Deterministic Finite Automata (DFA)

DFAs

DFAs

Three Examples

Standard Conventions

Standard Conventions

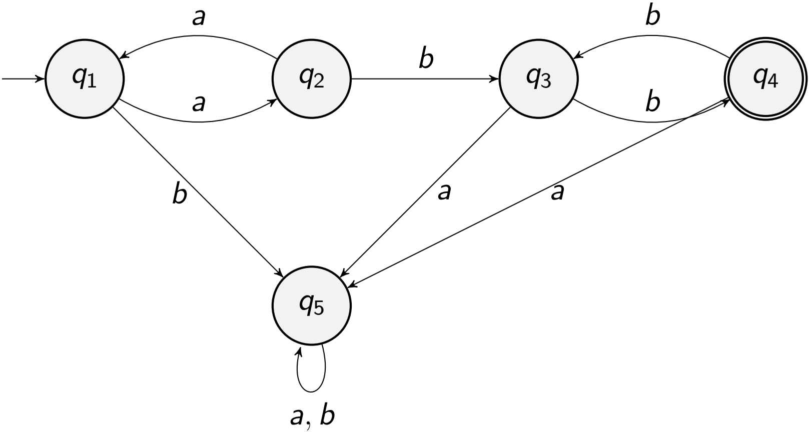
1. The state that has an arrow pointing to it (from nowhere, not from another state) is the **start** state.

Standard Conventions

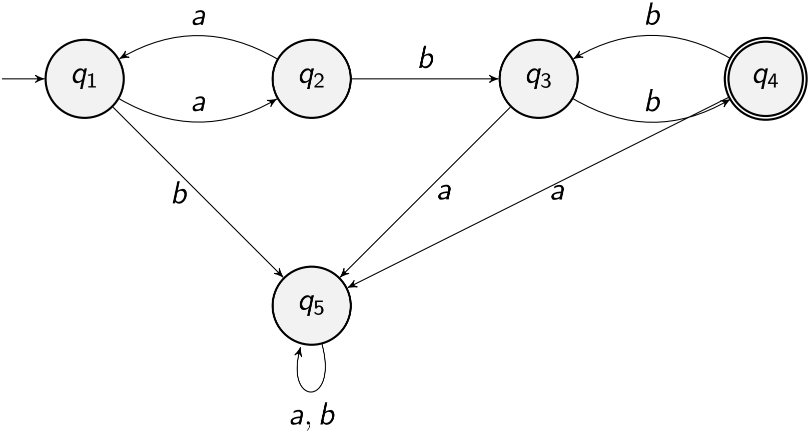
1. The state that has an arrow pointing to it (from nowhere, not from another state) is the **start** state.
2. The states that are circled are **final states**. If the machine ends up there, then the string is accepted.

DFA Diagram: A First Example

DFA Diagram: A First Example

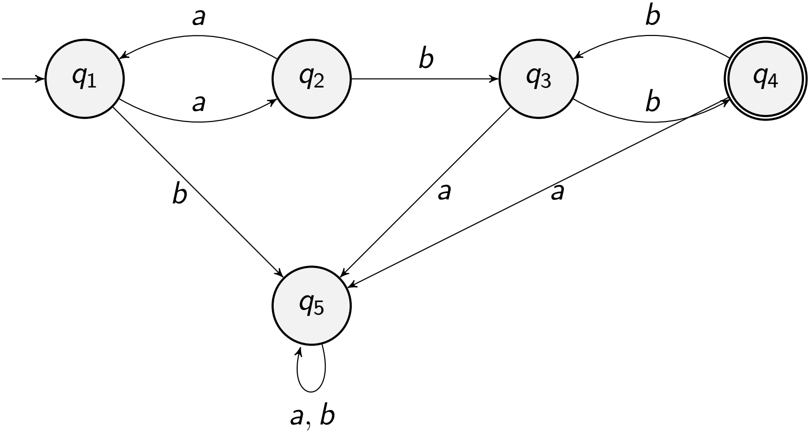


DFA Diagram: A First Example



What is the language?

DFA Diagram: A First Example

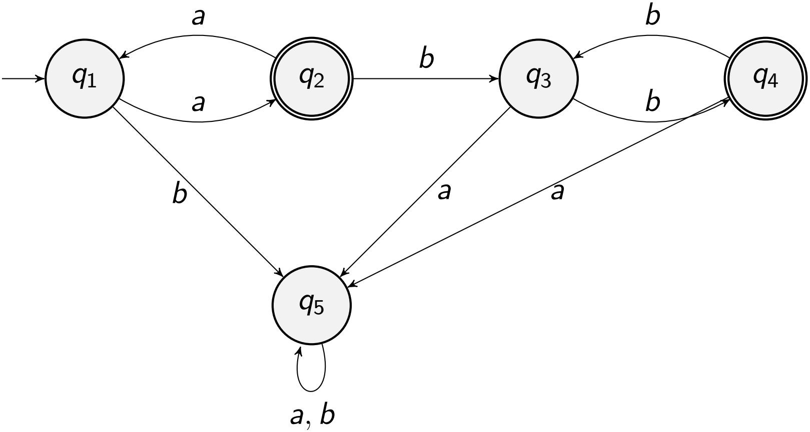


What is the language?

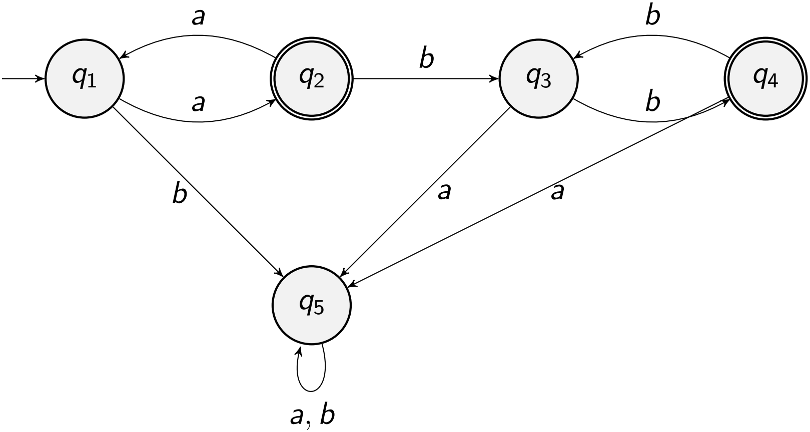
Odd number of a's followed by an even number of b's, but at least two.

