

## Strings and Languages

### Lecture 1

January 21, 2025

# Part I

## Strings

# String Definitions

## Definition

An **alphabet** is a **finite** set of symbols.

*Examples:*  $\Sigma = \{0, 1\}$ ,  $\Sigma = \{a, b, \dots, z\}$ ,  
 $\Sigma = \{\langle \text{moveforward} \rangle, \langle \text{moveback} \rangle\}$

# String Definitions

## Definition

An **alphabet** is a **finite** set of symbols.

*Examples:*  $\Sigma = \{0, 1\}$ ,  $\Sigma = \{a, b, \dots, z\}$ ,  
 $\Sigma = \{\langle \text{moveforward} \rangle, \langle \text{moveback} \rangle\}$

## Definition

A **string/word** over  $\Sigma$  is a **finite sequence** of symbols over  $\Sigma$ .

*Examples:* '0101001', 'string', ' $\langle \text{moveback} \rangle \langle \text{rotate90} \rangle$ '

# String Definitions

## Definition

An **alphabet** is a **finite** set of symbols.

*Examples:*  $\Sigma = \{0, 1\}$ ,  $\Sigma = \{a, b, \dots, z\}$ ,  
 $\Sigma = \{\langle \text{moveforward} \rangle, \langle \text{moveback} \rangle\}$

## Definition

A **string/word** over  $\Sigma$  is a **finite sequence** of symbols over  $\Sigma$ .

*Examples:* '0101001', 'string', ' $\langle \text{moveback} \rangle \langle \text{rotate90} \rangle$ '

- 1  $\epsilon$  is the **empty string**.
- 2 The **length** of a string  $w$  (denoted by  $|w|$ ) is the number of symbols in  $w$ . For example,  $|101| = 3$ ,  $|\epsilon| = 0$

# String Definition Contd

## Definition

A **string/word** over  $\Sigma$  is a **finite sequence** of symbols over  $\Sigma$ .

*Examples:* '0101001', 'string', '**<moveback><rotate90>**'

- 1  $\epsilon$  is the **empty string**.
- 2 The **length** of a string  $w$  (denoted by  $|w|$ ) is the number of symbols in  $w$ . For example,  $|101| = 3$ ,  $|\epsilon| = 0$

## Definition

For integer  $n \geq 0$ ,  $\Sigma^n$  is set of all strings over  $\Sigma$  of length  $n$ .

*Example:*  $\{0, 1\}^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$ .

## Definition

$\Sigma^*$  is the set of all strings over  $\Sigma$ .

# Formally

Formally strings are defined recursively/inductively:

- $\epsilon$  is a string of length 0
- $ax$  is a string if  $a \in \Sigma$  and  $x$  is a string.  
The length of  $ax$  is  $1 + |x|$

The above definition helps prove statements rigorously via induction.

- Alternative recursive definition useful in some proofs:  $xa$  is a string if  $a \in \Sigma$  and  $x$  is a string. The length of  $xa$  is  $1 + |x|$

## Convention

- $a, b, c, \dots$  denote elements of  $\Sigma$
- $w, x, y, z, \dots$  denote strings
- $A, B, C, \dots$  denote sets of strings

# Much ado about nothing

- $\epsilon$  is a **string** containing no symbols. It is not a set
- $\{\epsilon\}$  is a **set** containing one string: the empty string. It is a set, not a string.
- $\emptyset$  is the **empty set**. It contains no strings.
- $\{\emptyset\}$  is a **set** containing one element, which itself is a set that contains no elements.



# Concatenation and properties

- If  $x$  and  $y$  are strings then  $xy$  denotes their concatenation. Formally we define concatenation recursively based on definition of strings:
  - $xy = y$  if  $x = \epsilon$
  - $xy = a(wy)$  if  $x = aw$

Sometimes  $xy$  is written as  $x \bullet y$  to explicitly note that  $\bullet$  is a binary operator that takes two strings and produces another string.

- concatenation is associative:  $(uv)w = u(vw)$  and hence we write  $uvw$
- **not** commutative:  $uv$  not necessarily equal to  $vu$
- identity element:  $\epsilon u = u\epsilon = u$

# Substrings, prefix, suffix, exponents

## Definition

- 1  $v$  is **substring** of  $w$  iff there exist strings  $x, y$  such that  $w = xvy$ .
  - If  $x = \epsilon$  then  $v$  is a **prefix** of  $w$
  - If  $y = \epsilon$  then  $v$  is a **suffix** of  $w$
- 2 If  $w$  is a string then  $w^n$  is defined inductively as follows:  
 $w^n = \epsilon$  if  $n = 0$   
 $w^n = ww^{n-1}$  if  $n > 0$

Example:  $(\textit{blah})^4 = \textit{blahblahblahblah}$ .

