Structuring the Text of the *Oxford English Dictionary* through Finite State Transduction

by

Rick Kazman [1]

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, 1986

[1]Present Address: Program in Computational Linguistics, Department of Philosophy, Carnegie-Mellon University, Pittsburgh, PA 15213

Abstract

Structuring the Text of the *Oxford English Dictionary* through Finite State Transduction

By Fall 1986 the *Oxford English Dictionary* will have been completely entered into machine-readable form as a first step toward creating an integrated version of the Dictionary and its *Supplement*. The ability to update and revise the *OED* requires the addition of a considerable amount of structure to the keyboarded text. Various software approaches to transducing the text of the *OED* in order to add this structure were evaluated, and eventually *INR* and *lsim* were chosen. The use of *INR*, a program for computing finite automata, necessitated that the structure of the *OED* be described as a regular language. The methods used to describe the *OED*, resolve ambiguities and deal with space limitations are detailed. These methods are not limited to the *OED*, but may be applied to any text in which one wishes to augment the structural information.

The contents of this paper, including statements of fact or opinion, are solely the responsibility of the author and not of the Oxford University Press nor the University of Waterloo. Financial support from the University of Waterloo, the Office of the Canadian Secretary of State (through the Centres of Specialization Fund) and from the Oxford University Press is gratefully acknowledged.

Acknowledgements

The original idea of augmenting the tagging of the *OED* by writing a grammar for it, and using this grammar to direct a transducer was Gaston Gonnet's. He had done a considerable amount of investigative work on this approach long before I was involved with the *New OED* project and provided the initial framework for a grammar.

I was tempted into the project by Frank Tompa, who shortly thereafter became my supervisor. He provided a considerable amount of moral, emotional and intellectual support from the very start of this project when the methods used were untried and the early results were not tremendously encouraging.

Another great debt is owed to Howard Johnson, who wrote *INR* and, along with Frank Tompa, co-wrote *lsim*, and who provided uncounted hours of advice, debugging, references and general good humour.

I would also like to express my thanks to the staff of the Oxford University Press in Oxford, England, who took me into their fold and their pubs for eight months, and made me feel like one of them. Special thanks go to Timothy Benbow, Alan Prescott, James Howes, Edmund Weiner, John Simpson, Dave Harrison, Blaise Machin, John Cahill and Yvonne Warburton.

The staff of the *Centre for the New OED* at the University of Waterloo—Gayle Johannesen, Donna Lee Berg and Maureen Searle—also deserve mention for their support and encouragement from the very start of the project.

Finally, I would like to thank my parents, who have never stopped encouraging me and believing in me, through all the vicissitudes of my somewhat spotted academic career.

Contents

List of Figures

Chapter 1

Introduction

1.1 What is the *OED*?

The *Oxford English Dictionary* (*OED*) [MBCO33] is the largest, most complex and most prestigious dictionary in and of the English language. The *OED* is such a monumental work that its original publication spanned over 44 years, from January 1884 to April 1928. Background work on the dictionary was, however, begun much earlier. The generally accepted conception date for the *OED* is November 1857, when Richard Trench, the Dean of Westminster, urged the Philological Society of England to produce a new dictionary which would correct the serious deficiencies which then existed in English language dictionaries. Sir James Murray, the true father of the *OED*, its main designer and its greatest editor, began his official association with the Oxford University Press (OUP) in March of 1879 (although he was unofficially associated with the *OED* project in its planning stages by 1876), but, sadly, he did not live to see the *OED*'s complete publication: he died in 1915. Even so, he personally edited over seven thousand pages, nearly half of the original *OED*.

Murray's, the Philological Society's and OUP's tasks were to collect, edit, and examine specimens of the English language and to publish a dictionary based on ***historical*** principles. In fact, the *OED* was originally called *A New English Dictionary on a Historical Basis*—this title has been retained as a sub-title in later editions. It was the historical examination of the English language and the depth of its scholarship which originally distinguished the *OED* from other English language dictionaries. Each word in the language was examined afresh, and categorized into parts of speech, homonyms[1] and senses according to its actual usage, as evidenced by the written quotations which accompany each sense of each word. The *OED*'s tremendous collection of quotations illustrating English usage, and the accompanying analyses of these quotations, has, and still does, set the *OED* apart from other dictionaries and accounts for the *OED*'s enormous proportions.

[1]Homonyms are, in the *OED*, distinct words which have the same form.

The *OED* was an unabashedly ambitious work. During the early stages of development on the Dictionary its proportions caused grave concern among the Delegates of the the Oxford University Press who could not foresee that it would ever sell in sufficiently large numbers to justify its size and cost. Initially, Murray thought that the Dictionary could be produced in as few as 5,000 pages. Upon more careful analysis of the task at hand he increased his estimate to 7,000 pages, a figure to which he agreed in his initial contract with Oxford University Press in 1879. This figure was based on the supposition that the Dictionary could be written using only four times the space of Noah Webster's *American Dictionary of the English Language*, the largest English dictionary at the time. In 1881, having worked on the letter A for two years, and faced with the knowledge that the Dictionary could not possibly be completed as proposed in 7,000 pages, Murray managed to get the Delegates to increase the upper bound on the size of the Dictionary to 8,400 pages. Nevertheless, the problem of size continued to harangue the editors of the *OED* until its completion. The editors persevered though, and managed to create an enormous work of scholarship. In the words of the last *OED* editor, C. T. Onions, in the 1933 preface to the Dictionary:

> The aim of this Dictionary is to present in alphabetical series the words that have formed the English vocabulary from the time of the earliest records down to the present day, with all the relevant facts concerning their form, sense-history, pronunciation, and etymology. It embraces not only the standard language of literature and conversation, whether current at the moment, or obsolete, or archaic, but also the main technical vocabulary, and a large measure of dialectical usage and slang ... There is no aspect of English linguistic history that the Dictionary has not illuminated; its findings have called for the revision of many philological statements and the reconsideration of many judgements on textual matters. So wide is its scope and so intensive its treatment that it has served for students, both native and foreign, as a lexicon of many languages, and, thought it deals primarily with words, it is virtually an encyclopædic treasury of information about things. [MBCO33, preface, page v.]

Since the last volume of the *OED* was published in 1928, there has been a single volume *Supplement* of additions and corrections, published in 1933, and its replacement, a four volume *Supplement* [Bur72] (originally intended to be three volumes), the first volume of which went to press in 1972, and the final volume of which will be available in May, 1986. The Dictionary, as it stands today, is 15,487 pages long, with the 1972-86 *Supplement* adding approximately 5,600 pages more. In total, it contains about 350,000 characters of text. Although its size alone distinguishes the *OED* from other dictionaries, its complexity and scholarly basis have made it the pre-eminent dictionary of the English language.[2]

[2]For a comprehensive treatment of James Murray's life and the development of the *OED*, see [Mur79].

1.2 Why Computerize the *OED*?

The decision to computerize the *OED* was motivated by the imminent completion of the four volume *Supplement*. It was painfully obvious to the editors that as soon as the *Supplement* was completed it too would be out of date. The English language is changing faster than lexicographers can presently study and chronicle it, particularly in the areas of scientific and technical words. Part of the problem of staying abreast of the language stems from the method by which the *Supplements* have been produced. Until now they have been printed using hot metal type, which must be laboriously manually set—it is likely the only (and last) major book in the world to be so produced. Even without this constraint, the editing and gathering of materials is a process which is long and cumbersome, so that merely preparing the materials for the printer has typically taken years per volume. Furthermore, even if OUP decided to publish yet another multi-volume supplement to the *OED* using more modern editing, gathering and typesetting methods, it would render the *OED* overly cumbersome to be of any use. If a reader wanted to know that he had completely examined a word's meaning, he would be forced to check three separate sources: the *OED*, the 1972 *Supplement* and the new *Supplement*. Clearly this situation is untenable, for the *OED* will certainly be with us in the 21st century, and publishing further supplements will make the Dictionary so bulky and formidable that none but the most stout-hearted will use it. The only apparent solution, to solve the problem of multiple supplements and to hasten the excessively long compilation time which supplements currently require, was to computerize the process. The hope was that computerization would also facilitate the job of the lexicographers, by allowing them direct access to other on-line databases, by providing fast, inexpensive proofs of entries, by allowing them to follow and verify cross-references[3] to other parts of the Dictionary electronically, by being able to manipulate entry text and immediately see the results, and so on. The benefits of having the text of dictionaries on-line have already been felt throughout the publishing industry, reducing the revision times of dictionaries up to ten-fold [Nor82].

In addition, a computerized dictionary is, in itself, a valuable commodity to a publisher. Many applications for computerized dictionaries have been recognized since computers were first developed. "In 1949, when the few working computers were all in military laboratories, the mathematician Warren Weaver, one of the pioneers of communication theory, pointed out that the techniques developed for code breaking might be applicable to machine translation" [Win84, page 131]. From this beginning, investigators began to examine the feasibility of using computers to translate natural languages. Although this objective has not been satisfactorily achieved, at present, computerized dictionaries have been put to a variety of experimental and, sometimes, profitable uses: spelling correction, automatic hyphenation, natural language recognition, and so on. These will be discussed in more detail in Chapter 2, but the point is that dictionaries themselves are increasingly being viewed as unique and wealthy repositories of information about language, with a great number of potential

[3]These are pointers directing the reader to another *OED* entry for further information.

applications.

The computerization of the *OED* will take place in two distinct phases. The first phase, that of entering the *OED* into machine readable format, integrating the *OED* with the *Supplement* and publishing the *New OED*, is being undertaken by the Oxford University Press. The second phase, that of creating a version of the *OED* which will be more suitable for long-term dictionary maintenance and augmentation, available to the user community as a computerized dictionary in a convenient form, is being undertaken by the University of Waterloo.

The fundamental reason to computerize the *OED* is its size. This sounds rather simplistic, but it is this characteristic, the *OED*'s main strength and greatest burden, which underlies all of the problems, past and present, with publishing the Dictionary and keeping it up to date. Given that publishing a second supplement is not a practical solution, OUP has no choice but to integrate the *OED* and *Supplement* before, or concurrent with, considering any further updates to the corpus. The integration problem is huge and would likely take decades to complete without computer aid. There are approximately 240,000 main entries in the *OED*, and 66,000 main entries in the supplement, of which about 33,000 are common to both works, and must be integrated.[4] The 33,000 *new* words in the *Supplement* are trivial to integrate, for they simply need to be sorted into their proper lexical positions among the Dictionary entries. This will be a largely automatic operation, although not fully automatic because the *OED* does occasionally violate alphabetical order among its entries. The entries which truly need to be integrated though, can be rather complex. These represent words which already exist in the *OED*, but about which the *Supplement* has something more to say. It may be that earlier or later examples of the word have been found, or that the usage of the word has changed (e.g., it has become obsolete or been revived, or is only used in some restricted sense or geographical area, etc.). It may be that the *Supplement* is simply correcting errors or oversights in the *OED*, or clarifying points of ambiguity.

Whatever the reason for the entry, the integration problem for these words is rather difficult. The integrated Dictionary must read properly, but the *Supplement* was designed to be read in conjunction with the *OED*, or on its own. Consequently many entries cannot be simply cut-and-pasted together by an automatic process with the expectation of a polished final product.

One expected benefit from the computerization is that the integration process will be greatly facilitated (although not completely automated) by automatically analyzing the structure of the entries which require integration. These entries will be combined on the basis of instructions embedded within the *Supplement* text which will be identified during the computerization process (instructions which were originally aimed at the reader), and on a number of computer-enforced rules. This automatic integration will doubtlessly save many hours of tiresome, repetitive work.

[4]The figures in this section are all taken from [New84]. They are estimates only and should be taken as only being accurate to within 25%.

Another benefit from the computerization will be cross-reference checking and resolution. There are currently 325,000 cross-references in the *OED*, with a further 100,000 in the *Supplement*. Many of these are certain to be incorrect if for no other reason than that the Dictionary was compiled over a 44 year time span. The writer of an entry in "B" which contained a cross-reference to a word beginning with an "V" could not possibly know how many homonyms that word in "V" would have, or even if the word would be included as a separate main entry or merely as a subordinate entry (i.e., a word which is related to a main entry and listed simply as a logical derivation of that main word, rather than as a distinct entry). Thus, cross-references which look *forward* in the dictionary are bound to be error-prone. For example, in the *OED* entry for "bivalve" *a.* and *sb.* contains a cross-reference to the entry "valve".[5] When the entry for valve was actually written though, it was split into two parts of speech, substantive and verb, and the substantive had two homonyms. There is no indication to the reader of the entry for "bivalve" which of these three targets is the correct one. When the computerized Dictionary text is loaded into a database though, the cross-references can (and will) be checked for consistency. Furthermore, when the Dictionary and *Supplement* are integrated, many words will have their cross-reference "addresses" changed (for example, by having a new sense or even a new homonym from the *Supplement* inserted between two existing ones in the *OED* thus forcing the old sense or homonym numbers to be altered), and every entry which pointed at the old cross-reference address will need to be updated accordingly. Finally, some cross-references are sure to be wrong owing to nothing more sophisticated than human error, and in a work of about 70 million words they could easily go unnoticed or unreported, even after 100 years of use. The cross-reference checking process is not too difficult with computer aid, but extremely tedious and sure to be fraught with error if done manually.

1.3 Why Use a Transducer?

To understand why it was necessary to use a transducer to process the Dictionary text, one must look at the history of the *New OED* project. After examining and rejecting other means of entering the Dictionary text into a computer readable medium (such as by optical scanning), it was decided that a manual system of mark-up and data-entry would be used. The precise details behind this decision are beyond the scope of this work, but many technical problems were encountered during the search for an automatic scanner owing to the huge number of special characters in the Dictionary, the large number of fonts and type sizes, the poor quality of the printed pages (resulting from broken type and "bleeding" ink), the structural complexity of the entries and the requirements of the database and integration. These needs were too numerous and varied to be met by any existing system other than manual data entry. The most serious problem, though, was

[5] We will henceforth adhere to the *Supplement* and *New OED* standard of not capitalizing headword lemmas, unless they are proper names or acronyms.

that the automatic reading systems could not understand the *structure* of the Dictionary—all information of this sort would be lost in the marked-up Dictionary, since it could not be completely reconstructed from the typography alone. A human reader comprehends the structure of a dictionary through various *implicit* means: semantics, ordering, special symbols, bracketing, keywords and fonts. One of our tasks in creating a computerized form of a dictionary is to make this structure *explicit*.

In the encoded version of the *OED*, tags for tables, mathematical and chemical formulas, the start of important sections of an entry and any printable characters which are not in the EBCDIC alphabet are given as a special code which begins with a '+', contains one or two letters or one to three numbers, and ends with a blank. For instance a paragraph mark (¶) is encoded as '+33 ', a lower-case alpha (α) is encoded as '+76 ', and a double M-dash (—) is encoded as '+108 '. In addition, a few of the tags can take a parameter. These tags will henceforth be referred to as ICC tags.[6]

In an early study of the requirements of the mark-up system it was decided that there were 41 distinct elements in the Dictionary which should, ideally, be identified.[7] When it came time to design the mark-up system, though, it was realized that this number was "a little too large for those involved in data capture, or indeed subsequent editing, to remember and distinguish easily" [Wei84b, page 1]. In the end, only 20 elements of the original 41 were identified by the ICC mark-up system. Nevertheless, the original requirements remained, and it was realized that some considerable refinement and augmentation of the mark-up was necessary.

Furthermore, aside from the lack of refinement in the original tagging, several more serious problems existed for any software which was to support the *New OED*. One problem was that *all* of the font and section tags were merely "begin" tags—they indicated where a font or section began, but not where it ended. This was a serious shortcoming of the tagging and one which, though quite simple in theory, was, in practice, rather difficult to solve. To ignore this shortcoming would have resulted in far more complex code for *every* program that dealt with the Dictionary text. In effect, each program would have to solve the end-tag problem afresh. A second problem was that most of the ICC tags reflected the *typography* of the Dictionary, while ignoring its structure. This problem is common to computerized dictionary projects:

> [Typographical] information is required by the composing machine which has to translate the data into type, but it is not particularly useful for rational

[6] See Appendix A for a complete list of tags used in encoding the *OED*. In consultation with OUP, these tags were designed by International Computaprint Corporation (ICC) of Fort Washington, Pennsylvania, which has been contracted by OUP to enter the *OED* into a machine readable form. The project is greatly indebted to ICC, whose employees performed the superhuman task of entering over 300,000,000 characters in one and a half years, with very few errors. Examples of problems with the transduction of the text given throughout this thesis are in no way intended to be criticisms of ICC or its employees.

[7] These elements were identified in [Wei84a].

> dictionary editing. For this purpose, coding on the basis of the function of the word in the dictionary would be far more useful, and this coding could easily be translated into typographical codes when the book was to be printed, since there is a direct mapping from function to typography but not *vice versa*. [Nor82, page 216]

The job of augmenting the tags seemed to be a natural task for a transducer, although it was not clear, at first, whether a finite state transducer was conceptually powerful enough to do the job. We were also unsure of how to go about implementing such a transducer, since most of the research and all of the commercially available parsing packages are tailored towards context-free computer languages. We wanted to avoid writing an application specific parser, if at all possible, but the available parser-generators were not capable of handling a language sufficiently complex to specify the *OED*. After some experimentation, frustration and soul-searching we settled upon *INR*[8] and *lsim*[9] to do the job. This approach has turned out to be quite satisfactory.

1.3.1 Why *INR*?

INR, while a parser-generator, is quite unlike the commercially available products; this was its greatest asset for us. Because it was not designed for computer languages, it did not have any built-in assumptions or limitations specific to computer languages. *INR* simply takes a grammar consisting of regular or rational expressions and converts it into an automaton.[10] A rational expression is defined as follows:

> Suppose that we have a regular language L over some alphabet Φ and two morphisms α and β defined on $\Phi\star$. For each word in L we will consider the pair $(\alpha(w), \beta(w))$ as being in the relation. We will call any binary relation that can be described in this way a rational relation. [Joh83, page 20]

INR is a general-purpose tool, not particularly tailored to any one application and designed to be free from arbitrary constraints. This was of paramount importance, since in all of the other approaches that we tried, we ran into "hard-wired" limitations: exceeding table limits, memory limits, stack sizes, or simply using a tool in a way that it was never intended (and so, rather inefficiently). Furthermore, since *INR* was developed locally, we

[8] *INR* is a prototype tool developed by J. H. Johnson of the Department of Computer Science at the University of Waterloo, for approximate string matching. It accepts input in the form of regular expressions and computes their non-deterministic finite automata. It is described in [Joh84], an unpublished paper. A technical report, available from the University, is expected to be published soon.

[9] *lsim* is a program developed by J. H. Johnson and F. W. Tompa of the Department of Computer Science at the University of Waterloo for the *New OED* project which acts as a transducer for a 2-tape automaton. The basic algorithm for *lsim* was designed by Johnson, and implemented and refined by Tompa.

[10] See [GL70] or [HU79] for good introductions to finite automata and regular languages.

could make changes to the software, when faced with problems. *INR* did, in fact, have some limitations in practice, but these coincided fairly closely with the limits imposed by the host computers and operating systems under which *INR* was running,[11] and so raising these limits (which we could have done without too much difficulty, having access to the source code) would have gained us little. Furthermore, when *INR*'s limits were being approached, it was an indication that the grammar was being designed inefficiently, and that significant mental effort and physical re-organization of the grammar was in order.

It was discovered, after some months of experimentation, that most of the *OED*'s structure could be represented as a rational language, and hence as a deterministic finite automaton, and so *INR* could be used to create the automaton which would transduce the marked-up text. We will discuss this process in detail in Chapters 3 and 4.

Another benefit of using *INR* is that the resulting automaton, although just a data structure, implicitly dictates how it is to be executed. Each row of the automaton consists of the triple: from-state (source), symbol (preceded by its tape number) and to-state (destination). As a trivial (but readable) example, the single tape automaton in figure 1.1 accepts strings starting and ending with an "a" and containing an even number of "b"s. The automaton reads as follows: the first item in each line is the state number, the next item is the symbol (preceded by a tape number if the automaton has more than one tape) and the final item is the transition state. The state numbers (START) and (FINAL) are special symbols for the start and end states of the automaton respectively. Thus, in the example automaton, if we are currently in state 2 and see an "a" on tape 0, we move to state 3. If we see a "b", then we move to state 4. The –| symbol indicates end-of-file, so if we are in state 3, then we can immediately reach the final state, after verifying that there is no further input.

State	*Symbol*	*Transition*
(START)	a	2
2	a	3
2	b	4
3	–|	(FINAL)
4	b	2

Figure 1.1: An Example of an Automaton for "a (bb)⋆ a"

[11] The systems which currently support the transduction process are a VAX 11/780 running 4.2 BSD UNIX and an IBM 4341 running VM/CMS.

In practice, the execution of automata is quite efficient and the program, *lsim*, which we used to simulate the automata, was developed in a relatively short period of time, although it was the third approach to automaton simulation which we tried (the other approaches failed because of the huge number of states needed to simulate the automaton). For these reasons—flexibility, generality and efficiency—*INR* has proven to be an excellent choice of software for describing a transducer to parse the text of the *OED*.

1.4 Thesis Outline

The remainder of the thesis will be organized as follows: Chapter 2 will outline the background needed to approach the task of transducing the *OED* and will detail some related work in computerized dictionaries and the transduction of text; Chapter 3 briefly describes the tools considered for the transduction and the tools actually used to transduce the *OED*—*INR* and *lsim*—how they work, how they were used, what their limitations are and the results of the transduction process; Chapter 4 takes an in-depth look at the process of writing a grammar for the *OED*, and the process of coming to understand the structure of the Dictionary; finally, Chapter 5 provides some conclusions on the effectiveness of finite state transduction for structuring the *OED* along with some suggestions for future improvements.

The three appendices provide a list of the tags which ICC utilized in marking up the *OED*, two sample grammars used in the transduction process and two text proofs, created from the transduction of marked-up ICC text.

At this point, a word of caution is in order: throughout this thesis, many examples of grammar fragments are given. No claim is made that these fragments are truly representative of the actual grammars used in the *OED* project, or that these fragments would truly recognize or transduce the *OED*. The fragments are always drawn from past or present grammars, but have been tailored and simplified for the purpose of exposition. Furthermore, the complete grammars, given in Appendix B, represent the state of the grammars as of August, 1985. The grammars are undergoing continual refinement at Oxford University Press. The examples of Appendix B are merely given as illustrations of *complete OED* grammars.

Chapter 2

Background

2.1 Machine Readable and Computerized Dictionaries

Computerized dictionaries have been the subject of ever-increasing interest and awareness for the past two decades.[1] Researchers have come to recognize that dictionaries contain a unique cornucopia of information relevant to natural language understanding, speech generation, machine translation, spelling, style correction and content analysis of text, and many other areas. The terms machine-readable dictionary and computerized dictionary are often used interchangeably by writers on these subjects, but Michiels [Mic81] makes the useful distinction that a machine-readable dictionary is simply a dictionary which has been encoded in some machine readable form, whereas a computerized dictionary is the result of structuring and organizing this information.

2.1.1 Machine Readable Dictionaries

Almost every major dictionary publisher in the world has one or more dictionaries in machine-readable format, and many are currently planning to create computerized dictionaries from these. The reason for this is simple—almost all currently published books are typeset by computers [Nor82], so the simple act of typesetting a dictionary creates a machine-readable form as a by-product. For many years, publishers and printers did not appreciate the value of their typesetting tapes, and they remained unexploited, except as templates for newer editions of the dictionary. To this day, typesetting is the driving force behind the creation of machine-readable dictionaries, but typesetting information alone is of little use when attempting to create a database from the dictionary text. Consequently, a considerable amount of structuring must be done on these corpora in order to create computerized dictionaries from them.

[1] Much of the material in this section is drawn from Robert Amsler's review of machine-readable dictionaries [Ams84] and Barbara Ann Kipfer's review of computational lexicography [Kip84].

The best known machine-readable dictionaries in and about English are the *American Heritage Dictionary*, the *Random House Dictionary*, the *Oxford Advanced Learner's Dictionary of Current English*, the *Longman Dictionary of Contemporary English, Collins English Dictionary, Webster's Seventh Collegiate Dictionary* and *Webster's Eighth.* In addition to this, almost every large, historical dictionary has an ongoing computerization project: these include the *Dictionary of Old English* [dH85], the *Dictionary of the Older Scottish Tongue*, the *Dictionary of the Old Spanish Language*, the *Dictionary of American Regional English* [Cas80], the *Trésor de la Langue Française*, and, of course, the *OED* [Wei85b].

Machine-readable dictionaries are compiled from one or more of the following techniques: manual keyboarding of the original text, optical scanning of the same, extracting citations from machine readable sources, and keyboarding hand-written dictionary slips (pieces of paper containing quotations illustrating a usage of a word). Each of these techniques has its peculiar problems, and considerable proof-reading and correction of newly compiled machine-readable dictionaries is required in order for them to represent their sources accurately.

2.1.2 Computerized Dictionaries

Computerized dictionaries, while not common, are becoming so rather rapidly, motivated by economic and academic pressures. Many machine-readable dictionaries have been transformed into fully structured computerized dictionaries, although a considerable amount of human editing has typically been necessary, particularly in the pioneering efforts, such as *Webster's Seventh* [OR72]. More recent projects, such as the *Dictionary of Old English* at the University of Toronto and the *OED*, rely quite heavily on computers for sorting, resolving cross-references, formatting, editing and displaying text, search and retrieval, and text manipulation. However, none of these projects currently utilizes the full potential of computerized dictionaries—as replacements for and enhancements of the printed books. There are, however, several computerized encyclopedias [WB85] which have made good use of the medium. Fox, et al. [FBP80] outline some of the requirements of such systems and address many of the issues with which designers of computerized dictionaries will have to deal in the near future: size, cost, user interface, physical components, scope and format.

Phase 2 of the *New OED* project will also have to address many of these issues. Currently, researchers are contemplating and investigating the design of a database to support the computerized *OED*, the storage media needed (conventional disk versus optical disk), the prospective markets (home versus business or educational) and the packaging and marketing of the product (selling personal copies of the Dictionary versus charging for time on a centralized time-shared system) [ST84,Tom86].

2.1.3 Currently Available Corpora

Computer readable corpora are specially prepared collections of machine-readable text. These texts are entered into machine-readable form for one of the following reasons: so that they can be studied (with computer aid) as examples of natural language, or so that they can be created, printed and distributed electronically. The first type of corpus is far less common than the second type, since it does not offer immediate financial returns. The number of computer readable corpora is quite large—too large to mention more than a representative sample here.

The earliest and best known corpus created for study, is the *Brown University Corpus of Present-Day Edited American English.* It is a corpus of approximately one million words, composed of 500 carefully selected samples from a wide variety of sources from the year 1961. This corpus was used as a basis for many studies of English usage and has recently been augmented by tagging *every* word with its grammatical class [Kuv82b]. Houghton Mifflin used this corpus in creating the *American Heritage Dictionary* in 1969. So successful was this project, that Houghton Mifflin created their own corpus, the *American Heritage Intermediate Corpus.* This was a five million word collection of extracts from texts relevant to students from grades three to nine. This corpus was used to prepare the *American Heritage School Dictionary.* Clearly, the way of the future had been marked, and a great deal of lexicographic attention has since been paid to text corpora and statistical analyses of English derived from them.

Many other dictionary projects have compiled quite impressive corpora to support their lexicographic activities. The *Dictionary of Old English* and the *Dictionary of the Older Scottish Tongue* both have their *entire* corpora on-line. This is possible because their corpora are relatively small (the Old English one is about three million words, occupying forty megabytes of storage) and fixed—new finds of Old English or Scottish are rare. The *Trésor de la Langue Française* project, at the University of Nancy, has the complete texts of all monolingual French dictionaries stored on-line.

Many institutions, other than those involved in dictionary production find it worthwhile to create and maintain large machine-readable lexical data banks. The Institute for Scientific Information in the United States has, as one of its databases, a 450,000 word list (as of 1980) of terms taken from the titles of journal articles. This list is growing at the rate of 500 to 1,000 newly coined terms per year. The translation bureau of Canada maintains a large bilingual (English/French) data bank to support one of the most active translation projects in the world. McNaught [McN82] gives a good overview of the state of linguistic data banks in Europe.

Corpora produced on and for electronic media, have also had a large effect on dictionary production since they represent a huge fund of easily accessible samples of language usage. More importantly, they represent "ordinary language" more closely than the literary sources historically referenced as exemplars of English usage. These corpora fall into

several categories: news services such as the *New York Times News Service* or the *Associated Press*, computerized public information services such as *The Source* and *CompuServe*, computerized encyclopedias, such as *Encyclopedia Brittanica* and the *Academic American Encyclopedia*, and commercial information retrieval systems such as *LEXIS* and *NEXIS*.

2.1.4 Uses of Computerized Dictionaries

Computerized dictionaries simultaneously represent a challenge and a great source of hope for computer scientists and lexicographers alike. For computer scientists, having large, powerful, well-structured dictionaries on-line reveals new vistas of possibilities for intelligently dealing with text. For lexicographers, computerized dictionaries and other on-line corpora give them the power to make their dictionaries larger, more consistent in style and content, more specialized or general, as desired, more frequently revisable, more representative of actual word usage and more sensitive to the changing needs of users.

