



# Formal Languages-Course 1.

Géraud Sénizergues

Bordeaux university

04/05/2020

Master computer-science MINF19, IEI, 2019/20

# contents

# Introduction

# What are formal languages

A language is, informally, a way to communicate by means of sequences of symbols (whether written symbols or spoken elementary sounds).
Examples :
*Natural* languages :Written english - spoken english- written international scientific english- written Vietnamese- written French, etc ...
*Artificial* languages : Formal mathematical language (as defined by some logical system, the "sequent calculus" for example), Lisp (a programming language), Lambda-calculus (a "theoretical" programming language), etc ...

# What are formal languages

Two aspects for each language :
the *syntax* : what are the correct sequences of symbols (words, sentences, discourses,...)
the *semantics* : what does mean a sentence ? One hopes the meaning of a sentence can be deduced from the structure of the sentence and the meaning of each elementary component of the sentence (compositionality principle).

# What are formal languages

Both aspects (syntax and semantics) can be described by

mathematical models.

A model for the syntax is a formal language i.e. a set of words.
A model for the semantics is a formal semantics i.e. a mathematical definition of a functional :
$\{$correct sentences$\} \rightarrow \{$mathematical functions$\}$ that associates to every sentence its meaning.

# Brief history

1850-1920-now : mathematicians defined formal languages expressing mathematics [Frege, Peano, Pieri, Hilbert, Russel-Whitehead ]. These were languages used to model the mathematical proofs. Theses languages were not rigorously used in practise. They were necessary to study the power of mathematics : what is provable and what is not provable.
1920-1960-now : linguists modeled natural languages by several kinds of formal grammars : AB-grammars, context-free grammars, Lambek-grammars, etc ...
1950-now : computer-scientists defined programming languages as formal languages, using rational expressions, grammars, automata,

# Brief history

1960-now :logicians [Buchi, Rabin, Shellah, Muchnik-Semenov,etc ...] used automata in order to show decidability (and other properties) of some logical theories.
1970-now :Some formal languages expressing mathematics were practically implemented on computers ; thus formal proofs could be produced for many mathematical theorems [Automath 1970], HOL [produced the formal proof of the Kepler conjecture], Mizar, COQ [ACM software prize,2013], PVS,etc ...

# What shall we study

For describing a formal language $L$ we can use :
- "algebraic operations"
operations on words and operations on languages : $w \in L$ if $w$ can be obtained by a specific sequence of operations
- "rewriting"
$w \in L$ if $w$ can be obtained from a starting word $w_0$ by by a sequence of rewritings i.e. of local modifications
- "abstract machines" (automata)
$w \in L$ if the automaton $A$, on input $w$,computes, and eventually answers "yes".

# What shall we study

Algorithmic problems :
**INPUT** : a description of a language (a rational expression, a grammar),
**OUTPUT** : an automaton that recognizes the language.
The "automaton"is a mathematical object that can be easily turned into a program that solves the problem on a computer.

**INPUT** : a word and a language,
**QUESTION** : does the word belong to the language ?
Example : given an allegedly C-program P : is P really a C-program ?

# What shall we study

**INPUT** : two languages L,M :
**QUESTION** : $L = M$ ?
Example : given a specification of a language and a program that analyzes words, does the program implement the specification ? i.e. does it recognize exactly what was defined as correct words ?

**INPUT** : a description D of some kind,
**OUTPUT** : a description E of another kind, defining the same language.
Example : translate an automaton into a rational expression.

# Where is it useful

Compiling : a compiler is a device that translates a given source-language $S$ into a given target-language $T$, while preserving the semantics. The first step of a compiler consists in the lexical analysis of the input-text

$$P \to P_1$$

The second step consists in the syntactic analysis of the text $P_1$ :

$$P_1 \to P_2$$

where $P_2$ is the abstract-syntax tree of $P_1$. These two steps strongly lean on formal languages theory.

