5

INTRODUCTION TO LINGUISTICS

Programming languages are, first and foremost, languages. Languages have been studied by philosophers for several thousand years. It seems appropriate to begin our study of programming languages by reviewing the more fundamental ideas of language.

Language is fundamentally associated with representation, communication, meaning, and truth. These same issues arise in programming languages, although they are rarely discussed in the literature.

Modern studies of language attempt to study language in a systematic way, often using mathematical formalisms. A group known as *logical positivists* that included Bertrand Russell, Ludwig von Wittgenstein, and Rudolf Carnap was particularly influential. The positivists are important because of their profound impact on computer science. The positivists strove to employ rigorous accounts of logic and of meaning in attempts to penetrate, and in some cases to dispel, traditional philosophical questions. The positivists sought complete meaning by investigating language through *linguistic analysis*.

5.1 FOCUS PROBLEM

The purpose of this chapter is to develop the technical vocabulary of linguistics and language. The "deliverable" for this chapter is an initial decomposition of the term "compiler."

5.1.1 Technical Vocabulary

The philosophy of language (linguistics) introduces a set of standard, technical concepts that can be applied to any language, natural or formal. All languages have *syntax, semantics,* and *pragmatics* (see Figure 5.1).

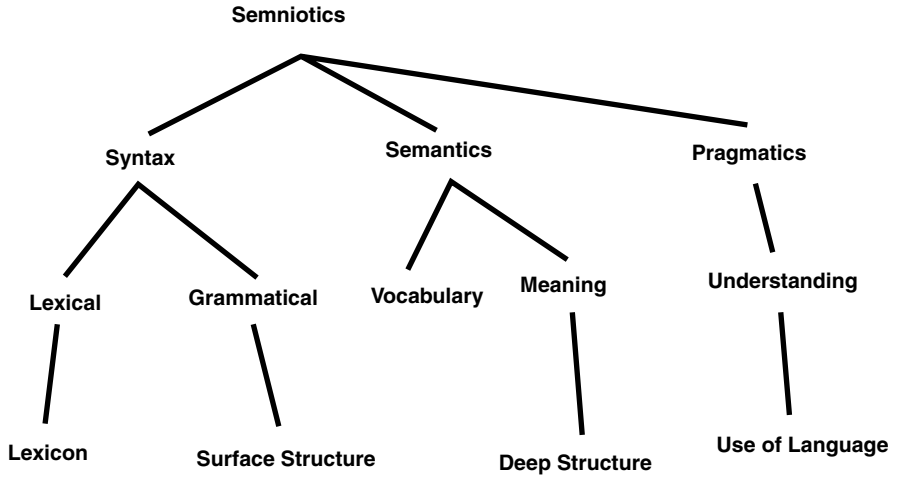

Figure 5.1 Linguistics Word Association Map

What are these properties? How do they describe elements of a given language? What subproperties do they each imply? In order to develop a compiler, we must first have an understanding of the problem *from the problem-poser's standpoint.*

5.1.2 Deliverable

By the end of the chapter, we need to have a written understanding of what the basic parts of the compiler should be, what its data should be, and what the overall processing sequence should be.

The method for this deliverable is the following:

1. Read the chapter. Make a list of all words that relate to language and linguistics.
2. Classify each word as to its part of speech: noun, verb, etc.
3. Choose a simple expression from our list of examples.
4. Write a simple story of how the example is processed, using the following rules:
   - Common nouns are the names of data types.
   - Proper nouns are variable names.
   - Verbs are the names of processes.

Suppose we choose `[let [a 1]]` as our example. The story should explain how the line is taken from the file, parsed, and coded. Obviously, at this point, there will be many details lacking, but some very basic information gained.

5.2 PHILOSOPHY OF LINGUISTICS

Natural (human) languages are powerful and complex systems. The science of this capacity and of these systems is *linguistics*. The term *computational linguistics* generally means studying natural language computationally. We adopt, where necessary, the more descriptive term *programming language linguistics* as the study of languages used to describe computations, which may include descriptions meant for other than the computer. Like other sciences, and perhaps to an unusual degree, linguistics confronts difficult foundational, methodological, and conceptual issues.

