Formal language

From Wikipedia, the free encyclopedia

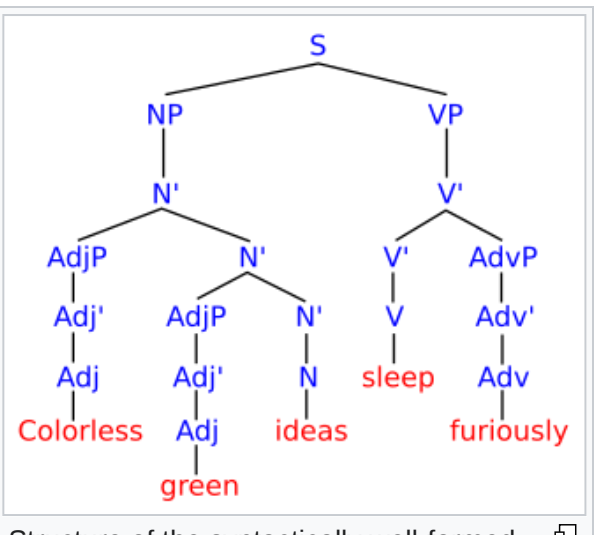
This article is about a technical term in mathematics and computer science. For any formal type of language usage, see Literary language. For studies about natural languages, see Formal semantics (natural language).

In logic, mathematics, computer science, and linguistics, a **formal language** is a set of strings whose symbols are taken from a set called "alphabet".

The alphabet of a formal language consists of symbols that concatenate into strings (also called "words").[1] Words that belong to a particular formal language are sometimes called *well-formed words*. A formal language is often defined by means of a formal grammar such as a regular grammar or context-free grammar.

In computer science, formal languages are used, among others, as the basis for defining the grammar of programming languages and formalized versions of subsets of natural languages, in which the words of the language represent concepts that are associated with meanings or semantics. 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. 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.

The field of **formal language theory** studies primarily the purely syntactic 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.



Structure of the syntactically well-formed, although thoroughly nonsensical, English sentence, *"Colorless green ideas sleep furiously"* (historical example from Chomsky 1957)

History [edit]

In the 17th century, Gottfried Leibniz imagined and described the characteristica universalis, a universal and formal language which utilised pictographs. Later, Carl Friedrich Gauss investigated the problem of Gauss codes.[2]

Gottlob Frege attempted to realize Leibniz's ideas, through a notational system first outlined in *Begriffsschrift* (1879) and more fully developed in his 2-volume Grundgesetze der Arithmetik (1893/1903).[3] This described a "formal language of pure language."[4]

In the first half of the 20th century, several developments were made with relevance to formal languages. Axel Thue published four papers relating to words and language between 1906 and 1914. The last of these introduced what Emil Post later termed 'Thue Systems', and gave an early example of an undecidable problem.[5] Post would later use this paper as the basis for a 1947 proof "that the word problem for semigroups was recursively insoluble",[6] and later devised the canonical system for the creation of formal languages.

In 1907, Leonardo Torres Quevedo introduced a formal language for the description of mechanical drawings (mechanical devices), in Vienna. He published "Sobre un sistema de notaciones y símbolos destinados a facilitar la descripción de las máquinas" ("On a system of notations and symbols intended to facilitate the description of machines").[7] Heinz Zemanek rated it as an equivalent to a programming language for the numerical control of machine tools.[8]

Noam Chomsky devised an abstract representation of formal and natural languages, known as the Chomsky hierarchy.[9] In 1959 John Backus developed the Backus-Naur form to describe the syntax of a high level programming language, following his work in the creation of FORTRAN.[10] Peter Naur was the secretary/editor for the ALGOL60 Report in which he used Backus–Naur form to describe the Formal part of ALGOL60.

Words over an alphabet [edit]

Main article: Alphabet (formal languages)

An *alphabet*, in the context of formal languages, can be any set; its elements are called *letters*. An alphabet may contain an infinite number of elements;[note 1] however, most definitions in formal language theory specify alphabets with a finite number of elements, and many results apply only to them. It often makes sense to use an alphabet in the usual sense of the word, or more generally any finite character encoding such as ASCII or Unicode.

