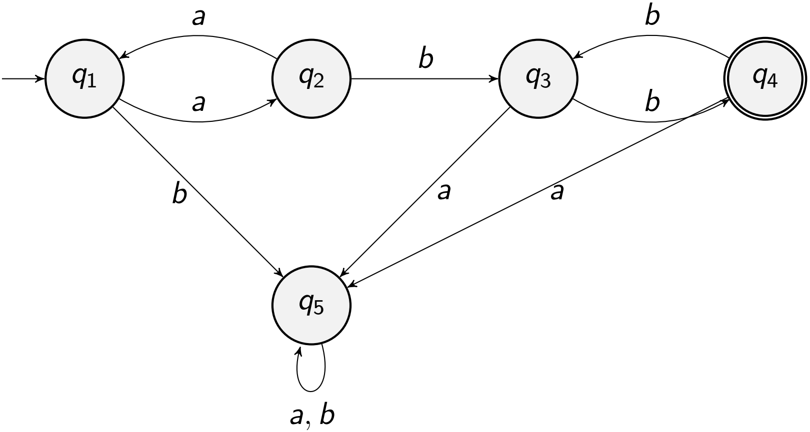
Review for CMSC 452 Midterm

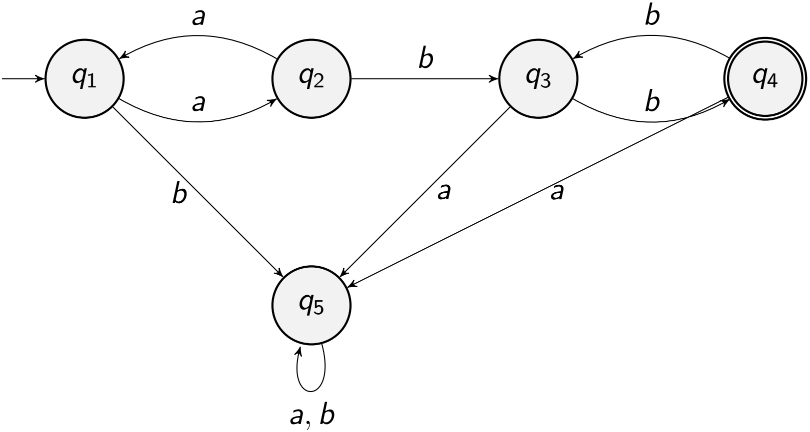
Deterministic Finite Automata (DFA)

DFA Diagram

DFA Diagram

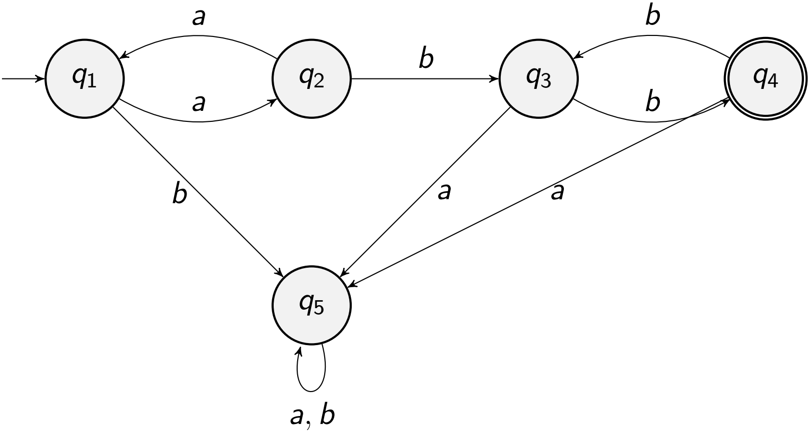


DFA Diagram



What is the language?

DFA Diagram

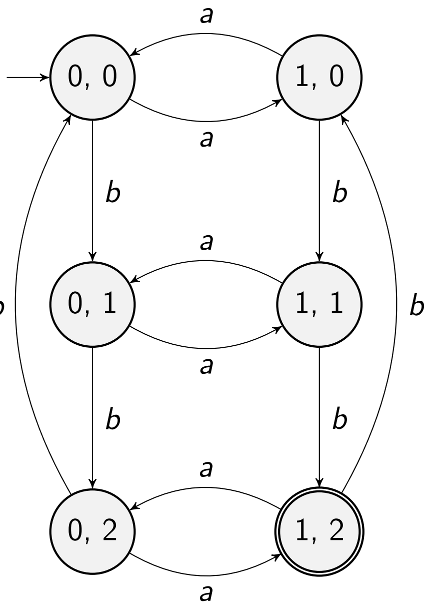


What is the language?

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

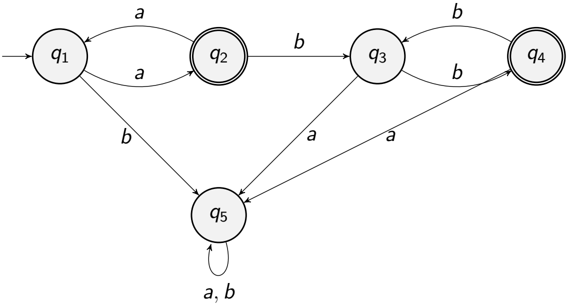
$$\{w : \#_a(w) \equiv 1 \pmod 2 \wedge \#_b(w) \equiv 2 \pmod 3\}$$

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

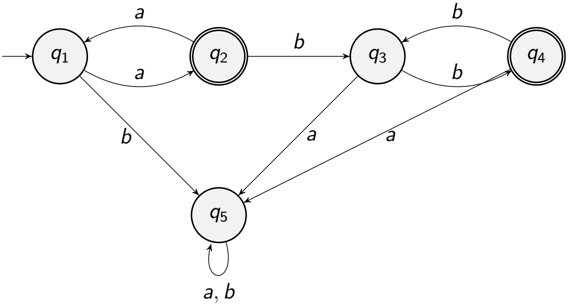


Transition Table

Transition Table

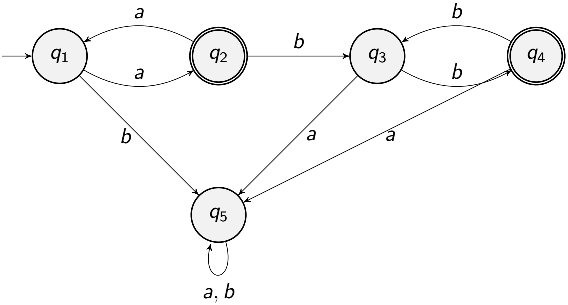


Transition Table



Transition Table:

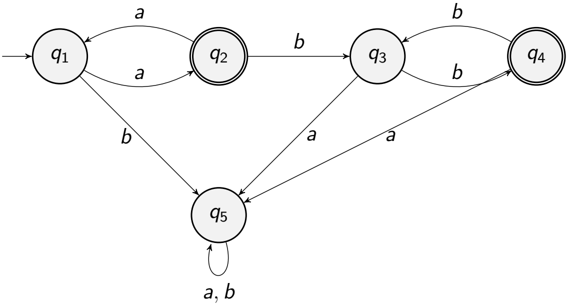
Transition Table



Transition Table:

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

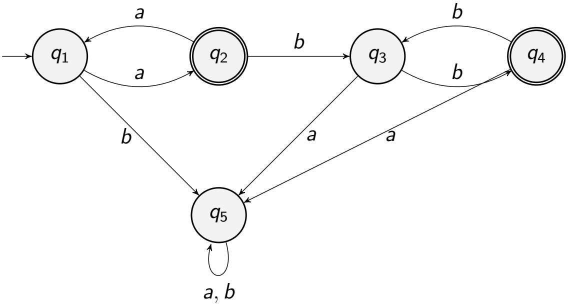
Transition Table



Transition Table:

