Properties of Regular Languages

Closure, Decision Algorithms, and the Pumping Lemma

Computer Science Theory

Slide Set 4

Table of Contents

Learning Outcomes

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

- prove closure of the regular languages using automaton constructions and set identities;
- decide membership, emptiness, finiteness, equality, containment, and universality for regular languages;
- state and explain the quantifier structure of the pumping lemma;
- use the pumping lemma to prove selected languages nonregular; and
- recognize invalid pumping-lemma arguments and repair them.

Part 1: Core Closure Properties

Self-Study Header: Operating within the Regular World

A class of languages is **closed** under an operation if applying that operation to members of the class always results in a member of the same class.

Part 1: Core Closure Properties

Self-Study Header: Operating within the Regular World

A class of languages is **closed** under an operation if applying that operation to members of the class always results in a member of the same class.

Closure Theorem

The family of regular languages is closed under union, intersection, concatenation, complement, and Kleene star.

Part 1: Core Closure Properties

Self-Study Header: Operating within the Regular World

A class of languages is **closed** under an operation if applying that operation to members of the class always results in a member of the same class.

Closure Theorem

The family of regular languages is closed under union, intersection, concatenation, complement, and Kleene star.

Formally, if L_1 and L_2 are regular languages, the following are also regular:

1. **Union:** $L_1 \cup L_2$
2. **Concatenation:** $L_1 L_2$
3. **Kleene star:** L_1^*
4. **Complement:** $\overline{L_1} = \Sigma^* \setminus L_1$
5. **Intersection:** $L_1 \cap L_2$

Proof by Construction: Complementation

Theorem: If L is regular, then $\overline{L}$ is regular.

Proof by Construction: Complementation

Theorem: If L is regular, then $\overline{L}$ is regular.

The Construction

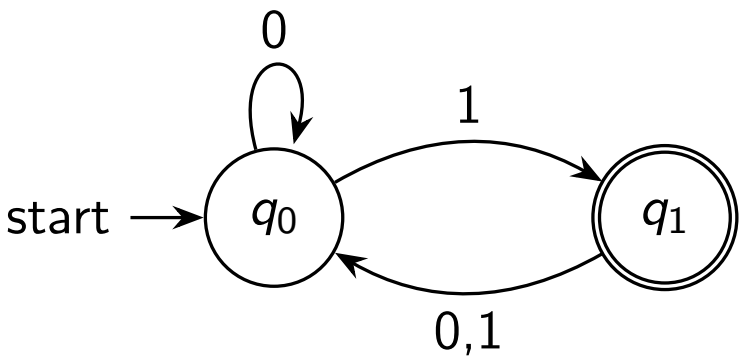
Because L is regular, a *complete* DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizes L. Construct $M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$ by swapping the accepting and nonaccepting states.

Proof by Construction: Complementation

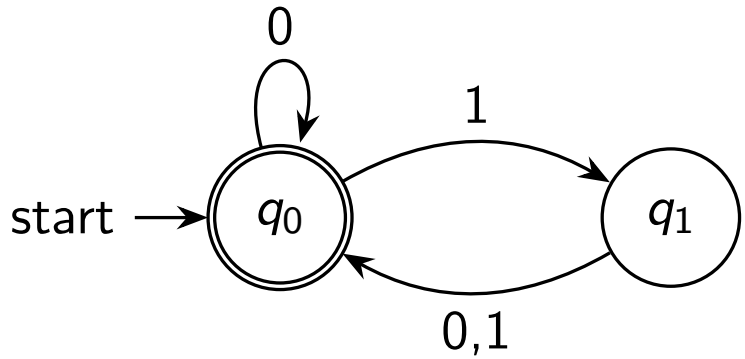
Theorem: If L is regular, then $\overline{L}$ is regular.

The Construction

Because L is regular, a *complete* DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizes L. Construct $M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$ by swapping the accepting and nonaccepting states.