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 it is composed of. For any alphabet, there is only one word of length 0, the *empty word*, which is often denoted by e, ε, λ or even Λ. By concatenation one can combine two words to form a new word, whose length is the sum of the lengths of the original words. The result of concatenating a word with the empty word is the original word.

In some applications, especially in logic, the alphabet is also known as the *vocabulary* and words are known as *formulas* or *sentences*; this breaks the letter/word metaphor and replaces it by a word/sentence metaphor.

Definition [edit]

Given a non-empty set Σ, a **formal language L over Σ** is a subset of Σ^*, which is the set of all possible finite-length words over Σ. We call the set Σ **the alphabet of L**. On the other hand, given a formal language L over Σ, a word $w \in \Sigma^*$ is *well-formed* if $w \in L$. Similarly, an expression $E \subseteq \Sigma^*$ is *well-formed* if $E \subseteq L$. Sometimes, a formal language L over Σ has a set of clear rules and constraints for the creation of all possible well-formed words from Σ^*.

In computer science and mathematics, which do not usually deal with natural languages, the adjective "formal" is often omitted as redundant. On the other hand, we can just say "a formal language L" when its alphabet Σ is clear in the context.

While formal language theory usually concerns itself with formal languages that are described by some syntactic rules, the actual definition of the concept "formal language" is only as above: a (possibly infinite) set of finite-length strings composed from a given alphabet, no more and no less. In practice, there are many languages that can be described by rules, such as regular languages or context-free languages. The notion of a formal grammar may be closer to the intuitive concept of a "language", one described by syntactic rules. By an abuse of the definition, a particular formal language is often thought of as being accompanied with a formal grammar that describes it.

Examples [edit]

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 that does not contain "+" or "=" and does not start with "0" is in L.
- 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.

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, "+" means addition, "23+4=555" is false, etc.

Constructions [edit]

For finite languages, one can explicitly enumerate all well-formed words. For example, we can describe a language L as just L = {a, b, ab, cba}. The degenerate case of this construction is the **empty language**, which contains no words at all (L = ∅).

However, even over a finite (non-empty) alphabet such as Σ = {a, b} there are an infinite number of finite-length words that can potentially be expressed: "a", "abb", "ababba", "aaababbbbaab", ... Therefore, formal languages are typically infinite, and describing an infinite formal language is not as simple as writing *L* = {a, b, ab, cba}. Here are some examples of formal languages:

- $L = \Sigma^*$, 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}.

Language-specification formalisms [edit]

Formal languages are used as tools in multiple disciplines. However, formal language theory rarely concerns itself with particular languages (except as examples), but is mainly concerned with the study of various types of formalisms to describe languages. 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 (an algorithm that asks a sequence of related YES/NO questions) produces the answer YES.

Typical questions asked about such formalisms include:

- What is their expressive power? (Can formalism *X* describe every language that formalism *Y* can describe? Can it describe other languages?)
- What is their recognizability? (How difficult is it to decide whether a given word belongs to a language described by formalism *X*?)
- What is their comparability? (How difficult is it to decide whether two languages, one described in formalism *X* and one in formalism *Y*, or in *X* again, are actually the same language?).

Surprisingly often, the answer to these decision problems is "it cannot be done at all", or "it is extremely expensive" (with a characterization of how expensive). Therefore, formal language theory is a major application area of computability theory and complexity theory. Formal languages may be classified in the Chomsky hierarchy based on the expressive power of their generative grammar as well as the complexity of their recognizing automaton. Context-free grammars and regular grammars provide a good compromise between expressivity and ease of parsing, and are widely used in practical applications.

Operations on languages [edit]

Certain operations on languages are common. This includes the standard set operations, such as union, intersection, and complement. Another class of operation is the element-wise application of string operations.

Examples: suppose L_1 and L_2 are languages over some common alphabet Σ.