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

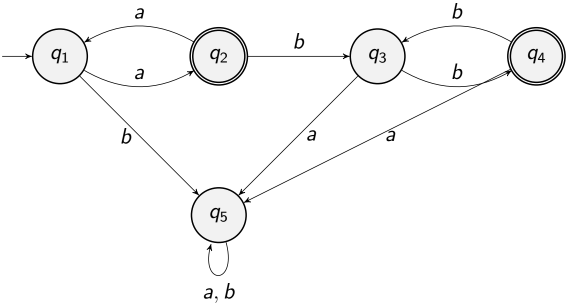
Transition Table



Transition Table:

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

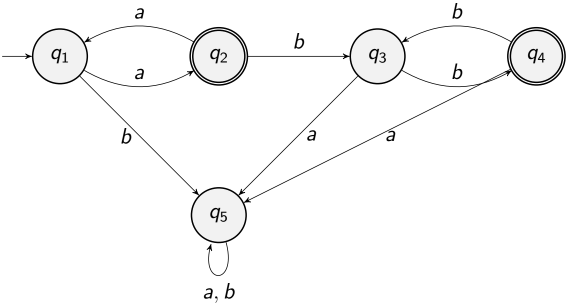
Transition Table



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 Table



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

Divisibility

Divisibility

We get a DFA (a trick?) for Mod 11.

Trick for Mod 11. All $\equiv$ are Mod 11

Is there a trick for mod 11?

Trick for Mod 11. All $\equiv$ are Mod 11

Is there a trick for mod 11?
We derive it together!

Trick for Mod 11. All $\equiv$ are Mod 11

Is there a trick for mod 11?
We derive it together!
$10^0 \equiv 1$

Trick for Mod 11. All $\equiv$ are Mod 11

Is there a trick for mod 11?
We derive it together!
$10^0 \equiv 1$
$10^1 \equiv 10 \equiv -1$

Trick for Mod 11. All $\equiv$ are Mod 11

Is there a trick for mod 11?
We derive it together!
$10^0 \equiv 1$
$10^1 \equiv 10 \equiv -1$
$10^2 \equiv 10 \equiv 10 \equiv -1 \times -1 \equiv 1.$

Trick for Mod 11. All $\equiv$ are Mod 11

Is there a trick for mod 11?
We derive it together!
$10^0 \equiv 1$
$10^1 \equiv 10 \equiv -1$
$10^2 \equiv 10 \equiv 10 \equiv -1 \times -1 \equiv 1.$
$10^3 \equiv 10^2 \times 10 \equiv 1 \times -1 \equiv -1.$
Pattern is $1, -1, 1, -1, \ldots$.

Trick for Mod 11. All $\equiv$ are Mod 11

Is there a trick for mod 11?
We derive it together!
$10^0 \equiv 1$
$10^1 \equiv 10 \equiv -1$
$10^2 \equiv 10 \equiv 10 \equiv -1 \times -1 \equiv 1.$
$10^3 \equiv 10^2 \times 10 \equiv 1 \times -1 \equiv -1.$
Pattern is $1, -1, 1, -1, \ldots.$

Thm $d_n \cdots d_0 \equiv d_0 - d_1 + d_2 - \cdots \pm d_n.$

Trick for Mod 11. All $\equiv$ are Mod 11

Is there a trick for mod 11?
We derive it together!
$10^0 \equiv 1$
$10^1 \equiv 10 \equiv -1$
$10^2 \equiv 10 \equiv 10 \equiv -1 \times -1 \equiv 1.$
$10^3 \equiv 10^2 \times 10 \equiv 1 \times -1 \equiv -1.$
Pattern is $1, -1, 1, -1, \ldots$.

Thm $d_n \cdots d_0 \equiv d_0 - d_1 + d_2 - \cdots \pm d_n$.

Proof may be on HW or Midterm or Final or some combination.

DFA for Mod 11

Need to keep track of both the running weighted sum mod 11 and if you are reading an even or odd place.

DFA for Mod 11

Need to keep track of both the running weighted sum mod 11 and if you are reading an even or odd place.

$Q = \{0, \ldots, 10\} \times \{0, 1\}$

DFA for Mod 11

Need to keep track of both the running weighted sum mod 11 and if you are reading an even or odd place.

$Q = \{0, \ldots, 10\} \times \{0, 1\}$

$s = (0, 0)$.

DFA for Mod 11

Need to keep track of both the running weighted sum mod 11 and if you are reading an even or odd place.

$Q = \{0, \ldots, 10\} \times \{0, 1\}$

$s = (0, 0)$.

Final state: Not going to have these, this is DFA-classifier.

DFA for Mod 11

Need to keep track of both the running weighted sum mod 11 and if you are reading an even or odd place.

$Q = \{0, \ldots, 10\} \times \{0, 1\}$

$s = (0, 0)$.

Final state: Not going to have these, this is DFA-classifier.

$$\delta((i,j),\sigma) \begin{cases} (i+\sigma \pmod{11}, j+1 \pmod{2}) \text{ if } j=0 \\ (i-\sigma \pmod{11}, j+1 \pmod{2}) \text{ if } j=1 \end{cases} \tag{1}$$

DFA for Mod 11

Need to keep track of both the running weighted sum mod 11 and if you are reading an even or odd place.

$Q = \{0, \ldots, 10\} \times \{0, 1\}$

$s = (0, 0)$.

Final state: Not going to have these, this is DFA-classifier.

$$\delta((i,j),\sigma) \begin{cases} (i+\sigma \pmod{11}, j+1 \pmod{2}) \text{ if } j=0 \\ (i-\sigma \pmod{11}, j+1 \pmod{2}) \text{ if } j=1 \end{cases} \tag{1}$$

We keep track of a running weighted sum mod 11 and position of the symbol mod 2.

DFA for Mod 11

Need to keep track of both the running weighted sum mod 11 and if you are reading an even or odd place.

$Q = \{0, \ldots, 10\} \times \{0, 1\}$

$s = (0, 0)$.

Final state: Not going to have these, this is DFA-classifier.

$$\delta((i,j),\sigma) \begin{cases} (i+\sigma \pmod{11}, j+1 \pmod{2}) \text{ if } j=0 \\ (i-\sigma \pmod{11}, j+1 \pmod{2}) \text{ if } j=1 \end{cases} \tag{1}$$

We keep track of a running weighted sum mod 11 and position of the symbol mod 2.

22 states.

DFA for Mod 11

Need to keep track of both the running weighted sum mod 11 and if you are reading an even or odd place.

$Q = \{0, \ldots, 10\} \times \{0, 1\}$

$s = (0, 0)$.

Final state: Not going to have these, this is DFA-classifier.

$$\delta((i,j),\sigma) \begin{cases} (i+\sigma \pmod{11}, j+1 \pmod{2}) \text{ if } j=0 \\ (i-\sigma \pmod{11}, j+1 \pmod{2}) \text{ if } j=1 \end{cases} \tag{1}$$

We keep track of a running weighted sum mod 11 and position of the symbol mod 2.

22 states.

Classifier If end in $(i, 0)$ or $(i, 1)$ then number is $\equiv i$.

Mods Can Get More Complicated: Mod 224

$n = 224$.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.

Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.
Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.
Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.
Start State s. $\delta(s, \sigma) = (\sigma, 0)$.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.
Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.
Start State s. $\delta(s, \sigma) = (\sigma, 0)$.
$\delta((i, 0), \sigma) = (i + 10\sigma \pmod{224}, 1)$.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.

Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.
Start State s. $\delta(s, \sigma) = (\sigma, 0)$.
$\delta((i, 0), \sigma) = (i + 10\sigma \pmod{224}, 1)$.
$\delta((i, 1), \sigma) = (i + 100\sigma \pmod{224}, 2)$.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.

Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.
Start State s. $\delta(s, \sigma) = (\sigma, 0)$.
$\delta((i, 0), \sigma) = (i + 10\sigma \pmod{224}, 1)$.
$\delta((i, 1), \sigma) = (i + 100\sigma \pmod{224}, 2)$.
$\delta((i, 2), \sigma) = (i + 104\sigma \pmod{224}, 3)$.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.

Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.
Start State s. $\delta(s, \sigma) = (\sigma, 0)$.
$\delta((i, 0), \sigma) = (i + 10\sigma \pmod{224}, 1)$.
$\delta((i, 1), \sigma) = (i + 100\sigma \pmod{224}, 2)$.
$\delta((i, 2), \sigma) = (i + 104\sigma \pmod{224}, 3)$.
$\delta((i, 3), \sigma) = (i + 144\sigma \pmod{224}, 0, g)$.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.

Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.
Start State s. $\delta(s, \sigma) = (\sigma, 0)$.
$\delta((i, 0), \sigma) = (i + 10\sigma \pmod{224}, 1)$.
$\delta((i, 1), \sigma) = (i + 100\sigma \pmod{224}, 2)$.
$\delta((i, 2), \sigma) = (i + 104\sigma \pmod{224}, 3)$.
$\delta((i, 3), \sigma) = (i + 144\sigma \pmod{224}, 0, g)$.
$\delta((i, 0, g), \sigma) = (i + 96\sigma \pmod{224}, 1, g)$.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.

Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.
Start State s. $\delta(s, \sigma) = (\sigma, 0)$.
$\delta((i, 0), \sigma) = (i + 10\sigma \pmod{224}, 1)$.
$\delta((i, 1), \sigma) = (i + 100\sigma \pmod{224}, 2)$.
$\delta((i, 2), \sigma) = (i + 104\sigma \pmod{224}, 3)$.
$\delta((i, 3), \sigma) = (i + 144\sigma \pmod{224}, 0, g)$.
$\delta((i, 0, g), \sigma) = (i + 96\sigma \pmod{224}, 1, g)$.
$\delta((i, 1, g), \sigma) = (i + 64\sigma \pmod{224}, 2, g)$.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.

Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.
Start State s. $\delta(s, \sigma) = (\sigma, 0)$.
$\delta((i, 0), \sigma) = (i + 10\sigma \pmod{224}, 1)$.
$\delta((i, 1), \sigma) = (i + 100\sigma \pmod{224}, 2)$.
$\delta((i, 2), \sigma) = (i + 104\sigma \pmod{224}, 3)$.
$\delta((i, 3), \sigma) = (i + 144\sigma \pmod{224}, 0, g)$.
$\delta((i, 0, g), \sigma) = (i + 96\sigma \pmod{224}, 1, g)$.
$\delta((i, 1, g), \sigma) = (i + 64\sigma \pmod{224}, 2, g)$.
$\delta((i, 2, g), \sigma) = (i + 192\sigma \pmod{224}, 3, g)$.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.
Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.
Start State s. $\delta(s, \sigma) = (\sigma, 0)$.
$\delta((i, 0), \sigma) = (i + 10\sigma \pmod{224}, 1)$.
$\delta((i, 1), \sigma) = (i + 100\sigma \pmod{224}, 2)$.
$\delta((i, 2), \sigma) = (i + 104\sigma \pmod{224}, 3)$.
$\delta((i, 3), \sigma) = (i + 144\sigma \pmod{224}, 0, g)$.
$\delta((i, 0, g), \sigma) = (i + 96\sigma \pmod{224}, 1, g)$.
$\delta((i, 1, g), \sigma) = (i + 64\sigma \pmod{224}, 2, g)$.
$\delta((i, 2, g), \sigma) = (i + 192\sigma \pmod{224}, 3, g)$.
$\delta((i, 3, g), \sigma) = (i + 121\sigma \pmod{224}, 4, g)$.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.

Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.
Start State s. $\delta(s, \sigma) = (\sigma, 0)$.
$\delta((i, 0), \sigma) = (i + 10\sigma \pmod{224}, 1)$.
$\delta((i, 1), \sigma) = (i + 100\sigma \pmod{224}, 2)$.
$\delta((i, 2), \sigma) = (i + 104\sigma \pmod{224}, 3)$.
$\delta((i, 3), \sigma) = (i + 144\sigma \pmod{224}, 0, g)$.
$\delta((i, 0, g), \sigma) = (i + 96\sigma \pmod{224}, 1, g)$.
$\delta((i, 1, g), \sigma) = (i + 64\sigma \pmod{224}, 2, g)$.
$\delta((i, 2, g), \sigma) = (i + 192\sigma \pmod{224}, 3, g)$.
$\delta((i, 3, g), \sigma) = (i + 121\sigma \pmod{224}, 4, g)$.
$\delta((i, 4, g), \sigma) = (i + 160\sigma \pmod{224}, 5, g)$.

Mods Can Get More Complicated: Mod 224

$n = 224$. Pattern is
$(1, 10, 100, 104, 144, \overline{96, 64, 192, 121, 160, 32})$

We define δ first. Then the Q will be all the states we encountered.

Plan Tiers for the $1, 10, 100, 104$ weights then a grid machine.
Distinguish tier-states from grid-states with marker g.
Start State s. $\delta(s, \sigma) = (\sigma, 0)$.
$\delta((i, 0), \sigma) = (i + 10\sigma \pmod{224}, 1)$.
$\delta((i, 1), \sigma) = (i + 100\sigma \pmod{224}, 2)$.
$\delta((i, 2), \sigma) = (i + 104\sigma \pmod{224}, 3)$.
$\delta((i, 3), \sigma) = (i + 144\sigma \pmod{224}, 0, g)$.
$\delta((i, 0, g), \sigma) = (i + 96\sigma \pmod{224}, 1, g)$.
$\delta((i, 1, g), \sigma) = (i + 64\sigma \pmod{224}, 2, g)$.
$\delta((i, 2, g), \sigma) = (i + 192\sigma \pmod{224}, 3, g)$.
$\delta((i, 3, g), \sigma) = (i + 121\sigma \pmod{224}, 4, g)$.
$\delta((i, 4, g), \sigma) = (i + 160\sigma \pmod{224}, 5, g)$.
$\delta((i, 5, g), \sigma) = (i + 32\sigma \pmod{224}, 0, g)$.

Nondeterministic Finite Automata (NFA)

NFA's Intuitively

1. An NFA is a DFA that can guess.
2. NFAs do not really exist.
3. Good for $\cup$ since can guess which one.
4. An NFA accepts iff SOME guess accepts.

Every NFA-lang a DFA-lang!

Thm If L is accepted by an NFA then L is accepted by a DFA.
Pf Sketch L is accepted by NFA $(Q, \Sigma, \Delta, s, F)$ where

1. Get rid of e-transitions using reachability.
2. Get rid of non-determinism by using power sets. Possibly 2^n blowup.

Regular Expressions

Examples

1. $b^*(ab^*ab^*)^*ab^*$
2. $b^*(ab^*ab^*ab^*)^*$
3. $(b^*(ab^*ab^*)^*ab^*) \cup (b^*(ab^*ab^*ab^*)^*)$

Every Regex-Lang is an NFA-Lang

Lemma If a language is generated by a regular expression, it is recognized by an NFA.

Every Regex-Lang is an NFA-Lang

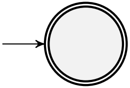
Lemma If a language is generated by a regular expression, it is recognized by an NFA.
Pf By **strong induction** on the length of α.

