BILL, RECORD LECTURE!!!!

BILL RECORD LECTURE!!!

Deterministic Finite Automata (DFA)

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

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

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

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

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

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

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

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

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

$\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$

$i = 1$ case is just $\Sigma^1 = \Sigma$.

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

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

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

$\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$

$i = 1$ case is just $\Sigma^1 = \Sigma$.

What about $i = 0$ case?

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

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

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

$\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$

$i = 1$ case is just $\Sigma^1 = \Sigma$.

What about $i = 0$ case?

$\Sigma^0 = \{e\}$, **the empty string**.

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

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

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

$\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$

$i = 1$ case is just $\Sigma^1 = \Sigma$.

What about $i = 0$ case?

$\Sigma^0 = \{e\}$, **the empty string**.

The empty string is useful for the same reason 0 and 1 are useful:

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

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

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

$\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$

$i = 1$ case is just $\Sigma^1 = \Sigma$.

What about $i = 0$ case?

$\Sigma^0 = \{e\}$, **the empty string**.

The empty string is useful for the same reason 0 and 1 are useful:
If $w \in \mathbb{R}$ then $w + 0 = w$.

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

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

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

$\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$

$i = 1$ case is just $\Sigma^1 = \Sigma$.

What about $i = 0$ case?

$\Sigma^0 = \{e\}$, **the empty string**.

The empty string is useful for the same reason 0 and 1 are useful:
If $w \in \mathbb{R}$ then $w + 0 = w$.
If $w \in \mathbb{R}$ then $w \times 1 = w$.

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

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

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

$\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$

$i = 1$ case is just $\Sigma^1 = \Sigma$.

What about $i = 0$ case?

$\Sigma^0 = \{e\}$, **the empty string**.

The empty string is useful for the same reason 0 and 1 are useful:
If $w \in \mathbb{R}$ then $w + 0 = w$.
If $w \in \mathbb{R}$ then $w \times 1 = w$.
If w is a string of a's and b's, then $w \cdot e = w$ (this is concatenation).

Alphabets and Strings

Σ will be our alphabet. Usually $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b\}$.
Sometimes $\Sigma = \{a, b, c\}$ or bigger.
A sequence of symbols of an alphabet is a **string**.

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

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

$\Sigma^i = \{\sigma_1 \cdots \sigma_i : \sigma_1, \ldots, \sigma_i \in \Sigma\}$

$i = 1$ case is just $\Sigma^1 = \Sigma$.

What about $i = 0$ case?

$\Sigma^0 = \{e\}$, **the empty string**.

The empty string is useful for the same reason 0 and 1 are useful:
If $w \in \mathbb{R}$ then $w + 0 = w$.
If $w \in \mathbb{R}$ then $w \times 1 = w$.
If w is a string of a's and b's, then $w \cdot e = w$ (this is concatenation).

Notation $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \cdots$. is the set of all strings including e.

Concatenation, number-of-a's, Prefix

Let $x, y \in \Sigma^*$. Then xy is the **concatenation of x and y**. We sometimes write it as $x \cdot y$.

Concatenation, number-of-a's, Prefix

Let $x, y \in \Sigma^*$. Then xy is the **concatenation of x and y**. We sometimes write it as $x \cdot y$.

If $A, B \subseteq \Sigma^*$ then $A \cdot B = \{x \cdot y : x \in A \wedge y \in B\}$.

Concatenation, number-of-a's, Prefix

Let $x, y \in \Sigma^*$. Then xy is the **concatenation of x and y**. We sometimes write it as $x \cdot y$.

If $A, B \subseteq \Sigma^*$ then $A \cdot B = \{x \cdot y : x \in A \wedge y \in B\}$.

Note that $\Sigma\Sigma$ is $\Sigma \cdot \Sigma$.

Concatenation, number-of-a's, Prefix

Let $x, y \in \Sigma^*$. Then xy is the **concatenation of x and y**. We sometimes write it as $x \cdot y$.

If $A, B \subseteq \Sigma^*$ then $A \cdot B = \{x \cdot y : x \in A \wedge y \in B\}$.

Note that $\Sigma\Sigma$ is $\Sigma \cdot \Sigma$.