DFA Diagram: A Second Example

DFA Diagram: A Second Example

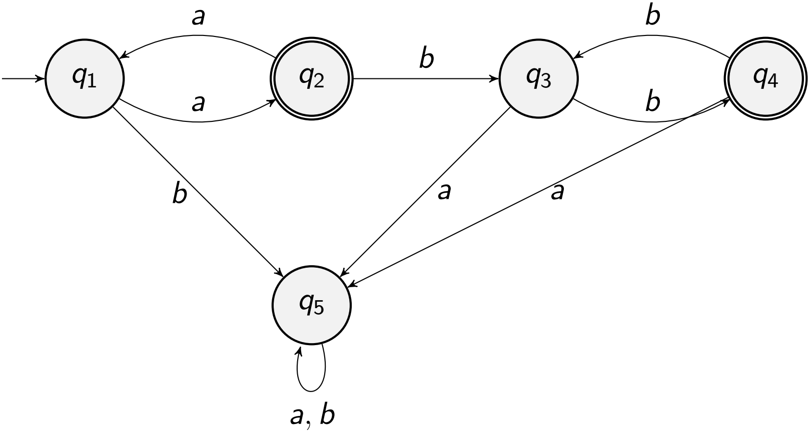


DFA Diagram: A Second Example



What is the language?

DFA Diagram: A Second Example

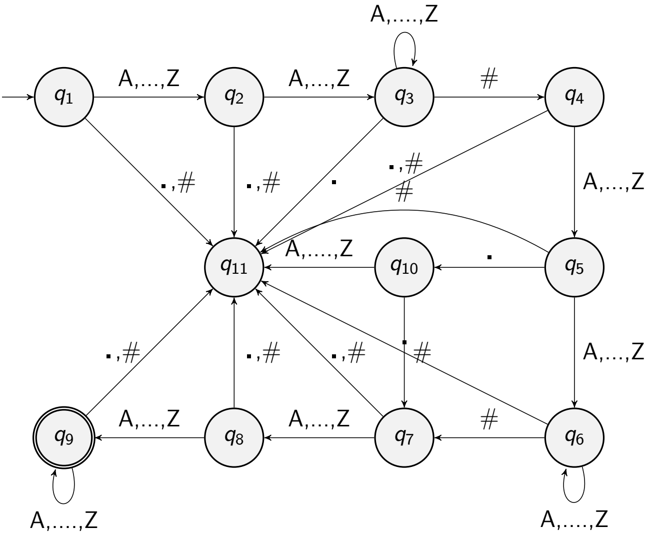


What is the language?

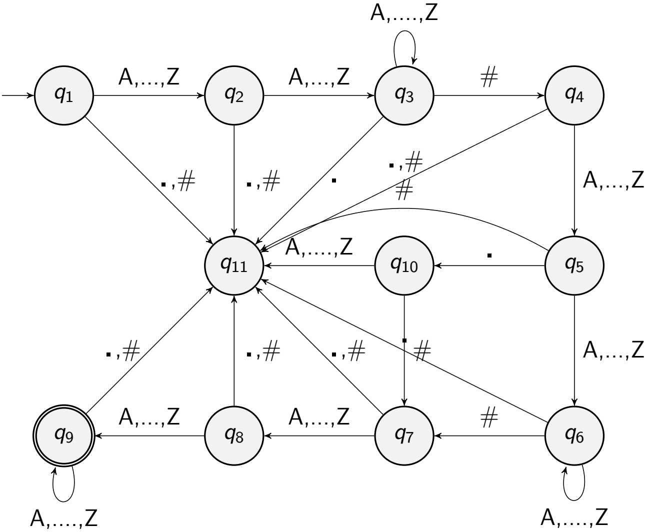
Odd number of *a*'s followed by an even number of *b*'s.

DFA Diagram: A Third Example

DFA Diagram: A Third Example

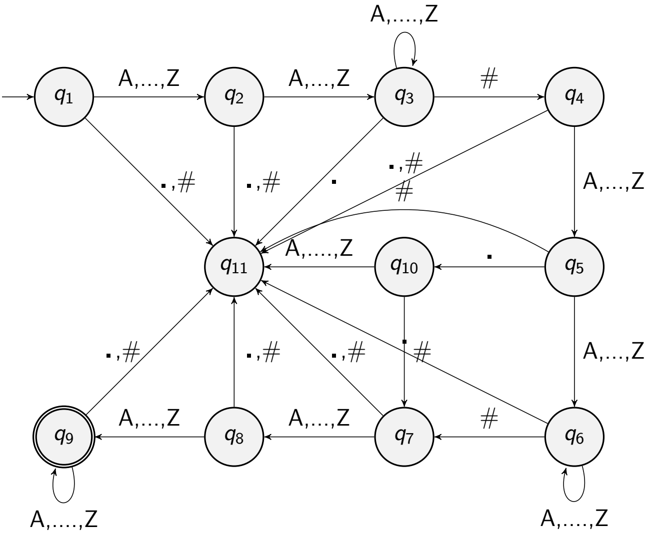


DFA Diagram: A Third Example



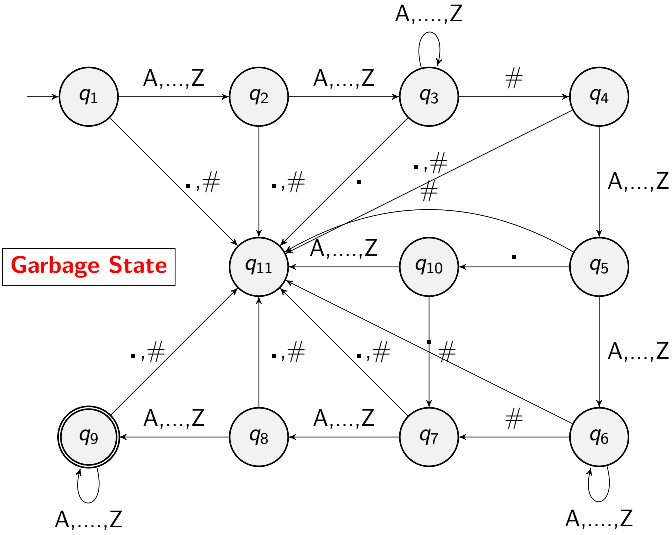
What is the language?

