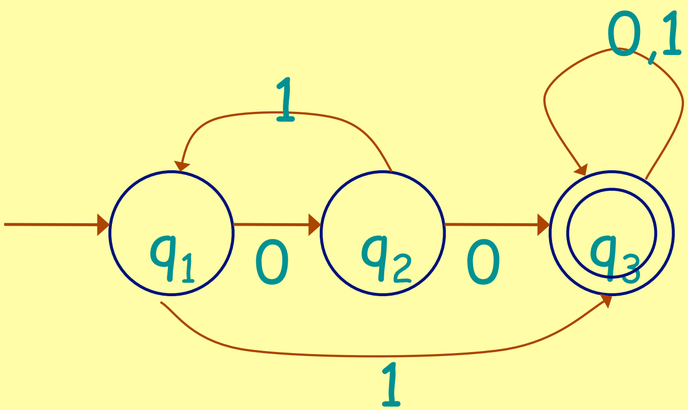
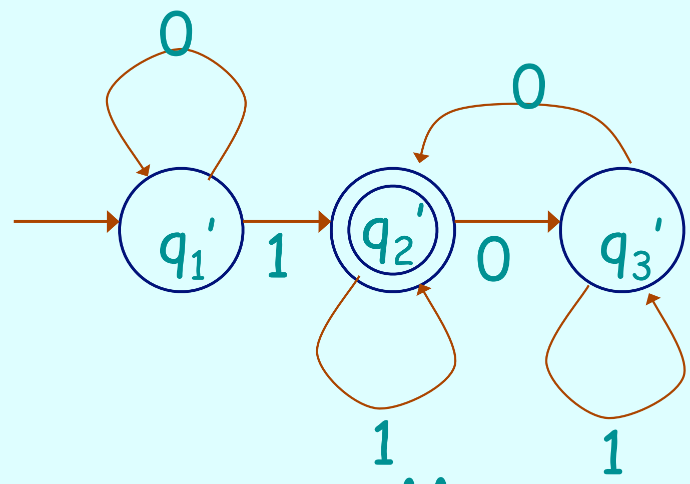
Introduction to the Theory of Computation

Set 3 — Regular Languages (2)

Example

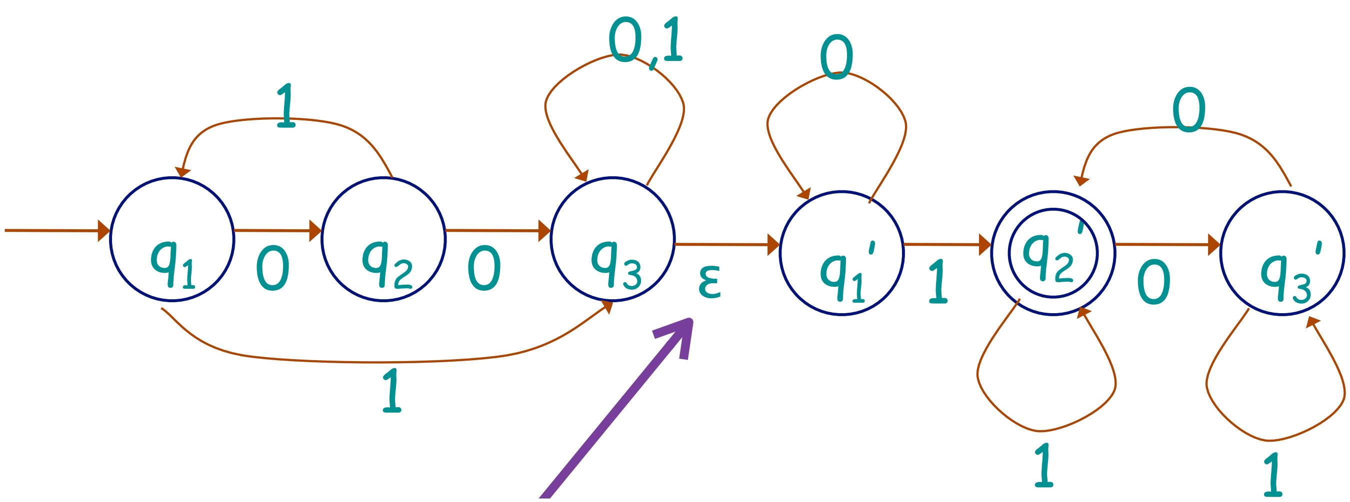


M_1



M_2

Find M such that $L(M) = L(M_1) \bullet L(M_2)$



Nondeterministic Finite Automaton (NFA)

DFA *vs.* NFA

Deterministic FA	Nondeterministic FA
DFA	NFA

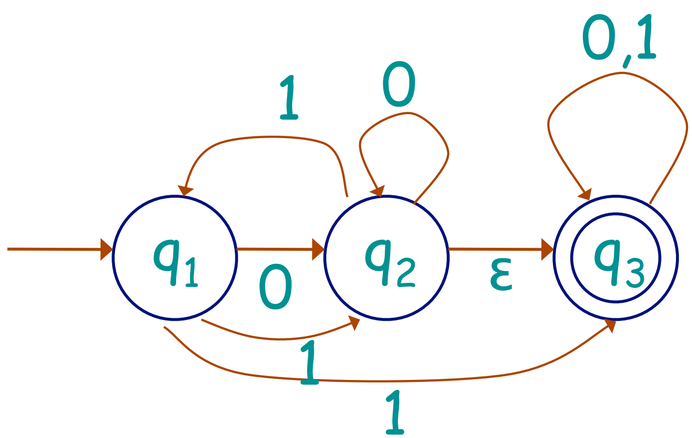
Computing on an NFA

For each symbol of the string, keep track of **all possible transitions** in parallel

When input ends, there may be several possible ending states

- If **at least one** of the possibilities is an <u>accepting state</u>, then the NFA <u>accepts</u> the string