# Set Concatenation

## Definition

Given two sets  $A$  and  $B$  of strings (over some common alphabet  $\Sigma$ ) the concatenation of  $A$  and  $B$  is defined as:

$$AB = \{xy \mid x \in A, y \in B\}$$

Example:  $A = \{fido, rover, spot\}$ ,  $B = \{fluffy, tabby\}$  then  
 $AB = \{fidofluffy, fidotabby, roverfluffy, rovertabby, spotfluffy, spottabby\}$ .

## Definition

- 1  $\Sigma^n$  is the set of all strings of length  $n$ . Defined inductively as follows:  
$$\Sigma^n = \{\epsilon\} \text{ if } n = 0$$
$$\Sigma^n = \Sigma\Sigma^{n-1} \text{ if } n > 0$$
- 2  $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$  is the set of all finite length strings
- 3  $\Sigma^+ = \bigcup_{n \geq 1} \Sigma^n$  is the set of non-empty strings.

# Exercise

Answer the following questions taking  $\Sigma = \{0, 1\}$ .

- 1 What is  $\Sigma^0$ ?
- 2 How many elements are there in  $\Sigma^3$ ?
- 3 How many elements are there in  $\Sigma^n$ ?
- 4 Does  $\Sigma^*$  have strings of infinite length?
- 5 If  $|u| = 2$  and  $|v| = 3$  then what is  $|u \bullet v|$ ?
- 6 Let  $u$  be an arbitrary string  $\Sigma^*$ . What is  $\epsilon u$ ? What is  $u \epsilon$ ?
- 7 Is  $uv = vu$  for every  $u, v \in \Sigma^*$ ?
- 8 Is  $(uv)w = u(vw)$  for every  $u, v, w \in \Sigma^*$ ?

# Canonical order and countability of strings

## Definition

An set  $A$  is **countably infinite** if there is a bijection  $f$  between the natural numbers and  $A$ .

Alternatively:  $A$  is countably infinite if  $A$  is an infinite set and there is an enumeration of elements of  $A$

# Canonical order and countability of strings

## Definition

An set  $A$  is **countably infinite** if there is a bijection  $f$  between the natural numbers and  $A$ .

Alternatively:  $A$  is countably infinite if  $A$  is an infinite set and there is an enumeration of elements of  $A$

## Theorem

$\Sigma^*$  is countably infinite for every finite  $\Sigma$ .

Enumerate strings in order of increasing length and for each given length enumerate strings in dictionary order (based on some fixed ordering of  $\Sigma$ ).

Example:  $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \dots\}$ .

$\{a, b, c\}^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, \dots\}$

# Exercise

**Question:** Is  $\Sigma^* \times \Sigma^* = \{(x, y) \mid x, y \in \Sigma^*\}$  countably infinite?



# Exercise

**Question:** Is  $\Sigma^* \times \Sigma^* = \{(x, y) \mid x, y \in \Sigma^*\}$  countably infinite?

**Question:** Is  $\Sigma^* \times \Sigma^* \times \Sigma^* = \{(x, y, z) \mid x, y, z \in \Sigma^*\}$  countably infinite?

# Part II

## Languages

# Languages

## Definition

A **language**  $L$  is a set of strings over  $\Sigma$ . In other words  $L \subseteq \Sigma^*$ .

## Examples:

- $\Sigma = \{0, 1\}$ .  $L = \{\epsilon, 0, 01, 111\}$  a finite language.
- $\Sigma$  is the set of English letters.  $L$  is the set of all words in a standard dictionary. Also a finite language
- $\Sigma = \{0, 1\}$ .  $L = \{0^n 1^n \mid n \geq 0\} = \{\epsilon, 01, 0011, 000111, \dots\}$ .
- $\Sigma$  is the set of symbols on a standard keyboard.  $L$  is the set of all valid programs in Python.

# Languages

## Definition

A **language**  $L$  is a set of strings over  $\Sigma$ . In other words  $L \subseteq \Sigma^*$ .

Standard set operations apply to languages.

