Formal Grammars

August 24, 2014

Brian A. Malloy

Formal grammar

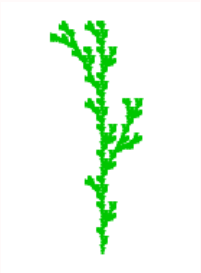
From Wikipedia, the free encyclopedia

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. Why Study Grammars?

- Grammars started when Noam Chomsky, early 1950's, attempted to provide a precise specification for the structure of natural language.
- Chomsky wanted to specify the syntax of language using precise math rules.
- Chomsky's work influenced others to study properties of strings.
- With computers, people learned that all forms of information can be represeneted as strings: numbers, name, pictures, sound, ...

- The collection of strings, known as a language, became central to computer science.
- **The syntax of programming languages can be specified by a grammar.**
- Fractals and L-Systems can be specified or generated with a grammar:



2. What's a Grammar?

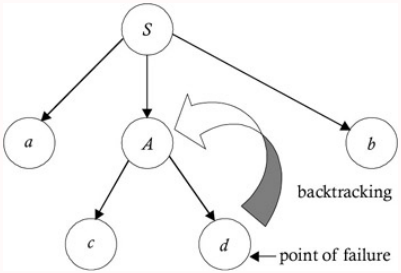
- Def: Formally, a grammar G is a four tuple (N, T, S, P) where N & T are disjount sets of symbols known as non-terminals and terminals, $S \in N$ is the start symbol, and P is a relation on $N \cup T$ of production rules.
- N: non-terminals are generally represented as cap letters, and do not appear 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.

3. 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.

3.1. How it works

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



3.2. Intuition

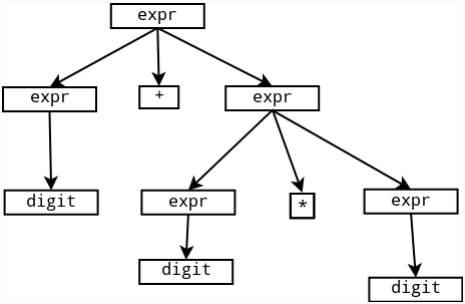
- Try to find a leftmost derivation,
- by searching for parse trees,
- using a top-down expansion of the grammar rules!
- Tokens are consumed from left to right.




3.3. Example: Expression Grammar

Derive 3 + 2 * 7:

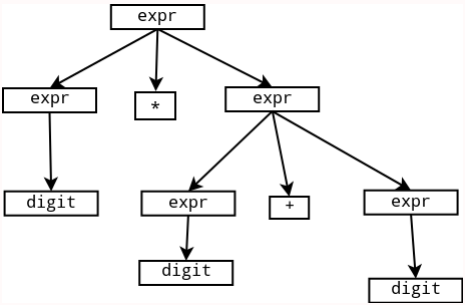
```
expr    : expr '+' expr
        | expr '*' expr
        | DIGIT
```



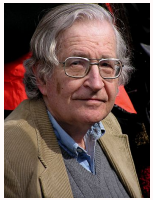
3.4. Example: Second Derivation

Derive 3 + 2 * 7:

- Which one is correct?
- If there are two different parse trees for the same grammar, what does this mean?



4. Classes of Grammars: The Chomsky Hierarchy



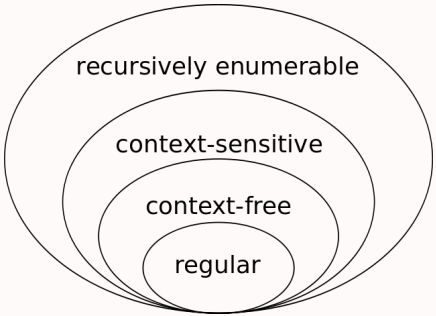
- A grammar defines or generates a language
- A grammar enables the use of a computer to systematically model a language
- A language is an infinite set of strings or symbols

4.1. The Hierarchy

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

- We're interested in Regular and CFGs
- Regular grammars can specify tokens
- CFGs can specify language constructs, or syntactic categories

4.2. Set Inclusion



5. Regular Grammars

- Can specify terminals, or tokens in the language:
  - Reserved or keywords: **if**, **while**, ...
  - Constants: 3.5, 75
  - Special symbols: (; :? ...
- Operators:
  - + one or more repetitions
  - ∗ zero or more repetitions
  - | or
  - Parens for grouping
  - Concatenation: one char followed by another

5.1. Describe these Regular Expressions

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

Go Back
Full Screen
Quit