DFA Diagram: A Third Example



What is the language? **Messy**

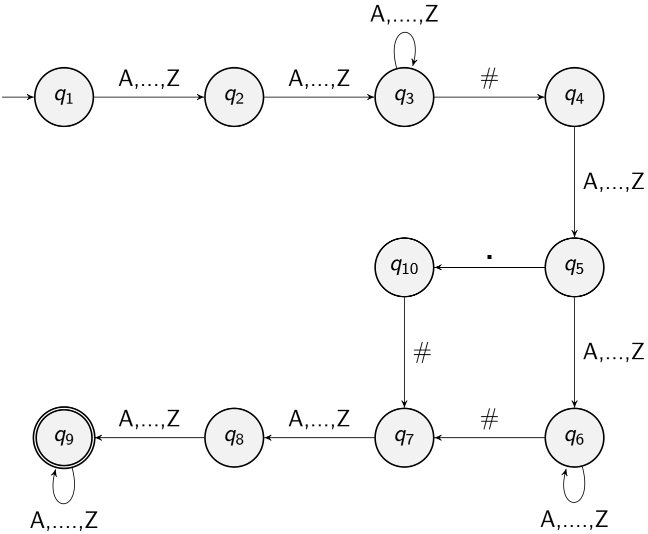
DFA Diagram: A Third Example



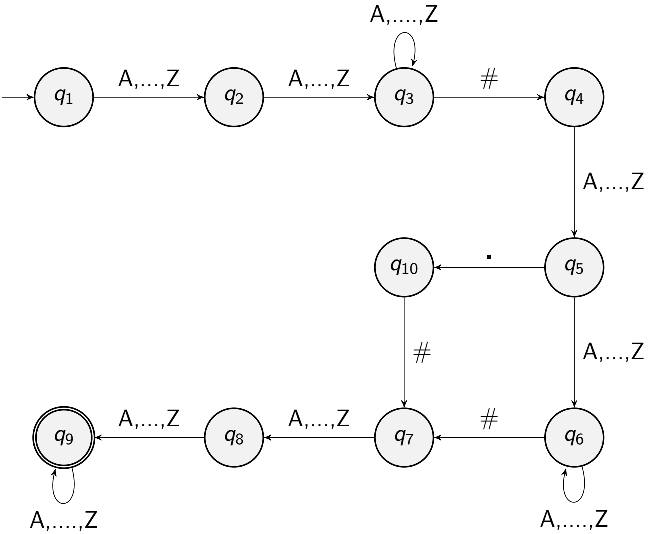
What is the language? **Messy**

Third Example without Garbage State

Third Example without Garbage State



Third Example without Garbage State



What is the language?

Short Detour

Short Detour

Modular Arithmetic

Modular Arithmetic: Definitions

Modular Arithmetic: Definitions

- $x \equiv y \pmod{N}$ if and only if N divides $x - y$.

Modular Arithmetic: Definitions

- $x \equiv y \pmod{N}$ if and only if N divides $x - y$.
- $25 \equiv 35 \pmod{10}$.

Modular Arithmetic: Definitions

- $x \equiv y \pmod{N}$ if and only if N divides $x - y$.
- $25 \equiv 35 \pmod{10}$.
- $100 \equiv 2 \pmod{7}$ since $100 = 7 \times 14 + 2$.

Modular Arithmetic II: Convention

Common usage:

$$100 \equiv 2 \pmod{7}$$

Modular Arithmetic II: Convention

Common usage:

$$100 \equiv 2 \pmod{7}$$

Commonly if we are in mod n we have a large number on the left and then a number between 0 and $n-1$ on the right.

Modular Arithmetic II: Convention

Common usage:

$$100 \equiv 2 \pmod{7}$$

Commonly if we are in mod n we have a large number on the left and then a number between 0 and $n-1$ on the right.

When dealing with mod n we assume the entire universe is $\{0, 1, \ldots, n-1\}$.

Modular Arithmetic: $+, -, \times$

$\equiv$ is mod 26 for this slide. (This slide is from CMSC456.)

Modular Arithmetic: $+, -, \times$

$\equiv$ is mod 26 for this slide. (This slide is from CMSC456.)

1. Addition: $x + y$ is easy: wrap around. E.g., $20 + 10 \equiv 30 \equiv 4$. Only use the number 30 as an intermediary value on the way to the real answer.

Modular Arithmetic: $+, -, \times$

$\equiv$ is mod 26 for this slide. (This slide is from CMSC456.)

1. Addition: $x + y$ is easy: wrap around. E.g., $20 + 10 \equiv 30 \equiv 4$. Only use the number 30 as an intermediary value on the way to the real answer.
2. $-7 \equiv x$ where $0 \le x \le 25$.

Modular Arithmetic: $+, -, \times$

$\equiv$ is mod 26 for this slide. (This slide is from CMSC456.)

1. Addition: $x + y$ is easy: wrap around. E.g., $20 + 10 \equiv 30 \equiv 4$. Only use the number 30 as an intermediary value on the way to the real answer.
2. $-7 \equiv x$ where $0 \le x \le 25$.
   Pedantic: $-y$ is the number such that $y + (-y) \equiv 0$.

Modular Arithmetic: $+, -, \times$

$\equiv$ is mod 26 for this slide. (This slide is from CMSC456.)

1. Addition: $x + y$ is easy: wrap around. E.g., $20 + 10 \equiv 30 \equiv 4$. Only use the number 30 as an intermediary value on the way to the real answer.
2. $-7 \equiv x$ where $0 \le x \le 25$.
   Pedantic: $-y$ is the number such that $y + (-y) \equiv 0$.
   $-7 \equiv 19 \pmod{26}$ because $19 + 7 \equiv 0 \pmod{26}$.