- For languages  $A, B$ , their **union** is  $A \cup B$ , **intersection** is  $A \cap B$ , and **difference** is  $A \setminus B$  (also written as  $A - B$ ).
- For language  $A \subseteq \Sigma^*$  the **complement** of  $A$  is  $\bar{A} = \Sigma^* \setminus A$ .
- For languages  $A, B$  the **concatenation** of  $A, B$  is  $AB = \{xy \mid x \in A, y \in B\}$ .

# Exponentiation, Kleene star etc

## Definition

For a language  $L \subseteq \Sigma^*$  and  $n \in \mathbb{N}$ , define  $L^n$  inductively as follows.

$$L^n = \begin{cases} \{\epsilon\} & \text{if } n = 0 \\ L \bullet (L^{n-1}) & \text{if } n > 0 \end{cases}$$

And define  $L^* = \bigcup_{n \geq 0} L^n$ , and  $L^+ = \bigcup_{n \geq 1} L^n$

# Exercise

## Problem

Answer the following questions taking  $A, B \subseteq \{0, 1\}^*$ .

- 1 Is  $\epsilon = \{\epsilon\}$ ? Is  $\emptyset = \{\epsilon\}$ ?
- 2 What is  $\emptyset \bullet A$ ? What is  $A \bullet \emptyset$ ?
- 3 What is  $\{\epsilon\} \bullet A$ ? And  $A \bullet \{\epsilon\}$ ?
- 4 If  $|A| = 2$  and  $|B| = 3$ , what is  $|A \bullet B|$ ?

# Exercise

## Problem

Consider languages over  $\Sigma = \{0, 1\}$ .

- 1 What is  $\emptyset^0$ ?
- 2 If  $|L| = 2$ , then what is  $|L^4|$ ?
- 3 What is  $\emptyset^*$ ,  $\{\epsilon\}^*$ ,  $\epsilon^*$ ?
- 4 For what  $L$  is  $L^*$  finite?
- 5 What is  $\emptyset^+$ ,  $\{\epsilon\}^+$ ,  $\epsilon^+$ ?

# Languages and Computation

What are we interested in computing? Mostly functions.

**Informal definition:** An algorithm  $\mathcal{A}$  computes a function  $f : \Sigma^* \rightarrow \Sigma^*$  if for all  $w \in \Sigma^*$  the algorithm  $\mathcal{A}$  on input  $w$  terminates in a finite number of steps and outputs  $f(w)$ .

Examples of functions:

- Numerical functions: length, addition, multiplication, division etc
- Given graph  $G$  and  $s, t$  find shortest paths from  $s$  to  $t$
- Given program  $M$  check if  $M$  halts on empty input
- Posts Correspondence problem



# Languages and Computation

## Definition

A function  $f$  over  $\Sigma^*$  is a boolean if  $f : \Sigma^* \rightarrow \{0, 1\}$ .

# Languages and Computation

## Definition

A function  $f$  over  $\Sigma^*$  is a boolean if  $f : \Sigma^* \rightarrow \{0, 1\}$ .

**Observation:** There is a bijection between boolean functions and languages.

- Given boolean function  $f : \Sigma^* \rightarrow \{0, 1\}$  define language  $L_f = \{w \in \Sigma^* \mid f(w) = 1\}$

# Languages and Computation

## Definition

A function  $f$  over  $\Sigma^*$  is a boolean if  $f : \Sigma^* \rightarrow \{0, 1\}$ .

**Observation:** There is a bijection between boolean functions and languages.

- Given boolean function  $f : \Sigma^* \rightarrow \{0, 1\}$  define language  $L_f = \{w \in \Sigma^* \mid f(w) = 1\}$
- Given language  $L \subseteq \Sigma^*$  define boolean function  $f : \Sigma^* \rightarrow \{0, 1\}$  as follows:  $f(w) = 1$  if  $w \in L$  and  $f(w) = 0$  otherwise.

# Language recognition problem

## Definition

For a language  $L \subseteq \Sigma^*$  the language recognition problem associated with  $L$  is the following: given  $w \in \Sigma^*$ , is  $w \in L$ ?

# Language recognition problem

## Definition

For a language  $L \subseteq \Sigma^*$  the language recognition problem associated with  $L$  is the following: given  $w \in \Sigma^*$ , is  $w \in L$ ?

- Equivalent to the problem of “computing” the function  $f_L$ .
- Language recognition is same as boolean function computation
- How difficult is a function  $f$  to compute? How difficult is the recognizing  $L_f$ ?