If $x \in \{a, b\}^*$ then $\#_a(x)$ is the number of a's in x.

Concatenation, number-of-a's, Prefix

Let $x, y \in \Sigma^*$. Then xy is the **concatenation of x and y**. We sometimes write it as $x \cdot y$.

If $A, B \subseteq \Sigma^*$ then $A \cdot B = \{x \cdot y : x \in A \wedge y \in B\}$.

Note that $\Sigma\Sigma$ is $\Sigma \cdot \Sigma$.

If $x \in \{a, b\}^*$ then $\#_a(x)$ is the number of a's in x.
Same for $\#_b$, $\#_0$, etc.

Concatenation, number-of-a's, Prefix

Let $x, y \in \Sigma^*$. Then xy is the **concatenation of x and y**. We sometimes write it as $x \cdot y$.

If $A, B \subseteq \Sigma^*$ then $A \cdot B = \{x \cdot y : x \in A \wedge y \in B\}$.

Note that $\Sigma\Sigma$ is $\Sigma \cdot \Sigma$.

If $x \in \{a, b\}^*$ then $\#_a(x)$ is the number of a's in x.
Same for $\#_b$, $\#_0$, etc.

If $x, y \in \{a, b\}^*$ then $x \preceq y$ means that x is a prefix of y.

Concatenation, number-of-a's, Prefix

Let $x, y \in \Sigma^*$. Then xy is the **concatenation of x and y**. We sometimes write it as $x \cdot y$.

If $A, B \subseteq \Sigma^*$ then $A \cdot B = \{x \cdot y : x \in A \wedge y \in B\}$.

Note that $\Sigma\Sigma$ is $\Sigma \cdot \Sigma$.

If $x \in \{a, b\}^*$ then $\#_a(x)$ is the number of a's in x.
Same for $\#_b$, $\#_0$, etc.

If $x, y \in \{a, b\}^*$ then $x \preceq y$ means that x is a prefix of y.
For example, aab is a prefix of $aabbaaba$.

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 $0 \leq x \leq 25$.

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 $0 \leq x \leq 25$.
   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 $0 \leq x \leq 25$.
   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 $0 \le x \le 25$.
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:

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: 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:

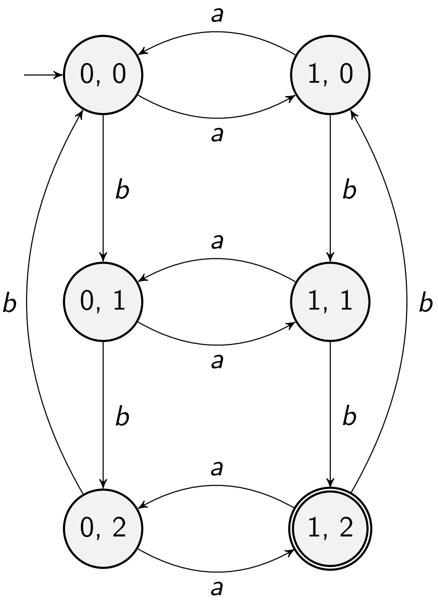
$$\{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$$

Examples of DFA's Before Formal Def

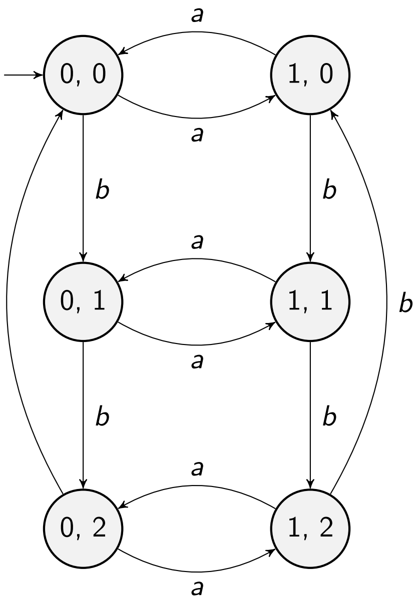
We do examples of DFA's before defining them formally.

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

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



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



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

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

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

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

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

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

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.

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

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.

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

Thm Any DFA for the lang has at least 6 states.
Proof Assume DFA M has ≤ 5 states.

