# T065001: Introduction to Formal Languages

## Lecture 2: Finite-state automata, regular languages, nondeterminism (1)

*Chapter 1.1 in Sipser's textbook*

2025-04-21

(Lecture slides by Yih-Kuen Tsay)

## Finite Automata

- *What is a computer?*
- Real computers are complicated.
- To set up a manageable mathematical theory of computers, we use an idealized computer called a *computational model*.
- The *finite automaton* (finite-state machine) is the simplest of such models.
- It represents a computer with an extremely limited amount of memory.

## Finite Automata (cont.)

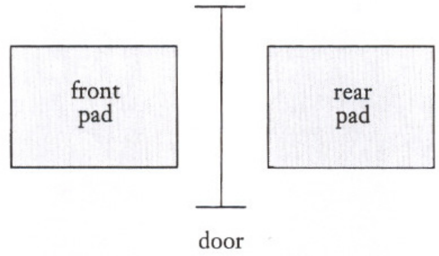


FIGURE 1.1
Top view of an automatic door

## Finite Automata (cont.)

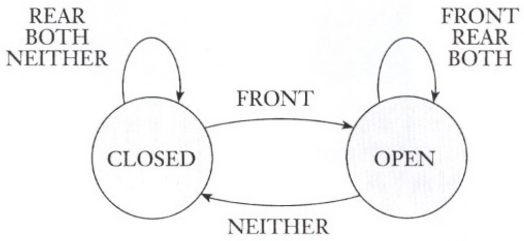
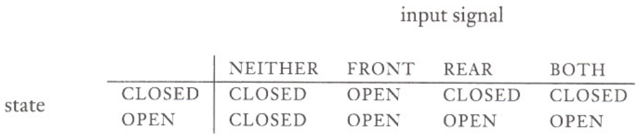


FIGURE **1.2**
State diagram for automatic door controller

## Finite Automata (cont.)



| | | input signal | | | |
|---|---|---|---|---|---|
| | | NEITHER | FRONT | REAR | BOTH |
| state | CLOSED | CLOSED | OPEN | CLOSED | CLOSED |
| | OPEN | CLOSED | OPEN | OPEN | OPEN |

**FIGURE 1.3**
State transition table for automatic door controller

## Another Example

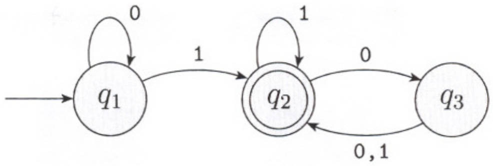


**FIGURE 1.4**
A finite automaton called $M_1$ that has three states

## Formal Definition

- Though state diagrams are easier to grasp intuitively, we need the formal definition, too.
- A formal definition is precise so as to resolve any uncertainties about what is allowed in a finite automaton.

### Definition (1.5)

A *finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set of *states*,
2. $\Sigma$ is a finite set of symbols (the *alphabet*),
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start* state, and
5. $F \subseteq Q$ is the set of *accept* states.

## Formal Definition

- Though state diagrams are easier to grasp intuitively, we need the formal definition, too.
- A formal definition is precise so as to resolve any uncertainties about what is allowed in a finite automaton.

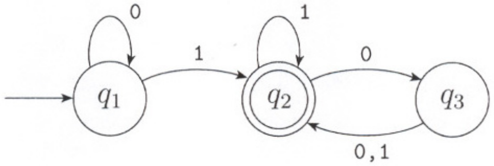
### Definition (1.5)

A *finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set of *states*,
2. $\Sigma$ is a finite set of symbols (the *alphabet*),
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start* state, and
5. $F \subseteq Q$ is the set of *accept* states.

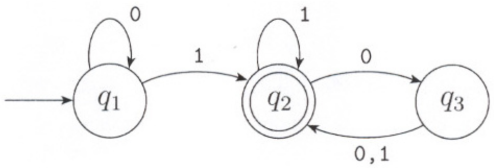
**Remark:** This is also called a "deterministic finite automaton (DFA)".

## Another Example (cont.)



Formally, $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

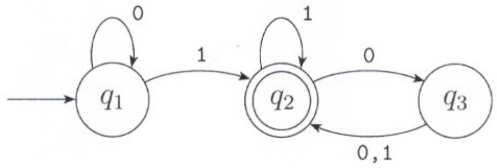
## Another Example (cont.)



