WIKIPEDIA

Formal grammar

In formal language theory, a **grammar** (when the context is not given, often called a **formal grammar** for clarity) is a set of production rules for strings in a formal language. The rules describe how to form strings from the language's alphabet that are valid according to the language's syntax. A grammar does not describe the meaning of the strings or what can be done with them in whatever context—only their form.

Formal language theory, the discipline that studies formal grammars and languages, is a branch of applied mathematics. Its applications are found in theoretical computer science, theoretical linguistics, formal semantics, mathematical logic, and other areas.

A formal grammar is a set of rules for rewriting strings, along with a "start symbol" from which rewriting starts. Therefore, a grammar is usually thought of as a language generator. However, it can also sometimes be used as the basis for a "recognizer"—a function in computing that determines whether a given string belongs to the language or is grammatically incorrect. To describe such recognizers, formal language theory uses separate formalisms, known as automata theory. One of the interesting results of automata theory is that it is not possible to design a recognizer for certain formal languages.[1] Parsing is the process of recognizing an utterance (a string in natural languages) by breaking it down to a set of symbols and analyzing each one against the grammar of the language. Most languages have the meanings of their utterances structured according to their syntax—a practice known as compositional semantics. As a result, the first step to describing the meaning of an utterance in language is to break it down part by part and look at its analyzed form (known as its parse tree in computer science, and as its deep structure in generative grammar).

Contents

History

Pāṇini's treatise *Astadyayi* gives formal production rules and definitions to describe the formal grammar of Sanskrit.[2]

Introductory example

A grammar mainly consists of a set of rules for transforming strings. (If it *only* consisted of these rules, it would be a semi-Thue system.) To generate a string in the language, one begins with a string consisting of only a single *start symbol*. The *production rules* are then applied in any order, until a string that contains neither the start symbol nor designated *nonterminal symbols* is produced. A production rule is applied to a string by replacing one occurrence of the production rule's left-hand side in the string by that production rule's right-hand side (*cf.* the operation of the theoretical Turing machine). The language formed by the grammar consists of all distinct strings that can be generated in this manner. Any particular sequence of production rules on the start symbol yields a distinct string in the language. If there are essentially different ways of generating the same single string, the grammar is said to be ambiguous.

For example, assume the alphabet consists of *a* and *b*, the start symbol is *S*, and we have the following production rules:

1. $S \rightarrow aSb$
2. $S \rightarrow ba$

then we start with *S*, and can choose a rule to apply to it. If we choose rule 1, we obtain the string *aSb*. If we then choose rule 1 again, we replace *S* with *aSb* and obtain the string *aaSbb*. If we now choose rule 2, we replace *S* with *ba* and obtain the string *aababb*, and are done. We can write this series of choices more briefly, using symbols: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aababb$. The language of the grammar is then the infinite set $\{a^n bab^n \mid n \geq 0\} = \{ba, abab, aababb, aaababbb, \dots\}$, where a^k is a repeated times (and in particular represents the number of times production rule 1 has been applied).

Formal definition

The syntax of grammars

In the classic formalization of generative grammars first proposed by Noam Chomsky in the 1950s,[3][4] 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 of *terminal symbols* that is disjoint from *N*.
- A finite set *P* of *production rules*, each rule of the form

  (\Sigma \cup N)^{*}N(\Sigma

  where is the Kleene star operator and denotes set union. That is, each production rule maps from one string of symbols to another, where the first string (the "head") contains an arbitrary number of symbols provided at least one of them is a nonterminal. In the case that 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 (often , *e* or) in order to avoid confusion.

- A distinguished symbol S\ir that is the *start symbol*, also called the *sentence symbol*.