# Where is it useful

Computer-architecture : building a finite-automaton is a step on the way to build a sequential boolean circuit that computes a given function.
Computational biology : many algorithms on long words (that model proteins or DNA-sequences) are based on automata or on rewriting-systems.
Automatic language processing : in order to treat natural language text and extract its semantics or to produce some natural language sentence expressing some meaning, the natural language must be modeled after some grammar and the semantics must be modeled as some function of the structure of the sentences (expressed by some tree).

# Formal languages : Words

# Free Monoid

Let $\Sigma$ be a "set of symbols" (mathematically, $\Sigma$ is just some set). We call word over the alphabet $\Sigma$ any finite sequence of symbols from $\Sigma$ :
Examples :

$$\Sigma = \{a, b, c\},$$

$$w_1 = aabcabbc, \quad w_2 = bcabbcaaaa, \quad w_3 = cccaa, \quad w_4 = c.$$

Length :

$$|w_1| = 8, \quad |w_2| = 10, \quad |w_3| = 5, \quad |w_4| = 1$$

the length of a word $u$ is the number of positions of $u$.
One can denote the letter in position $i$ of a word $u$ by $u[i]$. For example :

$$w_3[0] = c, w_3[1] = c, w_3[2] = c, w_3[3] = a, w_3[4] = a$$

# Free Monoid

Length relative to a letter :
For a letter $x \in \Sigma$ and a word $u \in \Sigma^*$ we note $|u|_x$ the number of positions of $u$ that are labelled by letter $x$. Formally :

$$|u|_x = \mathrm{Card}\{i \in [0, |u| - 1], u[i] = x\}.$$

## Example

$$u = abaccbaaa$$

$$|u|_a = 5, \quad |u|_b = 2, \quad |u|_c = 2$$

$$|u| = 9 = |u|_a + |u|_b + |u|_c.$$

# Free Monoid

**Product :**

$$w_1 \cdot w_2 = aabcabbc \cdot bcabbcaaaa = aabcabbcbcabbcaaaa$$

For words the "concatenation product" of $u, v$ is obtained by putting $u$ before $v$ and omitting the right-end of $u$ and left-end of $v$.

$$w_3 \cdot w_4 = cccaa \cdot c = cccaac.$$

**Empty-word :**
We denote by $\varepsilon$ the empty-word : it has length 0. The product by $\varepsilon$ does not modify any word i.e. for every word $u$ :

$$u \cdot \varepsilon = \varepsilon \cdot u = u$$

# Free Monoid

Product and length : for every words $u, v$ :

$$|u \cdot v| = |u| + |v|.$$

for every words $u, v$ and letter $x$ :

$$|u \cdot v|_x = |u|_x + |v|_x.$$

# Free Monoid

$\Sigma^*$ : we denote by $\Sigma^*$ the set of all words over the alphabet $\Sigma$.
The law $\cdot$ has the properties : for every $u, v, w \in \Sigma^*$

$$u \cdot (v \cdot w) = (u \cdot v) \cdot w. \tag{1}$$

$$u \cdot \varepsilon = \varepsilon \cdot u = u \tag{2}$$

# Free Monoid

Properties (1,2) are summarized by :

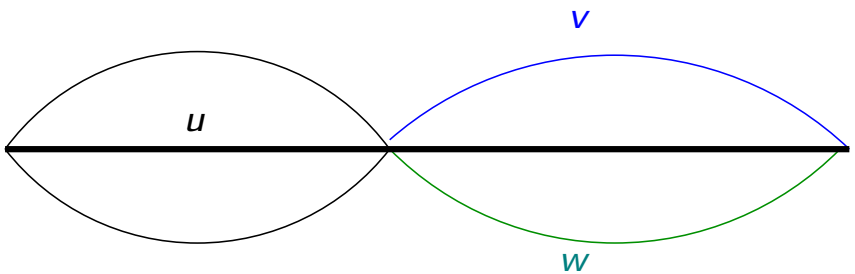
$$\langle \Sigma^*, \cdot, \varepsilon \rangle \text{ is a monoid}$$