We seek systematic explanations of a language's syntax (the organization of the language's properly constructed expressions, such as phrases and sentences), its semantics (the ways expressions exhibit and contribute to meaning), and its pragmatics (the practices of communication in which the expressions find use). This study is called *semniotics*.

The study of syntax has been guided since the 1960s by the work of Noam Chomsky. Chomsky, who lends his name to the Chomsky Hierarchy in theoretical computer science, takes a cognitivist approach. Human linguistic capacities, he holds, are innate and come from a dedicated cognitive faculty whose structure is the proper topic of linguistics. Chomsky's direct contribution to programming languages—indeed, a fundamental contribution—comes from a 1957 article concerning a hierarchy of syntactic structures. In programming languages, syntax is specified using the principles of *formal languages* in general and context-free languages in particular, largely in line with Chomsky's ideas. The fundamental contribution of formal grammars and production systems is the work of Emil Post.

Semantics, on the other hand, is far more difficult. Ideas of programming languages are received primarily from *logic* and *recursive functions*. Here many of the great strides have been made by philosophers, including Gottlob Frege, Alonzo Church, Bertrand Russell, Ludwig von Wittgenstein, Rudolf Carnap, Richard Montague, Saul Kripke, Dana Scott, and Christopher Strachey to name but a few. The role of semantics in programming language linguistics is development and careful application of formal, mathematical models for characterizing linguistic form and meaning.

Philosophical interest in pragmatics typically has had its source in a prior interest in semantics in a desire to understand how meaning and truth are situated in the concrete practices of linguistic communication. As an example of the role of pragmatics, consider the computational concept of *increment an integer variable* `i` and the C statement `i = i+1`. More than one person has commented that "this is a ridiculous statement since no integer is equal to its successor." We can immediately see that programming language pragmatics is its own study.

Modern programming systems use many different input media, not just keyboards. Therefore, we should look broadly at *language* and investigate

semniotics, which is the study of signs and signification in general, whether linguistic or not. In the view of the scholars in this field, the study of linguistic meaning should be situated in a more general project that encompasses gestural communication, artistic expression, etc. For example, mathematical notation constitutes a language in just this way.

5.3 MEANING: LANGUAGE, MIND, AND WORLD

Language, of course, is not a subject in isolation (see Crimmins 1998). On one hand, language is used for communications, and on the other it is used to represent thoughts in the mind. The relationships among means of communication constitute the meaning of language.

Language may be studied from many directions, but we are interested in how language relates to two "communities": one is the programmer community and the other is the computer development community. Yes, this is a bit of whimsy, but in its essence, programming languages are about transferring information from the human programmer to the computer. In this act, the compiler of the programming language must be aware of the communication aspect as well as the preservation of intention.

In some real sense, meaningfulness in programming language derives from the mental content and intent of the programmer, perhaps including the contents of beliefs, thoughts, and concepts. This means there is a cognitive aspect to understanding programming languages.

It has not gone unquestioned that the mind indeed can assign meaning to language; in fact, skepticism about this has figured quite prominently in philosophical discussions of language. It has also played a large role in developing programming languages. There are three basic approaches to assigning meaning to programming language constructs.

1. Operational. Operational meaning was the first meaning theory based on the naïve idea that the computer evaluation of a construct was the meaning of the construct. But what if one changes computer systems?
2. Axiomatic. Axiomatic theories of meaning are based on specifying the connotation of a construct by logical means. *Connotation* means that an essential property or group of properties of a thing is named by a term in logic.
3. Denotational. Denotational semantics was proposed as a mechanism for defining meaning by specifying a mapping from syntactic constructs to values. Often called Strachey–Scott semantics for its proposers, denotational semantics is a part of λ-calculus and recursive functions. Denotation is a direct specific meaning (value) as distinct from an implied or associated idea in connotation.

It is important to understand the role of language in shaping our thoughts. The Sapir–Whorf hypothesis states that there is a systematic relationship between the grammatical categories of the language a person speaks and how that person both understands the world and behaves in it. While the stronger versions of this hypothesis are no longer believed by researchers, it is clear that having language is so crucial to our ability to frame the sophisticated thoughts that appear essential to language use and understanding, that many doubt whether mind is *prior* to language in any interesting sense.

