Formal Language (3A)

- Regular Language

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

Formal Language

a **formal language** is
a set of **strings** of **symbols**
together with a set of **rules**
that are specific to it.

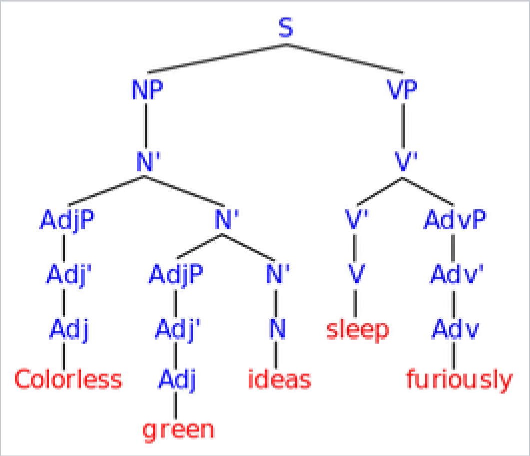
https://en.wikipedia.org/wiki/Formal_language

Alphabet and Words

The **alphabet** of a formal language is
the set of **symbols**, **letters**, or **tokens**
from which the **strings** of the language may be formed.

The **strings** formed from this alphabet
are called **words**

the **words** that belong to a particular formal language
are sometimes called **well-formed words**
or **well-formed formulas**.



Structure of a syntactically well-formed, although nonsensical, English sentence (historical example from Chomsky 1957).

https://en.wikipedia.org/wiki/Formal_language

Formal Language

A **formal language** (**formation rule**)
is often defined by means of
a **formal grammar**
such as a **regular grammar** or
context-free grammar,

https://en.wikipedia.org/wiki/Formal_language

Formal Language and Natural Language

The field of **formal language** theory studies primarily the purely **syntactical aspects** of such languages— that is, their internal **structural patterns**.

Formal language theory sprang out of linguistics, as a way of understanding the **syntactic regularities** of **natural languages**.

formalized versions of subsets of **natural languages** in which the words of the language represent **concepts** that are associated with particular **meanings** or **semantics**.

https://en.wikipedia.org/wiki/Formal_language

Formal Language and Programming Languages

In computer science, formal languages are used among others as the basis for defining the **grammar** of **programming languages**

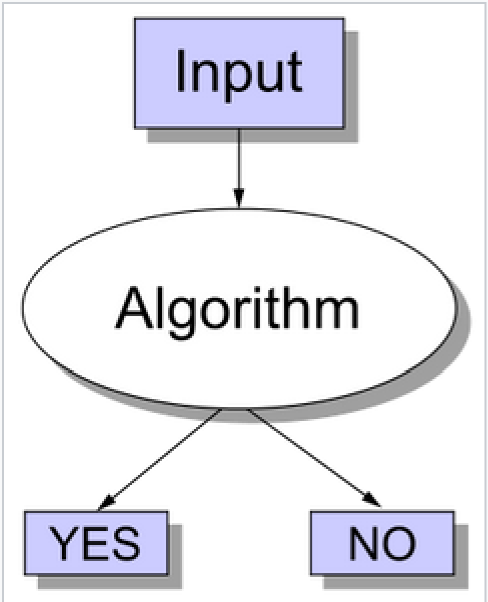
https://en.wikipedia.org/wiki/Formal_language

Formal Language and Complexity Theory

In computational **complexity theory**, **decision problems** are typically defined as formal languages, and

complexity classes are defined as the sets of the formal languages that can be parsed by machines with limited computational power.

These inputs can be natural numbers, but may also be values of some other kind, such as strings over the binary alphabet {0,1} or over some other finite set of symbols. The subset of strings for which the problem returns "yes" is a formal language, and often decision problems are defined in this way as formal languages.



A *decision problem* has only two possible outputs (*yes* or *no*) on any input.

https://en.wikipedia.org/wiki/Formal_language
https://en.wikipedia.org/wiki/Decision_problem

Formal Language and Logic / Mathematics

In **logic** and the foundations of **mathematics**,
formal languages are used to represent
the **syntax** of **axiomatic systems**,
and **mathematical formalism** is
the philosophy that all of mathematics
can be reduced to the **syntactic manipulation**
of formal languages in this way.

https://en.wikipedia.org/wiki/Formal_language

Alphabet

An **alphabet** can be any set
think a **character set** such as ASCII.
the elements of an alphabet are called its **letters**.

an **infinite** number of elements

a **finite** number of elements

https://en.wikipedia.org/wiki/Formal_language

Words over an alphabet

A **word** over an **alphabet** can be
any finite sequence (i.e., string) of **letters**.