Formally, $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,

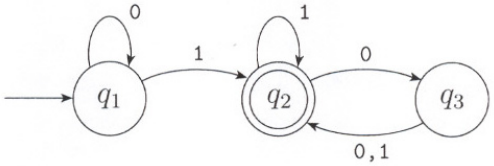
## Another Example (cont.)



Formally, $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0, 1\}$,

## Another Example (cont.)



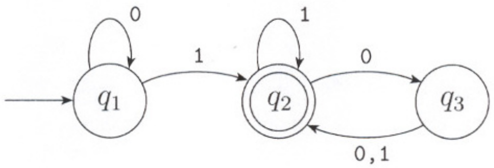
Formally, $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0, 1\}$,
3. $\delta$ is given as

|  | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_2$ | $q_2$ |

,

## Another Example (cont.)



Formally, $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

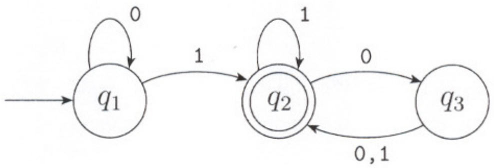
1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0, 1\}$,
3. $\delta$ is given as

|  | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_2$ | $q_2$ |

,

4. $q_1$ is the start state, and

## Another Example (cont.)



Formally, $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0, 1\}$,
3. $\delta$ is given as

|  | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_2$ | $q_2$ |

,

4. $q_1$ is the start state, and
5. $F = \{q_2\}$.

## Formal Definition (cont.)

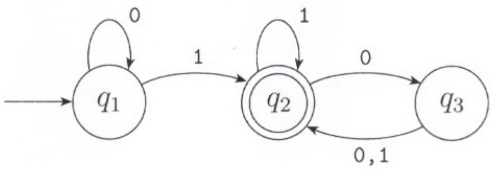


FIGURE 1.6
The finite automaton $M_1$

A DFA *accepts* a string if it stops at an accept state after processing the string symbol by symbol, starting from the start state. (Otherwise, it *rejects* the string.) For instance, $M_1$ above accepts 011 and 010100.

"Follow the arrows."

## Formal Definition of Computation

We already have an informal idea of how a machine computes, i.e., how a machine accepts or rejects a string. Below is a formalization.

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1 w_2 \ldots w_n$ be a string over $\Sigma$.
- We say that $M$ *accepts* $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ exists such that
    1. $r_0 = q_0$,
    2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, 1, \ldots, n-1$, and
    3. $r_n \in F$.

## Strings and Languages

- An *alphabet* is any finite set of *symbols*.
- A *string* over an alphabet is a finite sequence of symbols from that alphabet.
- The *length* of a string $w$, written as $|w|$, is the number of symbols that $w$ contains.
- The string of length 0 is called the *empty string*, written as $\varepsilon$.
- The *concatenation* of $x$ and $y$, written as $xy$, is the string obtained from appending $y$ to the end of $x$.
- A *formal language* is a set of strings. (Referred to as a *language* from now on.) A language can be finite or infinite.

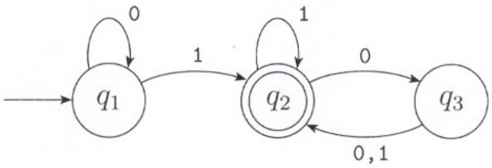
**Example:** $\{\mathtt{a}^n\,\mathtt{b}^n \mid n \geq 0\}$ is a language over the alphabet $\{\mathtt{a}, \mathtt{b}\}$. It consists of all strings of the form $\mathtt{aa}\ldots\mathtt{abb}\ldots\mathtt{b}$ with an equal number of as and bs. Note that $\varepsilon$ also belongs to this language.

## Language Recognizers

- Let $A$ be the set of all strings that a machine $M$ accepts.
- We say that $A$ is the *language of machine $M$* and write $L(M) = A$.
- We also say that *$M$ recognizes $A$* (or that $M$ accepts $A$).
- A machine is said to accept the empty language $\emptyset$ if it accepts no strings.

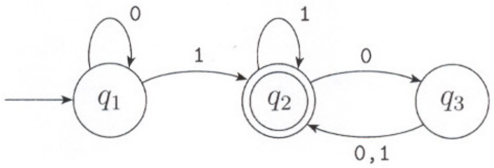
## Language Recognizers

- Let $A$ be the set of all strings that a machine $M$ accepts.
- We say that $A$ is the *language of machine $M$* and write $L(M) = A$.
- We also say that *$M$ recognizes $A$* (or that $M$ accepts $A$).
- A machine is said to accept the empty language $\emptyset$ if it accepts no strings.