Every Regex-Lang is an NFA-Lang

Lemma If a language is generated by a regular expression, it is recognized by an NFA.
Pf By **strong induction** on the length of α.
Base Cases $|\alpha| = 1$. Then $\alpha = e$ or $\alpha = \sigma$.

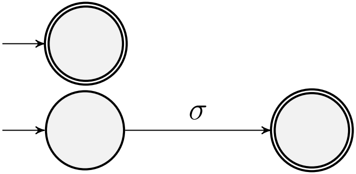
Every Regex-Lang is an NFA-Lang

Lemma If a language is generated by a regular expression, it is recognized by an NFA.
Pf By **strong induction** on the length of α.
Base Cases $|\alpha| = 1$. Then $\alpha = e$ or $\alpha = \sigma$.



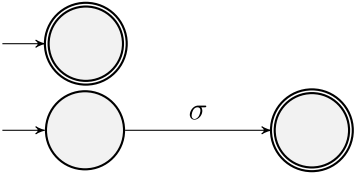
Every Regex-Lang is an NFA-Lang

Lemma If a language is generated by a regular expression, it is recognized by an NFA.
Pf By **strong induction** on the length of α.
Base Cases $|\alpha| = 1$. Then $\alpha = e$ or $\alpha = \sigma$.



Every Regex-Lang is an NFA-Lang

Lemma If a language is generated by a regular expression, it is recognized by an NFA.
Pf By **strong induction** on the length of α.
Base Cases $|\alpha| = 1$. Then $\alpha = e$ or $\alpha = \sigma$.



We skip rest of the proof.

DFA $\subseteq$ REGEX

Given a DFA M we want a Regex for $L(M)$.

DFA $\subseteq$ REGEX

Given a DFA M we want a Regex for $L(M)$.

Key We will find, for every pair of states (i, j) the regex that represents strings that take you from state i to state j.
Will assume M has state set $\{1, \ldots, n\}$.

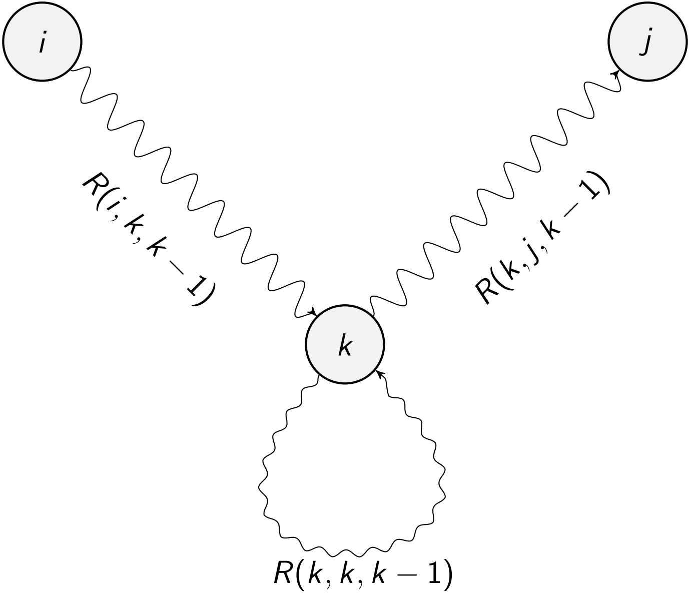
DFA $\subseteq$ REGEX

Given a DFA M we want a Regex for $L(M)$.

Key We will find, for every pair of states (i, j) the regex that represents strings that take you from state i to state j.
Will assume M has state set $\{1, \ldots, n\}$.

$R(i, j, k) = \{w : \delta(i, w) = j$ but only use states in $\{1, \ldots, k\}$ $\}$.

Inductive Step $R(i, j, k)$ as a Picture



Complete Proof on One Slide

For all $1 \leq i, j \leq n$:

$$R(i,j,0) = \begin{cases} \{\sigma : \delta(i,\sigma) = j\} & \text{if } i \neq j \ \} \\ \{\sigma : \delta(i,\sigma) = j\} \cup \{e\} & \text{if } i = j \ \} \end{cases} \qquad (2)$$

Complete Proof on One Slide

For all $1 \leq i, j \leq n$:

$$R(i,j,0) = \begin{cases} \{\sigma : \delta(i,\sigma) = j\} & \text{if } i \neq j \ \} \\ \{\sigma : \delta(i,\sigma) = j\} \cup \{e\} & \text{if } i = j \ \} \end{cases} \qquad (2)$$

All $R(i,j,0)$ are Regex.

Complete Proof on One Slide

For all $1 \leq i, j \leq n$:

$$R(i,j,0) = \begin{cases} \{\sigma : \delta(i,\sigma) = j\} & \text{if } i \neq j \ \} \\ \{\sigma : \delta(i,\sigma) = j\} \cup \{e\} & \text{if } i = j \ \} \end{cases} \quad (2)$$

All $R(i,j,0)$ are Regex.

For all $1 \leq i, j \leq n$ and all $0 \leq k \leq n$

$$R(i,j,k) = R(i,j,k-1) \bigcup R(i,k,k-1) R(k,k,k-1)^* R(k,j,k-1)$$

Complete Proof on One Slide

For all $1 \leq i, j \leq n$:

$$R(i,j,0) = \begin{cases} \{\sigma : \delta(i,\sigma) = j\} & \text{if } i \neq j \ \} \\ \{\sigma : \delta(i,\sigma) = j\} \cup \{e\} & \text{if } i = j \ \} \end{cases} \qquad (2)$$

All $R(i,j,0)$ are Regex.

For all $1 \leq i, j \leq n$ and all $0 \leq k \leq n$

$$R(i,j,k) = R(i,j,k-1) \bigcup R(i,k,k-1) R(k,k,k-1)^* R(k,j,k-1)$$

If ALL $R(i,j,k-1)$ are Regex, then ALL $R(i,j,k)$ are Regex.

Textbook Regular Expressions

We allow numbers as exponents. For example the following is not a regex but is a trex:

$$\{a, b\}^* a \{a, b\}^n.$$

Textbook Regular Expressions

We allow numbers as exponents. For example the following is not a regex but is a trex:

$$\{a, b\}^* a\{a, b\}^n.$$

Often the trex is shorter than the regex.

Closure Properties

Summary of Proofs of Closure Properties

Summary of Proofs of Closure Properties

Prod means product construction where you use $Q_1 \times Q_2$

Summary of Proofs of Closure Properties

Prod means product construction where you use $Q_1 \times Q_2$
Def means by Definition, e.g., $L_1 \cup L_2$ for regex.

Summary of Proofs of Closure Properties

Prod means product construction where you use $Q_1 \times Q_2$

Def means by Definition, e.g., $L_1 \cup L_2$ for regex.

Swap means swapping final and non-final states.

Summary of Proofs of Closure Properties

Prod means product construction where you use $Q_1 \times Q_2$

Def means by Definition, e.g., $L_1 \cup L_2$ for regex.

Swap means swapping final and non-final states.

***e*-trans** means by using e-transitions, e.g., $L_1 \cdot L_2$ for NFAs.

Summary of Proofs of Closure Properties

Prod means product construction where you use $Q_1 \times Q_2$

Def means by Definition, e.g., $L_1 \cup L_2$ for regex.

Swap means swapping final and non-final states.

***e*-trans** means by using e-transitions, e.g., $L_1 \cdot L_2$ for NFAs.

X means hard to prove, e.g., $\overline{L}$ for NFA.

Summary of Proofs of Closure Properties

Prod means product construction where you use $Q_1 \times Q_2$

Def means by Definition, e.g., $L_1 \cup L_2$ for regex.

Swap means swapping final and non-final states.

***e*-trans** means by using e-transitions, e.g., $L_1 \cdot L_2$ for NFAs.

X means hard to prove, e.g., $\overline{L}$ for NFA.