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

Thm Any DFA for the lang has at least 6 states.
Proof Assume DFA M has ≤ 5 states.
On input e, the empty string, goes to state q_e.

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

Thm Any DFA for the lang has at least 6 states.
Proof Assume DFA M has ≤ 5 states.
On input e, the empty string, goes to state q_e.
On input a goes to state q_a.

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

Thm Any DFA for the lang has at least 6 states.
Proof Assume DFA M has ≤ 5 states.
On input e, the empty string, goes to state q_e.
On input a goes to state q_a.
On input b goes to state q_b.

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

Thm Any DFA for the lang has at least 6 states.
Proof Assume DFA M has ≤ 5 states.
On input e, the empty string, goes to state q_e.
On input a goes to state q_a.
On input b goes to state q_b.
On input bb goes to state q_{bb}.

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

Thm Any DFA for the lang has at least 6 states.
Proof Assume DFA M has ≤ 5 states.
On input e, the empty string, goes to state q_e.
On input a goes to state q_a.
On input b goes to state q_b.
On input bb goes to state q_{bb}.
On input ab goes to state q_{ab}.

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

Thm Any DFA for the lang has at least 6 states.
Proof Assume DFA M has ≤ 5 states.
On input e, the empty string, goes to state q_e.
On input a goes to state q_a.
On input b goes to state q_b.
On input bb goes to state q_{bb}.
On input ab goes to state q_{ab}.
On input abb goes to state q_{abb}.

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

Thm Any DFA for the lang has at least 6 states.
Proof Assume DFA M has ≤ 5 states.
On input e, the empty string, goes to state q_e.
On input a goes to state q_a.
On input b goes to state q_b.
On input bb goes to state q_{bb}.
On input ab goes to state q_{ab}.
On input abb goes to state q_{abb}.
Since ≤ 5 states two of these go to the same state, say q_{aa} and q_{bb}.

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

Thm Any DFA for the lang has at least 6 states.
Proof Assume DFA M has ≤ 5 states.
On input e, the empty string, goes to state q_e.
On input a goes to state q_a.
On input b goes to state q_b.
On input bb goes to state q_{bb}.
On input ab goes to state q_{ab}.
On input abb goes to state q_{abb}.
Since ≤ 5 states two of these go to the same state, say q_{aa} and q_{bb}.
$aa \cdot abb$ goes to state q which must accept since $aaabb \in L$.

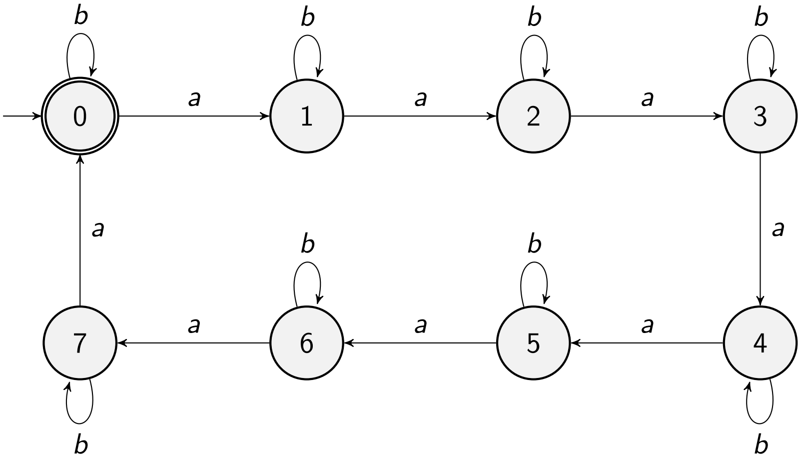
$\{w : \#_a(w) \equiv 1 \pmod 2 \wedge \#_b(w) \equiv 2 \pmod 3\}$

Thm Any DFA for the lang has at least 6 states.
Proof Assume DFA M has ≤ 5 states.
On input e, the empty string, goes to state q_e.
On input a goes to state q_a.
On input b goes to state q_b.
On input bb goes to state q_{bb}.
On input ab goes to state q_{ab}.
On input abb goes to state q_{abb}.
Since ≤ 5 states two of these go to the same state, say q_{aa} and q_{bb}.
$aa \cdot abb$ goes to state q which must accept since $aaabb \in L$.
$bb \cdot abb$ goes to state q which accepts. OH, but $bbabb \notin L$.
Contradiction.