It is called the free-monoid over $\Sigma$. Other (useful) properties : for every $u, v, w \in \Sigma^*$

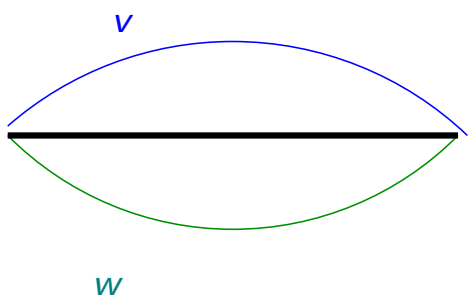
$$u \cdot v = u \cdot w \Rightarrow v = w \text{ and } u \cdot w = v \cdot w \Rightarrow u = v \qquad (3)$$

The free monoid is *cancellative*.

# Free Monoid :left-cancellation

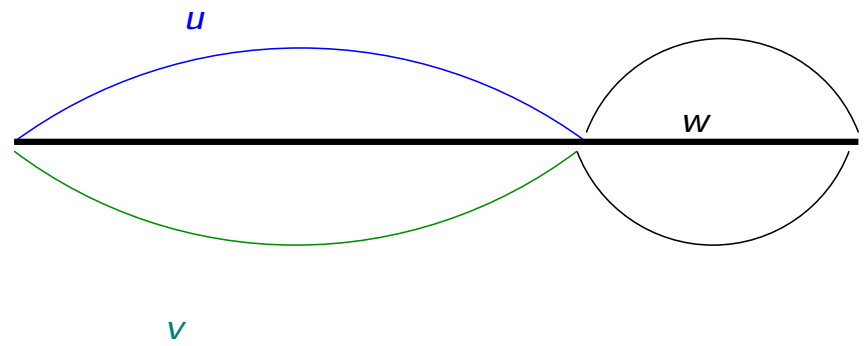



$uv = uw$

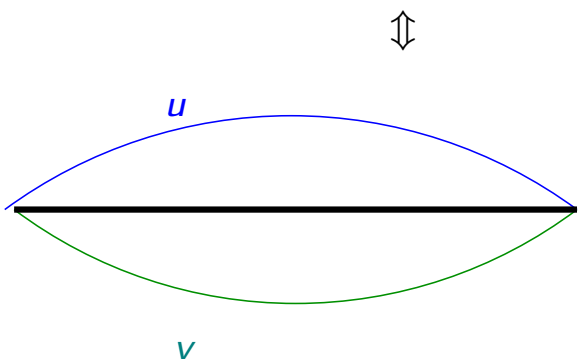



$v = w$

# Free Monoid :right-cancellation




$uw = vw$




$u = v$

# Free Monoid :reversal

The reversal of a word $u$ of length $\ell$ is defined by :

$$u^R := u[\ell - 1]u[\ell - 2] \cdots u[0].$$

Example :

$$u = abcbbc, \quad u^R = cbbcba.$$

**Proposition**

$\forall u, v \in \Sigma^*, (u \cdot v)^R = v^R \cdot u^R, \quad \varepsilon^R = \varepsilon.$

# Free Monoid :factors

A word $v$ is called a factor of a word $w$ iff, there exist words $\alpha, \beta$ such that