Property	DFA	NFA	regex
$L_1 \cup L_2$	Prod	e-trans	Def
$L_1 \cap L_2$	Prod	Prod	X
$\overline{L}$	Swap	X	X
$L_1 \cdot L_2$	X	e-trans	Def
L^*	X	e-trans	Def

Summary of Blowup for Closure Properties

X means **Can't Prove Easily**

Summary of Blowup for Closure Properties

X means **Can't Prove Easily**

n_1, n_2 are number of states in a DFA or NFA.

Summary of Blowup for Closure Properties

X means **Can't Prove Easily**

n_1, n_2 are number of states in a DFA or NFA.

$\ell_, \ell_2$ are length of regex.

Summary of Blowup for Closure Properties

X means **Can't Prove Easily**

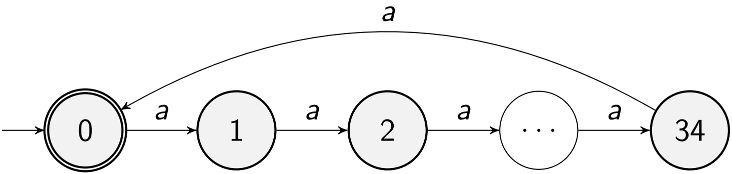
n_1, n_2 are number of states in a DFA or NFA.

$\ell_, \ell_2$ are length of regex.

Closure Property	DFA	NFA	Regex
$L_1 \cup L_2$	$n_1 n_2$	$n_1 + n_2 + 1$	$\ell_1 + \ell_2$
$L_1 \cap L_2$	$n_1 n_2$	$n_1 n_2$	X
$L_1 \cdot L_2$	X	$n_1 + n_2$	$\ell_1 + \ell_2$
$\overline{L}$	n	X	X
L^*	X	$n + 1$	$\ell + 1$

Number of States for DFAs and NFAs

Minimal DFA for $L_1 = \{a^i : i \equiv 0 \pmod{35}\}$



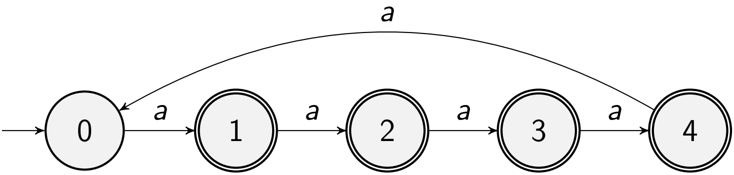
Min DFA for $L_2 = \{a^i : i \not\equiv 0 \pmod{35}\}$

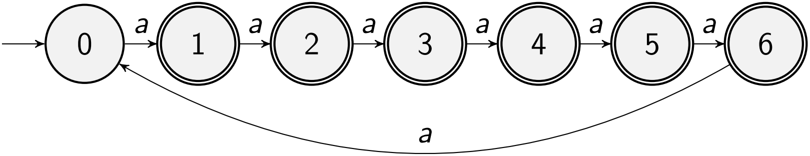
Min DFA for $L_2 = \{a^i : i \not\equiv 0 \pmod{35}\}$

$\exists$ DFA for L_2: 35 states: swap final-$\overline{\text{final}}$ states in DFA for L_1.

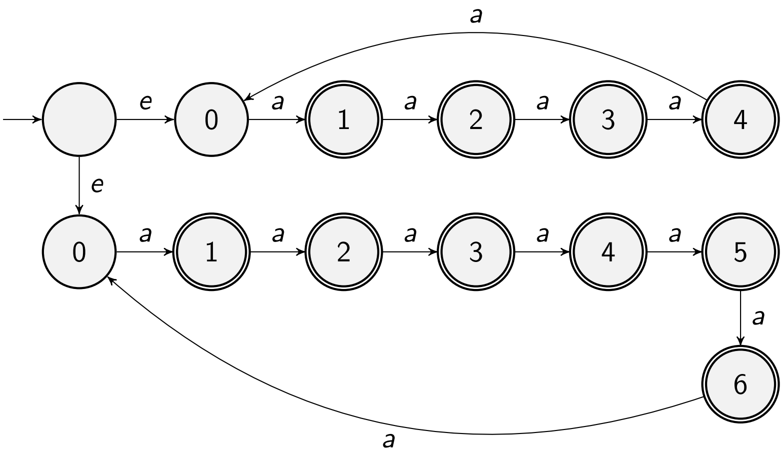
Small NFA for $L_2 = \{a^i : i \not\equiv 0 \pmod{35}\}$

Need these two NFA's.





Small NFA for $L_2 = \{a^i : i \not\equiv 0 \pmod{35}\}$



$$L_2 = \{a^i : i \not\equiv 0 \pmod{35}\}$$

$L_2 = \{a^i : i \not\equiv 0 \pmod{35}\}$

DFA for L_2 requires 35 states.

$L_2 = \{a^i : i \not\equiv 0 \pmod{35}\}$

DFA for L_2 requires 35 states.

NFA for L_2 can be done with $1 + 5 + 7 = 13$ states.

DFA for $L_4 = \{a^i : i \neq 1000\}$

DFA for $L_4 = \{a^i : i \neq 1000\}$

1. There is a DFA for L_4 that has 1000 states.
2. Any DFA for L_3 has ≥ 1000 states.

Small NFA for $L_4 = \{a^n : n \neq 1000\}$

Small NFA for $L_4 = \{a^n : n \neq 1000\}$

Two NFA's:

Small NFA for $L_4 = \{a^n : n \neq 1000\}$

Two NFA's:

NFA A:

Small NFA for $L_4 = \{a^n : n \neq 1000\}$

Two NFA's:

NFA A:

- Does NOT accept a^{1000}.

Small NFA for $L_4 = \{a^n : n \neq 1000\}$

Two NFA's:

NFA A:

- Does NOT accept a^{1000}.
- Accepts all words longer than 1000.

Small NFA for $L_4 = \{a^n : n \neq 1000\}$

Two NFA's:

NFA A:

- Does NOT accept a^{1000}.
- Accepts all words longer than 1000.
- Do not care about words shorter than 1000.

Small NFA for $L_4 = \{a^n : n \neq 1000\}$

Two NFA's:

NFA A:

- Does NOT accept a^{1000}.
- Accepts all words longer than 1000.
- Do not care about words shorter than 1000.

NFA B:

Small NFA for $L_4 = \{a^n : n \neq 1000\}$

Two NFA's:

NFA A:

- Does NOT accept a^{1000}.
- Accepts all words longer than 1000.
- Do not care about words shorter than 1000.

NFA B:

- Does NOT accept a^{1000}.

Small NFA for $L_4 = \{a^n : n \neq 1000\}$

Two NFA's:

NFA A:

- Does NOT accept a^{1000}.
- Accepts all words longer than 1000.
- Do not care about words shorter than 1000.

NFA B:

- Does NOT accept a^{1000}.
- Accepts all words shorter than 1000.

Small NFA for $L_4 = \{a^n : n \neq 1000\}$

Two NFA's:

NFA A:

- Does NOT accept a^{1000}.
- Accepts all words longer than 1000.
- Do not care about words shorter than 1000.

NFA B:

- Does NOT accept a^{1000}.
- Accepts all words shorter than 1000.
- Do not care about words longer than 1000.

Small NFA for $L_4 = \{a^n : n \neq 1000\}$

Two NFA's:

NFA A:

- Does NOT accept a^{1000}.
- Accepts all words longer than 1000.
- Do not care about words shorter than 1000.

NFA B:

- Does NOT accept a^{1000}.
- Accepts all words shorter than 1000.
- Do not care about words longer than 1000.

Create the union of NFA's A and B.