One must be careful about the relationship of language and the world. According to this picture, the key to meaning is the notion of a truth-condition. The meaning of a statement determines a condition that must be met if it is to be true. According to the truth-conditional picture of meaning, the core of what a statement means is its truth-condition—which helps determine the way reality is said to be in it—and the core of what a word means is the contribution it makes to this.

While the truth-conditional picture of meaning has dominated semantics, some philosophers urge that the key to meaning is a notion of correct use. According to this alternative picture, the meaning of a sentence is the rule for its appropriate utterance. Of course, the two pictures converge if sentences are correctly used exactly when they are true. The challenge illustrates a sense in which the Mind/Language and Language/World connections can seem to place a tension on the notion of meaning (meaning is whatever we cognitively grasp, but the meaning of language just is its bearing on the world).

Our first task in understanding programming languages is to understand the terminology used to describe such languages. That terminology is itself a language. These early cases will guide you to an understanding of how humans process language; that process is the basis for developing a program to process and translate language.

CASE 34. WHAT DOES AN INFINITE LOOP MEAN?

For most computer science students, all this background is new information and you might ask, "Why should I care?" In order to convince you, the reader, that all is not as simple as we would like, write a short paper on the following question:

What is the meaning of the C statement `while(1);`?

5.4 WHAT IS SYNTAX?

When first approaching a problem that we have never solved before, we are in a learning mode. We must effectively learn enough about the underlying concepts of the problem so we can form a coherent picture of the problem.

A fundamental tenet of learning theory is that we must first determine what we actually know. We start out by investigating two things we do know about: we all speak natural languages and we all can program.

For this portion of the course, we need to understand language in the broader context of natural language. This provides us with important anchor points. As we discover what we know about natural language, we can ask, "Well, how does this particular idea play out in programming languages?"

Since you have made it this far in your education, you have undoubtedly studied at least one natural language. You may be lucky enough to speak several languages. While you can learn a language without being completely aware of technicalities, we must now understand those technicalities because discussing programming languages requires these technical issues.

We are interested in invariant parts of language and we call these structures. The term *structure* in this context means "the mutual relation of the constituent parts or elements of a language as they determine its character; organization, framework."*

Syntax is a term that denotes the study of the ways in which words combine into such units as *phrase*, *clause*, and *sentence*. The sequences that result from the combinations are referred to as *syntactic structures*. The ways in which components of words are combined into words are studied in *morphology*; syntax and morphology together are generally regarded as the major constituents of grammar (informally, grammar is synonymous with syntax and excludes morphology). Syntactic descriptions do not usually go beyond the level of the sentence.†

Morphology is the study of the structure of words, as opposed to syntax, which is the study of the arrangement of words in the higher units of phrases, clauses, and sentences. The two major branches are inflectional morphology (the study of inflections) and lexical morphology (the study of *word formation*).‡

---

* *The Oxford Dictionary of English Grammar*, Ed. Sylvia Chalker and Edmund Weiner. Oxford University Press, 1998. Oxford Reference Online. Oxford University Press. Clemson University. 12 February 2005 <http://www.oxfordreference.com/views/ENTRY.html?subview=Main&entry=t28.e1424>

† *Concise Oxford Companion to the English Language*, Ed. Tom McArthur. Oxford University Press, 1998. Oxford Reference Online. Oxford University Press. Clemson University. 12 February 2005 <http://www.oxfordreference.com/views/ENTRY.html?subview=Main&entry=t29.e1200>

‡ *Concise Oxford Companion to the English Language*, Ed. Tom McArthur. Oxford University Press, 1998. Oxford Reference Online. Oxford University Press. Clemson University. 12 February 2005 <http://www.oxfordreference.com/views/ENTRY.html?subview=Main&entry=t29.e804>

Grammar is "the system by which words are used together to form meaningful utterances. It denotes both the system as it is found to exist in the use of a language (also called descriptive grammar) and the set of rules that form the basis of the standard language, i.e., the variety of a language that is regarded as most socially acceptable at a given time (also called prescriptive grammar)." *

Lexical structures are those relating to lexicography, "the principles and practices of dictionary making. Belonging to, or involving units that belong to, the lexicon. E.g. a lexical entry is an entry in the lexicon; a lexical item or lexical unit is a word, etc. which has such an entry; rules are lexically governed if they apply only to structures including certain lexical units."†

The purpose of this chapter is to explore these definitions and to understand how they play out in programming language.