$$\alpha \cdot v \cdot \beta = w$$

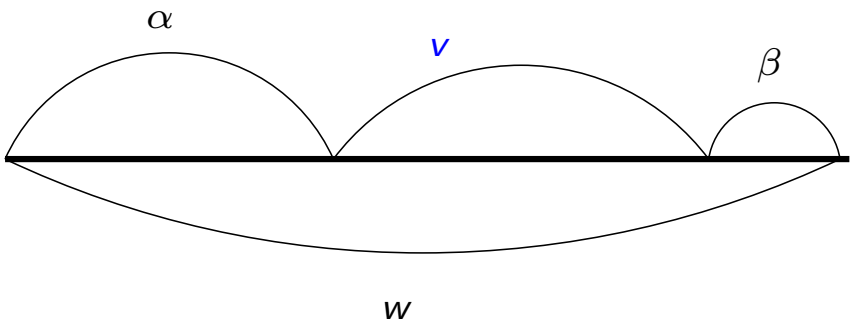
A word $v$ is called a prefix of a word $w$ iff, there exists a word $\beta$ such that

$$v \cdot \beta = w$$

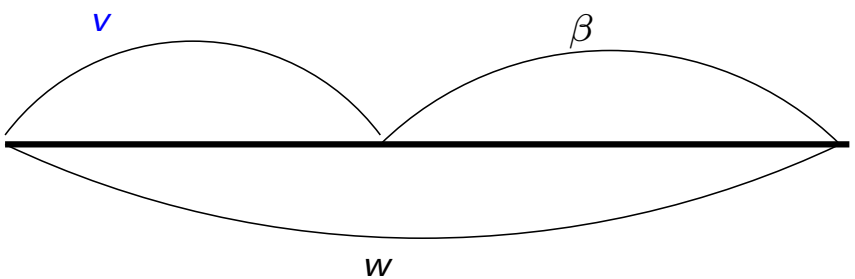
A word $v$ is called a suffix of a word $w$ iff, there exist a word $\alpha$ such that

$$\alpha \cdot v = w$$

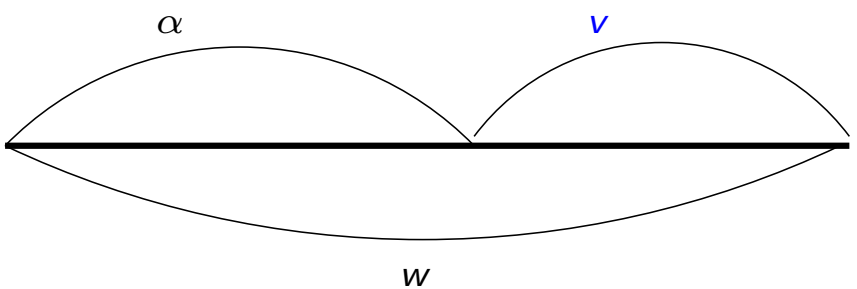
# Free Monoid :factors




$v$ is factor of $w$




$v$ is prefix of $w$




$v$ is suffix of $w$

# Free Monoid :reversal-factors

## Example

$\Sigma = \{a, b, c\}$.

$$u = abb, v_1 = aaabaab, v_2 = ababab, v_3 = bbabb$$

$u$ *is* not *factor of* $v_1$.
$u$ *is* not *factor of* $v_2$.
$u$ *is factor of* $v_3$.
$u$ *is suffix of* $v_3$.
$u$ *is* not *prefix of* $v_3$

# Free Monoid :reversal-factors

## Example

*(continued)* $\Sigma = \{a, b, c\}$.

$$u = abb, v_4 = ababbabbbabbbb$$

*u is factor of* $v_4$ *:*

$$v_4 = ababbabbbabbbb = ababbabbbabbbb = ababbabbbabbbb$$

*we say : u has 3 occurrences as a factor of* $v_4$ *:*
*one in position 2, one in position 5, one in position 9.*
*u is not prefix of* $v_4$*.*
*u is not suffix of* $v_4$*.*

# Free Monoid :reversal-factors

## Example

*(continued)* $\Sigma = \{a, b, c\}$.

$$u = abb, v_5 = abbcabb$$

*u prefix and suffix of* $v_5$ *:*

$$v_5 = abbcabb = abbcabb.$$

# Free Monoid :reversal-factors

## Exercice

*Let* $\Sigma = \{a, b, c\}$.