Machine translation provided the first motivation for the use of computers with dictionaries and in lexicography in general, with the Georgetown University Machine Translation Research and Language Projects in 1956. To this day, computerized translations are of poor quality and must be manually post-edited, however the increasing power and availability of unilingual computerized dictionaries in *every* major language as well as bi- and multi-lingual computerized dictionaries should help to improve the performance of machine translation systems. Not only will the machine translation systems cover a greater scope of language, but they should be able to understand the structure of languages better by having ready access to paradigms of word usage, semo-syntactic codes associated with words, and so on. Machine translation systems are needed most desperately for translation of scientific and technical documents—one machine translation project has recorded a three-fold increase in the volume of such translation from 1974 to 1980 [OT82]. Fortunately, in recognition of the speed with which scientific and technical terms are coined, a great deal of work is being done in the area of terminology banks [McN82,Ams84]. Given the general quality of translation in scientific and technical documents currently, along with the heavy emphasis on specialized language (which tends to be more universal than natural languages as a whole), the prospects for machine translation seem quite bright.

Without doubt the widest application for computerized dictionaries at the current time is as automatic spelling-checkers and guides to hyphenation. These uses are discussed in [FBP80,Kuv82a,Pet82,Rob83,Ams84,Knu84,Win84]. These are often included as functions of a word processing or typesetting program, but may just as easily stand on their own. Spelling checkers typically refer to some form of word list in order to test their input, rather than consulting a complete dictionary. Since current systems use only a word list, they are oblivious to context. For this reason, if a user makes a spelling mistake which creates another correctly spelled (but syntactically or semantically incorrect) word, current spelling checkers will not report this word as an error, because they have no notion of the word classes and grammar of the language which they are checking. Whether the system

uses word lists or a complete dictionary, it needs methods of prefix and suffix resolution and inflection handling to work properly (since dictionaries typically do not list all inflections of a word, and never include all possible prefix and suffix combinations). Hyphenation programs can similarly consult complete dictionaries or use algorithms detailing how to hyphenate an English word properly. The latter method is more efficient, but still needs to refer to a dictionary of exceptions and a specialized dictionary of letter combinations [Knu84, pages 449–455].

An obvious and pressing need for computerized dictionaries is in the area of natural language recognition. Winograd [Win84] identifies two areas where computerized dictionaries are needed in natural language comprehension: as a fund of information of the grammatical class, case and number of words for programs which perform morphological, lexical and syntactic analysis of language; and as a repository of semantic information for the semantic analysis phase of natural language understanding.

Computerized dictionaries are also used in programs which function as grammar and style critics of prose, such as the *Writer's Workbench* [MFGK82], *EPISTLE* [2] [HJBC82] and the *Electric Webster* [Rob83]. These programs provide a variety of services to the writer, from checking spelling and grammar, to searching for split infinitives, awkward prose and hackneyed phrases, providing readability measures such as average word and sentence length, frequency of sentence types (simple, compound etc.), and so on.

Walker [Wal85] describes two tools being developed at Bell Communications Research which rely heavily upon dictionaries and other "knowledge resources". The first of these tools, *FORCE4*, automatically analyzes the content of *New York Times News Service* articles to determine the primary subject of the stories. This process is sometimes called automatic indexing. The second tool, *THOTH*, assists a reader by automatically elaborating text as it is being read—it attempts to determine the important *concepts* of stories and elaborates upon these concepts at the reader's prompting. Both of these tools are aimed at giving the reader more powerful tools for accessing the content of stories, and they rely heavily upon machine-readable dictionaries.

Very briefly, some other potential uses of computerized dictionaries include: the creation of word games such as crossword puzzles; as a teaching tool, aimed at developing reading and writing skills and increasing vocabulary; as pronunciation guides for speech generation; and for the playing of word games.

[2] *EPISTLE*, an IBM product, is now know as CRITIQUE [Ric85].

2.2 Transduction of Text

2.2.1 Machine Readable Text Parsing

The vast majority of currently available machine-readable dictionaries are in a format equivalent to the ICC tagged form of the *OED*—that is, they are tagged with mainly typographical, rather than structural, information. Not only this, but the software that manipulates the text of these machine-readable dictionaries is typically ad hoc [Nor82]: it makes no attempt to understand the overall structure of the dictionary, but concentrates on a particular task, such as lemmatization, field conversion or error checking. In addition, it seems that the dictionaries which have been fully structured have been done so at the cost of considerable manual labour [Urd66,OR72].

Several dictionary projects, took a middle-ground approach to structuring a machine-readable dictionary. That is, their approach was not quite ad hoc, but they did not attempt to write a complete grammar of their dictionaries either. One of these, a project to computerize a English-Swedish dictionary published by Essette Studium, used a program which analyzed the typographical codes relevant to a word, as well as the word's position within an entry, to determine its function. This method was costly and limited, identifying only seven discrete structures within an entry. Reichert, et al. [ROP69] similarly created a program for partially parsing the machine-readable forms of two of Merriam-Webster's dictionaries in one of the pioneering efforts of machine-readable dictionary creation.

Nagao, et al. [NTUT82] used what they call a "universal data translator/verifier" based on an augmented transition network to transduce the texts of two medium-sized (60,000 entries) works—a Japanese-Japanese and a Japanese-English dictionary. An augmented transition network is a finite network of states, connected by arcs labelled with words or lexical categories. With each arc may be associated a set of conditions, restricting the circumstances under which an arc may be followed, and a number of actions which are special-purpose routines used for building structures and controlling output.[3] The definition of the augmented transition network in the Nagao project was as a set of triples, consisting of: <*condition, action, next-state*>. The *conditions* are either character sets or sequences to be matched by input. The *action* is a user-written function which is executed when the condition is satisfied. These *actions* usually manipulate buffers and perform output. The *next-state* specifies the next tuple to which control is transferred after the current tuple is satisfied.

The methods, objectives and benefits from using a data translator/verifier to transduce the text of a dictionary closely resemble those found in our use of *INR/lsim*. The data translator/verifier was able to structure the text of the dictionaries automatically and check for some errors in the input (many of which human proof-readers would typically miss). The augmented transition network approach differs from our approach with *INR* in several

[3] Augmented transition networks are described at some length in [Win83, Chapter 5].

important ways. The structure of the augmented transition network is not represented as a grammar, but rather as the totality of all *states, conditions* and user-written *actions*. In fact, the language describing the augmented transition network may be regular, context-free, or some more powerful language, depending on the complexity of the *actions*. In addition, the efficiency, and hence performance, of the augmented transition network depends heavily on the complexity and efficiency of the user-written *actions*. This seems to be borne out in practice: in [NTUT82], the authors report that execution of the data translator/verifier was very time consuming—and they were dealing with dictionaries which are certainly less complex than the *OED*.

The use of parsing to add structure to text is not confined to dictionary projects. Weyer and Borning [WB85] used a parser that recognizes the hierarchical structure of articles in the *Academic American Encyclopedia*. This allows them to model the hierarchical aspects of the articles in their prototype electronic encyclopedia project. The parser recognizes the entry structure from the format of headings within the article, and builds a hierarchical description of the article from the input. It also recognizes and identifies abbreviations, cross-references and measurements.

2.2.2 Applications of Finite State Automata

Regular languages and finite state automata have been used in an enormous variety of applications. An attempt to list them here would be futile. Aho and Ullman [AU77, pages 118-120] give a good overview to the wide variety of areas in which one may utilize automata in their section on "The Scanner Generator as Swiss Army Knife".

A few applications of finite state automata to the processing of text are worthy of mention however. Applications of finite state automata to pattern matching in general are quite common. Aho and Corasick [AC75] detail an algorithm for searching for keywords in text strings using finite state automata. Rather than adding some sort of tagging to matching strings as we do for the dictionary, they choose to output only the location of every matching string. Johnson [Joh83] has described a model for transducing inaccurate strings to produce perform approximate string matching in order to minimize the effects of inconsistency in recording these strings. This model has since been implemented using *INR*.

2.3 Grammatical Inference

Hunt [Hun75, pages 144-167] describes a pattern recognition task, which he calls the "Grammatical Inference Problem". This problem is closely allied to the problem of attempting to discover a grammar which describes the *OED*. Very briefly, the problem is defined as follows: one has some input data, in the form of a set of strings; each of these strings is

either identified or rejected by some unknown grammar; we then apply some procedure for sorting the acceptable strings from the unacceptable ones, and this tells us something about the nature of the unknown grammar.

Hunt defines two models of grammatical inference problems. In one model, we have samples which contain both positive and negative sentences–that is, sentences which we know are in the language, and those which we know are not–and in the other model, we have only positive sentences. The first model is called "informant presentation" and the second is called "text presentation". Text presentation is analogous to the process of learning the structure of the *OED*, since we did not have available to us any ***negative*** sentences, that is, any *OED* entries which were not valid. We had only positive sentences, the text of the *OED*, and a rough idea of the possible range of negative sentences derived from our discussions with OUP's lexicographic staff. Hunt gives another example of this sort of problem, namely, attempting to write a grammar for a dead language.

Hunt defines some ***theoretical*** ways in which one can arrive at a correct grammar of a text presentation language, but these involve, in the worst case, looking at every valid sentence within the language. Hunt does not, unfortunately, offer any practical solutions to the grammatical inference problem for text presentation languages.

Chapter 3

Creating a Transducer

3.1 The Early Approaches

3.1.1 *YACC* and *Lex*

When it became apparent that it would be necessary to transduce the *OED*, it was realized that there was no pre-packaged piece of software which was an immediate candidate for the job. One of the most obvious places to look, when searching for a tool to aid in the transduction process, was at the commercially available parser-generators for computer languages, since the task of parsing a computer language was superficially similar to the task of transducing the *OED* and the parser-generators were well researched and commonly available. For this purpose, we turned to what are likely the best known and most widely used products of this type: *YACC* [Joh75] and *Lex* [Les75].

We attempted to use each of these products, both together and separately and they seemed, based upon a relatively brief acquaintance, to possess sufficiently powerful syntax to describe the *OED*, but failed to produce working transducers from the input grammar. In both cases, the grammars for working transducers would have been so complex, that they would have been extremely difficult to modify or verify. *YACC* failed because it required the input grammar to be LALR(1), while *Lex* failed because of exceeding its inherent size limitations.[1] Since these were complex and widely distributed pieces of software, we did

[1]LALR stands for "lookahead left to right". The "1" refers to the number of input symbols ahead of the current one needed to disambiguate structures in the grammar. These types of grammars are described in [AU77]. However, the *OED* needs to be described with a LALR(k) language, which *YACC* can not easily handle. One possible solution to this difficulty would be to tokenize the input, using *Lex*, however this was attempted and resulted in an unnatural split of the grammars, which obscured the true structure of the *OED*. *Lex* by itself can easily deal with an arbitrary amount of lookahead, but it had other problems: it optimizes its transition tables for inclusion in a compiler, which resulted in tremendous size increases in the *OED* grammars; and repeated use of its lookahead feature leads to exponential time behaviour (in terms of the lookahead).

not feel that we could justify either attempting to modify them or requesting modifications to be made to them, so we abandoned these products as potential tools for the project.

3.1.2 *Maple*

Maple[2] is a mathematical manipulation language. It was designed for, but not limited to, the symbolic computation of mathematical expressions, and it has a general high-level input language. There were three approaches taken to the transduction of the *OED* using Maple, along with a special mechanism to deal with fonts, as a tool.

The first approach was to write a compiler-compiler in Maple which would accept, as input, a general context-free language with the addition of a special mechanism to deal with fonts. This would then produce a C program as a transducer. This approach was successful to the extent that it would produce a working transducer, but the transducer ran in $O(n^3)$ time in the size of the entry, which was unacceptably slow (since it had to process a total of about half a gigabyte of data, of which the largest single entry is nearly 500,000 characters).[3]

The second approach taken using Maple was, again, as a compiler-compiler, but this time it translated an input grammar into a minimized deterministic finite automaton. This, too, had mixed success. The main problem with this approach was that Maple took an excessively long time to create the minimized DFA, for even a subset of the grammar. The resulting automaton worked well as a recognizer and ran very quickly but was not capable, without modification, of being a transducer.

The third approach was to use Maple to create an automaton, but not to attempt to minimize it. Instead, while the automaton was running, the simulator was to keep track of a set of current states. Once this had been computed, a second pass would be made on the already accepted input and the productions that were used in the accepting phase were used to create a transducer. This approach was never completed, although it looked viable, because by that point we were having success using *INR*.

3.1.3 *Prolog*

The Prolog language was suggested as a feasible tool for creating a transducer because it is relatively easy to convert a grammar written as regular expressions into a Prolog program. Furthermore, the resulting program can be directly executed with available software, i.e., a Prolog interpreter.[4] In fact, it is quite simple to have Prolog do the grammar transformation

[2]Developed by K. O. Geddes and G. H. Gonnet of the Department of Computer Science, University of Waterloo.

[3]These figures are based upon [Wei85a], and upon our own estimates.

[4]The research investigating the feasibility of Prolog was conducted by Rob Veitch of the Computer Systems Group at the University of Waterloo.

as well—this is precisely what was done. A Prolog program was written which could read a grammar as input and create from it a semantically equivalent Prolog program. The program/interpreter combination formed a depth-first search parser with backtracking and infinite look-ahead. Once the grammar could be successfully transformed into a Prolog program it was tested using *OED* input text.

This first product of the research was only a recognizer, however it provided a simple method of evolving into a transducer. Once the input had been recognized, the Prolog program saved the parse-tree that it had used and consulted a list detailing which rules produced output and which did not. Knowing this, it made a second pass over the input, this time creating output.

Unfortunately, this approach ran aground on the same problems which had plagued the other approaches—time and space—and these problems occurred with only a subset of the full *OED* grammars. It was believed that a tokenizer would speed things up somewhat, but the space problems would still remain. In the end, the Prolog approach was not deemed to be viable, so the tokenizer was never implemented and further research was abandoned.

3.1.4 *Macros*

Another approach attempted was to describe a mapping from a regular language to a PL/S (a derivative of PL/1) program.[5] The basic idea was that each operator in a regular language could be described by a semantically equivalent structure made up entirely of PL/S statements. Each of these structures was simply a skeleton with a name. The skeletons were implemented as macros, so that a PL/S pre-processor could replace the macro name with the appropriate PL/S statements interleaved with the "arguments" of the macro. For instance, the optionality operator (?) in a regular language could be implemented as a macro called "maybe", and where one would write "Label?" in an *INR* grammar, indicating an optional label, one would write "maybe(Label)" in the macro language. In this way it could convert a grammar written in a regular language into a PL/S program.

This approach seemed to be successful in recognizing the *OED*, although it was not tested rigorously. It was never tested as a transducer, again, because of the success with *INR*, but appeared to be a feasible approach.

3.2 *INR* and *lsim*

3.2.1 *INR*

INR, as stated in Chapter 1, is a program which reads an input grammar, consisting of one or more generalized regular or rational expressions, and computes the deterministic

[5] This approach was conceived and developed by Brian Marks of IBM, UK Ltd.

and/or non-deterministic finite automata which describe them. *INR* was used to create the automata which are intended to describe and transduce the *OED*. Two of the grammars which accomplish this task (pass 1 and pass 2 of the Dictionary grammars) are given in Appendix B.

We will not here attempt to provide a full description of *INR*'s capabilities but will concentrate on that aspect of the software which was useful in creating the *OED* grammars—the language for specifying generalized rational and regular expressions. In the *OED* grammars, each rule begins with a rule name or, in *INR* terms, a variable or automaton name. After the rule name there is an equals sign which indicates that the automaton to the right is assigned to the name on the left.[6] All expressions end with a semi-colon. The most common operators in *INR* have their commonly understood functions in set notation or the notation of regular expressions: the minus (-) operator indicates set difference; the plus (+) and star ($\star$) operators mean Kleene + and $\star$; parentheses indicate groupings; the or-bar (|) indicates alternation; curly brackets {} enclose a set, the members of which are separated by commas (which, thus, also mean alternation in this context); and a question-mark indicates optionality. Parentheses, unfortunately, have a second meaning which will be explained shortly. Comments, incidentally, are allowed within *INR*, so long as they are preceded by a number sign (#). The # means "ignore all text up to and including the next carriage return". *INR* supports many other operators useful in describing regular expressions such as set intersection and symmetric difference (exclusive or), to name but two, but these will not be discussed here, as they were not used in constructing the *OED* grammars.

The above operators are sufficient to describe a language based upon generalized regular expressions, and most of the *OED* can be described this way. However, in order to support the transduction process, we must be able to define *rational* expressions as well. *INR* is a tool for creating generalized rational expressions, as well as regular ones, thus it supports the notion of tapes within an automaton. In *INR*, symbols may be preceded by a digit, followed by a period, to indicate the tape to which they refer. Symbols which are not preceded by an explicit tape number are assumed to refer to tape 0.

A common shorthand notation for referring to tapes within automata is through tuple formation. Tokens or automata enclosed within parentheses and separated by commas are formed into a tuple, and their tapes are numbered as follows: the tapes of each automaton in the tuple begin their numbering at the first number greater than the highest tape number of the preceding automaton. An example should help to clarify this process. If we have two automata, A1 and A2, each of which has two tapes (numbered 0 and 1), then when we form the tuple (A1,A2), it has *four* tapes, and the tapes of A2 within the tuple are re-numbered 2 and 3 (the *third* and *fourth* tapes respectively). In the *OED* grammars, this technique is regularly used to create a 2-tape automaton from two 1-tape automata.

[6] Assigning an automaton to a variable name is not a requirement of *INR*, but is convenient to use, because large and/or frequently referenced sections of the grammar may be referenced by this name and used as building blocks of the grammar. In theory, a grammar could be written as a single rule, but in practice this would be too complex for any but the most trivial examples.

As stated earlier, in order to describe the structure of the *OED* concisely and simply, we have made use of some of the more advanced features of *INR*. These features are used in situations where we must make semantic decisions based upon context, keywords or look-ahead.

One such feature is the ElseOR operator (||)—also called the Extending Or. The ElseOR is useful in multi-tape automata where some alternatives in the grammar are to be taken in preference to others, causing different output to be produced. In *INR*, as in set theory, no order is specified among the elements of a set, or structures separated by or-bars (which are semantically equivalent). However, in some cases the semantic needs of the grammar dictate that a particular structure should be tried first, and only when it does not adequately specify the input, should the transducer use another structure. The ElseOR provides this function.

Another feature of *INR* which is useful in specifying a complex grammar is the active complement (:acomp) operator. It is useful because it provides the ability to accept text up to some specified delimiter. Specifically, using :acomp allows one to specify the language consisting of all strings in the *complement* of whatever language is given as its left (or prefixed) operand. The complement is constructed over the active alphabet, that is, the set of letters used in the operand. So, an expression of the form:

(Alph* {a,b} Alph*):acomp

where "Alph" is a previously defined alphabet (say, all lower-case roman letters), would accept any string consisting of lower case letters, but not containing an "a" or a "b". The string may, of course, be null, unless the null string is explicitly included in the set with "a" and "b". In effect, the operand of the :acomp operator specifies the grammar of all strings which are *not* valid. All other strings composed of characters from Alph will be successfully matched.

The final important feature of *INR* for the *OED* grammars is the composition operator (@). This operator takes both a left and a right operand, and joins the last tape of the left-hand m-tape automaton with the first tape of the right-hand n-tape automaton. The resulting automaton will have $m+n-2$ tapes, and will be constrained as a result of the join. In effect, the resulting automaton is constrained to those states which satisfied both the left and right operands. Its effect is analogous to that of a natural join in a relational database, or, in operating system terms, it is as though the output from the left-hand automaton is routed to the right-hand automaton's input stream. A small grammar is given in figure 3.1, to illustrate the effect of composition. As a word of introduction, *INR* grammars are all "declaration before use" i.e., a rule must be defined before it can be referenced by a later rule. A rule may not refer to itself either. In all of the example grammars in this paper, the last rule given is the final product of the grammar.

This grammar's rules enforce the following constraints: Rule1 accepts any string made up of zero or more characters from Alph, followed by a single "c"; Rule2 accepts any

```
#
# This grammar gives a simple example of the use
# of the composition operator @.
#
Alph    =    {a,b,c,d,e,f,g,h,i,j,k,l} ;
Rule1   =    (Alph* 'c') $ (0,0) ;
Rule2   =    (Alph - {'a','b'} | 'aa' | 'bb')* $ (0,0) ;
Final   =    (Rule1 @ Rule2) ;
```

Figure 3.1: The Composition Operator—Simple Example

combinations of characters from Alph, any number of times, but the letters "a" and "b" must occur in pairs. Thus, Rule1 accepts strings of the form: "afdjlefkifcfc", "aabbc" or even simply "c", and Rule2 allows strings of the form "" (the null string), "dhejfjggelbbf", and "aabbaaaaffaa". When the two rules are composed into the rule Final, only two of the above strings would be accepted: "c" and "aabbc", because only these two strings end in "c" and have "a" and "b" in pairs (if they occur at all). Both Rule1 and Rule2 include the construct "$ (0,0)". The $ means "projection" in *INR*: the automaton defined to the left of the $ is projected onto the tapes defined by the right-hand operand, in this case the tuple (0,0), which means (tape 0, tape 1)—recall the earlier discussion about tape renumbering within a tuple. The effect of the "$ (0,0)" is to take a one-tape automaton—an acceptor of a language—and to make a two-tape automaton out of it which copies valid input strings to output.

The above example, while being a reasonable use of composition, could have just as easily been achieved by other techniques within *INR*, namely by either directly specifying a grammar which ends with a "c" and contains the letters "a" and "b" in pairs only, or by taking the intersection of Rule1 and Rule2, rather than their composition.

A more complex example of composition is given in figure 3.2.[7] The resulting automaton would be extremely difficult to describe without using composition. In this figure we have a grammar for translating a string of letters, digits and punctuation into a simple 3-digit code. The rule Filter translates all upper-case letters into lower-case, and all digits and punctuation into the letter "a". This rule is then composed with the rule Sound. Sound does four things. First, it translates all lower-case letters into digits (and there will only be lower-case letters as input to Sound, because it is receiving the output from Filter). This is composed with a rule which eliminates all duplicate digits (which is achieved by using the join operator (@@)—a form of composition where the "joined" tape is not thrown away—on

[7]This example is derived from a grammar written by J. H. Johnson describing a transducer which implements Soundex encoding [Joh83].

```
#
# This grammar gives a sophisticated example of the use
# of the composition operator @.
#
Alph        = {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z} ;
CopyAlph    = Alph $ (0,0) ;
Digit       = {0,1,2,3,4,5,6,7} ;
CopyDigit   = Digit $ (0,0) ;

Sound       = CopyAlph*
                  @ { ({b,f,p,v}, 1),
                      ({c,g,j,k,q,s,x,z}, 2),
                      ({d,t}, 3),
                      ({l}, 4),
                      ({m,n}, 5),
                      ({r}, 6),
                      ({a,e,h,i,o,u,w,y}, 7) }*
                  @ ( { (0+,0), (1+,1), (2+,2), (3+,3),
                      (4+,4), (5+,5), (6+,6), (7+,7) }*
                      @@ ( (Digit* {'00','11','22','33','44','55','66','77'} Digit*):acomp) )
                  @ CopyDigit* (^,'00')
                  @ (CopyDigit :3) (Digit,^)* ;

Filter      = { ({A,a},a), ({B,b},b), ({C,c},c), ({D,d},d), ({E,e},e),
                ({F,f},f), ({G,g},g), ({H,h},h), ({I,i},i), ({J,j},j),
                ({K,k},k), ({L,l},l), ({M,m},m), ({N,n},n), ({O,o},o),
                ({P,p},p), ({Q,q},q), ({R,r},r), ({S,s},s), ({T,t},t),
                ({U,u},u), ({V,v},v), ({W,w},w), ({X,x},x), ({Y,y},y),
                ({Z,z},z), ({0,1,2,3,4,5,6,7,8,9,'.','&',',',},a) }* ;

Final       = Filter @ Sound ;
```

Figure 3.2: The Composition Operator—Complex Example

two rules, one which reads in one or more copies of a digit and outputs a single copy, and the other which disallows pairs of digits from occurring, by means of the :acomp operator). All of this is then composed with a rule which adds "00" onto the end of each string to ensure that all strings are at least three digits long. Lastly, this is composed with a rule which takes the first three numbers in the string, and throws the rest away. So, the rule Final has the effect of taking a string as input, and producing a three digit representation of it as output. It is unlikely that this grammar could have been written as clearly or easily without using the complement (and join) operators.

3.2.2 *lsim*

lsim simulates the workings of a transducer represented by a two-tape automaton created by *INR*. *lsim* reads a two-tape automaton created by *INR*, along with a file of input, and creates an output file and a report file, listing run statistics and any errors which occurred. To achieve this, *lsim* interprets tape 0 as input, tape 1 as output, and walks around the "graph" represented by the automaton as directed by its input stream. It is quite common practice to represent automata as graphs, where states are nodes, transitions are arcs and tape symbols are labels on the arcs. The entire "parse" graph is not typically manifested at any given time though; rather *lsim* conceptually represents the interesting (i.e. currently used) portion of the graph as a tree, where the leaves represent possible future courses of action, which will eventually be determined by the input: multiple possible paths in the tree are caused by non-determinism in the automaton, but these multiple paths will eventually pare down to a single one, assuming that the automaton has been described unambiguously. Output occurs when the path from the root of the tree to a node, say w, has no branches, and contains at least one output transition. In such cases, the output is physically done by *lsim*, the path from the root to w is discarded and w then becomes the new root of the tree. Input occurs wherever necessary in order to build and prune the tree—*lsim* is input driven in this respect.

Since *lsim* only supports 2-tape automata, we will restrict our discussion to that case, but bear in mind that no such restriction exists in *INR*.[8] In reality, many of the grammar rules use a third tape, but this tape is only used to keep track of fonts within the Dictionary. The means by which this is achieved will be explained in Section 4.4.

A construction such as ('aa', 'bb') means: when "aa" is seen on input, produce "bb" on output. In fact, *INR* will create the automaton given in figure 3.3 to describe the rational expression ('aa', 'bb'). The parentheses surrounding the above example indicate tuple formation.

To use a more complex example from the *OED* grammars, a construction such as ('+QN ',' <quot>') means: when a '+QN ' (beginning of quotation) tag is seen on in-

[8]There is currently a 10 tape limit in *INR*, but it could easily be increased. In actual fact, we have never found a practical use for more than 3 tapes.

State	*Symbol*	*Transition*
(START)	0.a	2
2	0.a	3
3	1.b	4
4	1.b	5
5	-|	(FINAL)

Figure 3.3: An Automaton for ('aa', 'bb')

put, produce '<quot>' (the *new* begin-quotation tag) on output; whereas a construction such as ('^,' </qtxt>') or (,' </qtxt>') means: on null input produce '</qtxt>' on output.[9]

Owing to the tuple notation for I/O and the ability to omit the tape number when referring to tape 0, almost all mention of tapes may be removed from the grammars although implicitly they are fundamental—notice that they are included in the given automaton. The readability is greatly enhanced by these two devices. The above I/O operation could have been equivalently denoted by preceding the symbols with tape numbers. As stated earlier, one can explicitly refer to tapes in *INR* by preceding a symbol with a tape number followed by a period. Thus 0.a means "a on tape 0" to *INR*, and is interpreted as "read a" by *lsim*. Similarly 1.a means "write a" to *lsim*. As a rule, we avoid this notation, as it is rather cumbersome. An equivalent method of indicating tape numbers is to surround an automaton with square brackets, which means: increment the tape number(s) of the enclosed automaton. Thus, a construct such as:

(a, b, c, d)
is semantically equivalent to
0.a 1.b 2.c 3.d
or
a [b] [[c]] [[[d]]]

A few symbols are, however, preceded with explicit tape numbers in the *OED* grammars. These are all symbols which have special meanings to *lsim*. 1.ECHO and 1.NOECHO (or equivalently [ECHO] and [NOECHO]) mean "copy" or "do not copy" input to output, respectively. In the transduction of marked-up *OED* text, most of the time we want to copy input to output—we set 1.ECHO and leave it—but when we are *replacing* an input tag with one of our own, we do not want to copy the input tag to output. We then set 1.NOECHO, make the transduction and then reset 1.ECHO. These symbols are more than just a notational convenience—the use of them greatly reduces the size of the automata

[9]The ^ symbol represents the null string to *INR*.

created to transduce the *OED*, because we do not need to specify output (tape 1) transitions explicitly for the majority of the grammar. The final special symbol, 1.RESTART, aids in error recovery, as its name suggests. The restart symbol is inserted into the grammar in places which have been empirically determined to be safe (i.e., unambiguous) places from which to re-commence transduction after an error occurs.

An error, incidentally, occurs whenever the grammar has not adequately described the Dictionary, so that at a given point the transducer either has no valid transition to make on input, or is ambiguous.[10] In either case *lsim* announces which type of error it has encountered, prints the offending context and then skips over text until it reaches the next restart point.

3.3 Uses of *INR* and *lsim* for the *New OED*

3.3.1 Tagging the *OED*

INR and *lsim* are quite general tools for creating and simulating automata. The first and most pressing task for these tools in the *New OED* project was to convert and augment the manually entered ICC tags in the Dictionary text, so that the tagging reflected the underlying *structure* of the *OED*, and so that important elements of the Dictionary which were not distinguished by their ICC tagging, would now be distinguished by the output tagging. All the original ICC tags will eventually be replaced by the transduction process.

Every structural tag inserted by the transducer has a matching end-tag, and the convention is: <tag> matches </tag>. Tags may be nested, and their boundaries will not overlap. In other words, structures of the following form can be found:

<tag1> ... <tag2> ... <tag3> ... </tag3> ... </tag2> ... </tag1>

but not

<tag1> ... <tag2> ... <tag3> ... </tag2> ... </tag1> ... </tag3>

See figure 3.4 for a complete list of the output tags used in the *OED* grammars, along with their meanings. In addition, for each output tag in the table we include a status, which will indicate whether an equivalent tag existed in the original ICC input i.e., whether the transducer could merely replace the ICC tag with the appropriate output tag (and add the equivalent end-tag) or whether it had to infer the existence of this tag through context. Valid statuses will be "new" and "old", which have their obvious meanings, "mixed" which means that ICC has tagged *some*, but not all of these structures, and "augmented" which means that additional information has been added to the output tag which did not exist in the ICC tag.