The set of all words over an **alphabet Σ**
is usually denoted by **Σ*** (using the **Kleene star**).

The **length** of a word is the number of letters
only one word of **length 0**, the **empty word** (e / ε / λ or even Λ)
By **concatenation** one can combine two words to form a new word

in logic, the **alphabet** is also known as the **vocabulary**
and **words** are known as **formulas** or **sentences**;

the **letter**/**word** metaphor : formal language
a **word**/**sentence** metaphor : logic

https://en.wikipedia.org/wiki/Formal_language

Kleene star

Given a set V define

V_0 = {ε} (the language consisting only of the empty string),
V_1 = V

and define recursively the set

V_{i+1} = { wv : w ∈ V_i and v ∈ V } for each i>0.

$$V^* = \bigcup_{i \in \mathbb{N}} V_i = \{\varepsilon\} \cup V \cup V_2 \cup V_3 \cup V_4 \cup \ldots.$$

*: zero or more

$$V^+ = \bigcup_{i \in \mathbb{N} \setminus \{0\}} V_i = V_1 \cup V_2 \cup V_3 \cup \ldots.$$

+: one or more

$$\mathbb{N}^0 = \mathbb{N}_0 = \{0, 1, 2, \ldots\}$$
$$\mathbb{N}^* = \mathbb{N}^+ = \mathbb{N}_1 = \mathbb{N}_{>0} = \{1, 2, \ldots\}.$$

https://en.wikipedia.org/wiki/Kleene_star

$$\mathbb{N} = \{0, 1, 2, \ldots\}.$$
$$\mathbb{Z}^+ = \{1, 2, \ldots\}.$$

Kleene star examples (1)

{"ab","c"}* = { ε, "ab", "c", "abab", "abc", "cab", "cc", "ababab", "ababc", "abcab", "abcc", "cabab", "cabc", "ccab", "ccc", ...}.

{"a", "b", "c"}+ = { "a", "b", "c", "aa", "ab", "ac", "ba", "bb", "bc", "ca", "cb", "cc", "aaa", "aab", ...}.

{"a", "b", "c"}* = { ε, "a", "b", "c", "aa", "ab", "ac", "ba", "bb", "bc", "ca", "cb", "cc", "aaa", "aab", ...}.

∅* = {ε}.

∅+ = ∅

∅* = { } = ∅,

https://en.wikipedia.org/wiki/Kleene_star

Kleene star examples (2)

{ab, c}* =
{ ε,
ab, c,
abab, abc, cab, cc,
ababab, ababc, abcab, abcc, cabab, cabc, ccab, ccc, … }

{a, b, c}+ =
{ a, b, c,
aa, ab, ac, ba, bb, bc, ca, cb, cc,
aaa, aab, aac, aba, abb, abc, aca, acb, acc, baa, bab, bac, bba, bbb, bbc, bca, bcb, bcc, … }

{a, b, c}*
{ ε
a, b, c,
aa, ab, ac, ba, bb, bc, ca, cb, cc,
aaa, aab, aac, aba, abb, abc, aca, acb, acc, baa, bab, bac, bba, bbb, bbc, bca, bcb, bcc, … }

https://en.wikipedia.org/wiki/Kleene_star

Kleene star examples (3)

regular expression
((1*) 0 (1*) 0 (1*))*,

(1*) = {ε, 1, 11, 111, …}

$$\begin{Bmatrix} e \\ 1 \\ 11 \\ 111 \\ \vdots \end{Bmatrix} 0 \begin{Bmatrix} e \\ 1 \\ 11 \\ 111 \\ \vdots \end{Bmatrix} 0 \begin{Bmatrix} e \\ 1 \\ 11 \\ 111 \\ \vdots \end{Bmatrix}$$

$$\begin{matrix} 00\{e,1,11,111,\cdots\} & 100\{e,1,11,111,\cdots\} & 1100\{e,1,11,111,\cdots\} & 11100\{e,1,11,111,\cdots\} \\ 010\{e,1,11,111,\cdots\} & 1010\{e,1,11,111,\cdots\} & 11010\{e,1,11,111,\cdots\} & 111010\{e,1,11,111,\cdots\} \\ 0110\{e,1,11,111,\cdots\} & 10110\{e,1,11,111,\cdots\} & 110110\{e,1,11,111,\cdots\} & 1110110\{e,1,11,111,\cdots\} \\ 01110\{e,1,11,111,\cdots\} & 101110\{e,1,11,111,\cdots\} & 1101110\{e,1,11,111,\cdots\} & 11101110\{e,1,11,111,\cdots\} \\ \vdots & \vdots & \vdots & \vdots \end{matrix}$$

https://en.wikipedia.org/wiki/Kleene_star

Formal Language Definition

A **formal language L** over an **alphabet Σ** is
a **subset of Σ***, that is, a set of **words** over that alphabet.