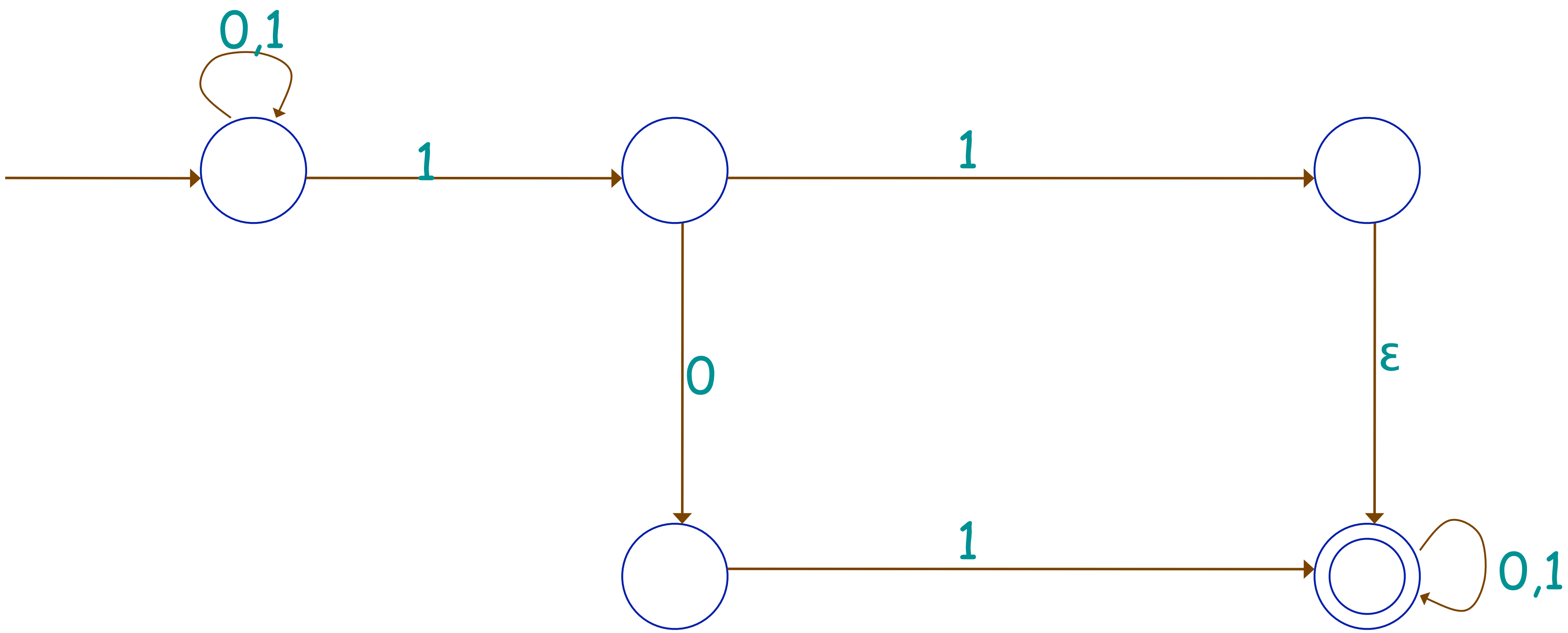
Example



Does this NFA accept the string 001?

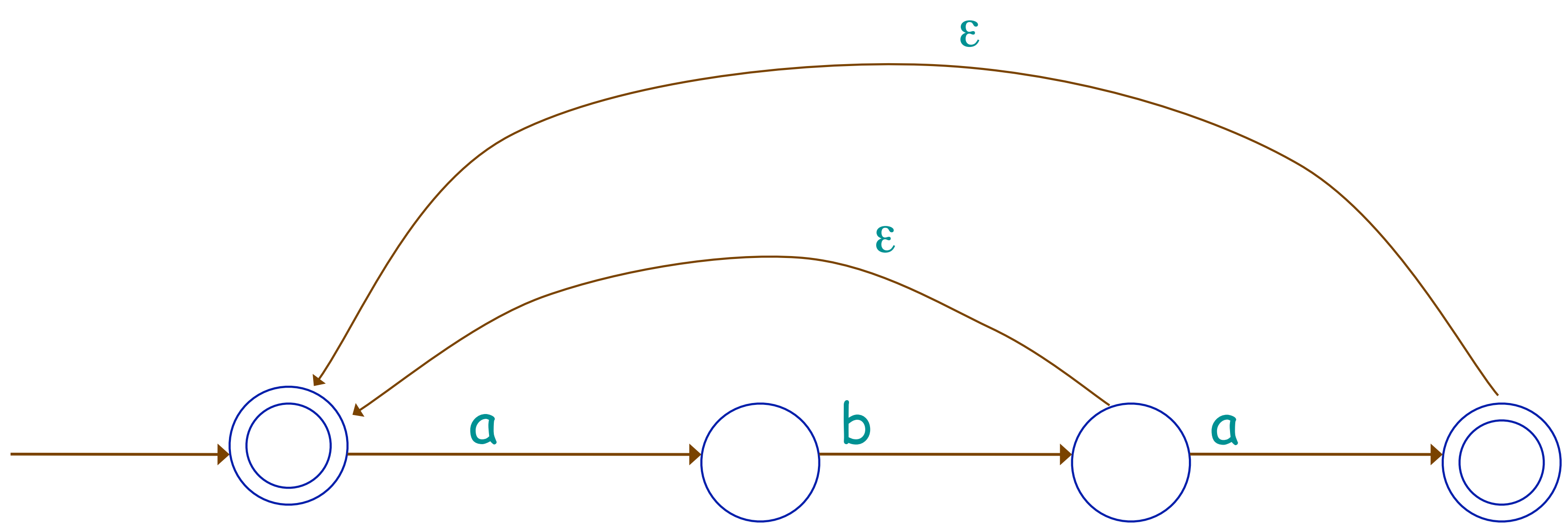
From state(s)	Input	To state(s)

Example



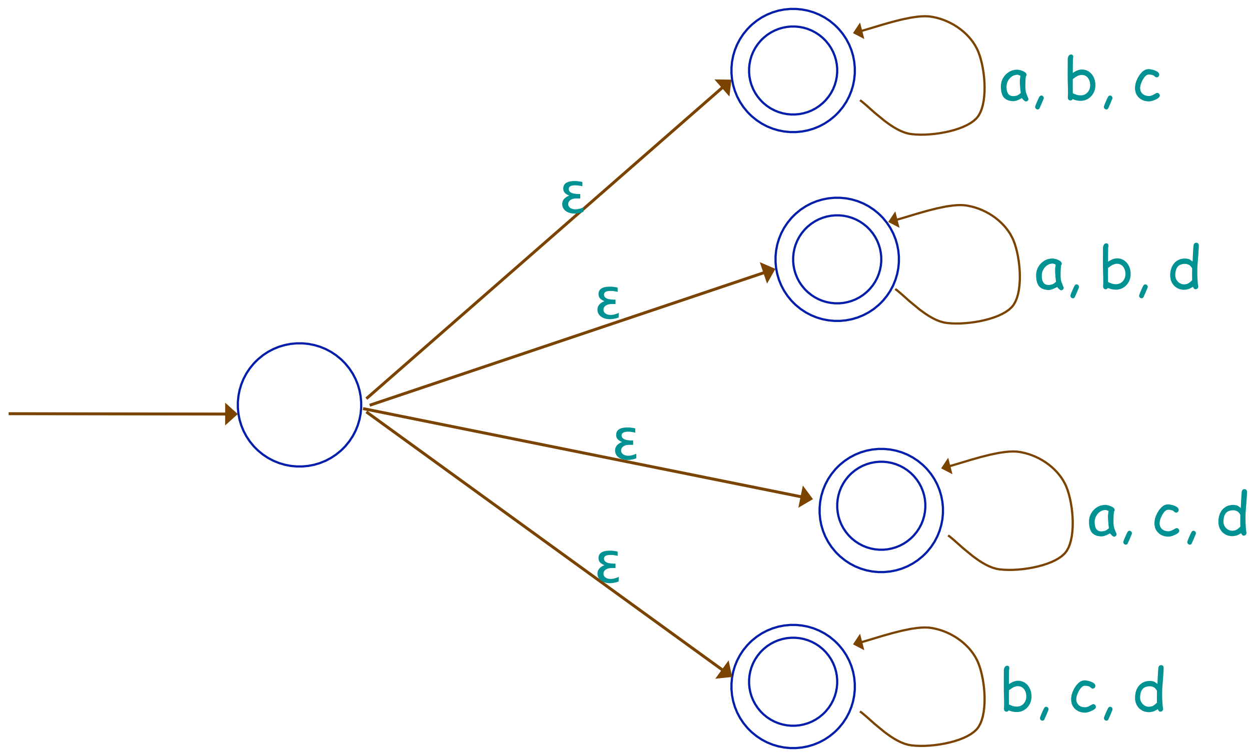
Any string containing two 1's separated by at most one 0