CASE 35. WHAT CAN WE LEARN FROM A DICTIONARY?

The central concept in *vocabulary* is the dictionary. The dictionary in a compiler is generally called a *symbol table,* although there may, in fact, be many different tables, depending on the designer's view. Central to understanding the symbol table is understanding how an English language dictionary is laid out. Study an unabridged dictionary, such as *Merriam-Webster* or the *Oxford English Dictionary*. Develop a data representation for the *structure* of the dictionary and outlines for the following *queries*:

1. Looking up a word to determine whether or not it's in the dictionary
2. Looking up a word when you want to know the definition within a specific part of speech
3. How to handle multiple definitions of the same part of speech

This case provides a good opportunity to describe *mind maps* as a brainstorming device. Consider Figure 5.2. The concept behind mind mapping is word or phase associations. In the figure, the word "Dictionary" was placed in the center of the palette.‡ The right-hand side of the map was drawn by considering the words or terms that are suggested by the

* *Pocket Fowler's Modern English Usage*, Ed. Robert Allen. Oxford University Press, 1999. Oxford Reference Online. Oxford University Press. Clemson University. 12 February 2005 <http://www.oxfordreference.com/views/ENTRY.html?subview=Main&entry=t30 .e1674>

† *The Concise Oxford Dictionary of Linguistics*. P. H. Matthews. Oxford University Press, 1997. Oxford Reference Online. Oxford University Press. Clemson University. 12 February 2005 <http://www.oxfordreference.com/views/ENTRY.html?subview=Main&entry=t36 .e1719>

‡ The FreeMind software, available at SourceForge, was used for this example.

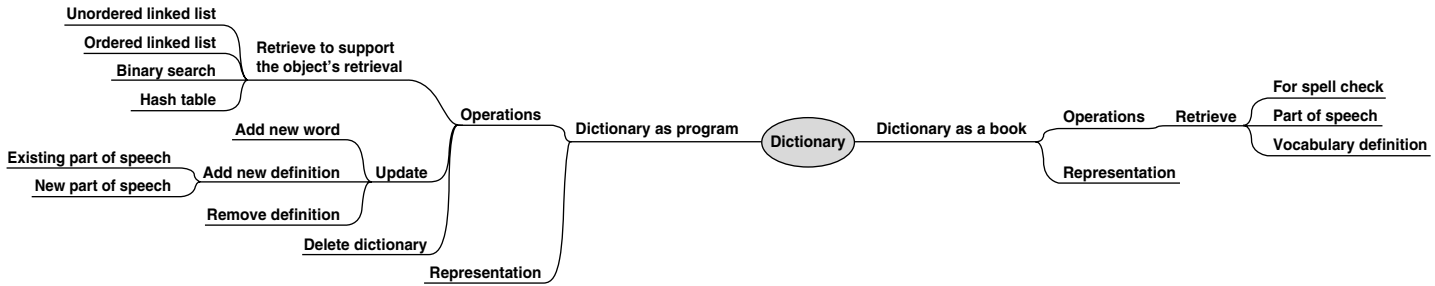

Figure 5.2 Mind Map of the Dictionary Case

word dictionary. In this case, the searching functions came to mind. The left-hand side was derived in the same way, but with the information of the right-hand side clearly in mind. The left-hand side also makes use of the CRUD mnemonic for developing data structures: **C**reate, **R**ead, **U**pdate, and **D**estroy. These are the four issues that must be addressed in design.

CASE 36. HOW DOES WHAT WE LEARNED PLAY OUT IN PROGRAMMING LANGUAGES?

The preceding case should have given you some ideas about describing the symbols of a language. A dictionary entry is a word often with a large number of meanings. Each word also has a part of speech. Common words often have many meanings that are quite different in different contexts.

We might expect to find context in our study. *Context* is "the speech, writing, or print that normally precedes and follows a word or other element of language. The meaning of words may be affected by their context. If a phrase is quoted out of context, its effect may be different from what was originally intended."*

Forgetting the contextual issue for the time being, our next task is simply to identify all the parts of speech that commonly occur in current programming languages. Let's be specific.

http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html

is the Java Language Specification. (A more reliable way to find this is to search for "Java Language Specification" in a search engine.) Your assignment is to develop a list of categories of symbols that appear in Java. If you spend some time in the Table of Contents, you will find exactly what you need to know in one place. For some ideas on how to read technical literature, see Section 5.7 on Focused SQRRR.

You must develop a table with the following columns:

Category	Use	Spelling Rules

It turns out that the spelling rules are well modeled by finite state machine diagrams. The specification provides enough detailed information to make this easy.

* *Concise Oxford Companion to the English Language*, Ed. Tom McArthur. Oxford University Press, 1998. Oxford Reference Online. Oxford University Press. Clemson University. 12 February 2005 <http://www.oxfordreference.com/views/ENTRY.html?subview=Main&entry=t29.e309>

5.5 GRAMMAR

CASE 37. ENGLISH GRAMMAR WITH PRODUCTION RULES

The purpose of the preceding cases is to develop a vocabulary about words. Natural language sentences consist of words, but not just any arbitrary order. Natural language grammars specify the order of words, but it is not that simple.

English is organized in a hierarchy of types: sentences, clauses, phrases, and words. Sentences are made up of one or more clauses, clauses consist of one or more phrases, and phrases consist of one or more words. From our preceding work, we know that words are classified by their part of speech, so we might substitute *part of speech* for *word*. Also, the hierarchy is not just a hierarchy because a clause can appear in a phrase, for example.

Our purpose is to understand how grammars are developed and used so we can develop a similar idea in programming language. The mechanism often used is to develop a syntax tree that represents the relationship of the hierarchy in the sentence at hand.

I will describe the *thinking process*. Let's consider a *simple sentence*.

My brother is a lawyer.

This is a simple sentence that I denote as *S*. A simple sentence is defined to consist of a *noun phrase (NP)* followed by a *verb phrase (VP)*. Symbols such as *NP* are called *categorical symbols* because they stand for whole sets of possible sequences of symbols. One way I can describe this rule is to invent some simple notation to write with. I choose to use a notation similar to that used in formal languages in computer science.* I will encode the sentence: "A simple sentence is a noun phrase followed by a verb phrase." by

$$S \rightarrow NP\ VP$$

The *application* of that rule now divides the sentence *S* into two pieces. To indicate this relationship, I will rewrite the sentence, collecting the various pieces. This rewriting process is called a *derivation*: the *rule* (column 2) is applied to the input (column 1) and the result then becomes input to the next step:

* This isn't surprising because computer science originally took the notation from linguistics in the 1950s and 1960s.

Step	*Input String*	*Rule Applied*
1	[S: My brother is a lawyer]	S → NP VP
2	[S: [NP: My brother] [VP: is a lawyer]]	

The verb phrase "is a lawyer" is the next thing to look at. A verb phrase is a verb (*V*) followed by a noun phrase

$$VP \rightarrow V\,NP$$

Step	*Input String*	*Rule Applied*
1	[S: My brother is a lawyer]	S→ NP VP
2	[S: [NP: My brother] [VP: is a lawyer]]	VP → V NP
3	[S: [NP: My brother] [VP: [V: is] [NP: a lawyer]]]	NP → ADJ N
4	[S: [NP: [ADJ My] [N: brother]][VP: [V: is] [NP: [ADJ: a][N: lawyer]]]]	

where we use the rule (twice) that a noun phrase can be an adjective followed by a noun.

Each of the bracketed units is a word, a phrase, or a clause. We refer to these as *constituents*, defined as a word or a group of words that act syntactically as a unit.

Although the brackets are able to portray the grammatical hierarchy, this notation is cumbersome. If we are to make use of the ideas of grammars, then we need to have something more computable. It turns out that *hierarchy* is a synonym for *tree* and trees are something computer scientists know how to deal with. A tree for the statement "My brother is a lawyer" based on our grammar is shown in Figure 5.3. The tree is drawn with two types of arcs between nodes. The solid arcs are *grammatical*, or what is known as the *deep structure*. The dashed arcs indicate *bindings* of specific words to grammatical *terminals*. A *binding* is an association of a term to its definition. The power of the grammar is that a countable number of sentences can be developed by binding different words to the terminals; for example, "Our house is a home." Meaning is derived from the deep structure.

CASE 38. GRAMMARS IN PROGRAMMING LANGUAGES

We have completed exercises that explore morphology and we know something about grammar. We need to finish syntax considerations by looking at the Java grammar. You should have seen that the Web site document contains the actual grammar that can be used to define Java. This grammar is larger than the simple example above, but it is exactly the same idea.

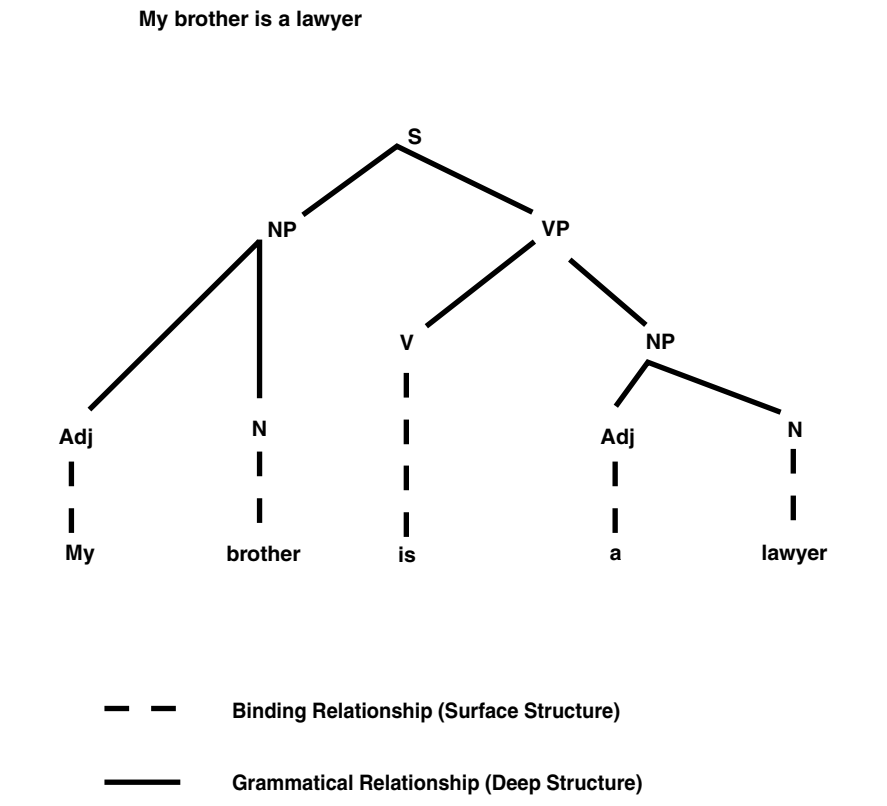

Figure 5.3 Syntax Tree for "My Brother Is a Lawyer"

If you have forgotten the details of what a grammar is or how it is specified, then you can read Chapters 1 and 2 of the Java Specification.

Here is a simple Java program:

```
public static void main(String args[]) {
        System.out.println(''Hello World'');
    }
```

Chapter 16 has the full Java grammar. Your task is to start with "MethodDecl" and follow it until you have derived a simple function.

The basic points for Case 38 can be portrayed by a concept map. A concept map is similar to a mind map but there is one very important difference. The nodes in a concept map are concepts, but two concepts must be connected by a relationship. In the concept map shown in Figure 5.4, the concepts are enclosed in figures; the relationships have no enclosing

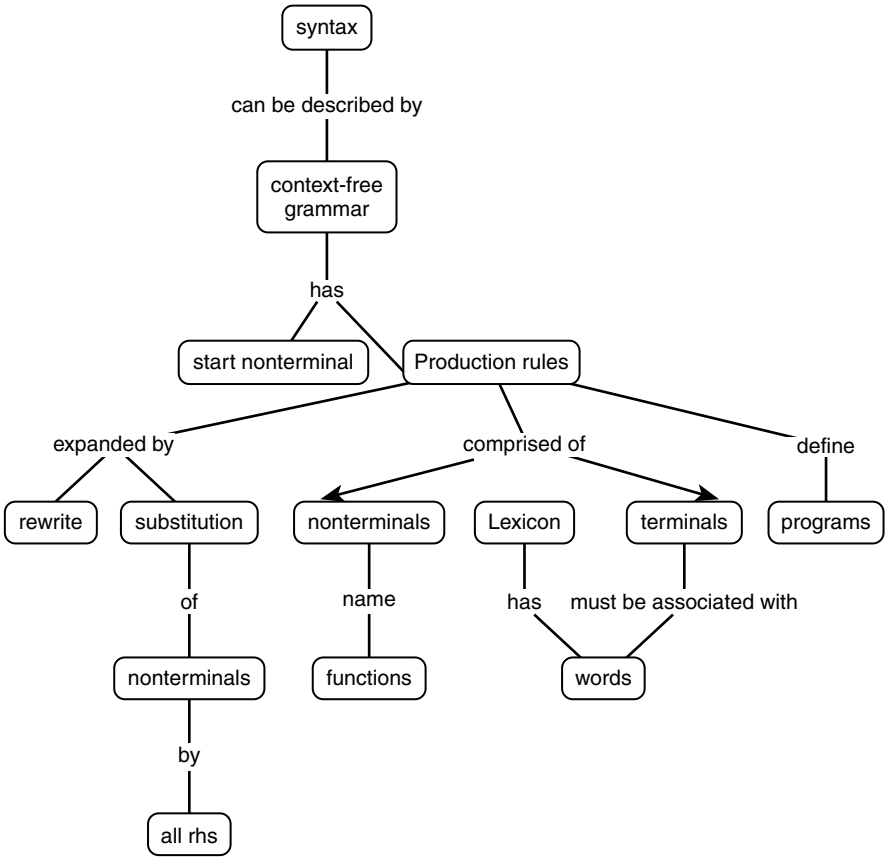

Figure 5.4 Concept Map for Syntax and Grammars

lines. Concept maps are directed, bipartite graphs. This particular concept map was drawn by software available from the Institute for Human and Machine Cognition at the University of West Florida.

5.6 SEMANTICS

To this point we have only been concerned with the *form* of the sentence. We have not made much of a distinction between the written and the spoken sentence. Because programming languages are often written (but need not be: consider visual programming languages) and the project certainly is, the discussion is slanted toward written information. Based on the discussion in Case 37, it is clear that sentences in either mode can be written in a tree wherein the leaves are the exact words and the interior nodes represent production names. While this tree represents a good bit of the

Word	*Part of Speech*	*Usage*
My	adjective	modifies *brother*
brother	noun	subject
is	verb	verb
a	article	modifies *lawyer*
lawyer	lawyer	predicate noun

Figure 5.5 Usage of Words in "My Brother Is a Lawyer"

information needed to understand a sentence, it is not the whole picture. How can we investigate this process and turn it to our understanding of programming languages?

Semantics is the study of meaning and this is much different from syntax. Because graphical methods are desirable in the computing context, it would be nice to have a mechanism to consider semantics from a graphical point of view. One mechanism is *sentence diagrams*. The purpose of a sentence diagram is to demonstrate the relationships among the words of a sentence, as well as the purpose of groups of words. For example, the case of: "My brother is a lawyer" has the usage pattern shown in Figure 5.5. A diagram of this sentence, along with its parse tree, is shown in Figure 5.6.

1. *brother* is the *subject*.
2. *is* is the *verb*.
3. *lawyer* is the *predicate*.
4. *my* is an *adjective* modifying *brother*.
5. *a* is an *adjective* modifying *lawyer*.

5.7 FOCUSED SQRRR

Focused SQRRR stands for "Focused SURVEY, QUESTION, READ, RECITE, REVIEW (FSQRRR)." FSQRRR is a technique for reading when there is a specific question to be answered. Again, the purpose is to minimize reading material that does not bear on the problem at hand.

In FSQRRR, we start with *focus questions*. The focus question may be given to you by your instructor or it may be a question that you have formulated in trying to solve a problem. The question must be written out before you start!

5.7.1 Survey and Question

When using this form, write the questions in the spaces provided. Remember that this is a working document: brevity is fine as long as clarity does not suffer.