- The *concatenation* $L_1 \cdot L_2$ consists of all strings of the form vw where v is a string from L_1 and w is a string from L_2.
- The *intersection* $L_1 \cap L_2$ of L_1 and L_2 consists of all strings that are contained in both languages
- The *complement* $\neg L_1$ of L_1 with respect to Σ consists of all strings over Σ that are not in L_1.
- The Kleene star: the language consisting of all words that are concatenations of zero or more words in the original language;
- *Reversal*:
  - Let ε be the empty word, then $\varepsilon^R = \varepsilon$, and
  - for each non-empty word $w = \sigma_1 \cdots \sigma_n$ (where $\sigma_1, \ldots, \sigma_n$ are elements of some alphabet), let $w^R = \sigma_n \cdots \sigma_1$,
  - then for a formal language L, $L^R = \{w^R \mid w \in L\}$.
- String homomorphism

Such string operations are used to investigate closure properties of classes of languages. A class of languages is closed under a particular operation when the operation, applied to languages in the class, always produces a language in the same class again. For instance, the context-free languages are known to be closed under union, concatenation, and intersection with regular languages, but not closed under intersection or complement. The theory of trios and abstract families of languages studies the most common closure properties of language families in their own right.[11]

Closure properties of language families (L_1 Op L_2 where both L_1 and L_2 are in the language family given by the column). After Hopcroft and Ullman.

Operation		Regular	DCFL	CFL	IND	CSL	recursive	RE
Union	$L_1 \cup L_2 = \{w \mid w \in L_1 \lor w \in L_2\}$	Yes	No	Yes	Yes	Yes	Yes	Yes
Intersection	$L_1 \cap L_2 = \{w \mid w \in L_1 \land w \in L_2\}$	Yes	No	No	No	Yes	Yes	Yes
Complement	$\neg L_1 = \{w \mid w \notin L_1\}$	Yes	Yes	No	No	Yes	Yes	No
Concatenation	$L_1 \cdot L_2 = \{wz \mid w \in L_1 \land z \in L_2\}$	Yes	No	Yes	Yes	Yes	Yes	Yes
Kleene star	$L_1^* = \{\varepsilon\} \cup \{wz \mid w \in L_1 \land z \in L_1^*\}$	Yes	No	Yes	Yes	Yes	Yes	Yes
(String) homomorphism h	$h(L_1) = \{h(w) \mid w \in L_1\}$	Yes	No	Yes	Yes	No	No	Yes
ε-free (string) homomorphism h	$h(L_1) = \{h(w) \mid w \in L_1\}$	Yes	No	Yes	Yes	Yes	Yes	Yes
Substitution φ	$\varphi(L_1) = \bigcup_{\sigma_1 \cdots \sigma_n \in L_1} \varphi(\sigma_1) \cdot \ldots \cdot \varphi(\sigma_n)$	Yes	No	Yes	Yes	Yes	No	Yes
Inverse homomorphism h^{-1}	$h^{-1}(L_1) = \bigcup_{w \in L_1} h^{-1}(w)$	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Reverse	$L^R = \{w^R \mid w \in L\}$	Yes	No	Yes	Yes	Yes	Yes	Yes
Intersection with a regular language R	$L \cap R = \{w \mid w \in L \land w \in R\}$	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Applications [edit]

Programming languages [edit]

Main articles: Syntax (programming languages) and Compiler-compiler

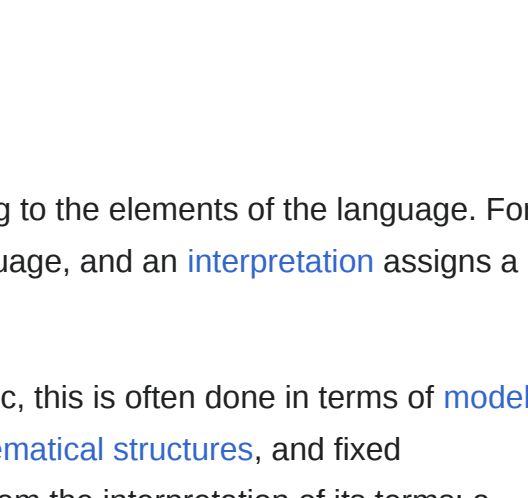
A compiler usually has two distinct components. A lexical analyzer, sometimes generated by a tool like `lex`, identifies the tokens of the programming language grammar, e.g. identifiers or keywords, numeric and string literals, punctuation and operator symbols, which are themselves specified by a simpler formal language, usually by means of regular expressions. At the most basic conceptual level, a parser, sometimes generated by a parser generator like `yacc`, attempts to decide if the source program is syntactically valid, that is if it is well formed with respect to the programming language grammar for which the compiler was built.