Modular Arithmetic: $+, -, \times$

$\equiv$ is mod 26 for this slide. (This slide is from CMSC456.)

1. Addition: $x + y$ is easy: wrap around. E.g., $20 + 10 \equiv 30 \equiv 4$. Only use the number 30 as an intermediary value on the way to the real answer.
2. $-7 \equiv x$ where $0 \le x \le 25$.
   Pedantic: $-y$ is the number such that $y + (-y) \equiv 0$.
   $-7 \equiv 19 \pmod{26}$ because $19 + 7 \equiv 0 \pmod{26}$.
   Shortcut: $-y \equiv 26 - y$.

Modular Arithmetic: $+, -, \times$

$\equiv$ is mod 26 for this slide. (This slide is from CMSC456.)

1. Addition: $x + y$ is easy: wrap around. E.g., $20 + 10 \equiv 30 \equiv 4$. Only use the number 30 as an intermediary value on the way to the real answer.
2. $-7 \equiv x$ where $0 \le x \le 25$.
   Pedantic: $-y$ is the number such that $y + (-y) \equiv 0$.
   $-7 \equiv 19 \pmod{26}$ because $19 + 7 \equiv 0 \pmod{26}$.
   Shortcut: $-y \equiv 26 - y$.
3. Mult: xy is easy: wrap around. E.g., $20 \times 10 \equiv 200 \equiv 18$.

Modular Arithmetic: $+, -, \times$

$\equiv$ is mod 26 for this slide. (This slide is from CMSC456.)

1. Addition: $x + y$ is easy: wrap around. E.g., $20 + 10 \equiv 30 \equiv 4$. Only use the number 30 as an intermediary value on the way to the real answer.
2. $-7 \equiv x$ where $0 \le x \le 25$.
   Pedantic: $-y$ is the number such that $y + (-y) \equiv 0$.
   $-7 \equiv 19 \pmod{26}$ because $19 + 7 \equiv 0 \pmod{26}$.
   Shortcut: $-y \equiv 26 - y$.
3. Mult: xy is easy: wrap around. E.g., $20 \times 10 \equiv 200 \equiv 18$.
   Shortcut to avoid big numbers:

Modular Arithmetic: $+, -, \times$

$\equiv$ is mod 26 for this slide. (This slide is from CMSC456.)

1. Addition: $x + y$ is easy: wrap around. E.g., $20 + 10 \equiv 30 \equiv 4$. Only use the number 30 as an intermediary value on the way to the real answer.
2. $-7 \equiv x$ where $0 \le x \le 25$.
   Pedantic: $-y$ is the number such that $y + (-y) \equiv 0$.
   $-7 \equiv 19 \pmod{26}$ because $19 + 7 \equiv 0 \pmod{26}$.
   Shortcut: $-y \equiv 26 - y$.
3. Mult: xy is easy: wrap around. E.g., $20 \times 10 \equiv 200 \equiv 18$.
   Shortcut to avoid big numbers:

$$20 \times 10 \equiv -6 \times 10 \equiv -2 \times 30 \equiv -2 \times 4 \equiv -8 \equiv 18.$$

Modular Arithmetic: $+, -, \times$

$\equiv$ is mod 26 for this slide. (This slide is from CMSC456.)

1. Addition: $x + y$ is easy: wrap around. E.g., $20 + 10 \equiv 30 \equiv 4$. Only use the number 30 as an intermediary value on the way to the real answer.
2. $-7 \equiv x$ where $0 \le x \le 25$.
   Pedantic: $-y$ is the number such that $y + (-y) \equiv 0$.
   $-7 \equiv 19 \pmod{26}$ because $19 + 7 \equiv 0 \pmod{26}$.
   Shortcut: $-y \equiv 26 - y$.
3. Mult: xy is easy: wrap around. E.g., $20 \times 10 \equiv 200 \equiv 18$.
   Shortcut to avoid big numbers:

   $$20 \times 10 \equiv -6 \times 10 \equiv -2 \times 30 \equiv -2 \times 4 \equiv -8 \equiv 18.$$

4. Division: Next Slide.

Modular Arithmetic: $\div$

$\equiv$ is mod 26 for this slide.
$\frac{1}{3} \equiv x$ where $0 \leq x \leq 25$.

Modular Arithmetic: $\div$

$\equiv$ is mod 26 for this slide.
$\frac{1}{3} \equiv x$ where $0 \leq x \leq 25$.
Pedantic: $\frac{1}{y}$ is the number such that $y \times \frac{1}{y} \equiv 1$.

Modular Arithmetic: $\div$

$\equiv$ is mod 26 for this slide.
$\frac{1}{3} \equiv x$ where $0 \le x \le 25$.
Pedantic: $\frac{1}{y}$ is the number such that $y \times \frac{1}{y} \equiv 1$.
$\frac{1}{3} \equiv 9$ since $9 \times 3 = 27 \equiv 1$.

Modular Arithmetic: $\div$

$\equiv$ is mod 26 for this slide.

$\frac{1}{3} \equiv x$ where $0 \leq x \leq 25$.

Pedantic: $\frac{1}{y}$ is the number such that $y \times \frac{1}{y} \equiv 1$.

$\frac{1}{3} \equiv 9$ since $9 \times 3 = 27 \equiv 1$.

Shortcut:

Modular Arithmetic: $\div$

$\equiv$ is mod 26 for this slide.
$\frac{1}{3} \equiv x$ where $0 \le x \le 25$.
Pedantic: $\frac{1}{y}$ is the number such that $y \times \frac{1}{y} \equiv 1$.
$\frac{1}{3} \equiv 9$ since $9 \times 3 = 27 \equiv 1$.
Shortcut: there is an algorithm that finds $\frac{1}{y}$ quickly.