Sometimes the sets of **words** are grouped into **expressions**, whereas **rules** and **constraints** may be formulated for the creation of '**well-formed expressions**'.

https://en.wikipedia.org/wiki/Formal_language

Formal Language Examples (1)

The following **rules** describe
a **formal language L**
over the **alphabet Σ** = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, = }:

- every nonempty string is in **L**
  - that does not contain "**+**" or "**=**"
  - does not start with "**0**"
- the string "**0**" is in **L**.
- a string containing "**=**" is in **L**
  - if and only if there is exactly one "**=**",
  - and it separates two valid strings of **L**.
- a string containing "**+**" but not "=" is in **L**
  - if and only if every "**+**" in the string separates two valid strings of **L**.
- no string is in **L** other than those implied by the previous rules.

https://en.wikipedia.org/wiki/Formal_language

Formal Language Examples (2)

Under these rules,
the string "**23+4=555**" is in L,
but the string "**=234=+**" is not.

This formal language expresses

- **natural numbers**,
- well-formed **additions**,
- and well-formed **addition equalities**,

but it expresses only what they look like (their **syntax**),
not what they mean (**semantics**).

for instance, nowhere in these rules is
there any indication that "0" means the number zero,
or that "+" means addition.

https://en.wikipedia.org/wiki/Formal_language

Formal Language Examples (3)

- L = Σ*, the set of all **words** over Σ;
- L = {"a"}* = {"a"^{n}}, where n ranges over the natural numbers

  and "a"^{n} means "a" repeated n times

  (this is the set of words consisting only of the symbol "a");
- the set of **syntactically correct** programs in a given programming language (the syntax of which is usually defined by a **context-free grammar**);
- the set of inputs upon which a certain **Turing machine** halts; or
- the set of **maximal strings** of alphanumeric ASCII characters on this line, i.e., the set {"the", "set", "of", "maximal", "strings", "alphanumeric", "ASCII", "characters", "on", "this", "line", "i", "e"}.

https://en.wikipedia.org/wiki/Formal_language

Formal Language Examples (4)

For instance, a **language** can be given as

- those strings generated by some **formal grammar**;
- those strings described or matched by a particular **regular expression**;
- those strings accepted by some automaton,

  such as a **Turing machine** or **finite state automaton**;
- those **strings** for which some **decision procedure** produces the answer YES.

  (an algorithm that asks a sequence of related YES/NO questions)

https://en.wikipedia.org/wiki/Formal_language

Formal Grammar Example

the alphabet consists of **a** and **b**,
the start symbol is **S**,
the **production rules**:

1. **S** → **aSb**
2. **S** → **ba**

then we start with **S**, and can choose a rule to apply to it.
Application of rule 1, the string **aSb**.
Another application of rule 1, the string **aaSbb**.
Application of rule 2, the string **aababb**

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aababb$$

The **language** of the grammar is then the infinite set

$$\{a^n bab^n \mid n \geq 0\} = \{ba, abab, aababb, aaababbb, \ldots\}$$

https://en.wikipedia.org/wiki/Formal_language

Syntax of Formal Grammars

a **grammar** G consists of the following components:

- A finite set **N** of **nonterminal symbols**,
  that is disjoint with the strings formed from G.
- A finite set **Σ** of **terminal symbols**
  that is disjoint from **N**.
- A finite set **P** of **production rules**,
  $$(\Sigma \cup N)^* N (\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*$$
- A distinguished symbol **S** ∈ **N** that is the **start symbol**,
  also called the **sentence symbol**.

A grammar is formally defined as the tuple (**N** ,**Σ**, **P**, **S**)

often called a rewriting system
or a phrase structure grammar

https://en.wikipedia.org/wiki/Formal_language

Terminal and Non-terminal Symbols

Terminal symbols are the **elementary symbols** of the language defined by a formal grammar.

