Ambiguous grammar

From Wikipedia, the free encyclopedia

In computer science, an **ambiguous grammar** is a context-free grammar for which there exists a string that can have more than one leftmost derivation, while an **unambiguous grammar** is a context-free grammar for which every valid string has a unique leftmost derivation. Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars. Any non-empty language admits an ambiguous grammar by taking an unambiguous grammar and introducing a duplicate rule or synonym (the only language without ambiguous grammars is the empty language). A language that only admits ambiguous grammars is called an inherently ambiguous language, and there are inherently ambiguous context-free languages. Deterministic context-free grammars are always unambiguous, and are an important subclass of unambiguous CFGs; there are non-deterministic unambiguous CFGs, however.

For real-world programming languages, the reference CFG is often ambiguous, due to issues such as the dangling else problem. If present, these ambiguities are generally resolved by adding precedence rules or other context-sensitive parsing rules, so the overall phrase grammar is unambiguous.

Contents

Examples

Trivial language

The simplest example is the following ambiguous grammar for the trivial language, which consists of only the empty string:

$A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$

…meaning that the empty string ϵ can be produced by either of two equivalent productions, and thus has two leftmost derivations.

Another ambiguous grammar for the trivial language is:

$A \rightarrow A \mid \varepsilon$

…meaning that a production can either be itself again, or the empty string. Thus the empty string has leftmost derivations of length 1, 2, 3, and indeed of any length, depending on how many times the rule $A \rightarrow A$ is used.

This language also has the unambiguous grammar, consisting of a single production rule:

$A \rightarrow \varepsilon$

…meaning that the unique production can only produce the empty string, which is the unique string in the language.

In the same way, any grammar for a non-empty language can be made ambiguous by adding duplicates.

Unary string

The regular language of unary strings of a given character, say `'a'` (the regular expression `a*`), has the unambiguous grammar:

$A \rightarrow aA \mid \varepsilon$

…but also has the ambiguous grammar:

$A \rightarrow aA \mid Aa \mid \varepsilon$

These correspond to producing a right-associative tree (for the unambiguous grammar) or allowing both left- and right- association. This is elaborated below.

Addition and subtraction

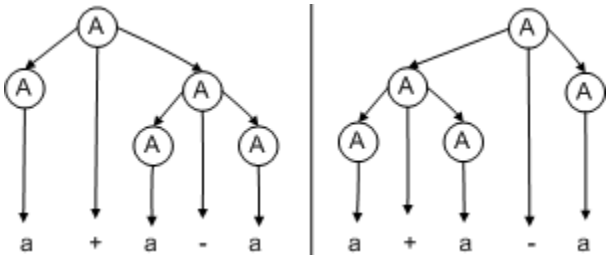
The context free grammar

$A \rightarrow A + A \mid A - A \mid a$

is ambiguous since there are two leftmost derivations for the string a + a + a:

$A \rightarrow A + A$	$A \rightarrow A + A$
$\rightarrow a + A$	$\rightarrow A + A + A$ (First A is replaced by A+A. Replacement of the second A would yield a similar derivation)
$\rightarrow a + A + A$	$\rightarrow a + A + A$
$\rightarrow a + a + A$	$\rightarrow a + a + A$
$\rightarrow a + a + a$	$\rightarrow a + a + a$

As another example, the grammar is ambiguous since there are two parse trees for the string a + a – a:



The language that it generates, however, is not inherently ambiguous; the following is a non-ambiguous grammar generating the same language:

$A \rightarrow A + a \mid A - a \mid a$

Dangling else

A common example of ambiguity in real-world programming languages is the dangling else problem. In many languages, the `else` in an If–then(–else) statement is optional, which results in nested conditionals being ambiguous, at least in terms of the CFG.

Concretely, in many languages one may write conditionals in two forms: the if-then form, and the if-then-else form – the else clause is optional:

In a grammar containing the rules

```
Statement = if Condition then Statement |
            if Condition then Statement else Statement |
            ...
Condition = ...
```

some ambiguous phrase structures can appear. The expression

```
if a then if b then s else s2
```

can be parsed as either

```
if a then (if b then s) else s2
```

or as

```
if a then (if b then s else s2)
```

depending on whether the `else` is associated with the first `if` or second `if`.

This is resolved in various ways in different languages. Sometimes the CFG is modified so that it is unambiguous, such as by requiring an `endif` statement or making `else` mandatory. In other cases the CFG is left ambiguous, but the ambiguity is resolved by making the overall phrase grammar context-sensitive, such as by associating an `else` with the nearest `if`. In this latter case the grammar is unambiguous, but the CF grammar is ambiguous.