$$u = abbaa, v_1 = babbaab, v_2 = abbabbaabb$$

$$v_3 = aaabbabbabba, v_4 = bbaa, v_5 = abababbaa$$

*1- For which* $i \in [1, 5]$ *is true that* $u$ *is factor of* $v_i$ *?*
*2- For which* $i \in [1, 5]$ *is true that* $u$ *is prefix of* $v_i$ *?*
*3- For which* $i \in [1, 5]$ *is true that* $u$ *is suffix of* $v_i$ *?*

# Free Monoid :reversal-factors

## Exercice

*Let us suppose that $u, v$ are words such that $u$ is factor of $v$. Does it imply, in general, that :*
*1- $u$ is factor of $v^R$ ?*
*2- $u^R$ is factor of $v^R$ ?*
*3- $u^R$ is factor of $v$ ?*
**Hint** *: use Proposition 2.2.*

## Exercice

*Let us suppose that $u, v$ are words such that $u$ is prefix of $v$. Does it imply, in general, that :*
*1- $u$ is prefix of $v^R$ ?*
*2- $u^R$ is prefix of $v^R$ ?*
*3- $u^R$ is suffix of $v^R$ ?*

# Free Monoid :reversal-factors

## Exercice

*Let $\Sigma$ be an alphabet. Show that : for every words $u_1, u_2, v \in \Sigma^*$, if $u_1, u_2$ are both prefixes of $v$, then ($u_1$ is prefix of $u_2$) or ($u_2$ is prefix of $u_1$)*

## Exercice

*Let $\Sigma$ be an alphabet. A word $v \in \Sigma^*$ is called square-free if it does not have any factor of the form $w \cdot w$ for any $w \neq \varepsilon$.*
*For example $v_0 = abcabaca$ is square-free, abcbcba is not square-free since $cb \cdot cb$ is factor of $v_0$.*
*Show that, on the alphabet $\Sigma = \{a, b\}$ there are only finitely many square-free words. Compute the set of all square-free words on $\{a, b\}$.*

# Free Monoid :reversal-factors

## Exercice

*(difficult, but solved in combinatorics-textbooks)*
*A word $v$ is called cube-free if it does not have any factor of the form $w \cdot w \cdot w$ for any word $w \neq \varepsilon$.*
*Show that, on the alphabet $\Sigma = \{a, b\}$ there are infinitely many cube-free words.*

# Free Monoid :reversal-factors

## Exercice

*Let $\Sigma$ be an alphabet. Show that : for every words $u, v, u', v' \in \Sigma^*$, $u \cdot v = u' \cdot v'$ if and only if there exists some word $w \in \Sigma^*$ such that,*

$$(u = u' \cdot w \text{ and } v' = w \cdot v) \text{ or } (u' = u \cdot w \text{ and } v = w \cdot v').$$

**Hint** *: Distinguish the case where $|u| \leq |u'|$ and the case where $|u'| \leq |u|$ ; a picture will be helpful.*

This property is called : equidivisibility of the free monoid.

# Free Monoid :powers

The power $u^n$ of the word $u$ is defined by induction on the natural integer $n$ :

$$u^0 = \varepsilon, \quad u^{n+1} = u^n \cdot u$$

For example : $u^1 = u$
$u^2 = u \cdot u$, (it is called the square of $u$),
$u^3 = u \cdot u \cdot u$, (it is called the cube of $u$).

# Formal languages :Languages

# Definition-Product

## Definition

*We call* language *over the alphabet* $\Sigma$ *any subset* $L$ *of* $\Sigma^*$.

Examples :
Let $\Sigma = \{a, b, c\}$. Here are some languages over $\Sigma$ :

$$L_1 = \{a, ab, ac, bcc\}, \quad L_2 = \{ac, ca\},$$

$L_3 = \{u \in \Sigma^* \mid |u| \text{ is even } \}$, $L_4 = \{u \in \Sigma^* \mid \exists \alpha, \beta \in \Sigma^* \mid u = \alpha \cdot bba \cdot \beta\}$.