Example



Zero or more concatenations of the strings aba and ab

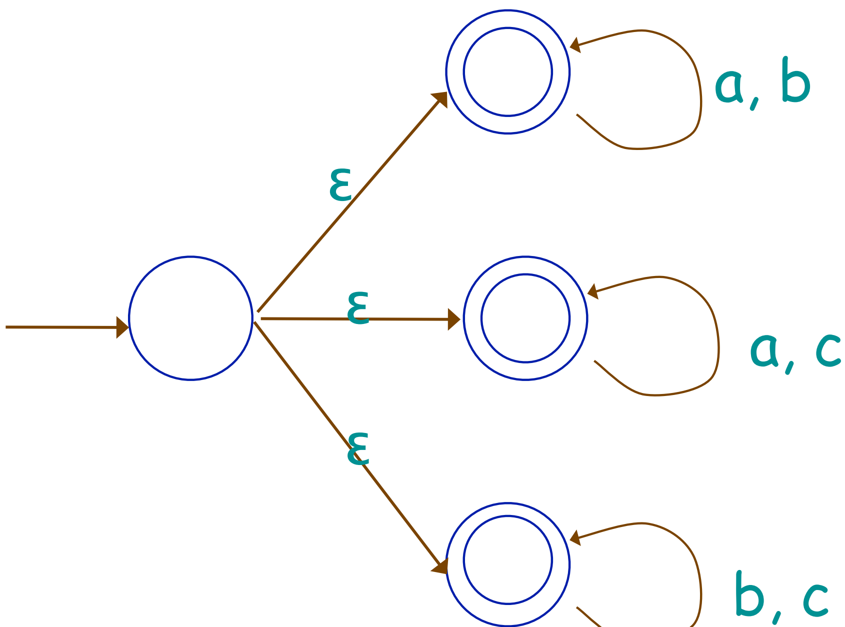
Example



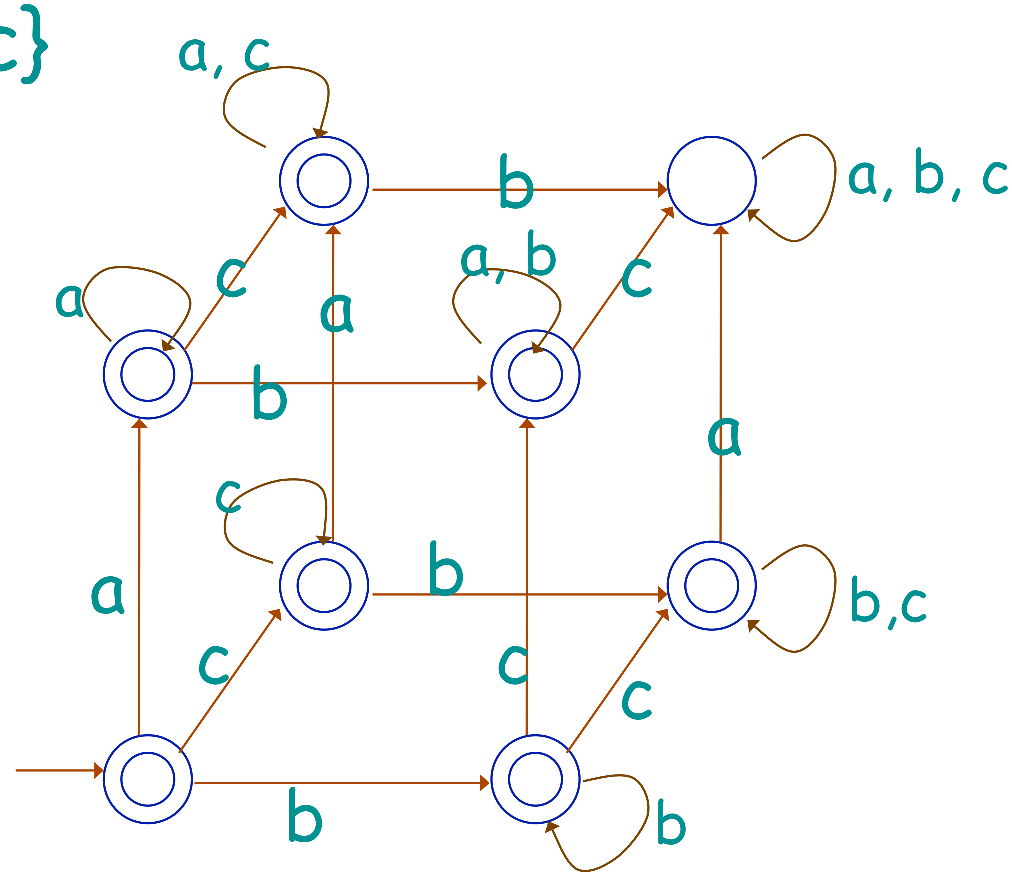
Strings of any length containing at most three of the symbols in Σ = {a, b, c, d}

The Utility of Non-Determinism

Strings of any length containing at most two of the symbols in $\Sigma = \{a, b, c\}$



NFA



DFA

Nondeterministic Finite Automaton (NFA)

[Formal Definition]

A nondeterministic finite automaton is a 5-tuple (Q,Σ,δ,q_0,F)**, where**

- Q **is a finite set of states**
- Σ **is a (finite) alphabet**
- $\delta : Q \times \boxed{\Sigma_\varepsilon} \to \mathcal{P}(Q)$ **is the transition function**
  - δ **maps to *sets* of states**
- q_0 **is the start state, and**
- $F \subseteq Q$ **is the set of accept states**

Equivalence of DFAs and NFAs

Theorem: Every nondeterministic finite automaton has an equivalent deterministic finite automaton

- Both FAs accept the same language

Proof method

- Construction
- Similar to method used for recognizing strings
  - Follow all paths in parallel where states represent parallel paths

Proof Idea

Given NFA $M_1=(Q,\Sigma,\delta,q_0,F)$ construct DFA $M_2=(Q',\Sigma,\delta',q_0',F')$ with $L(M_1)=L(M_2)$

Intuition

- **Recall** $\delta: Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$
- **Our DFA's transition function will generate paths within** $\mathcal{P}(Q)$

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

Defining M_2

Determine Q', q_0', and F'

- Q' = $\mathcal{P}$(Q)
- q_0' = {q_0}
- F' = {R $\in$ Q' | R $\cap$ F $\neq$ $\emptyset$}

R contains at least one of M_1's accept states

Defining δ'

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

(ignoring ε jumps for now)