A grammar is formally defined as the tuple (N,\Sigma. Such a formal grammar is often called a rewriting system or a phrase structure grammar in the literature.[5][6]

The semantics of grammars

The operation of a grammar can be defined in terms of relations on strings:

- Given a grammar G=(N,\Sigma , the binary relation (pronounced as "G derives in one step") on strings in (\Sigm is defined by:

  {\displaystyle x{\underset {G}{\Rightarrow }}y\iff\exists u,v,p,q\in (\Sigma

- the relation (pronounced as *G derives in zero or more steps*) is defined as the reflexive transitive closure of
- a *sentential form* is a member of (\Sigm that can be derived in a finite number of steps from the start symbol ; that is, a sentential form is a member of \left\{w\in (\Sigma \cup N)^{*}\mid S{\overset . A sentential form that contains no nonterminal symbols (i.e. is a member of) is called a *sentence*.[7]
- the *language* of , denoted as {\b, is defined as all those sentences that can be derived in a finite number of steps from the start symbol ; that is, the set \left\{w\in \Sigma ^{*}\mid .

Note that the grammar G=(N,\Sigma is effectively the semi-Thue system (N\cup , rewriting strings in exactly the same way; the only difference is in that we distinguish specific *nonterminal* symbols, which must be rewritten in rewrite rules, and are only interested in rewritings from the designated start symbol to strings without nonterminal symbols.

Example

For these examples, formal languages are specified using set-builder notation.

Consider the grammar where N=\left\ , \Sigma , is the start symbol, and consists of the following production rules:

1. S\rightarr
2. S\right
3. Ba\right
4. Bb\rig

This grammar defines the language {\displaystyle L(G)=\left\ where 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 's, followed by the same number of 's.

Some examples of the derivation of strings in L(are:

- {\bol {S}}
- {\begin{ali {\boldsyml {S}}& {\underset {1} {\Rightarro
- {\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

(Note on notation: P{\ {i} reads "String P generates string Q by means of production i", and the generated part is each time indicated in bold type.)

The Chomsky hierarchy

When Noam Chomsky first formalized generative grammars in 1956,[3] he classified them into types now known as the Chomsky hierarchy. The difference between these types is that they have increasingly strict production rules and can therefore express fewer formal languages. Two important types are *context-free grammars* (Type 2) and *regular grammars* (Type 3). The languages that can be described with such a grammar are called *context-free languages* and *regular languages*, respectively. Although much less powerful than unrestricted grammars (Type 0), which can in fact express any language that can be accepted by a Turing machine, these two restricted types of grammars are most often used because parsers for them can be efficiently implemented.[8] For example, all regular languages can be recognized by a finite state machine, and for useful subsets of context-free grammars there are well-known algorithms to generate efficient LL parsers and LR parsers to recognize the corresponding languages those grammars generate.

Context-free grammars

A *context-free grammar* is a grammar 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*.

The language {\displaystyle L(G)=\left\ defined above is not a context-free language, and this can be strictly proven using the pumping lemma for context-free languages, but for example the language {\displaystyle (at least 1 a followed by the same number of 's) is context-free, as it can be defined by the grammar with N=\lef, \Sigma , the start symbol, and the following production rules:

1. $S \rightarrow aSb$
2. S\rigl

A context-free language can be recognized in O(i time (*see* Big O notation) by an algorithm such as Earley's algorithm. That is, for every context-free language, a machine can be built that takes a string as input and determines i O(i time whether the string is a member of the language, where is the length of the string.[9] Deterministic context-free languages is a subset of context-free languages that can be recognized in linear time.[10] There exist various algorithms that target either this set of languages or some subset of it.

Regular grammars

In regular grammars, the left hand side is again only a single nonterminal symbol, but now the right-hand side is also restricted. The right side may be the empty string, or a single terminal symbol, or a single terminal symbol followed by a nonterminal symbol, but nothing else. (Sometimes a broader definition is used: one can allow longer strings of terminals or single nonterminals without anything else, making languages easier to denote while still defining the same class of languages.)

