Chapter Two

GRAMMARS, INFERENCE, PARSING AND TRANSDUCTION

“Speak English!” said the Eaglet. “I don’t know the meaning of half those long words, and, what’s more, I don’t believe you do either!”

Lewis Carroll, Alice’s Adventures in Wonderland

At the very beginning of my own work on TTS synthesis, I was looking for a short but accurate introduction to grammars, parsers, and lexicons; an introduction in which the related terminology would be comprehensively explained, with not too many details, but not being too cursory either; an introduction in which the goals and, better still, the coverage limits of grammar models and the efficiency limits of parsers could be perceived.¹ I now know that this research was bound to fail, for even though grammars are often based on simple facts, their implications are particularly hard to appreciate. What is more, linguistic theories often insist on their peculiarities, rather than on their common properties. I thus grasped pieces of information throughout my research and painfully tried to form a coherent view. It was not, however, until I read the inescapable (Gazdar and Mellish, 1989), the didactic (Charniak, 1993) and (Winograd, 1983), the very complete (Sabah, 1989), the rigorous (Miclet, 1984) and the brilliant (and concise) (Gibbon, 1991), and after I had attended the comprehensive introduction to natural language processing courses of Prof. Martin Kay at the ELSNET summer school on prosody (University College of London, 1993) that I began to have deeper insights in this intricate area. The following pages are merely a concise presentation of some basic facts extracted from these references. It does not claim to be complete nor truly accurate, for it inevitably oversimplifies some aspects of grammars. It should be seen as a first introduction to computational linguistics, a map on which essential topics are positioned and shown to constitute if not a coherent set then at least a collection of interrelated approaches. We hope that it can nevertheless be of some help to readers who, like I did, are trying to find their way in what appears at first sight to be a linguistic maze.

¹Following my own conviction that the real expert is one who knows what things cannot do and why not.
2.1. Basic concepts and terminology

The words *syntax* and *grammar* are often wrongly interchanged. Syntax (*sun-tacho* in Greek: to weave together) is the set of hidden dependency relationships among the words of a language, considered through their part of speech only, which makes all sequences of words not equally acceptable. Grammars are a formal way to express these constraints. One refers to *the* syntax of a language, but *many* grammars can describe it. Furthermore, grammars are not restricted to syntactic descriptions. They should be seen as tools to highlight the hidden hierarchical organization of symbolic data strings, whatever their linguistic bearing: sequences of phonemes, morphemes, words, meanings, intonation patterns. All respect a particular syntax.

Grammars really were first formalized by Chomsky [1957]. He described so-called *phrase structure grammars* as sets of *rewrite* (or *production*) *rules* of the form:

\[ \alpha \to \beta \]  

(2.1)

in which \( \alpha \) and \( \beta \) are any sequences of *symbols* chosen from the *vocabulary* (the set of admissible symbols) \( V \), composed itself of two disjoint subsets: \( V_T \), the *terminal* vocabulary (i.e., the set of all terminal symbols) and \( V_N \), the *non-terminal* vocabulary. Nonterminal symbols differ from terminal ones in that they never appear in surface forms but participate in the description of sentences. We shall write single elements of \( V_T \) as lowercase roman characters \((a, b, c, \ldots)\), single elements of \( V_N \) as upper case roman characters \((A, B, C, \ldots)\), and sequences of elements of \( V \) (*phrases*) as Greek characters \((\alpha, \beta, \gamma, \ldots)\). Rule (2.1) simply states that \( \alpha \) can be replaced by \( \beta \).

By introducing restrictions on \( \alpha \) and \( \beta \), Chomsky defined a complexity scale ranging from 3 (the most restrictive grammars) to 0 (the most including ones). Even though phrase structure grammars are less used today, at least in their initial form, this classification is still applied, for it is strongly connected to the algorithmic complexity and storage requirements of the corresponding parsers. Phrase structure grammars also have an important didactic function, in that they allow progressive introduction of the grammar terminology.

We shall systematically distinguish three complementary problems related to language processing, respectively termed as *inference*, *parsing*, and *transduction*:

1. Assuming a given complexity for the language to parse (that is, choosing the right set of restrictions on \( \alpha \) and \( \beta \)), one has to find the production rules that account for a finite subset of its surface forms, termed as the *training database*, from an initial nonterminal symbol \( S \) (an abstraction for *sentence*). This problem is known as *grammar inference*. It does not, in general, admit a single solution. In order to restrain the number of plausible grammars for a given training database, it is generally assumed that the database is *complete* for the grammar to be found—that is,

- All the terminal symbols are included, at least once, in the database;

---

2Symbols are understood here in a very wide sense: they may be characters, phonemes, morphemes, words, and so on.