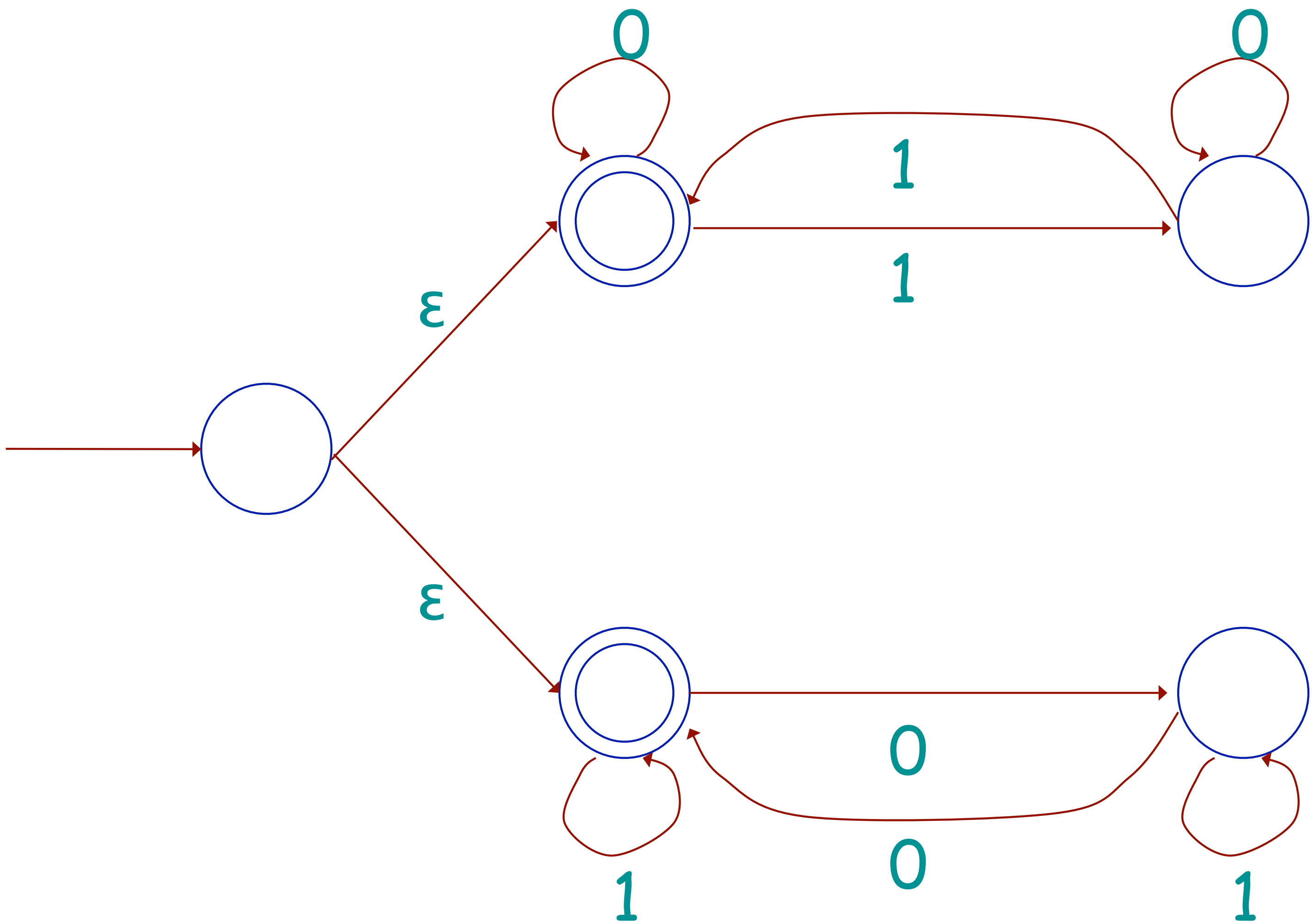
Collaborative Exercise

For each NFA

- Informally describe the behavior of the NFA
- Construct a DFA accepting the same language