Sums of 32's and 33's

Thm

1) $(\forall n \geq 1001)(\exists x, y \in \mathbb{N})[n = 32x + 33y + 9]$.

Sums of 32's and 33's

Thm

1) $(\forall n \ge 1001)(\exists x, y \in \mathbb{N})[n = 32x + 33y + 9]$.

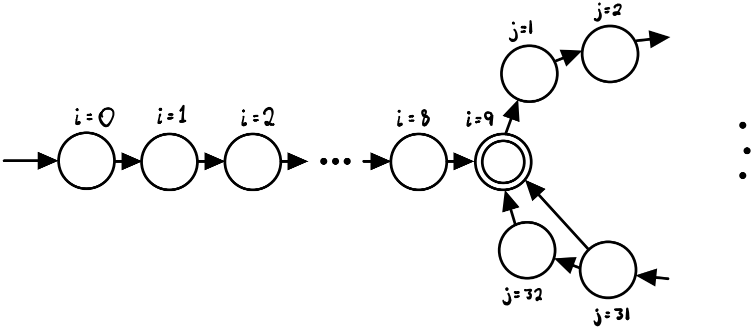
2) $(\neg \exists x, y \in \mathbb{N})[1000 = 32x + 33y + 9]$.

NFA A

Idea Start state, then 8 states, then a loop of size 33 with a shortcut at 32.

NFA A

Idea Start state, then 8 states, then a loop of size 33 with a shortcut at 32.



Why Works for $\{a^i : i \geq 1001\}$ and More

By the loop Theorem for 32, 33, the NFA

1. Accepts $\boldsymbol{\{a^i : i \geq 1001\}}$.
2. Might accept more.
3. DOES NOT accept a^{1000}.

Number of States for $\{a^i : i \geq 1001\}$

1. Start state

Number of States for $\{a^i : i \geq 1001\}$

1. Start state
2. A chain of 9 states including the start state.

Number of States for $\{a^i : i \geq 1001\}$

1. Start state
2. A chain of 9 states including the start state.
3. A loop of 33 states. The shortcut on 32 does not affect the number of states.

Number of States for $\{a^i : i \geq 1001\}$

1. Start state
2. A chain of 9 states including the start state.
3. A loop of 33 states. The shortcut on 32 does not affect the number of states.

Total number of states: $9 + 33 = 42$.

Still Need NFA B

Still Need NFA B

Idea

Still Need NFA B

Idea

$1000 \equiv 0 \pmod 2$ 2-state DFA for $\{a^i : i \not\equiv 0 \pmod 2\}$.

Still Need NFA B

Idea

$1000 \equiv 0 \pmod 2$ 2-state DFA for $\{a^i : i \not\equiv 0 \pmod 2\}$.

$1000 \equiv 1 \pmod 3$ 3-state DFA for $\{a^i : i \not\equiv 1 \pmod 3\}$.

Still Need NFA B

Idea

$1000 \equiv 0 \pmod 2$ 2-state DFA for $\{a^i : i \not\equiv 0 \pmod 2\}$.

$1000 \equiv 1 \pmod 3$ 3-state DFA for $\{a^i : i \not\equiv 1 \pmod 3\}$.

$1000 \equiv 0 \pmod 5$ 5-state DFA for $\{a^i : i \not\equiv 0 \pmod 5\}$.

Still Need NFA B

Idea

$1000 \equiv 0 \pmod 2$ 2-state DFA for $\{a^i : i \not\equiv 0 \pmod 2\}$.

$1000 \equiv 1 \pmod 3$ 3-state DFA for $\{a^i : i \not\equiv 1 \pmod 3\}$.

$1000 \equiv 0 \pmod 5$ 5-state DFA for $\{a^i : i \not\equiv 0 \pmod 5\}$.

$1000 \equiv 6 \pmod 7$ 7-state DFA for $\{a^i : i \not\equiv 6 \pmod 7\}$.

Still Need NFA B

Idea

$1000 \equiv 0 \pmod 2$ 2-state DFA for $\{a^i : i \not\equiv 0 \pmod 2\}$.

$1000 \equiv 1 \pmod 3$ 3-state DFA for $\{a^i : i \not\equiv 1 \pmod 3\}$.

$1000 \equiv 0 \pmod 5$ 5-state DFA for $\{a^i : i \not\equiv 0 \pmod 5\}$.

$1000 \equiv 6 \pmod 7$ 7-state DFA for $\{a^i : i \not\equiv 6 \pmod 7\}$.

$1000 \equiv 10 \pmod{11}$ 11-state DFA for $\{a^i : i \not\equiv 10 \pmod{11}\}$.

Still Need NFA B

Idea

$1000 \equiv 0 \pmod 2$ 2-state DFA for $\{a^i : i \not\equiv 0 \pmod 2\}$.

$1000 \equiv 1 \pmod 3$ 3-state DFA for $\{a^i : i \not\equiv 1 \pmod 3\}$.

$1000 \equiv 0 \pmod 5$ 5-state DFA for $\{a^i : i \not\equiv 0 \pmod 5\}$.

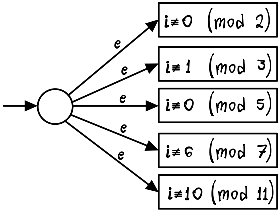
$1000 \equiv 6 \pmod 7$ 7-state DFA for $\{a^i : i \not\equiv 6 \pmod 7\}$.

$1000 \equiv 10 \pmod{11}$ 11-state DFA for $\{a^i : i \not\equiv 10 \pmod{11}\}$.

Could go on to 13,17, etc. But we will see we can stop here.

Machine B

Machine B



NFA for $\{a^i : i \le 999\}$ AND More, but NOT a^{1000}

Thm Let M be the NFA from the last slide with the Mods.
$M(a^{1000})$ is rejected. This is obvious.

NFA for $\{a^i : i \le 999\}$ AND More, but NOT a^{1000}

Thm Let M be the NFA from the last slide with the Mods. $M(a^{1000})$ is rejected. This is obvious.

We omit the proof that it works but note that we use that the product of the mods

$$2 \times 3 \times 5 \times 7 \times 11 = 2310 > 1000.$$

How Many States for $\{a^i : i \le 999\}$ AND More, but NOT a^{1000}?

$2 + 3 + 5 + 7 + 11 = 28$ states.
Plus the start state, so 29.

NFA for $\{a^i : i \neq 1000\}$

NFA for $\{a^i : i \neq 1000\}$

1. We have an NFA on 42 states that accepts $\{a^i : i \geq 1001\}$
   This includes the start state.

NFA for $\{a^i : i \neq 1000\}$

1. We have an NFA on 42 states that accepts $\{a^i : i \geq 1001\}$ This includes the start state.
2. We have an NFA on 29 states that accepts $\{a^i : i \leq 999\}$ and other stuff, but NOT a^{1000}. This includes the start state.

NFA for $\{a^i : i \neq 1000\}$

1. We have an NFA on 42 states that accepts $\{a^i : i \geq 1001\}$ This includes the start state.
2. We have an NFA on 29 states that accepts $\{a^i : i \leq 999\}$ and other stuff, but NOT a^{1000}. This includes the start state.

Take NFA of union using e-transitions for an NFA and do not count start state twice, so have

$$42 + 29 - 1 = 70 \text{ states.}$$

Can We Do Better than 70 States?

