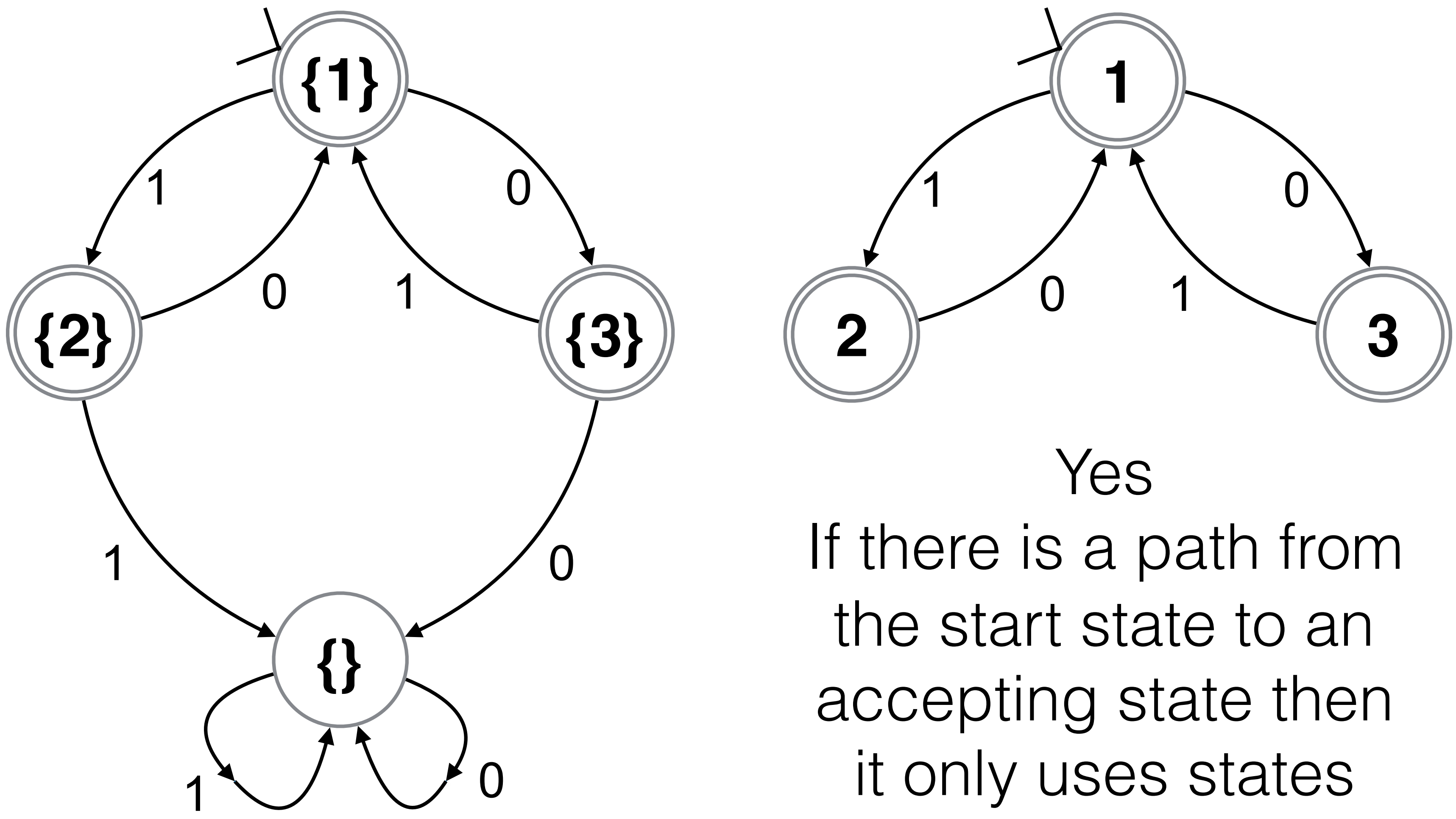
What is the DFA
given by the subset construction?

Are these two machines equivalent?



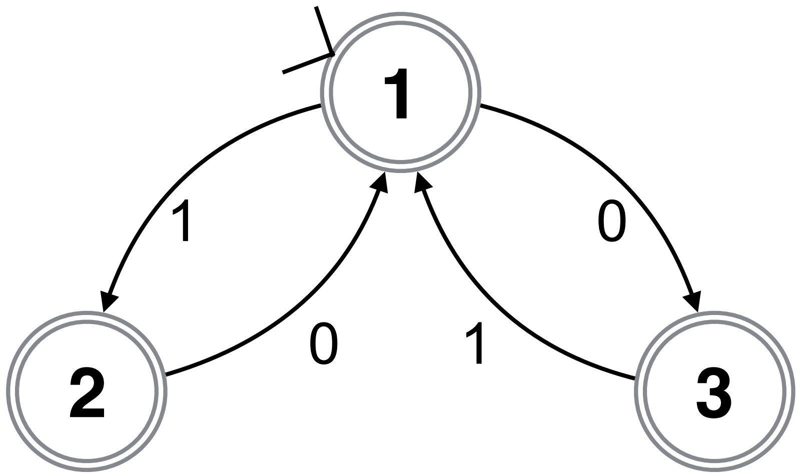
The subset construction adds
each singleton,
plus the empty set
which acts as a new
black-hole state,
which is not an accepting state
any missing transitions go there.