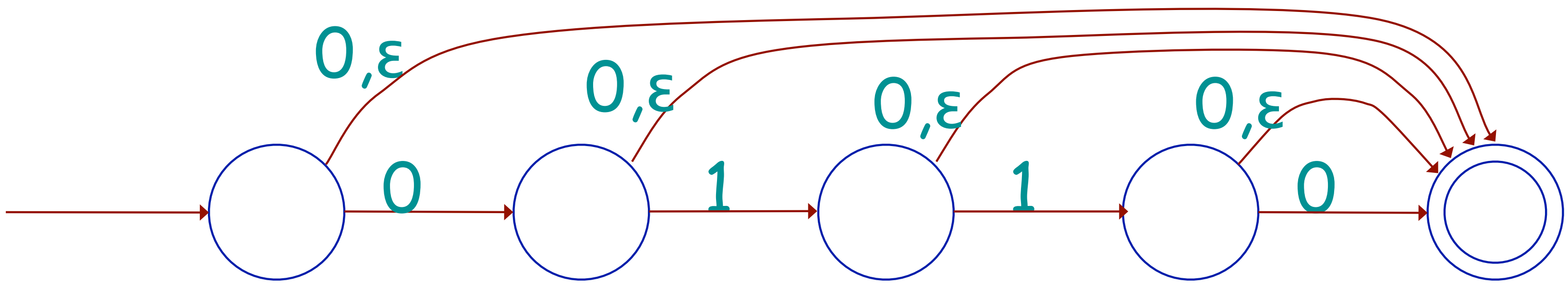
NFA 1

$\Sigma = \{0,1\}$



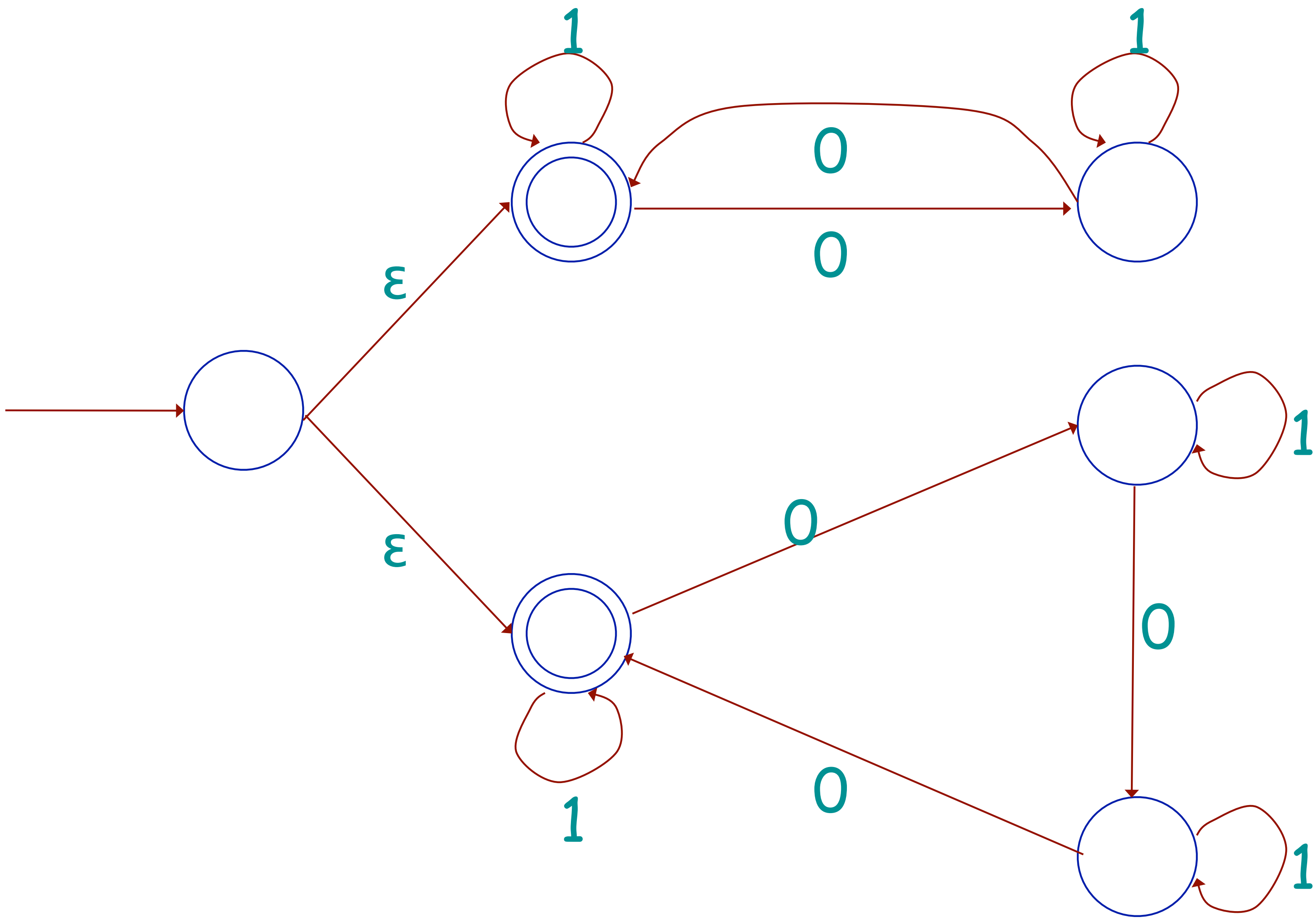
NFA 2

$\Sigma = \{0,1\}$



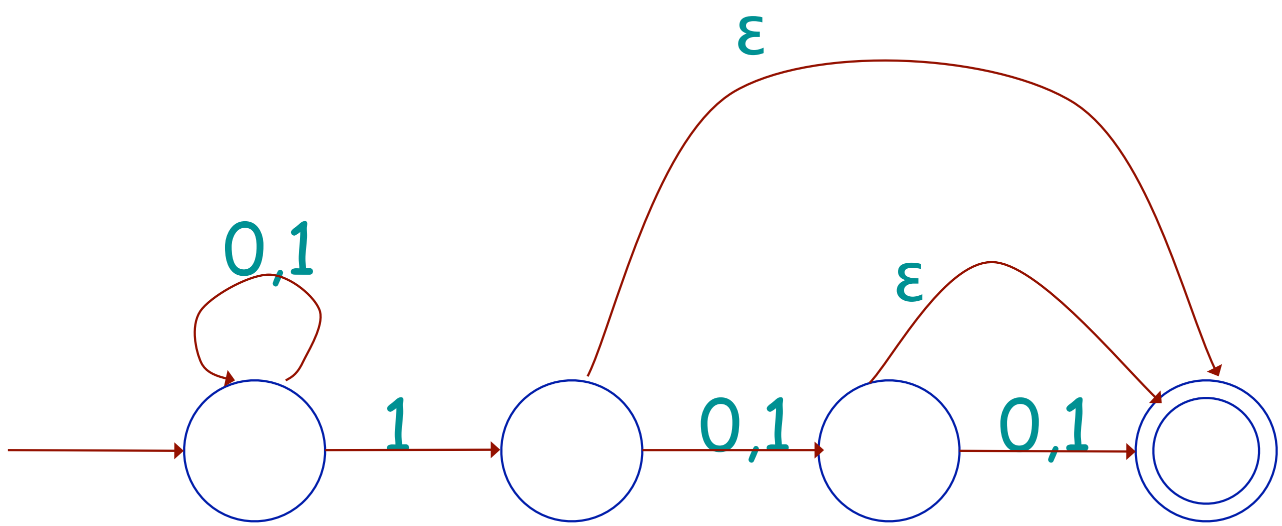
NFA 3

$\Sigma = \{0,1\}$



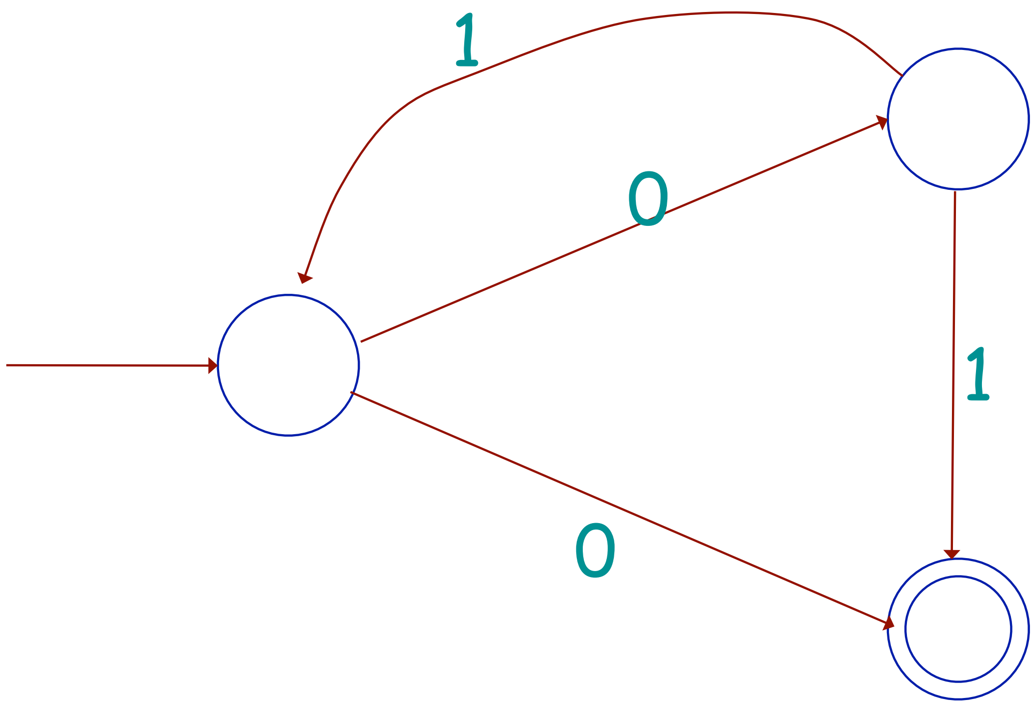
NFA 4

$\Sigma = \{0,1\}$



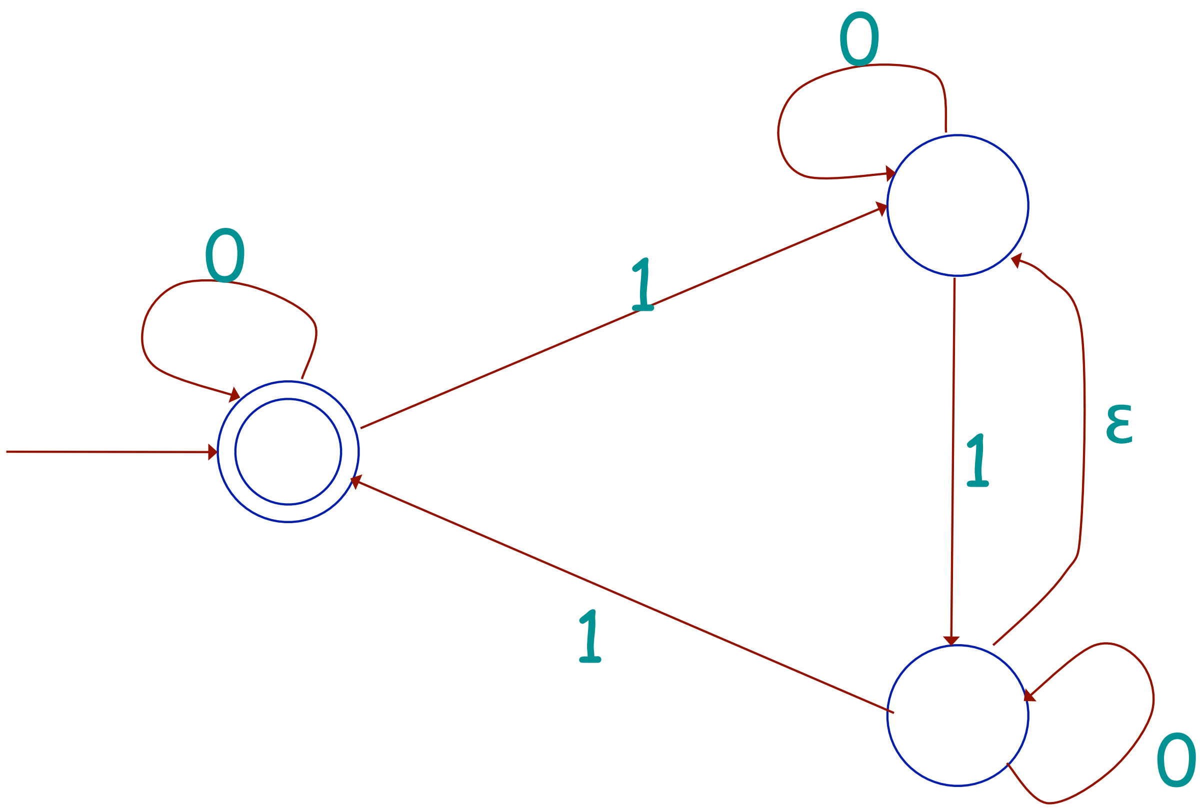
NFA 5

$\Sigma = \{0,1\}$



NFA 6

$\Sigma = \{0,1\}$



Closure of NFA's Under Regular Operations

Recall the following are the regular operators

- Union
- Concatenation
- Kleene star

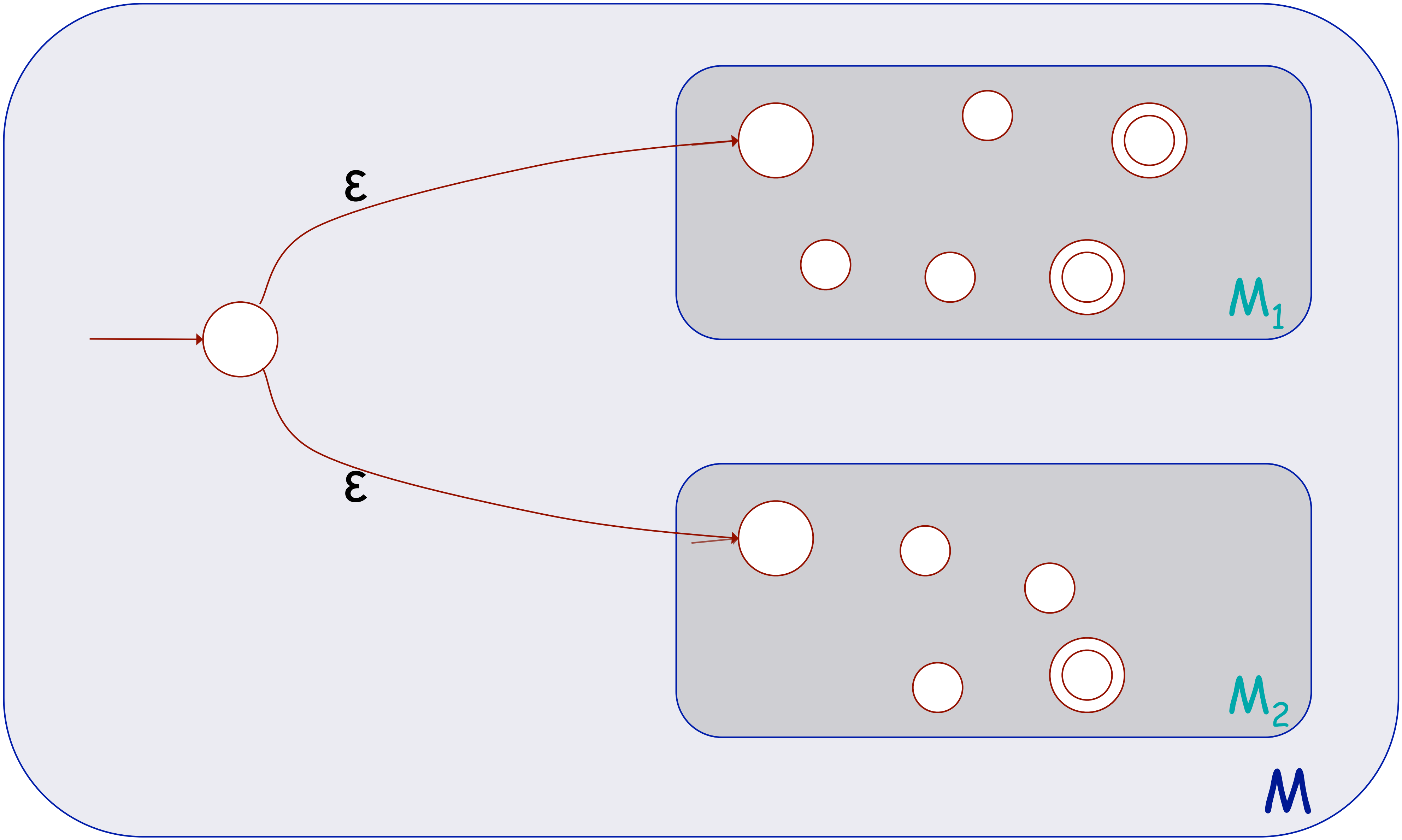
Union is a Regular Operation