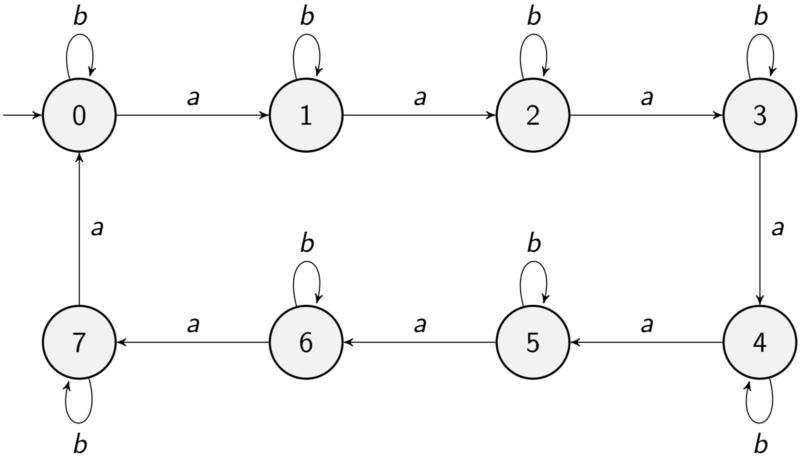
$\{w : \#_a(w) \equiv 1 \pmod 2 \wedge \#_b(w) \equiv 2 \pmod 3\}$

Thm Any DFA for the lang has at least 6 states.
Proof Assume DFA M has ≤ 5 states.
On input e, the empty string, goes to state q_e.
On input a goes to state q_a.
On input b goes to state q_b.
On input bb goes to state q_{bb}.
On input ab goes to state q_{ab}.
On input abb goes to state q_{abb}.
Since ≤ 5 states two of these go to the same state, say q_{aa} and q_{bb}.
$aa \cdot abb$ goes to state q which must accept since $aaabb \in L$.
$bb \cdot abb$ goes to state q which accepts. OH, but $bbabb \notin L$.
Contradiction.
Would need to do this argument with all pairs OR do it in a more general way. Might be on a HW, MIDTERM, or FINAL.

$\{w : \#_a(w) \equiv 0 \pmod 8\}$



DFA-Classifier for $\{w : \#_a(w) \equiv 0 \pmod 8\}$



$L = \{w : \#_a(w) \equiv 0 \pmod 8\}$

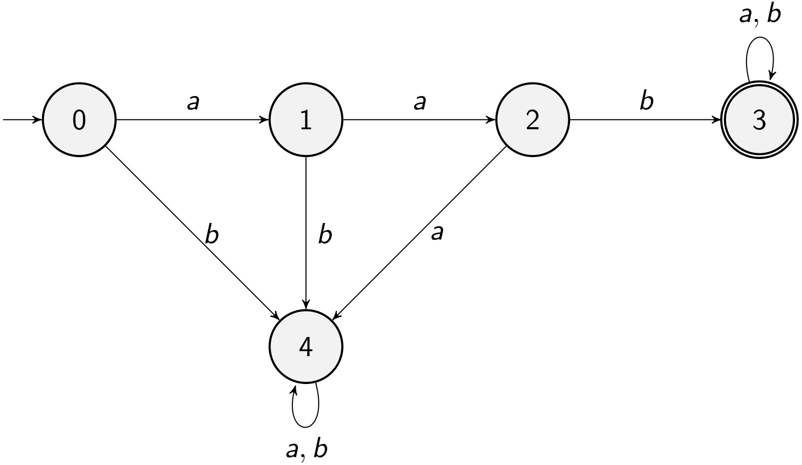
Thm Any DFA for L has at least 8 states.

$L = \{w : \#_a(w) \equiv 0 \pmod 8\}$

Thm Any DFA for L has at least 8 states.
Might be on a HW or exam.