DFA for L



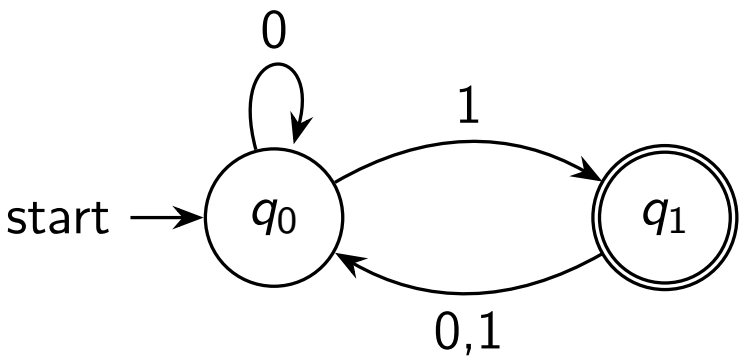
DFA for $\overline{L}$

Proof by Construction: Complementation

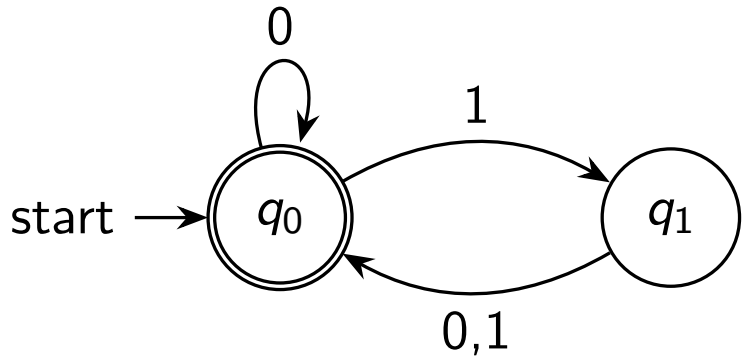
Theorem: If L is regular, then $\overline{L}$ is regular.

The Construction

Because L is regular, a *complete* DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizes L. Construct $M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$ by swapping the accepting and nonaccepting states.



DFA for L



DFA for $\overline{L}$

Crucial detail: Complete the DFA first by adding a sink state if necessary. Merely swapping final states in an NFA does not compute the complement, because acceptance is existential over its possible paths.

Proof by Construction: Intersection

Theorem: If L_1 and L_2 are regular, then $L_1 \cap L_2$ is regular.

Proof by Construction: Intersection

Theorem: If L_1 and L_2 are regular, then $L_1 \cap L_2$ is regular.

Cartesian Product Construction

Let $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$. We build a new DFA $M_\cap = (Q, \Sigma, \delta, q_0, F)$ that simulates both machines simultaneously:

- $Q = Q_1 \times Q_2$ (the states are pairs).
- $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$.
- $q_0 = (s_1, s_2)$.
- $F = F_1 \times F_2$ (accept iff **both** machines accept).

Proof by Construction: Intersection

Theorem: If L_1 and L_2 are regular, then $L_1 \cap L_2$ is regular.

Cartesian Product Construction

Let $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$. We build a new DFA $M_\cap = (Q, \Sigma, \delta, q_0, F)$ that simulates both machines simultaneously:

- $Q = Q_1 \times Q_2$ (the states are pairs).
- $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$.
- $q_0 = (s_1, s_2)$.
- $F = F_1 \times F_2$ (accept iff **both** machines accept).

Alternative Proof via De Morgan's Law

Alternatively, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$. Since regular languages are closed under union and complementation, they must be closed under intersection!

Advanced Operations

Regular languages are robust and closed under many other operations.

Advanced Operations

Regular languages are robust and closed under many other operations.

- **Reversal (L^R):** Reverse every transition, make the old start state the sole accept state, and add a new start state with ε-transitions to all old accept states.

Advanced Operations

Regular languages are robust and closed under many other operations.

- **Reversal (L^R):** Reverse every transition, make the old start state the sole accept state, and add a new start state with ε-transitions to all old accept states.
- **Difference ($L_1 \setminus L_2$):** Since $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$, and regular languages are closed under intersection and complement, they are closed under difference.

Advanced Operations

Regular languages are robust and closed under many other operations.

- **Reversal (L^R):** Reverse every transition, make the old start state the sole accept state, and add a new start state with ε-transitions to all old accept states.
- **Difference ($L_1 \setminus L_2$):** Since $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$, and regular languages are closed under intersection and complement, they are closed under difference.
- **Homomorphism ($h(L)$):** For each transition labeled a, replace the edge by a path spelling $h(a)$; use an ε-transition when $h(a) = \varepsilon$.

Advanced Operations

Regular languages are robust and closed under many other operations.

- **Reversal (L^R):** Reverse every transition, make the old start state the sole accept state, and add a new start state with ε-transitions to all old accept states.
- **Difference ($L_1 \setminus L_2$):** Since $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$, and regular languages are closed under intersection and complement, they are closed under difference.
- **Homomorphism ($h(L)$):** For each transition labeled a, replace the edge by a path spelling $h(a)$; use an ε-transition when $h(a) = \varepsilon$.
- **Right quotient:** $L_1/L_2 = \{x : \text{some } y \in L_2 \text{ satisfies } xy \in L_1\}$. If L_1 and L_2 are regular, then L_1/L_2 is regular.

Closure Proof Toolbox

Choose the representation that makes the operation easiest.

Automaton Constructions

Union/intersection: Use a product DFA; change only the accepting pairs ($\vee$ for union, $\wedge$ for intersection).