[10] In our experience the most common error is invalid input, but this produces the same error as if the transducer had no valid transition to make because the definition of the transducer is too restrictive to allow for a particular situation.

Begin tag	*End Tag*	*Status*	*Meaning*
<R>	</R>	old	Roman text
<I>	</I>	old	Italic text
<B>	</B>	old	Bold text
<SC>	</SC>	old	Small capitals text
<lab>	</lab>	mixed	Label
<labs>	</labs>	old	Label List
<hom>	</hom>	old	Homonym Number
<pos>	</pos>	mixed	Part of Speech
<sn>	</sn>	new	Sense Number
<xlem>	</xlem>	new	Cross Reference Lemma
<xra>	</xra>	new	Cross Reference Address
<rxra>	</rxra>	new	Relative XRA
<pron>	</pron>	new	Pronunciation
<hwlem>	</hwlem>	mixed	Headword Lemma
<hwgp>	</hwgp>	new	Headword Group
<vf>	</vf>	new	Variant Form
<vd>	</vd>	new	Variant Date
<vfl>	</vfl>	old	Variant List
<etym>	</etym>	old	Etymology (Main)
<et>	</et>	new	Etymology (Subordinate)
<enote>	</enote>	augmented	Etymological Note
<com>	</com>	new	Command (*Supplement* only)
<lem>	</lem>	old	Lemma (Bold or Italic)
<sen>	</sen>	augmented	Sense Section
<snote>	</snote>	augmented	Sense Note
<qlab>	</qlab>	new	Quotation Label
<qdat>	</qdat>	new	Quotation Date
<auth>	</auth>	new	Quotation Author
<srce>	</srce>	new	Quotation Source
<pna>	</pna>	new	Part or Act Number
<qnot>	</qnot>	new	Quotation Note
<qtxt>	</qtxt>	old	Quotation Text
<quot>	</quot>	old	Quotation
<hwsec>	</hwsec>	new	Headword Section
<signif>	</signif>	new	Signification
<entry>	</entry>	augmented	Main Entry

Figure 3.4: Output Tags in the Transduced Text

3.3.2 Describing the *OED*

As stated in Section 1.3, there were numerous compromises which ICC decided to make when designing the original mark-up. The editors of the Dictionary and the software which was to manipulate the text need far more information about the structure of entries in order to deal with them intelligently. In addition to surrounding important elements of the text with begin and end tags, the transducer was required to distinguish between structures which were not distinguished by the ICC mark-up. This is accomplished by understanding, as far as possible, the context in which a certain structure appears and, if possible, by discriminating on the basis of keywords, textual patterns or distinguishing typographical marks.[11] The output from the transduction process—text and tagging—will henceforth be known as "transduced text".

As an example of the type of problem which the transduction process had to solve, we will look at the use of the small capitals font. There are several important structures which appear in small capitals in the *OED*. The most important of these are authors' names in quotation banks and cross-reference lemmas (i.e., lemmas which refer to another headword in the Dictionary).[12] Examples of each are given below:

> 1601 SHAKS. *Jul. C.* III. ii. 49,
> I slewe my best Louer for the good of Rome.
>
> [f. TELEOLOGY + -IC.]

In the first example above, Shakespeare, as the author of the quotation illustrating the use of the word "lover", has his (abbreviated) name given in small capitals. In the second, the etymology for the word "teleologic" shows that it is simply the root "teleology" followed by the suffix "-ic". The interested reader of the Dictionary is thus instructed to examine these two other entries to obtain further etymological information. Book, part or act numbers can also appear in small capitals: for example, the above quotation comes from act 3 of *Julius Caesar*. The act number is given in small capitals to distinguish it from page numbers (which can be given as Roman numerals, if the quotation is taken from the preface of a book) and from scene numbers—for instance, in this case the quotation comes from scene 2. Finally, a handful of special symbols can appear in the small capitals font, with no particular structural meaning attributed to being in this font, for example: "A.D.", "B.C.", the "L" of "L-amino acid" etc. These various structures are all marked-up by ICC as shown below:

[11] This technique is not without its problems, because the editors of the *OED*, particularly the early editors, would blithely break their own rules regarding the form of an *OED* entry whenever they felt that the readability of the text would thereby be increased. This was a reasonable decision on their parts, for they designed the Dictionary for humans, not computers.

[12] A lemma, in this context, is any word which is defined in the Dictionary.

+SC Shaks. +I Jul. ...
+SC Teleology +22 -ic. +R ...
+SC iii. +R ii ...
+SC a.d. +R ...
+SC L-+R amino ...

The '+SC ' is the ICC tag indicating that the font is to change to small capitals, '+I ' signifies a change to italic font and '+R ' signifies a change to roman font. The transducer must be able to recognize something inherently different in each of the above examples and surround them with distinct, meaningful and appropriate tags. This is especially important in the second example, where the transducer must recognize that the "+" between "teleology" and "-ic" (represented by the ICC tag '+22 ') is merely a delimiter and that both "teleology" and "-ic" must be surrounded by their own sets of tags, separated by the "+".

This is by no means the most difficult example of tag augmentation which the transducer has to confront. It is, in fact, rather typical. Difficulties abound throughout the highly complex Dictionary. The transducer will inevitably make some mistakes—mistakes which will need to be corrected manually, although some (such as improperly identified cross-references) may at least be *detected* automatically by the software which uses the transduced text. The first such usage would be by a database loader, which would take transduced text and create an extremely simple preliminary database from it. The database is preliminary because a significant amount of work has to be done on the text before it properly represents the *New OED*. The major activities required to achieve this goal are: cross-reference resolution (which should catch improperly identified cross-references by attempting to match every cross-reference address with its target entry); integration of the Dictionary with the *Supplement*; addition of new words (approximately 3,000 are currently planned to be added); manual editing and correction of entries; and the production of proofs for proof-reading. The last two items may, of course, be repeated any number of times.

Another way in which *INR* has been useful in describing the *OED* is as a research tool for discovering the "deep" structure of the Dictionary. By approaching the task of grammar writing as one of designing grammatical structures which represent the physical structures of the *OED* (rather than, say, looking at the exercise as one of very complex string matching) the underlying structure of the *OED* may be uncovered. Of course, this is no guarantee of success, and many incorrect approaches were tried before the present one was fashioned, but by using *INR* and *lsim*, we had a test-bed for the various theoretical approaches to describing the structure of the *OED*. This has turned out to be a very significant aid in furthering the understanding of the *OED* for computerization. It has given shape to the design of the software which is to manage the Dictionary, as well as aiding the task of database design.

3.3.3 Aiding the Editing, Integration and Proof-reading Processes

In addition to creating a grammar which describes the *OED*, *INR* and *lsim* have several other potential and realized uses in the *New OED* project. The first two of these uses are physically concurrent with the transduction of the data and augmentation of the tagging, but are logically distinct. They are: identification of commands and instructions embedded within the text of the *Supplement* which will be used to direct the integration of the Dictionary and *Supplement*, and then deleted; and the creation of mark-up tags[13] for the production of proofs and for the text editor *LEXX*.[14]

As stated in Section 1.2, the *Supplement* is meant to stand by itself and be readable as a book in its own right, but, for some of its entries, it needs to be read in conjunction with the *OED* to obtain a complete understanding of a defined word. If a *Supplement* entry has been written for a completely new word in the language (which may simply be a new homonym) then it stands on its own as does any entry in the *OED*. If, on the other hand, a *Supplement* entry modifies an existing *OED* word, then the new entry will include some instructions to the reader detailing how to integrate the two pieces of text mentally. These instructions can appear anywhere within the sense sections of an entry (i.e., the part of an entry where a word is defined). Figure 3.5 illustrates two typical types of *Supplement* entries with integration commands. In the entry for "hogo" we are instructed to "Delete † and add later example" relating to sense 1.b. The † is the *OED* symbol meaning "obsolete", so this entry is one which was thought to be obsolete when the *OED* was written, but which has since appeared in public usage, so can no longer be considered obsolete. In the entry for "landdrost", we are instructed to "Add to def.: Under British . . ." and then we are told that "Further examples" are being given. In this case, the *Supplement* editors have decided that the original definition for "landdrost", while not incorrect, needed to be augmented and clarified somewhat. In addition, they have added further examples, giving more modern usages of the word. In the integrated *OED*, all of these commands will disappear, because their effects will be manifested, rendering them redundant.

Supplement instructions are not distinguished by ICC tagging, but they must be distinguished in the transduced text, by being surrounded by special tags. When this is done, a computer program can read the transduced Dictionary and *Supplement* files and attempt to integrate them on the basis of the instructions embedded within the *Supplement* entries which have been identified by the transducer. This process will undoubtedly be imperfect, because some of the integration depends upon semantic understanding of the text and some of the commands are so unusual or complex that the effort to create software to integrate them automatically could not be justified—the automatic integrator will deal only with the most regular and frequently occurring types of integration problems, and will attempt to

[13] Currently, the text is being marked-up with GML/Script tags at OUP and with LaTeX tags at the University of Waterloo.

[14] *LEXX* is a general purpose text editor, written by Mike Cowlishaw of IBM UK Ltd. for the *New OED* project.

hogo. **1. b.** Delete † and add later example.

1922 JOYCE *Ulysses* 368 Come near. Then get a hogo you could hang your hat on.

landdrost. Add to def.: Under British administration, the office was abolished. (Further examples.)

1801, etc. [see *HEEMRAD]. **1947** L. HASTINGS *Dragons are Extra* ii. 35 Any old leader or *landrost* of the Free State or Transvaal had just Botha's sort of serenity. **1952** E. H. BURROWS *Overburg Outspan* i. 16 Outwardly the old Dutch form persisted until 1827 when the *Collegies* were abolished, and the *landdrosts* replaced by Resident Magistrates and Civil Commissioners.

Figure 3.5: Two *Supplement* Entries

fail as sensibly as it can otherwise. Each automatically integrated entry will be carefully proof-read by a lexicographer, and modified if necessary using the text editor *LEXX*.

This leads to the next use of *INR* and *lsim*—the creation of mark-up for the text editor or for a typesetter. The lexicographers, whether composing new entries, editing old ones or modifying integrated ones, must be able to view the *OED* both as raw data and as a formatted piece of Dictionary text, both on-line and on paper. To this end the entries are marked-up with tags which indicate both the *visual* presentation of the material and its underlying logical structure. The purpose of tagging the *OED* is more than just a matter of surrounding "interesting" objects with tags which identify them. It can in fact be broken up into three logically, but not necessarily physically, distinct objectives. These are: 1) tagging of structures where the tags indicate the meaning of the enclosed data, as well as its appearance on the printed page; 2) tagging of structures where the tagging has *no* effect on the printed form of the enclosed object; and 3) replacement of ICC tags indicating special characters with their equivalents in either GML/Script, LaTeX, or some other text formatter's codes.

Fulfilling the first objective was the most important task in the transduction process. Structures of the first type are, to name a few: entries, headword lemmas and labels. The <entry> (begin entry) tag not only indicates that a new entry follows (up to the </entry> tag) but that the typesetter should leave a wide vertical space and begin printing the text that follows at the left-hand margin. The <hwlem> ... </hwlem> (headword lemma) tag pair indicates that the enclosed word (or words) is the lemma being defined, and tells the typesetter to print in a large bold font.[15] The <lab> ... </lab> tag pair delimits a label

[15]The actual type size used would depend upon the purpose of the copy being typeset—for instance, one

and tells the typesetter to print it in italic font.

The second objective is really a special case of the first. In these structures, the object surrounded by tags has structural significance, but the tagging has no effect whatsoever on the *appearance* of the enclosed text, i.e. on the typesetting of this structure. These structures either inherit the type style of their environment, or themselves contain tags which dictate the appearance (and potentially the nature) of the subordinate structures. For example, the <qtxt> ... </qtxt> (quotation text) tag pair delimits the text of a quotation, but says nothing about the way that that text should be printed.

The third objective is by far the simplest of the three. It simply requires a set of context independent pattern replacements. For example, the ICC tag for the upper-case Greek letter omega (Ω) is '+999 '. This ICC tag would be replaced with the keyword ":Omega." in Script, or the keyword "\Omega" in LaTeX, so that when the data is processed with the appropriate typesetter, the character Ω will appear in the text, exactly as it does in the Dictionary.

Keeping the objectives of the tagging in mind, it can be seen that there are two different ways in which the transduced text would need to be presented, in addition to its raw form—these are as structure proofs and as content proofs. A structure proof has all its purely typographical tags translated into their pictorial representations, so a "\Omega" would appear as Ω, but leaves structural tags (of the first *and* second type) as tags, merely indenting in a rigorous way to indicate the semantic level of the enclosed structure. Also, any tags of special interest could be highlighted, either by indentation, by spacing or by printing in a different font—for instance, the *Supplement* integration instructions and the tags which surround them may be of special interest (because they will be deleted in the integrated Dictionary), and could be given special prominence for this reason. A content proof translates *all* tags, so that the result looks like a finished *OED* entry except that the right margin remains ragged. See Appendix C for examples of each type of proof in a reasonable (though not yet finalized) format. As stated earlier, all forms of the transduced text must be available in printed form or on-line. As a final note, one could easily imagine any number of different levels of proof, from one which *only* shows structural tags to one which is indistinguishable from a finished *OED* entry.

The final major use of the transduction process is as an aid to the proof-reading of the ICC data. The transducer is particularly good at noticing and reporting situations which violate the grammatical specification of the Dictionary, such as missing tags or punctuation, tags which are out of order, improper spellings of keywords, and so on. In short, the proof-reading done by the transducer is complementary to that done by humans, since it cannot notice general semantic errors which humans can easily spot, but will point out subtle tagging errors which humans typically miss. For this reason, the transduction process has become an indispensable part of proof-reading of the ICC data at the Oxford University Press.

---

might want entries being typeset at a terminal to be printed larger than those on a printed page.

3.3.4 Potential Uses

A potential use for the transducer, and one which is quite likely to be realized before the *New OED* project is complete, is as a first test of newly composed entries. Whether new entries are composed using ICC tags or some other system of mark-up (such as the format of the transduced text) a grammar could be quickly and easily written to ensure that the newly composed entries conform to *OED* standards, in form at least. The grammar would be simple to construct because the difficult part of writing a grammar for the *OED*—that of understanding and specifying the structure of the *OED* as a regular language—has already been achieved. All that would need to be done is to specify the desired input conventions (i.e., how do we specify a headword, a cross-reference, a sense section etc.) and to decide whether the grammar for new entries should be *more* restrictive than the grammars for the *OED* and *Supplement*, thereby enforcing standards more tightly than has been possible in the past. In fact, the new grammar could be used to test *any* entry which has been modified at any stage in the construction of the *New OED*, whether by human or computer. This would ensure that edited and integrated entries in the *New OED* are consistent in form with those of the old *OED* and *Supplement*, and possibly even more rigorous in their structures. The extent of this rigour could only be determined by the editorial board, because this type of decision is tantamount to determining the format of entries for future versions of the *OED*. This type of issue will likely be deferred until the second phase of the *New OED* project.

Another potential use for *INR* and *lsim*, is as a research tool for the Dictionary. Once the structure of the Dictionary is understood and the text is tagged to reflect this understanding, it is relatively simple to write grammars which read the transduced text and select "interesting" elements out of it, while ignoring all "uninteresting" elements. For instance, if a researcher were only interesting in looking at quotations, or etymologies, or variant lists, then these could be selected for output, and the rest of the Dictionary would be passed over. This selection could become quite sophisticated—for instance, one could choose only first quotations, or only etymologies which included words from Old French, and so on. However, this technique suffers from several problems: the output capabilities of *INR* are quite limited at the current time—one can only output or suppress characters as they appear; one cannot perform any sophisticated testing or manipulation of the data; there is no facility for accumulation of output into lists or sets, and then performing further analysis or manipulation on these structures.

INR will not, however, be the only generalized research tool for the *OED*. A program having some facilities complementary to those offered by *INR* has already been developed. This program, called *Goedel*[16] interprets a programming language which has several features specifically tailored to manipulate the data structures which represent *OED* entries. This language can be used interactively, and contains most of the high-level control structures

[16]Developed by G. H. Gonnet at the University of Waterloo.

that one associates with a general purpose programming language—control of execution (if tests, while loops), variables, scoping, subroutines—as well as many features which are tailored specifically for the *OED*, such as functions which understand the superficial tagging structure of the transduced text, or which can remove punctuation, accents and special characters from text, leaving only letters and numbers, or which understands the various types of quotation dates.

Since *Goedel* is a generalized research and extraction tool, it can absorb other tools; for instance, it could use the automata created by *INR* to extend its pattern matching capabilities. While *INR* is limited (by design) in its function to creating automata from regular expression input, *Goedel* has, by design, no such limitation, and so it is potentially more powerful than *INR* as a research tool. Since the *Goedel* language is high-level and procedural, and its basic control structures are similar to those found in the majority of modern programming languages, it has shown itself to be slightly easier to learn and use than *INR*, which is less familiar and less intuitive to the new user.

3.4 Results of the Transduction Process

Now let us examine, in some detail, the end results of the transduction process. The transduction consisted of three passes: one pass written as a C program followed by two passes using *lsim* (simulating the automata produced by *INR*). These three passes will be discussed at some length in Chapter 4, however they bear a few words of explanation here. The first pass mainly served to normalize the data via a number of context-insensitive pattern replacement rules. This normalization was not crucial but it helped to control the size and complexity of the grammars. The two *INR/lsim* passes can be roughly broken up as follows: the first pass identifies all low level structures and ensures that they are distinguished by being surrounded by matching pairs of tags; the second pass, taking advantage of not having to worry about low level identification of structures, looks at a larger context and makes meaningful groupings of low level structures, as well as correcting errors made in the first pass which can only be detected by examining a larger context.

To get a better understanding of the results of the transduction process, we will examine, in some detail, the metamorphosis of a single *OED* entry, from its original form in the Dictionary, to its ICC encoding and finally to its final state as transduced text. The *OED* entry, for the word "malison" as a noun (or substantive, in *OED* terms) is given in figure 3.6. Its ICC encoded form is given in figure 3.7 and its transduced form is given in figure 3.8. The entry for "malison" was chosen because it is relatively small, but it includes almost all of the main structures of a paradigmatic *OED* entry.

Malison (mæ·lis n), *sb. arch.* and *dial.* Forms: 4 malisun(e, malysun, malesun, maliscun, malescun, malicun, malicoun, 4–5 malyson(e, malisoun(e, 4–6 malysoun, 5–6 maleso(u)n(e, 6 malisone, 7 mallison, 4– malison. [a. OF. *maleison*:—L. *maledictiōn-em* MALEDICTION.]

1. A curse, malediction.

a**1300** *Cursor M.* 2051 His malison on þam he laid. c**1300** *Havelok* 426 Haue he the malisun to-day Of alle þat eure speken may! c**1320** *Sir Beues* 3696, I praie Mahoun þar fore ȝeue þe is malisoun. c**1450** *Mirour Saluacioun* 580 The malison of oure for-modere shuld torne to benedictionne. **1583** *Leg. Bp. St. Androis* 283 Scho endit, And left hir malisone, consider, To Lowrie, and the land togeather. **1586** *Durham Depos.* (Surtees) 319 He answered, God's malison light on him, for he haith beggered me. **1691** RAY *Coll. Words, Gloss. Northanhymb.* 146 *Mallison,* q.d. *Malediction,* v. *Bennison.* **1721** RAMSAY *Lucky Spence* xvi, My malison light . . On them that drink and dinna pay. **1808** SCOTT *Marmion* V. xxv, A minstrel's malison is said. **1861** GOLDW. SMITH *Irish Hist.* 43 Their malison was almost as terrible as the curse of a priest. **1865** KINGSLEY *Herew.* xiii, Farewell, and my malison abide with thee!

†2. The state or condition of being cursed. *Obs.*

c**1375** *Sc. Leg. Saints* xliii. (*Cecile*) 277 Sa man, þat ves in malysone, mycht þar chese lestand benysone.

3. *dial.* A plague, torment. Also with sb. prefixed, as *cat-malison* (see CAT *sb.* 18), *horse-malison* one who is cruel to horses. (See E.D.D.)

Figure 3.6: The *OED* entry for **malison** *sb.*

+1000 0000000000 1 Malison
+PR (m+23 +11 lis+21 n),
+PS sb.
+LA arch. +R and +I dial.
+VL Forms: 4 +B malisun(e, malysun, malesun, maliscun, malescun, malicun, malicoun, +R 4+14 5 +B malyson(e, malisoun(e, +R 4+14 6 +B malysoun, +R 5+14 6 +B maleso(u)n(e, +R 6 +B malisone, +R 7 +B mallison, +R 4+14 +0 +B malison.
+ET +OB a. OF. +I maleison+R :+13 L. +I maledictio+1 n-em
+SC Malediction. +EB

+SS 1. +RR A curse, malediction.
+QP +AI 1300 +I Cursor M. +R 2051
+QT His malison on +27 am he laid.
+QN +CI 1300 +I Havelok +R 426
+QT Haue he the malisun to-day Of alle +27 at eure speken may!
+QN +CI 1320 +I Sir Beues +R 3696,
+QT I praie Mahoun +927 ar fore +26 eue +27 e is malisoun.
+QN +CI 1450 +I Mirour Saluacioun +R 580
+QT The malison of oure for-modere shuld torne to benedictionne.
+QN 1583 +I Leg. Bp. St. Androis +R 283
+QT Scho endit, And left hir malisone, consider, To Lowrie, and the land together.
+QN 1586 +I Durham Depos. +R (Surtees) 319
+QT He answered, God's malison light on him, for he haith beggered me.
+QN 1691 +SC Ray +I Coll. Words, Gloss. Northanhymb. +R 146
+QT +I Mallison, +R q.d. +I Malediction, +R v. +I Bennison.
+QN 1721 +SC Ramsay +I Lucky Spence +R xvi,
+QT My malison light+10 On them that drink and dinna pay.
+QN 1808 +SC Scott +I Marmion +SC v. +R xxv,
+QT A minstrel's malison is said.
+QN 1861 +SC Goldw. Smith +I Irish Hist. +R 43
+QT Their malison was almost as terrible as the curse of a priest.
+QN 1865 +SC Kingsley +I Herew. +R xiii,
+QT Farewell, and my malison abide with thee!

+SS +31 2. +RR The state or condition of being cursed. +I +63 Obs.
+QP +CI 1375 +I Sc. Leg. Saints +R xliii. (+I Cecile+R) 277
+QT Sa man, +27 at ves in malysone, mycht +27 ar chese lestand benysone.

+SS 3. +IR dial. +R +63 A plague, torment. +63 Also with sb. prefixed, as +IL cat-malison +EL (see +SC Cat +I sb. +R 18), +IL horse-malison +EL one who is cruel to horses. +63 (See E.D.D.)

Figure 3.7: The ICC mark-up of **malison** *sb.*

<entry>
<hwsec>
<hwgp>
<hwlem>malison </hwlem>
<pron><R>(m+23 +11 lis+21 n), </R></pron>
<pos>sb.</pos>
</hwgp>
<labs><lab>arch. </lab><R>and </R><lab>dial. </lab></labs>
<vfl><R>Forms: <vd>4</vd></R><vf>malisun(e</vf>, <vf>malysun</vf>,
<vf>malesun</vf>, <vf>maliscun</vf>, <vf>malescun</vf>,
<vf> malicun</vf>, <vf>malicoun</vf>, <R><vd>4+14 5</vd></R>
<vf> malyson(e</vf>, <vf>malisoun(e</vf>, <R><vd>4+14 6</vd></R>
<vf> malysoun</vf>, <R><vd>5+14 6</vd></R><vf>maleso(u)n(e</vf>,
<R>6 </R><vf>malisone</vf>, <R>7 </R><vf>mallison</vf>,
<R><vd>4+14 </vd></R><vf>malison. </vf></vfl>
<etym><R>a. OF. </R><I>maleison</I><R>:+13 L. </R>
<I> maledictio+1 n-em </I><xra><xlem>malediction.</xlem></xra>
</etym>
</hwsec>

<signif>
<sen para=t lit='1.'><R>A curse, malediction. </R>
</sen>

<qbank>
<quot><qdat>+AI 1300 </qdat><srce><I>Cursor M. </I><R>2051</R></srce>
<qtxt><R>His malison on +27 am he laid. </R></qtxt></quot>

<quot><qdat>+CI 1300 </qdat><srce><I>Havelok </I><R>426 </R></srce>
<qtxt><R> Haue he the malisun to-day Of alle +27 at eure speken may! </R>
</qtxt></quot>

<quot><qdat>+CI 1320 </qdat><srce><I>Sir Beues </I><R> 3696, </R></srce>
<qtxt><R> I praie Mahoun +927 ar fore +26 eue +27 e is malisoun. </R>
</qtxt></quot>

<quot><qdat>+CI 1450 </qdat><srce><I>Mirour Saluacioun </I><R>580 </R>
</srce><qtxt><R>The malison of oure for-modere shuld torne to
benedictionne. </R></qtxt></quot>

<quot><qdat>1583 </qdat><srce><I>Leg. Bp. St. Androis </I><R>283 </R>
</srce><qtxt><R>Scho endit, And left hir malisone, consider, To
Lowrie, and the land together. </R></qtxt></quot>

<quot><qdat>1586 </qdat><srce><I>Durham Depos. </I><R>(Surtees) 319 </R>
</srce><qtxt><R>He answered, God's malison light on him, for he
haith beggered me.</R></qtxt></quot>

<quot><qdat>1691 </qdat><srce><auth>Ray </auth><I>Coll. Words, Gloss.
Northanhymb. </I><R>146 </R></srce><qtxt><I>Mallison, </I><R>q.d. </R>
<I>Maledíction, </I><R>v. </R><I>Bennison. </I></qtxt></quot>

<quot><qdat>1721 </qdat><srce><auth>Ramsay </auth><I>Lucky Spence </I>
<R>xvi, </R></srce><qtxt><R>My malison light+10 On them that
drink and dinna pay. </R></qtxt></quot>

<quot><qdat>1808 </qdat><srce><auth>Scott </auth><I>Marmion </I>
<pna>v. </pna><R>xxv, </R></srce><qtxt><R>A minstrel's malison
is said. </R></qtxt></quot>

<quot><qdat>1861 </qdat><srce><auth>Goldw. Smith </auth><I>Irish Hist. </I>
<R>43 </R></srce><qtxt><R>Their malison was almost as terrible
as the curse of a priest.</R></qtxt></quot>

<quot><qdat>1865 </qdat><srce><auth>Kingsley </auth><I>Herew. </I>
<R>xiii, </R></srce><qtxt><R>Farewell, and my malison abide
with thee! </R></qtxt></quot>

</qbank>

<sen para=t status=obs lit='2.'><R>The state or condition of being cursed. +63 </R>
<lab>Obs. </lab>
</sen>

<qbank>
<quot><qdat>+CI 1375 </qdat><srce><I>Sc. Leg. Saints </I><R>xliii. (</R>
<I>Cecile</I><R>) 277 </R></srce><qtxt><R>Sa man, +27 at ves in
malysone, mycht +27 ar chese lestand benysone. </R></qtxt></quot>

</qbank>

<sen para=t lit='3.'><lab>dial. </lab><R>+63 A plague, torment. +63 Also with sb.
prefixed, as </R>
<lem face=I>cat-malison </lem><R>(see </R>
<xra><xlem>cat </xlem><pos>sb.</pos><sn>18</sn></xra><R>), </R>
<lem face=I>horse-malison </lem>
<R>one who is cruel to horses. +63 (See E.D.D.) </R>
</sen>
</signif>
</entry>

Figure 3.8: The Transduced Text of **malison** *sb.*

We will now look in some detail at the exact nature of the results of the transduction. To begin with, the entire entry, in its transduced form, is surrounded by the <entry> ... </entry> tag pair. In the ICC tagged form, only the beginning of an entry is denoted, by the '+1000 ' tag.[17] Next, we have the <hwsec> ... </hwsec> tag pair surrounding the headword section. The first element of the headword section is the headword group. Neither of these structures are tagged by ICC. The headword group is an important structure to delineate though, because it uniquely identifies an entry. The headword group, denoted by the <hwgp> ... </hwgp> tag pair contains the headword lemma (in this case there is only one, but there may be several), followed by the pronunciation and part of speech. In this entry, there is no homonym number, but if there were, it too would be included in the headword group. The headword lemma, part of speech and homonym number uniquely identify an entry, so identification and grouping of these three elements is vital to the construction of a database. In fact, one can consider that a part of speech and homonym number exist for all *OED* entries, but if a given morpheme has only one homonym then it is omitted from the headword group and is nominally considered to be, by default, homonym 1. Similarly, if the part of speech is missing, then the entry is considered to be a noun, or, in *OED* terms, a substantive. ICC tagging includes begin tags for pronunciation and part of speech, but notice that in the transduced text, after the <pron> tag, there is a <R> tag. This indicates that roman font is in effect. The <R> ... </R> tag pair have been inserted in order to make explicit what is implicit in the ICC '+PR ' (begin pronunciation) tag.[18] This is necessary because one can also have italic font within a pronunciation. By the same reasoning, there is *no* <I> ... </I> tag pair in the following <pos> ... </pos> tag pair (part of speech), because *only* italic font is valid in this instance.

Following the headword group is the label list indicated by the <labs> ... </labs> tag pair. These labels tell the reader something about the usage of the headword lemma—if its usage is regional, or limited to a particular field of specialization, or if the word has become obsolete, and so on. In this example, "malison" has two labels explaining its usage: it is archaic and dialectal. Notice that in the ICC tagging, the second label is not indicated—it is simply a piece of italic text, following a piece of roman text. In the transduced text it is, however, completely tagged.