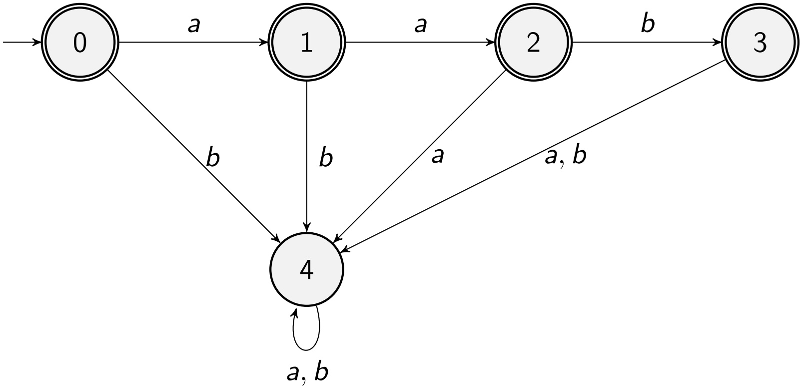
Example of DFA: $\{w : aab \preceq w\}$

Example of DFA: $\{w : aab \preceq w\}$

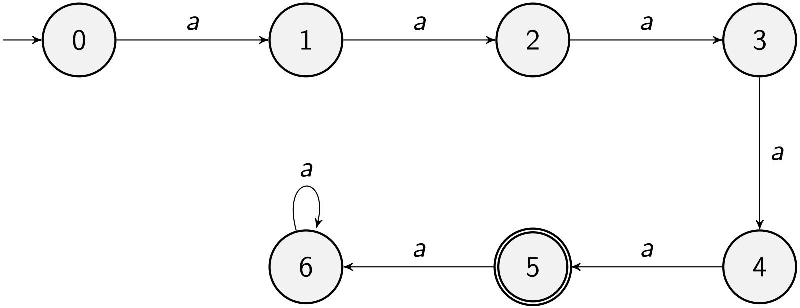


Example of DFA: $\{w : w \preceq aab\}$

Example of DFA: $\{w : w \preceq aab\}$



Example of DFA: $\{aaaaa\}$



Number of states: $\{aaaaa\}$

The DFA we drew for this had 7 states.
Thm Any DFA for this lang has ≥ 7 states.
Pf Assume there is a DFA with 6 states.

Number of states: $\{aaaaa\}$

The DFA we drew for this had 7 states.
Thm Any DFA for this lang has ≥ 7 states.
Pf Assume there is a DFA with 6 states.

Start state is q_0.

Number of states: $\{aaaaa\}$

The DFA we drew for this had 7 states.
Thm Any DFA for this lang has ≥ 7 states.
Pf Assume there is a DFA with 6 states.

Start state is q_0.

On input a end in q_1. From here a^4 gets to an accept.

Number of states: $\{aaaaa\}$

The DFA we drew for this had 7 states.
Thm Any DFA for this lang has ≥ 7 states.
Pf Assume there is a DFA with 6 states.

Start state is q_0.

On input a end in q_1. From here a^4 gets to an accept.

On input a^2 end in q_2. From here a^3 gets to an accept.

Number of states: $\{aaaaa\}$

The DFA we drew for this had 7 states.
Thm Any DFA for this lang has ≥ 7 states.
Pf Assume there is a DFA with 6 states.

Start state is q_0.

On input a end in q_1. From here a^4 gets to an accept.

On input a^2 end in q_2. From here a^3 gets to an accept.

On input a^3 end in q_3. From here a^2 gets to an accept.

Number of states: $\{aaaaa\}$

The DFA we drew for this had 7 states.
Thm Any DFA for this lang has ≥ 7 states.
Pf Assume there is a DFA with 6 states.

Start state is q_0.

On input a end in q_1. From here a^4 gets to an accept.

On input a^2 end in q_2. From here a^3 gets to an accept.

On input a^3 end in q_3. From here a^2 gets to an accept.

On input a^4 end in q_4. From here a^1 gets to an accept.

Number of states: $\{aaaaa\}$

The DFA we drew for this had 7 states.
Thm Any DFA for this lang has ≥ 7 states.
Pf Assume there is a DFA with 6 states.

Start state is q_0.

On input a end in q_1. From here a^4 gets to an accept.