Complement: Complete a DFA and replace F by $Q \setminus F$.

Concatenation/star: Connect ε-NFAs using the same wrappers as in Thompson's construction.

Reversal: Reverse the transitions and exchange the roles of the initial and final boundaries.

Homomorphism: Replace each symbol edge by a path spelling its image.

Closure Proof Toolbox

Choose the representation that makes the operation easiest.

Automaton Constructions

Union/intersection: Use a product DFA; change only the accepting pairs ($\vee$ for union, $\wedge$ for intersection).

Complement: Complete a DFA and replace F by $Q \setminus F$.

Concatenation/star: Connect ε-NFAs using the same wrappers as in Thompson's construction.

Reversal: Reverse the transitions and exchange the roles of the initial and final boundaries.

Homomorphism: Replace each symbol edge by a path spelling its image.

Set Identities Save Work

Difference, symmetric difference, and De Morgan's laws reduce new closure questions to operations already known to preserve regularity.

Part 2: Elementary Questions

Self-Study Header: Algorithmic Decidability

Given a standard representation of a regular language L (DFA, NFA, regular expression, or regular grammar), can we algorithmically answer fundamental questions about it?

Part 2: Elementary Questions

Self-Study Header: Algorithmic Decidability

Given a standard representation of a regular language L (DFA, NFA, regular expression, or regular grammar), can we algorithmically answer fundamental questions about it?

1. The Membership Question

Given a string w and a standard representation of L, is $w \in L$?

Algorithm: For a DFA, trace the unique path labeled w from q_0 and test whether it ends in F.
Running time: $O(|w|)$ for a supplied DFA. Converting an NFA, RE, or grammar to a DFA is preprocessing and may increase the representation size substantially.

Part 2: Elementary Questions

Self-Study Header: Algorithmic Decidability

Given a standard representation of a regular language L (DFA, NFA, regular expression, or regular grammar), can we algorithmically answer fundamental questions about it?

1. The Membership Question

Given a string w and a standard representation of L, is $w \in L$?

Algorithm: For a DFA, trace the unique path labeled w from q_0 and test whether it ends in F.
Running time: $O(|w|)$ for a supplied DFA. Converting an NFA, RE, or grammar to a DFA is preprocessing and may increase the representation size substantially.

Because the standard representations are effectively convertible, a DFA algorithm establishes decidability for all of them, though not necessarily with the same complexity.

Emptiness and Finiteness

2. The Emptiness Question

Given a regular language L, is $L = \emptyset$?

Algorithm: Run BFS or DFS from q_0. The language is nonempty exactly when some state in F is reachable.

Emptiness and Finiteness

2. The Emptiness Question

Given a regular language L, is $L = \emptyset$?

Algorithm: Run BFS or DFS from q_0. The language is nonempty exactly when some state in F is reachable.

3. The Finiteness Question

Given a regular language L, is L finite or infinite?

Algorithm: Keep the *useful* states that are reachable from q_0 and can reach a final state. Then L is infinite exactly when the subgraph induced by these useful states contains a directed cycle.

The Equality Algorithm

Question: Given two regular languages L_1 and L_2, is $L_1 = L_2$?

The Equality Algorithm

Question: Given two regular languages L_1 and L_2, is $L_1 = L_2$?
We answer this using the **symmetric difference**:

$$L_3 = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2).$$

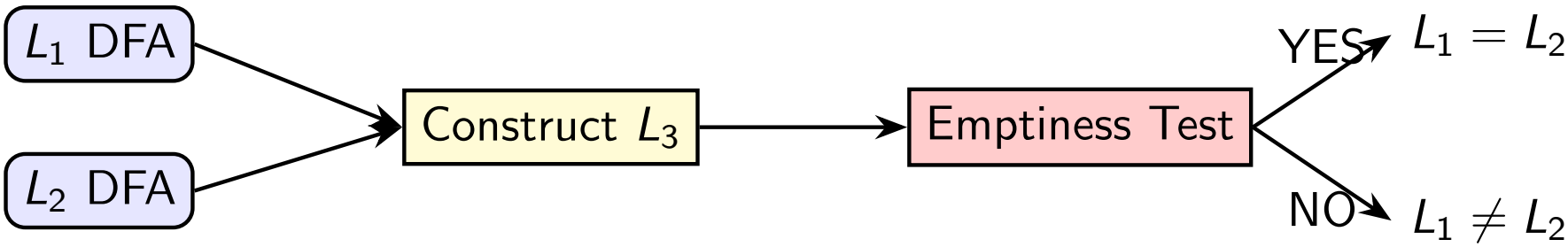
L_3 contains all strings that are in one language but not the other. $L_1 = L_2$ if and only if $L_3 = \emptyset$.