The next structure found in "malison" is the variant forms list, indicated by <vfl> ... </vfl>. This describes the known spelling variants of the headword, along with the centuries in which these spellings were found. Only the last digit of the century is given with "1" standing for all years up to 1100. The variant dates are either given as a single century, as a range of centuries or as a number of disjoint centuries, for example: "5"

[17] In fact, ICC does not have a 'begin entry' tag. It was inserted by the first pass in order to have a standard tagging convention throughout an entry.

[18] The tags indicating roman font have since been dropped from the output tagging. It was felt that roman was the *default* font of the Dictionary, and so only modulations from this font needed to be indicated.

indicates that the form was used in the 15th century, "5-7" indicates that the form was used from the 15th to the 17th century and "5,7" indicates that the form was used in the 15th and 17th centuries, but not the 16th. *Within* the variant forms list are: the variant forms themselves (which are only indicated as pieces of bold text in the ICC tagging), denoted by the <vf> ... </vf> tag pair, and pieces of roman text. Within the roman text, however, are found the variant dates, denoted by <vd> ... </vd>. The variant dates are not tagged at all by ICC, but can be tagged by the transducer by having it recognize their form as well as the context in which they appear.

Following the variant forms list is the etymology, which contains some roman and italic text, and a cross-reference to the *OED* entry "malediction". A complete cross reference is identified by a <xra> ... </xra> tag pair, which surrounds the entire cross-reference address consisting of: the cross-reference lemma, part of speech, homonym number and sense number. Only the cross-reference lemma (indicated by the <xlem> ... </xlem> tag pair) is mandatory within a cross-reference address, and that is all that this particular example contains, since this is sufficient to identify the target of the reference uniquely. The entire etymology has been surrounded by the <etym> ... </etym> tag pair, and the ICC tags '+OB ' and '+EB ' (signifying left and right square brackets respectively) have been deleted, since they are present in all etymologies and can thus be considered to be part of, or generated by, the surrounding tags. The etymology is the final structure within the headword section. It is an optional structure in an entry, as are the variant list, label list, pronunciation, part of speech and homonym number. After inserting the </etym> tag, the transducer has inserted a </hwsec> tag. Similarly, it inserts a <signif> tag next, to indicate the beginning of the signification. The signification is a grouping of one or more sense sections (identified by the <sen> ... </sen> tag pair) and quotation banks (identified by the <qbank> ... </qbank> tag pair).

The first sense section is a simple piece of roman text, defining the first (and hence earliest, according to *OED* convention) use of the word "malison". Notice that the sense number (1.) has become an *attribute* of the <sen> tag, as has "para=t" (an abbreviation for "paragraph=true"), which indicates that this sense begins a new paragraph.

The first, and by far the longest, quotation bank contains a number of individual quotations, identified by '+QP ' or '+QN ' in the ICC tagging, and surrounded by the <quot> ... </quot> tag pair in the transduced text. The start of the quotation text itself is identified by '+QT ' in the ICC tagging, whereas in the transduced form the text is surrounded by the <qtext> ... </qtext> tag pair. Furthermore, the transducer has identified the quotation date (<qdat> ... </qdat>) and the quotation source (<srce> ... </srce>) in each quotation. The quotation date may be a range of dates (as in variant dates, except that in quotations the full year is given, where known), possibly preceded by the ICC tags '+AI ', meaning "ante" or '+CI ', meaning "circa".

In the second sense section, the <sen> tag has three parameters, two of which are identical to the first sense, and another "status=obs", which indicates that this sense is

obsolete. This piece of information was indicated to the transducer by the ICC tag '+31 ', which is the typographical tag for a dagger (†). As mentioned earlier, the dagger is the visual clue to the reader that a sense (or an entire entry) is obsolete. Now, in the transduced text, it is understood as a piece of structural information rather than just a typographical mark—it is a status which modifies the understanding of the entire sense.

In the final sense section, the <sen> tag is followed by a label, indicating that this sense of "malison" is dialectic. There are two lemmas associated with this sense, and these are indicated by the <lem> ... </lem> tag pair. Both lemmas have the attribute "face=I" indicating that they are in italic font. Following the first lemma is a cross-reference to the entry for "cat", as a substantive. Finally, the sense, signification and entry are all closed, with their respective end-tags.

3.5 Limitations of *INR* and *lsim*

3.5.1 Textual Ambiguities in the *OED*

That most of the *OED* may be described unambiguously is a reflection of the genius of its designer and first editor, Sir James Murray, who, working under tremendous pressure as well as shortages of time, space and money, managed to achieve an extraordinary level of coherency and consistency. There are several inconsistencies in the first 100 pages of the *OED*—etymological notes at the end of entries, superfluous words inserted between headwords, etymologies not surrounded by square brackets, and so on—but after that the style seemed to become more consistent. Nevertheless, some ambiguities remain which the transduction process simply cannot resolve. Many of these ambiguities are quite simple to resolve, by referring to a semantic and/or contextual understanding of the text at hand, but even this is not always sufficient. For instance, we know that small capitals are used to represent both authors' names in quotations, as well as cross-reference lemmas. Furthermore, it is not unusual to find a cross-reference within a quotation. This occurs when the quotation has already been given illustrating a word in some other part of the *OED*. In such a case, only the quotation date would be given, followed by the cross-reference enclosed in square brackets. However, a quotation author may also be found enclosed in square brackets, when the editors of the Dictionary were not positive if this person actually authored the given quotation, or if the author's name comes as part of an editorial note (which is enclosed within square brackets as well). Thus, it is possible to imagine an ambiguity which the transducer, acting without semantic information, cannot resolve. In some cases, there is no way to distinguish between an author's name and a cross-reference by context alone.

Pronunciations occurring in variant forms lists cannot, in general, be distinguished from ordinary text in roman and italic fonts. These pronunciations are not identified by ICC

tagging, as are the pronunciations in headword groups, which begin with a '+PR ' tag. Pronunciations in variant lists are simply collections of roman and italic characters, enclosed by parentheses. It may be possible to identify *some* pronunciations, if they contain special characters or character combinations, such as a n^y (which stands for the "ny" sound in words such as "signore"), which have been determined, empirically, never to appear outside pronunciations, but this technique would not work in general, because many *OED* pronunciations consist entirely of ordinary letters. For example, the pronunciation for the word "bit" is: "(bit)".

Similarly, ambiguities exist within headword lemmas. The *OED* and *Supplement* have two types of headwords, simple and complex. A simple headword is merely a single word. A complex headword, is two or more words, separated by commas and possibly even pronunciations. If the headwords are only separated by commas, then they may be alternate spellings of the same word, such as "centre, center", or they may be a phrase, the words of which are separated by commas, such as "kinder, kirche, küche" (meaning "children, church, kitchen"). One would want to mark-up the first example as:

<hwlem>centre</hwlem>, <hwlem>center</hwlem>

whereas one would want to mark-up the second as:

<hwlem>kinder, kirche, küche</hwlem>

Another type of ambiguity found within headword lemmas is between abbreviated headwords, such as "Mr.", acronyms, such as "N.A.T.O." and ordinary headwords which end with a period (because they are not followed by homonym numbers, pronunciations or parts of speech), such as "incompetence". The transducer can either include the period as part of the headword lemma, as follows:

<hwlem>Mr.</hwlem>
<hwlem>N.A.T.O.</hwlem>
<hwlem>incompetence.</hwlem>

or exclude the period from the headword lemma:

<hwlem>Mr</hwlem>.
<hwlem>N.A.T.O</hwlem>.
<hwlem>incompetence</hwlem>.

In either case, it would incorrectly label some of the entries. In situations like this, as with all ambiguities, we can only hope to choose a tagging strategy which will minimize the number of entries that will be incorrectly tagged, and to be aware of the shortcomings.

Again, this type of ambiguity is simply beyond the capabilities of finite state transduction. The only salvation in these instances, is that they are comparatively rare. This is not surprising, of course, since if they were frequently occurring problems, the *OED* editors would more than likely have designed some sort of distinguishing representation for them.

3.5.2 Context-free Problems of the *OED*

There are several requirements of the transduced text which the *OED* grammars cannot practically fulfill. This is because the grammars are limited to the descriptive power of a regular language. The text of the *OED* would be more suitably (but not completely) modelled as context-free language, even though the *OED* is not, properly speaking, structurally context-free. For instance, one does not find infinite nesting in the Dictionary, however the nesting can be as deep as eight levels in sense sections (this will be explained in detail shortly). Similarly, the *OED* is modelled as being infinite, which it is not. Thus, even though the *OED* is not context-free, modelling it as a context-free language would be a more appropriate abstraction for the grammars. This same abstraction technique is commonly found in the design of compilers for programming languages which model their language as context-free and infinite, when in fact there are hard, real limitations imposed on the language by hardware stack sizes and memory which make the language, in reality, finite.

The *OED* has several "context-free" features, the most important of these being sense levels. The *OED* has eight potential sense levels—that is, hierarchical levels of definition, where each successively lower level is more restrictive and more precise than its parent—numbered 0 to 7. The sense levels are summarized in figure 3.9, but to give an example, we will use the *OED* entry for "hat trick". This entry has two senses, "1" and "2", each with their attendant quotation bank, however, the *Supplement* also has an entry for "hat trick" which not only adds an earlier example to sense 2, but adds a new sub-sense "b" to sense 2. If one does not understand the implicit sense structure in an entry, then this instruction does not make sense. When the integrator attempts to form these two entries into one, it will have to realize that the old sense 2 was (implicitly) sub-sense "a" of sense 2, and that both sub-senses, the new and the old, belong within the larger context of sense 2.

It is necessary to tag the sense levels, just as it is necessary to tag the rest of the *OED*, so that the tagging accurately reflects the fundamental structure of the Dictionary. In this way, one would properly group lower sense levels under their parents, so that the entire sense structure of an entry could be represented as a tree where, by following the path from the root down to any given node and reading the sense text at each level, one would get a complete, precise definition of the sense at that node. Furthermore, understanding the sense levels allows one to group quotations properly with their associated sense sections. This is important for the integration process, because when quotations are added, their locations are given in terms of the sense to which they belong. This is also important for the database design and for human understanding. For example, if one encounters several

Sense level	*Symbol Description*	*Example*
0	Unlabelled definition text	
1	Upper-case Roman letter	A.
2	Upper-case Roman numeral	I.
3	One or more asterisks	*
4	Arabic numeral	1.
5	One or more asterisks	*
6	Lower-case Roman or Greek letter	a. *or* α
7	Parenthesized lower-case Italic letter or parenthesized Roman numeral	(a) *or* (i)

Figure 3.9: Sense Levels in the *OED*

level 6 senses labelled, say, "a", "b" and "c", with no intervening quotations banks or new paragraphs, which are then followed by a quotation bank, understanding the sense levels tells one that the final quotation bank contains individual quotations exemplifying each of the three senses. If the senses were separated by new paragraphs though, then only the last sense is exemplified by the final quotation bank. If, on the other hand, the senses were at different levels, say, a level 2 sense labelled "I", a level 4 sense labelled "1" and a level 6 sense labelled "a", then the quotation bank would exemplify the level 6 sense directly, and would contribute to the overall understanding and exemplification of the higher level senses.

The current sense structure of the transduced text is flat—no information about sense levels is included at all—and some program will clearly have to insert the sense level information at a later stage. This is a comparatively simple task for a program, but it is beyond the capabilities of finite state transduction.

Since the senses are in fact limited to eight levels, one could theoretically account for all possible combinations explicitly in a regular language and tag the senses accordingly, but this "solution" is untenable. As with Sir James Murray in his creation of the *OED*, every project is eventually confronted with the constraints of time, space and money. In this case, all three of these constraints related directly to the limitations of the computer hardware and software. The grammar had to be made to work within these limitations. The full grammar, with all sense levels made explicit, would have been far too large to have been processed by *INR*, *lsim* and either computer on which the transduction took place.

3.5.3 Rearranging Text

Another limitation of *INR* and *lsim* in the *New OED* project, which stems from the same cause as the above problem—the lack of a stacking mechanism—is the ability to rearrange arbitrary pieces of text. The needs of the transduction frequently dictate that pieces of text be rearranged. This can be accomplished to a limited extent in the grammars, but arbitrary text can only be moved across a constant literal (such as a tag) boundary. For instance, cross-reference lemmas are often preceded and followed by punctuation or other typographical characters which do not properly belong in the lemma, but are not distinguished from the lemma itself by the ICC tagging. These characters must be excluded from the <xlem> ...</xlem> tag pair. The way to do this without a stacking mechanism is to write a grammar which first recognizes the context that indicates the start of a cross-reference lemma, and which next recognizes the text to be moved. Only *after* this text has been transduced does the transducer insert the <xlem> tag. In effect, the text is not rearranged but the tagging is. Any more complex rearrangement of the text, such as moving some text from one structure to another, is beyond the general capabilities of the transducer (as was the case with context-free problems, any *specific* problem can be dealt with in isolation, but the general problem remains unsolved). There is some consolation, though, in knowing that there are few situations where textual rearrangement of this sort is a pressing need.

Chapter 4

Writing an OED Grammar

Writing an *OED* grammar was an iterative process. In each iteration we would postulate a grammar to describe the *OED*, run the grammar against ICC data, discover a problem with the transduced text, attempt to understand the true nature and cause of the problem and finally devise a method of dealing with the problem. This solution was then implemented and the process was repeated. This chapter will discuss the various sorts of problems which we came upon in writing an *OED* grammar, the ways of categorizing these problems, and the solutions which we adopted to deal with them.

The problems that we encountered fell into four main categories. These were (in order of increasing difficulty): problems of physical limitations, problems of data ordering, problems of description and problems of conception.

Problems of physical limitations and problems of data ordering (discussed in Sections 4.1 and 4.2 respectively) are closely related. Creating extremely large grammars with a finite amount of available memory on our host computers necessitated the adoption of a number of strategies aimed at reducing the size of the automata produced by the grammars. Several strategies are discussed and one of these, the process of data normalization, is described in detail.

Problems of description, described in Sections 4.3 and 4.4 are those concerned with finding adequate ways of describing the *OED* as a regular language. For the most part, the process is fairly straight-forward, but we have given special attention to two problem areas: dealing with textual ambiguities and dealing with font information.

Problems of conception are difficult to define and categorize. These are the psychological traps into which one may fall when attempting to find an adequate structure to describe

a textual pattern. Problems of this sort were unavoidable, since we had no grammatical models or paradigms of *OED* structure to follow—we only had the evidence of example pieces of text from the *OED*, and the advice of OUP's resident lexicographers on which to increase our understanding and base our decisions. Section 4.5 describes the process of designing a grammar to represent a structure within the *OED*, and gives some examples of the sorts of pitfalls which can plague such a design.

Finally, Section 4.6 describes the transduction of cross-references: pointers found within *OED* entries to other parts of the Dictionary. Cross-references are interesting because they represent a microcosm of the complete *OED* grammars. Every technique used in the complete grammars is also used in order to tag and disambiguate the various components of, and different types of, cross-references properly.

4.1 Dealing with Space Limitations

In developing the grammars for the *OED*, the size of the automata produced by *INR* was a constant concern and, occasionally, a severe limitation.[1] The reasons for the creation of large automata were manifold. First of all, since we did not understand, in the early stages of grammar development, how to represent the *OED* succinctly as a regular language, we often conjectured a very large, verbose grammar to represent a given structure in the *OED* when a much more concise one would do.

It was often unclear, though, how to go about changing the grammar so that it would better represent the *OED* and create a smaller automaton (in fact, at times, any *one* of those two objectives was difficult to meet). As the *OED* was being modelled, there came a time when the working grammar grew so large that we would exhaust the physical resources of the host computer before *INR* could create the final automaton. At this point, the process of experimentation was greatly retarded, because, if *INR* could not create the final automaton, then we could not test the grammar through simulation. This meant that we could not use *INR* as a research tool to test hypotheses regarding the structure of the Dictionary, much less for its original purpose—to transform the data to enhance the tagging. Similarly, we had little idea of how to go about reducing the size of the grammar while retaining

[1] There are three ways in which the size of an automaton was reported by *INR*: number of states, number of transitions and storage used. The number of states was limited to 65,536 and the storage was limited to the memory capacity of the host machine, which was typically around 8 megabytes. The number of transitions was not, by itself, critical, except insofar as large numbers of transitions created automata which required a large amount of memory.

```
Text          =   (Alph* {'+1000 ', '+1006 ', '+ET '} Alph*): acomp ;
Etymology     =   '+ET +OB ' 2.Wroman TextChars*
                  (   XRLemma
                  |   PartsOfSpeech
                  |   HomonymNumber
                  |   SenseNumber
                  |   ARoman TextChars+
                  |   AItalic TextChars+
                  |   '+EN ' 2.Wroman TextChars+
                  )*  '+EB ' ;
EnforcedEtym  =   Etymology @ FontGram ;
MainEntry     =   {'+1000 ', '+1006 '} Text (EnforcedEtym Text)? ;
Dictionary    =   MainEntry+ ;
```

Figure 4.1: A Specialized Grammar for Etymologies

its descriptive power. Thus, the size of the automaton prevented us from being able to construct experiments aimed at reducing its size.

The solution that we eventually adopted was to split the grammar into small, specialized grammars. Each of these grammars was limited to describing one particular major structure within the *OED*, while disregarding all other parts of the Dictionary as unimportant. In effect, this technique is analogous to "modularization" in the writing of software . Specialized grammars were written for etymologies, quotation banks, sense sections, headword groups and variant lists. A sample grammar used to recognize etymologies is given in figure 4.1. In this grammar, the "Text" rule allows the transducer to recognize all text up to a '+1000 ', '+1006 ' or '+ET '. This is done through the use of the ":acomp" operator, described in Section 3.2.1. In this way, we can skip over all "uninteresting" (from the point of view of a specialized grammar) text without having to specify a complex, and space-consuming, set of rules for it. Thus, we are free to concentrate on one specific aspect of the *OED* at a time—in this case entries and their respective etymologies. This technique has quite general applicability. It has been been used extensively in designing specialized grammars which extract information from the *OED* based upon some search criteria. Essentially, we wrote a grammar of some "interesting" piece of text, and would skip over all other text.

As our expertise with the grammars increased, so did our expectations of the transducers. Once we overcame the problems of describing main structures of the *OED*, we attempted to deal with progressively more difficult problems such as tagging integration commands in the ***Supplement*** and relative cross-references in the Dictionary and ***Supplement*** (the techniques used to identify and tag relative cross-references will be discussed in Section 4.6.2). These difficult tagging problems were invariably awkward to transduce cor-

rectly, since they involved disambiguation based upon keywords, rather than on tags. This sort of disambiguation was very expensive, in terms of the size of the automaton created to express it. In most of the structures of the *OED* we can use a restricted character set to identify the structure's boundaries, i.e., we don't accept a quotation bank as part of a sense section because '+QP ', the tag which begins a quotation bank, is not included in any character set in a sense section. Unfortunately, this simple, inexpensive technique is of no use in tagging *Supplement* commands.

Supplement commands can only be identified as structures in roman font which begin with one of the following words: "Add", "Additional", "Delete", "Earlier", "Example", "For", "Further", "In etym", "Later", "Restrict" or "Substitute". It is obvious that no restricted character set will filter out these words while accepting ordinary roman text. Thus we must use more complex methods of discriminating between ordinary text and words which have a special significance to the Dictionary. One way of accepting roman text while not accepting any of the above words is to use the set difference operator—we subtract the above keywords from the set of all possible words in roman font. This might be written as, for example:

Roman TextChars* – ComWord

where "TextChars" is an alphabet of valid text characters and ComWord contains the set of words which introduce *Supplement* commands. This method is much more expensive, in terms of the size of automaton produced, than merely restricting a character set. Consequently, recognizing structures such as *Supplement* commands and relative cross-references caused large increases in the size of the automata.

Another technique which was extremely effective in reducing the size of the automata created by *INR* was the use of the 1.ECHO and 1.NOECHO symbols. Had we not adopted this convention (discussed in Section 3.2.2) then we could not have transduced the *OED*—the automata needed to specify the grammars would have been too large for the host computer systems, as well as *INR* and *lsim*. Each automaton would be approximately double its current size, because, for every tape 0 symbol, there would have to be a related tape 1 symbol in order to copy input to output.

Another cause of increases in the size of the grammars was unavoidable: as we grew to learn more about the complexity and diversity of the Dictionary, we needed to create progressively larger character sets, sets of keywords and explicit combinations of structures to model it accurately. None of these increases were significant on their own, but their cumulative effect was profound. A significant portion of this problem was, however, alleviated by the normalization procedure which will be described below. Next to the adoption of the 1.ECHO and 1.NOECHO symbols, the largest decrease in the size of the automata produced from the *OED* grammars was due to the effects of the data normalization program.

4.2 Normalization of ICC Data

When investigating ways to reduce the ever-increasing size of the automata produced by the *OED* grammars, we realized that there was a great deal of needless complexity in the grammar which served no other purpose but to deal with special ICC codes, and combinations of ICC codes which had no effect on either the typography or structure of the *OED*. For instance, there were '+LC,ppp,c ' codes which were inserted into the text of the *OED* for the keyboarders and proof-readers. These codes indicated the current page (represented by 'ppp') and column (represented by 'c') of the Dictionary which was being input. Similarly, an ICC code of '+0 ' indicated a special sort of space in the input text. This code was felt to be needed because many characters of the *OED* text were encoded as tags, and the tags themselves could end with a space, so, in order to indicate that there was a *true* space after one of these tags, a '+0 ' was added.

Since the *New OED* is going to be completely re-typeset, both of these codes are meaningless in the transduced output. Furthermore, since these codes have no structural meaning in the *OED*, they can show up *anywhere* in the input text. We felt that this placed a needless burden on all parts of the grammars, and so we decided to drop them from the input to the transducers.

Another needless source of complexity in the grammars stemmed from a certain amount of arbitrariness in the ordering and even the inclusion of many tags. For instance, the '+R ' tag, indicating roman font, appears very often in the ICC text in places where it is not needed. It seems that the ICC keyboarders considered '+R ' to be "safe ground"—a tag which, since it represented the basic font of the entire Dictionary, could never cause problems by its (possibly redundant) inclusion. It could certainly not hurt the *typography* of the Dictionary, which was ICC's main concern and main test of correctness in their tagging. So, one finds many redundant combinations of tags such as '+R +R ' (two roman font tags with no intervening text), '+R +SS ' (a roman tag preceding a begin sense section tag, which itself sets bold font) and so on. Other tags are used redundantly, but the roman font tag is by far the most common. Having to allow for meaningless combinations of tags causes needless complexity in the grammars. It also violates one of the guiding tenets of the grammar: that no structure may be empty. By allowing a roman tag to precede another structural tag directly, we are allowing a roman font structure to exist with absolutely no textual content. This is not disastrous, but it needlessly increases the size of the automaton and the text.

Many tags appear in an arbitrary variety of combinations in the ICC text. This always occurs in situations where the tag order does not affect the typography, but it does affect the semantic context of the structures involved. Finally, some tags are consistently forgotten by the ICC keyboarders—tags which do not affect the typography, or only affect it subtly, and tags which only affect the structure of the text in ways which are important to the transduced text. Several examples of the sort of problem found in the ordering and consistency of ICC tags are given in figure 4.2. Take, for instance, the last example in the figure. ICC

Incorrect Tags	*Corrected Tags*	*Problem*
+SC +22	+22 +SC	Tag order
+63 +EL	+EL +63	Tag order
+R +IA	+IA +R	Tag order
+R +B	+B	Redundant Tag
+R +QN	+QN	Redundant Tag
+R +RR	+RR	Redundant Tag
+SN (	+SN +R (	Missing tag
+EB +NT	+EB +SR +NT	Missing tag

Figure 4.2: More Problems With ICC Tagging

often omitted the '+SR ' tag between an etymology and a note. The resulting text will be typeset properly, because a note (the '+NT ' tag) indicates that a new paragraph starts in small roman font. However, the structure of the text, as indicated by the ICC tagging is incorrect, because the note does not occur within any recognizable structure in the *OED*. Consequently, the data normalization process must insert the '+SR ' tag, indicating that the note is within a sense section, in order to conform these notes with the overall structure of an *OED* entry.

One final problem with the ICC tagging was that the tags did not consistently end with a space. This, as with all of the above problems was not insurmountable. The grammars could have been written to allow for any of the given combinations of tag patterns—to allow for optional spaces after tags, optional '+0 ' and '+LC,ppp,c' tags almost anywhere, and so on—but since these variations did not contribute anything to our understanding or representation of the Dictionary, we felt that they should not burden the grammars. The solution to this problem was to create a program (written in C) to normalize the data. This program ensured that every tag ended with a space and performed pattern matching and replacement. It was also used to create a index file from the deleted '+LC,ppp,c' tags. This file was then used to help the OUP proof-readers locate errors in the ICC tagged text which the transducer reported. The normalization program was run as a first pass against the raw ICC data, and its output became the input to the transducers.

The data normalization program performed one other valuable service: it tentatively identified parts of speech and labels in the ICC data. We blindly identify parts of speech and labels in the data normalization pass and correct our errors, based on a knowledge of context, in the first transduction pass. This same technique of tentatively identifying a structure in one pass and then correcting errors in the tagging, based upon a knowledge of context, in the next pass, is also used in identifying sense numbers in the *OED*, as discussed in the next section. Labels and parts of speech are not consistently distinguished by ICC tagging. They are often tagged as nothing more than pieces of italic text, and so are difficult

(i.e., expensive) for the transducer to identify. Unlike sense numbers, we can distinguish them from ordinary text in italics, but only if we have an exhaustive list of them. This is because they do not share a particular structure, as do sense numbers, but are merely specific keywords in italic font. Creating an exhaustive list of parts of speech was not too difficult—they number less than 100 (these are not always truly distinct parts of speech, but are sometimes different versions of the same word, for example, "pron." and "pronoun")—but creating a list of labels was a task of a different order of magnitude. Currently, the list of labels is about 1,000 entries long. The process of initially identifying parts of speech and labels can be done independently of context, and so was a relatively simple task for the data normalization program. The program simply searches for a '+I ' tag followed by a part of speech, in which case the '+I ' tag is converted into a '+PP ', or a '+I ' followed by a label, in which case the '+I ' is converted into a '+LL '. Again, this task could have been done in the grammar, using either the set difference or the :acomp operator, but the resulting automaton would have been enormous.

Essentially, the use of the data normalization program was an example of a divide-and-conquer approach to the problem of transducing the data. By applying it to selective context-independent tasks only, and leaving the context-dependent tasks to the transducer, we were able to achieve the desired level of tagging while controlling the size of the automata.

4.3 Dealing with Textual Ambiguities

Dealing with textual ambiguities is the most difficult and most interesting part of creating a grammar to describe the *OED*. The techniques needed to disambiguate many of the structures of the Dictionary adequately include: using two transduction passes on the data; making use of font information; and using some of the advanced features of *INR*, such as the ElseOR and :acomp operators. The :acomp operator, however, will not be discussed in this section, because its use is discussed in Sections 4.1 and 4.6.2.

As stated in Section 3.5.1, not all of the textual ambiguities in the *OED* can be disambiguated by a finite state transducer. In many cases, the approach that we have chosen to take is to make an attempt to distinguish between syntactically ambiguous objects, and to be aware of the ways in which this disambiguation will occasionally fail so that appropriate action can be taken at a later stage of the project—either by some other software or by humans. The disambiguation of superficially similar structures is facilitated by having two separate transduction passes against the data. In the first pass, a given structure will be tagged based upon its form, and in the second pass, if the structure is one which is potentially ambiguous, the location of that structure is then used to determine its true nature.

This technique is used most frequently and most successfully with sense numbers in cross-reference addresses. Sense numbers appear in roman font, and consist of numbers,

```
EndSense   =   {'(', ')', ',', ':', ';', '=', '+15 ', '+16 ', '+17 ', '+22 ', '+63 ', '+P '} ;
SenseTail  =   TextChars - EndSense ;
SenseGram  =   ({SNUC, SNromanUC, Number} {' ', '.', '+64 '}
               | SNLC '.' ) SenseTail* ;
SenseText  =   Roman
               ( (' <sn>') SenseGram (' </sn>')
                    ((' <R>') EndSense TextChars* (' </R>'))?
               || (' <R>') TextChars+ (' </R>') ) ;
```

Figure 4.3: A Grammar to Tag Sense Numbers

letters and punctuation. For example, a cross-reference to a particular sense of the verb "set" could conceivably appear as: SET *v.* III 19 d. Unfortunately, the bulk of the text in the *OED* also consists of numbers, letters and punctuation in roman font. In order to disambiguate sense numbers from ordinary roman text, the following technique was devised: in the first pass, any piece of roman text which has the *form* of a sense number is tagged accordingly (with the <sn> ... </sn> tag pair); in the second pass, improperly tagged sense numbers are re-tagged as roman text. For example, in the following text, a variant form followed by a variant date, and a piece of italic text in an etymology, followed by "f.", meaning "formed on", the roman text has the superficial form of a sense number. Consequently, the ICC text:

+B Mharatta +R 9 +B ...
+I madhuka, +R f. +I ...

is tagged by the first pass as follows:

<vf>Mharatta </vf><sn>9 </sn> ...
<I>madhuka, </I><sn>f. </sn> ...

and is then corrected by the second pass as follows:

<vf>Mharatta </vf><vd>9 </vd> ...
<I>madhuka, </I><R>f. </R> ...

