

Formal languages are not concerned with what anything means, but rather only what is or is not a correctly formed sentence (i.e., “well formed formula (wff)”) in the language. The ‘sentences’ are simply sequences of symbols from some ‘alphabet’ and do not have any inherent meaning. A formal language is simply a (possibly infinite) set of the sequences that are in the language.

One way of describing a formal language, is to give a grammar that says how to generate them. Grammars consist of a set of rewriting (*production*) rules, each consisting of two sequences of symbols. Some of these symbols occur in at least one wff (these are called *terminals*) and some never do (these are *non-terminals*). The general procedure is to start with some designated *start symbol* and apply one of the rewriting rules to get a sequence of symbols and keep on iteratively applying rewriting rules to a part of the sequence. The left-hand side of each rule must always contain at least one non-terminal, and the rewriting stops when one is left with a sequence made up entirely of terminal symbols. All of the sequences that can be produced in this manner wff in the language described by the grammar.

For example, we might define a language with:

Production Rules:	Where:
$S \rightarrow wABx$	— the start symbol is $S$
$wA \rightarrow yA$	— the <i>non-terminals</i> are $S$ , $A$ , and $B$
$ABx \rightarrow z$	— the <i>terminals</i> are $w$ , $x$ , $y$ , and $z$

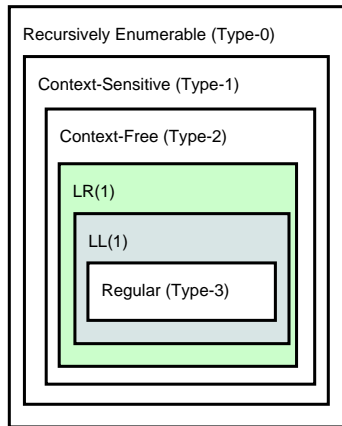
There are exactly two wff in this language:  $wz$  and  $yz$ .

$wz$  can be produced using the derivation:  $S \rightarrow wABx \rightarrow wz$ .

$yz$  can be produced using the derivation:  $S \rightarrow wABx \rightarrow yABx \rightarrow yz$ .

The class of languages that can be described with this type of grammar is known as “recursively enumerable” languages. If you the kinds of symbols that can appear on the left and right side and the relationships between the two sequences, you can define other less-general classes of formal languages. Four of these are types of languages grammars in the Chomsky Hierarchy (see Figure). The most restricted class is the regular languages, and all regular languages are also context-free languages. All context-free languages are also context-sensitive languages, and all context-sensitive languages are also recursively-enumerable languages. (As an aside, these languages can also be defined, in terms of the abstract machine required to look at a sequence of symbols and decide whether the sequence is or is not a wff for a given language. To recognize arbitrary recursively-enumerable languages requires a Turing machine.)

For our purposes, the most interesting class of languages is the context-free languages. The syntax of most programming languages can be (and is) described



by context-free grammar.<sup>1</sup> Context-free grammars are restricted so that the left-hand side of every production rule consists of exactly one *non-terminal*. The right-hand side may, however, still be an arbitrary sequence of *non-terminal* and *terminal* symbols.

When applied to programming languages, context-free grammars are usually written in a notation called BNF (Backus Normal Form or Backus-Naur Form). In BNF, one normally uses words instead of letters as the individual symbols,<sup>2</sup> and non-terminals are distinguished from terminal by bracketing the non-terminals with  $\langle$  and  $\rangle$ . BNF also replaces ' $\rightarrow$ ' with  $::=$ .

The following is an example of a BNF grammar:

```

<expr> ::= <expr> + <expr>
<expr> ::= <expr> - <expr>
<expr> ::= <num>
<num>  ::= 1
<num>  ::= 2

```

This would include wff such as 1, 2, 1 + 1, 1 + 2 - 1, 1 - 1 - 1

BNF also uses a '|' symbol as short-hand to separate multiple production rules arising from the same non-terminal, so the grammar above could be rewritten:

```

<expr> ::= <expr> + <expr> | <expr> - <expr> | <num>
<num>  ::= 1 | 2

```

For more information: Mitchell 4.1; Aho, et al., *Compilers: Principles, Techniques, and Tools*; Wikipedia, Formal languages

<sup>1</sup>More specifically, most programming languages grammars belong to a subclass of context-free grammars known as LR grammars, which can be parsed by automatic tools, such as yacc. Some, like Pascal, belong to even smaller subclasses, such as LL(1).

<sup>2</sup>Programming Languages usually have separate 'lexical' rules that define a regular language for putting together letters to form words. These rules are often expressed using regular expressions, which can be processed using automatic tools, such as lex.