The Equality Algorithm

Question: Given two regular languages L_1 and L_2, is $L_1 = L_2$?
We answer this using the **symmetric difference**:

$$L_3 = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2).$$

L_3 contains all strings that are in one language but not the other. $L_1 = L_2$ if and only if $L_3 = \emptyset$.



More Decidable Questions

Once emptiness and the product construction are available, several other questions follow immediately.

Reductions to Emptiness

$$L = \Sigma^* \iff \overline{L} = \emptyset \qquad \text{(universality)},$$

$$L_1 \subseteq L_2 \iff L_1 \cap \overline{L_2} = \emptyset \qquad \text{(containment)},$$

$$L_1 \cap L_2 = \emptyset \qquad \text{(disjointness)}.$$

More Decidable Questions

Once emptiness and the product construction are available, several other questions follow immediately.

Reductions to Emptiness

$$L = \Sigma^* \iff \overline{L} = \emptyset \qquad \text{(universality)},$$

$$L_1 \subseteq L_2 \iff L_1 \cap \overline{L_2} = \emptyset \qquad \text{(containment)},$$

$$L_1 \cap L_2 = \emptyset \qquad \text{(disjointness)}.$$

Direct Product Test

For equality, build the product DFA and search for a reachable pair in which exactly one component is accepting. Such a path labels a concrete counterexample string distinguishing the two languages.

Part 3: Concept & The Pigeonhole Principle

Self-Study Header: The Limits of Finite Memory

A DFA has finite memory: its current state records everything relevant about the prefix read so far. What happens when a computation is longer than the number of available states?

Part 3: Concept & The Pigeonhole Principle

Self-Study Header: The Limits of Finite Memory

A DFA has finite memory: its current state records everything relevant about the prefix read so far. What happens when a computation is longer than the number of available states?

The Pigeonhole Principle

If p pigeons are placed into fewer than p holes, some hole has to have more than one pigeon in it.

Part 3: Concept & The Pigeonhole Principle

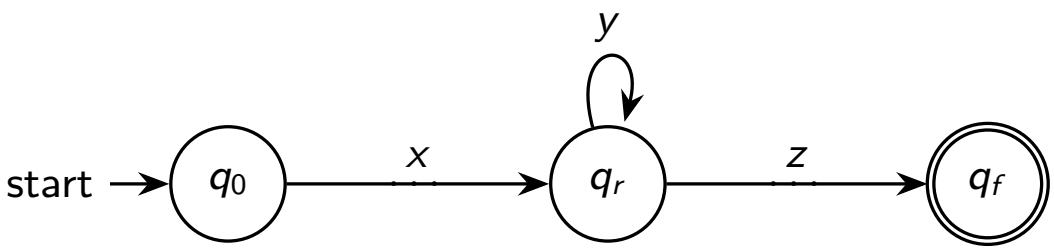
Self-Study Header: The Limits of Finite Memory

A DFA has finite memory: its current state records everything relevant about the prefix read so far. What happens when a computation is longer than the number of available states?

The Pigeonhole Principle

If p pigeons are placed into fewer than p holes, some hole has to have more than one pigeon in it.

If a DFA with n states reads a string of length at least n, consider the states visited before reading any symbol and after each of the first n symbols. Among these $n + 1$ visits, some state repeats. The intervening nonempty input labels a loop within the first n symbols.



Formal Statement: The Pumping Lemma

Pumping lemma for regular languages

If L is regular, then there exists $p \geq 1$ such that every $s \in L$ with $|s| \geq p$ can be written as $s = xyz$ satisfying:

1. $xy^i z \in L$ for every $i \geq 0$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Formal Statement: The Pumping Lemma

Pumping lemma for regular languages

If L is regular, then there exists $p \geq 1$ such that every $s \in L$ with $|s| \geq p$ can be written as $s = xyz$ satisfying:

1. $xy^i z \in L$ for every $i \geq 0$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Breaking it down:

- Condition 1: The same decomposition works for *every* pumping value $i \geq 0$.
- Condition 2: The loop consumes at least one symbol, so pumping actually changes the string when $i \neq 1$.
- Condition 3: The loop lies within the first p symbols.

Why the Pumping Lemma Is True

Let a DFA for L have p states, and let $s \in L$ with $|s| \geq p$. During the first p input symbols, record the $p+1$ visited states:

$$q_0, q_1, \ldots, q_p.$$

Two visits must be equal; choose $0 \leq r < t \leq p$ with $q_r = q_t$. Split s as follows:

$$\underbrace{s_1 \cdots s_r}_{x} \underbrace{s_{r+1} \cdots s_t}_{y} \underbrace{s_{t+1} \cdots s_{|s|}}_{z}.$$

Then $|y| = t - r > 0$ and $|xy| = t \leq p$. Because reading y returns the DFA to the same state, the loop may be traversed any number of times; hence $xy^i z \in L$ for every $i \geq 0$.

