Finite Automata

Determinism, Nondeterminism, and DFA Minimization

Computer Science Theory

Slide Set 2

Table of Contents

Learning Outcomes

By the end of this slide set, you should be able to:

- define DFAs and ε-NFAs formally and trace their computations;
- design finite automata whose states record exactly the needed information about an input prefix;
- convert an ε-NFA to an equivalent DFA using subset construction;
- justify a design or conversion using an invariant;
- identify reachable, distinguishable, and equivalent DFA states; and
- minimize a DFA using table filling and certify that no further merging is possible.

Part 1: Intuitive Introduction to DFAs

Motivating model: A Simplified Door Controller

Finite automata are excellent models for computers with an extremely limited amount of memory.

Part 1: Intuitive Introduction to DFAs

Motivating model: A Simplified Door Controller

Finite automata are excellent models for computers with an extremely limited amount of memory.

Example: Automatic Door Controller

The controller has states **OPEN** and **CLOSED**. Its input records whether a person is detected at the front, rear, both sides, or neither side. In this toy policy, any detection opens the door; no detection closes it. Timing and other physical details are omitted.

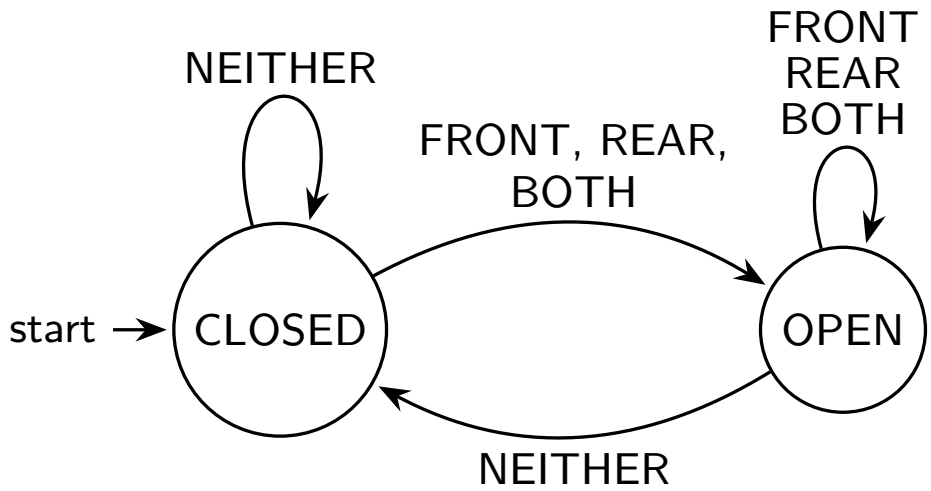
Part 1: Intuitive Introduction to DFAs

Motivating model: A Simplified Door Controller

Finite automata are excellent models for computers with an extremely limited amount of memory.

Example: Automatic Door Controller

The controller has states **OPEN** and **CLOSED**. Its input records whether a person is detected at the front, rear, both sides, or neither side. In this toy policy, any detection opens the door; no detection closes it. Timing and other physical details are omitted.



This diagram models control states. To use such a machine as a *language recognizer*, we must also specify which states accept.

Formal Definition of a DFA

To resolve any ambiguities, we require a mathematically precise definition.

Formal Definition of a DFA

To resolve any ambiguities, we require a mathematically precise definition.

The 5-Tuple Definition

A deterministic finite automaton is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where:

1. Q is a finite, nonempty set of states,
2. Σ is a finite input alphabet,
3. $\delta : Q \times \Sigma \to Q$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Formal Definition of a DFA

To resolve any ambiguities, we require a mathematically precise definition.

The 5-Tuple Definition

A deterministic finite automaton is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where:

1. Q is a finite, nonempty set of states,
2. Σ is a finite input alphabet,
3. $\delta : Q \times \Sigma \to Q$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Crucial detail: δ is total. For every $q \in Q$ and $a \in \Sigma$, exactly one transition $\delta(q, a)$ is defined.

Extended Transition Function

The transition function δ strictly handles single symbols. To describe the processing of entire strings, it is convenient to introduce the extended transition function $\delta^* : Q \times \Sigma^* \to Q$.

Extended Transition Function

The transition function δ strictly handles single symbols. To describe the processing of entire strings, it is convenient to introduce the extended transition function $\delta^* : Q \times \Sigma^* \to Q$.

Recursive Definition of δ^*

We define δ^* recursively as follows:

1. $\delta^*(q, \varepsilon) = q$.
2. $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$.

Extended Transition Function

The transition function δ strictly handles single symbols. To describe the processing of entire strings, it is convenient to introduce the extended transition function $\delta^* : Q \times \Sigma^* \to Q$.

Recursive Definition of δ^*

We define δ^* recursively as follows:

1. $\delta^*(q, \varepsilon) = q$.
2. $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$.

Language recognized by M:

$$L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\}.$$

A language is **regular** if it is recognized by some DFA.

Extended Transition Function

The transition function δ strictly handles single symbols. To describe the processing of entire strings, it is convenient to introduce the extended transition function $\delta^* : Q \times \Sigma^* \to Q$.

Recursive Definition of δ^*

We define δ^* recursively as follows:

1. $\delta^*(q, \varepsilon) = q$.
2. $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$.

Language recognized by M:

$$L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\}.$$

A language is **regular** if it is recognized by some DFA. A useful composition law (induction on $|v|$) is

$$\delta^*(q, uv) = \delta^*(\delta^*(q, u), v).$$

Reading uv means reading u first, then continuing with v.

Reading a DFA: Runs and Boundary Cases

For $w = a_1 \cdots a_m$, the unique run is $r_0, \ldots, r_m$, where

$$r_0 = q_0, \qquad r_i = \delta(r_{i-1}, a_i) \quad (1 \le i \le m).$$

The machine accepts exactly when $r_m \in F$.

Reading a DFA: Runs and Boundary Cases

For $w = a_1 \cdots a_m$, the unique run is $r_0, \ldots, r_m$, where

$$r_0 = q_0, \qquad r_i = \delta(r_{i-1}, a_i) \quad (1 \le i \le m).$$

The machine accepts exactly when $r_m \in F$.

- Visiting an accepting state midway is not enough; consume the *entire* input before deciding.
- The empty word consumes no symbols: $\varepsilon \in L(M)$ iff $q_0 \in F$.
- $F = \emptyset$ is allowed and gives $L(M) = \emptyset$. If $F = Q$, then $L(M) = \Sigma^*$.

Reading a DFA: Runs and Boundary Cases

For $w = a_1 \cdots a_m$, the unique run is $r_0, \ldots, r_m$, where

$$r_0 = q_0, \qquad r_i = \delta(r_{i-1}, a_i) \quad (1 \le i \le m).$$

The machine accepts exactly when $r_m \in F$.

- Visiting an accepting state midway is not enough; consume the *entire* input before deciding.
- The empty word consumes no symbols: $\varepsilon \in L(M)$ iff $q_0 \in F$.
- $F = \emptyset$ is allowed and gives $L(M) = \emptyset$. If $F = Q$, then $L(M) = \Sigma^*$.

How to read the drawing

An incoming arrow marks the start; a double circle marks acceptance. An edge labeled $0, 1$ abbreviates two transitions, not the word 01. A DFA has no ε-moves; ε is not an input symbol.

A Design Method: What Must the State Remember?

1. Give each state a precise meaning about the prefix read so far.
2. Choose the state whose meaning holds for the empty prefix.
3. For every state and symbol, update that meaning in one step.
4. Mark states whose meaning guarantees the required condition *if the input ends now*.
5. Test boundary words, then prove the state meanings by induction.

A Design Method: What Must the State Remember?

1. Give each state a precise meaning about the prefix read so far.
2. Choose the state whose meaning holds for the empty prefix.
3. For every state and symbol, update that meaning in one step.
4. Mark states whose meaning guarantees the required condition *if the input ends now*.
5. Test boundary words, then prove the state meanings by induction.

Reusable correctness proof

Base: the invariant holds before reading input. **Step:** each transition preserves the invariant after one symbol. **Conclusion:** the accepting states correspond exactly to the desired words. This proves both soundness and completeness.

Typical memories: a count modulo k, progress through a required prefix, or the longest relevant suffix. Avoid storing irrelevant history.

Designing DFAs: Case Study 1

Goal: Construct a DFA accepting all strings over $\{0, 1\}$ with an **odd number of 1s**.

Designing DFAs: Case Study 1

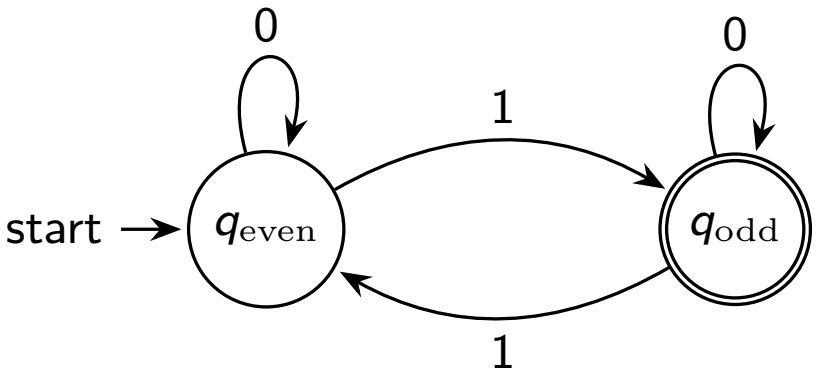
Goal: Construct a DFA accepting all strings over $\{0, 1\}$ with an **odd number of 1s**.

Thought Process: We don't need to remember the entire string. We only need to remember whether the number of 1s seen so far is even or odd.

Designing DFAs: Case Study 1

Goal: Construct a DFA accepting all strings over $\{0, 1\}$ with an **odd number of 1s**.

Thought Process: We don't need to remember the entire string. We only need to remember whether the number of 1s seen so far is even or odd.



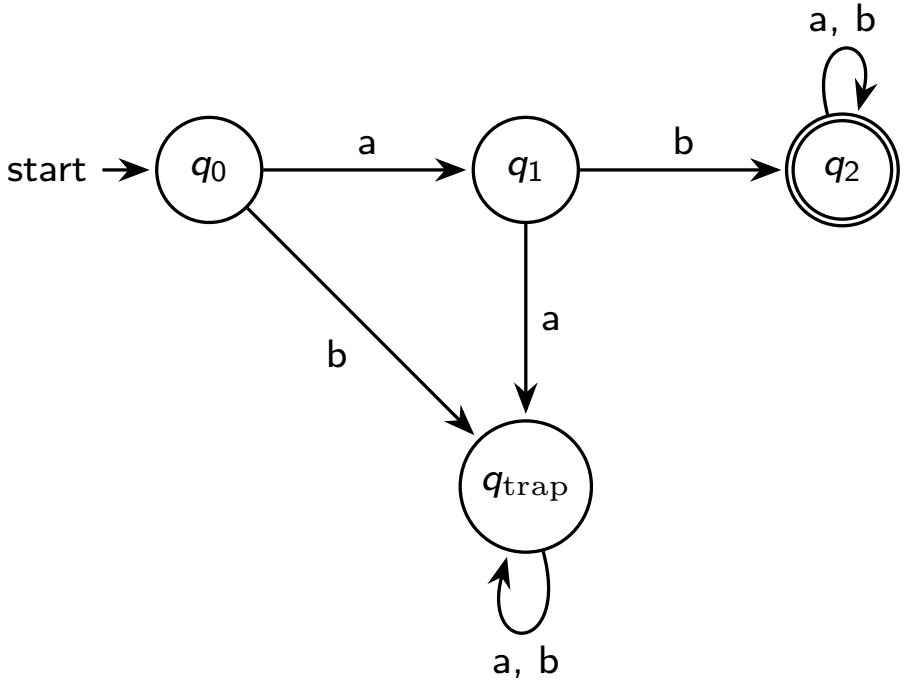
- **States:** $Q = \{q_{\text{even}}, q_{\text{odd}}\}$.
- **Start state:** q_{even} (zero 1s is even).
- **Accepting states:** $F = \{q_{\text{odd}}\}$.

Designing DFAs: Case Study 2 & Trap States

Goal: Construct a DFA over $\{a, b\}$ accepting strings beginning with the prefix ab.

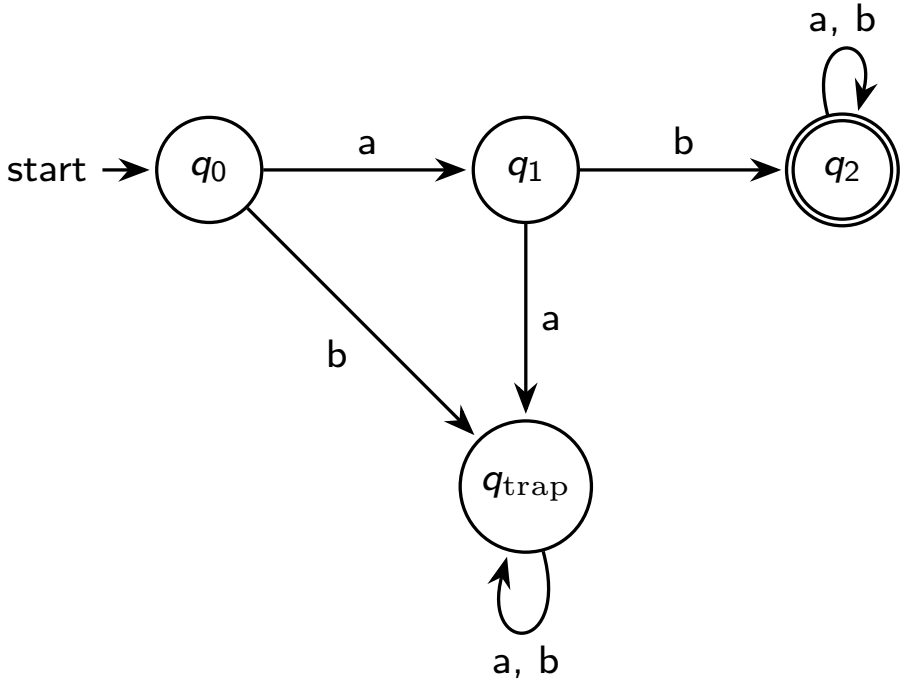
Designing DFAs: Case Study 2 & Trap States

Goal: Construct a DFA over $\{a, b\}$ accepting strings beginning with the prefix ab.