Modular Arithmetic: $\div$

$\equiv$ is mod 26 for this slide.
$\frac{1}{3} \equiv x$ where $0 \le x \le 25$.
Pedantic: $\frac{1}{y}$ is the number such that $y \times \frac{1}{y} \equiv 1$.
$\frac{1}{3} \equiv 9$ since $9 \times 3 = 27 \equiv 1$.
Shortcut: there is an algorithm that finds $\frac{1}{y}$ quickly.
We will NOT study the algorithm later.

Modular Arithmetic: $\div$

$\equiv$ is mod 26 for this slide.
$\frac{1}{3} \equiv x$ where $0 \le x \le 25$.
Pedantic: $\frac{1}{y}$ is the number such that $y \times \frac{1}{y} \equiv 1$.
$\frac{1}{3} \equiv 9$ since $9 \times 3 = 27 \equiv 1$.
Shortcut: there is an algorithm that finds $\frac{1}{y}$ quickly.
We will NOT study the algorithm later.

$\frac{1}{2} \equiv x$ where $0 \le x \le 25$.

Modular Arithmetic: $\div$

$\equiv$ is mod 26 for this slide.
$\frac{1}{3} \equiv x$ where $0 \le x \le 25$.
Pedantic: $\frac{1}{y}$ is the number such that $y \times \frac{1}{y} \equiv 1$.
$\frac{1}{3} \equiv 9$ since $9 \times 3 = 27 \equiv 1$.
Shortcut: there is an algorithm that finds $\frac{1}{y}$ quickly.
We will NOT study the algorithm later.

$\frac{1}{2} \equiv x$ where $0 \le x \le 25$. Think about it.

Modular Arithmetic: $\div$

$\equiv$ is mod 26 for this slide.
$\frac{1}{3} \equiv x$ where $0 \le x \le 25$.
Pedantic: $\frac{1}{y}$ is the number such that $y \times \frac{1}{y} \equiv 1$.
$\frac{1}{3} \equiv 9$ since $9 \times 3 = 27 \equiv 1$.
Shortcut: there is an algorithm that finds $\frac{1}{y}$ quickly.
We will NOT study the algorithm later.

$\frac{1}{2} \equiv x$ where $0 \le x \le 25$. Think about it.
No such x exists.

Modular Arithmetic: $\div$

$\equiv$ is mod 26 for this slide.
$\frac{1}{3} \equiv x$ where $0 \le x \le 25$.
Pedantic: $\frac{1}{y}$ is the number such that $y \times \frac{1}{y} \equiv 1$.
$\frac{1}{3} \equiv 9$ since $9 \times 3 = 27 \equiv 1$.
Shortcut: there is an algorithm that finds $\frac{1}{y}$ quickly.
We will NOT study the algorithm later.

$\frac{1}{2} \equiv x$ where $0 \le x \le 25$. Think about it.
No such x exists.
Fact: A number y has an inverse mod 26 if y and 26 have no common factors. Numbers that have an inverse mod 26:

$$\{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$$

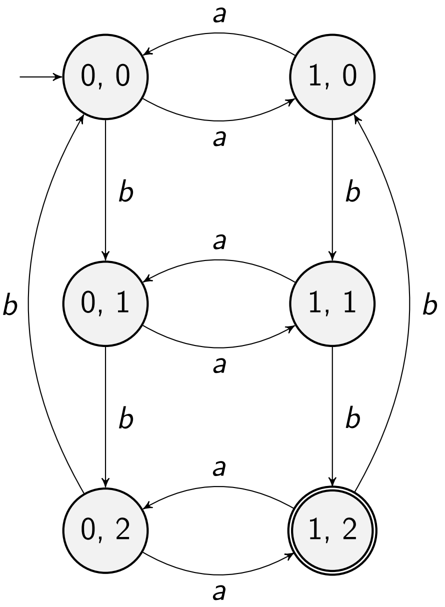
End of Detour

End of Detour

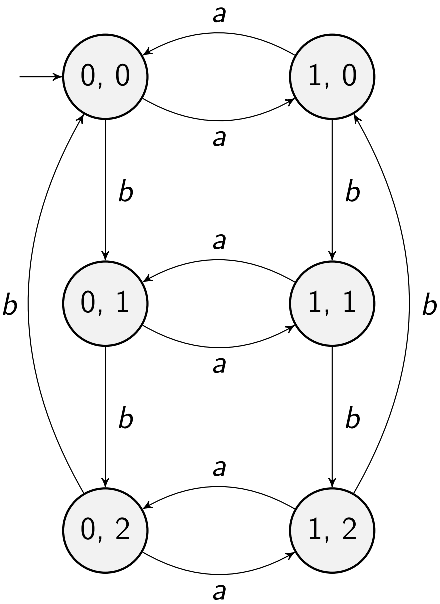
Another Example

$$\{w : \#_a(w) \equiv 1 \pmod 2 \wedge \#_b(w) \equiv 2 \pmod 3\}$$

$\{w : \#_a(w) \equiv 1 \pmod{2} \wedge \#_b(w) \equiv 2 \pmod{3}\}$



$((\#_a(w) \pmod 2), \#_b(w) \pmod 3))$



$((\#_a(w) \pmod 2), \#_b(w) \pmod 3))$

A DFA-classifier does not ACCEPT and REJECT. It classifies.

$((\#_a(w) \pmod 2), \#_b(w) \pmod 3))$

A DFA-classifier does not ACCEPT and REJECT. It classifies.
If w is fed to the DFA in the last slide, the resulting state is

$$(\#_a(w) \pmod 2), \#_b(w) \pmod 3))$$

$((\#_a(w) \pmod 2), \#_b(w) \pmod 3))$

A DFA-classifier does not ACCEPT and REJECT. It classifies. If w is fed to the DFA in the last slide, the resulting state is

$$(\#_a(w) \pmod 2, \#_b(w) \pmod 3)$$

The first DFA **accepted** $(1,2)$-strings and **rejected** the rest.

$((\#_a(w) \pmod 2, \#_b(w) \pmod 3))$

A DFA-classifier does not ACCEPT and REJECT. It classifies. If w is fed to the DFA in the last slide, the resulting state is

$$(\#_a(w) \pmod 2, \#_b(w) \pmod 3)$$

The first DFA **accepted** $(1, 2)$-strings and **rejected** the rest.