Quantifiers and the Lemma's Limitation

The logical order is the heart of a correct nonregularity proof:

$$\exists p\ \forall s\ \exists (x, y, z)\ \forall i \geq 0.$$

To contradict the lemma, reverse the game after assuming regularity:

$$\forall p\ \exists s\ \forall (x, y, z)\ \exists i \geq 0 \quad \text{such that } xy^i z \notin L,$$

subject to $s = xyz$, $|s| \geq p$, $|y| > 0$, and $|xy| \leq p$.

Quantifiers and the Lemma's Limitation

The logical order is the heart of a correct nonregularity proof:

$$\exists p\ \forall s\ \exists (x, y, z)\ \forall i \geq 0.$$

To contradict the lemma, reverse the game after assuming regularity:

$$\forall p\ \exists s\ \forall (x, y, z)\ \exists i \geq 0 \quad \text{such that } xy^i z \notin L,$$

subject to $s = xyz$, $|s| \geq p$, $|y| > 0$, and $|xy| \leq p$.

Necessary, Not Sufficient

Every regular language satisfies the pumping lemma, but satisfying the lemma does not by itself prove that a language is regular. Finite languages satisfy it vacuously by choosing p larger than every string length.

The Adversarial Game

The pumping lemma is used **negatively** to prove that a language is not regular. Its quantifiers can be remembered as a four-step game.

The Adversarial Game

The pumping lemma is used **negatively** to prove that a language is not regular. Its quantifiers can be remembered as a four-step game.

How to play the Pumping Game

1. **Opponent:** Gives you the pumping length p promised by the assumption that the language is regular.

The Adversarial Game

The pumping lemma is used **negatively** to prove that a language is not regular. Its quantifiers can be remembered as a four-step game.

How to play the Pumping Game

1. **Opponent:** Gives you the pumping length p promised by the assumption that the language is regular.
2. **You:** Pick a strategic string $s \in L$ such that $|s| \geq p$.

The Adversarial Game

The pumping lemma is used **negatively** to prove that a language is not regular. Its quantifiers can be remembered as a four-step game.

How to play the Pumping Game

1. **Opponent:** Gives you the pumping length p promised by the assumption that the language is regular.
2. **You:** Pick a strategic string $s \in L$ such that $|s| \geq p$.
3. **Opponent:** Chooses any valid split $s = xyz$ with $|y| > 0$ and $|xy| \leq p$.

The Adversarial Game

The pumping lemma is used **negatively** to prove that a language is not regular. Its quantifiers can be remembered as a four-step game.

How to play the Pumping Game

1. **Opponent:** Gives you the pumping length p promised by the assumption that the language is regular.
2. **You:** Pick a strategic string $s \in L$ such that $|s| \geq p$.
3. **Opponent:** Chooses any valid split $s = xyz$ with $|y| > 0$ and $|xy| \leq p$.
4. **You:** After seeing the split, choose $i \geq 0$ so that $xy^i z \notin L$.

The Adversarial Game

The pumping lemma is used **negatively** to prove that a language is not regular. Its quantifiers can be remembered as a four-step game.

How to play the Pumping Game

1. **Opponent:** Gives you the pumping length p promised by the assumption that the language is regular.
2. **You:** Pick a strategic string $s \in L$ such that $|s| \geq p$.
3. **Opponent:** Chooses any valid split $s = xyz$ with $|y| > 0$ and $|xy| \leq p$.
4. **You:** After seeing the split, choose $i \geq 0$ so that $xy^i z \notin L$.

If this strategy works for every valid split, the initial regularity assumption is false.

Example 1: Equal Blocks

Goal: Prove $L = \{a^n b^n : n \geq 0\}$ is not regular.

Example 1: Equal Blocks

Goal: Prove $L = \{a^n b^n : n \geq 0\}$ is not regular.

1. Assume L is regular. The opponent provides pumping length p.

Example 1: Equal Blocks

Goal: Prove $L = \{a^n b^n : n \geq 0\}$ is not regular.

1. Assume L is regular. The opponent provides pumping length p.
2. **We choose:** $s = a^p b^p$. Note that $s \in L$ and $|s| = 2p \geq p$.

Example 1: Equal Blocks

Goal: Prove $L = \{a^n b^n : n \geq 0\}$ is not regular.

1. Assume L is regular. The opponent provides pumping length p.
2. **We choose:** $s = a^p b^p$. Note that $s \in L$ and $|s| = 2p \geq p$.
3. Opponent divides $s = xyz$.