A sample grammar to achieve the first stage of this tagging process is given in figure 4.3. The SenseText rule indicates that sense numbers occur in roman font and are described by the SenseGram rule. SenseGram asserts that sense numbers begin with an upper-case letter,

an upper-case Roman numeral or an Arabic number[2] followed by a blank, a period or a '+64 ' (the ICC code for a thin space) *or* it begins with a lower-case letter followed by a period. In either-case, the sense numbers are followed by zero or more occurrences of SenseTail, which is any valid text character, except for those included in the EndSense rule. In effect, what we have done is, rather than define a complete grammar of sense numbers, define a grammar which merely delimits sense numbers. SenseGram defines the characters which begin a sense number and EndSense defines those characters that end it—i.e., if we accept one of the EndSense characters, then we know that we can no longer be accepting a sense number. The set of characters to include in EndSense was empirically determined. Since a sense number may be followed by ordinary roman text, after accepting a SenseGram, we must optionally accept arbitrary roman text, as long as it *begins* with a character from EndSense (this restriction is necessary to avoid ambiguity between SenseGram and TextChars+).

The final portion of the SenseText rule is separated from the initial part by a || symbol—the ElseOR operator (recall the discussion of ElseOR from Section 3.2.1). The TextChars+ construction to the right of the ElseOR serves as a sort of safety valve—if the input cannot be accepted by the left-hand operand (the SenseGram) then it is accepted by the right-hand operand as arbitrary roman text and surrounded by the <R> ... </R> tag pair. Notice that if the alternation operator (|) were used here in place of the ElseOR operator, then an ambiguity would occur whenever an input pattern was accepted by SenseGram, because this could also be accepted by TextChars+. In this way, all text in roman font is certain to be accepted correctly and unambiguously, and that which bears superficial resemblance to a sense number is tagged as such.

In the second transduction pass, the mistakes of the first pass are corrected. The second pass grammar narrows and refines the definition of sense numbers so that improperly labelled sense numbers are discovered. This is accomplished by simply asserting in the grammar that a sense number must be associated with a cross-reference. If it is not, then the <sn> ... </sn> tag pair is replaced with <R> ... </R>. In this way, the ambiguity between roman text and sense numbers is satisfactorily resolved.

Another ambiguity in the *OED* is found in what we have called "quotation notes". This ambiguity was discussed briefly in Section 3.5.1. Quotation notes are editorial notes, enclosed within square brackets, in quotations. In fact, a quotation may be nothing more than a date, followed by one of these notes. Some typical examples of quotation notes are given in figure 4.4.

Within the quotation note, one most often finds cross-references, informing the reader that the referenced entry and sense contain a quotation having the same date, illustrating

[2]The shorthand notation for the various sense number alphabets is as follows: "SN" means sense number, "UC" means upper-case, "LC" means lower-case and "roman" means a Roman numeral; thus SNromanUC is the alphabet of upper-case Roman sense numbers.

1398 [see BUZZING *vbl. sb.*[1]].
1857 *Mary-ale [see ALE A. 3].
1822 SARA COLERIDGE tr. *Dobrizhoffer's Hist. Abipones* [Paraguay] ...
1387 TREVISA *Higden* (Rolls) II. 67 Elidurus . . fonde his brother Archgalon maskynge [L. *aberrantem*] in a wode [FABYAN *Chron.* II. xl. (1811) 28 *has* maskelyng or wandryng in the thykest of y^e^ wood].

Figure 4.4: Examples of Quotation Notes

the current headword. Occasionally, the quotation note contains pure editorial text, with no cross-reference (as in the third quotation in figure 4.4), explaining some point of interest to the reader. Ordinarily, this does not pose a problem for the transduction, however, if the editorial text happens to contain an author's name (which, you will recall, is also rendered in the small capitals font) then it will be labelled (incorrectly) as a cross-reference lemma. This is a situation which must be manually corrected by the editorial staff of the Oxford University Press. An example of this problem is found in the fourth quotation of figure 4.4. "Fabyan" is an author's name, but will be tagged as a cross-reference lemma. Fortunately, authors' names within quotation notes are quite rare, and so pose little practical hardship.

Another ambiguity in quotation notes arises from the *OED* convention of enclosing part of an author's name within square brackets. Examples of this practice are given in figure 4.5. This occurs when the author's name did not appear in full in the cited text. The *OED* editors have supplied the additional information for the reader and enclosed it within square brackets. However, when the transducer encounters a '+15 ', the ICC tag for a left square bracket, it is ambiguous whether this indicates the end of the author's name and the start of a quotation note, or just the start of a piece of editorial text elucidating an author's full name. The resolution of this ambiguity is achieved by checking the current font (exactly how this is done will be explained in Section 4.4). Quite simply, if the text is currently in the small capitals font, then the transducer interprets the '+15 ' as part of the author's name, whereas if the text is in italic, roman or bold font, the '+15 ' is interpreted as the start of a quotation note.

Another source of ambiguities resulted from the transition, in the development of the transducer, from merely scanning the text of the *OED* to producing output. In the early stages of the development of the grammars, *lsim* did not exist, so the only way to test the utility of grammars was by using a program, *scan1*[3] which simulated a *one*-tape automaton, thus the automaton was an acceptor, or recognizer, only. It could either succeed or fail to accept its input using the given automaton, and report on its progress. Since *scan1* was incapable of producing output, its measure of success was if it could find a path from a

[3]Developed by J. H. Johnson at the University of Waterloo.

1606 G. W[OODCOCKE] *Hist. Ivstine ...*
1607 R. C[AREW] tr. *Estienne's World of Wonders ...*
1642 J. M[ARSH] *Argt. conc. Militia ...*

Figure 4.5: Examples of Editorial Notes in Authors' Names

start state to a final state, through which the input could be accepted. Unfortunately, *scan1* didn't report the existence of many such paths (which would cause ambiguities in *lsim* if the different paths produced different outputs). So, when *lsim* was developed, the automata which had very successfully *recognized* large pieces of *OED* text failed miserably as transducers, producing countless ambiguities. These ambiguities did not represent any true problems with the source text such as we have detailed here and in Section 3.5.1, but rather pointed out areas where a new conceptual approach needed to be taken. This created a minor but inevitable setback in the development of the grammars, and necessitated some fruitful re-evaluation of the design of the grammars used to describe the *OED*.

4.4 Fonts in the *OED*

4.4.1 Recognizing Fonts

There are two motivations for keeping track of, and understanding, the fonts of the Dictionary when transducing its text. The first reason is the obvious one: to be able to instruct the typesetter to reproduce the typography of the *OED* faithfully. The second reason is that when the transducer is tagging ICC text, it can properly tag many pieces of text which would otherwise be ambiguous, by referring to a knowledge of its current font. The technique for achieving this is illustrated in the grammar section given in figure 4.6.

The technique for enforcing fonts in the grammar may be clarified somewhat by an analogy. If we were writing an ordinary program for transducing the text, we would store the current font in a global variable. When we accepted a tag from input which set the current font, we would update this variable with the corresponding information, and when we needed to know what font we were in, we would simply check the contents of the global variable. Since there is no notion of a variable in a regular language, we must adopt some other technique for storing and checking font information. This is now detailed.[4]

[4]This technique was first suggested by J. H. Johnson.

```
ABold        =   '+B ' 2.Wbold ;
AItalic      =   '+I ' 2.Witalic ;
ARoman       =   '+R ' 2.Wroman ;
ASmcaps      =   '+SC ' 2.Wsmcaps ;
Bold         =   ABold | 2.Rbold ;
Italic       =   AItalic | 2.Ritalic ;
Roman        =   ARoman | 2.Rroman ;
Smcaps       =   ASmcaps | 2.Rsmcaps ;
SenseSection =   '+SS ' 2.Wbold StatusMark?
                 (Bold SenseNumber '.' | Bold Index | (ARoman | Bold) ('*')+
                 | ARoman '(' (AItalic LCase | {Digit,i,v,x}+ | LCase) Roman ')' )
                 SenseBody ;
FontGram     =   ( (Wsmcaps Rsmcaps*) | (Witalic Ritalic*)
                 | (Wroman Rroman*) | (Wbold Rbold*) )* ;
EnforcedFont =   SenseSection @ FontGram ;
```

Figure 4.6: A Grammar to Enforce Fonts

The grammar in figure 4.6 uses tape 2 (the automaton's *third* tape, since tapes number from 0) as a "slate" for keeping track of its current font. In effect, whenever one of the font tags ('+R ', '+B ', '+SC ' or '+I ') is accepted from input, a symbol is "written" on tape 2. This function is achieved by the appropriate font rule: ARoman, ABold, ASmcaps and AItalic. The "A" in front of each of these rule names stands for "Assert", meaning that the transducer can definitely *assert* that it is in a particular font if it has just accepted the relevant ICC font tag. These rules write one of the following symbols on tape 2: Wroman, Wbold, Wsmcaps or Witalic, respectively. Whenever the grammar needs to *check* that it is in a given state, it writes one of the following symbols on tape 2: Rroman, Rbold, Rsmcaps or Ritalic. The rule EnforcedFont composes the SenseSection grammar with a special grammar, FontGram, which enforces constraints on the fonts (recall the discussion of the composition operator (@) from Section 3.2.1) by constraining the order of symbols of tape 2 as follows: it requires that one of the "W" symbols precedes zero or more of the corresponding "R" symbols. The "R" and "W" of these symbols stand for "Read" and "Write", respectively. This notation uses tape 2 as though it were the global variable described above.[5]

[5]Notice that the FontGram rule does not precede its symbols with a tape number, because they all refer to tape 0 in that automaton. This is necessary because the composition operator in EnforcedFont composes the last tape (tape 2) of SenseSection, the left-hand operand, with the first tape (tape 0) of FontGram,

One can conceptualized the technique as follows: when the transducer sees a font tag in its input stream, it writes the appropriate symbol on tape 2, and when it (later) needs to check if it is in a particular font, it "reads" from the tape to see if that symbol was the one most recently written. Of course, tape 2 is not a global variable, however, we can restrict the automaton through the use of FontGram, to only those states and transitions where, for example, Roman is "written" on the tape before it can be "read". The effects of this composition are not that any physical writing is done in a special area of storage, but rather that valid paths through the EnforcedFont automaton are constrained to those which not only satisfy the description of SenseSection, but which satisfy FontGram as well.

Given that fonts are enforced by FontGram, the other rules in the grammar may use the rules ARoman, ABold, ASmcaps and AItalic whenever an explicit font tag is required on input, and to use the rules Roman, Bold, Smcaps and Italic whenever they merely need to check the current font. Checking the current font involves either accepting an ICC font tag and writing the appropriate symbol on tape 2, or recording that a read has been "requested", confident in the knowledge that, because of the constraints on the automaton imposed by FontGram, this will only occur when the desired font is, in fact, the one most recently asserted.

The SenseSection rule in figure 4.6, for instance, first accepts a '+SS ', indicating the start of a numbered sense section which, incidentally, begins a new paragraph. The '+SS ' tag has the side-effect of setting bold font, so, to reflect this, the Wbold symbol is immediately written on tape 2. Next, a StatusMark, which indicates any special status of the sense (such as whether the sense is obsolete), is optionally accepted. Following this, the sense number is accepted, in one of six different forms. A simple SenseNumber or an Index (a Greek letter) are accepted in bold font. The Bold rule is used here because, even though the current font is bold, ICC will occasionally insert a redundant '+B ' tag in this position. Alternatively, one or more asterisks may be accepted in either bold or roman font. Roman font here must be explicitly asserted through accepting a '+R ' tag but bold font only needs to be checked. Similarly, a roman left bracket may be accepted, followed by either a lower-case letter in italic font (which is, again, explicitly tested for by the AItalic rule) or by a Roman or Arabic numeral or a lower-case letter in roman font. Finally, a roman right bracket is accepted. In this case, the Roman rule is used, because we need to check that we are still in roman font (if we accepted one of the roman font sense numbers), or to accept a '+R ' tag if we were previously in italic font. In each case, only valid combinations of fonts, as determined by FontGram, are allowed.

The use of fonts in figure 4.6 was rather naive. It was only given as a rather simple example, in order to illustrate the mechanism of font enforcement. One could easily imagine other ways of writing the same grammar, though, which would not involve the writing of symbols on tape 2 and composing the grammar with FontGram to enforce an ordering on these tape 2 symbols. A more critical use of the special font grammar is given in

the right-hand operand.

```
QuotationSource =
                (    QuotationAuthor
                |    AItalic TextChars+
                |    ABold Date
                |    ARoman TextChars+
                |    ASmcaps PartOrAct
                |    (2.Rroman | 2.Rbold | 2.Ritalic) '+15 ' QuotationNote '+16 '
                )+ ;
```

Figure 4.7: A Complex Use of Font Enforcement

figure 4.7. This is the grammar rule which resolves the ambiguity between square brackets surrounding parts of an author's name, and square brackets surrounding a quotation note, as was promised in Section 4.3.

The QuotationSource rule in figure 4.7 resolves the ambiguity between the two uses of square brackets (which are represented by ICC tags '+15 ' and '+16 ' for left and right brackets respectively) as follows: the only valid places that a '+15 ' can appear in a QuotationSource are in a QuotationAuthor, or just preceding a QuotationNote. However, before the '+15 ' can be accepted, a 2.Rroman, 2.Rbold or 2.Ritalic symbol must be written. FontGram ensures that only the appropriate symbol will be written, and only if one of Wroman, Wbold or Witalic was the most recently written "W" symbol on tape 2. If the most recently written symbol was Wsmcaps, then the restrictions that FontGram imposes on the automaton will not allow any of those three "R" symbols to be written, hence the '+15 ' can *only* be accepted by the QuotationAuthor rule. In this way, an ambiguity is successfully avoided.

There are other ways of resolving ambiguities than explicitly encoding and then examining font information, but they are no more powerful than this technique. For instance, whenever the font was potentially ambiguous, we could have simply introduced some redundancy into the grammar to deal with it, i.e., to create separate streams in the grammar for each font possibility. We have chosen the current technique because it removes most font considerations from the main grammar, and enforces them quietly and elegantly in a distinct grammar.

4.4.2 Problems with the Small Capitals Font

The small capitals font presented several problems for the transduction process not found within other fonts. One reason for this is because small capitals is not truly a font at all. The

+SC Goode, +R etc. ...
+SC Sir.) +NT ...
+SC Manatee: +R see ...

Figure 4.8: Examples of Problematic Text in Small Capitals

characters in this font do not have a distinct type style, but are, rather, a special variant of roman font where capital letters in the small capitals font are identical to capital letters in roman font, and lower-case letters are merely sized down upper-case romans. Furthermore, the small capitals font only has relevance to alphabetic characters. Since there is no concept of small capitals period, or semi-colon or apostrophe, digit, and so on, whenever one of these non-alphabetic characters followed a word which was in small capitals, the ICC employees who were inputting the text did not feel obligated to insert a '+R ' tag between the word and one of these subsequent characters, whereas they *did* insert the '+R ' tag if the small-capitals text was followed by ordinary alphabetic roman text. This is because ICC considered small capitals to be a special case of roman font, rather than a distinct character set, and so felt that there was no need to distinguish punctuation characters between the two. This does not have any adverse consequences on the typography of the Dictionary, but it has serious consequences on the structural tagging. In each of the examples given in figure 4.8, the ICC tagging, if followed blindly, would lead to incorrect transduction output tagging of the structure given in small capitals: in the first example, the author's name "Goode" would include a comma; in the second, the cross-reference would be to "Sir.)" and not simply to the word "Sir" itself; and in the third, the cross-reference would include the colon at the end of "Manatee". In each of these cases, the transducer must be able to tell, from context, where the actual structure ends and where the following punctuation begins, and then tag it appropriately.

The other important problem with the small capitals font, also stems from its close connection with ordinary roman text. Since an upper-case letter in roman or small capitals is printed exactly the same way, when the *OED* has a cross-reference to a single letter entry (and there are 26 single letter entries), ICC often incorrectly encoded the cross-reference lemmas as being in roman font, rather than in small capitals. Without the proper font being indicated, the transducer has no way of knowing that this is, in fact, a cross-reference. This will cause many cross-references to be lost, which has a significant impact on the integrity of the *New OED*.

Bouwnd, -en, obs. form of BOUND, -EN.

Mareschal, -cy, -sy, obs. ff. MARSHAL, -CY.

Kegeree variant of KEDGEREE.

Cinosure: see CYNOSURE.

Figure 4.9: Sample Cross-reference Entries

4.5 Learning the Structure of the *OED*

4.5.1 Paradigms Versus Reality

Understanding the fundamental structure of the *OED* consumed a very large part of the development time in writing the transducers. In some respects, this knowledge, ***not*** the transducers themselves, is the most valuable product of the research. The grammar went through a great number of transformations in our attempts to have it model the *OED* properly, not all of which represented unmitigated progress. In the early stages of development of the grammars *INR* was, more than anything, a research tool which could test hypotheses regarding the structure of the *OED*. Although we had the services and expertise of the *OED* lexicographers to draw upon, and they had a great wealth of knowledge regarding the Dictionary, they had not been accustomed to thinking of the Dictionary in purely structural terms, and certainly never attempted to describe the Dictionary as formally as through a grammar. To make matters worse, for every rule which we devised to describe (and hence delimit) a structure in the *OED*, we would eventually and with painful regularity come upon a counter-example. The *OED* contained a huge store of conceptual traps into which we could stumble.

As an illustration of the way that a cursory (or even a reasonably exhaustive) study of the *OED* can lead one to false conclusions concerning its structure we will take a brief look at cross-reference entries. These are entries which merely serve as pointers to guide the reader to other entries in the Dictionary. For instance, if a word has a common written form and one or more uncommon forms, under each of the uncommon spellings there will be an entry which does nothing more than advise the reader to refer to the common spelling for the complete entry. Several examples of cross-reference entries are given in figure 4.9, and their ICC marked-up forms are given in figure 4.10.

The cross-reference entries appear to be nothing but pointers to other parts of the data

+1006 Bouwnd, -en, +SR obs. form of +SC Bound, -en.
+1006 Mareschal, -cy, -sy, +SR obs. ff. +SC Marshal, -cy.
+1006 Kegeree +SR variant of +SC Kedgeree.
+1006 Cinosure: +SR see +SC Cynosure.

Figure 4.10: ICC Mark-up of Cross-reference Entries

and are extremely simple in structure. Upon inspection of the above examples, and indeed upon inspection of the vast majority of such entries within the *OED*, one would be motivated to create something like the simple paradigmatic structure of figure 4.11 to represent them.

In this grammar, a cross-reference entry begins with a '+1006 ' tag (distinguishing it from a main entry, which begins with a '+1000 ' tag) followed by one or more headwords and then a colon or a comma. A '+SR ' tag is then expected, which indicates the beginning of an unnumbered sense-section, consisting of only a linking word followed by one or more cross-references (the pointers to the *true* entries).[6]

Although the grammar of figure 4.11 will accept the vast majority of cross-reference entries, if one were actually to attempt to recognize the *OED* using it, it would inevitably fail, most frequently because the grammar does not specify the inclusion of labels, and does not allow for arbitrary pieces of sense text after the cross-references. However, these are not the only shortcomings of the grammar. In fact, it turns out that, contrary to superficial appearances, *any* structure that can occur in an *OED* main entry can occur in a cross-reference entry as well, although most structures, such as etymologies, variant lists and quotation banks occur with significantly lower frequencies. The "problem of conception" lesson learned from this experience was that a seemingly obvious paradigm for a structure in the Dictionary can often be totally unsuitable and often misleading as a basis for designing a grammar to recognize that structure.

4.5.2 Hierarchical Versus Iterative Grammars

The most fundamental conceptual change that occurred in our understanding of the *OED* was a transition from trying to describe the *OED* as a mainly hierarchical structure, to describing it as a mainly iterative structure. Again, this was a process of moving away

[6]We have not given every detail of this grammar but the rules which we have omitted have their obvious functions, for instance, the PartsOfSpeech rule accepts a part of speech and the HeadwordLemma rule accepts a headword lemma.

```
XRAs          =   Smcaps HeadwordLemma
                  (   HomonymNumber
                  |   SenseNumber
                  |   PartsOfSpeech
                  )*  ;
Headwords     =   HeadwordLemma
                  (   HomonymNumber
                  |   PartsOfSpeech
                  |   Pronunciation
                  )*  ;
LinkingWord   =   {'see', 'variant of', variants of', 'variant form of',
                  'var. ff.', 'obs. form of', 'obs. forms of',
                  'obs. ff.', 'obs. variant of', 'obs. pl. of'} ;
XRefEntry     =   '+1006 ' Headwords {': ', ', '} '+SR '
                  LinkingWord XRAs ;
```

Figure 4.11: A Simple Grammar to Describe Cross-reference Entries

from the seemingly obvious paradigm, which presented itself to us in the early stages of designing the grammars, and divining a more appropriate, more accurate understanding. This process is illustrated quite graphically in the grammar used to recognize variant lists. An early attempt at a grammar to describe variant lists is given in figure 4.12. Like the cross-reference grammars given in figure 4.11 the grammar in figure 4.12 has been simplified somewhat, in order to remove some of the less important details; however, the basic structure and main elements of the grammar are unchanged.

The grammar of figure 4.12 recognizes that variant lists are of two basic types: some run-on text, enclosed by parentheses; or a list, beginning with the words "Forms" or "Also", and containing (in order) labelled variants, dated variants and indexed variants. Labelled variants are a list of variants modified by a geographical or usage label, such as *Sc.*, meaning "Scottish", or *Pa. pple.*, meaning "past participle". A variant list contains one or more variant forms—lemmas in bold font— separated by commas, potentially enclosed within brackets, potentially further labelled. Dated variants are variants preceded by dates and, potentially, labels. Indexed variants are entire variant lists (consisting of labelled and dated variants) each of which are preceded by an index—a greek letter α, β etc. Indexed variant lists represent what was perceived, by the composer of the entry, to be parallel paths in the evolution of a word over the centuries, and so the lists are given separately.

Even a casual examination of this variant forms grammar reveals that it is very complicated, repetitive and convoluted. The same sorts of structures keep appearing over and

```
VariantDate       =   Roman Digit ('+14 ' Digit? )? ;
VariantLabel      =   LabelList | Italic ('vulgar ' | 'erron. ') | Roman 'Pl. ' ;
VariantForm       =   Bold HeadwordLemma (' ' Pronunciation)? ;
BracketedVariant  =   Roman '(' VariantLabel? VariantForm
                          (', ' VariantForm)* Roman ')' ;
VariantList       =   (VariantForm | BracketedVariant)
                          (', ' (VariantForm | BracketedVariant))* ;
LabelledVariants  =   VariantLabel VariantList ;
DatedVariants     =   VariantDate VariantLabel? VariantList
                          (', ' LabelledVariants)* ;
IndexedVariants   =   Index VariantDate? VariantLabel? VariantList
                          (', ' LabelledVariants)* (', ' DatedVariants)* ;
Variants          =   Index? VariantDate? VariantLabel? VariantList
                          (', ' LabelledVariants)*
                          (', ' DatedVariants)*
                          ('. ' IndexedVariants)* ;
VariantForms      =   '+VL ' 2.Wroman 'Also '? Variants '. '
                  |   '+VL ' 2.Wroman '(' ArbText+ ') '
                  |   '+VL ' 2.Wroman 'Forms: ' Variants '. ' ;
```

Figure 4.12: A Hierarchical Grammar to Describe Variant Forms Lists

```
VariantForm   =   Bold VarText+ ;
VariantDate   =   Roman Digit ('+14 ' Digit?)? ;
Variants      =
                  (   Label
                  |   XRA
                  |   Roman TextChars+
                  |   VariantDate
                  |   VariantForm
                  )* ;
VariantForms  =   '+VL ' TextChars+ Variants ;
```

Figure 4.13: An Iterative Grammar to Describe Variant Forms Lists

over again, but with slight variations—the addition of a label or a date, the enclosure of a structure within parentheses, and so on. For us, the logical conclusion of this trend was to abandon the hierarchical view of variant lists, and to move toward what could be described as an iterative view. The iterative view has the shortcoming that it has less inherent structure than does the hierarchical view, but it could be argued that if the hierarchical grammar could not be made to recognize the variety of variant lists, then it does not accurately reflect their structure. It is, in fact, an imposed, inaccurate structure. A slightly simplified version of the iterative grammar for variant lists is given in figure 4.13.

The most striking feature of the iterative grammar is that is considerably smaller and simpler than the equivalent hierarchical grammar. This is because it does not attempt to describe nearly so restricted a set of possible combinations of patterns as does the hierarchical grammar. This renders it simpler to read and more flexible. Furthermore, *INR* creates a much smaller automaton from the iterative grammar, because it does not have to enumerate as many possible combinations of states. This last point was of great importance during the development of the transducer. It was stated earlier that several approaches to the transduction were abandoned because the software could not create a large enough grammar to describe the *OED*, and even though the use of *INR* was less plagued with built-in limitations than the early approaches to the transduction, some physical limitations were inevitable. Thus, the change from a hierarchical approach to an iterative one was very useful in that it produced a much smaller automaton. In general, it was our experience that whenever the grammar became so large that it approached or exceeded one of the system limitations, it was an indication that our approach to describing the *OED* was incorrect—we had encountered a problem of conception.

The iterative grammar accepts the various structures which can appear in a variant list, but without regard to their overall contexts—a label, a date or a variant form can appear

anywhere, in any combination. The various structures are bridged by punctuation and possibly some free-form text (denoted by "Roman TextChars+"), but these pieces of bridge material are only discriminated from the structures themselves by the output tagging. No attempt is made to recognize the position of structures within a hierarchy.[7] This has the shortcoming that we lose some information—for instance, we can no longer determine, from the tagging, which variants are modified by a given date or label. It would also appear to be a rather hazardous approach to the transduction of the data, since it will recognize a great number of meaningless combinations of structures. We do not, however, feel that this will be a problem, because we are not attempting to transduce arbitrary data; we are working with a known piece of text, and so meaningless combinations of structures are rather unlikely. The proof of this is that the *OED* has been heavily edited, and in continuous use for over one hundred years. In addition, all of the transduced text will be carefully proof-read, and so the few mistakes which have been transduced as though they were valid, will be corrected at that stage.

As a final note, there is one major structure of the *OED* which is adequately modelled by a mainly hierarchical structure, and that is quotation banks. The *OED* editors endeavoured to construct quotation banks with more attention to a stereotypical form than has been seen in other parts of the Dictionary. For this reason, they can be adequately represented by a hierarchical structure.

4.6 Transducing Cross-references

The process of transducing cross-references will now be examined in some detail as a typical example of the problems involved in structuring the *OED*, and the solutions which we have found for these problems. Some of these issues have been mentioned in earlier sections, but it is important to see them grouped together in order to appreciate the variety of techniques needed to transduce cross-references, and, by extension, the entire *OED*. We will also mention some situations where the transducer does not adequately, or does not correctly, tag cross-references.

[7] This is not strictly true, in general, as one can see from examining the complete grammars in Appendix B, but the types of distinctions which are made based upon hierarchical position (in XRAs, for instance) are not relevant to this example.

4.6.1 Multiple Cross-references

Multiple cross-references are references from one *OED* entry to several other entries. They come in several styles, and each must be handled differently. The simplest sort of multiple cross-reference is when two cross-references are lumped together by the ICC tagging as one. For example, the etymology for "teleologic", given in Section 3.3.2 is really two cross-references, but since they are only separated by a "+" (which can be represented in the small-capitals font), they are not distinguished as separate structures by the ICC tagging. In cases such as this, we simply recognize the "+" as a separator and tag the two words as separate cross-references.

A more important and more difficult to manage class of multiple cross-references includes those with multiple homonyms or sense numbers. In this case, a single cross-reference with a single lemma points to two or more target entries in the *OED*. For instance, the *OED* entry for "overshoot" as a verb has, in its etymology, a cross-reference to *seven* senses of the prefix "over-", and in the entry for "mackless", there is a cross-reference which points to two different homonyms of "makeless" as an adjective. We have decided to group each of these cross-references together as a single structure. The reasoning for this is twofold: since *INR* (being restricted to dealing with regular languages) does not provide any mechanism to rearrange arbitrary pieces of text, we could not construct complete multiple cross-references from the single one; and even if we could construct these multiple cross-references, this would lead to a structuring of the *OED* which did not obviously reflect its typography. Instead, we have left these multiple cross-references as they are, i.e., we have tagged them as though they were a single reference, deferring, to the cross-reference resolution and integration software, the problems of properly resolving them.

4.6.2 Relative Cross-references

Relative cross-references are context-dependent references to nearby entries in the Dictionary, and are characterized by keywords such as: "next" (meaning the next entry); "prec." (meaning the preceding entry); "sense" (referring to a specific sense within the same entry); and "quot." (referring to a specific quotation within the same entry). These cross-references are not distinguished by any ICC structural or typographical tags, yet it is vital that they are distinguished by the output tagging, because many of these references will change during the integration process, as a result of senses being renumbered and new entries being inserted between two entries that were previously adjacent. These relative cross-references can appear in an entry anywhere that an ordinary cross-reference would, and they occur in roman font. For this reason, they can only be distinguished from ordinary roman text by searching for the above keywords just as with the recognition of *Supplement* commands.