The second DFA **classifies** strings without judgment.

Short Detour

Short Detour

Alphabets, Strings, and Languages

Alphabets and Strings

Alphabets and Strings

Def An **alphabet** Σ is a set of letters (or characters).

Alphabets and Strings

Def An **alphabet** Σ is a set of letters (or characters).

- For Examples 1 and 2: $\Sigma = \{a, b\}$.

Alphabets and Strings

Def An **alphabet** Σ is a set of letters (or characters).

- For Examples 1 and 2: $\Sigma = \{a, b\}$.
- For Example 3: $\Sigma = \{$A, ..., Z, $\#$, .$\}$.

Alphabets and Strings

Def An **alphabet** Σ is a set of letters (or characters).

- For Examples 1 and 2: $\Sigma = \{a, b\}$.
- For Example 3: $\Sigma = \{$A, ..., Z, $\#$, .$\}$.

Def A **string** or **word** is a sequence of symbols from an alphabet Σ.

Alphabets and Strings

Def An **alphabet** Σ is a set of letters (or characters).

- For Examples 1 and 2: $\Sigma = \{a, b\}$.
- For Example 3: Σ = {A, ..., Z, #, .}.

Def A **string** or **word** is a sequence of symbols from an alphabet Σ.

- $\Sigma^2 = \Sigma\Sigma = \{\sigma_1\sigma_2 : \sigma_1 \in \Sigma \wedge \sigma_2 \in \Sigma\}$.

Alphabets and Strings

Def An **alphabet** Σ is a set of letters (or characters).

- For Examples 1 and 2: $\Sigma = \{a, b\}$.
- For Example 3: $\Sigma = \{$A, ..., Z, $\#$, .$\}$.

Def A **string** or **word** is a sequence of symbols from an alphabet Σ.

- $\Sigma^2 = \Sigma\Sigma = \{\sigma_1\sigma_2 : \sigma_1 \in \Sigma \wedge \sigma_2 \in \Sigma\}$.
- $\Sigma^3 = \Sigma\Sigma\Sigma = \{\sigma_1\sigma_2\sigma_3 : \sigma_1 \in \Sigma \wedge \sigma_2 \in \Sigma \wedge \sigma_3 \in \Sigma\}$.

Alphabets and Strings

Def An **alphabet** Σ is a set of letters (or characters).

- For Examples 1 and 2: $\Sigma = \{a, b\}$.
- For Example 3: $\Sigma = \{$A, ..., Z, $\#$, .$\}$.

Def A **string** or **word** is a sequence of symbols from an alphabet Σ.

- $\Sigma^2 = \Sigma\Sigma = \{\sigma_1\sigma_2 : \sigma_1 \in \Sigma \wedge \sigma_2 \in \Sigma\}$.
- $\Sigma^3 = \Sigma\Sigma\Sigma = \{\sigma_1\sigma_2\sigma_3 : \sigma_1 \in \Sigma \wedge \sigma_2 \in \Sigma \wedge \sigma_3 \in \Sigma\}$.
- $\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$

Alphabets and Strings

Def An **alphabet** Σ is a set of letters (or characters).

- For Examples 1 and 2: $\Sigma = \{a, b\}$.
- For Example 3: $\Sigma = \{$A, ..., Z, $\#$, .$\}$.

Def A **string** or **word** is a sequence of symbols from an alphabet Σ.

- $\Sigma^2 = \Sigma\Sigma = \{\sigma_1\sigma_2 : \sigma_1 \in \Sigma \wedge \sigma_2 \in \Sigma\}$.
- $\Sigma^3 = \Sigma\Sigma\Sigma = \{\sigma_1\sigma_2\sigma_3 : \sigma_1 \in \Sigma \wedge \sigma_2 \in \Sigma \wedge \sigma_3 \in \Sigma\}$.
- $\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$
- $i = 1$ case is just $\Sigma^1 = \Sigma$.

Alphabets and Strings

Def An **alphabet** Σ is a set of letters (or characters).

- For Examples 1 and 2: $\Sigma = \{a, b\}$.
- For Example 3: $\Sigma = \{\text{A, ..., Z, \#, .}\}$.

Def A **string** or **word** is a sequence of symbols from an alphabet Σ.

- $\Sigma^2 = \Sigma\Sigma = \{\sigma_1\sigma_2 : \sigma_1 \in \Sigma \wedge \sigma_2 \in \Sigma\}$.
- $\Sigma^3 = \Sigma\Sigma\Sigma = \{\sigma_1\sigma_2\sigma_3 : \sigma_1 \in \Sigma \wedge \sigma_2 \in \Sigma \wedge \sigma_3 \in \Sigma\}$.
- $\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$
- $i = 1$ case is just $\Sigma^1 = \Sigma$.
- $i = 0$ case is just $\Sigma^0 = \{e\}$ **(the empty string)**.

Alphabets and Strings

Def An **alphabet** Σ is a set of letters (or characters).

- For Examples 1 and 2: $\Sigma = \{a, b\}$.
- For Example 3: $\Sigma = \{\text{A}, \ldots, \text{Z}, \#, .\}$.

Def A **string** or **word** is a sequence of symbols from an alphabet Σ.

- $\Sigma^2 = \Sigma\Sigma = \{\sigma_1\sigma_2 : \sigma_1 \in \Sigma \wedge \sigma_2 \in \Sigma\}$.
- $\Sigma^3 = \Sigma\Sigma\Sigma = \{\sigma_1\sigma_2\sigma_3 : \sigma_1 \in \Sigma \wedge \sigma_2 \in \Sigma \wedge \sigma_3 \in \Sigma\}$.
- $\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$
- $i = 1$ case is just $\Sigma^1 = \Sigma$.
- $i = 0$ case is just $\Sigma^0 = \{e\}$ **(the empty string)**.
- **Notation Kleene star:** $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \cdots$ is the set of all strings over the alphabet Σ (including e).

Languages

Languages

Def A **language** over an alphabet Σ is a subset of Σ^*.

Languages

Def A **language** over an alphabet Σ is a subset of Σ^*.