   Crucial insight: Since $|xy| \leq p$, both x and y lie in the first block of a's. Thus $y = a^k$ for some $1 \leq k \leq p$.

Example 1: Equal Blocks

Goal: Prove $L = \{a^n b^n : n \geq 0\}$ is not regular.

1. Assume L is regular. The opponent provides pumping length p.
2. **We choose:** $s = a^p b^p$. Note that $s \in L$ and $|s| = 2p \geq p$.
3. Opponent divides $s = xyz$.

   Crucial insight: Since $|xy| \leq p$, both x and y lie in the first block of a's. Thus $y = a^k$ for some $1 \leq k \leq p$.
4. **We pump:** Choose $i = 2$. The new string is xy^2z.
   - $s = xy^1z$ had p a's and p b's.
   - Pumping adds k extra a's.
   - The new string $xy^2z = a^{p+k}b^p$.

Example 1: Equal Blocks

Goal: Prove $L = \{a^n b^n : n \geq 0\}$ is not regular.

1. Assume L is regular. The opponent provides pumping length p.
2. **We choose:** $s = a^p b^p$. Note that $s \in L$ and $|s| = 2p \geq p$.
3. Opponent divides $s = xyz$.

   Crucial insight: Since $|xy| \leq p$, both x and y lie in the first block of a's. Thus $y = a^k$ for some $1 \leq k \leq p$.
4. **We pump:** Choose $i = 2$. The new string is xy^2z.
   - $s = xy^1z$ had p a's and p b's.
   - Pumping adds k extra a's.
   - The new string $xy^2z = a^{p+k}b^p$.

Since $k > 0$, the numbers of a's and b's differ. Thus $xy^2z \notin L$, contradicting the pumping lemma. Therefore L is not regular.

Example 2: Duplicated Strings

Goal: Prove $F = \{ww : w \in \{0,1\}^*\}$ is not regular.

Example 2: Duplicated Strings

Goal: Prove $F = \{ww : w \in \{0,1\}^*\}$ is not regular.

1. Assume F is regular. Opponent provides pumping length p.

Example 2: Duplicated Strings

Goal: Prove $F = \{ww : w \in \{0,1\}^*\}$ is not regular.

1. Assume F is regular. Opponent provides pumping length p.
2. **We choose:** $s = 0^p10^p1 = (0^p1)(0^p1)$.

Example 2: Duplicated Strings

Goal: Prove $F = \{ww : w \in \{0,1\}^*\}$ is not regular.

1. Assume F is regular. Opponent provides pumping length p.
2. **We choose:** $s = 0^p 1 0^p 1 = (0^p 1)(0^p 1)$.
3. Because $|xy| \leq p$, a valid split has $y = 0^k$ for some $1 \leq k \leq p$.

Example 2: Duplicated Strings

Goal: Prove $F = \{ww : w \in \{0,1\}^*\}$ is not regular.

1. Assume F is regular. Opponent provides pumping length p.
2. **We choose:** $s = 0^p 1 0^p 1 = (0^p 1)(0^p 1)$.
3. Because $|xy| \leq p$, a valid split has $y = 0^k$ for some $1 \leq k \leq p$.
4. **We pump:** Choose $i = 2$. The new string is xy^2z.
   - The first block of 0's now has $p + k$ zeros.
   - The new string is $0^{p+k} 1 0^p 1$.

Example 2: Duplicated Strings

Goal: Prove $F = \{ww : w \in \{0,1\}^*\}$ is not regular.

1. Assume F is regular. Opponent provides pumping length p.
2. **We choose:** $s = 0^p 1 0^p 1 = (0^p 1)(0^p 1)$.
3. Because $|xy| \leq p$, a valid split has $y = 0^k$ for some $1 \leq k \leq p$.
4. **We pump:** Choose $i = 2$. The new string is xy^2z.
   - The first block of 0's now has $p + k$ zeros.
   - The new string is $0^{p+k} 1 0^p 1$.