Of course, compilers do more than just parse the source code – they usually translate it into some executable format. Because of this, a parser usually outputs more than a yes/no answer, typically an abstract syntax tree. This is used by subsequent stages of the compiler to eventually generate an executable containing machine code that runs directly on the hardware, or some intermediate code that requires a virtual machine to execute.

Formal theories, systems, and proofs [edit]

Main articles: Theory (mathematical logic) and Formal system

In mathematical logic, a *formal theory* is a set of sentences expressed in a formal language.



This diagram shows the syntactic divisions within a formal system. Strings of symbols may be broadly divided into nonsense and well-formed formulas. The set of well-formed formulas is divided into theorems and non-theorems.

A *formal system* (also called a *logical calculus*, or a *logical system*) consists of a formal language together with a deductive apparatus (also called a *deductive system*). The deductive apparatus may consist of a set of transformation rules, which may be interpreted as valid rules of inference, or a set of axioms, or have both. A formal system is used to derive one expression from one or more other expressions. Although a formal language can be identified with its formulas, a formal system cannot be likewise identified by its theorems. Two formal systems $\mathcal{FS}$ and $\mathcal{FS}'$ may have all the same theorems and yet differ in some significant proof-theoretic way (a formula A may be a syntactic consequence of a formula B in one but not another for instance).

A *formal proof* or *derivation* is a finite sequence of well-formed formulas (which may be interpreted as sentences, or propositions) each of which is an axiom or follows from the preceding formulas in the sequence by a rule of inference. The last sentence in the sequence is a theorem of a formal system. Formal proofs are useful because their theorems can be interpreted as true propositions.

Interpretations and models [edit]

Main articles: Formal semantics (logic), Interpretation (logic), and Model theory

Formal languages are entirely syntactic in nature, but may be given semantics that give meaning to the elements of the language. For instance, in mathematical logic, the set of possible formulas of a particular logic is a formal language, and an interpretation assigns a meaning to each of the formulas—usually, a truth value.

The study of interpretations of formal languages is called formal semantics. In mathematical logic, this is often done in terms of model theory. In model theory, the terms that occur in a formula are interpreted as objects within mathematical structures, and fixed compositional interpretation rules determine how the truth value of the formula can be derived from the interpretation of its terms; a *model* for a formula is an interpretation of terms such that the formula becomes true.

See also [edit]

- Combinatorics on words
- Formal method
- Free monoid
- Grammar framework
- Mathematical notation
- String (computer science)

Notes [edit]

1. ^ For example, first-order logic is often expressed using an alphabet that, besides symbols such as ∧, ¬, ∀ and parentheses, contains infinitely many elements x_0, x_1, x_2, … that play the role of variables.

References [edit]

Citations [edit]