Recognizing ambiguous grammars

The general decision problem of whether a grammar is ambiguous is undecidable because it can be shown that it is equivalent to the Post correspondence problem.[1] At least, there are tools implementing some semi-decision procedure for detecting ambiguity of context-free grammars.[2]

The efficiency of context-free grammar parsing is determined by the automaton that accepts it. Deterministic context-free grammars are accepted by deterministic pushdown automata and can be parsed in linear time, for example by the LR parser.[3] This is a subset of the context-free grammars which are accepted by the pushdown automaton and can be parsed in polynomial time, for example by the CYK algorithm. Unambiguous context-free grammars can be nondeterministic. For example, the language of even-length palindromes on the alphabet of 0 and 1 has the unambiguous context-free grammar $S \rightarrow 0S0 \mid 1S1 \mid \varepsilon$. An arbitrary string of this language cannot be parsed without reading all its letters first which means that a pushdown automaton has to try alternative state transitions to accommodate for the different possible lengths of a semi-parsed string.[4] Nevertheless, removing grammar ambiguity may produce a deterministic context-free grammar and thus allow for more efficient parsing. Compiler generators such as YACC include features for resolving some kinds of ambiguity, such as by using the precedence and associativity constraints.

Inherently ambiguous languages

Inherent ambiguity was proven with Parikh's theorem in 1961 by Rohit Parikh in an MIT research report.[5]

While some context-free languages (the set of strings that can be generated by a grammar) have both ambiguous and unambiguous grammars, there exist context-free languages for which no unambiguous context-free grammar can exist. An example of an inherently ambiguous language is the union of

$\{a^n b^m c^m d^n | n, m > 0\}$ with $\{a^n b^n c^m d^m | n, m > 0\}$. This set is context-free, since the union of two context-free languages is always context-free. But Hopcroft & Ullman (1979) give a proof that there is no way to unambiguously parse strings in the (non-context-free) subset $\{a^n b^n c^n d^n | n > 0\}$ which is the intersection of these two languages.[6]

See also

- GLR parser, a type of parser for ambiguous and nondeterministic grammars
- Chart parser, another type of parser for ambiguous grammars
- Semantic ambiguity

References

1. **^** Hopcroft, Motwani, Ullman (2001), Theorem 9.20, p.405-406
2. **^** Axelsson, Roland; Heljanko, Keijo; Lange, Martin (2008). "Analyzing Context-Free Grammars Using an Incremental SAT Solver". *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP'08), Reykjavik, Iceland*. Lecture Notes in Computer Science **5126**. Springer-Verlag. pp. 410–422.
3. **^** 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). *Information and Control* **8** (6): 607–639. doi:10.1016/S0019-9958(65)90426-2 (https://dx.doi.org/10.1016%2FS0019-9958%2865%2990426-2). Retrieved 29 May 2011.
4. **^** Hopcroft, John; Motwani, Rajeev; Ullman, Jeffrey (2001). *Introduction to automata theory, languages, and computation* (2nd ed.). Addison-Wesley. pp. 249–253.
5. **^** Parikh, Rohit (January 1961). *Language-generating devices*. Quarterly Progress Report, Research Laboratory of Electronics, MIT.
6. **^** p.99-103, Sect.4.7

- Gross, Maurice (September 1964). "Inherent ambiguity of minimal linear grammars". *Information and Control* (Information and Control) **7** (3): 366–368. doi:10.1016/S0019-9958(64)90422-X (https://dx.doi.org/10.1016%2FS0019-9958%2864%2990422-X).
- Michael, Harrison (1978). *Introduction to Formal Language Theory*. Addison-Wesley. ISBN 0201029553.
- Hopcroft, John E.; Ullman, Jeffrey D. (1979). *Introduction to Automata Theory, Languages, and Computation* (1st ed.). Addison-Wesley.
- Hopcroft, John; Mowani, Rajeev; Ullman, Jeffrey (2001). *Introduction to Automata Theory, Languages and Computation* (2nd ed.). Addison Wesley. p. 217.
- Brabrand, Claus; Giegerich, Robert; Møller, Anders (March 2010). "Analyzing Ambiguity of

Context-Free Grammars". *Science of Computer Programming* (Elsevier) **75** (3): 176–191. doi:10.1016/j.scico.2009.11.002 (https://dx.doi.org/10.1016%2Fj.scico.2009.11.002).

External links

- dk.brics.grammar (http://www.brics.dk/grammar) - a grammar ambiguity analyzer.
- CFGAnalyzer (http://www2.tcs.ifi.lmu.de/~mlange/cfganalyzer/index.html) - tool for analyzing context-free grammars with respect to language universality, ambiguity, and similar properties.