INTRODUCTION
WHAT IS COMPUTATIONAL LINGUISTICS?

> The study of generative grammar developed from the confluence of two intellectual traditions: traditional and structuralist grammar, and the study of formal systems. Although there are important precursors, it was not until the mid–1950s that these intellectual currents truly merged, as ideas adapted from the study of formal systems came to be applied to the far more complex systems of natural language in something approaching their actual richness, and in subsequent years, their actual variety, thus making it possible, really for the first time, to give some substance to Humboldt's aphorism that language involves "the infinite use of finite means," the "finite means" being those that constitute the I–language.
>
> Noam Chomsky, 1986
> *Knowledge of Language*

Why must we learn a computer language in order to interact with a computer? Why can we not simply type in the information in English sentences and get the machine to respond in English sentences? What difficulties block the construction of an English–French–German translation program? The goal of this book is to make explicit some of the main problems facing any project to develop a computing machine that can read and write in English or that can translate among English, German, and French. This work is a chapter in **computational linguistics**, the general study of the problems involved in representing human language data on digital computers.

Problems in computational linguistics fall into two classes: *Conceptual problems*: What is a language? What is meant by English? What does it mean to understand English? What is a grammar of English? How does a human being know that a certain sound evokes a specific meaning? What constitutes knowledge of language? How is knowledge of language acquired? How is knowledge of language put to use? *Technical problems*: How can we represent English words, phrases, and sentences on a computing machine? How can we encode information about sentence types (indicative, question, imperative, etc.),

parts of speech (nouns, verbs, adjectives, etc.), endings (plural, tense, etc.), and a myriad of other grammatical information in a notation that a computer can digest? At the end of this study, you will not have all the answers, but you will understand the questions better.

For this study, the ideas of Noam Chomsky provide a basic conceptual framework and linguistic terminology: *grammar*, *language*, *morphology*, *derivation*, *sentence*, and *syntax*. I discuss the technical problems involved in representing English on a computing machine in a Chomsky–type generative grammar framework using Prolog. Hence, this book serves as an introduction to the concepts of Noam Chomsky's linguistic theory and as a primer for mastering the fundamentals of logic–based programming in general and Prolog in particular. Most books on the Chomsky perspective rarely discuss computational implementations, and even rarelier, present a grammar encoded in a computer language.

Because the goal is to motivate the two technical terms *explanation* and *intelligence*, my presentation is selective and not encyclopedic. In linguistics, I focus on simple lexical and syntactic facts that are glaringly obvious and anyone can understand, such as the use of *rarelier* in the above paragraph. Clearly the correct form is *more rarely*. English grammar contains *friendly/friendlier* and *early/earlier*, but not *rarely/*rarelier*. One assumes that an English speaker can differentiate between *more rarely*, a *well–formed* (*grammatical*) expression and **rarelier*, an *ill–formed* (*ungrammatical*) expression. Some researchers have directed attention to the fact that many sentences exist that are sort of in between grammatical and ungrammatical. They argue that this middle ground of semigrammatical sentences renders useless the distinction between *grammatical* and *ungrammatical*. My position is that the grammatical versus ungrammatical distinction remains useful despite the disputed middle territory and the people like e.e. cummings who live there. I simply base all decisions on obvious cases and let the computational model divide the middle ground as it will. I cannot let the obvious fact that there is a twilight obscure the fundamental difference between day and night.

In computation, I concentrate on two basic data structures. Chapter 4, on irregular verbs, shows how to represent *tables of data* in Prolog. Chapter 5 indicates how to represent a bibliography in Prolog and introduces *files with records containing variable length fields*. In these Prolog formulations, I consider almost all linguistic structures to be either tables of data or files of records with variable field length. I follow the approaches of Maurice Gross, Dominique Perrin, and Max Silberztein by assuming that the organization and structure of the lexicon can be optimally formulated as a *finite state grammar*. The linguistic data and the computational mechanisms discussed here enable me to show that UNIVERSAL GRAMMAR provides a context that enables one to see that EXPLANATION IN THEORETICAL LINGUISTICS correlates directly with ROBUSTNESS IN COMPUTATIONAL

LINGUISTICS. This study also suggests how linguistic research on the four aspects of INTELLIGENCE that underlie normal human acquisition and use of language can provide the basis for the development of *artificial intelligence* and *expert systems* in sector III (services) of the economy.