On input a^2 end in q_2. From here a^3 gets to an accept.

On input a^3 end in q_3. From here a^2 gets to an accept.

On input a^4 end in q_4. From here a^1 gets to an accept.

On input a^5 end in q_5 which accepts.

Number of states: $\{aaaaa\}$

The DFA we drew for this had 7 states.
Thm Any DFA for this lang has ≥ 7 states.
Pf Assume there is a DFA with 6 states.

Start state is q_0.

On input a end in q_1. From here a^4 gets to an accept.

On input a^2 end in q_2. From here a^3 gets to an accept.

On input a^3 end in q_3. From here a^2 gets to an accept.

On input a^4 end in q_4. From here a^1 gets to an accept.

On input a^5 end in q_5 which accepts.

On input a^6 end in q_6.

Number of states: $\{aaaaa\}$

The DFA we drew for this had 7 states.
Thm Any DFA for this lang has ≥ 7 states.
Pf Assume there is a DFA with 6 states.

Start state is q_0.

On input a end in q_1. From here a^4 gets to an accept.

On input a^2 end in q_2. From here a^3 gets to an accept.

On input a^3 end in q_3. From here a^2 gets to an accept.

On input a^4 end in q_4. From here a^1 gets to an accept.

On input a^5 end in q_5 which accepts.

On input a^6 end in q_6.

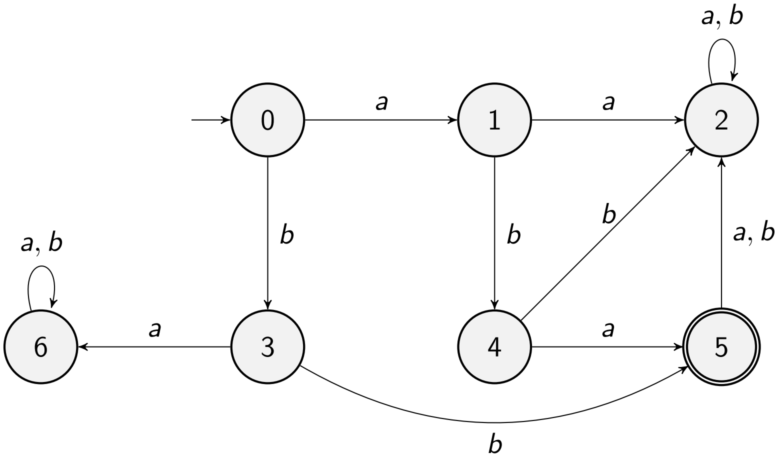
Two of q_i, q_j are the same state. See next slide.

Continuing proof

Assume $i < j$ and $q_i = q_j = q$.
Note that $i \leq 5$.
Input a^i ends in state q_i.
Input a^j ends in state q_j.
$a^i a^{5-i} = a^5$ ends in ACCEPT state.
$a^j a^{5-i} = a^{5+j-i}$ ends in REJECT state since $5 + j - i > 5$.
But these strings end in SAME state, so contradiction.

Example of DFA: $\{bb, aba\}$

Example of DFA: $\{bb, aba\}$



Any Finite Set can be recognized by a DFA

Let L be a finite set. Let the longest string in L be of length n.

Any Finite Set can be recognized by a DFA

Let L be a finite set. Let the longest string in L be of length n.

Draw a DFA with a diff state for every string of length $\leq n$.

Any Finite Set can be recognized by a DFA

Let L be a finite set. Let the longest string in L be of length n.

Draw a DFA with a diff state for every string of length $\leq n$.

Make the states for strings in L accept states.

Any Finite Set can be recognized by a DFA

Let L be a finite set. Let the longest string in L be of length n.

Draw a DFA with a diff state for every string of length $\leq n$.

Make the states for strings in L accept states.

This will take $\sim 2^n$ states. For many finite sets can do it with far fewer states.

DFA Intuitively

1. A DFA reads the input a letter at a time and never looks at it again. So one-scan.
2. A DFA only has a finite number of states, so $O(1)$ memory.
3. Contrast:
   3.1 A DFA can keep track of $\#_a(w) \pmod{17}$.
   3.2 A DFA cannot keep track of $\#_a(w)$.