## Language Recognizers

- Let $A$ be the set of all strings that a machine $M$ accepts.
- We say that $A$ is the *language of machine $M$* and write $L(M) = A$.
- We also say that *$M$ recognizes $A$* (or that $M$ accepts $A$).
- A machine is said to accept the empty language $\emptyset$ if it accepts no strings.
- Regarding the example automaton $M_1$, $L(M_1) = \{w \mid w$ contains at least one 1 and an even number of 0s follow the last 1$\}$.



## Language Recognizers (cont.)

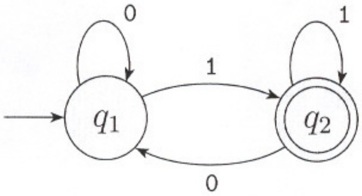


**FIGURE 1.8**
State diagram of the two-state finite automaton $M_2$

## Language Recognizers (cont.)

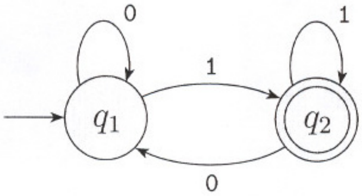


**FIGURE 1.8**
State diagram of the two-state finite automaton $M_2$

Note: $L(M_2) = \{w \mid w \text{ ends in a } 1\}$

## Language Recognizers (cont.)

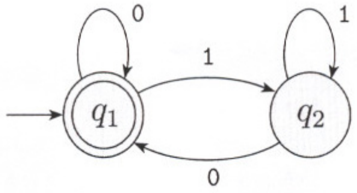


**FIGURE 1.10**
State diagram of the two-state finite automaton $M_3$

## Language Recognizers (cont.)

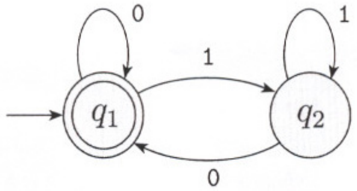


**FIGURE 1.10**
State diagram of the two-state finite automaton $M_3$

Note: $L(M_3) = \{w \mid w \text{ is the empty string or ends in a } 0\}$

## Language Recognizers (cont.)

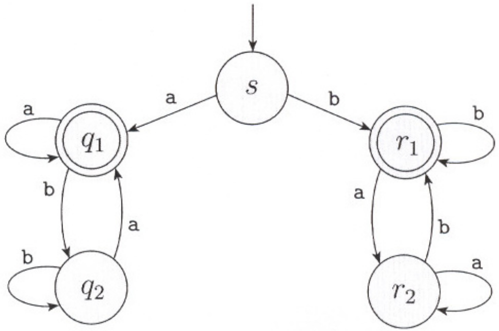


**FIGURE 1.12**
Finite automaton $M_4$

$$\begin{tikzcd}
 & & s \arrow[ll, "a"'] \arrow[rr, "b"] & & \\
q_1 \arrow[loop left, "a"] \arrow[d, bend right, "b"'] & & & & r_1 \arrow[loop right, "b"] \arrow[d, bend right, "a"'] \\
q_2 \arrow[loop left, "b"] \arrow[u, bend right, "a"'] & & & & r_2 \arrow[loop right, "a"] \arrow[u, bend right, "b"']
\end{tikzcd}$$

## Language Recognizers (cont.)

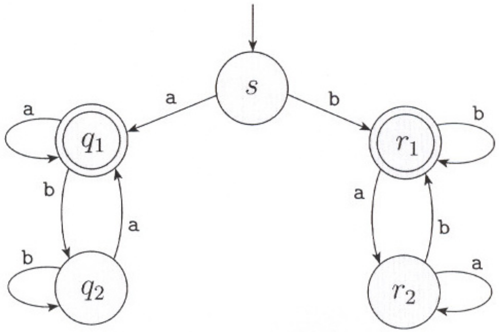


FIGURE **1.12**
Finite automaton $M_4$

Note: $M_4$ accepts all nonempty strings over the alphabet $\{\mathtt{a}, \mathtt{b}\}$ that start and end with the same symbol.