It is assumed that the reader has little understanding of the science of linguistics and no acquaintance with the universal grammar theories of Noam Chomsky. It is presupposed that the reader understands English well enough to recognize the basic facts I represent in Prolog: AMBIGUOUS SENTENCES: *You can't take religion too seriously*, for instance, is ambiguous between opposite readings. I draw on the intuitions about WELL–FORMEDNESS in English, that is, things that one can and cannot say. Consider contractions: *are not* in *You are not tall* can contract to *aren't* to yield *You aren't tall.* But *am not* in *I am not tall* cannot contract to **I amn't tall*. The contraction of *am not* is *ain't*, as in *I ain't tall.* One can say: *I am just that kind of guy*, *I'm just that kind of guy*, and *That's just the kind of guy I am*, but not **That's just the kind of guy I'm*. There are HOLES in the English orthography, and some sentences can be spoken but not written. One can say or write: *I want to go. You said too much. She is two.* One can say, but not write: *There are three to/two/too's in English.* It would be incorrect to write: *There are three too's in English. There are three two's in English.* Or, *There are three to's in English.* One can say and write: *There are three ways to write two in English: 2, II, and two*.

It is assumed that the reader has no acquaintance with symbolic programming languages such as Prolog and Lisp and has not encountered nonprocedural logic–based programming concepts such as *symbolic programming, unification, subsumption, backtracking, goal–directed searching, recursion, variable binding*, and *parallel processing.* I will define all the necessary terms and offer natural language examples to illustrate them in real data situations.

For those familiar with linguistics and Prolog, the goal of this study is to show how a parser (written in Prolog) can assign the representations (*syntactic structures*, *logical forms*, and linguistic objects at each level) developed by Chomsky and his school to sentences (*strings*). The derivations that assign structure to a string at each level are formulated as nonprocedural logic–based Prolog relations that are well–formedness conditions on representations. In brief, the REPRESENTATIONS are those offered by Chomskyian universal grammar, but the DERIVATIONAL PROCESSES that assign (or justify the assignment of) a structure to a string are expressed in terms of the nonprocedural processes of unification and subsumption and formulated as Prolog facts and relations. I sketch how to formulate a generative grammar as a Prolog program that will relate a string to its representation at the levels defined by Chomskyian linguistics.

In all cases, this presentation is expository and not polemical. I assume the version of linguistic theory offered by Chomsky (1975b, 1986b, 1987, 1992b)

and Chomsky and Lasnik (1991). I accept the definitions of UNIFICATION and SUBSUMPTION offered by Shieber (1986), and Pereira and Shieber (1987). I use the term PURE PROLOG in the sense of Sowa (1987). The Prolog relations offered here are formulated to resemble those discussed in Clocksin and Mellish (1981). This study presents only the linguistic theory of Noam Chomsky. Covington (1993), Gazdar and Mellish (1989), and Walker, McCord, Sowa, and Wilson (1987) present a variety of linguistic theories in Prolog formulations.

The best single book for those who wish to learn more about the intricacies of Prolog than is covered here is *Prolog: A Logical Approach* by T. Dodd (1990). The book presents clear and concise answers to most of the basic issues that arise in Prolog programming. One advantage is that all of the examples in Dodd's book run in Prolog–2. Not an unexpected fact given that Dodd is the designer of Prolog–2.