Theorem: The class of regular languages is closed under the union operation

Proof approach: Assume A_1 and A_2 are both regular languages with $A_1 = L(M_1)$ and $A_2 = L(M_2)$ then create an NFA M such that $L(M) = A_1 \cup A_2$

Method: Proof by construction

Construct M from M_1 and M_2



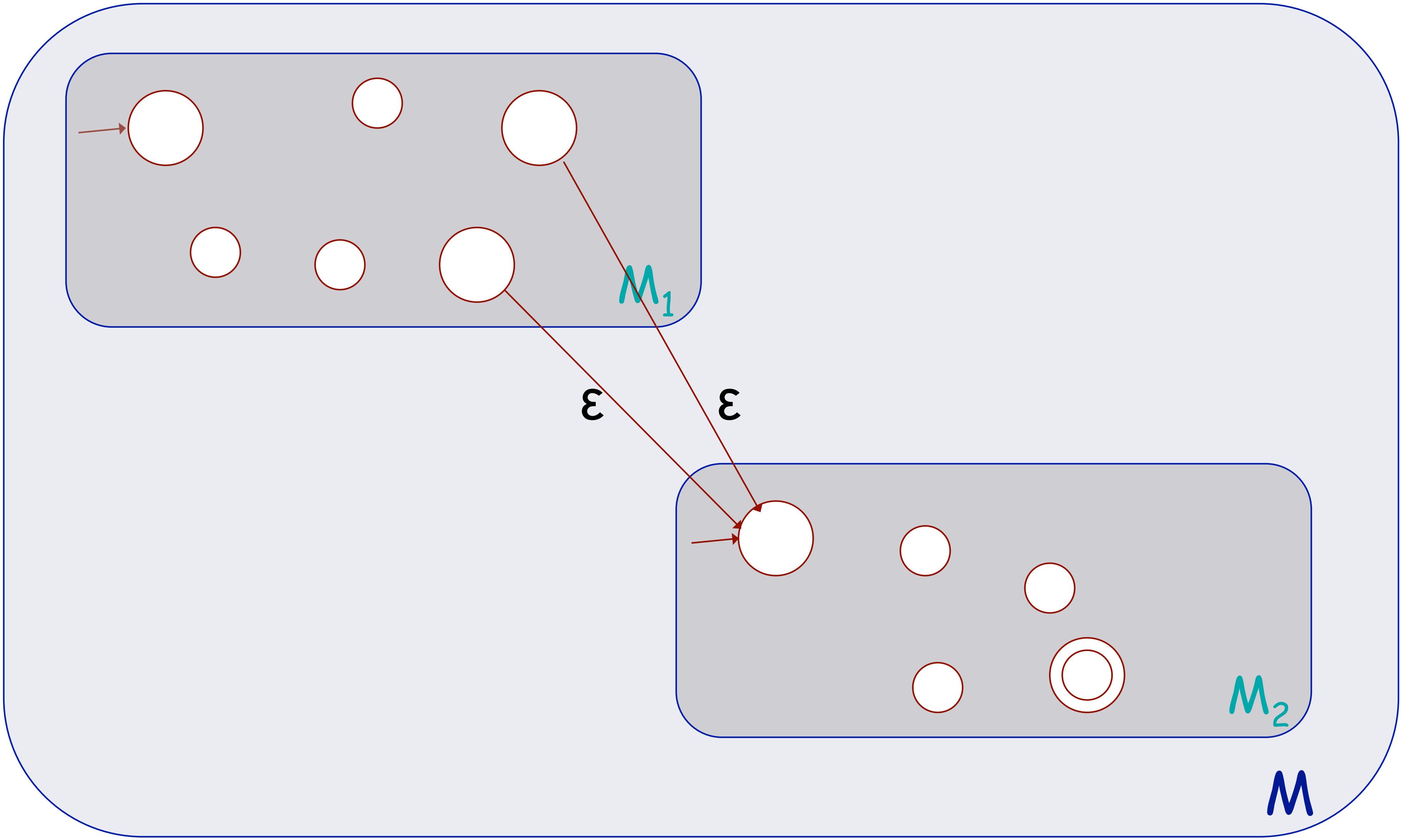
Concatenation is a regular operation

Theorem: The class of regular languages is closed under the concatenation operation

Proof approach: Assume A_1 and A_2 are both regular languages with $A_1=L(M_1)$ and $A_2=L(M_2)$ then create an NFA M such that $L(M) = A_1 \cdot A_2$

Method: Proof by construction

Construct M from M_1 and M_2



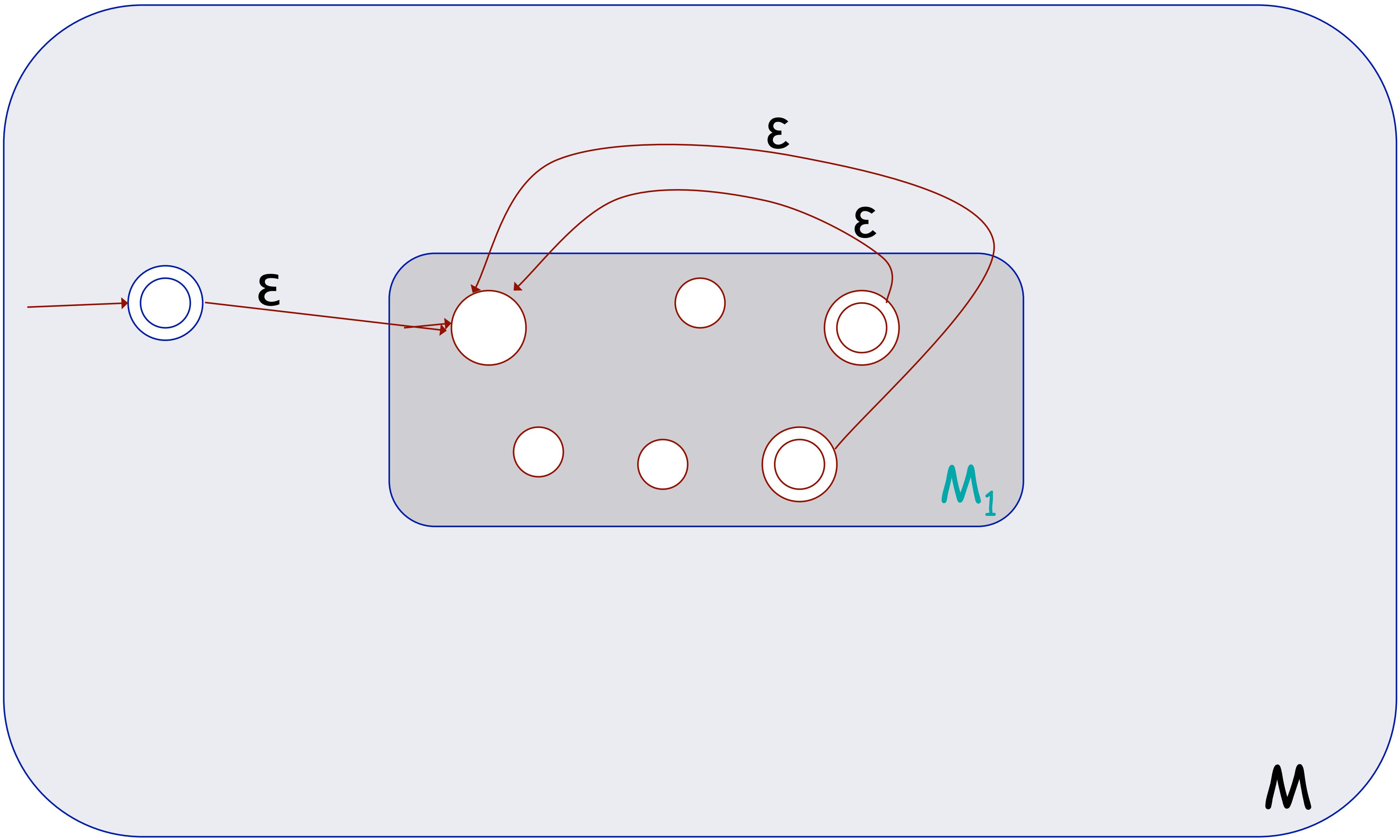
Kleene Star Is a Regular Operation

Theorem: The class of regular languages is closed under the Kleene star operation

Proof approach: Assume A_1 is a regular language with $A_1=L(M_1)$ and create an NFA M such that $L(M) = A_1*$

Method: Proof by construction

Construct M from M_1



Regular Languages

So far we have had to describe languages either with finite automata or with words

- Potentially clumsy or imprecise

Two other formal expressions that describe regular languages

  - Regular Grammars
  - Regular Expressions

Formal Grammars

Formal grammars contain a set of production rules for strings in a language.

The rules describe how to form valid strings under the language's alphabet.

Example: $\{S \rightarrow abA,\ S \rightarrow a,\ A \rightarrow aA,\ A \rightarrow b\}$

Formal Grammars

A formal grammar includes a set of **rules** for rewriting strings and a specified **start symbol** from which rewriting starts.

The set of **variables**, also known as **non-terminals**, will be written using capital letters to be consistent with JFLAP.

One of the variables is distinguished as the ***start symbol***.

The members of the **alphabet Σ** are also known as **terminals**.

Formal Grammars

Example:

Σ = {a, b} *(a.k.a* ***terminals****)*

variables = {S, A, B} *(a.k.a* ***non-terminals****)*

start symbol = S

rules = {S→abA, S→a, A→aA, A→b}

A derivation of the string **abaab:**

S → abA → abaA → abaaA → abaab

Regular Grammars

Regular grammars are another representation of regular languages.

- **They are equivalent in power to DFAs and NFAs.**

A right-linear grammar is a type of regular grammar.

In a right-linear grammar all rules must have at most one variable in the right-hand side and that variable must be to the right of any terminals.

Regular Grammars

In a right-linear grammar all rules must have at most one variable in the right-hand side and that variable must be to the right of any terminals.

Example:

$$\Sigma = \{0, 1\}$$

$$\textit{variables} = \{S, X, Y\}$$

$$\textit{start symbol} = S$$

$$\textit{rules} = \{S \to 0X,\ S \to 1X,\ S \to 0Y,\ X \to 01Y,\ Y \to 1X,\ Y \to \lambda\}$$

Regular Expressions (RE's)

Thus far we have described languages using finite automata, English words, and regular grammars

There is another precise and parsimonious expression to describe regular languages

- **Example:** All strings with at least one 1 becomes $\Sigma^{*} \cdot \{1\} \cdot \Sigma^{*}$, or more simply $\Sigma^{*}1\Sigma^{*}$

Where have you seen RE's?

Filename matching

ls *.txt   rm data1??.bak

Grep, Awk, Sed, …

Search expressions within text editors

Perl, Java, C++, …

RE Inductive Definition

R is a regular expression if R is

1. a for some $a \in \Sigma$
2. ε
3. $\varnothing$
4. $R_1 \cup R_2$ where R_1 and R_2 are both regular expressions
5. $R_1 \bullet R_2$ where R_1 and R_2 are both regular expressions (written R_1R_2)
6. $(R_1{}^*)$ where R_1 is a regular expression

Convention:
Abuse of notation.
These should be sets!

RE Examples

$\Sigma = \{0,1\}$

When appearing in regular expressions, union is often read as *"or"*.

$(\mathbf{a} \cup \mathbf{b}) \equiv$ "a or b"

JFLAP uses the plus symbol for union

$\mathbf{aba(b \cup a)b} \equiv$ `aba(b+a)b`

RE's and Regular Languages

Theorem: **A language is regular if and only if some regular expression describes it.**

Every regular expression has a corresponding DFA and vice versa.

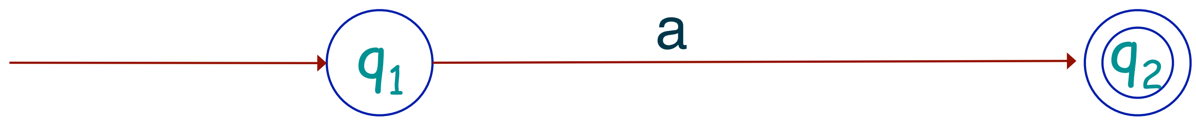
RE’s and Regular Languages

Lemma:

If a language is described by a regular expression, then it is regular.

