**Context-Free Grammars**

A context-free grammar (CFG) is a set of recursive rewriting rules (or *productions*) used to generate patterns of strings.

A CFG consists of the following components:

* a set of *terminal symbols*, which are the characters of the alphabet that appear in the strings generated by the grammar.
* a set of *nonterminal symbols*, which are placeholders for patterns of terminal symbols that can be generated by the nonterminal symbols.
* a set of *productions*, which are rules for replacing (or rewriting) nonterminal symbols (on the left side of the production) in a string with other nonterminal or terminal symbols (on the right side of the production).
* a *start symbol*, which is a special nonterminal symbol that appears in the initial string generated by the grammar.

To generate a string of terminal symbols from a CFG, we:

* Begin with a string consisting of the start symbol;
* Apply one of the productions with the start symbol on the left hand size, replacing the start symbol with the right hand side of the production;
* Repeat the process of selecting nonterminal symbols in the string, and replacing them with the right hand side of some corresponding production, until all nonterminals have been replaced by terminal symbols.

**A CFG for Arithmetic Expressions**

An example grammar that generates strings representing arithmetic expressions with the four operators +, -, \*, /, and numbers as operands is:

1. <expression> --> number
2. <expression> --> ( <expression> )
3. <expression> --> <expression> + <expression>
4. <expression> --> <expression> - <expression>
5. <expression> --> <expression> \* <expression>
6. <expression> --> <expression> / <expression>

The only nonterminal symbol in this grammar is <expression>, which is also the start symbol. The terminal symbols are {+,-,\*,/,(,),number}. (We will interpret "number" to represent any valid number.)

The first rule (or production) states that an <expression> can be rewritten as (or replaced by) a number. In other words, a number is a valid expression.

The second rule says that an <expression> enclosed in parentheses is also an <expression>. Note that this rule defines an expression in terms of expressions, an example of the use of recursion in the definition of context-free grammars.

The remaining rules say that the sum, difference, product, or division of two <expression>s is also an expression.

**Generating Strings from a CFG**

In our grammar for arithmetic expressions, the start symbol is <expression>, so our initial string is:

<expression>

Using rule 5 we can choose to replace this nonterminal, producing the string:

<expression> \* <expression>

We now have two nonterminals to replace. We can apply rule 3 to the first nonterminal, producing the string:

<expression> + <expression> \* <expression>

We can apply rule two to the first nonterminal in this string to produce:

(<expression>) + <expression> \* <expression>

If we apply rule 1 to the remaining nonterminals (the recursion must end somewhere!), we get:

(number) + number \* number

This is a valid arithmetic expression, as generated by the grammar.

When applying the rules above, we often face a choice as to which production to choose. Different choices will typically result in different strings being generated.

Given a grammar G with start symbol S, if there is some sequence of productions that, when applied to the initial string S, result in the string s, then s is in L(G), the language of the grammar.