The organization of material here is rhetorical and moves from the simplest Prolog mechanisms to the most difficult. From the linguistic point of view, this might seem odd since it leads to treating irregular verbs (*have, be, go*) and nouns (*woman/women*) before treating the regular cases (*look, print, girl/girls*). From the Prolog point of view it makes sense because I can initially present the linguistic data structures as simple tables and the linguistic processes as simple relational database operations on memory structures. This enables me to formulate all problems in terms of searching for information in tables and to utilize only pure Prolog. Hence, I can describe problems of lexical structure in terms of relational database mechanisms that are handled by the Prolog interpreter. From the user's point of view, this means that I do not introduce any nonbuilt–in Prolog functions until the seventh chapter, and, even better, it means that the material in the first six chapters should run on any Prolog machine with no modifications except for minor punctuation. From this Prolog point of view, regular processes in language, such as verbs that add *–ed* to form the past and nouns that add *–s* for the plural, are more complicated than the irregulars because they lead to formulating Prolog relations and to defining functions.

The fact that I present a pure Prolog implementation can be important for the reader on a shoestring budget. One can download a copy of *Prolog–2 shareware Prolog* from the New York University (NYU) E–mail Public Domain Bulletin Board for free. Prolog–2 on an IBM PC, either under DOS or windows, will run all the examples in this book. Appendices I and II contain instructions on how to obtain Prolog–2 (IBM) and Open–Prolog (Apple) by ftp from NYU or on a disk via the postal service, how to set it up, and how to run the examples.

Why try to represent English, French, and German on a computer? If successful, then one will not have to learn to program in computer languages (C, Fortran, Cobol). Instead one could simply type in English sentences. If human languages were represented on a computer, then one could translate from one

language to another automatically. On a more profound note, any problem that can be posed in any formal notation or in any formal language can be posed in English. If there was a computer that could correctly interpret English sentences, then it would correctly interpret any problem and give answers in unambiguous grammatical English sentences.

Prolog is optimally suited to characterizing complex sets of interrelations among large numbers of facts, especially when some facts follow one rule, some another, and some no rule at all. Authors have illustrated Prolog concepts by selecting complex databases, including the succession of the monarchy in England, the English vocabulary indicating family relations (*mother, father, sister, uncle, ancestor,* etc.), states and countries that border on each other, the Fibbonacci numbers, and the factorial function. Most, if not all textbooks, select a variety of problems taken from various data domains. My approach is different.

One of my goals is to show that the most complex database available to any person is one's knowledge of one's native language. This often becomes clear when a student starts to learn a foreign language. The amount of data that is instantly available to a speaker of English is staggering. With rare exceptions, the only problems discussed in this text are problems of implementing the data structures of English, French, and German in Prolog. I focus on problems linguists discuss under the rubrics of SELECTION and AGREEMENT in areas traditionally called DERIVATIONAL MORPHOLOGY, INFLECTIONAL MORPHOLOGY, SYNTAX, and SEMANTICS. I define these terms in the text, but in a nutshell, I assume that readers have intuitions of GRAMMATICALITY: that is, they know the following sentences preceded by an asterisk are odd, whereas those with no asterisk are well–formed. I program a computer in Prolog to be able to make these distinctions. **John gave a book me. John gave a book to me. *Mary explained us how to go. Mary explained to us how to go. *How many advices did Sue give you? How much advice did Sue give you? *A collie is intelligenter and more tall than a doberman. A collie is more intelligent and taller than a doberman. *This door is more tall than you are. This door is more tall than wide. *This door is taller than wide. *Each of the children outnumbered the adults. The children outnumbered the adults. *Some in the group is from New Jersey. Some of the group is from New Jersey. Some in the group are from New Jersey. *I bought the book who you recommended. I bought the book that/which you recommended. *Alice is a bird of a feather. *I stacked up the book. I stacked the books up.*

I assume the reader has intuitions about semantics. In particular, intuitions about AMBIGUITY, that is, the following sentences have two or more interpretations: *John decided on the boat. Flying planes can be dangerous.* And intuitions about PARAPHRASE, that is, either two sentences have the same meaning, or if one sentence is true, then another must also be true. For instance, in a lotto game, one could imagine describing a bonus plan like this: *If the number you pick is even, you win double if it is six or eight.* But it is odd to say: *If the*

number you pick is six or eight, you win double if it is even. If *The number is six or eight* is true, then *The number is even* must be true. If *The number is even is true*, then *The number is six or eight* may be true.

