Context Free Grammars

Shakir Ullah Shah

Context Free Grammars

Three fundamental areas covered in the book are

1. **Theory of Automata**
2. **Theory of Formal Languages**
3. **Theory of Turing Machines**

We have completed the first area.

We begin exploring the second area in this chapter.

Syntax as a Method for Defining Languages

Recursively definition of the set of valid arithmetic expressions as follows:

Rule 1: Any number is in the set AE.

Rule 2: If x and y are in AE, then so are (x), -(x), (x + y), (x - y), (x * y), (x/y), (x ** y)

where ** is our notation for exponentiation

Note that we use parentheses around every component factor to avoid ambiguity expressions such as 3 + 4 - 5 and 8/4/2.

There is a different way for defining the set AE: **using a set of substitution rules similar to the grammatical rules**.

Defining AE by substitution rules

Start → AE
AE → (AE)

AE → -(AE)

AE → (AE + AE)

AE → (AE - AE)

AE → (AE * AE)

AE → (AE/AE)

AE → (AE ** AE)

AE → AnyNumber

Rule 2: If x and y are in AE, then so are
(x), -(x), (x + y), (x - y), (x * y), (x/y), (x ** y)

Rule 1: Any number is in the set AE.

Example

We will show that ((3 + 4) * (6 + 7)) is in AE

Start => AE => (AE * AE)

=>((AE + AE) * (AE + AE))

=>((3 + 4) * (6 + 7))

1. Start → AE
2. AE → (AE + AE)
3. AE → (AE - AE)
4. AE → (AE * AE)
5. AE → (AE/AE)
6. AE → (AE ** AE)
7. AE → (AE)
8. AE → -(AE)
9. AE → AnyNumber

Definition of Terms

A word that cannot be replaced by anything is called **terminal**.

- In the above example, the terminals are the phrase AnyNumber, and the symbols + - * / ** ( )

A word that must be replaced by other things is called **nonterminal**.

- The nonterminals are Start and AE.

The sequence of applications of the rules that produces the finished string of terminals from the starting symbol is called a **derivation** or a **generation** of the word.

The grammatical rules are referred to as **productions**.

Symbolism for Generative Grammars

Definition:
A **context-free grammar (CFG)** is a collection of three things:

1. An alphabet ∑ of letters called **terminals** from which we are going to make strings that will be the words of a language.

2. A set of symbols called **nonterminals**, one of which is the symbol S, standing for “start here”.

3. A finite set of **productions** of the form:
One nonterminal → finite string of terminals and/or nonterminals

where the strings of terminals and nonterminals can consist of only terminals, or of only nonterminals, or of any mixture of terminals and nonterminals, or even the empty string. We require that at least one production has the nonterminal *S* as its left side.

Definition:
The **language generated by a CFG** is the set of all strings of terminals that can be produced from the start symbol S using the productions as substitutions.
A language generated by a CFG is called a **context-free language (CFL)**.

Notes:
The language generated by a CFG is also called the **language defined by the CFG**, or the **language derived from the CFG**, or the **language produced by the CFG**.

We insist that **nonterminals be designated by capital letters**, whereas **terminals are designated by lowercase letters and special symbols**.

Example

Let the only terminal be a and the productions be

- Prod1 S → aS
- Prod2 S → Λ

If we apply Prod 1 six times and then apply Prod 2, we generate the following:

- S => aS => aaS => aaaS => aaaaS
- => aaaaaS => aaaaaaS => aaaaaaΛ = aaaaaa

If we apply Prod2 without Prod1, we find that Λ is in the language generated by this CFG.
Hence, this CFL is exactly a*.
Note: the symbol “→” means “can be replaced by”, whereas the symbol “=>” means “can develop to”.

Let the terminals be a and b, the only nonterminal be S, and the productions be

Prod1 S → aS

Prod2 S → bS

Prod3 S → Λ

The word *ab* can be generated by the derivation

S => aS => abS => abΛ = ab

The word baab can be generated by

S => bS => baS => baaS => baabS => baabΛ = baab

Clearly, the language generated by the above CFG is

(a + b)*.

Let the terminals be a and b, the only nonterminal be S, and the productions be

Prod1 S → aS

Prod2 S → bS

Prod3 S → a

Prod4 S → b

The language generated by the above CFG is

(a + b)+.

Example

Let the terminals be a and b, the the nonterminal be S and X, and the productions be

Prod 1 $S \rightarrow XaaX$

Prod 2 $X \rightarrow aX$

Prod 3 $X \rightarrow bX$

Prod 4 $X \rightarrow \Lambda$

We already know from the previous example that the last three productions will generate any possible strings of a's and b's from the nonterminal X.

Hence, the words generated from S have the form

anything aa anything

Hence, the language produced by this CFG is

(a + b)*aa(a + b)*

which is the language of all words with a double a in them somewhere.

For example, the word *baabb* can be generated by

S ➔ XaaX ➔ bXaaX ➔ baaX ➔ baaX

➔ baabX ➔ baabbX ➔ baabbΛ = baabb

Example

Consider the CFG:

$S \rightarrow aSb$

$S \rightarrow \Lambda$

It is easy to verify that the language generated by this CFG is the **non-regular** language $\{a^n b^n\}$.

For example, the word a^4b^4 is derived by

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb$

$\Rightarrow aaaaSbbbb \Rightarrow aaaa\Lambda bbbb = aaaabbbb$

Disjunction Symbol |

Let us introduce the symbol | to mean disjunction (or).
We use this symbol to combine all the productions that have the same left side.
For example, the CFG

Prod 1 $S \rightarrow XaaX$

Prod 2 $X \rightarrow aX$

Prod 3 $X \rightarrow bX$

Prod 4 $X \rightarrow \Lambda$

can be written more compactly as

Prod 1 $S \rightarrow XaaX$

Prod 2 $X \rightarrow aX|bX|\Lambda$