The grammar to achieve this distinction is illustrated in figure 4.14. This figure is taken from a second pass grammar, so it looks different from most of the grammars which

```
ResWord    =   {'quot' 's'? '.', 'next', 'prec.', 'sense' 's'? ' '} ;
Etymology  =   1.RESTART 1.ECHO ' <etym>'
           (   IText ConvertSN?
           |   BText ConvertSN?
           |   (' </enote>' | ' <enote>') Alph* ConvertSN?
           |   XRA
           |   Linkage
           |   ' <R>' ((Alph* ResWord Alph*):acomp)
                    ( (,' <rxra>') ResWord ((,' <sn>') SenseGram (,' </sn>'))?
                    (,' </rxra>') (EndSense Alph* )? ' </R>'
           )*  ' </etym>' Alph* ;
```

Figure 4.14: A Grammar to Recognize Relative Cross-references

we have shown thus far. There is no mention of ICC tags in these rules—they have been replaced with output tags from the first pass. Also, this grammar contains constructions for performing output, such as the (,' <rxra>') and (,' </rxra>') tuples, which have the effect of surrounding relative cross-references with the <rxra> ... </rxra> tag pair. Furthermore, we no longer need to be concerned with keeping track of fonts in this grammar. Each meaningful part of the Dictionary text has been enclosed with a tag pair in the first pass, so there can no longer be ambiguity concerning the current font. Thus, we need only accept and pass over text that we are not interested in further transducing. To do this, we use rules such as BText and IText, for bold and italic text respectively, which accept the begin and end tags, and any text in between. The other interesting innovation in this grammar is the ConvertSN rule which converts text which has been improperly tagged as sense numbers (as determined by their position) back into ordinary roman text, as mentioned in Section 4.3.

Relative cross-references are distinguished from ordinary roman text in the example grammar by the utilization of the ":acomp" operator. After accepting <R>, the begin roman tag, we accept roman text by invoking the following construction: we surround the keywords (found in the ResWord rule) which indicate the start of a relative cross-reference by Alph* rules, and then take the active complement of this entire construction. This ensures that the only strings accepted as ordinary roman text are those with no occurrences of a ResWord in them. If we do find a ResWord, then we preface it with a <rxra> tag, and search for a sense number. The sense number may, in fact, be a quotation date, if the ResWords "quot." or "quots." were the ones accepted, since these are relative cross-references to a single quotation. After accepting a sense number, if one exists, we insert a </rxra> tag, indicating the end of the relative cross-reference. Finally, we accept any remaining roman text, and the terminating </R> tag. In this way all roman text is accepted, and relative cross-references are successfully distinguished.

The only problem with this method is that ordinary occurrences of the words "next" and "sense" in roman text are tagged as relative cross-references, along with the valid occurrences of these words as cross-references (it is unlikely that the other words in the ResWord rule would ever occur in ordinary text). This situation is not optimal, but it does no harm since the original roman text is not lost by being tagged as a cross-reference and it is certainly preferable to missing *true* cross-references. Furthermore, many of these false references will be weeded out by the cross-reference resolution program, which will be run at a later stage in order to match all cross-references to their targets.

4.6.3 Constructing Cross-references

The construction of correct cross-references is accomplished in two stages. The basic elements of cross-reference addresses—cross-reference lemmas, parts of speech, sense numbers, and homonym numbers—are identified in the first transduction pass, and then they are collected together and surrounded by the <xra> ... </xra> tag pair in the second pass. Some of the mechanisms used to identify the components of cross-references have already been discussed, specifically: the identification of parts of speech by the data normalization program and the identification of sense numbers. We will not discuss these mechanisms any further here, but will concentrate on the remaining steps in identifying cross-references: identifying homonym numbers and lemmas and putting the pieces together.

The identification of homonym numbers is straight-forward, since they are identified unambiguously by the ICC '+HM,' tag. The only complication created by homonym numbers is that the '+HM,' tag has the side effect of setting roman font so that after accepting a homonym number, we must optionally accept arbitrary roman text, as shown in figure 4.15.[8] Accepting roman text also necessitates the usual checking for sense numbers, utilizing Sensegram.

The identification of cross-reference lemmas has already been discussed somewhat in

[8] A few ICC tags have a side effect of setting a particular font. The majority of these are tags which begin a major section in an entry, such as '+QP ', the begin quotation paragraph tag, which sets bold font, or '+LA ', the begin label list tag, which sets italic font. The homonym number tag, along with a few others, such as '+AI ' and '+CI ', which represent the symbols for "ante" and "circa" respectively, set a font (bold) and *do not* begin major sections—in all other respects they are ordinary ICC tags representing a single character or function. One can only postulate that the justification for the anomalous use of these tags is that when ICC was designing their tags, they noticed that certain tags were virtually always followed by the same font change, so they included the font change function into the tag itself, reasoning

```
HomonymNumber =   1.NOECHO ('+HM,', ' <hom>') 1.ECHO Number (,' </hom>')
                  ( (,' <sn>') SenseGram (,' </sn>')
                        ((,' <R>') EndSense TextChars* (,' </R>'))?
                  || (,' <R>') TextChars+ (,' </R>') )? ;
```

Figure 4.15: A Grammar Rule to Transduce Homonym Numbers

```
XRLemma  =   ASmcaps  {'?', '+17 ', '(', '"}? (,' <xlem>') {Letter, '-'}
             ( HWChar | HWChar '+17 ' HWChar
             | (,' </xlem>') {'=', '+22 ', ', ', '; '} (,' <xlem>') HWChar )*
             (,' </xlem>')
             ( (,' <sn>') Digit {'.', ',', ' '}* (,' </sn>')
             | (,' <R>') {':', ';', ',', '*', '!', '=', '+15 ',
                  '+18 ', '+16 ', '+22 ', '+63 ', '+64 '}+ (,' </R>') )? ;
```

Figure 4.16: A Grammar Rule to Transduce Cross-reference Lemmas

Sections 3.3.2, 3.5.1 and 4.3. It is one of the most difficult tasks in the transduction process. For the most part, though, we have concentrated on detailing how the transducer recognizes, through context, whether it is dealing with a cross-reference lemma or with some other structure after it has accepted a '+SC ' tag from input. Now we will illustrate how we tag cross-reference lemmas once we have determined that that is indeed what we are transducing. The grammar rule for tagging these lemmas is given in figure 4.16.

It is obvious that cross-reference lemmas have a rather complex structure. The lemma (or group of lemmas) begins with the '+SC ' tag, followed by an optional question mark, apostrophe, left bracket or '+17 ' tag, which is the ICC tag for a left single quotation mark ('). Any of these characters may precede a cross-reference lemma (due to of the ambiguity of the small-capitals font—recall the examples given in figure 4.8) but we do not want to tag them as belonging *within* the lemma, so it is only after accepting one of these tags, if they are present, that the <xlem> tag is written. Next, a letter or a hyphen is accepted (these being the only valid characters at the beginning of a lemma) and then zero or more headword characters are accepted, as defined by the HWChar alphabet. There are three different ways that characters may be accepted in this part of the XRLemma rule. If the character is a normal upper or lower-case letter, an accent, or one of a few special characters (such as Æ) then it is accepted without further ado. If it is a left single quotation mark, then

that doing so would save useless key strokes.

it must be immediately followed by another HWChar character for it to be considered part of the lemma (this is to distinguish it from the '+17 ' at the beginning of the lemma, which we have excluded). Finally, if the character is what we have deemed to be a separator—an equals sign, a plus sign (the '+22 ' tag), a comma or a semi-colon—and this is followed by another HWChar character, then we believe that the separator is signalling the end of one headword and the beginning of the next, and so we precede the separator with the end cross-reference lemma tag (</xlem>), accept the separating character itself, and then write a *new* <xlem> tag on output, and continue accepting characters. In fact, we must accept at least one more character from the HWChar alphabet, which is why the separators are followed by the HWChar rule. In this way, we ensure that the character is, in fact, a separator between lemmas, and not just a character separating a cross-reference lemma from an ordinary piece of text. The final part of the XRLemma rule is concerned with tagging the characters which follow a cross-reference lemma. Notice that this is where a plus sign, equals sign, comma or semi-colon (along with a host of other characters) will be accepted, if they *terminate* the lemma. Notice also, that one or more digits may be accepted here, and are tagged as sense numbers. By rights, all of these characters should be preceded by a '+R ' but once again, this does not necessarily happen due to ambiguity in the meaning of the small capitals font.

The grammar which assembles the various parts of cross-references and packages them together is given in figure 4.17. This grammar accepts cross-references which begin with a single lemma, and which contain any number of homonyms, parts of speech and sense numbers. A homonym number may be optionally preceded by the word "and" in roman font, in order to allow for cross-references to two homonyms of the same morpheme, as was described in Section 4.6.1. The DownCase rule was added into the grammar for transducing cross-reference addresses to conform the majority of *OED* cross-references to the *Supplement* standard. The only headwords which have their first letter capitalized in the *Supplement* are proper names and acronyms. The DownCase rule is not, unfortunately, so selective and so its mistakes will have to be corrected manually for the integrated *OED* to be consistent. Fortunately though, the relative number of proper names and acronyms in the *OED* is quite small compared with the *Supplement*, so the amount of manual correction that this will create is minimal.

The technique used to accomplish the "downcasing" is worth a brief note. When the <xlem> tag is accepted, the closing angle bracket (>) of the tag is not immediately output. Instead, if the lemma begins with an upper-case letter, the tag is closed with the "down" parameter, indicating that the lemma has been downcased, and the lemma's initial capital letter is replaced with the appropriate lower-case one. If the lemma already begins with a lower-case letter (as with the second headword in a cross-reference to a multiple headword entry) or with a hyphen (as with the suffix "-ing") or an ICC tag (as with the entry for "ætiology" which begins with a '+923 ' tag), then the closing angle bracket of the <xlem> tag is simply replaced and the character is output unchanged.

```
DownCase =
        (   1.NOECHO (,'down>')
            {(A,a), (B,b), (C,c), (D,d), (E,e), (F,f), (G,g),
            (H,h), (I,i), (J,j), (K,k), (L,l), (M,m), (N,n),
            (O,o), (P,p), (Q,q), (R,r), (S,s), (T,t), (U,u),
            (V,v), (W,w), (X,x), (Y,y), (Z,z)}
            1.ECHO
        |   (,' >') {LCase, '-', '(', '+923 ', '+23 '}
        )   ;
XRA     =   (,' <xra>') ' <xlem' 1.NOECHO ' >' 1.ECHO DownCase Alph* ' </xlem>'
        (   ' <pos>' Alph+ ' </pos>'
        |   ' <sn>' Alph+ ' </sn>'
        |   ' <R>and </R>'? HomonymNumber
        )*  (,' </xra>') ;
```

Figure 4.17: A Grammar to Tag Cross-reference Addresses

4.6.4 Problems Transducing Cross-references

Most of the problems in transducing cross-references in the *OED* have been mentioned at one point or another in this document, but we will review them briefly and mention one additional problem. The most common difficulty that we encounter is that the transducer will incorrectly label some ordinary pieces of Dictionary text occurring in the small capitals font as cross-reference lemmas. Furthermore, it will miss some lemmas entirely due to improper tagging, such as those referencing single letter entries. Also, it will incorrectly split a reference to a multiple headword entry into two (or more) cross-references. For example, a cross-reference to the entry "kinder, kirche, küche" would be incorrectly split into three separate cross-references.

In theory, a piece of roman text following a cross-reference could become improperly labelled as a sense number, if it had the superficial form of a sense number. We haven't noticed any examples of this particular error in the first 500 pages of the dictionary, but if it does occur, then it will not likely be noticed until the cross-reference resolution software processes the transduced text.

Cross-references to italic lemmas occur quite frequently within the Dictionary. Italic lemmas are compound words listed under a main entry wherein each of the participant words in the compound retains their original meaning, and yet the compound is used frequently enough to bear mention. Cross-references to italic lemmas refer to either a lemma in the same entry or in another entry. If it occurs in the same entry, the lemma alone is used as a (relative) cross-reference, but if it occurs in a different entry, the italic lemma will precede

CHIASTOLITE
MAKELESS *a.*[1] and [2]
LIE *sb.*[1] 2 b.
(see quot. 1851).
as (sense 1)

Figure 4.18: Sample Cross-reference Addresses

+SC Chiastolite
+SC Makeless +I a. +HM,1 and +HM,2
+SC Lie +I sb.+HM,1 2+64 b.
+R (see quot. 1851). +63
+R as (sense 1)

Figure 4.19: ICC Mark-up of Cross-reference Addresses

the true cross-reference address. Neither of these types of cross-references are, as yet, tagged by the transducer, although it is certainly within the capabilities of the grammars.

4.6.5 Results of Transducing Cross-references

In figures 4.18, 4.19 and 4.20, we have given several examples of cross-reference address, as they appear in the Dictionary, as they appear in their ICC marked-up forms, and as they appear after the transduction process. The first example is the simplest form of cross-reference possible—a single lemma with no part of speech, homonym or sense number. The second example is the one described in Section 4.6.1—a reference to two homonyms of the word "makeless" as an adjective. The third example includes every possible part of a cross-reference address: a lemma, part of speech, homonym number and sense number (the '+64 ' in the sense number is the ICC tag for a thin space). Notice that in each of the transduced lemmas, the first letter has been downcased.

The final two examples in figures 4.19 and 4.20 are relative cross-references. The first one is a reference to a quotation within the same entry. Its quotation date is tagged as a sense number—this is not a really descriptive name, but the date has the same function that a sense number has in an ordinary cross-reference. The second example is a reference to a

<xra><xlem>chiastolite </xlem></xra>

<xra><xlem>makeless </xlem><pos>a.</pos><hom>1</hom>
<R>and </R><hom>2</hom></xra>

<xra><xlem>lie </xlem><pos>sb.</pos><hom>1</hom><sn>2 +64 b. </sn></xra>

<R>(see <rxra>quot. <sn>1851</sn></rxra>). +63 </R>

<R>as (<rxra>sense <sn> 1</sn></rxra>) </R>

Figure 4.20: The Transduced Form of Cross-reference Addresses

different sense within the same entry. Notice that in both examples the <rxra> ... </rxra> tag pair occur *within* the roman font tag pair. This is because relative cross-references, in contrast with most structures in the Dictionary, are not denoted by being in a special font or by having a special structure, they are simply interesting pieces of roman text.

Chapter 5

Conclusions

5.1 The Future of Grammars in the *New OED* Project

It is certain that there will be potential uses for grammars in the *New OED* project for quite some time. The current grammars will, of course, continue to be refined and modified so that they better model and, consequently, tag the *OED*. The refinement of the grammars has been an ongoing task at the Oxford University Press, and the results have been encouraging (although the continued expansion of requirements upon the grammars has, at present, necessitated the introduction of a third *INR/lsim* pass). In recent months, changes have been made to the grammars to begin to grapple with the problems of: cross-references to italic lemmas (discussed in Section 4.6.4); better distinguishing ordinary cross-references from quotation authors or other text in the small capitals font; and adding end tags to match the begin tags of structures such as <enote> and <snote> (etymological and sense notes, respectively) which are not explicitly denoted by ICC tagging, and the boundaries of which are difficult to detect because they span many other structures.

Progress in the development of the grammars is still far from linear. For instance, we have recently decided that there are some situations where we need the '+0 ' tag (representing a blank) which was removed by the data normalization program, described in Section 4.2. Consequently, we will have to change both the data normalization program and the grammars to accommodate this tag. Nevertheless, removing the '+0 ' tag was beneficial for the same reason that breaking the grammar up into smaller specialized grammars was beneficial. It allowed us to concentrate on one particular aspect of the problem,

without having to worrying about all other details simultaneously. Now that we have had the experience of creating working grammars, it will be a relatively trivial task to go back to them and re-insert the recognition of '+0 ' tags in the appropriate locations.

Another development in recent months has been the implementation of a grammar to describe and tag mathematical and chemical formulas. These formulas occur rarely in the *OED*, and more frequently, although still not often, in the ***Supplement***, with its heavier emphasis on scientific, mathematical and technical language. The ICC representations of these formulas must be transduced if they are to be understood and typeset properly. This involves delineating the scope of important structures in the formulas, and translating this knowledge into the appropriate mark-up. There are many other areas for possible development in the grammars, such as attempting to distinguish acronyms from other headwords, so that they are not (incorrectly) downcased, distinguishing pronunciations in variant lists and sense sections, tagging cited forms in etymologies, and so on. Furthermore, new uses for the transducers are constantly presenting themselves.

The grammars will be of great importance during the first and second phases of the *New OED* project and beyond because they represent the structure of the Dictionary. Any intelligent piece of software that wishes to exploit the transduced text of the *OED* will have to take into account its structure, as represented by the grammars. This, however, does not necessitate the use of *INR* and *lsim*. In fact, now that the Dictionary text has a fully bracketed syntax, there is a powerful argument for representing the *OED* as a context-free language rather than a regular language. Having end tags for every structure means that the transduced text can be easily described as a context-free language, which is not true of the ICC tagged text. Furthermore, being able to describe the Dictionary as a context-free language allows the grammar to describe and check certain features of the dictionary easily. For example, sense hierarchies could be described, which the transduction process is currently unable to do (see Section 3.5.2).

If *INR* is to be used effectively and widely in the *New OED* project then it is imperative that we address two problems. *INR* must be somehow rendered more palatable to the layman, and we must develop more rigorous ways of verifying the correctness of the grammars. It is our belief and our experience that *INR*, in its current form, cannot be quickly mastered by a novice, even if that novice already has considerable expertise in other areas of computer science. One of the difficulties has to do with teaching someone to *think* in terms of regular languages. Another difficulty is in dealing with the notational complexity of *INR*, and, more noticeably, of the *OED* itself.

The transduction process has "tamed" the complexity of the *OED* somewhat, by imposing a considerable amount of order upon what was largely free-form text, and by removing font considerations from most of the output. Any further manipulation of the transduced text will be much simpler than the original transduction process because there are far fewer details with which to concern oneself. The *OED* has been given a regular, consistent format, free of anomalies.

In Section 3.3.4, we discussed possible uses for *INR/lsim* in the future of the *New OED* project. One of the most attractive and practical uses is as a generalized research tool for the *New OED*, where users could compose their own specification of the parts of the text which they wanted to extract, as well as optionally specifying some conditions which the extracted text had to meet. For instance, a common request of users is to extract all earliest quotations for words which meet a certain condition, say, that the etymology for the entry indicates that the word came from Old Frisian. We may not finally use *INR* and *lsim* to accomplish this task, but they are convenient tools for testing different methods of manipulating the *OED*, and different types of user interfaces.

The question of user interface design is of tremendous importance to the *New OED* project. *INR*, for instance, would be of far wider utility if it possessed a more amenable front end. We are currently investigating a front end for a generalized research tool which moves away from the representation of the *OED* as a regular language. In this tool, a user specifies the desired grammar by "filling in a form", in much the same way as users can compose a database query in a Query-By-Example system.[1] The user is presented with a paradigm of an *OED* entry, and merely checks off the elements which are to be extracted (such as headword and first quotation), and any special criteria which those elements must possess for them to be accepted (such as the inclusion of the string "OFris." in the etymology). The front end then translates this request into an appropriate query, using *INR* or some other tool. This will not likely be an adequate approach for *every* user query, but will, based upon our current experience, satisfy the majority of requests.

The concept of describing the *OED* with a grammar has widespread applicability. At present, neither *Goedel* (the generalized extraction tool developed for manipulating the transduced text of the *OED*) nor *LEXX* (the text editor written for the *New OED* project) take into account the structure of the dictionary which they are manipulating. Both programs understand the <tag> matches </tag> convention, but little more. With a knowledge of the structure of the *OED* these programs could perform far more sophisticated services for the user. For instance, *LEXX* could ensure that lexicographers, entering or modifying *OED* text, only use valid characters in a given scope. In fact, *LEXX* could become a very powerful "critic" of newly composed or modified entries in the Dictionary, checking the hierarchical structure and style of entries. The use of grammars to dictate the allowable form of entries which have been manually composed or altered is one which is easily within the grasp of our current expertise, and which is vitally important to the ensured integrity of data in the *New OED*. We expect to be pursuing this goal in the near future.

A knowledge of the structure of the Dictionary would enable a user of *Goedel* to state many queries very simply which can only be satisfied currently through composing long, complex programs. For example, a user could extract only those quotations which contain dates between 1600 and 1605 and which have Shakespeare or Bacon as the author, and so

[1] See Chapter 14 of [Dat85] for a simple explanation of Query-By-Example.

on. This sort of query is simple with a knowledge of the grammar, but rather complicated without it.

We have already had considerable success with *INR/lsim* as a research tool, and with the grammars as a description of the *OED*, and we are confident that we can continue to use these, as well as using our experience within the grammar writing process to design better, more powerful tools.

5.2 Transducing Other Dictionaries

Other potential applications for the techniques developed include transducing other dictionaries in the same manner as the *OED*. The benefits of doing so are numerous. The savings in manual effort are enormous—instead of using manual effort or laborious ad hoc programs to conform a dictionary to a model, as many computerized dictionary projects have done, we spend our time learning the grammar of the dictionary, and once this is accomplished, we can produce the necessary tagging automatically. Having had the experience of designing the grammars for the *OED*, we are confident that we could apply substantially the same techniques to other dictionaries and create grammars in a fraction of the time taken in the *New OED* project (eight months). In addition, any dictionary which we attempt to transduce in the future will almost certainly be smaller and more regular than the *OED*, so the grammars that we create to describe them will be simpler and likely more reliable. A further benefit is that we can use the same format of output tagging as was used with the *OED*, thus moving an important step closer to creating a common structure for all dictionaries—a highly desirable objective, not only because we can utilize all of our current software without modification but because the dictionaries will doubtlessly complement each other. At the same time, we can normalize pronunciations, labelling and part of speech conventions, and so on, either within a given dictionary or between all of our transduced dictionaries.

5.3 Verification of the Grammars

Aside from improving the "user friendliness" of *INR*, it would be advantageous to develop some sort of measure of reliability of the grammars, particularly when they are in a state of flux (as occurs whenever we attempt to increase the descriptive power of the grammars). The development of the grammars has always been an uncertain process. In effect, we wrote grammars primarily by looking at a large number of valid examples of Dictionary entries and then conjecturing a model to describe those entries. We had no examples of entries which were *not* valid in the *OED*. Consequently, we had a simple method of testing whether our grammar was too restrictive, based upon its performance in transducing a given sample

of *OED* text, but we had no way of testing whether the same grammar was too permissive. We could, and did, hypothesize upon the structure of entries, the possible combinations of sub-structures within a structure, and the allowable characters in specialized alphabets, but we had no way of absolutely verifying these hypotheses short of processing every character in the Dictionary. In fact, it is certain that the grammars describe, and would accept, theoretical *OED* entries which would never be composed or allowed by a lexicographer. This is because the grammars for the *OED* and *Supplement*, like grammars of natural languages, and unlike those of computer languages, are descriptive, not prescriptive.

We did have the significant benefit of ready access to the Oxford University Press's resident *OED* lexicographers during the development of the grammars. They could advise on the probable structure of the Dictionary, the meanings of various notational conventions, the correctness of the output tagging, and so on, but the *OED* is such a large work, that no human could ever be expected to remember or understand the dizzying variety of structures which comprise it.

In the end, we had to resort to visual inspection of the output data, and periodic checks by the Oxford University Press lexicographical staff, as our "measure" of reliability. This situation is changing though, and is certain to change more. As more and more people and programs use the transduced form of the text—people editing the text with *LEXX*, analyzing the text with *Goedel*, manipulating the text with the integration and cross-reference resolution programs—errors in the transduction will be much more likely to be noticed. Furthermore, every piece of transduced text will be composed into a content proof, and possibly a structure proof as well, and will be proof-read.

For a time, we considered the concept of creating some software which could analyze a piece of transduced text in comparison to either a previous version or some paradigmatic version of the same text. In this way, whenever a change was made to the grammars, there would be a test made to see how the output was affected and whether the results of the change actually brought the transduced text closer to its ideal form. This program was never developed because it was felt that it would not necessarily be a meaningful indicator of the state of grammar development. This technique could detect whenever a change to the grammars had significantly degraded the quality of the output, a situation also quickly and easily spotted by a human. The major problem with the technique is that, as with many other aspects of computerizing the *OED*, the enormous amount of data argues against the universal validity of such an approach. In effect, one could never be sure that, just because a chosen sample of text was transduced correctly, the grammars are equally valid for *all* parts of the Dictionary—the grammars might only be "fine-tuned" to the test data, and nothing more.

However, there are some tools which would be helpful in aiding the verification of the grammars—tools which one could use to determine what are the effects of a given change in the grammars. The first of these tools would be programs which could compute and display the differences between two grammars and between two pieces of transduced text.

The hope is that, given these reports, an experienced user could determine which changes to the grammar caused specific changes in the text, and if the differences represent some sort of progress in the tagging process. Also, if the new version of the data is *less* correct than the old version, the user could determine which change in the grammars caused the error.

Another tool which would be useful for grammar verification would be a program which could create different "views" of a grammar. Until now, the only way of proving the grammars was through simple manual examination of them, as well as the output which they created. This process could be facilitated by being able to extract subsets, or views, of the grammar. For instance, we could extract only the input specification of the grammar (the tape 0 transitions), or show the entire structure of the grammar, but without giving any of the low-level details, such as the contents of various alphabets, punctuation, font handling details (i.e., tape 2 transitions), and so on. In this way, a user could concentrate on the single aspect of the grammar which was of interest at the moment, and nothing else.

Another useful technique would be to be able to enumerate the valid strings accepted by an arbitrary part of the grammar. This would be quite useful for validation of the grammar, for it would be quite a trivial task for a lexicographer to examine this list and note any invalid constructions which are being accepted and, more importantly, any valid constructions which are being overlooked. This can be done currently, but each enumeration involves writing a special grammar, and then enumerating all valid strings in the resulting automaton. We do, however, possess a tool for examining an automaton created by *INR*, so this could conceivably be extended to provide this function.

5.4 *INR/lsim* in the First Phase of the *New OED* Project

In retrospect, the choice of *INR* and *lsim* as tools to transduce the *OED* seems to have been a judicious one. The transduced text of the *OED* meets all of the global requirements outlined in the *New OED* System Design Document [New85] and the specific requirements defined in the *New OED* Workbook ([Wei85a] and [Wei84a]), renders the text more readable and more meaningfully tagged (since the output tags are entered by a computer, and not a human, they could be made as verbose as we liked), provides a sufficiently comprehensive level of tagging to support typesetting programs which can faithfully reproduce the *OED* or *Supplement*, and creates a rigorous structure on top of the *OED* text to support the many pieces of software which will manipulate it: *Goedel, LEXX* and the integration and cross-reference resolution programs.

In addition, the transduction process itself has aided the proof-reading process, by detecting many errors in the source text which were missed by teams of proof-readers both at

the Oxford University Press and ICC.[2] All this is accomplished relatively efficiently. The transducers can process about five pages of Dictionary text, approximately 75,000 characters, per minute on a moderately loaded system. Furthermore, because the transducers are automata and the amount of lookahead needed to resolve ambiguities is relatively constant from grammar to grammar, this speed is, in our experience, independent of the size and complexity of the grammars used to create them.

For these reasons, we feel that the use of finite state transduction as an approach to structuring the text of the *OED*, and *INR/lsim* as tools to realize this approach, was well justified and highly successful.

[2]In a recent batch of data from ICC containing 33.9 million characters, the transduction process reported 419 errors. Of these, 311 were visible on the proofs prepared by ICC, and could have been spotted by the proof readers, who in fact spotted 126 of them.