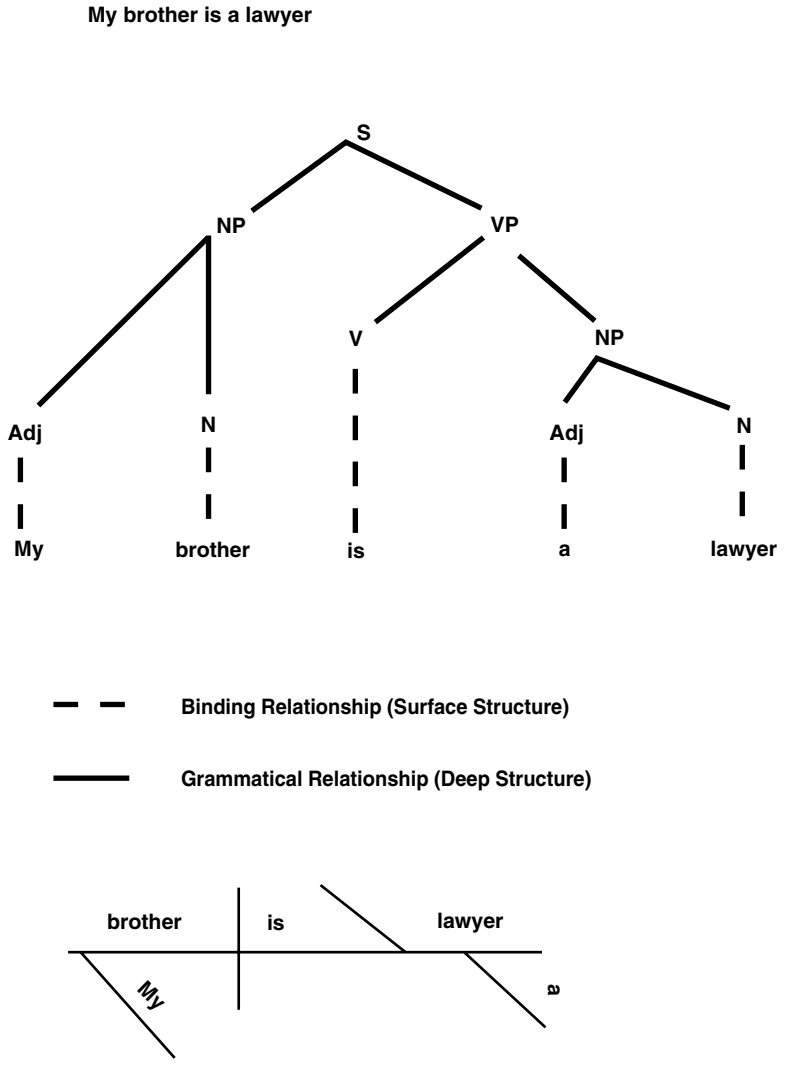

Figure 5.6 Syntax Tree and Diagram of Sentence

1. Write the focus question. Now, rewrite it using your own words. If the question contains technical terms you do not understand, then put the term in parentheses but then write out what you think the term means.
2. Read the title and turn it into a question that is suggested by the focus question.
3. Read the introduction and formulate questions related to the focus question. If the section introduction indicates that this

section does not bear on the focus question, come back to it only if you later believe you need to. (For example, the section that actually addresses the focus question might rely on information developed in this section.)
4. Turn headings and subheadings into questions as explained above.
5. Read captions under pictures, charts, and graphs again, looking for relevance to the focus question.
6. Read the summary. The summary should give you an outline of how the chapter helps answer the focus question.
7. Recall the instructor's comments.
8. Recall what you already know. This is especially crucial with respect to vocabulary.

5.7.2 Read

Before reading further, rewrite the focus question in light of what you have learned in the SQ portion.

1. Look for answers to your questions.
2. Look for answers to the teacher's questions and the book's questions.
3. Reread the captions.
4. Study graphic aids.
5. Carefully read italicized and bolded words.
6. Reread parts that are not clear.

5.7.3 Recite

Now try to put in your own words what you think the answer to the focus question is.

5.7.4 Review

Within 24 hours of your research:

1. Page through material and reacquaint yourself with important points.
2. Read written notes you have made.
3. Reread the focus question and rewrite it in light of what you now know.
4. Reread your answer to the focus question. Rewrite it.