## Language Recognizers (cont.)

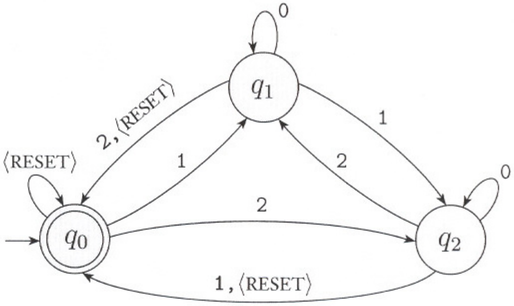
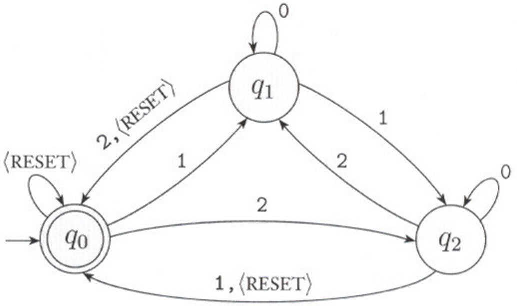


**FIGURE 1.14**
Finite automaton $M_5$

## Language Recognizers (cont.)



0
$q_1$
$2, \langle\text{RESET}\rangle$
1
0, ⟨RESET⟩
1
2
0
2
$q_0$
$q_2$
$1, \langle\text{RESET}\rangle$

FIGURE **1.14**
Finite automaton $M_5$

Note: $\Sigma = \{\langle\text{RESET}\rangle, 0, 1, 2\}$. $M_5$ accepts if the sum of the numerical input symbols it had read since the last $\langle\text{RESET}\rangle$ is a multiple of 3.

## Designing Finite Automata

The “reader as automaton” method:

1. Determine the necessary information needed to be remembered about the string as it is being read.
2. Represent the information as a finite list of possibilities and assign a state to each of the possibilities.
3. Assign the transitions by seeing how to go from one possibility to another upon reading a symbol.
4. Set the start state to be the state corresponding to the possibility associated with having seen 0 symbols so far.
5. Set the accept states to be those corresponding to possibilities where you want to accept the input read so far.

## Designing Finite Automata (cont.)

Consider constructing an automaton that recognizes binary strings with an odd number of 1's.

## Designing Finite Automata (cont.)

Consider constructing an automaton that recognizes binary strings with an odd number of 1's.

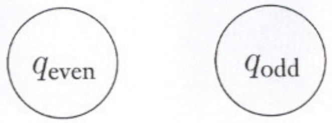


**FIGURE 1.18**
The two states $q_{\text{even}}$ and $q_{\text{odd}}$

## Designing Finite Automata (cont.)

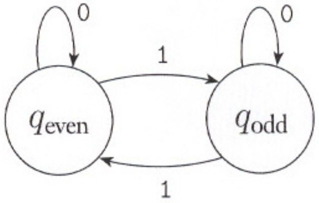


**FIGURE 1.19**
Transitions telling how the possibilities rearrange

## Designing Finite Automata (cont.)

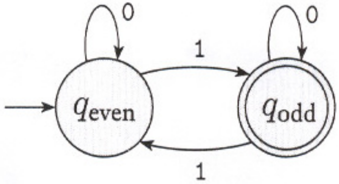


**FIGURE 1.20**
Adding the start and accept states

## Designing Finite Automata (cont.)

**Another example:** Design a finite automaton that recognizes the language of all strings over $\{0, 1\}$ that contain 001 as a substring.

## Designing Finite Automata (cont.)

**Another example:** Design a finite automaton that recognizes the language of all strings over $\{0, 1\}$ that contain 001 as a substring.

**Solution:**

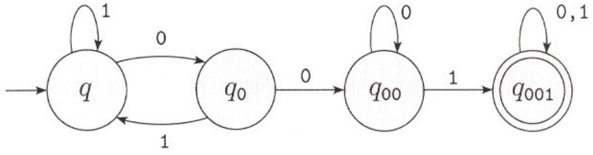


FIGURE 1.22
Accepts strings containing 001

## Regular Languages

**Definition (1.16)**

A language is called a *regular language* if some finite automaton recognizes it.

## Regular Languages

**Definition (1.16)**

A language is called a *regular language* if some finite automaton recognizes it.

**Examples:** $L(M_1)$, $L(M_2)$, $L(M_3)$, etc. in today's lecture are regular.

Regular languages have many nice properties.
E.g., if we take any two regular languages and combine them in a certain way then we always end up with a regular language, as we'll see shortly.