If everything proposed in this study came true, and all the processes of human language discussed were implemented on the largest, fastest computer available, there would still be an enormous amount of work to do to make the computer understand every English sentence one might throw at it. If one were a 100% successful in one's endeavor, it would still require considerably more work before one could feed the computer all of Shakespeare's plays and then ask: *Was "to be or not to be" really the question?* On the other hand, consider the problems that constrained Hamlet to say what he said the way he said it. For instance, Hamlet said: *To be or not to be, that is the question.* If we think of Hamlet looking down from heaven with hindsight, he might reflect: *To have been or not to have been, that was the question.* But if we think of Hamlet in heaven before his birth obtaining information about his future life from an angel, Hamlet could not say: **To will be or not to will be, that will be the question.* Hamlet's reflective indecision was both limited and constrained by the fact that he pondered in English. In German, Hamlet's question is well–formed in the past, present, and future: *Gewesen sein oder nicht gewesen sein, das ist die Frage gewesen*. *Sein oder nicht sein, das ist die Frage*. *Sein werden oder nicht sein werden, das wird die Frage sein.*

The Prolog machine programmed with English, German, and Chinese might never know what it is to be or not to be; it might never pose a question; and it might not be able to answer some questions, but it would know that the grammatical rules of English, German, and Chinese constrain the questions that can be asked and answered in these languages. The machine would know which questions are well–formed in each language and which sentences in one language can be translated readily into the other.

There are strong reasons for programming in Prolog if the goal is to obtain a theoretical understanding of the computational processes that underlie the constructions found in English, the principles of word formation, dictionary construction, sentence parsing, morphology, and translation. The basic operations of the pure Prolog machine (SUBSUMPTION and UNIFICATION) suffice to express many of the agreement and selection phenomena of human language, in particular, those represented by the italicized sentences earlier. Prolog is nonprocedural and pure Prolog lends itself to parallel processing. Insofar as the conditions defined in Chomsky's grammar are independently applied to determine the properties of a sentence (i.e., *modular*), they can be formulated as nonprocedural and parallel processing mechanisms.

This book derives from courses taught over five years in which students learned to encode the principles of sentence construction of human languages into Lisp, C, Awk, Pascal, Setl, Fortran, and other programming languages.

Experience has shown that students produce useful programs on their own in a very short time when taught Prolog. With other languages it can take considerable time before anything interesting emerges. I believe this follows from the NONPROCEDURAL nature of Prolog. Students do not have to learn how to formulate algorithms to obtain results; instead, they learn to express the facts of the language in an explicit terminology. For instance, in programming agreement phenomena into Prolog, one need not indicate *how* the agreement is assigned, that is, whether the verb must agree with the subject or the subject with the verb. Instead, one simply writes a Prolog biconditional statement that states that there is agreement between the subject and verb.

Prolog is not so much a programming language as it is a logical notation for expressing what you already know (the *database*) and what you want to find out (the *query*). Prolog provides a special way and specific notation for conceptualizing, organizing, and presenting problems. Programming is essentially formatting the data. It is very easy to get started and to get meaningful results quickly. Naturally, it takes time to get good and write efficient programs. For the reader interested in optimizing Prolog code and in mastering the intricacies of Prolog, there are many excellent books: Bratko (1990), Clocksin and Mellish (1981), Dodd (1990), Malpas (1987), Mueller and Page (1988), and Sterling and Shapiro (1987).

This book differs from all of these above in attitude, approach, and material covered. Most Prolog textbooks offer a solution in search of a problem. They primarily cover the complex mechanisms and data structures of Prolog and secondarily apply these tools to illustrative problems. I offer a problem in search of a solution. This study is *data driven* in that I try to program information about language structure into the computer, using as few instructions as possible. I do not present some Prolog mechanism and then try to show how it can be used to describe some data. Rather, I discuss some data structure of a human language and try to encode it into pure Prolog as one or more tables of data with complex interactions among the elements of the table and between different tables. If the human language data structure cannot be encoded as relations among table entries, that is, if it cannot be considered a problem in *relational database* management, I reluctantly define a new function.