1. ^ See e.g. Reghizzi, Stefano Crespi (2009). *Formal Languages and Compilation*. Texts in Computer Science. Springer. p. 8. Bibcode:2009flc..book.....C. ISBN 9781848820500. "An alphabet is a finite set"
2. ^ "In the prehistory of formal language theory: Gauss Languages". January 1992. Retrieved 30 April 2021.
3. ^ "Gottlob Frege". 5 December 2019. Retrieved 30 April 2021.
4. ^ Martin Davis (1995). "Influences of Mathematical Logic on Computer Science". In Rolf Herken (ed.). *The universal Turing machine: a half-century survey*. Springer. p. 290. ISBN 978-3-211-82637-9.
5. ^ "Thue's 1914 paper: a translation" (PDF). 28 August 2013. Archived (PDF) from the original on 30 April 2021. Retrieved 30 April 2021.
6. ^ "Emil Leon Post". September 2001. Retrieved 30 April 2021.
7. ^ Torres Quevedo, Leonardo. Sobre un sistema de notaciones y símbolos destinados a facilitar la descripción de las máquinas, (pdf), pp. 25–30, Revista de Obras Públicas, 17 January 1907.
8. ^ Bruderer, Herbert (2021). "The Global Evolution of Computer Technology". *Milestones in Analog and Digital Computing*. Springer. p. 1212. ISBN 978-3030409739.
9. ^ Jager, Gerhard; Rogers, James (19 July 2012). "Formal language theory: refining the Chomsky hierarchy". *Philosophical Transactions of the Royal Society B*. **367** (1598): 1956–1970. doi:10.1098/rstb.2012.0077. PMC 3367686. PMID 22688632.
10. ^ "John Warner Backus". February 2016. Retrieved 30 April 2021.
11. ^ Hopcroft & Ullman (1979), Chapter 11: Closure properties of families of languages.

Sources [edit]

Works cited

- Hopcroft, John E.; Ullman, Jeffrey D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Reading, Massachusetts: Addison-Wesley Publishing. ISBN 81-7808-347-7.

General references

- A. G. Hamilton, *Logic for Mathematicians*, Cambridge University Press, 1978, ISBN 0-521-21838-1.
- Seymour Ginsburg, *Algebraic and automata theoretic properties of formal languages*, North-Holland, 1975, ISBN 0-7204-2506-9.
- Michael A. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, 1978.
- Rautenberg, Wolfgang (2010). *A Concise Introduction to Mathematical Logic* (3rd ed.). New York: Springer Science+Business Media. doi:10.1007/978-1-4419-1221-3. ISBN 978-1-4419-1220-6.
- Grzegorz Rozenberg, Arto Salomaa, *Handbook of Formal Languages: Volume I-III*, Springer, 1997, ISBN 3-540-61486-9.
- Patrick Suppes, *Introduction to Logic*, D. Van Nostrand, 1957, ISBN 0-442-08072-7.

External links [edit]

- "Formal language", *Encyclopedia of Mathematics*, EMS Press, 2001 [1994]
- University of Maryland, Formal Language Definitions
- James Power, "Notes on Formal Language Theory and Parsing" Archived 21 November 2007 at the Wayback Machine, 29 November 2002.
- Drafts of some chapters in the "Handbook of Formal Language Theory", Vol. 1–3, G. Rozenberg and A. Salomaa (eds.), Springer Verlag, (1997):
  - Alexandru Mateescu and Arto Salomaa, "Preface" in Vol.1, pp. v–viii, and "Formal Languages: An Introduction and a Synopsis", Chapter 1 in Vol. 1, pp. 1–39
  - Sheng Yu, "Regular Languages", Chapter 2 in Vol. 1
  - Jean-Michel Autebert, Jean Berstel, Luc Boasson, "Context-Free Languages and Push-Down Automata", Chapter 3 in Vol. 1
  - Christian Choffrut and Juhani Karhumäki, "Combinatorics of Words", Chapter 6 in Vol. 1
  - Tero Harju and Juhani Karhumäki, "Morphisms", Chapter 7 in Vol. 1, pp. 439–510
  - Jean-Eric Pin, "Syntactic semigroups", Chapter 10 in Vol. 1, pp. 679–746
  - M. Crochemore and C. Hancart, "Automata for matching patterns", Chapter 9 in Vol. 2
  - Dora Giammarresi, Antonio Restivo, "Two-dimensional Languages", Chapter 4 in Vol. 3, pp. 215–267

V · T · E Automata theory: formal languages and formal grammars [show]
V · T · E Mathematical logic [show]
Authority control databases [show]
Categories: Formal languages | Theoretical computer science | Combinatorics on words | Mathematical linguistics