## The Regular Operations

### Definition (1.23)

Let $A$ and $B$ be languages. The three *regular operations* are defined as follows:

- **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

**EXAMPLE 1.24** ..........

Let the alphabet $\Sigma$ be the standard 26 letters $\{\texttt{a}, \texttt{b}, \ldots, \texttt{z}\}$. If $A = \{\texttt{good}, \texttt{bad}\}$ and $B = \{\texttt{boy}, \texttt{girl}\}$, then

## The Regular Operations

### Definition (1.23)

Let $A$ and $B$ be languages. The three *regular operations* are defined as follows:

- **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

**EXAMPLE 1.24**

Let the alphabet $\Sigma$ be the standard 26 letters $\{\texttt{a}, \texttt{b}, \ldots, \texttt{z}\}$. If $A = \{\texttt{good}, \texttt{bad}\}$ and $B = \{\texttt{boy}, \texttt{girl}\}$, then

$$A \cup B = \{\texttt{good}, \texttt{bad}, \texttt{boy}, \texttt{girl}\},$$

## The Regular Operations

### Definition (1.23)

Let $A$ and $B$ be languages. The three *regular operations* are defined as follows:

- **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

**EXAMPLE 1.24**

Let the alphabet $\Sigma$ be the standard 26 letters $\{\texttt{a}, \texttt{b}, \dots, \texttt{z}\}$. If $A = \{\texttt{good}, \texttt{bad}\}$ and $B = \{\texttt{boy}, \texttt{girl}\}$, then

$$A \cup B = \{\texttt{good}, \texttt{bad}, \texttt{boy}, \texttt{girl}\},$$

$$A \circ B = \{\texttt{goodboy}, \texttt{goodgirl}, \texttt{badboy}, \texttt{badgirl}\}, \text{ and}$$

## The Regular Operations

### Definition (1.23)

Let $A$ and $B$ be languages. The three *regular operations* are defined as follows:

- **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1x_2\ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

**EXAMPLE 1.24**

Let the alphabet $\Sigma$ be the standard 26 letters $\{\texttt{a}, \texttt{b}, \ldots, \texttt{z}\}$. If $A = \{\texttt{good}, \texttt{bad}\}$ and $B = \{\texttt{boy}, \texttt{girl}\}$, then

$$A \cup B = \{\texttt{good}, \texttt{bad}, \texttt{boy}, \texttt{girl}\},$$

$$A \circ B = \{\texttt{goodboy}, \texttt{goodgirl}, \texttt{badboy}, \texttt{badgirl}\}, \text{ and}$$

$$A^* = \{\varepsilon, \texttt{good}, \texttt{bad}, \texttt{goodgood}, \texttt{goodbad}, \texttt{badgood}, \texttt{badbad}, \\ \texttt{goodgoodgood}, \texttt{goodgoodbad}, \texttt{goodbadgood}, \texttt{goodbadbad}, \ldots\}.$$

■

## Closedness

- A collection of objects is *closed* under some operation if applying the operation to members of the collection returns an object still in the collection.
- We will show that the collection of regular languages is closed under all three regular operations.

## Closedness under Union

**Theorem (1.25)**

*The class of regular languages is closed under the union operation. In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$.*

- The proof is by construction. To prove that $A_1 \cup A_2$ is regular, we construct a finite automaton $M$ that recognizes $A_1 \cup A_2$.
- Suppose that a finite automaton $M_1$ recognizes $A_1$ and another $M_2$ recognizes $A_2$.
- Machine $M$ works by *simulating* both $M_1$ and $M_2$ and accepting if either simulation accepts.
- As the input symbols arrive one by one, $M$ remembers the state that each machine would be in if it had read up to this point.

But how can $M$ keep track of *both* states that $M_1$ and $M_2$ would be in?

## Closedness under Union (cont.)

**Idea:** Let $M$ have one state for each pair of states from $M_1$ and $M_2$!

### Theorem (1.25)

*The class of regular languages is closed under the union operation. In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$.*

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes $A_1$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes $A_2$.
- Construct $M = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$:
  1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
  2. $\Sigma$ is the same. (Generalization is possible.)
  3. For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$, let $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$.
  4. $q_0 = (q_1, q_2)$.
  5. $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$.

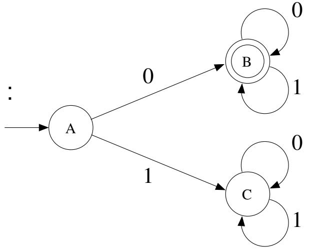
## Closedness under Union (cont.)

