# Parser Design II

Work in the theory of computability shows that many different forms of algorithm presentation are equivalent in the sense that one form can be converted to another. This is a very powerful idea and we will approach this from a categorical point of view.

## Theoretical Underpinnings

The conversion (*compilation* if you will) we want to make is between the production system and a typical programming language. The background is based on work on the **Chomsky Hierarchy.** The Chomsky Hierarchy is a subtheory of the theory of computation that indicates what abstract computing models are equivalent in the sense that programs in one model produces the same outputs as a particular algorithm specified in the other model. In this case, we are interested in two models: Production Systems and Recursive Functions. All current programming languages are examples of recursive function systems. Therefore, we will make the task one of going from the formal definition of production systems to some sort of programming language.

We know that there are four parameters in the grammar: *N, T, G, S*. Consider now trying to write a program. What is being specified by *N, T, G,* and *S*?

The key to Subtask 1 is to understand that the categorical symbols (symbols in *N)* are really the name for programs and that the programs implement the rules in *G*.

## Assignment

Use pseudo-code to convert the "S" production of the grammar to a program.