Product :for $L, M$ languages over $\Sigma$ we define

$$L \cdot M = \{u \cdot v \mid u \in L, v \in M\}$$

Example :

$$L_1 \cdot L_2 = \{aac, abac, acac, bccac, aca, abca, acca, bccca\}.$$

# Reversal

The reversal of a language $L$ is defined by :

$$L^R = \{u^R \mid u \in L\}.$$

## Example

$$L_1 = \{a, ab, ac, bcc\}, \quad L_2 = \{ac, ca\},$$
$$L_1{}^R = \{a, ba, ca, ccb\}, \quad L_2{}^R = \{ca, ac\}$$

# Empty-Unit

The empty language is denoted by $\emptyset$ : it has no element :

$$\forall u \in \Sigma^*, u \notin \emptyset.$$

The unit language $\{\varepsilon\}$ has the property that : for every language $L$ :

$$L \cdot \{\varepsilon\} = \{\varepsilon\} \cdot L = L.$$

while :

$$L \cdot \emptyset = \emptyset, \quad \emptyset \cdot L = \emptyset.$$

# Power

The power $L^n$ of the language $L$ is defined by induction on the natural integer $n$ :

$$L^0 = \{\varepsilon\}, \quad L^{n+1} = L^n \cdot L$$

For example : $L^1 = L$
$L^2 = L \cdot L$, (it is called the square of $L$),

## Example

$$L = \{ab, ca\}, L^2 = \{abab, abca, caab, caca\}$$

*note that* $abca \in L^2$ *but* $abca$ *is not a square.*

# Boolean operations

union :

$$L \cup M = \{u \in \Sigma^* \mid u \in L \text{ or } u \in M\}$$

intersection :

$$L \cap M = \{u \in \Sigma^* \mid u \in L \text{ and } u \in M\}$$

set difference :

$$L \setminus M = \{u \in \Sigma^* \mid u \in L \text{ and } u \notin M\}$$

complement :

$$CL = \Sigma^* \setminus L.$$

# Boolean operations

Note that :

$$L \setminus M = L \cap \mathsf{C}M$$

$\cup, \cap$ are both commutative and associative ; each of these two operations is distributive on the other :

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

# Star

star :

$$L^* := \bigcup_{n=0}^{\infty} L^n.$$

Example :
$L = \{ab, acc\}$

$$L^* = \{\varepsilon, ab, acc, abab, abacc, accab, accacc, ababab, abaccab, accabab, accaccab, ababacc, abaccacc, accabacc, accaccacc, \ldots\}$$

# Cross

cross :

$$L^+ := \bigcup_{n=1}^{\infty} L^n.$$

Example(continued) :
$L = \{ab, acc\}$

$$L^+ = \{ab, acc, abab, abacc, accab, accacc, ababab, abaccab, accabab, accaccab, ababacc, abaccacc, accabacc, accaccacc, \ldots\}$$

# Left-quotient

Left-quotient :

Let $u \in \Sigma^*, L \subseteq \Sigma^*$.

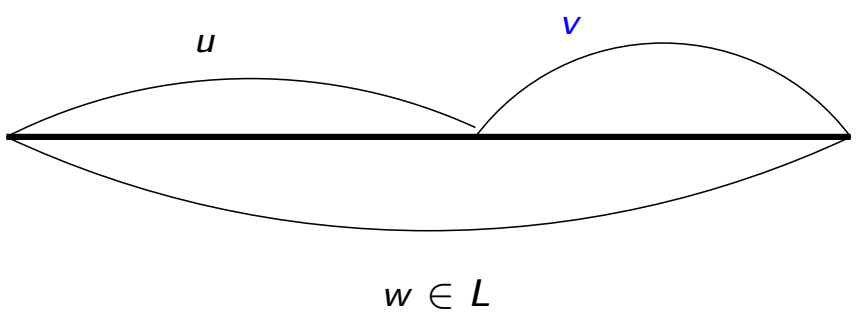
$$u^{-1}L = \{v \in \Sigma^* \mid uv \in L\}$$