Def Let M be a machine that accepts (or rejects) words. Then the **language** $L(M) = \{w : M \text{ accepts } w\}$.

Languages

Def A **language** over an alphabet Σ is a subset of Σ^*.

Def Let M be a machine that accepts (or rejects) words.
Then the **language** $L(M) = \{w : M \text{ accepts } w\}$.

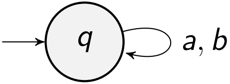
Draw the DFA that accepts the empty language over the alphabet $\{a, b\}$. I.e., $L = \{\}$.

Languages

Def A **language** over an alphabet Σ is a subset of Σ^*.

Def Let M be a machine that accepts (or rejects) words. Then the **language** $L(M) = \{w : M \text{ accepts } w\}$.

Draw the DFA that accepts the empty language over the alphabet $\{a, b\}$. I.e., $L = \{\}$.

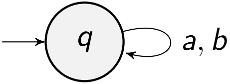


Languages

Def A **language** over an alphabet Σ is a subset of Σ^*.

Def Let M be a machine that accepts (or rejects) words. Then the **language** $L(M) = \{w : M \text{ accepts } w\}$.

Draw the DFA that accepts the empty language over the alphabet $\{a, b\}$. I.e., $L = \{\}$.



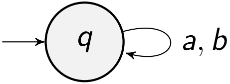
Draw the DFA that accepts the language L over the alphabet $\{a, b\}$ with only the empty word. I.e. $L = \{e\}$.

Languages

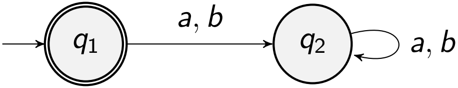
Def A **language** over an alphabet Σ is a subset of Σ^*.

Def Let M be a machine that accepts (or rejects) words. Then the **language** $L(M) = \{w : M \text{ accepts } w\}$.

Draw the DFA that accepts the empty language over the alphabet $\{a, b\}$. I.e., $L = \{\}$.



Draw the DFA that accepts the language L over the alphabet $\{a, b\}$ with only the empty word. I.e. $L = \{e\}$.



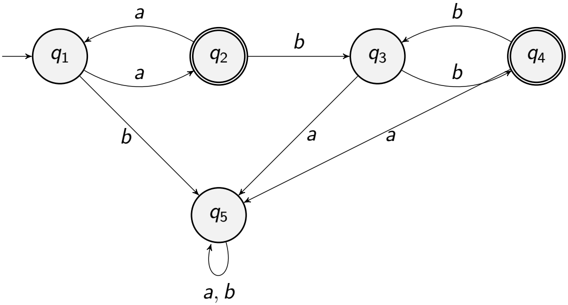
End of Detour

End of Detour

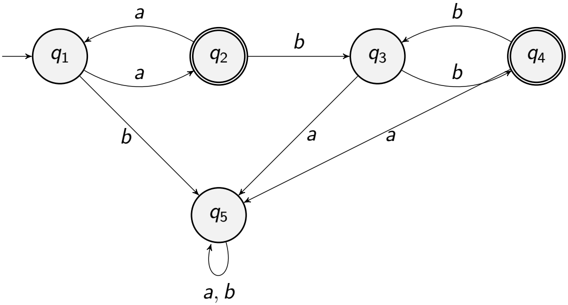
Start of Transition Tables

Recall Second Example

Recall Second Example

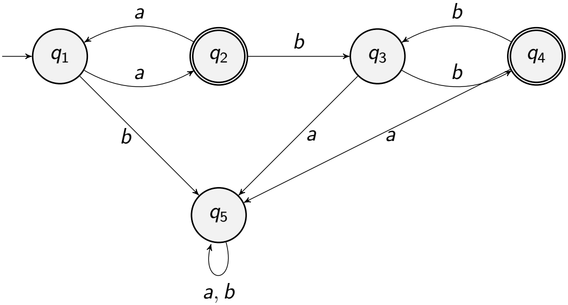


Recall Second Example



Transition Table:

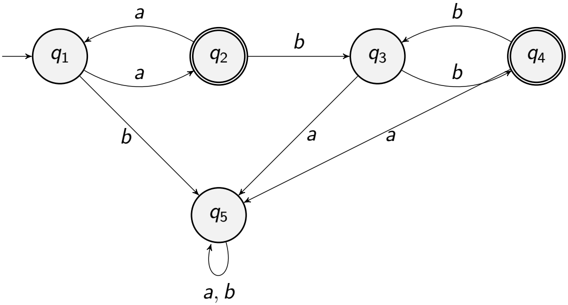
Recall Second Example



Transition Table:

- States: $\{q_1, q_2, q_3, q_4, q_5\}$

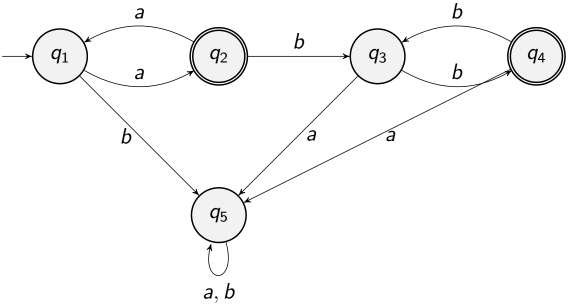
Recall Second Example



Transition Table:

- States: $\{q_1, q_2, q_3, q_4, q_5\}$
- Alphabet: $\{a, b\}$

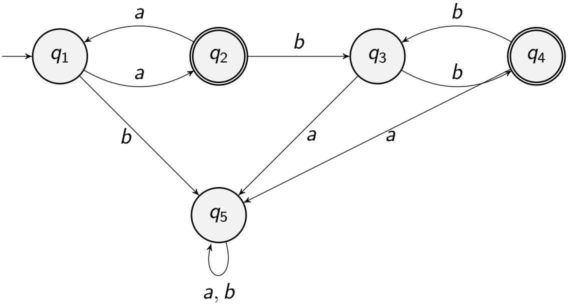
Recall Second Example



Transition Table:

- States: $\{q_1, q_2, q_3, q_4, q_5\}$
- Alphabet: $\{a, b\}$
- Start state: q_1