Appendix A

The ICC Tags

ICC Tag	*Meaning*
+AI	ante symbol
+AR	artwork
+B	start bold font
+BB	start of bridge over chemical formula
+BC	start chemical formula
+BE	end of bridge over chemical formula
+BF	start fraction
+BL	start bold lemma
+BS	start of bonded character
+BX	end of bonded character
+CE	end over- or under-text
+CI	circa symbol
+Cn	start of left-justified column *n* in a table
+Cn,ce	start of centred column *n* in a table
+Cn,qr	start of right-justified column *n* in a table
+CO	start over-text
+CU	start under-text
+DB,Ln	lower-left start of *n*-bond
+DB,Rn	lower-right start of *n*-bond
+DE	end of a division (no rule line)
+DN	start of denominator

ICC Tag	*Meaning*
+DP	inferior in pronunciation
+DT,Ln	upper-left start of *n*-bond
+DT,Rn	upper-right start of *n*-bond
+EB	right square bracket (in etymology)
+EC	end chemistry formula
+ED	end of a division (with rule line)
+EF	end fraction
+EL	end lemma
+EN	start etymological note
+EO	end overscore
+ET	start etymology section
+FB	formula start
+FE	formula end
+FS	foreign text
+G	gothic font
+GB	gothic bold font
+GI	gothic italic font
+H	hyphen
+HM	homonym number
+HW	alternate headword
+I	start italic font
+IA	lower-case alpha index
+IB	lower-case beta index
+ID	lower-case delta index
+IE	lower-case epsilon index
+IG	lower-case gamma index
+IH	lower-case eta index
+IL	start italic lemma
+IN	start of inferior script
+IR	reset sense number, start italic font
+IT	index lower-case theta
+IZ	index lower-case zeta
+LA	start of label list
+LB	start of link under chemical formula
+LC	page/column indicator (removed)
+LE	end of link under chemical formula
+LL	label (added during transduction)
+M	start medium size type
+NL	new line in a table

ICC Tag	*Meaning*
+NT	explanatory note
+NU	start of numerator
+OB	left square bracket (in etymology)
+OV	start overscore
+P	new paragraph in the text
+PP	part of speech (added during transduction)
+PR	start of pronunciation
+PS	start part of speech list (headword section)
+QN	start quotation
+QP	start quotation paragraph
+QT	start quotation text
+R	start roman font
+RB	return to baseline
+RR	reset sense number, start roman font
+S	start small size type
+SC	start small capitals font
+SN	start numbered sense section
+SP	start unnumbered sense section; new paragraph
+SR	start unnumbered sense section
+SS	start numbered sense section; new paragraph
+SU	start of superior script
+TE	end of table
+Tn	start of table with *n* columns
+UP	superior in pronunciation
+VB,n	bottom-centre start of vertical *n*-bond
+VL	start variant list
+VT,n	top-centre start of vertical *n*-bond
+0	word space (removed)
+1	macron (floating accent)
+2	umlaut (floating accent)
+3	acute (floating accent)
+4	grave (floating accent)
+5	circumflex (floating accent)
+6	breve (floating accent)
+7	hook (floating accent)
+8	apostrophe (floating accent)
+9	quote (floating accent)
+10	two dot ellipsis
+11	single stress dot

ICC Tag	*Meaning*
+12	double stress dots
+13	em-dash
+14	en-dash
+15	left square bracket
+16	right square bracket
+17	open single quote
+18	close single quote
+19	reverse r
+22	plus sign
+21	schwa
+23	ae diphthong
+24	oe diphthong
+25	zh (in pronunciation)
+26	Middle English lower-case yogh
+27	Old English thorn
+28	Old English thorn with bar
+29	equals sign
+30	nj (in pronunciation)
+31	dagger
+32	double vertical lines
+33	paragraph symbol
+34	syllable bar
+35	divided d
+36	page mark
+37	open double quote
+38	close double quote
+39	lower-case d with bar
+40	lower-case o with slash
+41	upper-case A with T over
+42	b with bar
+43	Arabic apostrophe
+44	fat a
+45	th (in pronunciation)
+46	h with bar
+47	Old English lower-case yogh
+48	Polish lower-case el
+49	German z
+50	hacek (floating accent)
+51	angstrom (floating accent)
+52	dot above (floating accent)

ICC Tag	*Meaning*
+53	dot below (floating accent)
+54	cedilla (floating accent)
+55	frown (floating accent)
+56	tilde (floating accent)
+57	underline (floating accent)
+59	circle below (floating accent)
+60	iota subscript (floating accent)
+63	wide space
+64	thin space
+65	t with umlaut below
+66	s with umlaut below
+74	lower-case digamma
+75	lower-case variant sigma
+76	lower-case alpha
+77	lower-case beta
+78	lower-case gamma
+79	lower-case delta
+80	lower-case epsilon
+81	lower-case zeta
+82	lower-case eta
+83	lower-case theta
+84	lower-case iota
+85	lower-case kappa
+86	lower-case lambda
+87	lower-case mu
+88	lower-case nu
+89	lower-case xi
+90	lower-case omicron
+91	lower-case pi
+92	lower-case rho
+93	lower-case sigma
+94	lower-case tau
+95	lower-case upsilon
+96	lower-case phi
+97	lower-case chi
+98	lower-case psi
+99	lower-case omega
+100	left brace
+101	right brace

ICC Tag	*Meaning*
+102	section mark
+103	English pound
+104	copyright symbol
+105	index finger
+106	degree
+107	three dot ellipsis
+108	double em-dash
+109	paragraph symbol
+110	large blank
+111	cent symbol
+112	inverted semi-colon
+201	minus symbol
+202	multiplication symbol
+203	division symbol
+204	single prime
+205	double prime
+206	vertical bar
+207	square root
+208	union symbol
+209	intersection symbol
+210	left subset symbol
+211	right subset symbol
+212	greater than symbol
+213	less than symbol
+214	stress symbol
+215	integral symbol
+216	identical symbol
+217	swing symbol
+218	plus/minus symbol
+219	infinity symbol
+220	partial symbol
+221	element symbol
+222	not equals symbol
+223	greater than/equals symbol 1
+224	less than/equals symbol 1
+225	greater than/equals symbol 2
+226	less than/equals symbol 2
+227	greater than/equals symbol 3
+228	less than/equals symbol 3

ICC Tag	*Meaning*
+229	right arrow symbol
+230	left arrow symbol
+231	double arrows
+232	triangle
+233	upside down triangle
+234	sideways triangle
+235	V up
+236	V down
+237	left angle bracket
+238	right angle bracket
+239	raised dot
+240	open box
+241	single bond
+242	double bond
+243	percent symbol
+244	therefore symbol
+245	triple prime
+246	consequence symbol
+270	oversized left square bracket
+271	oversized right square bracket
+272	oversized left brace
+273	oversized right brace
+274	oversized left round bracket
+275	oversized right round bracket
+276	oversized slash
+277	oversized summation symbol
+278	oversized integral symbol
+300	trademark symbol
+301	male symbol
+302	Rx (pharmaceutical) symbol
+303	3 asterisks
+304	Maltese cross
+305	macron, not floating
+306	Old English z yogh
+307	Phoenecian M
+308	early Greek M
+309	early Latin M
+310	3/4 symbol
+311	6/8 symbol

ICC Tag	*Meaning*
+312	double l with swing
+313	raised caret
+314	Hebrew waw
+315	Hebrew yod
+316	Hebrew he
+317	musical flat symbol
+318	musical note
+319	musical sharp symbol
+320	medial symbol
+321	Mercury symbol
+322	star sign
+323	mid vowel symbol
+324	minority symbol
+325	script M
+326	script U
+327	Mostra
+328	short mordent
+329	long mordent
+330	sentence symbol
+331	Christ symbol
+332	assertion sign
+333	Hebrew apostrophe
+334	acute, non-floating
+335	grave, non-floating
+336	circumflex, non-floating
+337	tilde, non-floating
+338	cedilla, non-floating
+339	morpheme symbol
+340	ankh symbol
+341	Old English *and* symbol
+342	script ampersand
+343	al segno symbol
+344	accolade symbol
+345	at sign
+346	opposition symbol
+347	sextile aspect symbol
+348	femal sign
+349	bullseye sign

ICC Tag	*Meaning*
+923	upper-case ae diphthong
+924	upper-case oe diphthong
+925	upper-case zh (in pronunciation)
+926	Middle English upper-case yogh
+927	Old English upper-case thorn
+939	upper-case d with bar
+940	upper-case o with slash
+947	Old English upper-case yogh
+948	Polish upper-case el
+976	upper-case alpha
+977	upper-case beta
+978	upper-case gamma
+979	upper-case delta
+980	upper-case epsilon
+981	upper-case zeta
+982	upper-case eta
+983	upper-case theta
+984	upper-case iota
+985	upper-case kappa
+986	upper-case lambda
+987	upper-case mu
+988	upper-case nu
+989	upper-case xi
+990	upper-case omicron
+991	upper-case pi
+992	upper-case rho
+993	upper-case sigma
+994	upper-case tau
+995	upper-case upsilon
+996	upper-case phi
+997	upper-case chi
+998	upper-case psi
+999	upper-case omega
+1000	start main entry (added during transduction)
+1006	start cross-reference entry (added during transduction)

Appendix B

The Complete INR Dictionary

Grammars

The grammars given in this appendix are not meant to be viewed as final authorities on the structure of an *OED* entry. They are not foolproof, and do not achieve everything which one could hope for in disambiguiating and tagging the text of the Dictionary. Rather, they are examples of grammars at a fairly advanced stage of development. These examples are given as useful illustrations of complete *working* grammars which attempt to describe the *OED*. They are still being developed and refined at the Oxford University Press, as of the completion of this work. The copyright for these grammars rests with the Oxford University Press.

B.1 The Pass 1 Dictionary Grammar

```
Nl          =   '' ;

Bold        =   1.NOECHO '+B ' 1.ECHO 2.Wbold | 2.Rbold ;
Italic      =   1.NOECHO '+I ' 1.ECHO 2.Witalic | 2.Ritalic ;
Roman       =   1.NOECHO '+R ' 1.ECHO 2.Wroman | 2.Rroman ;
Smcaps      =   1.NOECHO '+SC ' 1.ECHO 2.Wsmcaps | 2.Rsmcaps ;
ABold       =   1.NOECHO '+B ' 1.ECHO 2.Wbold ;
AItalic     =   1.NOECHO '+I ' 1.ECHO 2.Witalic ;
ARoman      =   1.NOECHO '+R ' 1.ECHO 2.Wroman ;
ASmcaps     =   1.NOECHO '+SC ' 1.ECHO 2.Wsmcaps ;

Digit       =   {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ;
LCase       =   {a, b, c, d, e, f, g, h, i, j, k, l, m,
                 n, o, p, q, r, s, t, u, v, w, x, y, z} ;
UCase       =   {A, B, C, D, E, F, G, H, I, J, K, L, M,
                 N, O, P, Q, R, S, T, U, V, W, X, Y, Z} ;
Letter      =   LCase | UCase
            |   '+' {'H', 'UP', 'SU', 'RB', '35'} ' '
            |   '+' '9'? {'23', '24', '25', '26',
                 '27', '39', '40', '47', '48'} ' ' ;
Accent      =   '+' {'1', '2', '3', '4', '5', '6', '7', '8', '9',
                 '5' Digit, '60', 'OV', 'EO'} ' ' ;
Foreign     =   '+' {'FS', '43', '46', '49', '48', '948', '75'} ' '
            |   ('+' '9'? {'76','77','78','79',{8,9} Digit}) ' ' ;
Punctuation =   {'.', ':', ';', ' ', '?', '!', ',', '-'} ;
QuoteMark   =   '"' | '+' {'17', '18', '37', '38'} ' ' ;

GrabBag     =   {'+22 ', '%', '&', '=', '/', '(', ')', '*', '$'} ;
Special     =   ('+' {'AI', 'AMP', 'BF', 'CE', 'CI', 'CO', 'CU', 'DE',
                      'DN', 'ED', 'EF', 'FB', 'FE', 'G', 'GB', 'GI',
                      'IN', 'M', 'NL', 'NU', 'P', 'RB', 'S', 'TE',
                      '10', '11', '12', '13', '14', '19', '20', '21',
                      '28', '30', '32', '34', '36', '44',
                      '45', '50', '52', '63', '64', '65',
                      '110', '111', '112', '241', '242', '243', '244',
                      '246', '274', '275', '277', '278',
                      '342', '348', '349', '353'} ' ')
            |   ('+' {'10', '20', '21', '22', '23', '30', '31', '32',
                      '33', 'T'} Digit ' ')
            |   ('+C' Digit ',' {'qr', 'ce'} ' ') ;
```

```
Index          =  '+' {'IA', 'IB', 'IG', 'ID', 'IE', 'IZ'} ' ' ;
TypoMark       =  '+' {'31', '32', '33'} ' ' ;
Trailer        =  {'+P ', '+16 ', '+63 ', '+22 ', '='}* ;
PhonetChar     =  {Letter, Punctuation, '"', '(', ')'}
               |  '+' {'1', '2', '3', '4', '6', '7', '11', '12', '13',
                       '17', '18', '19', '20', '21', '30', '32', '34',
                       '44', '45', '50', '52', '97', 'DP'} ' ' ;
LabelText      =  (Letter | Accent | Digit | Punctuation | QuoteMark)
               |  GrabBag - {'=', '+22 '}
               |  ('+' {'11', '14', '63', '97'} ' ')
               |  ('+SU +13 ' {0,1,2} '+RB ') ;
GarbText       =  {Accent, TypoMark, Index, PhonetChar, '*', '(', ')' '+14 '?}
               |  '+' {'15', '16', '29', '63'} ' '
               |  (,' <vd>') Digit '+14 '? (,' </vd>') ;
HWChar         =  {Letter, Accent, '(', ')', '-', '"', ' ', '.'}
               |  '+' {'11', '12', '17'} ' ' ;
StatusMark     =  ((,' status=')
                   (('+15 ','spu') | ('+31 ','obs') | ('+32 ','ali') | ('+33 ','err')) )+ ;
TextChars      =  {Digit, Letter, Punctuation, QuoteMark, Accent,
                   GrabBag, Foreign, Special} ;
Alph           =  TextChars: alph ;
SenseText      =  TextChars | TypoMark | {'+15 ', '+16 '} ;

SNUC           =  {A, B, C, D, E, F, G, H, I, J, K, L, M} ;
SNLC           =  {'a' 'a'?, 'b' 'b'?, 'c' 'c'?, 'd' 'd'?, 'e' 'e'?, 'f' 'f'?,
                   'g' 'g'?, 'h' 'h'?, 'i' 'i'?, 'j' 'j'?, 'k' 'k'?, 'l' 'l'?,
                   'm' 'm'?, 'n' 'n'?, 'o' 'o'?, 'p' 'p'?, 'q' 'q'?, 'r' 'r'?,
                   's', 't', 'u', 'v', 'w', 'x', 'y', 'z'} ;
SNromanUC      =  {'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX'} ;
SNromanLC      =  {'iii', 'iv', 'v'} ;
PMarker        =  1.NOECHO '+PP ' 1.ECHO 2.Witalic ;
LMarker        =  1.NOECHO '+LL ' 1.ECHO 2.Witalic ;
```

```
Parts          =  (
                  {'a', 'adj' 's'?, 'adv' {'s', 'b'}?, 'attrib',
                  'conj', 'dem', 'impers', 'indef', 'in comb',
                  'int' 'r'?, 'numeral', 'pa. pple', 'pa. t',
                  'pers', 'phr', 'pl', 'pred', 'pref',
                  'prep', 'poss', 'ppl' 'e'?, 'pr',
                  'pron', 'sb' 's'?, 'sing', 'suff',
                  'trans', 'v', 'vbl', 'vbs'} '.'
               |  {'stem', 'plural', 'prefix', 'suffix',
                  'particle', 'phrase'} '.'? ) ;
Label          =  (
                  {'Aeronaut', 'Agric', 'Anat', 'Antiq', 'Arch', 'Astr', 'Astrol',
                  'Austr', 'Austral', 'Anglo-'? {'Ir', 'Ind'},
                  'absol', 'appos', 'arch', 'attrib', 'Biol',
                  'Bot', 'Build', 'Canad', 'Chem', 'Comb', 'Comm',
                  'Conch', 'colloq', 'collect', 'Const', 'Cryst', 'dial',
                  'Eccl', 'ellipt', 'Engin', 'Ent', 'erron', 'euphem', 'fig',
                  'Geol', 'Geom', 'Gram', 'Her' 'b'?, 'Hort', {H,h} 'ist',
                  'Ichth', 'Math', 'Mech', 'Med', 'Metaph', 'Mil', 'Min',
                  'Mus', 'N. Amer', 'N.Z', 'Nat. Hist', 'Naut',
                  {N,n} 'once-wd' 's'?, 'Numism', 'Ornith',
                  {O,o} 'bs', 'Opt', 'pass', 'Path', 'pers' '. sing'?,
                  'Petrol', 'Philol', 'Philos', 'Phren', 'Phys' 'iol'?,
                  'poet', 'Pros', 'Psych' 'ol'?, 'refl', 'Rhet',
                  'S. Afr', 'Sc', 'Surg', 'Theol', 'transf', 'Trig',
                  'Typog', 'U.S' '.A'?, 'vulg', 'Zool'} '.'
               |  {'Anthropology', 'Australian', 'Bookbinding', 'Electronics',
                  'Genetics', 'Irish', 'jocular', 'Linguistics',
                  'Metallurgy', {N,n} 'once-use' 's'?, 'Parasitology',
                  'Physic' {'s', 'al'}, 'rare', 'slang', 'Statistics',
                  'Taxonomy', 'West Indian'} '.'?
               )  ','? ' '? ;
SectionLabel   =  {'intr', 'pl' 'ur'?, 'Proverb' {'s', 'ial'}?,
                  'phr' 'ase'? 's'?, 'sing', 'trans', 'ironical',
                  'with', 'clause', 'erroneous', 'use',
                  'gerund', 'causatively', 'Substantive', 'impers',
                  'allusive' 'ly'?, 'appositively', 'Billiards',
                  'Confused', 'humorously', 'pedantic',
                  'indirect', 'passive', {P,p} 'ersonified',
                  'punningly', 'quasi-' {'adj', 'adv'},
                  'Tanning'} '.'? ' '?
               |  Label ;
```

```
Number          =   Digit+ ;
Date            =   Number
                    (   '+' {'10', '14', '16', '107', 'AI', 'CI'} ' '
                    |   {Number, '.'} ' '?
                    |   (' </qdat>') ',' ' '? (' <qdat>')
                    )* ;
SenseNumber     =       SNUC | SNromanUC | Number | SNLC | '(' {SNromanLC, SNLC} ')' ;
EndSense        =   {'(', ')', ',', ':', ';', '=', '+22 ', '+15 ', '+16 ', '+17 ', '+63 ', '+P '} ;
SenseTail       =   {TextChars, Index} - EndSense ;
SenseGram       =   ( {SNUC, SNromanUC, Number} {' ', '.' ' '?, '+64 '}
                    | SNLC '. ' )
                    SenseTail* ;
PartGram        =   PMarker (' <pos>') Parts (','? ' '? Parts)* ','?
                    (1.NOECHO ' ' 1.ECHO)? (' </pos>') ;
HeadwordLemma   =   {Letter, '-', '"} (HWChar |
                    (' </hwlem>') ({', ', TypoMark}+, ' <hwlem>') HWChar)* ;
XRLemma         =   ASmcaps {'? ', '+17 ', '(', ""}? (,Nl ' <xlem>') {Letter, '-'}
                    ( HWChar | HWChar '+17 ' HWChar | (' </xlem>')
                    ({'=', '+22 ', ', ', '; '},' <xlem>') HWChar )* (' </xlem>')
                    ( (' <sn>') Digit {'.', ',', ' '}* (' </sn>')
                    | (' <R>') ({':', ';', ',', '*', '!', '='} ' '?
                    | '+' {'15', '18', '16' ' .'?, '22', '63', '64'} ' ')+ (' </R>')
                    )? ;
HomonymNumber   =   1.NOECHO ('+HM,' ' <hom>') 1.ECHO Number
                    1.NOECHO ' ' 1.ECHO 2.Wroman (' </hom>') ;
HomographMarker =   (' <pos>') (Italic | PMarker | LMarker) Parts
                    (1.NOECHO ' ' 1.ECHO)? (' </pos>') (HomonymNumber 2.Wroman)? ;
Pronunciation   =   1.NOECHO ('+PR ' <pron>') 1.ECHO 2.Wroman
                    ((' <R>') PhonetChar+ (' </R>'))?
                    (   ARoman (' <R>') PhonetChar+ (' </R>')
                    |   AItalic (' <I>') PhonetChar+ (' </I>')
                    |   LMarker (' <lab>') LabelText+ (' </lab>')
                    )*  (' </pron>') ;
PartsOfSpeech   =   1.NOECHO '+PS ' 1.ECHO 2.Witalic
                    (ARoman? ((' <R>') {'?', '('} ' '? (' </R>'))?
                    (Roman (' <R>') ({'or', 'orig.', 'and', 'quasi-', 'prop.', 'rarely',
                    'usu. '? 'const. as', 'passing into', 'formerly'} ')'? ' '?)+
                    (' </R>') )? HomographMarker (ARoman?
                    (' <R>') {')', ',', ':', ';'} '.'? ' '? (' </R>'))? )+ ;
```

```
Headwords       =   (,' <hwgp>') (,Nl ' <hwlem>') HeadwordLemma (,' </hwlem>')
                    (HomonymNumber '. '?)? {', ', ': '}?
                    (ARoman (,' <R>') 'to ' (,' </R>'))?
                (   PartsOfSpeech
                |   Pronunciation '+63 '?
                |   1.NOECHO TypoMark? ('+HW ',Nl ' <hwlem>') 1.ECHO 2.Wbold
                    HeadwordLemma (,' </hwlem>') (HomonymNumber '. '?)? {', ', ': '}?
                |   ARoman (,' <R>') '('? {'etc.', 'and', 'usually'} ')'?
                    {',', ':'}? ' '? (,' </R>')
                |   ABold (,' <B>') {'.', ',', ';', ':'}? ' '? (,' </B>')
                    (TypoMark? (,Nl ' <hwlem>') HeadwordLemma (,' </hwlem>')
                    (HomonymNumber '. '?)? (ARoman ')')? {', ', ': '}? )?
                )*  (,' </hwgp>') ;
Labels          =   1.NOECHO ('+LA ',Nl ' <labs>') 1.ECHO 2.Witalic
                    ((,' <lab>') LabelText+ (,' </lab>'))?
                (   ARoman (,' <R>') LabelText+ (,' </R>')
                |   ABold (,' <vf>') LabelText+ (,' </vf>')
                |   (AItalic | LMarker | PMarker) (,' <lab>') LabelText+ (,' </lab>')
                )*  (,' </labs>') ;
VarText         =   {Letter, Accent, '*', '.', ' ', '''', '(', ')', '-', '&'}
                |   ('+' {'11', '13', '15', '16', '17', '23', '30'} ' ') ;
VariantForm     =   ABold (,Nl ' <vf>') (VarText | (,' </vf>')
                    ({'; ', ', ', Index, TypoMark}+, ' <vf>') VarText)+
                    (,' </vf>') ({Index, TypoMark} | {',', ':', ';', '+63 '} ' '?)* ;
Variants        =
                (   (AItalic | LMarker) (,' <lab>') LabelText+ (,' </lab>') Index?
                |   PartGram Trailer
                |   ARoman ( (,' <sn>') SenseGram (,' </sn>')
                    ((,' <R>') EndSense GarbText* (,' </R>'))?
                    || (,' <R>') GarbText+ (,' </R>') )
                |   HomonymNumber ( (,' <sn>') SenseGram (,' </sn>')
                    ((,' <R>') EndSense GarbText* (,' </R>'))?
                    || (,' <R>') GarbText+ (,' </R>') )?
                |   1.NOECHO (('+OB ',Nl ' <et>') | ('+EB ',' </et>'))
                    1.ECHO GarbText*
                |   XRLemma
                |   VariantForm
                )*  ;
VariantForms    =   1.RESTART
                    1.NOECHO ('+VL ',Nl ' <vfl>') 1.ECHO 2.Wroman
                    ((,' <R>') GarbText+ (,' </R>'))? Variants (,' </vfl>') ;
```

```
EtymText        =   TextChars
                |   ('+' {'12', '19', '20', '30', '31', '44', '45', 'DP'} ' ') ;
Etymology       =   1.RESTART
                    1.NOECHO ('+ET +OB ',Nl ' <etym>') 1.ECHO 2.Wroman
                    ((,Nl ' <R>') SenseText+ (,' </R>'))?
                (   (AItalic | LMarker) (,' <I>') SenseText+ (,' </I>')
                |   PartGram {'+P ', '+63 ', '+22 ', '='}*
                |   ABold (,Nl ' <B>') SenseText+ (,' </B>')
                |   ARoman ( (,' <sn>') SenseGram (,' </sn>')
                    ((,' <R>') EndSense SenseText* (,' </R>'))?
                    || (,' <R>') SenseText+ (,' </R>') )
                |   HomonymNumber ( (,' <sn>') SenseGram (,' </sn>')
                    ((,' <R>') EndSense SenseText* (,' </R>'))?
                    || (,' <R>') SenseText+ (,' </R>') )?
                |   ( 1.NOECHO ('+EN ',Nl ' <enote>') 1.ECHO
                    | 1.NOECHO ('+M ',Nl ' </enote>') 1.ECHO
                    ) ((,' <R>') SenseText+ (,' </R>'))?
                |   XRLemma
                )*  2.Wroman 1.NOECHO ('+EB ', ' </etym>')
                    1.ECHO {Punctuation, '=', '+22 '}* ;

LemmaText       =   {HWChar, TypoMark, Punctuation, Digit, GrabBag}
                |   ('+' {'10', '13', '14', '18', '63', '64', '76', '107', '108'} ' ') ;
Lemma           =   ( 1.NOECHO ('+IL ',' <lem face=I>') 1.ECHO 2.Witalic
                    | 1.NOECHO ('+BL ',' <lem face=B>') 1.ECHO 2.Wbold ) LemmaText*
                (   ABold (,' <B>') LemmaText+ (,' </B>')
                |   ARoman ( (,' <sn>') SenseGram (,' </sn>')
                    ((,' <R>') EndSense SenseText* (,' </R>'))?
                    || (,' <R>') SenseText+ (,' </R>') )
                |   HomonymNumber ( (,' <sn>') SenseGram (,' </sn>')
                    ((,' <R>') EndSense SenseText* (,' </R>'))?
                    || (,' <R>') SenseText+ (,' </R>') )?
                |   AItalic (,' <I>') LemmaText+ (,' </I>')
                |   PartGram Trailer
                |   LMarker (,' <lab>') LemmaText+ (,' </lab>')
                |   XRLemma
                )*  1.NOECHO ('+EL ',' </lem>') 1.ECHO 2.Wroman
                    ( (,' <pos><R>') Parts (,'? ' '? Parts)* (,' </R></pos>')
                    (1.NOECHO ' ' 1.ECHO)?
                    ((,' <R>') (Punctuation - ' ') SenseText* (,' </R>'))?
                    || (,' <R>') SenseText+ (,' </R>') )? ;
```

```
SenseStart      =   1.NOECHO '+RR ' 1.ECHO 2.Wroman
                    ((,Nl ' <R>') SenseText+ (,' </R>'))?
                |   1.NOECHO '+IR ' 1.ECHO 2.Witalic
                    ((,Nl ' <lab>') LabelText+ (,' </lab>')
                    {'+15 ', '=', '+22 '}? )? ;
SenseBody       =
                (   1.NOECHO (('+ET '? '+OB ',Nl ' <et>') | ('+EB ',' </et>')) 1.ECHO
                    ( (2.Rroman (,' <R>') SenseText+ (,' </R>'))
                    | ((2.Rsmcaps | 2.Rbold) (,' <R>')
                    {Punctuation, '=', '+22 ', '+63 '}+ (,' </R>'))
                    | (2.Ritalic (,' <lab>') SenseText+ (,' </lab>')) )?
                |   ABold (,Nl ' <B>') LemmaText+ (,' </B>') '+15 '?
                |   ARoman ( (,' <sn>') SenseGram (,' </sn>')
                    ((,' <R>') EndSense SenseText* (,' </R>'))?
                    || (,' <R>') SenseText+ (,' </R>') )
                |   HomonymNumber ( (,' <sn>') SenseGram (,' </sn>')
                    ((,' <R>') EndSense SenseText* (,' </R>'))?
                    || (,' <R>') SenseText+ (,' </R>') )?
                |   ( 1.NOECHO ('+NT ', Nl ' <snote>') 1.ECHO
                    | 1.NOECHO ('+M ', ' </snote>') 1.ECHO
                    ) ((,' <R>') SenseText+ (,' </R>'))?
                |   AItalic (,Nl ' <I>') SenseText+ (,' </I>')
                |   LMarker (,' <lab>') SenseText+ (,' </lab>')
                |   PartGram '+63 '?
                    ((,Nl ' <lab>') (Label ','? ' '?)+ (,' </lab>'))?
                    {'+P ', TypoMark, ')' '.'? ' '?, '; ',
                    '+15 ', '+16 ', '+22 ', '=', '+63 '}*
                |   XRLemma
                |   Lemma
                )+  ;
SenseSection    =
                (   1.RESTART
                (   1.NOECHO {('+SS ',Nl ' <sen para=t'), ('+SN ',Nl ' <sen')}
                    2.Wbold StatusMark? 1.ECHO (,' lit=' '")
                    (Bold SenseNumber '.' | Bold Index | (ARoman | Bold) ('*')+ |
                    ARoman '(' (AItalic LCase | {Digit,i,v,x}+ | LCase) Roman ')' )
                    (,'" '>') ';'? ' '? SenseStart
                |   1.NOECHO {('+SR ',Nl ' <sen'), ('+SP ',Nl ' <sen para=t')}
                    2.Wbold StatusMark? 1.ECHO (,'>') 2.Wroman ((,Nl ' <R>')
                    (SenseText - {TypoMark, '+15 '}) SenseText* (,' </R>'))?
                )   SenseBody? (,' </sen>')
                )+  ;
```

```
AuthorChars       =   {Letter, QuoteMark, Accent, Punctuation, GrabBag}
                  |   ('+' {'10', '13', '15', '16'} ' ') ;
QuotationLabel    =   (,Nl ' <qlab>') ((,' <R>')
                      ARoman? {'?' ' '?, TypoMark, '+36 ', '+15 '} (,' </R>'))?
                  (   (PMarker | LMarker | AItalic) (,' <I>') SectionLabel+ (,' </I>')
                      (ARoman (,' <R>') {Digit, Letter, '(', ')', ' ', '.'}+
                      (,' </R>') | (PMarker | LMarker | AItalic)
                      (,' <I>') SectionLabel+ (,' </I>') )*
                  |   (,' <R>') (SNLC '. ' | Index) (,' </R>') ((PMarker
                      | LMarker | AItalic) (,' <I>') SectionLabel+ (,' </I>'))?
                  |   ARoman (,' <R>') {{C,c} 'onst. ', 'quasi-'} (,' </R>')
                      (,' <I>') (AItalic | PMarker) LemmaText+ (,' </I>')
                  |   ARoman '(' AItalic (,' <I>') SNLC (,' </I>') ARoman ') '
                  )   (,' </qlab>') '+63 '? ;

QuotationDate     =   (,Nl ' <qdat>') ARoman? ('?' ' '?)?
                      (ABold? {'+AI ', '+CI '} 2.Wbold)? Bold Date (,' </qdat>') ;

QuotationAuthor   =   ASmcaps ((,' <R>') {'? ', '&', '('} (,' </R>'))?
                      (,Nl ' <auth>') (UCase | '+' {'15', '17', '923'} ' ')
                      AuthorChars+ (,' </auth>')
                      ((,' <R>') Digit {Digit, '.', ' '}* (,' </R>'))? ;

QuotationNote     =
                  (   (AItalic | LMarker) (,' <I>') TextChars+ (,' </I>')
                  |   PartGram {'+63 ', '+', '='}*
                  |   ARoman ( (,' <sn>') SenseGram (,' </sn>')
                      ((,' <R>') EndSense TextChars* (,' </R>'))?
                      || (,' <R>') TextChars+ (,' </R>') )
                  |   HomonymNumber ( (,' <sn>') SenseGram (,' </sn>')
                      ((,' <R>') EndSense TextChars* (,' </R>'))?
                      || (,' <R>') TextChars+ (,' </R>') )?
                  |   ABold (,' <qdat>') Date (,' </qdat>')
                  |   XRLemma
                  )*  ;
```

```
QuotationSource   =
                  (   QuotationAuthor
                  |   (AItalic | LMarker) (,' <I>') {TextChars, '+16 '}+ (,' </I>')
                  |   ABold (,' <qdat>') Date (,' </qdat>')
                  |   ARoman (,' <R>') ((Alph* {': see ', ': cf. ', '+I ', '+QP',
                      '+QN', '+QT', '+B ', '+SC', '+R ', '+15', '+PP', '+LL',
                      '+SS', '+SN', '+SR', '+SP', '+1000', '+1006'}
                      Alph* | ''): acomp) (,' </R>')
                  |   ASmcaps '('? (,' <pna>') {a, b, i, v, x, l, c, '.', ',', ' ',
                      '+15 ', '+16 ', '+102 '}+ (,' </pna>') ((,' <R>')
                      {Digit, '(', ')'} {Digit, '.', ' ', ',', '/'}* (,' </R>'))?
                  |   (,Nl ' <qnot>') ( (2.Rroman | 2.Rbold | 2.Ritalic)
                      '+15 ' | Roman {': see ', ': cf. '} ) TextChars*
                      QuotationNote ('+16 ' TextChars*) (,' </qnot>')
                  )   ;

QuotationText     =   1.NOECHO ('+QT ',Nl ' <qtxt>') 1.ECHO 2.Wroman
                      ((,' <R>') SenseText+ (,' </R>'))?
                  (   ARoman ( (,' <sn>') SenseGram (,' </sn>')
                      ((,' <R>') EndSense SenseText* (,' </R>'))?
                      || (,' <R>') SenseText+ (,' </R>') )
                  |   HomonymNumber ( (,' <sn>') SenseGram (,' </sn>')
                      ((,' <R>') EndSense SenseText* (,' </R>'))?
                      || (,' <R>') SenseText+ (,' </R>') )?
                  |   ABold (,' <B>') SenseText+ (,' </B>')
                  |   (AItalic | LMarker)
                      (,' <I>') SenseText+ (,' </I>')
                  |   PartGram Trailer
                  |   XRLemma
                  |   ASmcaps (,' <SC>') HWChar* {'+13 ', '+204 '} HWChar* (,' </SC>')
                  )*  (,' </qtxt>') ;
```

```
QuotationHead     =
                  (     QuotationDate ((,Nl ' <srce>') QuotationSource+ (,' </srce>'))?
                  )     | (,Nl ' <srce>')
                  (     (ARoman (,' <R>') '('? {'So', 'In ', 'For', 'And'}
                        {LemmaText, '+15 ', '+16 '}+ (,' </R>')
                  ) | (
                        ( ARoman (,' <R>') {'+108 ', '+13 +13 ', 'Whence ', 'Not',
                        'Also', 'See'} LemmaText* (,' </R>')
                        | AItalic (,' <I>') {'Ibid', 'Beowulf', 'Mod', 'Prov',
                        'O' '.'? ' '? 'E.'} LemmaText* (,' </I>')
                        | QuotationAuthor
                        ) QuotationSource*
                  ) |   (,' <R>') ARoman 'Cf.' LemmaText* (,' </R>')
                        (,Nl ' <qnot>') QuotationNote (,' </qnot>') '+16 '?
                  )     (,' </srce>') ;
QuotationBody     =   '+15 '? QuotationHead QuotationText? ;
QuotationBank     =   (1.RESTART (,' <qbank>') 1.NOECHO '+QP '
                      (1.ECHO QuotationLabel 1.NOECHO '+QN '?)?
                      (,Nl ' <quot>') 1.ECHO 2.Wbold QuotationBody (,' </quot>')
                      (1.RESTART 1.NOECHO ('+QN ',Nl ' <quot>') 1.ECHO
                      2.Wbold QuotationBody (,' </quot>'))*
                      (,' </qbank>') )* ;

HeadwordSection   =   (,Nl ' <hwsec>') Headwords Labels? VariantForms?
                      Etymology? (,' </hwsec>') ;
Signification     =   (,Nl ' <signif>') (SenseSection QuotationBank)+ (,' </signif>') ;
MainEntry         =   1.RESTART
                      (,Nl ' <entry') 1.NOECHO ('+1000 ' | ('+1006 ',' xref=t'))
                      Digit+ ' 1 ' 2.Wbold ((,' status=')
                      (('+15 ','spu') 2.Wroman | ('+31 ','obs') | ('+32 ','ali')))?
                      (,' >') 1.ECHO HeadwordSection Signification (,Nl ' </entry>' Nl) ;
Dictionary        =   1.ECHO (MainEntry+ @( (Wbold Rbold*) | (Witalic Ritalic*)
                      | (Wroman Rroman*) | (Wsmcaps Rsmcaps*) )*) ;
Dictionary: save P1Dict ;
```