Designing DFAs: Case Study 2 & Trap States

Goal: Construct a DFA over $\{a, b\}$ accepting strings beginning with the prefix ab.



Teacher's Corner: The Trap State

A DFA may not simply omit a transition: δ must be total. Once the required prefix becomes impossible, the machine enters a nonaccepting **rejecting sink** (trap) state and remains there. A sink has self-loops on every symbol; a sink can also be accepting, as q_2 is here.

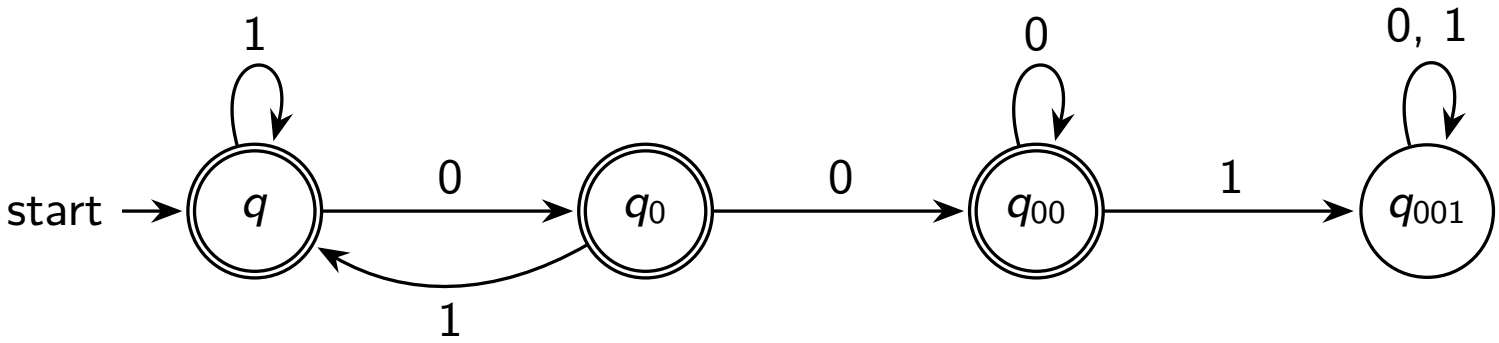
Designing DFAs: Case Study 3

Goal: Construct a DFA over $\{0, 1\}$ recognizing strings that do not contain the substring 001.

Designing DFAs: Case Study 3

Goal: Construct a DFA over $\{0, 1\}$ recognizing strings that do not contain the substring 001.

Strategy: Track the longest suffix of the input that is also a prefix of 001. Once 001 occurs, enter a rejecting trap state. In particular, after 000 the relevant suffix is still 00.



The states q, q_0, q_{00} are accepting because the forbidden pattern has not yet appeared. State q_{001} is a rejecting trap state.

Part 3: Introduction to Nondeterminism

Self-Study Header: The Power of Choice

Nondeterminism allows several possible next states. A DFA is a special case: replace each successor q by the singleton set $\{q\}$.

Part 3: Introduction to Nondeterminism

Self-Study Header: The Power of Choice

Nondeterminism allows several possible next states. A DFA is a special case: replace each successor q by the singleton set $\{q\}$.

- **Branching model:** We may imagine the machine exploring all permitted paths; this is a mathematical model, not literal hardware duplication.
- **Acceptance:** The machine accepts if *at least one* branch reaches an accept state at the end of the input.
- **Empty transitions:** In the ε-NFA model, a move labeled ε (or λ) changes state without consuming input. Taking such a move is a possible choice, not an obligation to follow it before all other moves.

Formal Definition of an NFA

An ε-NFA is a tuple $N = (Q, \Sigma, \delta, q_0, F)$. The components Q, Σ, q_0, F have the same meanings as for a DFA. We use Sipser's convention of allowing ε-moves in an NFA; other texts distinguish an NFA from an ε-NFA.

Formal Definition of an NFA

An ε-NFA is a tuple $N = (Q, \Sigma, \delta, q_0, F)$. The components Q, Σ, q_0, F have the same meanings as for a DFA. We use Sipser's convention of allowing ε-moves in an NFA; other texts distinguish an NFA from an ε-NFA.

The Transition Function δ

Let $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$. The transition function takes a state and an input symbol or the empty string and produces the **set** of possible next states.

$$\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q).$$

Formal Definition of an NFA

An ε-NFA is a tuple $N = (Q, \Sigma, \delta, q_0, F)$. The components Q, Σ, q_0, F have the same meanings as for a DFA. We use Sipser's convention of allowing ε-moves in an NFA; other texts distinguish an NFA from an ε-NFA.

The Transition Function δ

Let $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$. The transition function takes a state and an input symbol or the empty string and produces the **set** of possible next states.

$$\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q).$$

Here, $\mathcal{P}(Q)$ (or 2^Q) is the power set of Q. Since the output is a set of states:

- It can return one state: $\{q_1\}$.
- It can return several states: $\{q_1, q_2\}$.
- It can return $\emptyset$, meaning no path continues by that move.

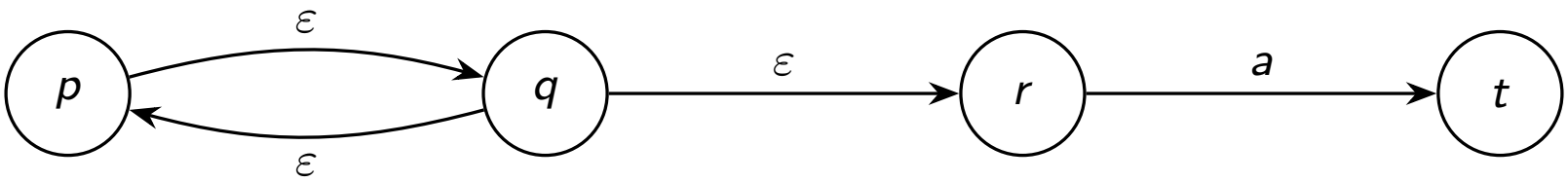
Every unspecified transition has value $\emptyset$. An NFA without ε-moves has $\delta(q, \varepsilon) = \emptyset$ for all q.

Computing Epsilon-Closure, Even with Cycles

For $R \subseteq Q$, let

$$E(R) = \{q : \text{some state in } R \text{ reaches } q \text{ using only } \varepsilon\text{-moves}\}.$$

Zero moves are allowed, so $R \subseteq E(R)$ and $E(\emptyset) = \emptyset$.

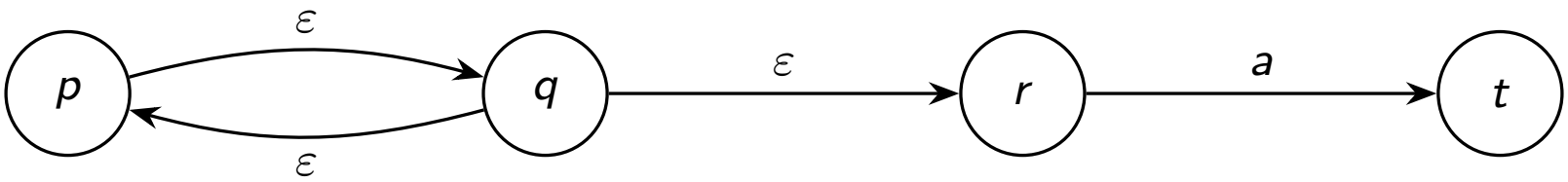


Computing Epsilon-Closure, Even with Cycles

For $R \subseteq Q$, let

$$E(R) = \{q : \text{some state in } R \text{ reaches } q \text{ using only } \varepsilon\text{-moves}\}.$$

Zero moves are allowed, so $R \subseteq E(R)$ and $E(\emptyset) = \emptyset$.



$E(\{p\}) = \{p, q, r\}$; do not cross the a-edge. Start a worklist with R and add unseen ε-successors until none remain. Mark each visited state, so cycles do not cause nontermination.

Acceptance means existence of a finite successful path

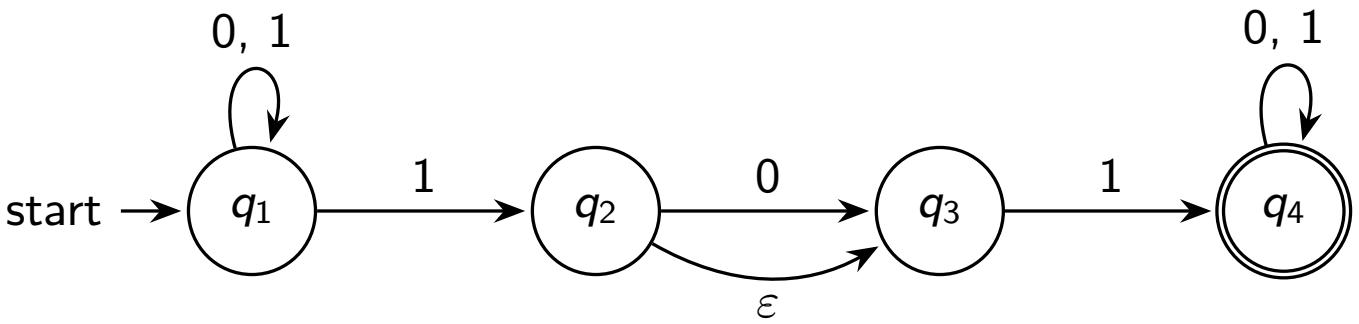
An infinite ε-loop is not an accepting computation. It also does not prevent acceptance if another finite path consumes the input and ends in an accepting state.

NFA Example

Goal: Accept strings containing either 101 or 11 as a substring.

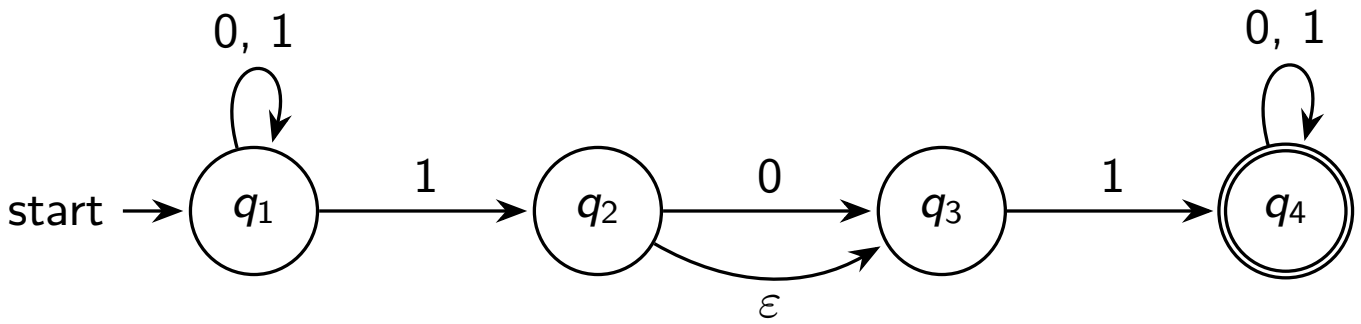
NFA Example

Goal: Accept strings containing either 101 or 11 as a substring.



NFA Example

Goal: Accept strings containing either 101 or 11 as a substring.



Why is this an NFA?

1. From q_1, reading 1 may remain in q_1 or move to q_2.
2. The path $q_2 \xrightarrow{0} q_3 \xrightarrow{1} q_4$ completes the substring 101.
3. Alternatively, $q_2 \xrightarrow{\varepsilon} q_3$ consumes no symbol, so the next 1 completes the substring 11.

Simulating an NFA: Sets, Not a Single Guess

Let R_w be all states reachable after consuming w, allowing ε-moves before, between, and after input symbols. Then

$$R_\varepsilon = E(\{q_0\}), \qquad R_{wa} = E\left(\bigcup_{q \in R_w} \delta(q, a)\right).$$

Accept exactly when $R_w \cap F \neq \emptyset$.