DFA Formally

Def A **DFA** is a 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**.

DFA Formally

Def A **DFA** is a 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**.

Def If M is a DFA and $x \in \Sigma^*$ then $\boldsymbol{M(x)}$ **accepts** if when you run M on x you end up in a **final state**.

DFA Formally

Def A **DFA** is a 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**.

Def If M is a DFA and $x \in \Sigma^*$ then $\boldsymbol{M(x)}$ **accepts** if when you run M on x you end up in a **final state**.

Def If M is a DFA then $L(M) = \{x : M(x) \text{ accepts}\}$.

DFA Formally

Def A **DFA** is a 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**.

Def If M is a DFA and $x \in \Sigma^*$ then $\boldsymbol{M(x)}$ **accepts** if when you run M on x you end up in a **final state**.
Def If M is a DFA then $L(M) = \{x : M(x) \text{ accepts}\}$.
Def Let $L \subseteq \Sigma^*$. If there exists a DFA M such that $L(M) = L$ then L is **regular**.

Can Represent DFA's as Diagram or Transition Table

- If it's a particular example and not too many states, like those drawn a few slides ago, then draw it.

Can Represent DFA's as Diagram or Transition Table

- If it's a particular example and not too many states, like those drawn a few slides ago, then draw it.
- If it is many states or a general case (next slide) then give the transition table (the definition of δ).

$\{w : \#_a(w) \equiv 0 \pmod{n} \wedge \#_b \equiv 0 \pmod{m}\}$

$$Q = \{0, \ldots, n-1\} \times \{0, \ldots, m-1\}$$

$\{w : \#_a(w) \equiv 0 \pmod{n} \wedge \#_b \equiv 0 \pmod{m}\}$

$$Q = \{0, \ldots, n-1\} \times \{0, \ldots, m-1\}$$
$$s = (0, 0)$$

$\{w : \#_a(w) \equiv 0 \pmod{n} \wedge \#_b \equiv 0 \pmod{m}\}$

$$Q = \{0, \ldots, n-1\} \times \{0, \ldots, m-1\}$$
$$s = (0, 0)$$
$$F = \{(0, 0)\}$$

$\{w : \#_a(w) \equiv 0 \pmod{n} \wedge \#_b \equiv 0 \pmod{m}\}$

$$Q = \{0, \ldots, n-1\} \times \{0, \ldots, m-1\}$$
$$s = (0, 0)$$
$$F = \{(0, 0)\}$$
$$\delta((i, j), a) = (i + 1 \pmod{n}, j).$$
$$\delta((i, j), b) = (i, j + 1 \pmod{m}).$$

$\{w : \#_a(w) \equiv 0 \pmod{n} \wedge \#_b \equiv 0 \pmod{m}\}$

$Q = \{0, \ldots, n-1\} \times \{0, \ldots, m-1\}$

$s = (0, 0)$

$F = \{(0, 0)\}$

$\delta((i, j), a) = (i + 1 \pmod{n}, j)$.
$\delta((i, j), b) = (i, j + 1 \pmod{m})$.

Number of states is nm. Is there a DFA for this lang with a smaller DFA?

$\{w : \#_a(w) \equiv 0 \pmod{n} \wedge \#_b \equiv 0 \pmod{m}\}$

$Q = \{0, \ldots, n-1\} \times \{0, \ldots, m-1\}$

$s = (0, 0)$

$F = \{(0, 0)\}$

$\delta((i, j), a) = (i + 1 \pmod{n}, j)$.
$\delta((i, j), b) = (i, j + 1 \pmod{m})$.

Number of states is nm. Is there a DFA for this lang with a smaller DFA?
No.

$\{w : \#_a(w) \equiv 0 \pmod{n} \wedge \#_b \equiv 0 \pmod{m}\}$

$Q = \{0, \ldots, n-1\} \times \{0, \ldots, m-1\}$

$s = (0, 0)$

$F = \{(0, 0)\}$

$\delta((i, j), a) = (i + 1 \pmod{n}, j)$.
$\delta((i, j), b) = (i, j + 1 \pmod{m})$.

Number of states is nm. Is there a DFA for this lang with a smaller DFA?
No. We may prove this later in the term.