Nonterminal symbols (or **syntactic variables**) are replaced by groups of **terminal symbols** according to the **production rules**.

A formal grammar includes a **start symbol**, a designated member of the set of **nonterminals** from which all the strings in the language may be derived by successive applications of the **production rules**.

In fact, the language defined by a grammar is precisely the set of **terminal strings** that can be so derived.

https://en.wikipedia.org/wiki/Terminal_and_nonterminal_symbols

Production Rules

$$(\Sigma \cup N)^* N (\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*$$

Head → Body

- each **production rule** maps from one string of symbols to another
- the first string (the "head") contains
  - an arbitrary number of symbols
  - provided at least one of them is a **nonterminal**. ***N***
- If the second string (the "body") consists solely of the **empty string**
  - i.e., that it contains no symbols at all
  - it may be denoted with a special notation (Λ , e or ϵ)

https://en.wikipedia.org/wiki/Formal_language

Grammar Examples (1)

Consider the grammar **G**
where **N = { S , B }**, **Σ = { a , b , c }**, **S** is the start symbol,
and **P** consists of the following production rules:

1. **S** → **aBSc**
2. **S** → **abc**
3. **Ba** → **aB**
4. **Bb** → **bb**

This grammar defines the language **L(G) =** $\{a^n b^n c^n \mid n \geq 1\}$
where a^n denotes a string of **n** consecutive **a**'s.

Thus, the language is the set of strings that consist of 1 or more **a**'s,
followed by the same number of **b**'s, followed by the same number of **c**'s.

https://en.wikipedia.org/wiki/Formal_language

Grammar Examples (2)

$$\boldsymbol{S} \underset{2}{\Rightarrow} \boldsymbol{abc}$$

$$\begin{aligned} \boldsymbol{S} &\underset{1}{\Rightarrow} \boldsymbol{aBSc} \\ &\underset{2}{\Rightarrow} aB\boldsymbol{abc}c \\ &\underset{3}{\Rightarrow} a\boldsymbol{aB}bcc \\ &\underset{4}{\Rightarrow} aa\boldsymbol{bb}cc \end{aligned}$$

$$\begin{aligned} \boldsymbol{S} &\underset{1}{\Rightarrow} \boldsymbol{aBSc} \underset{1}{\Rightarrow} aB\boldsymbol{aBSc}c \\ &\underset{2}{\Rightarrow} aBaB\boldsymbol{abc}cc \\ &\underset{3}{\Rightarrow} a\boldsymbol{aB}Babccc \underset{3}{\Rightarrow} aaB\boldsymbol{aB}bccc \underset{3}{\Rightarrow} aa\boldsymbol{aB}Bbccc \\ &\underset{4}{\Rightarrow} aaaB\boldsymbol{bb}ccc \underset{4}{\Rightarrow} aaa\boldsymbol{bb}bccc \end{aligned}$$

1. **S → aBSc**
2. **S → abc**
3. **Ba → aB**
4. **Bb → bb**

https://en.wikipedia.org/wiki/Formal_language

Context Free Grammars

Context-free grammars are those grammars in which the left-hand side of each **production rule** consists of only a single **nonterminal symbol**.

This restriction is non-trivial; not all languages can be generated by context-free grammars.

Those that can are called **context-free languages**.

$$\mathbf{S \rightarrow aSb}$$
$$\mathbf{S \rightarrow ba}$$

https://en.wikipedia.org/wiki/Terminal_and_nonterminal_symbols

Context Free Grammar Examples

The language **L(G) = {$a^n b^n c^n$ | n ≥ 1 }** is <u>not</u> a **context-free language**
the grammar **G**
where **N = { S , B }**, **Σ = { a , b , c }**, **S** is the start symbol,
and **P** consists of the following production rules:
1. **S** → **aBSc**
2. **S** → **abc**
3. **Ba** → **aB**
4. **Bb** → **bb**

The language **{$a^n b^n$ | n ≥ 1 }** is **context-free**
(at least 1 **a** followed by the same number of **b**)
the grammar G2 with **N = { S }**, **Σ = { a , b }**, **S** the start symbol,
and **P** the following production rules:
1. **S** → **a S b**
2. **S** → **a b**

https://en.wikipedia.org/wiki/Formal_language

Regular Expression Examples

.at	matches any three-character string ending with "at", including "hat", "cat", and "bat".
[hc]at	matches "hat" and "cat".
[^b]at	matches all strings matched by .at except "bat".
[^hc]at	matches all strings matched by .at other than "hat" and "cat".
^[hc]at	matches "hat" and "cat", but only at the beginning of the string or line.
[hc]at$	matches "hat" and "cat", but only at the end of the string or line.
\[.\]	matches any single character surrounded by "[" and "]" since the brackets are escaped, for example: "[a]" and "[b]".
s.*	matches s followed by zero or more characters, for example: "s" and "saw" and "seed".