The language {\displaystyle defined above is not regular, but the language {\displaystyle (at least 1 a followed by at least 1 , where the numbers may be different) is, as it can be defined by the grammar with N=\left\ , \Sigma , the start symbol, and the following production rules:

1. S\righ
2. A\righ
3. A\rigl
4. B\rigl
5. B\ri

All languages generated by a regular grammar can be recognized in O time by a finite state machine. Although, in practice, regular grammars are commonly expressed using regular expressions, some forms of regular expression used in practice do not strictly generate the regular languages and do not show linear recognitional performance due to those deviations.

Other forms of generative grammars

Many extensions and variations on Chomsky's original hierarchy of formal grammars have been developed, both by linguists and by computer scientists, usually either in order to increase their expressive power or in order to make them easier to analyze or parse. Some forms of grammars developed include:

- Tree-adjoining grammars increase the expressiveness of conventional generative grammars by allowing rewrite rules to operate on parse trees instead of just strings.[11]
- Affix grammars[12] and attribute grammars[13][14] allow rewrite rules to be augmented with semantic attributes and operations, useful both for increasing grammar expressiveness and for constructing practical language translation tools.

Recursive grammars

A recursive grammar is a grammar that contains production rules that are recursive. For example, a grammar for a context-free language is left-recursive if there exists a non-terminal symbol *A* that can be put through the production rules to produce a string with *A* as the leftmost symbol.[15] All types of grammars in the Chomsky hierarchy can be recursive.

Analytic grammars

Though there is a tremendous body of literature on parsing algorithms, most of these algorithms assume that the language to be parsed is initially *described* by means of a *generative* formal grammar, and that the goal is to transform this generative grammar into a working parser. Strictly speaking, a generative grammar does not in any way correspond to the algorithm used to parse a language, and various algorithms have different restrictions on the form of production rules that are considered well-formed.

An alternative approach is to formalize the language in terms of an analytic grammar in the first place, which more directly corresponds to the structure and semantics of a parser for the language. Examples of analytic grammar formalisms include the following:

- The Language Machine directly implements unrestricted analytic grammars. Substitution rules are used to transform an input to produce outputs and behaviour. The system can also produce the lm-diagram, which shows what happens when the rules of an unrestricted analytic grammar are being applied.
- Top-down parsing language (TDPL): a highly minimalist analytic grammar formalism developed in the early 1970s to study the behavior of top-down parsers.[16]
- Link grammars: a form of analytic grammar designed for linguistics, which derives syntactic structure by examining the positional relationships between pairs of words.[17][18]
- Parsing expression grammars (PEGs): a more recent generalization of TDPL designed around the practical expressiveness needs of programming language and compiler writers.[19]

See also

- Abstract syntax tree
- Adaptive grammar
- Ambiguous grammar
- Backus–Naur form (BNF)
- Categorial grammar
- Concrete syntax tree
- Extended Backus–Naur form (EBNF)
- Grammar framework
- L-system
- Lojban
- Post canonical system
- Shape grammar
- Well-formed formula

References

1. Meduna, Alexander (2014), *Formal Languages and Computation: Models and Their Applications* (https://books.google.com/books?id=KJ-NAgAAQBAJ&pg=PA233), CRC Press, p. 233, ISBN 9781466513457. For more on this subject, see undecidable problem