Let $U \subseteq \Sigma^*, L \subseteq \Sigma^*$.

$$U^{-1}L = \{v \in \Sigma^* \mid \exists u \in U, uv \in L\}$$

Note that

$$U^{-1}L = \bigcup_{u \in U} u^{-1}L$$

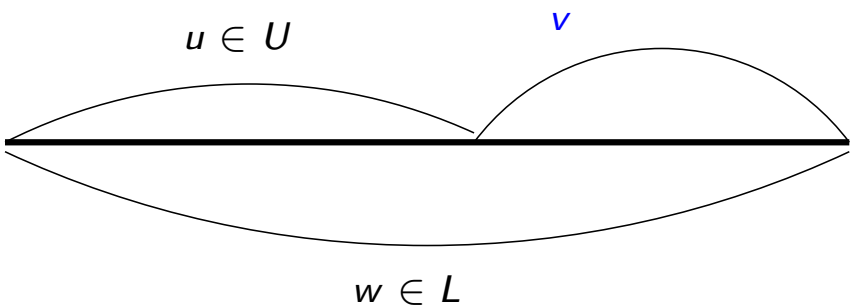



$v \in u^{-1}L$

# Left-quotient

Left-quotient :

$$U^{-1}L = \{v \in \Sigma^* \mid \exists u \in U, uv \in L\}$$




$v \in U^{-1}L$

# Right-quotient

Right-quotient :

Let $u \in \Sigma^*, L \subseteq \Sigma*$.

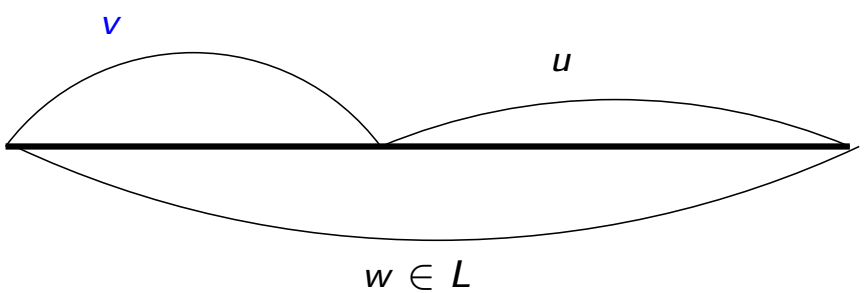
$$Lu^{-1} = \{v \in \Sigma^* \mid vu \in L\}$$

Let $U \subseteq \Sigma^*, L \subseteq \Sigma^*$.

$$LU^{-1} = \{v \in \Sigma^* \mid \exists u \in U, vu \in L\}$$

Note that

$$LU^{-1} = \bigcup_{u \in U} Lu^{-1}$$




$v \in Lu^{-1}$

# Quotient

## Example

$$Let\ L = \{ab, acb, abb, bba\}$$

$$(ab)^{-1}L = \{\varepsilon, b\},\quad (bb)^{-1}L = \{a\},\quad a^{-1}L = \{b, cb, bb\}$$

$$Lb^{-1} = \{a, ac, ab\},\quad La^{-1} = \{bb\},$$

# Quotient

## Example

*Let* $L = (aab)^*$

$$(aa)^{-1}L = b \cdot (aab)^*, \quad (b)^{-1}L = \emptyset, \quad (aab)^{-1}L = L.$$

$$L(aa)^{-1} = \emptyset, \quad Lb^{-1} = (aab)^* \cdot aa, \quad L(aab)^{-1} = L.$$