B.2 The Pass 2 Dictionary Grammar

```
Nl              =   '
' ;
Digit           =
                    {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ;
LCase           =
                    {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t,
                    u, v, w, x, y, z} ;
UCase           =
                    {A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T,
                    U, V, W, X, Y, Z} ;
Letter          =
                    LCase | UCase
                |   '+' {'H', 'UP', 'SU', 'RB', '35'} ' '
                |   '+' '9'? {'23', '24', '25', '26',
                    '27', '39', '40', '47', '48'} ' ' ;
DownCase        =
                    1.NOECHO (,' down>')
                    {(A,a), (B,b), (C,c), (D,d), (E,e), (F,f), (G,g),
                    (H,h), (I,i), (J,j), (K,k), (L,l), (M,m), (N,n),
                    (O,o), (P,p), (Q,q), (R,r), (S,s), (T,t), (U,u),
                    (V,v), (W,w), (X,x), (Y,y), (Z,z)} 1.ECHO
                |   (,' >') {LCase, '-', '(', '+923 ', '+23 '} ;
QuoteMark       =
                    '"' | '+' {'17', '18', '37', '38'} ' ' ;
Accent          =
                    '+' {'1', '2', '3', '4', '5', '6', '7', '8', '9',
                    '5' Digit, '60', 'OV', 'EO'} ' ' ;
Foreign         =
                    '+' {'FS', '43', '46', '49', '48', '948', '75'} ' '
                |   ('+' '9'? {'76','77','78','79',{8,9} Digit}) ' ' ;
GrabBag         =
                    {'+22 ', '%', '&', '=', '/', '(', ')', '*', '$'} ;
Punctuation     =
                    {'.', ':', ';', ' ', '?', '!', ',', '-'};
```

```
Special     =
              ('+' {'AI', 'AMP', 'BF', 'CE', 'CI', 'CO', 'CU', 'DE',
              'DN', 'ED', 'EF', 'FB', 'FE', 'G', 'GB', 'GI',
              'IN', 'M', 'NL', 'NU', 'P', 'RB', 'S', 'TE',
              '10', '11', '12', '13', '14', '19', '20', '21',
              '28', '30', '32', '34', '36', '44',
              '45', '50', '52', '63', '64', '65',
              '110', '111', '112', '241', '242', '243', '244',
              '246', '274', '275', '277', '278',
              '342', '348', '349', '353'} ' ')
            | ('+' {'10', '20', '21', '22', '23', '30', '31', '32',
              '33', 'T'} Digit ' ')
            | ('+C' Digit ',' {'qr', 'ce'} ' ') ;
Index       =
              '+' {'IA', 'IB', 'IG', 'ID', 'IE', 'IZ'} ' ' ;

TextChars   =
              {Digit, Letter, Punctuation, QuoteMark, Accent, GrabBag,
              Foreign, Special} ;
Alph        =
              TextChars: alph ;
RText       =
              ' <R>' Alph+ ' </R>' ;
IText       =
              ' <I>' Alph+ ' </I>' ;
ScText      =
              ' <SC>' Alph+ ' </SC>' ;
BText       =
              ' <B>' Alph+ ' </B>' ;
Label       =
              ' <lab>' Alph+ ' </lab>' ;
SNUC        =
              {A, B, C, D, E, F, G, H, I, J, K, L, M} ;
SNLC        =
              {'a' 'a'?, 'b' 'b'?, 'c' 'c'?, 'd' 'd'?, 'e' 'e'?, 'f' 'f'?,
              'g' 'g'?, 'h' 'h'?, 'i' 'i'?, 'j' 'j'?, 'k' 'k'?, 'l' 'l'?,
              'm' 'm'?, 'n' 'n'?, 'o' 'o'?, 'p' 'p'?, 'q' 'q'?, 'r' 'r'?,
              's', 't', 'u', 'v', 'w', 'x', 'y', 'z'} ;
SNromanUC   =
              {'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX'} ;
```

```
Linkage         =
                    {Punctuation, '(', ')', '=', '+22 ', '+P ', '+63 ',
                    '+15 ', '+16 '} ;
EndSense        =
                    {'(', ')', ',', ':', ';', '=', '+22 ', '+15 ', '+16 ', '+17 ',
                    '+63 ', '+P '} ;
SenseTail       =
                    {TextChars, Index} - EndSense ;
ResWord         =
                    {'quot' 's'? '.', 'next', 'prec.', 'sense' 's'? ' '} ;
Commands        =
                    ComWord Alph* ;
Number          =
                    Digit+ ;
SenseGram       =
                    (     {SNUC, SNromanUC} {' ', '.' ' '?, '+64 '} | Number
                    |     SNLC ) SenseTail* ;
HomonymNumber   =
                    ' <hom>' Number ' </hom>' ;
HomogMarker1    =
                    (' <pos>' Alph+ ' </pos>' Alph* | ' <pos>' Alph+
                    1.NOECHO ' </pos><pos>' 1.ECHO (',' ') Alph+ ' </pos>' Alph*)
                    HomonymNumber? ;
HomogMarker2    =
                    ' <pos>' Alph+ ' </pos>'
                    (' <R>'? Punctuation* ' </R>'? HomonymNumber)? ;

ConvertSN       =
                    1.NOECHO (' <sn>',' <R>') 1.ECHO Alph+
                    1.NOECHO (' </sn>',' </R>') 1.ECHO ;
ConvertNumber   =
                    1.NOECHO (' <sn>',' <R>') 1.ECHO (Alph - Digit) Alph*
                    1.NOECHO (' </sn>',' </R>') 1.ECHO
                |   1.NOECHO (' <sn>',' <vd>') 1.ECHO Digit ' '?
                    1.NOECHO (' </sn>',' </vd>') 1.ECHO ;
ConvertPOS      =
                    1.NOECHO (' <pos>',' <I>') 1.ECHO Alph+
                    1.NOECHO (' </pos>',' </I>') 1.ECHO ;
```

```
XRA             =
                    (,Nl ' <xra>') ' <xlem' 1.NOECHO ' >' 1.ECHO
                    DownCase Alph* ' </xlem>'
                (   ' <pos>' Alph+ ' </pos>'
                |   ' <sn>' Alph+ ' </sn>'
                |   ' <R>and </R>'? HomonymNumber
                )*  (,' </xra>') ;
Pronunciation   =
                    (,Nl) ' <pron>'
                (   RText
                |   IText
                |   Label
                )*  ' </pron>' '+63 '? ;

Headwords       =
                    (,Nl) ' <hwgp>'
                (   ' <hwlem' 1.NOECHO ' >' 1.ECHO DownCase Alph* ' </hwlem>'
                    Punctuation* ((HomogMarker1 | HomonymNumber) Punctuation*)?
                |   Pronunciation HomogMarker1?
                |   RText HomogMarker1?
                |   BText
                )*  ' </hwgp>' ;
VariantForm     =   ' <vf>' Alph+ ' </vf>' ;
Labels          =
                    (,Nl) ' <labs>'
                (   RText
                |   VariantForm
                |   Label
                )*  ' </labs>' ;
Variants        =
                (   VariantForm Alph* ConvertNumber?
                |   ' <R>' ( ' <vd>' Alph+ 1.NOECHO ' </vd><vd>' 1.ECHO Alph+ ' </v
                    | Alph | ' <vd>' Alph+ ' </vd>' Alph )*
                    (' <vd>' Alph+ ' </vd>')? ' </R>'
                    ConvertPOS? ConvertNumber?
                |   (' <et>' | ' </et>') Alph* ConvertNumber?
                |   XRA
                |   Label Alph* ConvertNumber?
                |   {Linkage, Index, TypoMark}
                )*  ;
VariantForms    =
                    1.RESTART 1.ECHO (,Nl) ' <vfl>' Variants ' </vfl>' ;
```

```
Etymology       =
                    1.RESTART 1.ECHO (,Nl) ' <etym>'
                (   IText ConvertSN?
                |   BText ConvertSN?
                |   (' </enote>' | ' <enote>') Alph* ConvertSN?
                |   XRA
                |   ' <R>' ((Alph* ResWord Alph*): acomp)
                    ((,' <rxra>') ResWord '.'? ' '? ((,' <sn>') SenseGram (,' </sn>'))?
                    (,' </rxra>') (EndSense Alph*)? )? ' </R>'
                    HomonymNumber? HomogMarker2? ConvertSN?
                |   Linkage
                )*  ' </etym>' Alph* ;

Lemma           =
                    ' <lem' {' face=I>', ' face=B>'} Alph*
                (   BText ConvertSN?
                |   RText (' <pos>' Alph+ ' </pos>')? ConvertSN?
                |   IText ConvertSN?
                |   Label ConvertSN?
                |   XRA
                )*  ' </lem>' ;
SenseBody       =
                (   (' <et>' | ' </et>') Alph* ConvertSN?
                |   (' <snote>' | ' </snote>') Alph* ConvertSN?
                |   BText ConvertSN?
                |   IText ConvertSN?
                |   Label ConvertSN?
                |   ' <R>' ((Alph* ResWord Alph*): acomp)
                    ((,' <rxra>') ResWord '.'? ' '? ((,' <sn>') SenseGram (,' </sn>'))?
                    (,' </rxra>') (EndSense Alph*)? )?
                    ' </R>' HomogMarker2? '+63 '? ConvertSN?
                |   Linkage
                |   XRA
                |   Lemma HomogMarker2? ConvertSN?
                |   ' <pos><R>' Alph+ ' </R></pos>'
                )+  ;
SenseSection    =
                (   1.RESTART 1.ECHO (,Nl) ' <sen' Alph* ' >' Alph*
                    (' <pos>' Alph+ ' </pos>')? SenseBody? ' </sen>'
                )+  ;
```

```
QuotationLabel    =
                      ' <qlab>'
                  (   IText
                  |   RText
                  |   '(' ' <I>' SNLC ' </I>' ') '
                  )+  ' </qlab>' '+63 '? ;
QuotationDate     =
                      (' <qdat>' Alph+ ' </qdat>' (',' ' '?)?)+
                      (1.NOECHO ' <qdat></qdat>' 1.ECHO)? ;
QuotationAuthor   =
                      ' <auth>' Alph+ ' </auth>' ;
QuotationNote     =
                  (   IText ConvertSN?
                  |   RText ConvertPOS? ConvertSN?
                  |   QuotationDate ConvertSN?
                  |   XRA
                  |   Linkage
                  )+  ;
QuotationSource   =
                      ' <srce>'
                  (   QuotationAuthor
                  |   IText
                  |   RText
                  |   ' <qdat>' Alph+ ' </qdat>'
                  |   ' <pna>' Alph+ ' </pna>'
                  |   ' <qnot>' Alph* ConvertSN? QuotationNote? Alph* ' </qnot>'
                  |   Linkage
                  )+  ' </srce>' ;
QuotationText     =
                      ' <qtxt>'
                  (   IText ConvertSN?
                  |   BText ConvertSN?
                  |   ScText ConvertSN?
                  |   RText ConvertPOS? ConvertSN?
                  |   XRA
                  |   {Punctuation, '(', ')', '+P ', '+63 ', '+15 ', '+16 ', '+22 ', '='}
                  )+ ' </qtxt>' ;
QuotationHead     =
                      (QuotationDate QuotationSource? | QuotationSource ) ;
```

```
QuotationBody     =
                      '+15 '? QuotationHead QuotationText? ;
QuotationBank     =
                  ( 1.RESTART 1.ECHO (,Nl) ' <qbank>' QuotationLabel?
                      (' <quot>' QuotationBody ' </quot>')+
                      ' </qbank>'
                  )* ;
HeadwordSection   =   (,Nl)
                      ' <hwsec>' Headwords Labels? VariantForms? Etymology? ' </hwsec>' ;
Signification     =   (,Nl)
                      ' <signif>' (SenseSection QuotationBank)+ ' </signif>' ;
MainEntry         =   1.RESTART 1.ECHO (,Nl Nl)
                      ' <entry' Alph* ' >' HeadwordSection Signification '+16 '?
                      ' </entry>' ;
Dictionary        =
                      1.ECHO MainEntry+ ;
Dictionary: save P2Dict ;
```

Appendix C

Sample OED Proofs

C.1 A Content Proof

man-ha·ndle, *v.* [f. MAN *sb.*[1] + HANDLE *v.*; in sense 3 cf. dial. ***manangle*** (Devon) to mangle, which may belong to MANGLE *v.* (AF. ***mahangler***).]

† 1. *trans.* To handle or wield a tool. *Obs.*

1457 R. FANNANDE *Mon. Christ's Hosp. Abingdon* xiii, The Mattok was man-handeled right wele a whyle.

2. *Naut.*, etc. 'To move by force of men, without levers or tackles' (Adm. Smyth).

1867 SMYTH *Sailor's Word-bk.* **1894** *Times* 27 Jan. 10/2 The larger weapons will be worked by electricity, but are also capable of being man-handled. **1902** *Blackw. Mag.* Mar. 331/2 I'm going to man-handle my gun down the slope. **1903** *Daily Chron.* 19 Feb. 3/3 Stalwart Punjabis . . hand out bags of stores, . . or manhandle a fractious, restive animal.

3. *slang.* To handle roughly; to pull or hustle about.

1865 *Hotten's Slang Dict., Man-handle,* to use a person roughly, as to take him prisoner, turn him out of a room, give him a beating. **1886** *Century Mag.* Apr. 905/1 Two of our roughs began to haze him: but they mistook their calling, and in two minutes were so mauled and manhandled that it was reported aft. **1888** CLARK RUSSELL *Death Ship* II. 253, I . . was for . . manhandling him, ghost or no ghost. **1891** KIPLING *Light that failed* iii, I'll catch you and man-handle you, and you'll die. **1894** R. H. DAVIS *Eng. Cousins* 24 The cry of 'Welsher', . . which sometimes on an English race-course means death from man-handling.

C.2 A Structure Proof

```
<entry>
  <hwsec>
    <hwgp><hwlem>man-ha·ndle</hwlem>, <pos>v.</pos></hwgp>
    <etym><R>f. </R>
      <xra><xlem>man </xlem><pos>sb.</pos><hom>1</hom></xra>
      <R>+</R>
      <xra><xlem>handle </xlem><pos>v.</pos></xra>
      <R>; in <rxra>sense <sn>3</sn></rxra> cf. dial. </R><I>manangle </I>
      <R>(Devon) to mangle, which may belong to </R>
      <xra><xlem>mangle </xlem><pos>v.</pos></xra><R>(AF. </R>
      <I>mahangler</I><R>).</R>
    </etym>
  </hwsec>

  <signif>
    <sen para=t status=obs lit='1.'> <lab>trans. </lab>
      <R>To handle or wield a tool. </R><lab>Obs. </lab>
    </sen>

    <qbank>
      <quot>
        <qdat>1457 </qdat>
        <srce><auth>R. Fannande </auth><I>Mon. Christ's Hosp. Abingdon </I>
          <R>xiii, </R></srce>
        <qtxt><R>The Mattok was man-handeled right wele a whyle. </R></qtxt>
      </quot>
    </qbank>
```

```
<sen para=t lit='2.'> <lab>Naut., </lab>
   <R>etc. 'To move by force of men, without levers or tackles' (Adm. Smyth). </R>
</sen>

<qbank>
   <quot>
      <qdat>1867 </qdat>
      <srce><auth>Smyth </auth><I>Sailor's Word-bk. </I></srce>
   </quot>
   <quot>
      <qdat>1894 </qdat>
      <srce><I>Times </I><R>27 Jan. 10/2 </R></srce>
      <qtxt><R>The larger weapons will be worked by electricity, but are also
         capable of being man-handled. </R></qtxt>
   </quot>
   <quot>
      <qdat>1902 </qdat>
      <srce><I>Blackw. Mag. </I><R>Mar. 331/2 </R></srce>
      <qtxt><R>I'm going to man-handle my gun down the slope. </R></qtxt>
   </quot>
   <quot>
      <qdat>1903 </qdat>
      <srce><I>Daily Chron. </I><R>19 Feb. 3/3 </R></srce>
      <qtxt><R>Stalwart Punjabis . . hand out bags of stores, . . or manhandle a
         fractious, restive animal. </R></qtxt>
   </quot>
</qbank>
```

```
<sen para=t lit='3.'> <lab>slang. </lab>
   <R>To handle roughly; to pull or hustle about. </R>
</sen>

<qbank>
   <quot>
      <qdat>1865 </qdat>
      <srce><I>Hotten's Slang Dict., </I></srce>
      <qtxt><I>Man-handle, </I><R>to use a person roughly, as to take him prisoner,
         turn him out of a room, give him a beating. </R></qtxt>
   </quot>
   <quot>
      <qdat>1886 </qdat>
      <srce><I>Century Mag. </I><R>Apr. 905/1 </R></srce>
      <qtxt><R>Two of our roughs began to haze him: but they mistook their calling,
         and in two minutes were so mauled and manhandled that it was reported
         aft. </R></qtxt>
   </quot>
   <quot>
      <qdat>1888 </qdat>
      <srce><auth>Clark Russell </auth><I>Death Ship </I><R>II. 253, </R></srce>
      <qtxt><R>I . . was for . . manhandling him, ghost or no ghost. </R></qtxt>
   </quot>
   <quot>
      <qdat>1891 </qdat>
      <srce><auth>Kipling </auth><I>Light that failed </I><R>iii, </R></srce>
      <qtxt><R>I'll catch you and man-handle you, and you'll die. </R></qtxt>
   </quot>
   <quot>
      <qdat>1894 </qdat>
      <srce><auth>R. H. Davis </auth><I>Eng. Cousins </I><R>24 </R></srce>
      <qtxt><R>The cry of 'Welsher', . . which sometimes on an English
         race-course means death from man-handling. </R></qtxt>
   </quot>
</qbank>


  </signif>
</entry>
```

Bibliography

[AC75] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, June 1975.

[Ams84] Robert A. Amsler. Machine-readable dictionaries. In Martha E. Williams, editor, *Annual Review of Information Science and Technology*, chapter 6, Knowledge Industry Publications, 1984.

[AU77] Alfred V. Aho and Jeffrey D. Ullman. *Principles of Compiler Design.* Addison-Wesley, Reading, 1977.

[Bur72] Robert W. Burchfield, editor. *A Supplement to The Oxford English Dictionary.* Clarendon Press, Oxford, 1972.

[Cas80] Frederic G. Cassidy. Some uses of computers in lexicography—the *DARE* experience. In Joseph Rabin and Gregory Marks, editors, *Databases in the Humanities and Social Sciences*, pages 185–189, North-Holland, Amsterdam, 1980.

[Dat85] C. J. Date. *An Introduction to Database Systems.* Volume 1, Addison-Wesley, Reading, 4th edition, 1985.

[dH85] Antonette diPaolo Healey. The dictionary of Old English and the final design of its computer system. 1985. Unpublished report, University of Toronto.

[FBP80] Mark S. Fox, Donald J. Bebel, and Alice C. Parker. The automated dictionary. *IEEE Computer*, 35–48, July 1980.

[GL70] M. Gross and A. Lentin. *Introduction to Formal Grammars.* Springer-Verlag, Berlin, 1970.

[HJBC82] G. E. Heidorn, K. Jensen, R. J. Byrd, and M. S. Chodorow. The EPISTLE text-critiquing system. *IBM Systems Journal*, 21(3):305–326, 1982.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, 1979.

[Hun75] E. B. Hunt. *Artificial Intelligence.* Academic Press, New York, 1975.

[Joh75] Stephen C. Johnson. *Yacc: Yet Another Compiler-Compiler.* Technical Report CSTR 32, Bell Laboratories, Murray Hill, New Jersey, 1975.

[Joh83] J. Howard Johnson. *Formal Models for String Similarity.* Technical Report CS-83-32, University of Waterloo, 1983.

[Joh84] J. Howard Johnson. *INR*: a program for computing finite automata. 1984. Unpublished report, University of Waterloo.

[Kip84] Barbara Ann Kipfer. *Workbook on Lexicography. Exeter Linguistic*, Wheaton, Exeter, 1984.

[Knu84] Donald E. Knuth. *The TEXbook.* Addison-Wesley, Reading, 1984.

[Kuv82a] Henry Kučera. The mathematics of language. In W. Morris and M. Berube, editors, *American Heritage Dictionary of the English Language*, pages 37–41, Houghton Mifflin, Boston, 1982.

[Kuv82b] Henry Kučera. Some grammatical properties of a large English data base. In J. Horecky, editor, *Coling 82*, pages 177–181, North-Holland, Amsterdam, 1982.

[Les75] Mike Lesk. *Lex—A Lexical Analyzer Generator.* Technical Report CSTR 39, Bell Laboratories, Murray Hill, New Jersey, 1975.

[MBCO33] James A. H. Murray, Henry Bradley, William A. Craigie, and Charles T. Onions, editors. *The Oxford English Dictionary.* Clarendon Press, Oxford, 1933.

[McN82] John McNaught. Specialised lexicography in the context of a British linguistic data bank. In J. Goetschalckx and L. Rolling, editors, *Lexicography in the Electronic Age*, pages 171–184, North-Holland, Amsterdam, 1982.

[MFGK82] N. H. Macdonald, N. H. Frase, P. Gingrich, and S. A. Keenan. The WRITER'S WORKBENCH: computer aids for text analysis. *IEEE Transactions on Communication*, 30:105–110, 1982.

[Mic81] Archibal Michiels. *Exploiting a Large Dictionary Database.* PhD thesis, University of Liege, Belgium, 1981.

[Mur79] K. M. Elisabeth Murray. *Caught in the Web of Words.* Clarendon Press, Oxford, 1979.

[New84] *New OED System Requirements.* Oxford, 1984. Internal document.

[New85] *New OED System Design.* Oxford, 1985. Internal document.

[Nor82] Ole Norling-Christensen. Commercial lexicography on the threshold of the electronic age. In J. Goetschalckx and L. Rolling, editors, *Lexicography in the Electronic Age*, pages 211–219, North-Holland, Amsterdam, 1982.

[NTUT82] M. Nagao, J. Tsujii, Y. Ueda, and M. Takiyama. An attempt to computerize dictionary data bases. In J. Goetschalckx and L. Rolling, editors, *Lexicography in the Electronic Age*, pages 51–73, North-Holland, Amsterdam, 1982.

[OR72] James Olney and Donald Ramsey. From machine-readable dictionaries to a lexicon tester: progress, plans, and an offer. *Computer Studies in the Humanities and Verbal Behavior*, 3(4):213–220, 1972.

[OT82] Ivan I. Oubine and Boris D. Tikhomirov. Machine translation systems and computer dictionaries in the information service. Ways of their development and operation. In J. Horecky, editor, *Coling 82*, pages 289–294, North-Holland, Amsterdam, 1982.

[Pet82] James L. Peterson. Use of *Webster's Seventh New Collegiate Dictionary* to construct a master hyphenation list. In *AFIPS Proceedings of the National Computer Conference*, pages 665–670, AFIPS Press, Arlington, 1982.

[Ric85] Stephen J. Richardson. *Enhanced Text Critiquing using a Natural Language Parser.* Technical Report RC 11332, IBM Research Division, 1985.

[Rob83] Dan Robinson. Much more than a spelling checker. *Creative Computing*, 9(11):108, 111–112, Nov. 1983.

[ROP69] Richard Reichert, John C. Olney, and James Paris. *Two Dictionary Transcripts and Programs for Processing Them.* Technical Report TM-3978/001/00, System Development Corporation, 1969.

[ST84] John Stubbs and Frank Wm. Tompa. Waterloo and the New Oxford English Dictionary project. 1984. Submitted to the Twentieth Annual Conference on Editorial Problems, University of Toronto, November 1984.

[Tom86] Frank Wm. Tompa. Database design for a dictionary of the future. 1986. Submitted to the Conference: A Workshop on Automating the Lexicon, Pisa, May 1986.

[Urd66] Laurence Urdang. The systems designs and devices used to process *The Random House Dictionary of the English Language. Computers and the Humanities*, 1(2):31–33, Nov. 1966.

[Wal85] Donald E. Walker. *Knowledge Resource Tools For Accessing Large Text Files.* Technical Report 85-21233-25, Bellcore, 1985.

[WB85] Stephen A. Weyer and Alan H. Borning. A prototype electronic encyclopedia. *ACM Transactions on Office Information Systems*, 3(1):63–88, Jan. 1985.

[Wei84a] Edmund S. C. Weiner. Elements in the dictionary text that should be identified by the mark-up system. 1984. *New OED* Workbook paper.

[Wei84b] Edmund S. C. Weiner. The omission of elements from the mark-up and its consequences. 1984. *New OED* Workbook paper.

[Wei85a] Edmund S. C. Weiner. The *New OED*: problems in the computerization of a dictionary. *University Computing*, (7):66–71, 1985.

[Wei85b] Edmund S. C. Weiner. The new Oxford English Dictionary. *Association for Literary and Linguistic Computing*, 13(1), 1985.

[Win83] Terry Winograd. *Language as a Cognitive Process.* Volume 1, Addison-Wesley, Reading, 1983.

[Win84] Terry Winograd. Computer software for working with language. *Scientific American*, 251(3):130–145, Sept. 1984.