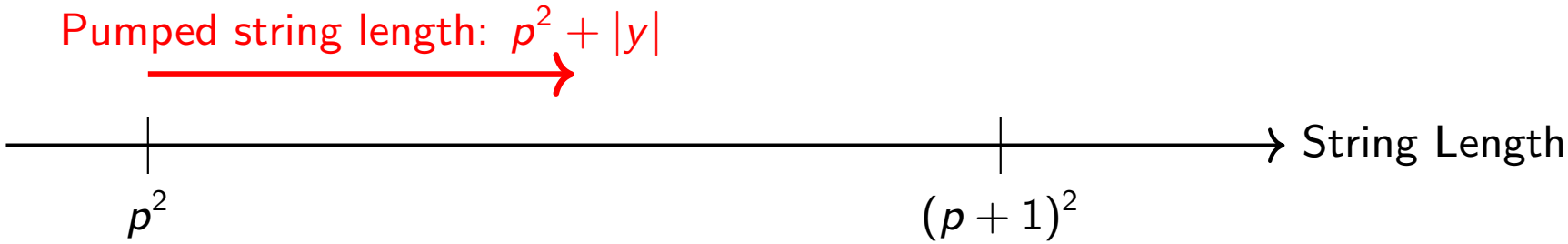
The pumped string has exactly two 1's, separated by $p + 1$ positions. If it were uu, the distance between these two corresponding 1's would equal half the total length, $p + 1 + k/2$, which is impossible for $k > 0$ (and if k is odd, the total length is odd). Hence $xy^2z \notin F$, a contradiction.

Example 3: Unary Perfect Squares

Goal: Prove $L = \{a^{n^2} : n \geq 0\}$ is not regular.

Example 3: Unary Perfect Squares

Goal: Prove $L = \{a^{n^2} : n \geq 0\}$ is not regular.

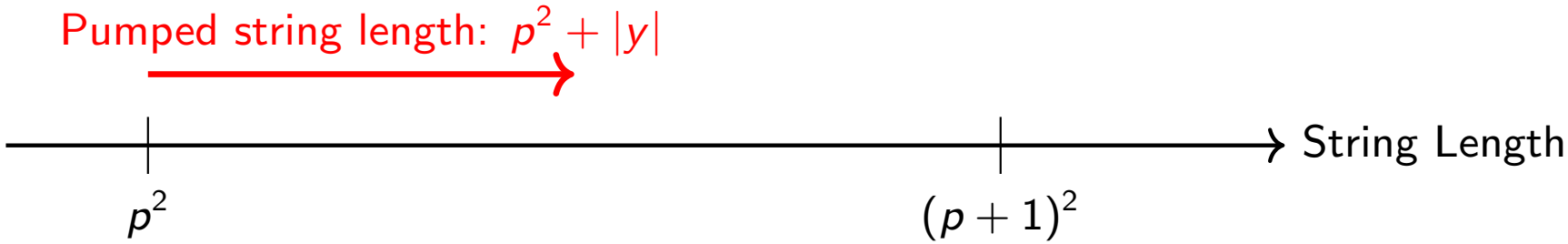


Since $|y| \leq p$, the pumped length is at most $p^2 + p$

The next square is $(p+1)^2 = p^2 + 2p + 1$

Example 3: Unary Perfect Squares

Goal: Prove $L = \{a^{n^2} : n \geq 0\}$ is not regular.



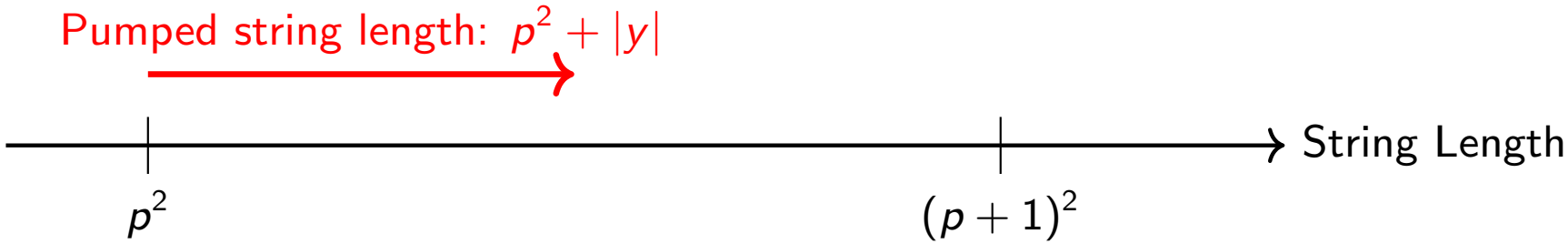
Since $|y| \leq p$, the pumped length is at most $p^2 + p$

The next square is $(p+1)^2 = p^2 + 2p + 1$

1. Pick $s = a^{p^2}$. Opponent divides $s = xyz$.
2. We pump $i = 2$. The length of xy^2z is $p^2 + |y|$.
3. Since $|y| \leq p$, then $p^2 < p^2 + |y| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$.

Example 3: Unary Perfect Squares

Goal: Prove $L = \{a^{n^2} : n \geq 0\}$ is not regular.



Since $|y| \leq p$, the pumped length is at most $p^2 + p$

The next square is $(p+1)^2 = p^2 + 2p + 1$

1. Pick $s = a^{p^2}$. Opponent divides $s = xyz$.
2. We pump $i = 2$. The length of xy^2z is $p^2 + |y|$.
3. Since $|y| \leq p$, then $p^2 < p^2 + |y| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$.