Simulating an NFA: Sets, Not a Single Guess

Let R_w be all states reachable after consuming w, allowing ε-moves before, between, and after input symbols. Then

$$R_\varepsilon = E(\{q_0\}), \qquad R_{wa} = E\left(\bigcup_{q \in R_w} \delta(q, a)\right).$$

Accept exactly when $R_w \cap F \neq \emptyset$. For the preceding NFA for containing 101 or 11, the start is q_1:

prefix	reachable set, after closure
ε	$\{q_1\}$
1	$\{q_1, q_2, q_3\}$
10	$\{q_1, q_3\}$
101	$\{q_1, q_2, q_3, q_4\}$

Simulating an NFA: Sets, Not a Single Guess

Let R_w be all states reachable after consuming w, allowing ε-moves before, between, and after input symbols. Then

$$R_\varepsilon = E(\{q_0\}), \qquad R_{wa} = E\left(\bigcup_{q \in R_w} \delta(q, a)\right).$$

Accept exactly when $R_w \cap F \neq \emptyset$. For the preceding NFA for containing 101 or 11, the start is q_1:

prefix	reachable set, after closure
ε	$\{q_1\}$
1	$\{q_1, q_2, q_3\}$
10	$\{q_1, q_3\}$
101	$\{q_1, q_2, q_3, q_4\}$

The last set contains q_4, so 101 is accepted. One unsuccessful path cannot establish rejection: *every* path must fail to accept. If the reachable set becomes empty, it stays empty on further input.

Part 4: Equivalence Theorem

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

Part 4: Equivalence Theorem

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

This is a surprising but highly useful result. It tells us that adding nondeterminism *does not increase* the computational power of a finite automaton—they both recognize exactly the class of regular languages.

Part 4: Equivalence Theorem

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

This is a surprising but highly useful result. It tells us that adding nondeterminism *does not increase* the computational power of a finite automaton—they both recognize exactly the class of regular languages.

Proof Idea: The Subset Construction Algorithm

If the NFA has k states, a DFA state records the set of NFA states reachable after the prefix read so far. At most 2^k subsets exist, though usually only some are reachable.

The Subset Construction Algorithm

Let $N = (Q, \Sigma, \delta_N, q_0, F_N)$ be an NFA. We build an equivalent DFA $M = (Q', \Sigma, \delta_M, q_0', F')$.

The Subset Construction Algorithm

Let $N = (Q, \Sigma, \delta_N, q_0, F_N)$ be an NFA. We build an equivalent DFA $M = (Q', \Sigma, \delta_M, q_0', F')$.

- **States:** Begin with $Q' = \mathcal{P}(Q)$ conceptually, then retain only subsets reachable from the start subset.

The Subset Construction Algorithm

Let $N = (Q, \Sigma, \delta_N, q_0, F_N)$ be an NFA. We build an equivalent DFA $M = (Q', \Sigma, \delta_M, q_0', F')$.

- **States:** Begin with $Q' = \mathcal{P}(Q)$ conceptually, then retain only subsets reachable from the start subset.
- ε**-closure:** $E(R)$ contains the states reachable from R using zero or more ε-moves.

The Subset Construction Algorithm

Let $N = (Q, \Sigma, \delta_N, q_0, F_N)$ be an NFA. We build an equivalent DFA $M = (Q', \Sigma, \delta_M, q_0', F')$.

- **States:** Begin with $Q' = \mathcal{P}(Q)$ conceptually, then retain only subsets reachable from the start subset.
- **ε-closure:** $E(R)$ contains the states reachable from R using zero or more ε-moves.
- **Start State (q_0'):** $q_0' = E(\{q_0\})$. We begin by calculating the ε-closure of the NFA's start state.

The Subset Construction Algorithm

Let $N = (Q, \Sigma, \delta_N, q_0, F_N)$ be an NFA. We build an equivalent DFA $M = (Q', \Sigma, \delta_M, q_0', F')$.

- **States:** Begin with $Q' = \mathcal{P}(Q)$ conceptually, then retain only subsets reachable from the start subset.
- **ε-closure:** $E(R)$ contains the states reachable from R using zero or more ε-moves.
- **Start State (q_0'):** $q_0' = E(\{q_0\})$. We begin by calculating the ε-closure of the NFA's start state.
- **Transition function:** For $R \in Q'$ and $a \in \Sigma$,

$$\delta_M(R, a) = E\left(\bigcup_{r \in R} \delta_N(r, a)\right).$$

The Subset Construction Algorithm

Let $N = (Q, \Sigma, \delta_N, q_0, F_N)$ be an NFA. We build an equivalent DFA $M = (Q', \Sigma, \delta_M, q_0', F')$.

- **States:** Begin with $Q' = \mathcal{P}(Q)$ conceptually, then retain only subsets reachable from the start subset.
- **ε-closure:** $E(R)$ contains the states reachable from R using zero or more ε-moves.
- **Start State (q_0'):** $q_0' = E(\{q_0\})$. We begin by calculating the ε-closure of the NFA's start state.
- **Transition function:** For $R \in Q'$ and $a \in \Sigma$,

$$\delta_M(R, a) = E\left(\bigcup_{r \in R} \delta_N(r, a)\right).$$

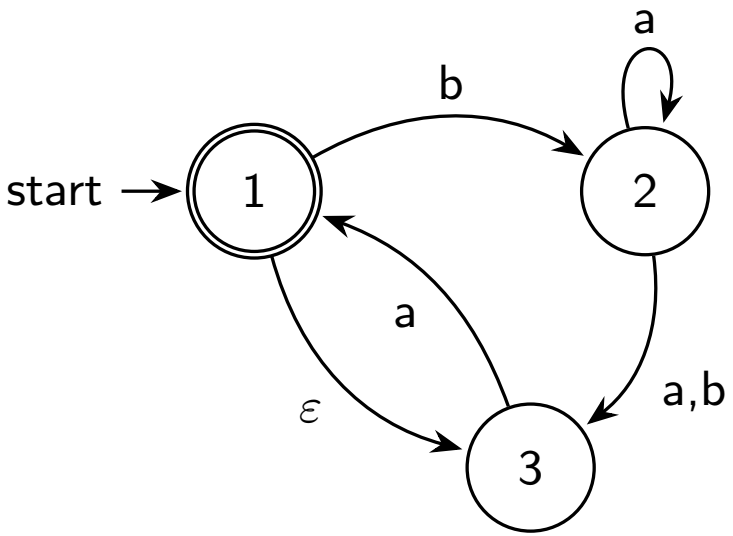
- **Accepting states:** $F' = \{R \in Q' : R \cap F_N \neq \emptyset\}$.

Correctness Invariant

After reading w, the DFA is in exactly the set of NFA states reachable from q_0 by paths whose non-ε labels spell w.

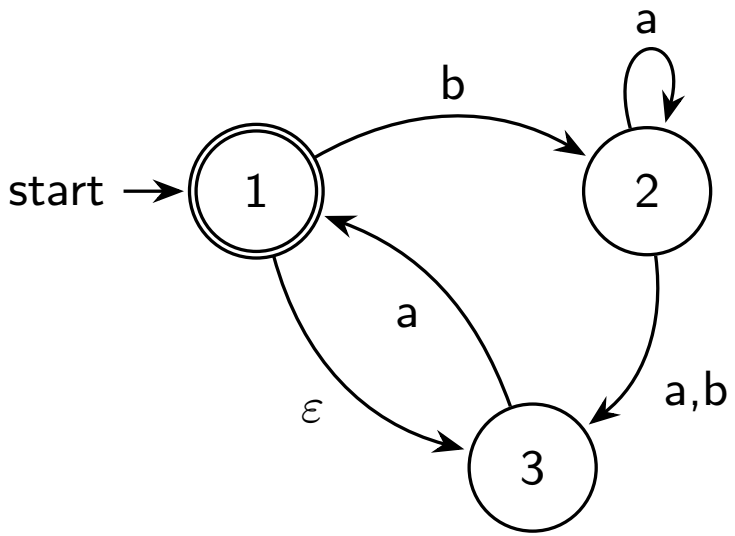
Worked Example: NFA to DFA

Consider this ε-NFA over $\{a, b\}$, with states $1, 2, 3$:



Worked Example: NFA to DFA

Consider this ε-NFA over $\{a, b\}$, with states $1, 2, 3$:



Start the subset construction:

- **Start State:** $E(\{1\}) = \{1, 3\}$ (since $1 \xrightarrow{\varepsilon} 3$).
- **From** $\{1, 3\}$ **on** a**:** $\delta_N(1, a) = \emptyset$, $\delta_N(3, a) = \{1\}$. Closure $E(\{1\}) = \{1, 3\}$. So, $\delta_M(\{1, 3\}, a) = \{1, 3\}$.
- **From** $\{1, 3\}$ **on** b**:** $\delta_N(1, b) = \{2\}$, $\delta_N(3, b) = \emptyset$. Closure $E(\{2\}) = \{2\}$. So, $\delta_M(\{1, 3\}, b) = \{2\}$.

Worked Example: NFA to DFA (Completed)

Continuing until no new subset appears gives the following six reachable DFA states.

DFA state	a	b	accepting?
$A = \{1, 3\}$	A	B	✓
$B = \{2\}$	C	D	
$C = \{2, 3\}$	E	D	
$D = \{3\}$	A	$\varnothing$	
$E = \{1, 2, 3\}$	E	C	✓
$\varnothing$	$\varnothing$	$\varnothing$	

Worked Example: NFA to DFA (Completed)

Continuing until no new subset appears gives the following six reachable DFA states.

DFA state	a	b	accepting?
$A = \{1, 3\}$	A	B	✓
$B = \{2\}$	C	D	
$C = \{2, 3\}$	E	D	
$D = \{3\}$	A	$\varnothing$	
$E = \{1, 2, 3\}$	E	C	✓
$\varnothing$	$\varnothing$	$\varnothing$	

A is the start state. Reachability witnesses for the rows are ε, b, ba, bb, baa, bbb, respectively. Exactly A and E contain the NFA's accepting state 1.

Stop when the reachable-subset worklist is empty

Process every new subset on every alphabet symbol, reusing a subset already seen. Only six of the $2^3 = 8$ possible subsets occur here. The empty subset is a single DFA state, not an omitted transition.

Optional state diagram in appendix

Part 5: Motivation for State Reduction

Self-Study Header: Finding the Smallest Equivalent DFA

For a given regular language, there are infinitely many DFAs that recognize it. For practical applications (like compiling and text searching), we want the **minimal** DFA—the one with the fewest possible states.

Part 5: Motivation for State Reduction

Self-Study Header: Finding the Smallest Equivalent DFA

For a given regular language, there are infinitely many DFAs that recognize it. For practical applications (like compiling and text searching), we want the **minimal** DFA—the one with the fewest possible states.

Indistinguishable States

States p and q are **equivalent** (indistinguishable), written $p \equiv q$, if for every suffix $w \in \Sigma^*$,

$$\delta^*(p, w) \in F \iff \delta^*(q, w) \in F.$$

Otherwise they are distinguishable; any witnessing w is a **distinguishing string**.

Part 5: Motivation for State Reduction

Self-Study Header: Finding the Smallest Equivalent DFA

For a given regular language, there are infinitely many DFAs that recognize it. For practical applications (like compiling and text searching), we want the **minimal** DFA—the one with the fewest possible states.

Indistinguishable States

States p and q are **equivalent** (indistinguishable), written $p \equiv q$, if for every suffix $w \in \Sigma^*$,

$$\delta^*(p, w) \in F \iff \delta^*(q, w) \in F.$$

Otherwise they are distinguishable; any witnessing w is a **distinguishing string**.

Equivalent reachable states represent the same future behavior and may be merged without changing the recognized language.

Before Minimizing: Reachability Is Not Equivalence

A state q is **reachable** if $\delta^*(q_0, w) = q$ for some word w. Find reachable states by graph search from q_0, following every symbol.

Before Minimizing: Reachability Is Not Equivalence

A state q is **reachable** if $\delta^*(q_0, w) = q$ for some word w. Find reachable states by graph search from q_0, following every symbol.

- Remove unreachable states: no input run visits them.
- A reachable nonaccepting state may still lead to acceptance; being nonaccepting does not make a state useless.
- A **dead** state has no accepting continuation. Reachable dead states all have the same future behavior, so merge them into one rejecting sink rather than deleting them from a complete DFA.

Before Minimizing: Reachability Is Not Equivalence

A state q is **reachable** if $\delta^*(q_0, w) = q$ for some word w. Find reachable states by graph search from q_0, following every symbol.

- Remove unreachable states: no input run visits them.
- A reachable nonaccepting state may still lead to acceptance; being nonaccepting does not make a state useless.
- A **dead** state has no accepting continuation. Reachable dead states all have the same future behavior, so merge them into one rejecting sink rather than deleting them from a complete DFA.

The prefix-*ab* machine

Its trap is reached by b. Deleting the trap would leave the start's b-transition undefined. The intermediate state reached by a is not dead: suffix b leads to acceptance.

For a fixed alphabet, the minimal complete DFA for $\emptyset$ has one nonaccepting sink; the minimal one for Σ^* has one accepting sink.

The Table-Filling Algorithm

Work with a complete DFA. A marked pair has a distinguishing suffix.

1. **Initialize:** Remove all unreachable states.

The Table-Filling Algorithm

Work with a complete DFA. A marked pair has a distinguishing suffix.

1. **Initialize:** Remove all unreachable states.
2. **Initialize the table:** List every unordered pair $\{p, q\}$ and mark pairs with exactly one accepting state.

The Table-Filling Algorithm

Work with a complete DFA. A marked pair has a distinguishing suffix.