This approach leads to neglecting some often discussed Prolog functions, such as *reverse*, that converts a string like *abcdef* to *fedcba*, and *purge*, that inputs an element and a string and removes that element from the string, for example, *purge(c,abcde,X)* returns *X = abde*. I neglect many such traditionally discussed functions for two reasons: mainly because they seem to play no role in describing the structures one finds in human languages, and second, because they are abundantly discussed in most Prolog textbooks.

One point about my labeling of programs and figures might cause some confusion. The programs and figures in this book are labeled by chapter, g0803

is the 3rd program in Chapter 8, Figure 8.13 is the 13th figure in Chapter 8. The NYU CWIS bulletin board is a main source of information for students in classes at NYU. Most graduate classes meet 14 times. Most undergraduate classes meet 28 times. Materials labeled for courses can be numbered up to 28, for example, g2805 would be the 5th program in the 28th meeting of a course currently being taught at NYU. If you download Figure 9.8 or Figure 17.3, these are course materials from the 9th and 17th meeting of a current class. There is some material on the included disk which is not in this book. This is NYU course materials from a graduate class in which this book was used.

Chomsky (1986b) defined Plato's Problem and Orwell's Problem:

> For many years, I have been intrigued by two problems, concerning human knowledge. The first is the problem of explaining how we can know so much given that we have such limited evidence. The second is the problem of explaining how we can know so little, given that we have so much evidence. The first problem we might call "Plato's Problem," the second, "Orwell's Problem,"... (p. xxv)

Chapters 1-8 indicate the basic thrust of Chomsky's approach to answering "Plato's Problem" by offering a computational model of generative grammar.

A linguistic version of "Orwell's Problem" might be stated: Why are the enormous resources of the Internet – most of which are free – not widely used by researchers in natural language? To the extent that the answer is, "I do not know where to start," the next section, *How to Use This Book*, will be useful (see Appendix III for information about the Internet). This is where you start. These materials are rhetorically organized to enable researchers who do not know where to begin to take their first steps without wasting time learning arcane computer formalisms and terminology. The next section describes the facilities of the computer laboratories at New York University and shows how these facilities relate to research and teaching in computational linguistics.

I agree wholeheartedly with one of the basic ideas presented by J. Weizenbaum (1976). Weizenbaum pointed out that,

> apart from such minor, though possibly not unhelpful, phenomena as the flashing of the computers' lights and the occasional motions of their tape reels, the only evidence of their structures that the computers provide is, after all, linguistic. They accept strings of linguistic inputs in the form of the texts typed on their console typewriters, and they respond with linguistic outputs written on the same instrument or onto magnetic media....If we strive to explain computers when bounded by the restriction that we may not break the computer open, then all explanations must be derived from linguistic

bases. (p. 135)

The question asked is simple:

> **If all interaction with the computer is linguistic, then why must humans give up their natural languages (English, French, etc.) and learn artificial languages (Fortran, Cobol, C, Awk, etc.) in order to interact with the computer?**

Currently, the tail seems to be wagging the dog. Why not bring the computer around to read and write in human languages? As computer memory becomes cheaper and faster, as the speed of central processing units increases, and as parallel processing becomes available, one may reasonably expect that one of the imminent major breakthroughs in the computer culture will be computers that read, write, and translate human languages. What integrations of software, firmware, and hardware are 25, or even 10, years away we can only guess at. But, as I hope to show, any machines that read and write in English in the next few years will probably be programmed in a nonprocedural language, like Prolog, using the universal grammar ideas introduced by Noam Chomsky. Whatever the eventual outcome, **computational linguistics** is the discipline that will integrate the computer technology with linguistic theory.