2. "Panini biography" (http://www-history.mcs.st-andrews.ac.uk/Biographies/Panini.html) *www-history.mcs.st-andrews.ac.uk*.
3. Chomsky, Noam (Sep 1956). "Three models for the description of language" (http://www.chomsky.info/articles/195609--.pdf) (PDF). *IRE Transactions on Information Theory* **2** (3): 113–124. doi:10.1109/TIT.1956.1056813 (https://doi.org/10.1109%2FTIT.1956.1056813). Archived from the original (http://ieeexplore.ieee.org/iel5/18/22738/01056813.pdf?isnumber=22738&prod=STD&arnumber=1056813&arnumber=1056813&arSt=+113&ared=+124&arAuthor=+Chomsky%2C+N.) (PDF) on 2013-10-18. Retrieved 2007-06-18.
4. Chomsky, Noam (1957). *Syntactic Structures*. The Hague: Mouton.
5. Ginsburg, Seymour (1975). *Algebraic and automata theoretic properties of formal languages*. North-Holland. pp. 8–9. ISBN 978-0-7204-2506-2.
6. Harrison, Michael A. (1978). *Introduction to Formal Language Theory*. Reading, Mass.: Addison-Wesley Publishing Company. p. 13. ISBN 978-0-201-02955-0.
7. Sentential Forms (http://www.seas.upenn.edu/~cit596/notes/dave/cfg7.html), Context-Free Grammars, David Matuszek
8. Grune, Dick & Jacobs, Ceriel H., *Parsing Techniques – A Practical Guide*, Ellis Horwood, England, 1990.
9. Earley, Jay, "An Efficient Context-Free Parsing Algorithm (http://ra.adm.cs.cmu.edu/anon/home/anon/usr/ftp/scan/CMU-CS-68-earley.pdf)," *Communications of the ACM*, Vol. 13 No. 2, pp. 94-102, February 1970.
10. Knuth, D. E. (July 1965). "On the translation of languages from left to right" (http://www.cs.dartmouth.edu/~mckeeman/cs48/mxcom/doc/knuth65.pdf) (PDF). *Information and Control* **8** (6): 607–639. doi:10.1016/S0019-9958(65)90426-2 (https://doi.org/10.1016%2FS0019-9958%2865%2990426-2). Retrieved 29 May 2011.
11. Joshi, Aravind K., *et al.*, "Tree Adjunct Grammars," *Journal of Computer Systems Science*, Vol. 10 No. 1, pp. 136-163, 1975.
12. Koster , Cornelis H. A., "Affix Grammars," in *ALGOL 68 Implementation*, North Holland Publishing Company, Amsterdam, p. 95-109, 1971.
13. Knuth, Donald E., "Semantics of Context-Free Languages (https://www.csee.umbc.edu/courses/331/fall16/01/resources/papers/Knuth67AG.pdf)," *Mathematical Systems Theory*, Vol. 2 No. 2, pp. 127-145, 1968.
14. Knuth, Donald E., "Semantics of Context-Free Languages (correction)," *Mathematical Systems Theory*, Vol. 5 No. 1, pp 95-96, 1971.
15. Notes on Formal Language Theory and Parsing (http://www.cs.may.ie/~jpower/Courses/parsing/parsing.pdf#search='indirect%20left%20recursion'), James Power, Department of Computer Science National University of Ireland, Maynooth Maynooth, Co. Kildare, Ireland. JPR02
16. Birman, Alexander, *The TMG Recognition Schema (http://bford.info/packrat/ref/birman70tmg.pdf)*, Doctoral thesis, Princeton University, Dept. of Electrical Engineering, February 1970.
17. Sleator, Daniel D. & Temperly, Davy, "Parsing English with a Link Grammar (https://arxiv.org/pdf/cmp-lg/9508004)," Technical Report CMU-CS-91-196, Carnegie Mellon University Computer Science, 1991.
18. Sleator, Daniel D. & Temperly, Davy, "Parsing English with a Link Grammar," *Third International Workshop on Parsing Technologies*, 1993. (Revised version of above report.)
19. Ford, Bryan, *Packrat Parsing: a Practical Linear-Time Algorithm with Backtracking (https://dspace.mit.edu/bitstream/handle/1721.1/87310/51972156-MIT.pdf;sequence=2)*, Master's thesis, Massachusetts Institute of Technology, Sept. 2002.

External links

- Yearly Formal Grammar conference

Retrieved from "https://en.wikipedia.org/w/index.php?title=Formal_grammar&oldid=864957481"