1. **Initialize:** Remove all unreachable states.
2. **Initialize the table:** List every unordered pair $\{p, q\}$ and mark pairs with exactly one accepting state.
3. **Iterate:** For every unmarked pair (p, q) and every symbol $a \in \Sigma$:
   - Compute $r = \delta(p, a)$ and $s = \delta(q, a)$.
   - If $r \neq s$ and $\{r, s\}$ is marked, mark $\{p, q\}$. If w witnesses the successor pair, aw witnesses this pair.

The Table-Filling Algorithm

Work with a complete DFA. A marked pair has a distinguishing suffix.

1. **Initialize:** Remove all unreachable states.
2. **Initialize the table:** List every unordered pair $\{p, q\}$ and mark pairs with exactly one accepting state.
3. **Iterate:** For every unmarked pair (p, q) and every symbol $a \in \Sigma$:
   - Compute $r = \delta(p, a)$ and $s = \delta(q, a)$.
   - If $r \neq s$ and $\{r, s\}$ is marked, mark $\{p, q\}$. If w witnesses the successor pair, aw witnesses this pair.
4. **Terminate:** Repeat step 3 until a full pass makes no new marks.
5. **Merge:** The remaining unmarked pairs are equivalent; collapse each equivalence class to one state.

Minimal-DFA Theorem

After unreachable states are removed and equivalent states are merged, the resulting complete DFA is minimal and is unique up to renaming states (over the same alphabet).

Minimization Example: Start with a Transition Table

Use the DFA below, with start q_0 and $F = \{q_2, q_4\}$.

state	0	1	a word reaching it
q_0	q_1	q_3	ε
q_1	q_2	q_4	0
q_2	q_1	q_4	00
q_3	q_2	q_4	1
q_4	q_4	q_4	01

Every state is reachable, so none can be removed at the first step.

Minimization Example: Start with a Transition Table

Use the DFA below, with start q_0 and $F = \{q_2, q_4\}$.

state	0	1	a word reaching it
q_0	q_1	q_3	ε
q_1	q_2	q_4	0
q_2	q_1	q_4	00
q_3	q_2	q_4	1
q_4	q_4	q_4	01

Every state is reachable, so none can be removed at the first step.

Predict before marking

q_1 and q_3 have identical successors and are both nonaccepting. They are equivalent. But q_2 and q_4, although both accepting, differ after input 0: q_2 rejects it and q_4 accepts it.

Identical transition rows with matching acceptance are sufficient for equivalence, but not necessary. Table filling handles the general case.

Minimization Example: Marking the Pairs

Put a distinguishing suffix in each marked cell. Initially, ε marks pairs with exactly one accepting state.

q_1				
q_2	ε	ε		
q_3			ε	
q_4	ε	ε		ε
	q_0	q_1	q_2	q_3

Minimization Example: Marking the Pairs

Put a distinguishing suffix in each marked cell. Initially, ε marks pairs with exactly one accepting state.

q_1	0			
q_2	ε	ε		
q_3	0	$\sim$	ε	
q_4	ε	ε	0	ε
	q_0	q_1	q_2	q_3

The three new marks have witness 0:

$$(q_0, q_1) \xrightarrow{0} (q_1, q_2), \quad (q_0, q_3) \xrightarrow{0} (q_1, q_2),$$
$$(q_2, q_4) \xrightarrow{0} (q_1, q_4).$$

Their successor pairs were already marked by ε.

The fixed point

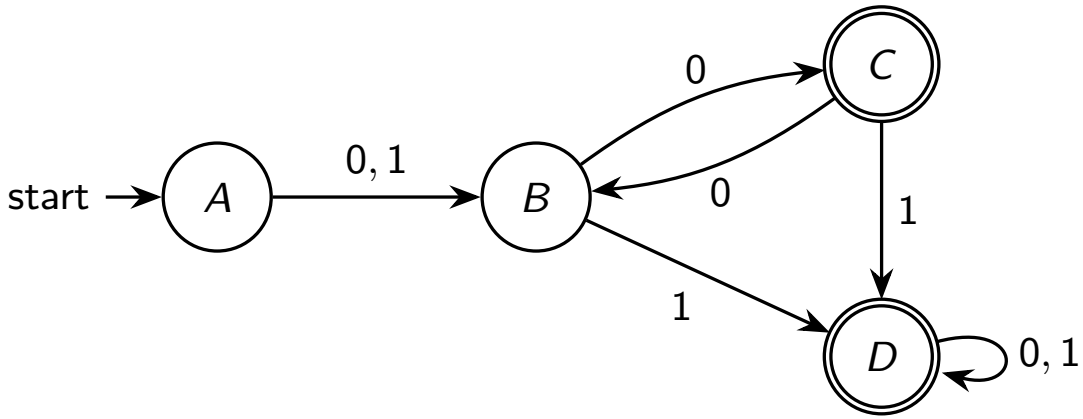
No further marks are possible. The cell $\sim$ remains unmarked: only $q_1 \equiv q_3$. A blank cell means equivalent only *after* the algorithm has reached its fixed point.

Minimization Example: The Quotient DFA

Create one state per equivalence class:

$$A = \{q_0\}, \quad B = \{q_1, q_3\}, \quad C = \{q_2\}, \quad D = \{q_4\}.$$

Start at A; accept at C, D. Each transition goes to the class of the old successor: $\delta_{\text{min}}([q], a) = [\delta(q, a)]$.

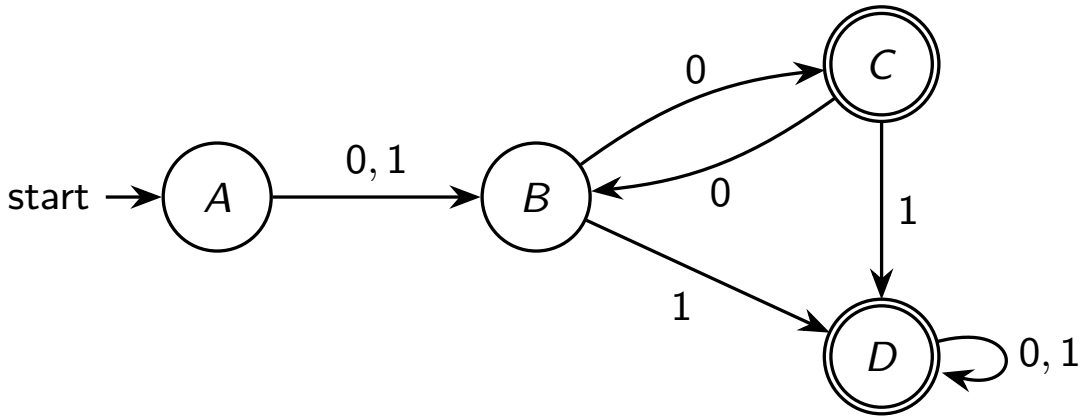


Minimization Example: The Quotient DFA

Create one state per equivalence class:

$$A = \{q_0\}, \quad B = \{q_1, q_3\}, \quad C = \{q_2\}, \quad D = \{q_4\}.$$

Start at A; accept at C, D. Each transition goes to the class of the old successor: $\delta_{\text{min}}([q], a) = [\delta(q, a)]$.



Minimality certificate. All four states are reachable. ε distinguishes every accepting/nonaccepting pair; 0 distinguishes A from B and C from D. Thus no two reachable states have the same future behavior.

Checkpoint Exercises

1. **DFA Design:** Draw the state diagram of a DFA over the alphabet $\Sigma = \{0, 1\}$ that recognizes the language: $\{w \mid w \text{ contains at least two 0s}\}$.

2. **NFA Trace:** Consider an NFA with states $\{q_0, q_1, q_2\}$. The start state is q_0.
   - $\delta(q_0, \varepsilon) = \{q_1\}$
   - $\delta(q_1, \varepsilon) = \{q_2\}$
   - $\delta(q_0, a) = \{q_0\}$

   What is the ε-closure of the start state q_0? That is, find $E(\{q_0\})$.

3. **Minimization Logic:** A complete DFA has exactly two reachable states, one accepting and one nonaccepting. Can these states be merged? Give a distinguishing string.

Common Mistakes and Quick Diagnostic Questions

- **Missing DFA edge:** is there exactly one successor for every state-symbol pair?
- **Premature acceptance:** what state or state set remains after the *last* input symbol?
- **Incorrect NFA rejection:** did all possible computations fail, or only the one you happened to follow?
- **Lost empty moves:** did you close the initial set and every set obtained after consuming a symbol?

Common Mistakes and Quick Diagnostic Questions

- **Missing DFA edge:** is there exactly one successor for every state-symbol pair?
- **Premature acceptance:** what state or state set remains after the *last* input symbol?
- **Incorrect NFA rejection:** did all possible computations fail, or only the one you happened to follow?
- **Lost empty moves:** did you close the initial set and every set obtained after consuming a symbol?
- **Wrong merge:** do two states agree for every suffix, not merely on being accepting now?
- **Wrong deletion:** is a state unreachable, or merely unable to accept? A complete DFA must retain a destination for every input.
- **Premature minimality:** have you removed unreachable states and continued marking until no new pair is distinguishable?

Summary: Design, Simulate, Convert, Minimize

- A DFA has one run per word. Its current state summarizes the relevant input history; acceptance is tested at the end.

Summary: Design, Simulate, Convert, Minimize

- A DFA has one run per word. Its current state summarizes the relevant input history; acceptance is tested at the end.
- An NFA accepts if some finite path consumes the whole word and finishes in an accepting state. State-set simulation accounts for every possible path, including ε-moves.

Summary: Design, Simulate, Convert, Minimize

- A DFA has one run per word. Its current state summarizes the relevant input history; acceptance is tested at the end.
- An NFA accepts if some finite path consumes the whole word and finishes in an accepting state. State-set simulation accounts for every possible path, including ε-moves.
- Subset construction turns those state sets into DFA states, preserving the language but possibly increasing the representation size.

Summary: Design, Simulate, Convert, Minimize

- A DFA has one run per word. Its current state summarizes the relevant input history; acceptance is tested at the end.
- An NFA accepts if some finite path consumes the whole word and finishes in an accepting state. State-set simulation accounts for every possible path, including ε-moves.
- Subset construction turns those state sets into DFA states, preserving the language but possibly increasing the representation size.
- Minimization discards unreachable states and merges states with identical future acceptance behavior. Distinguishing suffixes certify which states must remain separate.

Next steps

Slide Set 3 connects finite automata with regular expressions and regular grammars. Slide Set 4 studies closure, decision properties, and the limits of finite memory. The appendix here supplies optional depth.

Appendix Guide: Choose a Teaching Route

The main lecture retains the definitions, core state-design examples, subset-construction table, and complete marking/minimization sequence.

Supplementary material moved out of the main lecture

The detailed parity proof; the additional suffix-01 example; the determinized state diagram; and the further-practice sheet are now here. The table and its optional diagram have navigation links.

- **Practice route:** worked solutions, binary addition and comparison, and the shortest accepted-word challenge.
- **Proof route:** subset correctness, epsilon removal, marking correctness, partition refinement, and Myhill–Nerode.
- **Enrichment route:** double-reversal minimization, exponential state gaps, NFA lower bounds, and algorithmic costs.

These are optional extensions, not an additional required lecture. Closure properties and pumping arguments are left to Slide Set 4.

A Trace and a Proof: Odd Number of Ones

Recall the parity DFA: start even, toggle on 1, and stay on 0. Only the odd state accepts. Trace 0110:

prefix read	ε	0	01	011	0110
state	q_{even}	q_{even}	q_{odd}	q_{even}	q_{even}

The full word is rejected, despite the earlier visit to q_{odd}.

A Trace and a Proof: Odd Number of Ones

Recall the parity DFA: start even, toggle on 1, and stay on 0. Only the odd state accepts. Trace 0110:

prefix read	ε	0	01	011	0110
state	q_{even}	q_{even}	q_{odd}	q_{even}	q_{even}

The full word is rejected, despite the earlier visit to q_{odd}.

Inductive invariant

After reading w, the state is q_{even} iff $|w|_1$ is even, and q_{odd} iff $|w|_1$ is odd. Here $|w|_1$ counts the 1's.

- **Base:** $w = \varepsilon$ has zero 1's, so the start is correct.
- **Step:** appending 0 preserves parity; appending 1 flips it. The transitions implement exactly these updates.

A Trace and a Proof: Odd Number of Ones

Recall the parity DFA: start even, toggle on 1, and stay on 0. Only the odd state accepts. Trace 0110:

prefix read	ε	0	01	011	0110
state	q_{even}	q_{even}	q_{odd}	q_{even}	q_{even}

The full word is rejected, despite the earlier visit to q_{odd}.