# Language recognition problem

## Definition

For a language  $L \subseteq \Sigma^*$  the language recognition problem associated with  $L$  is the following: given  $w \in \Sigma^*$ , is  $w \in L$ ?

- Equivalent to the problem of “computing” the function  $f_L$ .
- Language recognition is same as boolean function computation
- How difficult is a function  $f$  to compute? How difficult is the recognizing  $L_f$ ?

Why two different views? Helpful in understanding different aspects.

# How many languages are there?

Recall:

## Definition

A set  $A$  is **countably infinite** if there is a bijection  $f$  between the natural numbers and  $A$ .

## Theorem

$\Sigma^*$  is countably infinite for every finite  $\Sigma$ .

The set of all languages is  $\mathbb{P}(\Sigma^*)$  the power set of  $\Sigma^*$

# How many languages are there?

Recall:

## Definition

A set  $A$  is **countably infinite** if there is a bijection  $f$  between the natural numbers and  $A$ .

## Theorem

$\Sigma^*$  is countably infinite for every finite  $\Sigma$ .

The set of all languages is  $\mathbb{P}(\Sigma^*)$  the power set of  $\Sigma^*$

## Theorem (Cantor)

$\mathbb{P}(\Sigma^*)$  is **not** countably infinite for any finite  $\Sigma$ .



# Cantor's diagonalization argument

## Theorem (Cantor)

$\mathbb{P}(\mathbb{N})$  is not countably infinite.

- Suppose  $\mathbb{P}(\mathbb{N})$  is countable infinite. Let  $S_1, S_2, \dots$ , be an enumeration of all subsets of numbers.
- Let  $D$  be the following diagonal subset of numbers.

$$D = \{i \mid i \notin S_i\}$$

- Since  $D$  is a set of numbers, by assumption,  $D = S_j$  for some  $j$ .
- **Question:** Is  $j \in D$ ?

# Cantor's diagonalization argument

## Theorem (Cantor)

$\mathbb{P}(\mathbb{N})$  is not countably infinite.

- Suppose  $\mathbb{P}(\mathbb{N})$  is countable infinite. Let  $S_1, S_2, \dots$ , be an enumeration of all subsets of numbers.
- Let  $D$  be the following diagonal subset of numbers.

$$D = \{i \mid i \notin S_i\}$$

- Since  $D$  is a set of numbers, by assumption,  $D = S_j$  for some  $j$ .
- **Question:** Is  $j \in D$ ?
  - If  $j \in S_j$  then  $j \notin D$  per definition of  $D$
  - If  $j \notin S_j$  then  $j \in D$  per definition of  $D$
- Contradiction in either case! Hence assumption is incorrect. No enumeration of  $\mathbb{P}(\mathbb{N})$  exists.

# Consequences for Computation

- How many  $\mathcal{C}$  programs are there? The set of  $\mathcal{C}$  programs is countably infinite since each of them can be represented as a string over a finite alphabet.
- How many languages are there? Uncountably many!
- Hence some (in fact almost all!) languages/boolean functions do not have any  $\mathcal{C}$  program to recognize them.

## Questions:

# Consequences for Computation

- How many  $\mathcal{C}$  programs are there? The set of  $\mathcal{C}$  programs is countably infinite since each of them can be represented as a string over a finite alphabet.
- How many languages are there? Uncountably many!
- Hence some (in fact almost all!) languages/boolean functions do not have any  $\mathcal{C}$  program to recognize them.

## Questions:

- Maybe all interesting languages/functions have  $\mathcal{C}$  programs? Are there interesting uncomputable languages?
- Why should  $\mathcal{C}$  programs be the definition of computability?
- Ok, there are difficult problems/languages. Which languages are computable and which ones have efficient algorithms?

# Easy languages

## Definition

A language  $L \subseteq \Sigma^*$  is **finite** if  $|L| = n$  for some integer  $n$ .

**Exercise:** Prove the following.

## Theorem

*The set of all finite languages is countably infinite.*

## Part III

# String Induction

# Inductive proofs on strings

Inductive proofs on strings and related problems follow inductive definitions.

## Definition

The **reverse**  $w^R$  of a string  $w$  is defined as follows:

- $w^R = \epsilon$  if  $w = \epsilon$
- $w^R = x^R a$  if  $w = ax$  for some  $a \in \Sigma$  and string  $x$

# Inductive proofs on strings

Inductive proofs on strings and related problems follow inductive definitions.