YES–59 states:

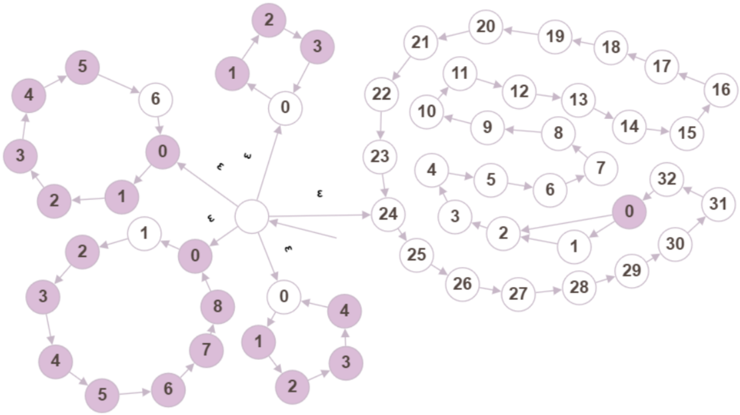


Figure: 59 State NFA for L_4

Math Needed for $\{a^i : i \neq n\}$

Frobenius Thm (aka The Chicken McNugget Thm)

Math Needed for $\{a^i : i \neq n\}$

Frobenius Thm (aka The Chicken McNugget Thm)

Thm If x, y are relatively prime then

- For all $z \geq xy - x - y + 1$ there exists $c, d \in \mathbb{N}$ such that $z = cx + dy$.
- There is no $c, d \in \mathbb{N}$ such that $xy - x - y = cx + dy$.

Math Needed for $\{a^i : i \neq n\}$

Frobenius Thm (aka The Chicken McNugget Thm)

Thm If x, y are relatively prime then

- For all $z \geq xy - x - y + 1$ there exists $c, d \in \mathbb{N}$ such that $z = cx + dy$.
- There is no $c, d \in \mathbb{N}$ such that $xy - x - y = cx + dy$.

We use this to get an NFA for $\{a^i : i \geq n+1\}$ by using $x, y \approx \sqrt{n}$.

Math Needed for $\{a^i : i \ne n\}$

Frobenius Thm (aka The Chicken McNugget Thm)

Thm If x, y are relatively prime then

- For all $z \ge xy - x - y + 1$ there exists $c, d \in \mathbb{N}$ such that $z = cx + dy$.
- There is no $c, d \in \mathbb{N}$ such that $xy - x - y = cx + dy$.

We use this to get an NFA for $\{a^i : i \ge n + 1\}$ by using $x, y \approx \sqrt{n}$. Want to get $xy - x - y \le n$ so can use the tail to get $xy - x - y + t = n + 1$.

Math Needed for $\{a^i : i \neq n\}$

Frobenius Thm (aka The Chicken McNugget Thm)

Thm If x, y are relatively prime then

- For all $z \geq xy - x - y + 1$ there exists $c, d \in \mathbb{N}$ such that $z = cx + dy$.
- There is no $c, d \in \mathbb{N}$ such that $xy - x - y = cx + dy$.

We use this to get an NFA for $\{a^i : i \geq n + 1\}$ by using $x, y \approx \sqrt{n}$.
Want to get $xy - x - y \leq n$ so can use the tail to get
$xy - x - y + t = n + 1$.
This leads to loops and tail that are roughly $\leq 2\sqrt{n}$ states.

Proving That a Language Is Not Regular

Pumping Lemma (PL)

$L_1 = \{a^n b^n : n \geq 0\}$ is Not Regular

$L_1 = \{a^n b^n : n \geq 0\}$ is Not Regular

Proof

$L_1 = \{a^n b^n : n \geq 0\}$ is Not Regular

Proof Assume L_1 is regular via DFA M with m states.

$L_1 = \{a^n b^n : n \geq 0\}$ is Not Regular

Proof Assume L_1 is regular via DFA M with m states.
Run M on $a^m b^m$.

$L_1 = \{a^n b^n : n \geq 0\}$ is Not Regular

Proof Assume L_1 is regular via DFA M with m states.
Run M on $a^m b^m$.
States encountered processing a^m:

$$q_0, q_1, q_2, \ldots, q_{m-1}$$

$L_1 = \{a^n b^n : n \geq 0\}$ is Not Regular

Proof Assume L_1 is regular via DFA M with m states.
Run M on $a^m b^m$.
States encountered processing a^m:

$$q_0, q_1, q_2, \ldots, q_{m-1}$$

By **PHP** some state is encountered twice.

$L_1 = \{a^n b^n : n \geq 0\}$ is Not Regular

Proof Assume L_1 is regular via DFA M with m states.
Run M on $a^m b^m$.
States encountered processing a^m:

$$q_0, q_1, q_2, \ldots, q_{m-1}$$

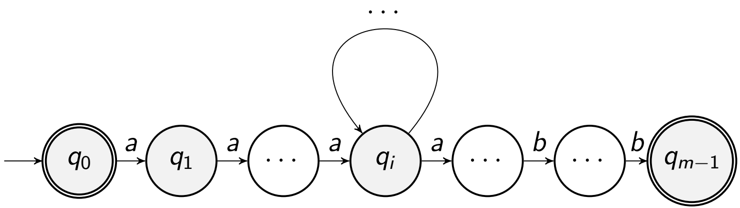
By **PHP** some state is encountered twice.
So there is a loop at that state where $k \geq 1$ a's are processed.

$L_1 = \{a^n b^n : n \geq 0\}$ is Not Regular

Proof Assume L_1 is regular via DFA M with m states.
Run M on $a^m b^m$.
States encountered processing a^m:

$$q_0, q_1, q_2, \ldots, q_{m-1}$$

By **PHP** some state is encountered twice.
So there is a loop at that state where $k \geq 1$ a's are processed.

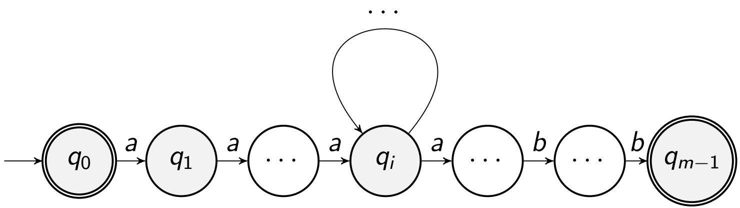


$L_1 = \{a^n b^n : n \geq 0\}$ is Not Regular

Proof Assume L_1 is regular via DFA M with m states.
Run M on $a^m b^m$.
States encountered processing a^m:

$$q_0, q_1, q_2, \ldots, q_{m-1}$$

By **PHP** some state is encountered twice.
So there is a loop at that state where $k \geq 1$ a's are processed.



$a^{n+k} b^n$ is accepted by following the loop again. Contradiction.

General Technique

General Technique

Pumping Lemma (PL) If L is regular then there exist n_0 and n_1 such that the following holds:

General Technique

Pumping Lemma (PL) If L is regular then there exist n_0 and n_1 such that the following holds:
For all $w \in L$, $|w| \geq n_0$ there exist x, y, z such that:

General Technique

Pumping Lemma (PL) If L is regular then there exist n_0 and n_1 such that the following holds:
For all $w \in L$, $|w| \geq n_0$ there exist x, y, z such that:

1. $w = xyz$ and $y \neq e$.

General Technique

Pumping Lemma (PL) If L is regular then there exist n_0 and n_1 such that the following holds:
For all $w \in L$, $|w| \geq n_0$ there exist x, y, z such that:

1. $w = xyz$ and $y \neq e$.
2. $|xy| \leq n_1$ (or can take $|yz| \leq n_1$ but not both.)

General Technique

Pumping Lemma (PL) If L is regular then there exist n_0 and n_1 such that the following holds:
For all $w \in L$, $|w| \geq n_0$ there exist x, y, z such that:

1. $w = xyz$ and $y \neq e$.
2. $|xy| \leq n_1$ (or can take $|yz| \leq n_1$ but not both.)
3. For all $i \geq 0$, $xy^iz \in L$.