Inductive invariant

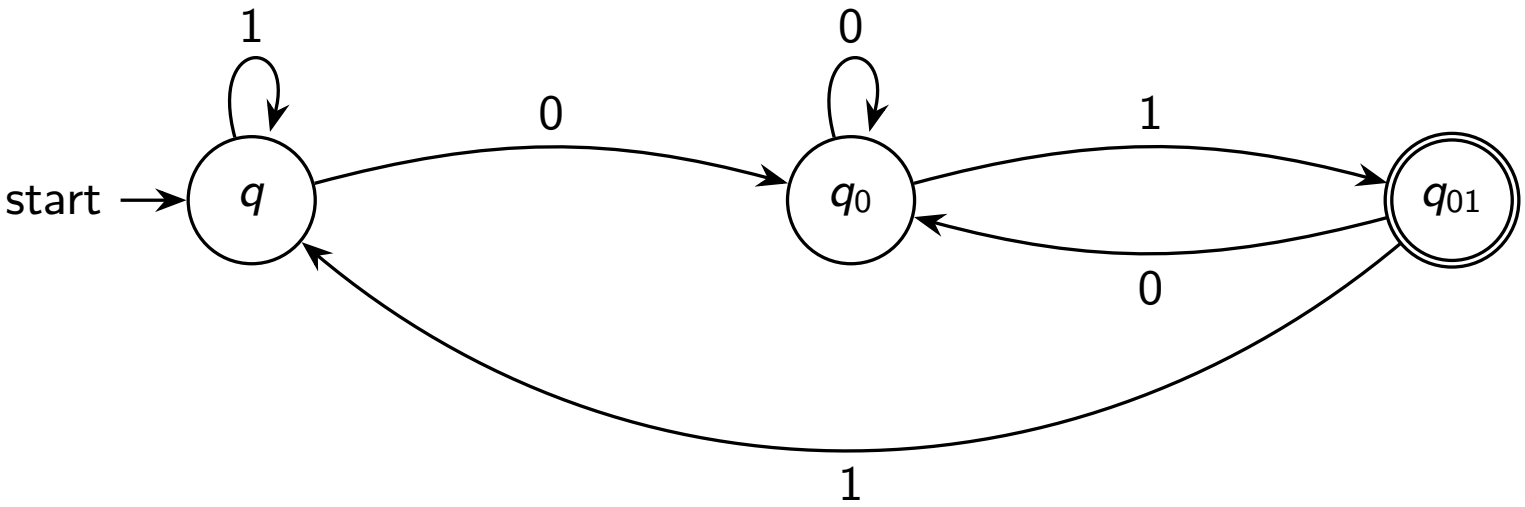
After reading w, the state is q_{even} iff $|w|_1$ is even, and q_{odd} iff $|w|_1$ is odd. Here $|w|_1$ counts the 1's.

- **Base:** $w = \varepsilon$ has zero 1's, so the start is correct.
- **Step:** appending 0 preserves parity; appending 1 flips it. The transitions implement exactly these updates.

Since only the odd state accepts, the invariant proves that the DFA accepts exactly the intended language, for every input length.

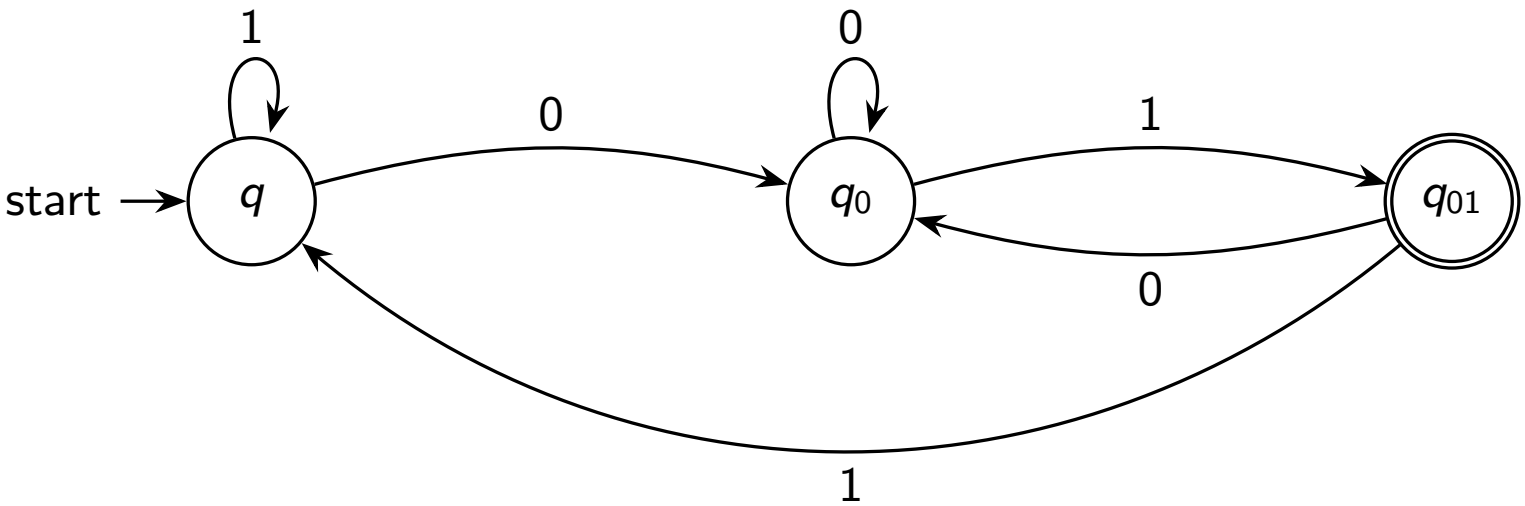
Designing DFAs: Ending in 01 Is Different

Goal: Accept binary words whose final two symbols are 01. A previously matched 01 may cease to be a suffix after more input.



Designing DFAs: Ending in 01 Is Different

Goal: Accept binary words whose final two symbols are 01. A previously matched 01 may cease to be a suffix after more input.



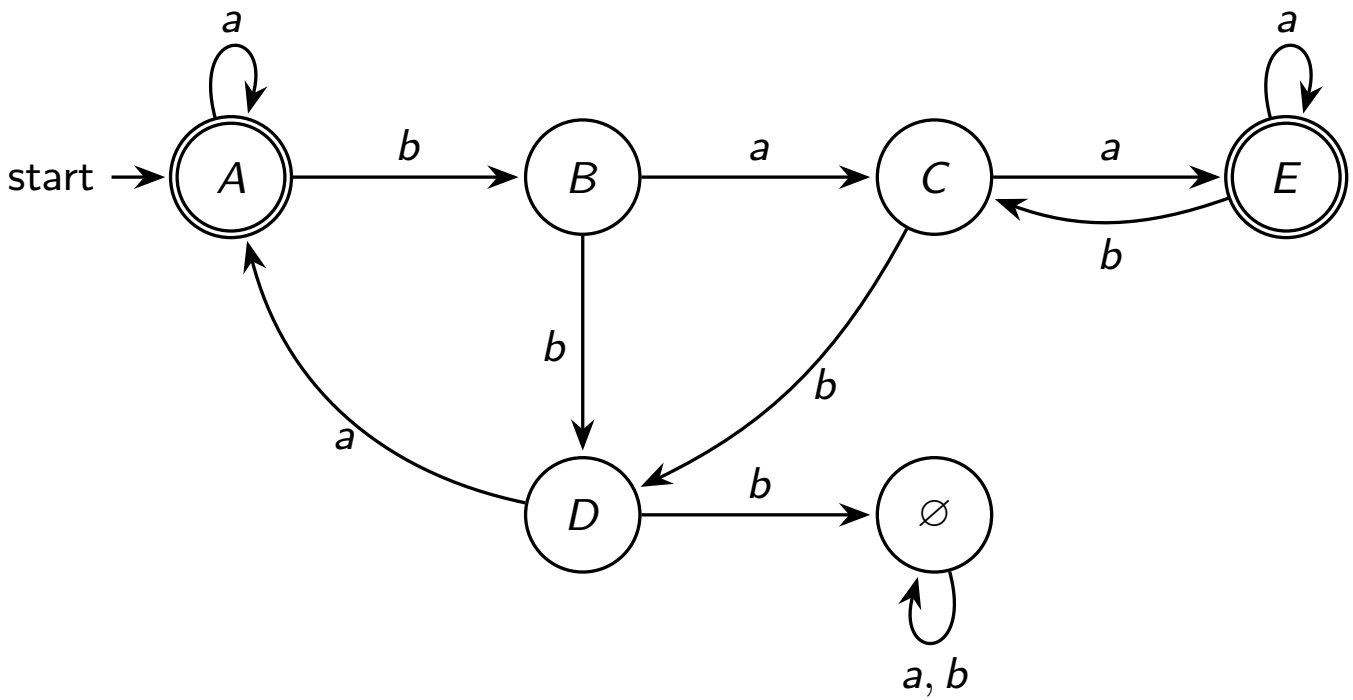
The states remember the longest suffix that is a prefix of 01: ε, 0, or 01. Test 01 (accept), 010 (reject), and 0101 (accept).

Three different specifications

Begins with, *contains*, and *ends with* are not interchangeable. Once a required substring occurs, success persists; a suffix condition must be checked again after every new symbol.

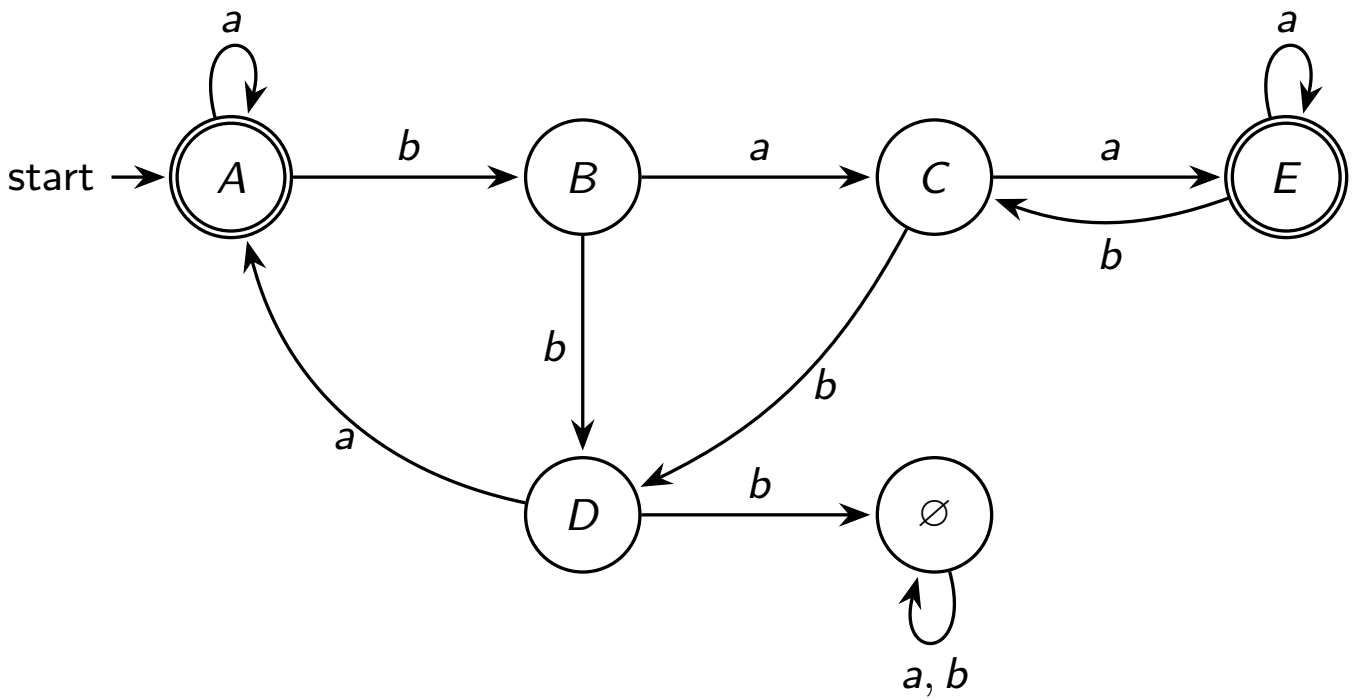
Worked Example: The Determinized State Diagram

This implements the main lecture's subset table; start at A. Here $A = \{1, 3\}$, $B = \{2\}$, $C = \{2, 3\}$, $D = \{3\}$, and $E = \{1, 2, 3\}$. Return to table



Worked Example: The Determinized State Diagram

This implements the main lecture's subset table; start at A. Here $A = \{1, 3\}$, $B = \{2\}$, $C = \{2, 3\}$, $D = \{3\}$, and $E = \{1, 2, 3\}$. Return to table



For example,

$$A \xrightarrow{b} B \xrightarrow{a} C \xrightarrow{a} E \quad \text{accepts } baa,$$

while $A \xrightarrow{b} B \xrightarrow{b} D \xrightarrow{b} \varnothing$ rejects bbb. The DFA also accepts ε, since A is accepting. Subset construction guarantees equivalence, not minimality.

Solutions to the Checkpoint Exercises

1. At least two zeros. States z_0, z_1, z_2 remember $\min(2, |w|_0)$. Start at z_0 and accept only at z_2:

	0	1
z_0	z_1	z_0
z_1	z_2	z_1
z_2	z_2	z_2

The invariant follows by induction. The states are all reachable; suffix 0 distinguishes z_0, z_1, and ε distinguishes z_2 from each other state. Three states are necessary.

Solutions to the Checkpoint Exercises

1. At least two zeros. States z_0, z_1, z_2 remember $\min(2, |w|_0)$. Start at z_0 and accept only at z_2:

	0	1
z_0	z_1	z_0
z_1	z_2	z_1
z_2	z_2	z_2

The invariant follows by induction. The states are all reachable; suffix 0 distinguishes z_0, z_1, and ε distinguishes z_2 from each other state. Three states are necessary.

2. Epsilon-closure. $E(\{q_0\}) = \{q_0, q_1, q_2\}$, since zero, one, or two ε-moves reach these states.

Solutions to the Checkpoint Exercises

1. At least two zeros. States z_0, z_1, z_2 remember $\min(2, |w|_0)$. Start at z_0 and accept only at z_2:

	0	1
z_0	z_1	z_0
z_1	z_2	z_1
z_2	z_2	z_2

The invariant follows by induction. The states are all reachable; suffix 0 distinguishes z_0, z_1, and ε distinguishes z_2 from each other state. Three states are necessary.