## Definition

The **reverse**  $w^R$  of a string  $w$  is defined as follows:

- $w^R = \epsilon$  if  $w = \epsilon$
- $w^R = x^R a$  if  $w = ax$  for some  $a \in \Sigma$  and string  $x$

## Theorem

*Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .*

Example:  $(dog \bullet cat)^R = (cat)^R \bullet (dog)^R = tacgod$ .



# Principle of mathematical induction

Induction is a way to prove statements of the form  $\forall n \geq 0, P(n)$  where  $P(n)$  is a statement that holds for integer  $n$ .

Example: Prove that  $\sum_{i=0}^n i = n(n+1)/2$  for all  $n$ .

Induction template:

- **Base case:** Prove  $P(0)$
- **Induction Step:** Let  $n > 0$  be **arbitrary** integer. Assuming that  $P(k)$  holds for  $0 \leq k < n$ , prove that  $P(n)$  holds.

Unlike the simple cases we will be working with various more complicated “structures” such as strings, tuples of strings, graphs etc. We need to translate a statement “ $Q$ ” into a (stronger or equivalent) statement that looks like “ $\forall n \geq 0, P(n)$ ” and then apply induction. We call  $\forall n \geq 0, P(n)$  the induction hypothesis.

# Proving the theorem

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Proof: by induction.

On what??  $|uv| = |u| + |v|$ ?

$|u|$ ?

$|v|$ ?

What does it mean to say “induction on  $|u|$ ”?

# By induction on $|u|$

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Proof by induction on  $|u|$  means that we are proving the following.

**Induction hypothesis:**  $\forall n \geq 0$ , for any string  $u$  of length  $n$  (for all strings  $v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ ).

# By induction on $|u|$

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Proof by induction on  $|u|$  means that we are proving the following.

**Induction hypothesis:**  $\forall n \geq 0$ , for any string  $u$  of length  $n$  (for all strings  $v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ ).

**Base case:** Let  $u$  be an arbitrary string of length 0.  $u = \epsilon$  since there is only one such string. Then

$$(uv)^R = (\epsilon v)^R = v^R = v^R \epsilon = v^R \epsilon^R = v^R u^R$$

# By induction on $|u|$

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Proof by induction on  $|u|$  means that we are proving the following.

**Induction hypothesis:**  $\forall n \geq 0$ , for any string  $u$  of length  $n$  (for all strings  $v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ ).

**Base case:** Let  $u$  be an arbitrary string of length 0.  $u = \epsilon$  since there is only one such string. Then

$$(uv)^R = (\epsilon v)^R = v^R = v^R \epsilon = v^R \epsilon^R = v^R u^R$$

Note that we did not assume anything about  $v$ , hence the statement holds for all  $v \in \Sigma^*$ .

# Inductive step

- Let  $u$  be an arbitrary string of length  $n > 0$ . Assume inductive hypothesis holds for all strings  $w$  of length  $< n$ .
- Since  $|u| = n > 0$  we have  $u = ay$  for some string  $y$  with  $|y| < n$  and  $a \in \Sigma$ .
- Then

# Inductive step

- Let  $u$  be an arbitrary string of length  $n > 0$ . Assume inductive hypothesis holds for all strings  $w$  of length  $< n$ .
- Since  $|u| = n > 0$  we have  $u = ay$  for some string  $y$  with  $|y| < n$  and  $a \in \Sigma$ .
- Then

$$(uv)^R =$$

# Inductive step

- Let  $u$  be an arbitrary string of length  $n > 0$ . Assume inductive hypothesis holds for all strings  $w$  of length  $< n$ .
- Since  $|u| = n > 0$  we have  $u = ay$  for some string  $y$  with  $|y| < n$  and  $a \in \Sigma$ .
- Then

$$\begin{aligned}(uv)^R &= ((ay)v)^R \\ &= (a(yv))^R && \text{associativity of concatenation} \\ &= (yv)^R a^R && \text{defn of reverse} \\ &= (v^R y^R) a^R && \text{by induction since } |y| < |u| \\ &= v^R (y^R a^R) && \text{associativity of concatenation} \\ &= v^R (ay)^R && \text{defn of reverse} \\ &= v^R u^R\end{aligned}$$



# Induction on $|v|$

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Proof by induction on  $|v|$  means that we are proving the following.

# Induction on $|v|$

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Proof by induction on  $|v|$  means that we are proving the following.