General Technique

Pumping Lemma (PL) If L is regular then there exist n_0 and n_1 such that the following holds:
For all $w \in L$, $|w| \geq n_0$ there exist x, y, z such that:

1. $w = xyz$ and $y \neq e$.
2. $|xy| \leq n_1$ (or can take $|yz| \leq n_1$ but not both.)
3. For all $i \geq 0$, $xy^i z \in L$.

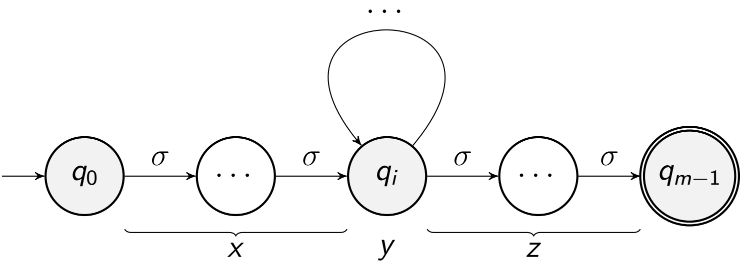
Proof by picture

General Technique

Pumping Lemma (PL) If L is regular then there exist n_0 and n_1 such that the following holds:
For all $w \in L$, $|w| \geq n_0$ there exist x, y, z such that:

1. $w = xyz$ and $y \neq e$.
2. $|xy| \leq n_1$ (or can take $|yz| \leq n_1$ but not both.)
3. For all $i \geq 0$, $xy^iz \in L$.

Proof by picture



How We Use the PL

How We Use the PL

We restate it in the way that we use it.

How We Use the PL

We restate it in the way that we use it.

PL If L is reg then **for large enough strings w in L** there exist x, y, z such that:

How We Use the PL

We restate it in the way that we use it.

PL If L is reg then **for large enough strings w in L** there exist x, y, z such that:

1. $w = xyz$ and $y \neq e$.

How We Use the PL

We restate it in the way that we use it.

PL If L is reg then **for large enough strings w in L** there exist x, y, z such that:

1. $w = xyz$ and $y \neq e$.
2. $|xy|$ **is short**.

How We Use the PL

We restate it in the way that we use it.

PL If L is reg then **for large enough strings w in L** there exist x, y, z such that:

1. $w = xyz$ and $y \neq e$.
2. $|xy|$ **is short**.
3. for all i, $xy^i z \in L$.

How We Use the PL

We restate it in the way that we use it.

PL If L is reg then **for large enough strings w in L** there exist x, y, z such that:

1. $w = xyz$ and $y \neq e$.
2. $|xy|$ **is short**.
3. for all i, $xy^iz \in L$.

We then find some i such that $xy^iz \notin L$ for the contradiction.

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.
3. For all $i \geq 0$, $xy^i z \in L_1$.

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.
3. For all $i \geq 0$, $xy^i z \in L_1$.

Take w long enough so that the xy part only has a's.

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.
3. For all $i \geq 0$, $xy^i z \in L_1$.

Take w long enough so that the xy part only has a's.
$x = a^j$, $y = a^k$, $z = a^{n-j-k} b^n$.

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.
3. For all $i \geq 0$, $xy^i z \in L_1$.

Take w long enough so that the xy part only has a's.
$x = a^j$, $y = a^k$, $z = a^{n-j-k} b^n$. Note $k \geq 1$.

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.
3. For all $i \geq 0$, $xy^i z \in L_1$.

Take w long enough so that the xy part only has a's.
$x = a^j$, $y = a^k$, $z = a^{n-j-k} b^n$. Note $k \geq 1$.
By the PL, all of the words

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.
3. For all $i \geq 0$, $xy^i z \in L_1$.

Take w long enough so that the xy part only has a's.
$x = a^j$, $y = a^k$, $z = a^{n-j-k} b^n$. Note $k \geq 1$.
By the PL, all of the words

$$a^j \left(a^k\right)^i a^{n-j-k} b^n$$

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.
3. For all $i \geq 0$, $xy^i z \in L_1$.

Take w long enough so that the xy part only has a's.
$x = a^j$, $y = a^k$, $z = a^{n-j-k} b^n$. Note $k \geq 1$.
By the PL, all of the words

$$a^j \left(a^k\right)^i a^{n-j-k} b^n \; = \; a^{n+k(i-1)} b^n$$

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.
3. For all $i \geq 0$, $xy^i z \in L_1$.

Take w long enough so that the xy part only has a's.
$x = a^j$, $y = a^k$, $z = a^{n-j-k} b^n$. Note $k \geq 1$.
By the PL, all of the words

$$a^j \left(a^k\right)^i a^{n-j-k} b^n \;=\; a^{n+k(i-1)} b^n$$

are in L_1.

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.
3. For all $i \geq 0$, $xy^i z \in L_1$.

Take w long enough so that the xy part only has a's.
$x = a^j$, $y = a^k$, $z = a^{n-j-k} b^n$. Note $k \geq 1$.
By the PL, all of the words

$$a^j \left(a^k\right)^i a^{n-j-k} b^n \;=\; a^{n+k(i-1)} b^n$$

are in L_1.
Take $i = 2$ to get

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.
3. For all $i \geq 0$, $xy^i z \in L_1$.

Take w long enough so that the xy part only has a's.
$x = a^j$, $y = a^k$, $z = a^{n-j-k} b^n$. Note $k \geq 1$.
By the PL, all of the words

$$a^j \left(a^k\right)^i a^{n-j-k} b^n \ = \ a^{n+k(i-1)} b^n$$

are in L_1.
Take $i = 2$ to get

$$a^{n+k} b^n \in L_1$$

REDO: $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular.
By PL, for long $a^n b^n \in L_1$, $\exists x, y, z$:

1. $y \neq e$.
2. $|xy|$ is short.
3. For all $i \geq 0$, $xy^i z \in L_1$.

Take w long enough so that the xy part only has a's.
$x = a^j$, $y = a^k$, $z = a^{n-j-k} b^n$. Note $k \geq 1$.
By the PL, all of the words

$$a^j \left(a^k\right)^i a^{n-j-k} b^n \;=\; a^{n+k(i-1)} b^n$$

are in L_1.
Take $i = 2$ to get

$$a^{n+k} b^n \in L_1$$

Contradiction since $k \geq 1$.

$L_3 = \{w : \#_a(w) \neq \#_b(w)\}$ is Not Regular

$L_3 = \{w : \#_a(w) \neq \#_b(w)\}$ is Not Regular

PL Does Not Help. When you increase the number of y's there is no way to control it so carefully to make the number of a's EQUAL the number of b's.

$L_3 = \{w : \#_a(w) \neq \#_b(w)\}$ is Not Regular

PL Does Not Help. When you increase the number of y's there is no way to control it so carefully to make the number of a's EQUAL the number of b's.

So what do to?

$L_3 = \{w : \#_a(w) \neq \#_b(w)\}$ is Not Regular

PL Does Not Help. When you increase the number of y's there is no way to control it so carefully to make the number of a's EQUAL the number of b's.

So what do to?

If L_3 is regular then $L_2 = \overline{L_3}$ is regular. But we know that L_2 is not regular. DONE!

$L_4 = \{a^{n^2} : n \in \mathbb{N}\}$ is Not Regular

$L_4 = \{a^{n^2} : n \in \mathbb{N}\}$ is Not Regular

Intuition Perfect squares keep getting further apart.

$L_4 = \{a^{n^2} : n \in \mathbb{N}\}$ is Not Regular

Intuition Perfect squares keep getting further apart.
Pumping Lemma says you can always add some constant k to produce a word in the language.

$L_4 = \{a^{n^2} : n \in \mathbb{N}\}$ is Not Regular

Intuition Perfect squares keep getting further apart.
Pumping Lemma says you can always add some constant k to produce a word in the language.

We Omit the Formal Proof.