2. Epsilon-closure. $E(\{q_0\}) = \{q_0, q_1, q_2\}$, since zero, one, or two ε-moves reach these states.

3. Two states of opposite acceptance. They cannot be merged: ε is a distinguishing suffix. Their outgoing transitions cannot change that fact.

Further Practice: Design, Trace, and Justify

1. Read a binary word most significant bit first. Design a DFA that accepts exactly when its numerical value is divisible by 3. For this exercise, allow leading zeros and define the empty word's value to be 0. State and prove the state invariant.

Further Practice: Design, Trace, and Justify

1. Read a binary word most significant bit first. Design a DFA that accepts exactly when its numerical value is divisible by 3. For this exercise, allow leading zeros and define the empty word's value to be 0. State and prove the state invariant.
2. Construct a DFA for words ending in 101. How does it differ from a DFA for words containing 101? Check 1010 and 10101.

Further Practice: Design, Trace, and Justify

1. Read a binary word most significant bit first. Design a DFA that accepts exactly when its numerical value is divisible by 3. For this exercise, allow leading zeros and define the empty word's value to be 0. State and prove the state invariant.
2. Construct a DFA for words ending in 101. How does it differ from a DFA for words containing 101? Check 1010 and 10101.
3. A unary DFA has $q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_3$, with a loop on q_3. Start at q_0 and accept only at q_3. Find a shortest suffix distinguishing q_0 and q_1. Explain why only marking accepting/nonaccepting pairs is insufficient.

What to submit

A complete transition table or diagram, the start and accepting states, and an invariant or distinguishing-suffix argument. Solutions follow on the next three slides; these are original textbook-style exercises.

Solution: Binary Values Modulo Three

Appending bit $b \in \{0, 1\}$ changes a binary value v into $2v + b$. Use states r_0, r_1, r_2 for the remainder modulo 3:

$$\delta(r_i, b) = r_{(2i+b) \bmod 3},$$

	0	1
r_0	r_0	r_1
r_1	r_2	r_0
r_2	r_1	r_2

Start at r_0; accept only at r_0.

Solution: Binary Values Modulo Three

Appending bit $b \in \{0,1\}$ changes a binary value v into $2v + b$. Use states r_0, r_1, r_2 for the remainder modulo 3:

$$\delta(r_i, b) = r_{(2i+b) \bmod 3},$$

	0	1
r_0	r_0	r_1
r_1	r_2	r_0
r_2	r_1	r_2

Start at r_0; accept only at r_0.

Proof and trace

Initially the value is 0. The update formula preserves the invariant "the state is the remainder of the prefix's value." Therefore the accepting condition is exactly divisibility by 3. For 110 (value 6): $r_0 \xrightarrow{1} r_1 \xrightarrow{1} r_0 \xrightarrow{0} r_0$.

Solution: Binary Values Modulo Three

Appending bit $b \in \{0,1\}$ changes a binary value v into $2v + b$. Use states r_0, r_1, r_2 for the remainder modulo 3:

$$\delta(r_i, b) = r_{(2i+b) \bmod 3},$$

	0	1
r_0	r_0	r_1
r_1	r_2	r_0
r_2	r_1	r_2

Start at r_0; accept only at r_0.

Proof and trace

Initially the value is 0. The update formula preserves the invariant "the state is the remainder of the prefix's value." Therefore the accepting condition is exactly divisibility by 3. For 110 (value 6): $r_0 \xrightarrow{1} r_1 \xrightarrow{1} r_0 \xrightarrow{0} r_0$.

Leading zeros leave the value unchanged. If a specification forbids the empty word, use a new nonaccepting start with the same outgoing transitions as r_0 and no incoming edges. Do not make r_0 nonaccepting: it must still accept nonempty representations of multiples of 3.

Solution: Ending in 101 versus Containing 101

For the suffix condition, remember the longest suffix that is a prefix of 101. Start at s_0; accept only at s_3:

state	remembered suffix	0	1
s_0	ε	s_0	s_1
s_1	1	s_2	s_1
s_2	10	s_0	s_3
s_3	101	s_2	s_1

Solution: Ending in 101 versus Containing 101

For the suffix condition, remember the longest suffix that is a prefix of 101. Start at s_0; accept only at s_3:

state	remembered suffix	0	1
s_0	ε	s_0	s_1
s_1	1	s_2	s_1
s_2	10	s_0	s_3
s_3	101	s_2	s_1

After 1010, the relevant suffix is 10, not ε. The next 1 therefore completes another match: 10101 is accepted. 1010 itself is rejected.

Solution: Ending in 101 versus Containing 101

For the suffix condition, remember the longest suffix that is a prefix of 101. Start at s_0; accept only at s_3:

state	remembered suffix	0	1
s_0	ε	s_0	s_1
s_1	1	s_2	s_1
s_2	10	s_0	s_3
s_3	101	s_2	s_1

After 1010, the relevant suffix is 10, not ε. The next 1 therefore completes another match: 10101 is accepted. 1010 itself is rejected.

Changing the specification changes one row

To recognize words *containing* 101, keep the first three rows and make s_3 an accepting sink: $\delta(s_3, 0) = \delta(s_3, 1) = s_3$. Now 1010 is accepted because an earlier match cannot be undone.

The invariant justifies each fallback transition and handles overlapping occurrences automatically.

Solution: Why Marking Must Continue

The unary chain accepts exactly the words of length at least 3:

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_3, \qquad \delta(q_3, a) = q_3, \qquad F = \{q_3\}.$$

Imagine synchronous rounds that use only marks from earlier rounds:

1. **Length 0:** ε marks every pair containing q_3.

Solution: Why Marking Must Continue

The unary chain accepts exactly the words of length at least 3:

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_3, \qquad \delta(q_3, a) = q_3, \qquad F = \{q_3\}.$$

Imagine synchronous rounds that use only marks from earlier rounds:

1. **Length 0:** ε marks every pair containing q_3.
2. **Length 1:** a marks (q_0, q_2) and (q_1, q_2). Their successors have opposite acceptance.

Solution: Why Marking Must Continue

The unary chain accepts exactly the words of length at least 3:

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_3, \qquad \delta(q_3, a) = q_3, \qquad F = \{q_3\}.$$

Imagine synchronous rounds that use only marks from earlier rounds:

1. **Length 0:** ε marks every pair containing q_3.
2. **Length 1:** a marks (q_0, q_2) and (q_1, q_2). Their successors have opposite acceptance.
3. **Length 2:** aa marks (q_0, q_1), since their successors (q_1, q_2) are distinguished by a.

Solution: Why Marking Must Continue

The unary chain accepts exactly the words of length at least 3:

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_3, \qquad \delta(q_3, a) = q_3, \qquad F = \{q_3\}.$$

Imagine synchronous rounds that use only marks from earlier rounds:

1. **Length 0:** ε marks every pair containing q_3.
2. **Length 1:** a marks (q_0, q_2) and (q_1, q_2). Their successors have opposite acceptance.
3. **Length 2:** aa marks (q_0, q_1), since their successors (q_1, q_2) are distinguished by a.

Neither ε nor a distinguishes q_0, q_1, so aa is shortest. All four states are reachable and pairwise distinguishable: the DFA is minimal.

Implementation detail

Updating marks immediately can propagate several rounds in one pass, depending on scanning order. Always use the fixed-point stopping rule; do not assume that one scan is enough.

Textbook Challenge: Checking Binary Addition

Adapted from Sipser's binary-addition problem, listed as Exercise 1.32 in the cited course handout [PSU13]. Each input symbol is a column of three bits:

$$\Sigma_3 = \left\{ \begin{bmatrix} x \\ y \\ z \end{bmatrix} : x, y, z \in \{0, 1\} \right\}.$$

Read columns *most significant first*. The three equal-length rows represent integers A, B, C. Leading zeros are allowed; empty rows mean zero. Build a DFA accepting exactly when $A + B = C$ (ordinary addition, not modulo).

Textbook Challenge: Checking Binary Addition

Adapted from Sipser's binary-addition problem, listed as Exercise 1.32 in the cited course handout [PSU13]. Each input symbol is a column of three bits:

$$\Sigma_3 = \left\{ \begin{bmatrix} x \\ y \\ z \end{bmatrix} : x, y, z \in \{0, 1\} \right\}.$$

Read columns *most significant first*. The three equal-length rows represent integers A, B, C. Leading zeros are allowed; empty rows mean zero. Build a DFA accepting exactly when $A + B = C$ (ordinary addition, not modulo).

A three-state solution without guessing a carry

Track $d = C_{\text{prefix}} - A_{\text{prefix}} - B_{\text{prefix}}$. On column (x, y, z), update $d' = 2d + z - x - y$. Keep states $d = 0$, $d = 1$, and a rejecting sink $\perp$ for all other values. Start at 0; only 0 accepts.

Why only these values? If $d \leq -1$, every next value is at most -1; if $d \geq 2$, every next value is at least 2. Neither can return to zero.

Binary Addition: Table, Trace, and Correctness

In the table, *xyz* denotes *one* column symbol, not three input steps.

	000	001	010	011	100	101	110	111
$\rightarrow *0$	0	1	$\perp$	0	$\perp$	0	$\perp$	$\perp$
1	$\perp$	$\perp$	1	$\perp$	1	$\perp$	0	1
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Binary Addition: Table, Trace, and Correctness

In the table, *xyz* denotes *one* column symbol, not three input steps.

	000	001	010	011	100	101	110	111
$\rightarrow *0$	0	1	$\perp$	0	$\perp$	0	$\perp$	$\perp$
1	$\perp$	$\perp$	1	$\perp$	1	$\perp$	0	1
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Two columns are enough to see the distinction

$01 + 01 = 10$: columns $001, 110$ give $0 \rightarrow 1 \rightarrow 0$; accept.

$01 + 01 \neq 11$: columns $001, 111$ give $0 \rightarrow 1 \rightarrow 1$; reject.

Binary Addition: Table, Trace, and Correctness

In the table, *xyz* denotes *one* column symbol, not three input steps.

	000	001	010	011	100	101	110	111
$\to *0$	0	1	$\perp$	0	$\perp$	0	$\perp$	$\perp$
1	$\perp$	$\perp$	1	$\perp$	1	$\perp$	0	1
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Two columns are enough to see the distinction

$01 + 01 = 10$: columns $001, 110$ give $0 \to 1 \to 0$; accept.

$01 + 01 \neq 11$: columns $001, 111$ give $0 \to 1 \to 1$; reject.

Proof. Before the sink is reached, induction on the columns shows that the state equals the numerical prefix difference. Appending a bit doubles each prefix value and adds that bit, giving the update formula. Once outside $\{0, 1\}$, the difference can never become zero. Hence the final state is 0 exactly when the full rows satisfy the equation.

The empty input is accepted ($0 + 0 = 0$). Add leading zero columns when more space is needed to represent the sum. Source: [PSU13].

Textbook Extension: A Binary Comparison with a Gap

Adapted from a UCSD extension of Sipser's comparison exercise [UCSD08]: read two-bit columns (x, y) most significant first; accept iff the top value U is at least the bottom value V plus 2. Leading zeros are allowed; empty rows have value zero.

Textbook Extension: A Binary Comparison with a Gap

Adapted from a UCSD extension of Sipser's comparison exercise [UCSD08]: read two-bit columns (x, y) most significant first; accept iff the top value U is at least the bottom value V plus 2. Leading zeros are allowed; empty rows have value zero. Track $d = U_{\text{prefix}} - V_{\text{prefix}}$, updated by $d' = 2d + x - y$. Four categories suffice: $L : d < 0$, $E : d = 0$, $P : d = 1$, and $G : d \geq 2$.

	00	01	10	11
L	L	L	L	L
$\rightarrow E$	E	L	P	E
P	G	P	G	G
$*G$	G	G	G	G

Textbook Extension: A Binary Comparison with a Gap

Adapted from a UCSD extension of Sipser's comparison exercise [UCSD08]: read two-bit columns (x, y) most significant first; accept iff the top value U is at least the bottom value V plus 2. Leading zeros are allowed; empty rows have value zero. Track $d = U_{\text{prefix}} - V_{\text{prefix}}$, updated by $d' = 2d + x - y$. Four categories suffice: $L : d < 0$, $E : d = 0$, $P : d = 1$, and $G : d \geq 2$.

	00	01	10	11
L	L	L	L	L
$\rightarrow E$	E	L	P	E
P	G	P	G	G
$*G$	G	G	G	G

Correctness invariant: the state is exactly the category of the prefix difference. Negative differences stay negative; values at least two stay at least two. The table implements the other cases explicitly.

$1110_2 - 1011_2 = 3$: $E \rightarrow E \rightarrow P \rightarrow G \rightarrow G$; accept. $1100_2 - 1011_2 = 1$: $E \rightarrow E \rightarrow P \rightarrow P \rightarrow P$; reject.

Challenge: A Bound on the Shortest Accepted Word

A useful witness bound

If an n-state ε-NFA accepts any word, it accepts a word of length at most $n - 1$. This also applies to a DFA.

Challenge: A Bound on the Shortest Accepted Word

A useful witness bound

If an n-state ε-NFA accepts any word, it accepts a word of length at most $n - 1$. This also applies to a DFA.