[hc]?at	matches "at", "hat", and "cat".
[hc]*at	matches "at", "hat", "cat", "hhat", "chat", "hcat", "cchchat", ...
[hc]+at	matches "hat", "cat", "hhat", "chat", "hcat", "cchchat",..., but not "at".
cat|dog	matches "cat" or "dog".

https://en.wikipedia.org/wiki/Regular_expression

Chomsky's four types of grammars

Grammar	Languages	Automaton	Production rules (constraints)*
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$ (no restrictions)
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite state automaton	$A \rightarrow \mathrm{a}$ and $A \rightarrow \mathrm{a}B$

* Meaning of symbols:

a = terminal
α = string of terminals, non-terminals, or empty
β = string of terminals, non-terminals, or empty
γ = string of terminals, non-terminals, never empty
A = non-terminal
B = non-terminal

https://en.wikipedia.org/wiki/Chomsky_hierarchy

Type-0 grammars

Unrestricted grammar

Type-0 grammars include all **formal grammars**.

They generate exactly all languages
that can be recognized by a **Turing machine**.

These languages are also known
as the **recursively enumerable** or
Turing-recognizable languages.

Note that this is different from the **recursive languages**,
which can be decided by an **always-halting Turing machine**.

https://en.wikipedia.org/wiki/Regular_expression

Type-0 grammars

Context-sensitive grammar

Type-1 grammars generate the **context-sensitive languages**.

These grammars have rules of the form $\alpha\ A\ \beta \rightarrow \alpha\ \gamma\ \beta$
with A a **nonterminal** and α, β, and γ strings of **terminals**
and/or **nonterminals**.

The strings α and β may be empty, but γ must be nonempty.

The rule $S \rightarrow \epsilon$ is allowed
if S does not appear on the right side of any rule.

The languages described by these grammars are exactly
all languages that can be recognized by a **linear bounded automaton**
(a **nondeterministic** Turing machine
whose tape is bounded by a constant times the length of the input.)

https://en.wikipedia.org/wiki/Regular_expression

Type-2 grammars

Context-free grammar

Type-2 grammars generate the context-free languages.

These are defined by rules of the form $A \to \gamma$
with A being a nonterminal and γ being a string of terminals and/or nonterminals.

These languages are exactly all languages
that can be recognized by a **non-deterministic pushdown automaton.**

Context-free languages—or rather its subset of deterministic context-free language—are the theoretical basis for the phrase structure of most **programming languages**, though their syntax also includes context-sensitive name resolution due to declarations and scope.

Often a subset of grammars is used to make parsing easier,
such as by an LL parser.

https://en.wikipedia.org/wiki/Regular_expression

Type-3 grammars (1)

Regular grammar

Type-3 grammars generate the regular languages.

restricts its rules to a single **nonterminal** on the **left**-hand side

a **right**-hand side consisting of a single **terminal**,
possibly followed by a **single nonterminal** (**right regular**).

the **right**-hand side consisting of a single **terminal**,
possibly preceded by a single **nonterminal** (**left regular**).

https://en.wikipedia.org/wiki/Regular_expression

Type-3 grammars (1)

Right regular and left regular generate the same languages.

However, if left-regular rules and right-regular rules are combined, the language need no longer be regular.

The rule $S \rightarrow \epsilon$ is also allowed here if S does not appear on the right side of any rule.

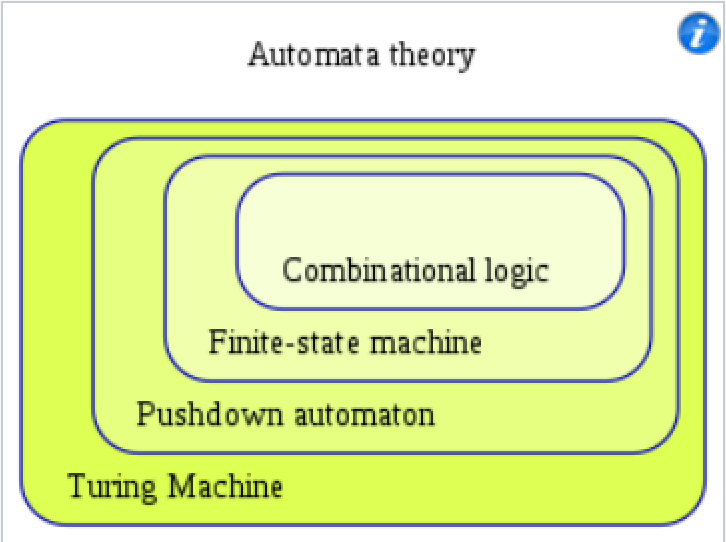
These languages are exactly all languages that can be decided by a **finite state automaton**.