**Induction hypothesis:**  $\forall n \geq 0$ , for any string  $v$  of length  $n$  (for all strings  $u \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ ).

# Induction on $|v|$

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Proof by induction on  $|v|$  means that we are proving the following.

**Induction hypothesis:**  $\forall n \geq 0$ , for any string  $v$  of length  $n$  (for all strings  $u \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ ).

**Base case:** Let  $v$  be an arbitrary string of length 0.  $v = \epsilon$  since there is only one such string. Then

$$(uv)^R = (u\epsilon)^R = u^R = \epsilon u^R = \epsilon^R u^R = v^R u^R$$

# Inductive step

- Let  $v$  be an arbitrary string of length  $n > 0$ . Assume inductive hypothesis holds for all strings  $w$  of length  $< n$ .
- Since  $|v| = n > 0$  we have  $v = ay$  for some string  $y$  with  $|y| < n$  and  $a \in \Sigma$ .
- Then

$$\begin{aligned}(uv)^R &= (u(ay))^R \\ &= ((ua)y)^R \\ &= y^R(ua)^R \\ &= ??\end{aligned}$$

# Inductive step

- Let  $v$  be an arbitrary string of length  $n > 0$ . Assume inductive hypothesis holds for all strings  $w$  of length  $< n$ .
- Since  $|v| = n > 0$  we have  $v = ay$  for some string  $y$  with  $|y| < n$  and  $a \in \Sigma$ .
- Then

$$\begin{aligned}(uv)^R &= (u(ay))^R \\ &= ((ua)y)^R \\ &= y^R(ua)^R \\ &= ??\end{aligned}$$

Cannot simplify  $(ua)^R$  using inductive hypothesis. Mainly because we defined reverse with  $w = ax$  rather than  $w = xa$ . Can simplify if we extend base case to include  $n = 0$  and  $n = 1$ . However,  $n = 1$  itself requires induction on  $|u|$ !

# Induction on $|u| + |v|$

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Induction on  $|u| + |v|$  means:

# Induction on $|u| + |v|$

## Theorem

*Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .*

Induction on  $|u| + |v|$  means:

**Induction hypothesis:**

# Induction on $|u| + |v|$

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Induction on  $|u| + |v|$  means:

**Induction hypothesis:**  $\forall n \geq 0$ , for any  $u, v \in \Sigma^*$  with  $|u| + |v| \leq n$ ,  $(uv)^R = v^R u^R$ .



# Induction on $|u| + |v|$

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Induction on  $|u| + |v|$  means:

**Induction hypothesis:**  $\forall n \geq 0$ , for any  $u, v \in \Sigma^*$  with  $|u| + |v| \leq n$ ,  $(uv)^R = v^R u^R$ .

**Base case:**  $n = 0$ . Let  $u, v$  be an arbitrary strings such that  $|u| + |v| = 0$ . Implies  $u, v = \epsilon$ . Easy to handle

# Induction on $|u| + |v|$

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Induction on  $|u| + |v|$  means:

**Induction hypothesis:**  $\forall n \geq 0$ , for any  $u, v \in \Sigma^*$  with  $|u| + |v| \leq n$ ,  $(uv)^R = v^R u^R$ .

**Base case:**  $n = 0$ . Let  $u, v$  be an arbitrary strings such that  $|u| + |v| = 0$ . Implies  $u, v = \epsilon$ . Easy to handle

**Inductive step:**  $n > 0$ . Let  $u, v$  be arbitrary strings such that  $|u| + |v| = n$ .

# Induction on $|u| + |v|$

## Theorem

Prove that for any strings  $u, v \in \Sigma^*$ ,  $(uv)^R = v^R u^R$ .

Induction on  $|u| + |v|$  means:

**Induction hypothesis:**  $\forall n \geq 0$ , for any  $u, v \in \Sigma^*$  with  $|u| + |v| \leq n$ ,  $(uv)^R = v^R u^R$ .

**Base case:**  $n = 0$ . Let  $u, v$  be an arbitrary strings such that  $|u| + |v| = 0$ . Implies  $u, v = \epsilon$ . Easy to handle

**Inductive step:**  $n > 0$ . Let  $u, v$  be arbitrary strings such that  $|u| + |v| = n$ .

**Subcase:**  $u = \epsilon$  then easy.

**Subcase:**  $|u| \geq 1$  implies  $u = ay$ . Similar to induction on  $|u|$ .