Are these two machines equivalent?



Yes
If there is a path from the start state to an accepting state then it only uses states 1, 2, 3

The two machines accept the same strings



A machine with
at most one transition
with a given label from a given state

If a machine has at most one transition
with a given label from any state
then it has at most one trace for any input string

The equivalent DFA produced by the subset construction
has one new state, a *black-hole* state,
which is not an accepting state.
All missing transitions go there.

Deterministic FSMs

Many authors give an informal definition of deterministic

- each state has at most one transition leaving the state for each input symbol.

Formal definition says, exactly one state …

- We consider the informal presentation to include an implicit “black hole”, or “sink” state, from which there is no escape.
- Where there is no explicit transition for a symbol, it takes us to the black hole.

Determinism

If we have a machine with at most one transition for each (q,s) pair, we can always convert to an equivalent DFA for which every state has exactly one transition leaving the state for each input symbol.

For the new machine there is exactly one trace for each input string

- Proof

  Add a new "black hole" state, ●

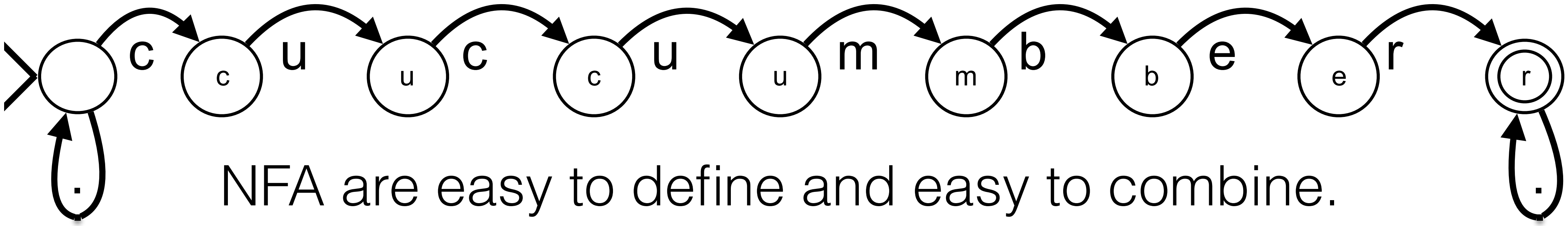
  For every pair (q, s) for which there is no state r with a transition T(q, s, r), add a transition T(q, s, ●).

  This includes a transition T(●, a, ●) for each $a \in \Sigma$. You cannot escape from the black hole.

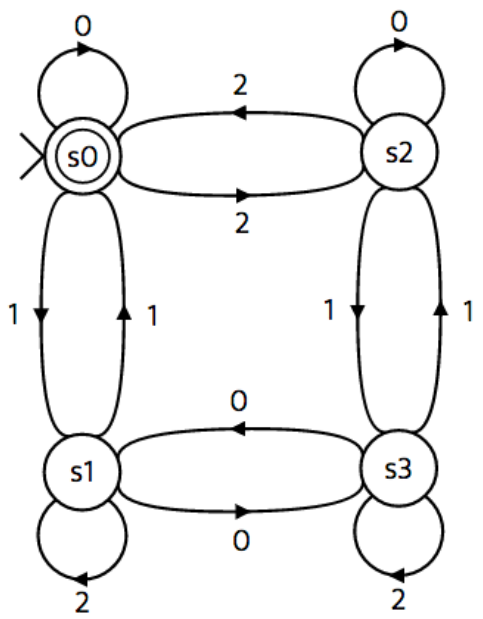
  The black hole ● is not an accepting state.

This machine accepts the same language as the original.

searching for cucumber



NFA are easy to define and easy to combine.
DFA are easy to implement




```
s0 []        = True
s0 ('0':xs)  = s0 xs
s0 ('1':xs)  = s1 xs
s0 ('2':xs)  = s2 xs

s1 []        = False
s1 ('0':xs)  = s3 xs
s1 ('1':xs)  = s0 xs
s1 ('2':xs)  = s1 xs

s2 []        = False
s2 ('0':xs)  = s2 xs
s2 ('1':xs)  = s3 xs
s2 ('2':xs)  = s0 xs

s3 []        = False
s3 ('0':xs)  = s1 xs
s3 ('1':xs)  = s2 xs
s3 ('2':xs)  = s3 xs
```