The pumped length lies strictly between consecutive squares, so $xy^2z \notin L$. This contradicts the pumping lemma.

Using Closure to Prove Nonregularity

Closure theorems can be used contrapositively.

Reduction Pattern

To prove a language K nonregular:

1. assume K is regular;
2. apply an operation that preserves regularity; and
3. obtain a language already known to be nonregular.

Using Closure to Prove Nonregularity

Closure theorems can be used contrapositively.

Reduction Pattern

To prove a language K nonregular:

1. assume K is regular;
2. apply an operation that preserves regularity; and
3. obtain a language already known to be nonregular.

Example

Let

$$K = \{w \in \{a, b\}^* : |w|_a = |w|_b\}.$$

If K were regular, then its intersection with the regular language a^*b^* would be regular. But

$$K \cap a^*b^* = \{a^n b^n : n \geq 0\},$$

contradicting Example 1. Hence K is not regular.

Checkpoint Exercises

1. **Closure Property Trap:** Given that L_1 is regular, and the union $L_1 \cup L_2$ is regular, does it necessarily follow that L_2 is regular?

Checkpoint Exercises

1. **Closure Property Trap:** Given that L_1 is regular, and the union $L_1 \cup L_2$ is regular, does it necessarily follow that L_2 is regular?

2. **A Surprising Result:** Let D contain the strings over $\{0, 1\}$ having equally many occurrences of 01 and 10 as consecutive substrings. Is D regular?

Checkpoint Exercises

1. **Closure Property Trap:** Given that L_1 is regular, and the union $L_1 \cup L_2$ is regular, does it necessarily follow that L_2 is regular?

2. **A Surprising Result:** Let D contain the strings over $\{0, 1\}$ having equally many occurrences of 01 and 10 as consecutive substrings. Is D regular?

3. **Choosing s:** If trying to prove $L = \{a^n b^n \mid n \geq 0\}$ is not regular, what is wrong with picking $s = a^2 b^2$ if the opponent's $p = 5$?

Teacher's Corner: Common Pitfalls

Pitfall 1: Picking a specific p

Students sometimes begin by fixing $p = 5$. This is invalid: the pumping length is an unknown constant. The strategy must work for every $p \geq 1$.

Teacher's Corner: Common Pitfalls

Pitfall 1: Picking a specific p

Students sometimes begin by fixing $p = 5$. This is invalid: the pumping length is an unknown constant. The strategy must work for every $p \geq 1$.

Pitfall 2: Making assumptions about y

Even for $s = a^p b^p$, you may not assume $y = a$. The opponent may choose $y = a^k$ for any $1 \leq k \leq p$. The proof succeeds because pumping *every* such nonempty block changes only the number of a's.

Also verify $s \in L$: for example, $(ab)^p \notin \{a^n b^n : n \geq 0\}$ when $p > 1$, so it cannot be used as the witness string for Example 1.

Hints for Checkpoint Exercises

Hints and Solutions

- **Exercise 1 (Closure Trap):** False. Let $L_1 = \Sigma^*$. Then $L_1 \cup L_2 = \Sigma^*$ for any L_2, including a nonregular language.
- **Exercise 2 (01 and 10 substrings):** Yes. Each change $0 \to 1$ must eventually be balanced by a change $1 \to 0$, except for a possible net change determined by the endpoints. Thus the counts are equal exactly for ε or when the first and last symbols agree; a DFA need only remember the first and current symbols.
- **Exercise 3 (Choosing s):** The witness must satisfy $|s| \geq p$. If $p = 5$, then $|a^2 b^2| = 4$, so the pumping lemma makes no promise about that string.

Hints for Checkpoint Exercises

Hints and Solutions

- **Exercise 1 (Closure Trap):** False. Let $L_1 = \Sigma^*$. Then $L_1 \cup L_2 = \Sigma^*$ for any L_2, including a nonregular language.
- **Exercise 2 (01 and 10 substrings):** Yes. Each change $0 \to 1$ must eventually be balanced by a change $1 \to 0$, except for a possible net change determined by the endpoints. Thus the counts are equal exactly for ε or when the first and last symbols agree; a DFA need only remember the first and current symbols.
- **Exercise 3 (Choosing s):** The witness must satisfy $|s| \geq p$. If $p = 5$, then $|a^2 b^2| = 4$, so the pumping lemma makes no promise about that string.

Pro-Tip: Pumping Down
Sometimes pumping up ($i = 2$) does not break the defining property. The lemma also requires $xy^0z = xz \in L$, so deleting y is often the cleaner route to a contradiction.