- Find an NFA corresponding to any regular expression
- Use inductive definition of RE’s

1. $R=a$ for some $a \in \Sigma$

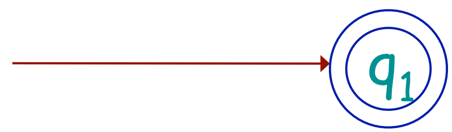


$N = \{\{q_1,q_2\},\Sigma,\delta,q_1,\{q_2\}\}$

where $\delta(q_1,a)=\{q_2\}$ and

$\delta(r,x)=\varnothing$ whenever $r=q_2$ or $x \neq a$

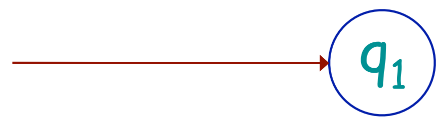
2. R=ε



$N = \{\{q_1\},\Sigma,\delta,q_1,\{q_1\}\}$

where $\delta(q_1,x)=\varnothing$ for all x

3. $R=\varnothing$



$$N = \{\{q_1\},\Sigma,\delta,q_1,\varnothing\}$$

where $\delta(q_1,x)=\varnothing$ for all x

Remaining Constructions

$R = R_1 \cup R_2$

$R = R_1 \bullet R_2$

$R = R_1^*$

These were all shown to be regular operators

- We know we can construct NFA's for R provided they exist for R_1 and R_2

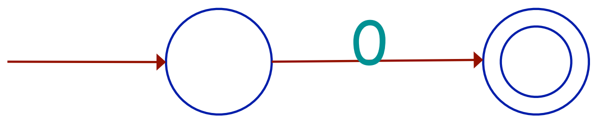
Example 1

$\Sigma = \{0,1\}$

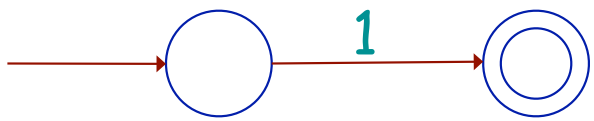
$R = \Sigma 1$

$R = (0 \cup 1) 1$

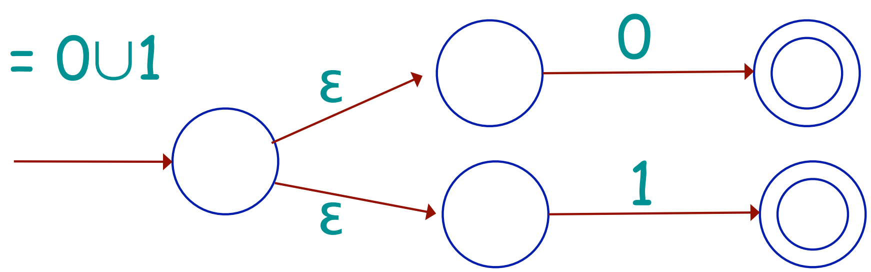
$R_1 = 0$



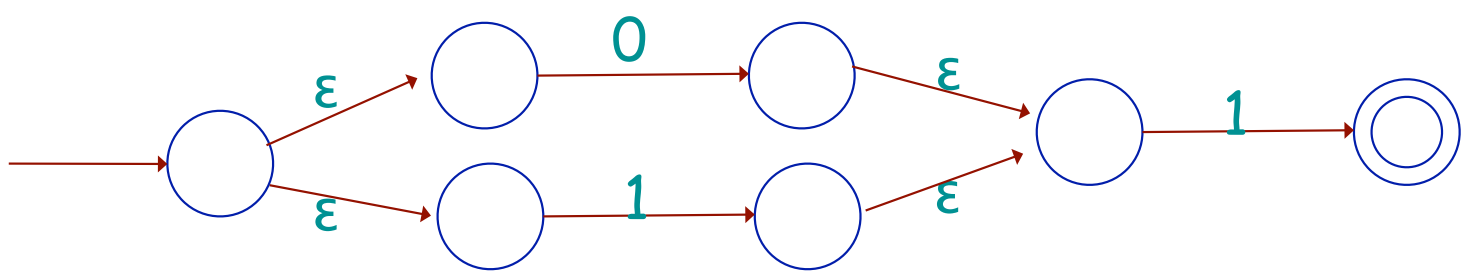
$R_2 = 1$



$R_3 = 0 \cup 1$



$R = \Sigma 1$

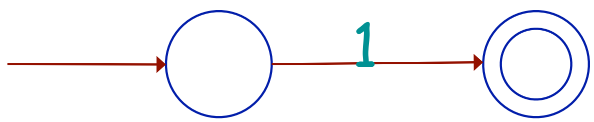


Example 2

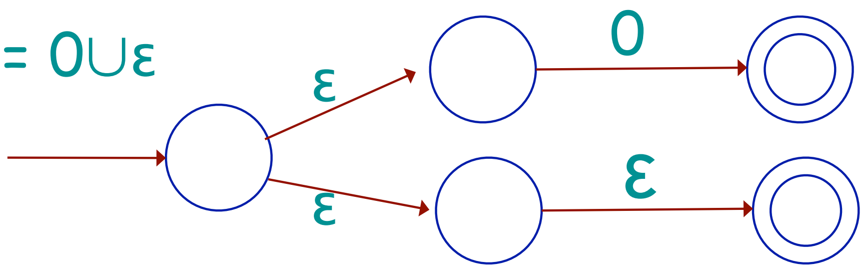
$\Sigma = \{0,1\}$

$R = 1(0 \cup \varepsilon)\Sigma^*$

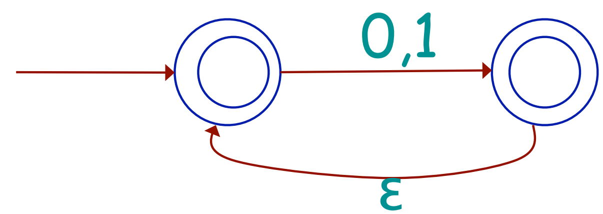
$R_1 = 1$



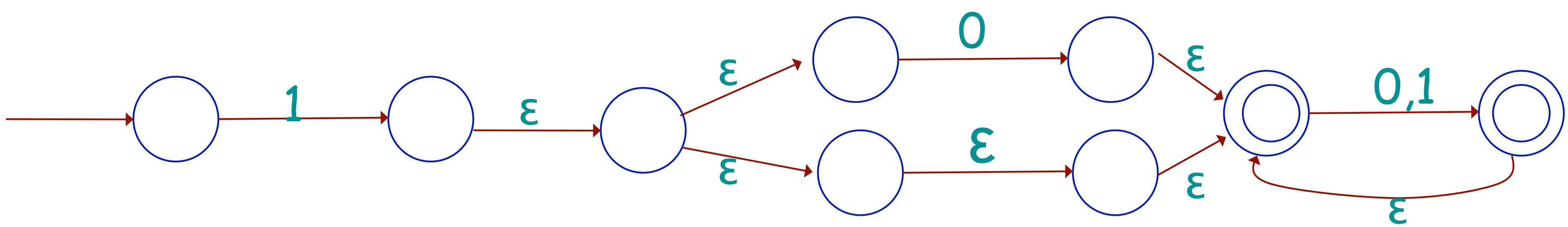
$R_2 = 0 \cup \varepsilon$



$R_3 = \Sigma^*$



$R = 1(0 \cup \varepsilon)\Sigma^*$



Equivalence of RE's and DFA's

We have seen that every RE has a corresponding NFA

- Therefore, every RE has a corresponding DFA
- Thus every RE describes a regular language

We need to show that every regular language can be described by a RE

Begin by showing how to convert all DFA's into GNFA's

- Generalized Nondeterministic Finite Automata