Formal Grammars

August 31, 2017

Brian A. Malloy

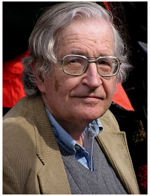
Formal grammar

From Wikipedia, the free encyclopedia

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.

1. Who Introduced Grammars?

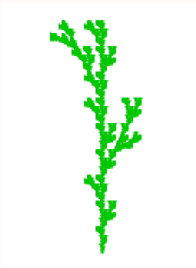
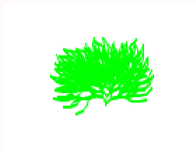
- In early 1950's, Chomsky attempted to specify the structure of natural language using math rules to specify the strings.
- He established the **Chomsky Hierarchy**, which we'll study later.
- His work influenced others to study properties of strings.



2. Why Study Grammars?

- Grammars enable programmers to automate many tasks that are tedious and error prone when performed manually.
- Programmers know that information can be represented as strings: e.g., numbers, name, pictures, sound, . . .
- One set of strings, computer languages, have become central to computer science.
- **The syntax of programming languages can be specified by a grammar.**

- Fractals and L-Systems can also be specified or generated with a grammar:


3. What's a Grammar?

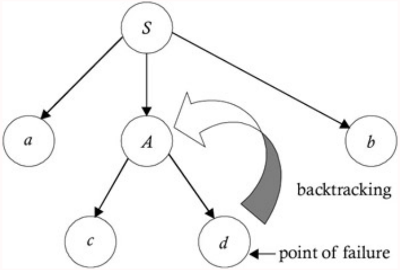
- Def: Formally, a grammar G is a four tuple (N, T, S, P) where N & T are disjount sets of symbols known as non-terminals and terminals, $S \in N$ is the start symbol, and P is a relation on $N \cup T$ of production rules.
- N: non-terminals are generally represented as cap letters, and do not appear in the language; they are used to derive sentences in the language.
- T: terminals are symbols in the language
- S is one of the non-terminals that indicates where to start when deriving a sentence in the language.
- P: rules used to derive a sentence.

4. Parsing

- parsing is the process of recognizing a string in the language.
- This is accomplished by breaking the string into symbols and analyzing each symbol against the grammar of the language.
- Most languages have their strings structured according to the syntax specified by the grammar.
- a parse tree is a step-by-step illustration of a derivation of a sentence using the grammar.

4.1. How it works

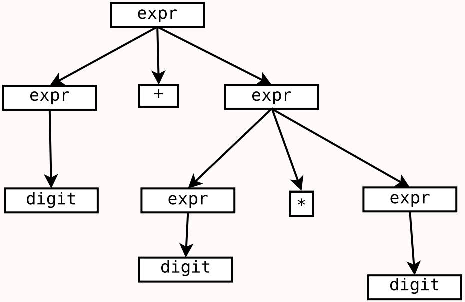
- Start with start symbol
- Try to regenerate the sentence by applying productions.
- Determine production by looking at next terminal in sentence.



4.2. Example: Expression Grammar

Derive 3 + 2 * 7:

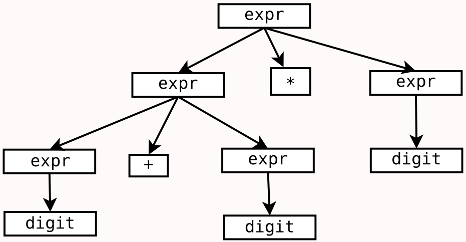
$$
\begin{array}{lll}
\text{expr} & : & \text{expr '+' expr} \\
 & | & \text{expr '*' expr} \\
 & | & \text{DIGIT}
\end{array}
$$



4.3. Example: Second Derivation

Derive $3 + 2 * 7$:

- Which tree is correct: Slide 4.2 or below?
- If there are two different parse trees for the same grammar, what does this mean?



5. Classes of Grammars: The Chomsky Hierarchy

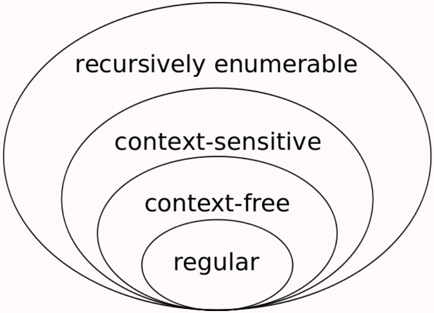
- A grammar defines or generates an infinite set of strings or symbols that we call language
- A grammar also enables the use of a computer to systematically model a language
- Chomsky's hierarchy establishes categories of grammars, with some categories that are more expressive than others.