Proof. Choose an accepting path with the fewest edges. If a state repeats, remove the intervening cycle to obtain a shorter accepting path, a contradiction. Thus the path visits at most n states and uses at most $n - 1$ edges. An edge consumes at most one symbol, so the accepted word has length at most $n - 1$.

Challenge: A Bound on the Shortest Accepted Word

A useful witness bound

If an n-state ε-NFA accepts any word, it accepts a word of length at most $n-1$. This also applies to a DFA.

Proof. Choose an accepting path with the fewest edges. If a state repeats, remove the intervening cycle to obtain a shorter accepting path, a contradiction. Thus the path visits at most n states and uses at most $n-1$ edges. An edge consumes at most one symbol, so the accepted word has length at most $n-1$.

Tightness and a boundary-case trap

An NFA consisting of an n-state chain of a-edges, with only its final state accepting, accepts exactly a^{n-1}. The bound is attained.

Do not replace "a word" by "a nonempty word": an n-state unary cycle with only its start accepting has shortest *nonempty* accepted word a^n, but already accepts ε.

This is an original challenge using the finite-path reasoning underlying automata algorithms; it is not a numbered textbook quotation.

Why Subset Construction Preserves the Language

Let R_w be the NFA's reachable set after w, including empty moves. The claim is

$$\delta_M^*(E(\{q_0\}), w) = R_w.$$

Induction on the length of the input

Base: for ε, both sides are $E(\{q_0\})$.

Step: assume the statement for w. A path spelling wa consists of a path spelling w, one a-transition from a state in R_w, and zero or more empty moves. Taking the union and then closure gives exactly R_{wa}, which is the DFA's next subset.

Why Subset Construction Preserves the Language

Let R_w be the NFA's reachable set after w, including empty moves. The claim is

$$\delta_M^*(E(\{q_0\}), w) = R_w.$$

Induction on the length of the input

Base: for ε, both sides are $E(\{q_0\})$.

Step: assume the statement for w. A path spelling wa consists of a path spelling w, one a-transition from a state in R_w, and zero or more empty moves. Taking the union and then closure gives exactly R_{wa}, which is the DFA's next subset.

A DFA subset accepts iff it contains an NFA accepting state. Thus

$$w \in L(M) \iff R_w \cap F_N \neq \emptyset \iff w \in L(N).$$

Conversely, every DFA can be regarded as an NFA with singleton successors and no empty moves. Hence the models have exactly the same language power.

Removing Epsilon-Moves without Determinizing

An ε-NFA can be converted to an equivalent NFA with no ε-moves, keeping Q, Σ, q_0 unchanged. For $a \in \Sigma$, set

$$\delta_0(q, a) = E\left(\bigcup_{r \in E(\{q\})} \delta(r, a)\right), \qquad F_0 = \{q : E(\{q\}) \cap F \neq \emptyset\}.$$

Removing Epsilon-Moves without Determinizing

An ε-NFA can be converted to an equivalent NFA with no ε-moves, keeping Q, Σ, q_0 unchanged. For $a \in \Sigma$, set

$$\delta_0(q, a) = E\left(\bigcup_{r \in E(\{q\})} \delta(r, a)\right), \qquad F_0 = \{q : E(\{q\}) \cap F \neq \emptyset\}.$$

Why both changes are needed

Each new edge represents a path of empty moves, one input symbol, and more empty moves. Conversely, each such path becomes a new edge. Splitting an old successful computation at its consumed symbols proves equivalence for nonempty words.

For the empty word, $q_0 \in F_0$ exactly when an old accepting state is reachable from q_0 by empty moves alone.

For example, if $p \xrightarrow{\varepsilon} f$ and $f \in F$, then p must also become accepting. Merely erasing the empty edge would lose acceptance of ε when starting at p.

A Model Trap: What If Every State Accepts?

In a *complete DFA*, $F = Q$ implies $L = \Sigma^*$: every word has a full run, and its last state accepts.

A Model Trap: What If Every State Accepts?

In a *complete DFA*, $F = Q$ implies $L = \Sigma^*$: every word has a full run, and its last state accepts.

The same statement is false for an NFA

Over $\{a, b\}$, take one start/accept state q, an a-loop, and no b-transition. Every state accepts, but the language is only a^*. On input b, no path consumes the whole word.

A Model Trap: What If Every State Accepts?

In a *complete DFA*, $F = Q$ implies $L = \Sigma^*$: every word has a full run, and its last state accepts.

The same statement is false for an NFA

Over $\{a, b\}$, take one start/accept state q, an a-loop, and no b-transition. Every state accepts, but the language is only a^*. On input b, no path consumes the whole word.

What is still guaranteed?

Such an NFA accepts ε, and every prefix of an accepted word is accepted: stop its successful path after that prefix. Its current state is accepting too. This property is called *prefix closure*.

Accepting states do not rescue a computation that has no permitted move. Always distinguish "all states are accepting" from "a complete run exists for every word."

Why Table Filling and Merging Are Correct

Soundness of marks

An initial mark has witness ε. If a successor pair is distinguished by w, the predecessor pair is distinguished by aw. Therefore every marked pair really is distinguishable.

Why Table Filling and Merging Are Correct

Soundness of marks

An initial mark has witness ε. If a successor pair is distinguished by w, the predecessor pair is distinguished by aw. Therefore every marked pair really is distinguishable.

Completeness of marks

Induct on the length of a distinguishing word. Length zero is marked initially. A word aw distinguishes a pair only if w distinguishes its a-successors; once those are marked, the pair is marked too. Finitely many pairs exist, so iteration terminates.

Why Table Filling and Merging Are Correct

Soundness of marks

An initial mark has witness ε. If a successor pair is distinguished by w, the predecessor pair is distinguished by aw. Therefore every marked pair really is distinguishable.

Completeness of marks

Induct on the length of a distinguishing word. Length zero is marked initially. A word aw distinguishes a pair only if w distinguishes its a-successors; once those are marked, the pair is marked too. Finitely many pairs exist, so iteration terminates.

Consequently, the unmarked pairs at termination are exactly equivalent states. If $p \equiv q$, then $\delta(p, a) \equiv \delta(q, a)$ for each a: otherwise a suffix w for the successors would give suffix aw for p, q. Thus the quotient transition does not depend on which representative is chosen. Equivalent states also agree on acceptance (use ε).

Partition Refinement: Another View of Minimization

Instead of marking pairs, begin with the nonempty blocks F and $Q \setminus F$. **Within each current block**, group states by the blocks reached on each symbol. Repeat until the partition stops changing.

Partition Refinement: Another View of Minimization

Instead of marking pairs, begin with the nonempty blocks F and $Q \setminus F$. **Within each current block**, group states by the blocks reached on each symbol. Repeat until the partition stops changing. For the five-state minimization example in the main lecture, initially $N = \{q_0, q_1, q_3\}$ and $F = \{q_2, q_4\}$:

state	old block	0-successor block	1-successor block
q_0	N	N	N
q_1, q_3	N	F	F
q_2	F	N	F
q_4	F	F	F

This splits the partition into $\{q_0\}, \{q_1, q_3\}, \{q_2\}, \{q_4\}$; another round changes nothing.

Partition Refinement: Another View of Minimization

Instead of marking pairs, begin with the nonempty blocks F and $Q \setminus F$. **Within each current block**, group states by the blocks reached on each symbol. Repeat until the partition stops changing. For the five-state minimization example in the main lecture, initially $N = \{q_0, q_1, q_3\}$ and $F = \{q_2, q_4\}$:

state	old block	0-successor block	1-successor block
q_0	N	N	N
q_1, q_3	N	F	F
q_2	F	N	F
q_4	F	F	F

This splits the partition into $\{q_0\}, \{q_1, q_3\}, \{q_2\}, \{q_4\}$; another round changes nothing.

Refine, never merge old blocks

Although q_1 and q_4 have the same successor-block signature above, they disagree on acceptance and stay separate. This is Moore-style refinement; Hopcroft's algorithm uses a more efficient splitter worklist.

A Sharper Bound on Distinguishing Suffixes

Theorem

In one complete DFA with $n \geq 2$ states, every distinguishable pair has a distinguishing word of length at most $n - 2$.

A Sharper Bound on Distinguishing Suffixes

Theorem

In one complete DFA with $n \geq 2$ states, every distinguishable pair has a distinguishing word of length at most $n - 2$.

Let Π_i group states agreeing on *all* suffixes of length at most i. The initial partition Π_0 separates accepting and nonaccepting states. If any distinguishable pair exists, both blocks are nonempty.

- Refining by successor blocks turns Π_i into Π_{i+1}.
- Every nonstable round adds at least one block; there are at most n blocks, so at most $n - 2$ strict refinements are possible.
- Once a round makes no change, all later rounds agree as well. Thus Π_{n-2} already captures full state equivalence.

A Sharper Bound on Distinguishing Suffixes

Theorem

In one complete DFA with $n \geq 2$ states, every distinguishable pair has a distinguishing word of length at most $n - 2$.

Let Π_i group states agreeing on *all* suffixes of length at most i. The initial partition Π_0 separates accepting and nonaccepting states. If any distinguishable pair exists, both blocks are nonempty.

- Refining by successor blocks turns Π_i into Π_{i+1}.
- Every nonstable round adds at least one block; there are at most n blocks, so at most $n - 2$ strict refinements are possible.
- Once a round makes no change, all later rounds agree as well. Thus Π_{n-2} already captures full state equivalence.

Tight example: a unary chain $q_0 \to q_1 \to \cdots \to q_{n-1}$, with only q_{n-1} accepting and an a-loop there. The shortest word distinguishing q_0 and q_1 is a^{n-2}.

Enumerating all these words can be expensive; refinement shares the work.

Myhill–Nerode: The Meaning of a Minimal State

For a language L, define equivalence of prefixes by

$$x \equiv_L y \quad \Longleftrightarrow \quad \forall z \in \Sigma^*,\ \big(xz \in L \iff yz \in L\big).$$

Theorem

L is regular iff $\equiv_L$ has finitely many equivalence classes. Their number is the number of states in the minimal complete DFA.

Myhill–Nerode: The Meaning of a Minimal State

For a language L, define equivalence of prefixes by

$$x \equiv_L y \quad \Longleftrightarrow \quad \forall z \in \Sigma^*,\ \big(xz \in L \iff yz \in L\big).$$

Theorem

L is regular iff $\equiv_L$ has finitely many equivalence classes. Their number is the number of states in the minimal complete DFA.

Lower bound. If two prefixes have different possible futures, any recognizing DFA must reach different states on them. Thus every DFA needs at least one state for each prefix class.

Myhill–Nerode: The Meaning of a Minimal State

For a language L, define equivalence of prefixes by

$$x \equiv_L y \quad \Longleftrightarrow \quad \forall z \in \Sigma^*,\ \big(xz \in L \iff yz \in L\big).$$

Theorem

L is regular iff $\equiv_L$ has finitely many equivalence classes. Their number is the number of states in the minimal complete DFA.

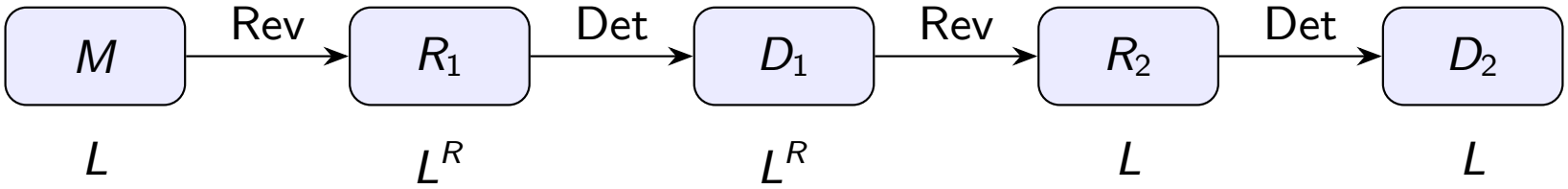
Lower bound. If two prefixes have different possible futures, any recognizing DFA must reach different states on them. Thus every DFA needs at least one state for each prefix class.

Construction. When there are finitely many classes, use them as states: start at $[\varepsilon]$, set $\delta([x], a) = [xa]$, and accept $[x]$ iff $x \in L$. The definition makes these choices well-defined.

These intrinsic classes explain minimality and uniqueness up to state names. State equivalence in a reachable DFA is their concrete realization.

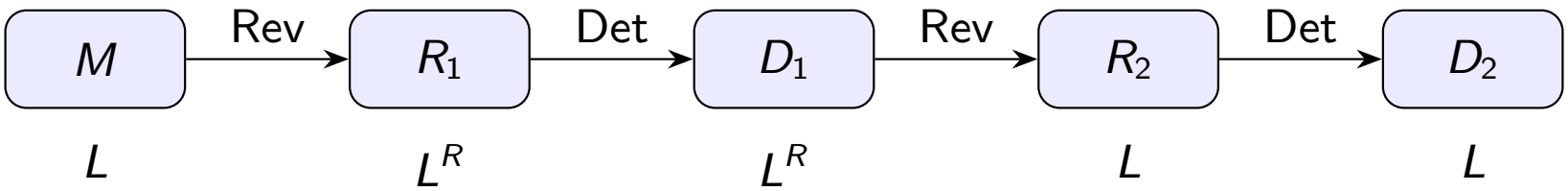
Brzozowski's Algorithm: Reverse and Determinize Twice

A surprisingly simple alternative to pair marking is double reversal with reachable-subset construction [B14].



Brzozowski's Algorithm: Reverse and Determinize Twice

A surprisingly simple alternative to pair marking is double reversal with reachable-subset construction [B14].



- **Rev:** reverse all edges; the old accepting states form the initial set, and the old start is the sole accepting state. A fresh start with epsilon edges can implement the initial set.
- **Det:** keep only reachable subsets, including $\emptyset$ if reached. An empty initial set becomes start state $\emptyset$.
- Starting with a DFA M, the final D_2 is the minimal complete DFA.

Elegant does not mean asymptotically faster

Intermediate subset automata may be exponentially large. This is not a replacement for Hopcroft's worst-case $O(n \log n)$ guarantee.

Why Double Reversal Produces a Minimal DFA

Key lemma

If A is a reachable DFA, then $\mathrm{Det}(\mathrm{Rev}(A))$ is minimal: its distinct reachable subset states are distinguishable.

Why Double Reversal Produces a Minimal DFA

Key lemma

If A is a reachable DFA, then $\mathrm{Det}(\mathrm{Rev}(A))$ is minimal: its distinct reachable subset states are distinguishable.

Take distinct subsets S, T and a state q belonging to exactly one of them. Since A is reachable, choose u with $\delta_A^*(q_0, u) = q$. In the reversed machine, a path labeled u^R leads from a state r to q_0 exactly when A has a path labeled u from q_0 to r. Determinism makes that endpoint uniquely q.

$$u^R \text{ is accepted from subset } S \quad \Longleftrightarrow \quad q \in S.$$

Thus u^R distinguishes S and T. All output states are reachable, so the DFA is minimal.

Why Double Reversal Produces a Minimal DFA

Key lemma

If A is a reachable DFA, then Det(Rev(A)) is minimal: its distinct reachable subset states are distinguishable.

Take distinct subsets S, T and a state q belonging to exactly one of them. Since A is reachable, choose u with $\delta_A^*(q_0, u) = q$. In the reversed machine, a path labeled u^R leads from a state r to q_0 exactly when A has a path labeled u from q_0 to r. Determinism makes that endpoint uniquely q.

$$u^R \text{ is accepted from subset } S \quad \Longleftrightarrow \quad q \in S.$$

Thus u^R distinguishes S and T. All output states are reachable, so the DFA is minimal.

Apply the lemma to the second determinization

The first determinization produces a reachable DFA D_1 for L^R. The lemma makes D_2 minimal, and reversing twice restores L. The argument also handles the one-state automata for the empty language.

Further perspective: [B14].

An Exponential Gap in Description Size

Fix $k \geq 1$. Let L_k contain binary words whose kth symbol from the right is 1 (words shorter than k are rejected).

An NFA with $k + 1$ states

At q_0, loop on $0, 1$ and also allow $q_0 \xrightarrow{1} q_1$. For $1 \leq i < k$, let either symbol move q_i to q_{i+1}. Only q_k accepts, and it has no outgoing transitions. Guess the relevant 1, then require exactly $k - 1$ more symbols.

An Exponential Gap in Description Size

Fix $k \geq 1$. Let L_k contain binary words whose kth symbol from the right is 1 (words shorter than k are rejected).

An NFA with $k + 1$ states

At q_0, loop on $0, 1$ and also allow $q_0 \xrightarrow{1} q_1$. For $1 \leq i < k$, let either symbol move q_i to q_{i+1}. Only q_k accepts, and it has no outgoing transitions. Guess the relevant 1, then require exactly $k - 1$ more symbols.

Every DFA needs at least 2^k states. Take two distinct length-k words x, y and a position j from the left where they differ. Append 0^{j-1} to both. Their original jth symbols are now kth from the right, so exactly one extended word belongs to L_k. All 2^k prefixes must therefore lead to distinct states.

An Exponential Gap in Description Size

Fix $k \geq 1$. Let L_k contain binary words whose kth symbol from the right is 1 (words shorter than k are rejected).

An NFA with $k + 1$ states

At q_0, loop on $0, 1$ and also allow $q_0 \xrightarrow{1} q_1$. For $1 \leq i < k$, let either symbol move q_i to q_{i+1}. Only q_k accepts, and it has no outgoing transitions. Guess the relevant 1, then require exactly $k - 1$ more symbols.

Every DFA needs at least 2^k states. Take two distinct length-k words x, y and a position j from the left where they differ. Append 0^{j-1} to both. Their original jth symbols are now kth from the right, so exactly one extended word belongs to L_k. All 2^k prefixes must therefore lead to distinct states.

A DFA storing the last k bits, initially padded with zeros, attains this bound. Hence nondeterminism can save exponentially many states without recognizing any additional languages.

Fooling Sets: A Lower Bound That Works for NFAs

A strong fooling-set criterion [GS96]

Suppose pairs (x_i, y_i), $1 \le i \le r$, satisfy

$$x_i y_i \in L \quad \text{for every } i, \qquad x_i y_j \notin L \quad \text{whenever } i \neq j.$$

Then every ε-NFA for L has at least r states.

Fooling Sets: A Lower Bound That Works for NFAs

A strong fooling-set criterion [GS96]

Suppose pairs (x_i, y_i), $1 \le i \le r$, satisfy

$$x_i y_i \in L \quad \text{for every } i, \qquad x_i y_j \notin L \quad \text{whenever } i \neq j.$$

Then every ε-NFA for L has at least r states.

Proof by splicing paths. Choose an accepting path for each $x_i y_i$, and let p_i be its state at the boundary after x_i and before y_i. If $p_i = p_j$, follow the first path's x_i portion and the second path's y_j portion. This accepts the forbidden word $x_i y_j$. Therefore all p_i are distinct. Epsilon moves do not affect the splice.

Fooling Sets: A Lower Bound That Works for NFAs

A strong fooling-set criterion [GS96]

Suppose pairs (x_i, y_i), $1 \le i \le r$, satisfy

$$x_i y_i \in L \quad \text{for every } i, \qquad x_i y_j \notin L \quad \text{whenever } i \neq j.$$

Then every ε-NFA for L has at least r states.

Proof by splicing paths. Choose an accepting path for each $x_i y_i$, and let p_i be its state at the boundary after x_i and before y_i. If $p_i = p_j$, follow the first path's x_i portion and the second path's y_j portion. This accepts the forbidden word $x_i y_j$. Therefore all p_i are distinct. Epsilon moves do not affect the splice.

Do not confuse DFA and NFA lower bounds

Distinguishable prefixes directly lower-bound DFA states. An NFA can reach many states after one prefix, so that argument alone does not lower-bound its state count. The path-splicing argument supplies the additional requirement here.

Solved Challenge: Can Nondeterminism Shorten a Chain?

For $L_m = \{a^m\}$, $m \geq 1$, use pairs

$$(x_i, y_i) = (a^i, a^{m-i}), \qquad 0 \leq i \leq m.$$

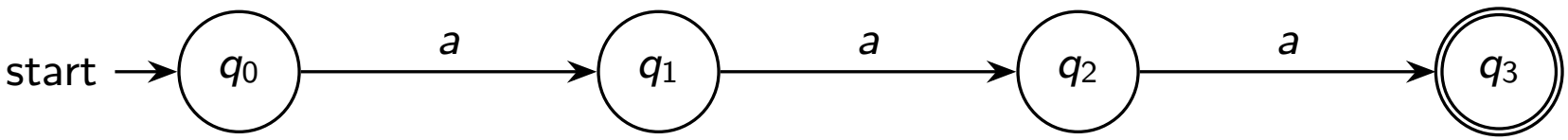
Each diagonal word is a^m. A cross word has length $m + i - j \neq m$, so every NFA needs at least $m + 1$ states by the fooling-set criterion.

Solved Challenge: Can Nondeterminism Shorten a Chain?

For $L_m = \{a^m\}$, $m \geq 1$, use pairs

$$(x_i, y_i) = (a^i, a^{m-i}), \qquad 0 \leq i \leq m.$$

Each diagonal word is a^m. A cross word has length $m + i - j \neq m$, so every NFA needs at least $m + 1$ states by the fooling-set criterion. A chain attains the bound. For $m = 3$:

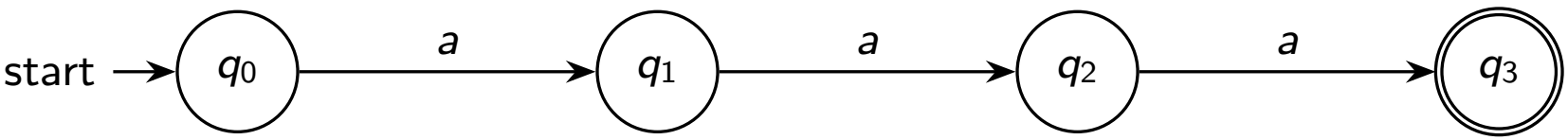


Solved Challenge: Can Nondeterminism Shorten a Chain?

For $L_m = \{a^m\}$, $m \geq 1$, use pairs

$$(x_i, y_i) = (a^i, a^{m-i}), \qquad 0 \leq i \leq m.$$

Each diagonal word is a^m. A cross word has length $m + i - j \neq m$, so every NFA needs at least $m + 1$ states by the fooling-set criterion. A chain attains the bound. For $m = 3$:



Why a complete DFA needs one more state

It must explicitly reject inputs longer than m, so add a sink. The $m + 1$ chain states have different accepted continuations $\{a^{m-i}\}$; the sink has none. All are reachable and pairwise distinguishable. Thus the minima are $m + 1$ for NFAs and $m + 2$ for complete DFAs, not an exponential difference in this example.

Algorithmic Costs: Representation Matters

Let $m \geq 1$ be input length, k the NFA state count, e its number of labeled edges, and n the supplied DFA state count. Fix a finite alphabet.

- **DFA simulation:** $O(m)$ time with constant-time transition lookup and one current-state identifier.
- **NFA set simulation:** $O(m(k + e))$ time by scanning outgoing edges and computing closure with visited states; $O(k)$ working space beyond the stored automaton. No explicit enumeration of paths is needed.

Algorithmic Costs: Representation Matters

Let $m \geq 1$ be input length, k the NFA state count, e its number of labeled edges, and n the supplied DFA state count. Fix a finite alphabet.

- **DFA simulation:** $O(m)$ time with constant-time transition lookup and one current-state identifier.
- **NFA set simulation:** $O(m(k+e))$ time by scanning outgoing edges and computing closure with visited states; $O(k)$ working space beyond the stored automaton. No explicit enumeration of paths is needed.
- **Determinization:** up to 2^k subset states. This is preprocessing; it may be too large even when direct simulation is easy.
- **Table filling:** a dependency-queue implementation achieves $O(n^2)$ time and space. Repeatedly rescanning the whole pair table can take longer; the classroom pseudocode alone does not ensure this bound.
- **Hopcroft minimization:** $O(n \log n)$ time for a fixed alphabet, using efficient partition refinement; see [H71].

Do not transfer the DFA theorem to NFAs

The pair-merging algorithm and uniqueness theorem here concern DFAs. Determinizing and minimizing gives a minimal DFA, not a minimal NFA.

References: Textbook Challenge Sources

[PSU13] Tim Sheard, Portland State University, CS 311, Fall 2013, Homework 2, problem 6, identifying the binary-addition exercise as Sipser 1.32.

[UCSD08] UC San Diego, CSE 105, Fall 2008, Homework 1, problem 5: binary comparison with a gap of two, described there as an extension of Sipser 1.34 (2nd ed.).

Attribution and conventions

The challenge statements above are adapted, and the worked solutions are written for this slide set. Exercise numbers refer to these verified handouts, not to an assumed numbering in every edition. The slides explicitly specify reading direction, leading zeros, and the value of an empty row.

The remaining practice problems are original textbook-style exercises aligned with Linz and Sipser, not claimed to be copied book exercises.

References: Optional Automata Results

[B14] Filippo Bonchi, Marcello M. Bonsangue, Helle H. Hansen, Prakash Panangaden, Jan J. M. M. Rutten, and Alexandra Silva, *Algebra-Coalgebra Duality in Brzozowski's Minimization Algorithm*, ACM Transactions on Computational Logic, 15(1), article 3, 2014.

[GS96] Ian Glaister and Jeffrey Shallit, *A Lower Bound Technique for the Size of Nondeterministic Finite Automata*, Information Processing Letters, 59(2), pp. 75–77, 1996.

These sources support optional enrichment beyond the basic textbook conversion and minimization procedures. The slides include proof sketches so that the results can be used without reading the full papers.

References and Suggested Teaching Route

[LR23] Peter Linz and Susan H. Rodger, *An Introduction to Formal Languages and Automata*, 7th ed., Jones & Bartlett Learning, 2023. Chapter 2: finite automata, equivalence, and state reduction.

[S13] Michael Sipser, *Introduction to the Theory of Computation*, 3rd ed., Cengage, 2013. Sections 1.1–1.2: finite automata and nondeterminism. See also his MIT 18.404J lecture notes (2020).

[H71] John Hopcroft, *An $n \log n$ Algorithm for Minimizing States in a Finite Automaton*, Stanford technical report STAN-CS-71-190, 1971.

Keep the worked state-design, subset, and marking examples in the main lecture. Use the appendix for proof enrichment, additional practice, and implementation tradeoffs. Adapted textbook challenges are identified explicitly in the source slide; the remaining exercises are original.