Additionally, this family of formal languages can be obtained by **regular expressions**.

Regular languages are commonly used
to define search patterns and
the lexical structure of programming languages.

https://en.wikipedia.org/wiki/Regular_expression

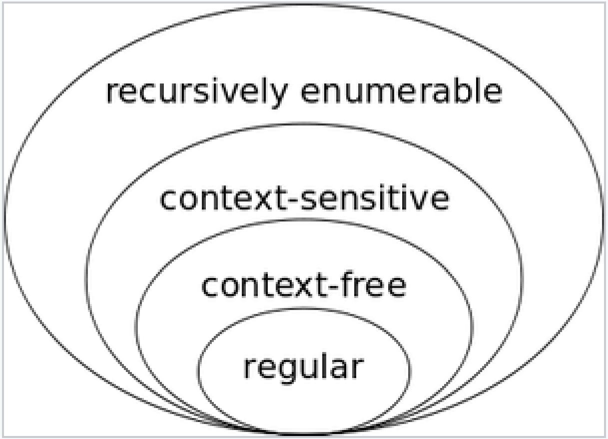
Class of Automata



Classes of automata
(Clicking on each layer will take you to an article on that subject)

https://en.wikipedia.org/wiki/Automata_theory

Chomsky Hierarchy



Regular language in classes of Chomsky hierarchy.

https://en.wikipedia.org/wiki/Regular_language

Class of Automata

Finite State Machine (FSM)	Regular Language
Pushdown Automaton (PDA)	Context-Free Language
Turing Machine	Recursively Enumerable Language

https://en.wikipedia.org/wiki/Automata_theory

Regular Language

a **regular language** (a **rational language**) is
a formal language that can be expressed
using a **regular expression**,
in the strict sense

Alternatively, a regular language can be defined
as a language recognized by a **finite automaton**.

The equivalence of **regular expressions** and
finite automata is known as **Kleene's theorem**.

Regular languages are very useful
in input parsing and programming language design.

https://en.wikipedia.org/wiki/Regular_language

Regular Language – Formal Definition

The collection of **regular languages** over an **alphabet** Σ
is defined recursively as follows:

The **empty language** ∅, and the **empty string language** {ε}
are **regular languages**.

For each a ∈ Σ (a belongs to Σ),
the **singleton language** {a} is a **regular language**.

If A and B are **regular languages**,
then A ∪ B (**union**), A • B (**concatenation**),
and A* (**Kleene star**) are **regular languages**.

No other languages over Σ are **regular**.

See regular expression for its syntax and semantics.
Note that the above cases are in effect the defining
rules of regular expression.

https://en.wikipedia.org/wiki/Regular_language

Equivalent Formalism

1. it is the language of a **regular expression** (by the above definition)
2. it is the language accepted by a **nondeterministic finite automaton** (**NFA**)
3. it is the language accepted by a **deterministic finite automaton** (**DFA**)
4. it can be generated by a **regular grammar**
5. it is the language accepted by an **alternating finite automaton**
6. it can be generated by a **prefix grammar**
7. it can be accepted by a read-only **Turing machine**

https://en.wikipedia.org/wiki/Regular_language

Regular Language Example

All **finite** languages are **regular**;

in particular the **empty** string language {ε} = Ø* is **regular**.

Other typical examples include the language
consisting of **all strings** over the **alphabet** {a, b}
which contain an even number of a’s, or
the language consisting of all strings of the form:
several as followed by several b’s.

A simple example of a language
that is **not regular** is the set of strings $\{ a^n b^n \mid n \geq 0 \}$.

Intuitively, it <u>cannot</u> be recognized with a **finite automaton**,
since a **finite automaton** has **finite memory**
and it cannot remember the exact number of a's.

https://en.wikipedia.org/wiki/Regular_language

References

[1] http://en.wikipedia.org/
[2]