To illustrate the theorem, here is an example by Adam Webber.

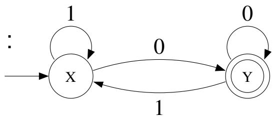
Let $A_1 =$ the set of strings over the alphabet $\{0, 1\}$ that start with a 0 and $A_2 =$ the set of strings over the alphabet $\{0, 1\}$ that end with a 0.

Both $A_1$ and $A_2$ are regular because $A_1 = L(M_1)$ and $A_2 = L(M_2)$ with:

$M_1$ :



$M_2$ :



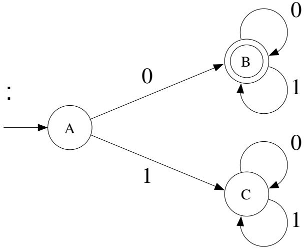
## Closedness under Union (cont.)

To illustrate the theorem, here is an example by Adam Webber.

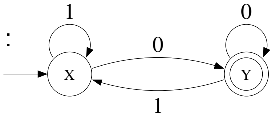
Let $A_1$ = the set of strings over the alphabet $\{0, 1\}$ that start with a 0 and $A_2$ = the set of strings over the alphabet $\{0, 1\}$ that end with a 0.

Both $A_1$ and $A_2$ are regular because $A_1 = L(M_1)$ and $A_2 = L(M_2)$ with:

$M_1$ :



$M_2$ :



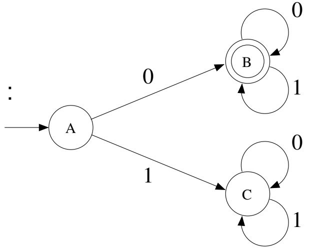
Now we can use $M_1$ and $M_2$ above to create an $M$ such that:

$L(M) = L(M_1) \cup L(M_2)$ = the set of strings over the alphabet $\{0, 1\}$ that start or end with a 0

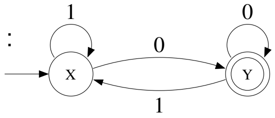
## Closedness under Union (cont.)

To illustrate the theorem, here is an example by Adam Webber.

$M_1$ :

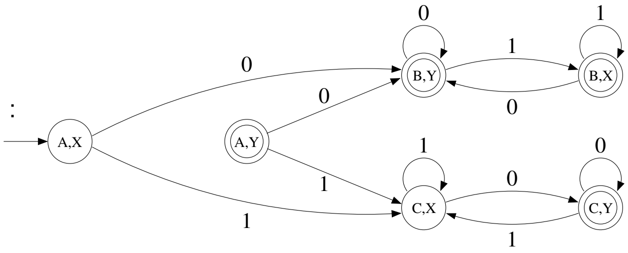


$M_2$ :



The construction in the proof of Theorem 1.25 yields the following:

$M$ :



## Closedness under Concatenation

**Theorem (1.26)**

*The class of regular languages is closed under the concatenation operation. In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \circ A_2$.*

- Proof by construction along the lines of the proof for closedness under union does not work in this case.

## Closedness under Concatenation

**Theorem (1.26)**

*The class of regular languages is closed under the concatenation operation. In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \circ A_2$.*

- Proof by construction along the lines of the proof for closedness under union does not work in this case.
- Suppose $A_1$ is the set of binary strings containing 001, while $A_2$ is the set of binary strings with an odd number of 1's.
    - The binary string 0010011 is in $A_1 \circ A_2$.
    - How can a machine, simulating $M_1$ and then $M_2$, know that it should not stop $M_1$ and move to $M_2$ after seeing the first occurrence of 001?

## Closedness under Concatenation

**Theorem (1.26)**

*The class of regular languages is closed under the concatenation operation. In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \circ A_2$.*

- Proof by construction along the lines of the proof for closedness under union does not work in this case.
- Suppose $A_1$ is the set of binary strings containing 001, while $A_2$ is the set of binary strings with an odd number of 1's.
  - The binary string 0010011 is in $A_1 \circ A_2$.
  - How can a machine, simulating $M_1$ and then $M_2$, know that it should not stop $M_1$ and move to $M_2$ after seeing the first occurrence of 001?
- We resort to a new technique called *nondeterminism*.
  TO BE CONTINUED...