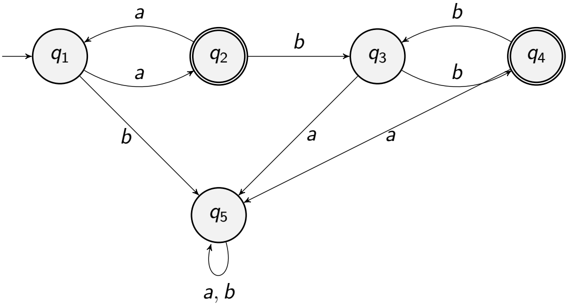
Recall Second Example



Transition Table:

- States: $\{q_1, q_2, q_3, q_4, q_5\}$
- Alphabet: $\{a, b\}$
- Start state: q_1
- Final states: $\{q_2, q_4\}$

Recall Second Example



Transition Table:

- States: $\{q_1, q_2, q_3, q_4, q_5\}$
- Alphabet: $\{a, b\}$
- Start state: q_1
- Final states: $\{q_2, q_4\}$

- Transition function

	a	b
q_1	q_2	q_5
q_2	q_1	q_3
q_3	q_5	q_4
q_4	q_5	q_3
q_5	q_5	q_5

Formal definition of DFAs

Formal definition of DFAs

Def A **DFA** M is a 5-tuple $(Q, \Sigma, \delta, s, F)$ where:

1. Q is a finite set of **states**.
2. Σ is a finite **alphabet**.
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**.
4. $s \in Q$ is the **start state**.
5. $F \subseteq Q$ is the set of **final states**.

Formal definition of DFAs

Def A **DFA** M is a 5-tuple $(Q, \Sigma, \delta, s, F)$ where:

1. Q is a finite set of **states**.
2. Σ is a finite **alphabet**.
3. $\delta : Q \times \Sigma \rightarrow Q$ is the **transition function**.
4. $s \in Q$ is the **start state**.
5. $F \subseteq Q$ is the set of **final states**.

Informally DFA M accepts w if when M is run on w it ends up in a final state.

Formal definition of DFAs

Def A **DFA** M is a 5-tuple $(Q, \Sigma, \delta, s, F)$ where:

1. Q is a finite set of **states**.
2. Σ is a finite **alphabet**.
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**.
4. $s \in Q$ is the **start state**.
5. $F \subseteq Q$ is the set of **final states**.

Informally DFA *M* accepts *w* if when M is run on w it ends up in a final state.

Formally
Def If M is a DFA and $x \in \Sigma^*$ is a word of length n, so $x = x_1 \cdots x_n$ where $x_i \in \Sigma$.
M* accepts *x if there is a sequence of states $q_0, q_1, \ldots, q_n$ such that $q_0 = s$, $q_i = \delta(q_{i-1}, x_i)$ for $1 \le i \le n$, and $q_n \in F$.

Formal definition of DFAs

Def A **DFA** M is a 5-tuple $(Q, \Sigma, \delta, s, F)$ where:

1. Q is a finite set of **states**.
2. Σ is a finite **alphabet**.
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**.
4. $s \in Q$ is the **start state**.
5. $F \subseteq Q$ is the set of **final states**.

Informally DFA M accepts w if when M is run on w it ends up in a final state.

Formally

Def If M is a DFA and $x \in \Sigma^*$ is a word of length n, so $x = x_1 \cdots x_n$ where $x_i \in \Sigma$.
M accepts x if there is a sequence of states $q_0, q_1, \ldots, q_n$ such that $q_0 = s$, $q_i = \delta(q_{i-1}, x_i)$ for $1 \le i \le n$, and $q_n \in F$.

Def Language $L \subseteq \Sigma^*$ is **regular** if there exists a DFA M such that $L(M) = L$.

Computer Implementation of DFAs

Recall Second Example

Recall Second Example

Transition Table:

- States: $\{q_1, q_2, q_3, q_4, q_5\}$
- Alphabet: $\{a, b\}$
- Start state: q_1
- Final states: $\{q_2, q_4\}$

- Transition function

	a	b
q_1	q_2	q_5
q_2	q_1	q_3
q_3	q_5	q_4
q_4	q_5	q_3
q_5	q_5	q_5

Recall Second Example

Transition Table:

- States: $\{q_1, q_2, q_3, q_4, q_5\}$
- Alphabet: $\{a, b\}$
- Start state: q_1
- Final states: $\{q_2, q_4\}$

- Transition function

	a	b
q_1	q_2	q_5
q_2	q_1	q_3
q_3	q_5	q_4
q_4	q_5	q_3
q_5	q_5	q_5

Implementation of Transition Table:

- States: $\{1, 2, 3, 4, 5\}$
- Alphabet: $\{1, 2\}$
- Start state: 1
- Final states: $\{2, 4\}$

- Transition function

	1	2
1	2	5
2	1	3
3	5	4
4	5	3
5	5	5

Recall Second Example

Transition Table:

- States: $\{q_1, q_2, q_3, q_4, q_5\}$
- Alphabet: $\{a, b\}$
- Start state: q_1
- Final states: $\{q_2, q_4\}$

- Transition function

	a	b
q_1	q_2	q_5
q_2	q_1	q_3
q_3	q_5	q_4
q_4	q_5	q_3
q_5	q_5	q_5

Implementation of Transition Table:

- States: $\{1, 2, 3, 4, 5\}$
- Alphabet: $\{1, 2\}$
- Start state: 1
- Final states: $\{2, 4\}$

- Transition function

	1	2
1	2	5
2	1	3
3	5	4
4	5	3
5	5	5

Linear time!

Diagrams Versus Transition Tables

Diagrams Versus Transition Tables

Finite state automata are essentially graphs. Same rules apply:

Diagrams Versus Transition Tables

Finite state automata are essentially graphs. Same rules apply:

- Diagrams are good for people to understand if the DFAs are small.

Diagrams Versus Transition Tables

Finite state automata are essentially graphs. Same rules apply:

- Diagrams are good for people to understand if the DFAs are small.
- Transition tables are good for algorithms and formal proofs.