5.1. The Hierarchy

Grammar	Language	Machine	Rules
Type 0	RE	Turing Machine	$\alpha \to \beta$
Type 1	CSG	LBA	$\alpha A \beta \to \alpha \gamma \beta$
Type 2	CFG	NPDA	$A \to \gamma$
Type 3	Regular	FSA	$A \to aB$

- We're interested in Regular and CFGs
- Regular grammars can specify tokens
- CFGs are more expressive than regular grammars.

5.2. Set Inclusion for expressivity



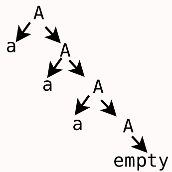
6. Regular Grammars

- A *right regular grammar* $G(N, T, S, P)$ has production rules P of the form:

$$\text{A} \quad : \text{aB} \mid \text{a} \mid \lambda, \text{ with } a \in T,\ A, B \in N$$

- It's *right regular* because the parse tree expands to the right. Derive: aaa

$$\text{A} \quad : \text{a A} \mid \lambda$$



Who Introduced . . .
Why Study Grammars?
What's a Grammar?
Parsing
Classes of . . .
Regular Grammars
Context Free . . .
Slide 13 of 21
Go Back
Full Screen
Quit

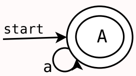
6.1. Use of Regular Grammars

- Can specify **terminals** (**tokens**) in language:
  - Reserved or keywords: **if**, **while**, . . .
  - Constants: 3.5, 75
  - Special symbols: (; :? . . .
- Used in editors
- Used in Unix commands (find, grep, etc.)

Who Introduced . . .
Why Study Grammars?
What's a Grammar?
Parsing
Classes of . . .
Regular Grammars
Context Free . . .
Slide 14 of 21
Go Back
Full Screen
Quit

6.2. DFA: Acceptor for Regular Grammars

- Def: a deterministic finite state machine accepts/rejects finite strings of symbols, and produces a unique computation of the automaton for each input string[1]
- If a grammar is regular, you can construct a DFA that accepts strings in the grammar.
- Consider a DFA for grammar in Slide 6.



[1] http://en.wikipedia.org/wiki/Deterministic_finite_automaton

Go Back
Full Screen
Quit

6.3. Operators for regular expressions:

- Regular grammars can be written as regular expressions using operators:
  - $+$ one or more repetitions
  - $*$ zero or more repetitions
  - $|$ or
  - Parens for grouping
  - Concatenation: one char followed by another

6.4. Describe these Regular Expressions

- $01^{*}0$
- $(0 \mid 1)^{*}$
- $0^{+}1^{+}$
- $0^{*}1^{*}$
- $a^{+}b^{+}$
- $[A - Z]^{*}$
- $[0 - 9]^{+}$
- $[A - Za - z\_][\_A - Za - z0 - 9]^{*}$

7. Context Free Grammars: CFGs

- The strings accepted by regular grammars do not include many language constructs, e.g. *balanced parens* or *expressions*.
- CFGs can accept both *balanced parens*, and *expressions*, as illustrated in Slide 4.2.
- A *context free grammar* $G(N, T, S, P)$ has production rules P of the form:

$$\mathrm{V} \quad \rightarrow \quad \mathrm{W}$$

- i.e., context doesn't matter so that the only requirement for a grammar to be context free is that the LHS is a non-terminal.

7.1. Equivalent CFGs

- Different CFGs can generate same language
- Given two CFGs, language equality is undecidable.
- Consider two different CFGs. Do they generate the same language?[2]

$$S \rightarrow (\ S \ ) \ S \mid \lambda$$

$$\begin{aligned} S &\rightarrow SS \\ S &\rightarrow (S) \\ S &\rightarrow () \end{aligned}$$

[2] http://en.wikipedia.org/wiki/Context-free_grammar

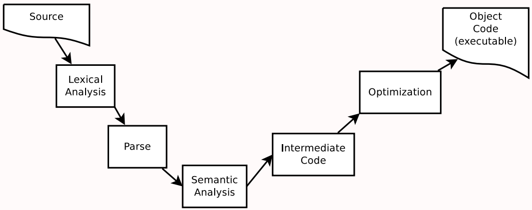
7.2. Parse Trees

- Def: A parse tree, or derivation tree, is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar[3]

- To build a parse tree, begin with the start symbol and expand until the string is formed in the leaves of the tree.

[3] http://en.wikipedia.org/wiki/Parse_tree

- Parse trees are formed (implicitly) during the **parse phase** of compilation:



- Consider a parse tree for $(())$, using the first balanced parens CFG in Slide 7.1: