Creation of a Prolog Fact Base from the Collins English Dictionary

Robert C. Wohlwend
Edward A. Fox

TR 88-24

TABLE OF CONTENTS

Introduction **1**

Literature Review **4**

Description of the Collins English Dictionary (CED) **7**

Relations in the Lexicon **17**

The Hierarchical Nature of the Dictionary 17

Alphabetical List of Relations 20

Lex and Yacc **30**

The Basics of Lex 30

Lex variables 31

General Structure of Lex Source 33

Definitions 33

Rules 37

Subroutines 39

Yacc Overview 39

Yacc variables 41

General Structure of Yacc Source 42

Declarations 42

Yacc rules 44

Yacc programs 45

Error handling 45

Concluding Remarks Concerning Lex and Yacc 46

The Nine Pass Approach 47

Pass1 50

Yacc Grammar 56

Explanation of Yacc Grammar 57

Pass2 60

Pass3 64

Pass4 68

Pass5 69

Pass6 71

Pass7 72

Pass8 75

Pass9 77

Dictionary Processing Approach 78

Cleaning Up 81

Statistics and Observations 84

Frequencies of Relations in the CED. 84

Frequencies of Categories 88

Frequencies of Homograph Numbers 92

Frequencies of Parts of Speech 93

CED Parts of Speech With Examples 98

Frequencies of Senses 100

Frequencies of Sub-senses 104

Conclusions 106

Appendix A. Pass1 110

C Program for Pass1 110

Yacc Program for Pass1 110

Lex Program for Pass1 114

Appendix B. Pass2 138

Appendix C. Pass3 145

Appendix D. Pass4 152

Appendix E. Pass5 158

Appendix F. Pass6 162

Appendix G. Pass7 164

Appendix H. Pass8 176

Appendix I. Pass9 179

Bibliography 181

LIST OF ILLUSTRATIONS

Figure 1. Hierarchical Structure of the CED 19

Figure 2. Dictionary Processing Approach 80

LIST OF TABLES

Table 1. Reference Table of Relations 28

Table 2. Reference Table of Syntax of Relations 29

Table 3. Table of Common Translations 50

Table 4. Table of Non-parseable Entries 81

Table 5. Comparative Frequencies of Items in the CED 85

Table 6. Comparative Frequencies of Categories in the CED 90

Table 7. Comparative Frequencies of Homograph Numbers in the CED 92

Table 8. Comparative Frequencies of Parts of Speech in the CED . 96

Table 9. Frequencies of Parts of Speech in the CED 99

Table 10. Comparative Frequencies of Sense Numbers in the CED . 102

Table 11. Comparative Frequencies of Sub-sense Numbers in the CED 104

INTRODUCTION

Machine readable dictionaries may be invaluable components of future information retrieval systems. The Virginia Tech department of Computer Science is fortunate to have available for research purposes machine readable versions of the Collins English Dictionary (CED) [HANK 79] and of the Oxford Advanced Learners Dictionary of Current English (OALDCE) [HORN 74] among others. McIllroy [MCIL 84] and Mitton [MITT 85] have put forth separate efforts in cleaning up the OALDCE. The CED is only available in its original typesetting form, and it required a great deal of effort and patience to transform this into a relational lexicon.

A number of recent efforts have been made to generate relational lexicons from dictionaries (eg., [AHLS 81], [AHLS 83], [AHLS 85], [WHIT 83], [EVEN 78]). The lexicon that will be discussed in this paper consists of a large set of relations in the form of PROLOG facts. This knowledge base will become a very important entity in the analysis and retrieval subsystems of the CODER project [FOXE 86].

The CODER (**CO**mposite **D**ocument **E**xpert/**E**xtended/**E**ffective **R**etrieval) system is an experimental system to investigate the application of artificial intelligence techniques to the task of analyzing, storing, and retrieving heterogenous collections of "composite documents". The CODER system is comprised of an analysis subsystem, a retrieval subsystem, and a common "spine" of knowledge bases and data management modules. The modules

contain managers that have the responsibility of handling the structure of the knowledge that is used and passed through the system. The knowledge bases include the document database which contains the documents, and the lexicon which holds information about relationships between words. The lexicon contains both general knowledge about words and specialized knowledge. The central spine is connected to blackboards through experts. The experts have specialized knowledge and make decisions and post hypotheses on the blackboards, each of which is controlled by a stategist. It is the lexicon that is of primary concern in this project.

The lexicon is to contain many facts about words. MU-PROLOG [NAIS 80], a LOGic PROgramming language which includes some useful extensions to more common PROLOG implementations, was chosen to represent these facts. The PROLOG facts are required to begin with a lower-case letter, followed by a list of one or more atoms or lists enclosed by and ending in a period. That is, a PROLOG fact is of the form: c_RELATION_NAME(list[1], list[2], ... list[n]). where list[i] is an atom (indivisible), a list enclosed with brackets, or an element enclosed with single quotes, which is also an atom (eg., noun, [noun, verb], 'verb' respectively). I chose to begin all facts (relations) with the prefix c_ to distinguish facts obtained from the Collins English Dictionary from other facts.

This paper documents the efforts taken to generate a relational lexicon from the machine readable form of the CED. First, related literature will be reviewed and background information concerning the CED, the relations,

and the software environment is provided. Next, the approach and the phases of processing the CED are described. Finally, observations and conclusions are made concerning the results of generating this relational lexicon.

LITERATURE REVIEW

A number of efforts have been undertaken to create a relational lexicon, that is, a database of words including relationships implied among them. Apresyan et al. [APRE 69] in a pioneer effort described a set of lexical functions that comprise the entries of the Explanatory Combinatory Dictionary (ECD) of modern Russian. Mel'cuk proposed a linguistic model [MELC 73] to correlate a given meaning with all synonomous texts that have that same meaning. This "meaning <==> text" model was used to create a lexicon of Russian text [MELC 85].

The creation of a relational lexicon requires the careful analysis of a text. Several dictionaries have been analyzed and reformatted in the past few years. Sherman [SHER 74] formatted the Webster's Seventh in an effort that concentrated heavily upon phonetic fields. Amsler's work with noun and verb kernels and his grammar to parse verb definitions in the MWPD [AMSL 80], [AMSL 81] are classic works in the area of definition parsing. Peterson [PETE 82] converted the Webster's Seventh New Collegiate Dictionary from machine-readable form into fields. White created several tools to preprocess dictionary entries and discusses these in [WHIT 83]. In an effort similar to Peterson's, Mitton [MITT 85] formatted the Oxford Advanced Learner's Dictionary of Current English.

Several people have indicated the value and need for relational lexicons. Evens and Smith organized a lexicon to support their question-answering system [EVEN 79]. In 1980 Fox [FOXE 80] expressed the need for the construction of a relational lexicon. He had determined that by using lexical and semantic relations in information retrieval systems there was a noticeable improvement in the system's overall performance. Evens [EVEN 85] discussed lexical semantic relations, the method of determining relations, and methods of automatically producing definitions of Webster's Seventh vocabulary that have no definitions.

There are a number of approaches to take in the automatic construction of a relational lexicon from machine-readable sources. A relational network was created by Evens [EVEN 82] from lexical-semantic relations that were extracted from the MWPD. In 1984 Hobbs discussed a methodology for the construction of knowledge bases [HOBB 84]. Ahlswede and Evens [AHLS 84] discuss their approach to creating a relational lexicon for a medical expert system from the Webster's Seventh. Ahlswede wrote a lexicon builder that will interactively generate lexical entries to form a lexical-semantic network. Further, he proposed the creation of an automatic lexicon builder based on his interactive generator [AHLS 85].

There have been a number of recent efforts to utilize machine-readable texts in connection with advanced computer systems. [FOXM 80] discusses the creation of an automated dictionary as well as the cost-factors involved in maintaining and expanding this online dictionary. Various text searching approaches in online encyclopedias and various sociological

effects of this form of text are analyzed in [HART 81]. There are numerous applications of an electronic lexicon, ranging from spelling verification and correction to grammar to word games [WEBB 84]. However, the creation of a relational lexicon from machine-readable texts requires sufficient knowledge of the original texts.

DESCRIPTION OF THE COLLINS ENGLISH DICTIONARY (CED)

One of the main goals of my work with this lexicon is to represent as much of the CED as possible in a small set of relations. In going beyond the processing for standard dictionary structures, the method that I most frequently employed was the creation of a new relation whenever the dictionary entry text did not comply with any previously defined relations. This section deals with the basic structure of the CED, and the method by which I obtained relations for the lexicon. This section is followed by a detailed description of each of these relations.

1. Headwords

A headword is a word or phrase that is defined in the dictionary. It is usually in lower-case, but can also be one of the following:

- Proper names.

  Ex: Aalto, Milazzo.

- Mixed alphanumerics and numerics.

  Ex: A1, B-1.

- Words with embedded, leading, and trailing apostrophies and hyphens.

  Ex: A'asia, across-the-board.

- Prefixes, suffixes, and combining forms.

  Ex: in- , -able, psycho- , -iatry.

- Compound words.

  Ex: elementary particle.

- Foreign terms.

  Ex: haut monde

- Abbreviations, symbols, and acronyms.

  Ex: A.A.A., K., JUGFET

2. Homograph Numbers

Many words have the same spelling but have different origins. These words are called homographs and are different entries in the dictionary. Associated with each entry is a homograph number (Ex: saw,1 as a noun compared with saw,2 the verb). The homograph number in the lexicon is the first number to follow the headword.

3. Variant Spellings

These alternate forms of the headword are common, acceptable spellings of English words. c_VAR_SPELL is the relation that takes into account any variant spellings. The terms 'sometimes, occasionally, often, masc., fem., U.S., before a vowel, esp.' and others

are commonly used after an 'or' to indicate the presence of a variant spelling. In the following examples 'center', 'capitalise', and 'actress' are the variant spellings.

Ex: centre or U.S. center.

Ex: capitalize or capitalise.

Ex: actor or fem. actress.

4. Syllabification of Headword and Variant

Syllabification breaks are given for all headwords and variant spellings. The relation c_SYLL indicates these breaks by placing an underscore character (_) between each syllable. The c_VAR_SYLL relation is similarly defined for variant spellings.

5. Pronunciation

The pronunciation fields of the CED were ignored for this lexicon. Given the needs of the CODER project, this information was too difficult to extract. Complex character code translation would have been necessary. Furthermore, occurrences were rather scattered; variant pronunciations even occurred within definition text. Additionally, these pronunciations appeared in the position the category information usually occupied.

6. Inflected Forms

The regular inflections are not given in the CED. Regular inflections are most commonly used to change tenses of a verb, quantity of a noun, and degree of an adjective.

Ex: goose pl. geese --> c_PLURAL('goose', 'geese')

Ex: aardwolf pl. wolves --> c_PLURAL('aardwolf', 'wolves')

Ex: abet vb. abets, abetting, abetted -->

c_PLURAL('abet', 'abets')

c_PLURAL('abet', 'abetting')

c_PLURAL('abet', 'abetted')

The c_PLURAL relation will be expanded to the c_INFLECT relation to distinguish between the plural forms of nouns, the irregular inflections of verbs, and the irregular forms of adjectives. The difference between plural forms of nouns, irregular inflections of verbs, and comparative and superlative forms of adjectives can be determined by knowing the part of speech of the headword. This distinction is discussed in more detail in the discussion of the c_PLURAL relations.

7. Parts of Speech

The CED recognizes the eight standard parts of speech: adjective, adverb, conjunction, interjection, noun, preposition, pronoun, and verb. A verb headword sense may be transitive (requiring a direct object) or intransitive (not requiring a direct object). Table 8 on page 94 and Table 9 on page 98 list all parts of speech. Other less

traditional parts of speech are also included in the CED, and are given below:

- determiner - words such as 'that', 'this', 'his', and 'my'. In many dictionaries these have been classified as demonstratives, possessives, adjectives, and/or pronouns.

- sentence connectors - words such as 'therefore' and 'however' have been previously listed as adverbs and conjunctions.

- sentence substitutes - words such as 'affirmative' and 'aye' that can stand alone as meaningful utterances, and are distinguished from interjections.

- adjective postpositive - words such as 'ablaze' and 'abaft' that are used predicatively or after the noun, but not before the noun.

- adjective prenominal - words such as 'acting' and 'all-around' that are used before the noun, but never predicatively.

- plural labelling of nouns - words such as 'trousers' and 'abuttals' that are senses of the noun usually used in the plural form.

- modifier - sense of a word such as 'absentee', that is commonly used much like an adjective in strength, and often found within the definition text itself.

8. Abbreviations, Symbols, and Acronyms

Abbreviations and symbols are incorporated in the part of speech relation c_POS as 'abbrev' and 'symbol'. Occasionally an abbreviation is found within an entry, as opposed to being an entry in itself. The c_ABBREV relation gives the abbreviation of the headword.

Ex: c_ABBREV('abampere', 'abamp.')

All acronyms are considered to be nouns.

9. Restrictive Labels and Phrases

A given sense occasionally has a label, which I call a category, that restricts the use of the given word (sense) to subject field, appropriateness, connotation, nationality, etc. These categories may be classified as:

- temporal categories - such as archaic, obsolete.

- usage categories - such as slang, informal, taboo.

- connotative labels - such as derogatory, offensive.

- subject-field labels - such as computer technol., astron., and banking.

- national and regional labels - such as Austral., N.Z., and S. African.

- trademarks - as in the headwords: Acrilan, Addressograph.

10. Senses

When a headword has more than one sense (where sense n corresponds to definition n), the first sense is the most common usage of the headword. Each succeeding sense is less frequently used than its predecessor.

11. Subdefinitions

Occasionally a sense of a word may have several slightly different meanings, but still lie in the realms of the given sense. Subdefinitions can be used to show a very slight variation in sense of a word, especially in certain technical fields. These subdefinitions can also show the use of a word as a different part of speech without changing the true meaning. For example, the word 'beige' is a noun indicating a color, but may also be used as an adjective as in 'beige gloves'. The authors of the CED treat each of these definitions as a subdefinition. Finally, subdefinitions can be used to show that a

number of senses of the word fall under the same category. A category pertains to the given sense of the word, as well as any sub-senses that may be found.

Ex: c_DEF(['content', 1,1,2,1], 'the chapters or divisions of a book',1).

Ex: c_DEF(['content', 1,1,2,2], 'a list, printed at the front of a book, of chapters or divisions together with', 1).

Ex: c_DEF(['content', 1,1,2,2], 'the number of the first page of each',2).

12. Examples of Typical Use

Example sentences and phrases illustrate the use of a particular sense of a word. The c_SAMP relation takes into account these sample sentences and phrases.

13. Usage Notes

Occasionally comments are found within the entry text that remark on the usage of a word (sense). These usage notes are represented in the lexicon in the c_USAGE relation.

14. Cross-References

A few different types of cross-references were evident in the CED.

- Related Adjectives - Some nouns with one national origin are related to adjectives with another national origin. For instance, a noun with Germanic origin such as 'wall' is related to the adjective 'mural' of French origin. The c_RELADJ relation of the lexicon corresponds to these related adjectives of the CED.

- Comparisons - Another form of cross-reference is evident in the c_COMPARE relation. The words 'See', 'See also', and 'Compare' introduce a comparison to a word elsewhere in the dictionary. The homograph number is always included in this relation if there is more than one homograph number for the target word. The sense and subsense numbers are also included when the comparison relates to a word or words with more than one sense.

- Alternative Names - 'Also', 'Also called', and 'Official Name:' introduce alternative names for the given headword. The c_ALSO relation in the lexicon represents an occurrence of an alternative name.

15. Biographical Entries

About 2500 famous people's names are given headword entries in the CED. These entries appear separately from place names of the same spelling. First names, pseudonyms, nicknames, titles, original names, etc., for these entries are found in the c_NLAST relation.

16. Morphological Variants

Morphological variants (different forms of the headword that are not headwords individually) can be found at the end of an entry. The part of speech of the morphological variant is also included. The meaning of the morphological variant is determined by the meaning of the suffix and headword of the variant. The c_MORPH functor identifies this type of relationship.

17. Etymologies

Etymologies appeared to be useless for our applications, and so were ignored.

RELATIONS IN THE LEXICON

THE HIERARCHICAL NATURE OF THE DICTIONARY

Figure 1 on page 19 shows the hierarchical structure of the CED. There are five levels in all. At the upper most or LEXEME level we have the word in question. For purposes of information retrieval, a given document is initially entered as a list of lexemes. It is the purpose of one of the CODER "managers" to determine the HEADWORD associated with each lexeme. At this headword level there are words that have different homographs (different etymological sources), but have the same spelling. A separate headword is produced for each homograph. Below the headword level is the SYNTAX level which is determined by the part of speech. Within an entry, the part of speech precedes the sense or senses that relate to that part of speech. Therefore, the SENSE level is the next level. Occasionally a word sense may even be subdivided into parts, that is subdefinitions, as explained in the description of the CED.[1] The SUB-SENSE level indicates this bottom-most level. In summary, the levels of hierarchy of the CED are: lexeme, headword, syntax, sense, sub-sense.

---

[1] The core meaning of the word at the given level remains the same, so no change in SYNTAX level is made. For example, the word "will" used in the context of Law may indicate either the "declaration of a person's wishes concerning the disposal of his property after he dies", or the document itself. The basic meaning of the word remains throughout both definitions, so each are subdefinitions.

A number is associated with each of these levels in the syntax of the relations that were created.

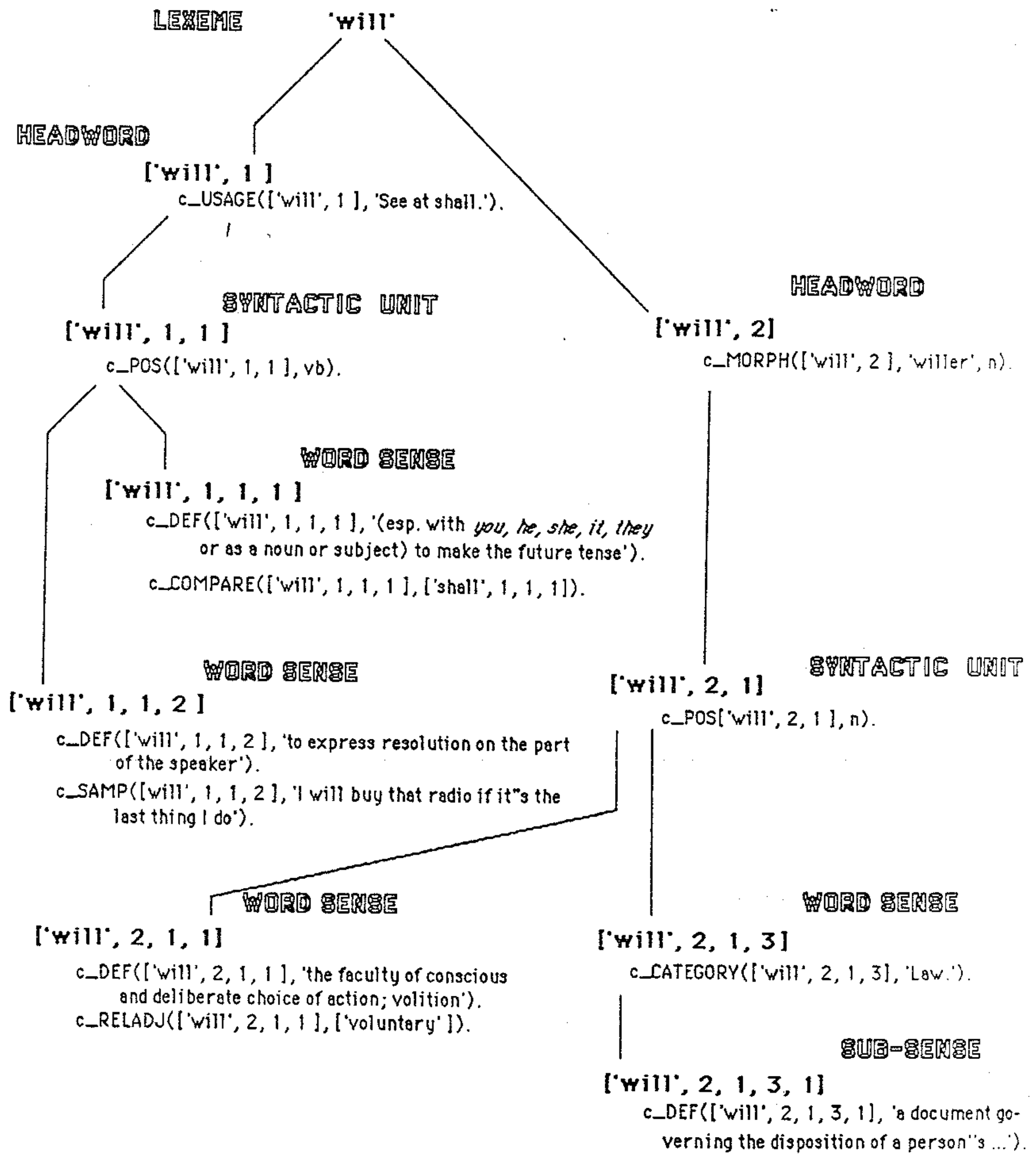

Figure 1. Hierarchical Structure of the CED

ALPHABETICAL LIST OF RELATIONS

The hierarchical nature of the dictionary is evident in the syntax of the relations. The first element, in the form of a list, which appears as the first entry in each of the relations describes this hierarchical level in which the relation is recognized. For example, a definition relation (c_DEF) is recognized at the SENSE or the SUB-SENSE level, so the first element is a list containing the LEXEME, the HEADWORD level, the SYNTAX level, the SENSE level, and possibly the SUB-SENSE level. This list may be generalized as: [LEXEME, HEADWORD, SYNTAX, SENSE, SUB-SENSE] where LEXEME is a quoted string and HEADWORD, SYNTAX, SENSE, and SUB-SENSE are integers (>0). The following relations and their format (by example) are given below:

c_ABBREV - This relation associates with the headword an abbreviation that is also found as a headword. This can be a very important cross-reference check for information retrieval. The following forms in the CED text produce this relation:

- abbrev.:
- abbrevs.:

Ex: c_ABBREV(['adjective',1], 'adj.').

c_ALSO_CALLED - This relation correlates the headword with an alternative name. The alternative name is introduced by any of the following words:

- also:
- also called:
- former name:

Ex: c_ALSO_CALLED(['air base',1], 'air station').

c_CATEGORY - The c_CATEGORY relation binds a sense of the headword to a temporal label such as archaic or obsolete, a usage label such as slang or informal, a connotative label such as offensive or derogatory, or subject field labels such as South African or Northeastern U.S. Occasionally, the category for a given sense is found within the definition text of the c_DEF relation (see the c_DEF relation description below). Most of the categories, however, appeared separately from the definition text, and a new c_CATEGORY relation was created. Abbreviations were often used to indicate a category, but these abbreviations were not found to be entirely consistent. The most reliable signal for a new category is a change in font to italics, where the first letter is usually capitalized. A list of the most common categories is found in Table 6 on page 88.

Ex: c_CATEGORY(['ambidextrous',1], 'Informal.').

c_COMPARE - This relation associates the headword with another word found elsewhere in the dictionary. The relation sometimes includes the homograph number and sense number. If it has neither, the comparison is to all senses of the headword. A distinction is made for words with more than one homograph number. The c_COMPARE relation is produced when introduced by any of the following words:

- see:
- see also:
- compare:

Ex: c_COMPARE(['set off',1],['counterclaim',1,1]).

c_DEF - This relation gives the definition text for a word. Some definitions may be longer than eighty characters in length, and MU-PROLOG allows statements no longer than 160 characters. The definition text was divided into eighty character (or less) blocks, and tagged with a number indicating the position of this block within the definition as a whole.

Ex: c_DEF(['gaby',1,1,1], 'a simpleton' ,1).

Ex: c_DEF(['gadroon',1,1,1], 'a moulding composed of a series of convex flutes and curves joined to form a' ,1).

c_DEF(['gadroon',1,1,1], 'decorative pattern, used esp. as an edge to silver articles' ,2).

c_DEF_NUM - This relation has an entry for each definition. The last number in this relation indicates the total number of eighty (or less) character blocks that are associated with the definition. Thus, the total number of definitions in the dictionary is the number of occurrences of this DEF_NUM relation.

Ex: c_DEF_NUM(['gaby',1,1,1], 1).

Ex: c_DEF_NUM(['gadroon',1,1,1], 2).

c_HEADWORD - Each headword in the dictionary becomes a part of the c_HEADWORD relation. The homograph number for a word is the second sub-entry in this relation. Headwords without homograph numbers are given the default of one for that second attribute.

Ex: c_HEADWORD(['ambience',1]).

c_MORPH - Morphological variants of the headword are expressed through the c_MORPH relation. The part of speech of each morphological variant is indicated by the third item in this relation.

Ex: c_MORPH(['assist',1], 'assister' , n).

Occasionally a headword has two or more morphological variants with the same part of speech associated with them. The preferred morphological variant is the first of the list. The programs will separate the list into individual c_MORPH relations, each with the given part of speech.

Ex: ambidextrous morph ambidexterity or ambidextrousness, n -->

c_MORPH(['ambidextrous',1], 'ambidexterity' , n).

c_MORPH(['ambidextrous',1], 'ambidextrousness' , n).

c_NLAST - The NLAST (not the last name) relation is produced when a font change occurs under certain situations mentioned in "Pass1" on page 50. The person's first name, nickname, original name, pseudonym, family name, title, or lifespan, individually or in combination, are appropriate for the c_NLAST relation.

Ex: c_NLAST(['Henry VIII',1], '1491-1547, king of England (1509-47); second son of Henry VII').

c_PAST - Occasionally the past tense of a verb appears in the CED. The c_PAST relation is formed when this occurs, and the second item in the PROLOG fact is this past form.

Ex: c_PAST(['shall',1,1,1], 'should').

Ex: c_PAST(['may',1,1,1], 'might').

c_PLURAL - The c_PLURAL relation gives the irregular plural form of nouns, irregular inflections of verbs (third person singular, present participle, and past participle), or the irregular comparative and superlative forms of adjectives. The c_PLURAL relation sometimes provides just the ending, and other times provides the related word in its entirety. Initially, when I saw potential problems in extracting the inflected forms, it was decided by the members of the CODER project to extract only the ending if it were too difficult to recognize when the dictionary provided the ending or the entire inflected form. The dictionary signals an ending with a dot, but this fact was not realized until

processing had begun. Nevertheless, for any c_PLURAL relation, the difference between plural forms of nouns, irregular inflections of verbs, and comparative and superlative forms of adjectives can be determined by knowing the part of speech. The following examples best explain the distinction made:

Ex: c_PLURAL(['abbacy',1], 'cies') is the plural form of a noun since 'abbacy' is a noun.

Ex: c_PLURAL(['good',1], 'better') is the comparative form of the adjective 'good' (Comparative forms are always presented before the superlative).

Ex: c_PLURAL(['lap',3], 'laps') is the third person singular form of the verb 'lap' (The order presented in the CED is: third person singular, present participle, past participle).

c_POS - The part of speech of the headword is provided in the c_POS relation. Table 8 on page 94 and Table 9 on page 98 illustrate the parts of speech, including examples and frequency counts.

Ex: c_POS(['amber',1,1], n).

c_RELADJ - Another cross-reference is evident in the RELADJ relation. Some nouns have related adjectives of similar origins to the given noun headword. The adjective is the second argument in the RELADJ relation.

Ex: c_RELADJ(['air',1],['aerial',1]).

c_SAMP - Example phrases and sentences are often given to illustrate the use of the word sense. These sample usages are not expected to be more than eighty characters in length, so they should fit into one MU-PROLOG fact.

Ex: c_SAMP(['ambi-',1,1,1], 'ambidextrous; ambivalence').

Semicolons are used to separate a list of compatible definitions at the given sense level. Semicolons also separate sample usages in the same way.

c_SINGULAR - Sometimes the singular labelling of a noun appears in the CED. Occasionally the singular form that is given is only the ending or the final syllable. Again the distinction is not made between the singular form and the ending.

Ex: c_SINGULAR(['Parcae',1,1,1], 'Parca').

c_SYLL - This relation syllabifies the headword by including an underscore character to divide the syllables.

Ex: c_SYLL(['amber',1],['am_ber']).

c_USAGE - Usage notes are included to comment on matters of usage of the headword. This relation is also restricted to eighty character (or fewer) blocks, as with the c_DEF relation, since usage notes tend to be relatively long. The final number in this relation indicates the position of a particular tuple with respect to the set of tuples that represent the entire usage note.

Ex: c_USAGE(['actual',1], 'Avoid excessive use of actual',1).

Ex: c_USAGE(['may',1], 'In careful written usage, may is used rather than can when reference is made to',1).

c_USAGE(['may',1], 'permission rather than capability',2).

c_USAGE_NUM - This relation is instantiated once for a given usage note. Hence the number of occurrences of usage notes is the sum of all occurrences of this relation.

Ex: c_USAGE_NUM(['actual',1], 1).

Ex: c_USAGE_NUM(['may',1], 2).

c_VAR_SPELL - Variant spellings of headwords are found in the c_VAR_SPELL relation. Often the U. S. spelling of a headword differs from the British spelling (e.g. center, centre). Being a British dictionary, the headword is the British spelling of the word; the U. S. spelling is the variant spelling and is the second argument of this relation.

Ex: c_VAR_SPELL(['centre',1], 'center').

Ex: c_VAR_SPELL(['ambience',1], 'ambiance').

c_VAR_SYLL - This relation syllabifies a variant spelling, again by dividing the syllables with an underscore character.

Ex: c_VAR_SYLL(['ambiance'],['am_bi_ance']).

The following table presents all relations that were created from the CED and a brief explanation to be used as a quick reference:

c_ABBREV	- Abbreviation of headword.
c_ALSO_CALLED	- Headword is also commonly called this.
c_CATEGORY	- Category (semantic label) of headword.
c_COMPARE	- Compare to another headword and sense(s).
c_DEF	- Definition of the headword.
c_DEF_NUM	- Number of (about 80) character blocks of the definition.
c_HEADWORD	- The headword entry.
c_MORPH	- Morphological variant of headword (incl. part of spch.).
c_NLAST	- Rest (ex. first/middle name) of a proper noun headword.
c_PAST	- Past form of headword.
c_PLURAL	- Plural of headword (sometimes just the ending).
c_POS	- Part of speech.
c_RELADJ	- Related adjective to headword.
c_SAMP	- Example of headword in context.
c_SINGULAR	- Singular form of headword (sometimes just the ending).
c_SYLL	- Syllabification of headword.
c_USAGE	- Usage notes providing guidance on usage of the headword.
c_USAGE_NUM	- Number of (up to 80) character blocks in the usage note.
c_VAR_SPELL	- Variant spelling(s) (if any).
c_VAR_SYLL	- Syllabification of variant spelling(s).

Table 1. Reference Table of Relations

The following list presents the syntax of the PROLOG facts:

```
c_ABBREV([ word, homnum, defnum ], relword).
c_ALSO_CALLED([ word, homnum, defnum ], relword).
c_CATEGORY([ word, homnum, defnum ], relword).
c_COMPARE([ word, homnum, defnum ],[ relword, homnum, posnum ]).
c_DEF([ word, homnum, defnum, subnum ], relword, blocknum).
c_DEF_NUM([ word, homnum, defnum, subnum ], relword, totalblocks).
c_HEADWORD([ word, homnum ]).
c_MORPH([ word, homnum ], relword, pos).
c_NLAST([ word, homnum, defnum ], relword).
c_PAST([ word, homnum, defnum ], relword).
c_PLURAL([ word, homnum ], relword).
c_POS([ word, homnum ], pos).
c_RELADJ([ word, homnum, defnum ], [ relword, homnum ]).
c_SAMP([ word, homnum, defnum, subnum ], relword).
c_SINGULAR([ word, homnum, defnum ], relword).
c_SYLL([ word, homnum ], syllword).
c_USAGE([ word, homnum ], relword, blocknum).
c_USAGE_NUM([ word, homnum ], relword, totalblocks).
c_VAR_SPELL([ word, homnum ], relword).
c_VAR_SYLL([ word, homnum ], syllword).
```

```
where:
    word        = 'lexeme' (in single quotes)
    homnum      = headword level (integer)
    relword     = 'word' or 'phrase' (in single quotes)
    posnum      = syntactic level (integer)
    pos         = part of speech (atomic element, no quotes)
    blocknum    = number of current block (integer)
    totalblocks = total number of blocks to contain text (integer)
    syllword    = 'syllabified word' (syllables connected by (_))
    defnum      = sense level (integer)
    subnum      = sub-sense level (integer)
```

Table 2. Reference Table of Syntax of Relations

LEX AND YACC

This section is included to give the reader a general understanding of some of the tools that were used in this project. It is provided to allow the "non-computer" type of person to be able to appreciate the processing that the dictionary went through during conversion to the final product - a PROLOG fact base. More detailed discussions of these tools may be found in [LESK 75], [JOHN 75], and [KERN 84].

The dictionary was transferred from magnetic tape to a SUN-2 workstation which runs one version of the UNIX (trademark of AT&T Bell Laboratories) operating system. It is in this environment that the project was developed. The UNIX operating system supports (and is written primarily in) the C programming language, which is the basis for the Lex and Yacc utilities. The original form of the CED was in typesetting codes, and it was decided that the Lex and Yacc utilities would be used to transform these codes into the form of PROLOG facts.

THE BASICS OF LEX

Lex is a lexical analyzer generator. Its source input is a list of regular expressions followed by semantic actions. Lex produces a program in the C programming language which matches these regular expressions against the input stream. If a match occurs, the given semantic action(s) will be taken. At any point in the input stream, the Lex generated pro-

gram will find which regular expression matches the longest stream of input. It will then execute any action(s) associated with that regular expression. Together the regular expression and the action(s) comprise a rule. Rules are described in a subsequent section.

When one wants to use Lex, one must first produce the Lex source code, compile it by using Lex to produce a program, and then compile and load this program with a library of Lex subroutines. For example, if one has a Lex file called file.l of regular expressions with semantic actions, one must produce the C program 'yylex' by typing the following:

```
lex file.l
```

The resulting 'yylex' program is placed in the file called 'lex.yy.c', which can be compiled by typing:

```
cc lex.yy.c -ll
```

The '-ll' option includes the Lex library of subroutines. The compiled version will then reside in 'a.out' until it is moved and given a new name such as 'file'.

LEX VARIABLES

Lex has a variety of internal variables. Below is a description of the principal ones a user might employ in writing lexical analyzers.

1. yytext - an external ("extern") character array that holds the matched pattern of the regular expression (the token found).

2. yyleng - the number of characters matched on the input stream (token length). The last character in the matched string is given by the C expression yytext[yyleng - 1].

3. yymore() - indicates that the next input expression to be recognized shall be appended to the end of the current input.

4. yyless(n) - a pushback function that will push the current input stream back n characters.

5. input() - returns the next input character.

6. output(c) - writes the character c on the output.

7. unput(c) - pushes the character c back onto the input stream.

8. yywrap() - called to do wrap-up when the Lex generated program reaches an end-of-file. It can be modified to allow the printing of summaries, tables, lists, etc. at the end of a program.

9. yylex.c - the program produced from the user's Lex file of regular expressions and semantic actions.

Note: 'Input', 'output', and 'unput' are macros, but may be redefined by the user.

GENERAL STRUCTURE OF LEX SOURCE

```
Definitions
%%
Rules
%%
Subroutines
```

Note: No blank lines are allowed, nor may comments begin any line. Comments are enclosed by /* and */ and may be placed anywhere after a regular expression or after a definition.

Definitions

In the definitions section, there are several options available:

1. Definitions in the form of 'name translation'

    Ex: `pattern    [tT][hH][*]*[iI][\¬]+[Ss][\-\+][\n]`

    Ex: `pattern2   {pattern}[iIsSwW][eE][iIrR]*[dD][\\]`

    This type of definition is provided so that in the rules section (see below) one can refer to patterns repeatedly without the need to recreate those patterns. A maximum of 40 'name translation' definitions are allowed.

2. Code may be included in two ways.

- Inserting a space character before each line of the included code.

- Delineating the code in the definitions section as follows:

```
%{
...
<code>
...
%}
```

This code can be # include files, definitions of variables to be used in the rules section, redefinitions of Lex default variables, etc.

3. Start conditions

The use of start conditions on rules allows the grouping of the entire set of rules. Any rule may be associated with a start condition and that rule will only be recognized while Lex is in that start condition. All start conditions must be defined in the definitions section.

%Start name1 name2 in the definitions section indicates that **name1** and **name2** are start conditions.

<name1>expression is a construct that may be referenced when the expression matches the input stream and Lex is in start condition **name1.**

BEGIN name1 may appear as part of the 'action' for a rule and has the effect of establishing **name1** as the current state or start condition during parsing.

4. Character set tables.

When bracketed by lines containing only %T, a table of a different character set may be included. This option allows the translation of parts or all of the character set of the input file, and is used when the interpretation of a character or characters has been changed by some I/O routines. The lines of this translation table are of the form:

{integer}{string}

where the character string is transformed into an integer. The following example maps the digits into 30 through 39.

Ex:

```
%T
30   0
31   1
32   2
33   3
34   4
35   5
```

```
36	6
37	7
38	8
39	9
%T
```

5. Internal array sizes. When compiling large Lex programs, often an error message will appear, indicating to the user to "try using %x num, too many x ". The default internal array sizes used by Lex are rather small numbers, so often one must increase the values of several of these array sizes by including statements like those shown in the examples below. A list of names of these sizes follows:

   p - the number of positions
   n - the number of states
   e - the number of tree nodes
   a - the number of transitions
   k - the number of packed character classes
   o - the output array size

Note: The larger the 'num', the longer the compile time.

Ex: %p 20000

Ex: %n 15000

Rules

In the rules section, all rules must have a regular expression followed by an action, where this action may be continued on any number of lines by delimiting it in braces. A regular expression contains textual characters which will match corresponding characters on the input stream and operator characters which specify choices, repetitions, and other features. For example, in the second item of the following list a quote (") is an operator and 'x' is a textual character. The following is a list of templates for forming regular expressions along with their descriptions:

1. x — the character "x"

2. "x" — an "x", even if x is an operator.

3. \x — an "x", even if x is an operator.

4. [xy] — the character x or the character y.

5. [x-z] — any of the characters in lexicographic order between x and z inclusive - (i.e. x, y or z).

6. [¬x] — any character but x.

7. . — any character but newline.

8. ¬x an x at the beginning of a line.

9. <y>x an x when Lex is in start condition y.

10. x$ an x at the end of a line.

11. x? an optional x.

12. x* 0,1,2... instances of x.

13. x+ 1,2,3... instances of x.

14. x|y an x or a y.

15. (x) an x.

16. x/y an x but only if followed by y.

17. {xxx} the translation of xxx from the definitions section.

18. x{m,n} m through n occurrences of x.

Example: The regular expression that follows accepts, in this order: an a at the beginning of a line ; A ; * ; the vertical bar character ; b or B ; c,d,e,C,D, or E ; any character but z ; any character but newline ; an optional f ; 0 or more occurrences of g ; 1 or more h ; i or j ; 0

or more occurrences of d,e,f,D,E, or F ; 1 to 5 occurrences of an r ; k, but only if followed by an l;

```
def  [d-fD-F]*
%%
¬aA"*"\|[bB][c-eC-E][¬z].f?(g)*(h)+i|j{def}r{1,5}(k/l)
          {
          printf("accepts: aA*|bDO-hieeerrkl");
          printf(" , but the true input stream was:");
          printf("%s", yytext);
          }
```

Subroutines

The user may include other subroutines that can be called from within an action. These subroutines are compiled with the rest of the Lex source.

YACC OVERVIEW

There are many instances when the processing of input varies not only at the lexical (token) level, but also at the level of relationships among tokens. This is the level at which Yacc becomes useful and necessary. Yacc (Yet Another Compiler Compiler) [JOHN 75] produces a parser called yyparse() that calls yylex() to retrieve tokens from the input stream.

Return(token) is the function that Lex uses to return a particular token to the parser generated by Yacc. The function yyparse is based on the use of terminals (tokens) and non-terminals, which combine to produce a grammar. This parser is built from grammar rules of the form:

```
A: body;
```

The user may include specific actions to be taken if a certain rule is matched. Actions are written in the programming language C, and may even access specific values within the enclosed rules through a $$ or $n construction. For example, in an action part that could be added to the rule:

```
A : B C D;
```

$1 has the value returned by B, $2 has the value returned by C, $3 has the value returned by D, and $$ receives the value to be returned by A.

The easiest way to write programs using Lex and Yacc is to write a simple C program (call it file.c) that has four lines:

```
#include <stdio.h>
#include <ctype.h>
#include "y.tab.c"
#include "lex.yy.c"
```

This program simply calls yyparse(). We can then compile file.c after calling Lex on file.l and Yacc on file.y, that is:

```
lex file.l
yacc file.y
cc file.c -ll
```

Note: When using the C program method, the Yacc library included by using the computer flags '-ly' is neither required nor desired.

YACC VARIABLES

Yacc has a wide variety of internal variables. Following is a description of the main ones a user might employ in writing grammars to be used with Yacc.

1. yylval - variable that can hold the value associated with a token. When a token is returned from the lexical analyzer produced by Lex to the parser generated by Yacc the value the token has is yylval.

2. error - a reserved token to allow error recovery to occur. This variable is explained further in the section entitled "ERROR HANDLING".

3. yyerrok - a function that resets the parser to its normal processing mode after an error has occurred.

4. yyclearin - a function that clears the lookahead token from the input buffer.

5. yyparse - the function produced by Yacc which repeatedly calls yylex for the next token.

6. yychar - the lookahead token number.

7. yydebug - if set to a nonzero value, this causes yyparse to provide a description of its actions and what input symbols have been read.

GENERAL STRUCTURE OF YACC SOURCE

```
Declarations
%%
Rules
%%
Programs
```

The following three sections describe the format of Yacc declarations, rules, and programs.

Declarations

1. The declarations of variables and #include files must be surrounded by %{ and %} and appear alone on single lines.

   Ex:
```
%{
#include y.tab.c
int x=0;
%}
```

2. The **%Start nonterminal**, construct indicates that the rule beginning with **nonterminal** is the grammar rule to begin the parse. (Default is to use the first rule in the grammar)

   Ex: %start firstrule

3. The **%token terminal** construct indicates that **terminal** may appear as the token name returned by yylex in the return(token) statement. All tokens returned must be declared in this section.

   Ex: %token MINUS

4. Precedence levels and associativity may be declared by using **%prec**, **%left**, **%right**, or **%nonassoc** followed by the list of tokens for which the declaration is made. In arithmetic multiplication has a higher precedence than subtraction. If it is desired that subtraction has the same precedence as multiplication, '%prec MULT' follows the rule containing the subtraction.

   Ex: %left MINUS, DIVIDE

5. **%union** is used to declare a union, and it associates names with each token and nonterminal symbol. These symbols have values that can be referenced by $$ or $n, where n is the nth symbol on the right hand side of the grammar rule.

6. **%type** is used to associate union member names with nonterminals.

The reader is urged to refer to [AHOA 74], [AHOA 75], [AHOA 77], [JOHN 75], [LESK 75] that are listed at the end of this paper for further explanation.

Yacc rules

Tokens are returned from the Lex program in some order. A dictionary entry may have the order of: headword followed by pronunciation, part of speech, definitions, and morphological variants, for example. This order is specified according to a 'grammar'. This grammar is described in the form of rules. A Yacc rule consists of a non-terminal (non-token) on the left hand side of the ':' divider followed by a combination of terminals (tokens) and/or non-terminals (non-tokens). When a rule has been recognized, specific actions may be invoked. These actions may be to return values, modify them, or obtain values of previous actions. In the rule:

```
A : B PLUS C
```

the value of this rule by default is the value of the first element in it, that is, the value of B. The value of the first element in a rule is $1; the value of the second element in a rule is $2, and so on. By adding the action { $$ = $1 + $3 } to this rule,

```
A : B PLUS C
        { $$ = $1 + $3 }
```

the value of the rule becomes the sum of the first and third elements. I will not describe how the parsers produced by Yacc work because this issue is quite complicated. For explanation of this, including conflict

resolution, ambiguity, and precedence, please refer to [AHOA 74], [AHOA 75], [AHOA 77], [JOHN 75], [LESK 75].

Yacc programs

Programs and functions may be included following the Yacc grammar rules section. Functions that are redefined are found in this program section. The function yyerror() that has been described briefly can be defined in this section. The next section describes some of the error handling capabilities and methods.

ERROR HANDLING

The token 'error' is a reserved word to indicating that if an error occurs while within this rule, the input will be flushed. For example:

```
A : error ':'
  | B C
  | D E F
  ;
```

is a rule which implies that if an error occurs within this rule, the input will be flushed to the next colon within the input stream. The processing will resume as if the token 'error' matched the input stream up to the position of that colon. Of course this could be modified to notify the user of the error. This is where the yyerror() function comes

into play. The yyerror() function is called whenever a token that is not expected (does not follow any production rule) is returned from the lexical analyzer produced by Lex. This function may be written to be as elaborate or as simple as desired. If no yyerror() function is written, a default function prints "syntax error" and aborts processing.

CONCLUDING REMARKS CONCERNING LEX AND YACC

Lex and Yacc can be extremely useful and powerful tools if used properly. Both of these utilities are very structured, user friendly, and quite easy to follow and to understand. These tools allow processing of data at the lexical or token level as well as the grammar level. Unfortunately, there is very little documentation to be found concerning these utilities. For a more in-depth look at Lex and Yacc the reader is urged to consult the references of [AHOA 74], [AHOA 75], [AHOA 77], [LESK 75], and [JOHN 75] that are listed at the end of this paper.

THE NINE PASS APPROACH

Initially, it was my intention that after the dictionary had completed one pass it would yield my final result -- a relational lexicon. As I became more and more familiar with this dictionary, I reached a point where I decided that I could not conceivably do this in a single pass. As I added passes to change some things, occasionally things that I did not want changed had been modified. Ultimately, I needed a total of nine passes to produce the lexicon in its present form. I presume that some of these passes could have been combined and modified so that there were fewer than nine passes. These nine passes in their present state are easy enough to understand, for those who are interested. The next nine sections describe each of the nine passes in order. A thorough understanding of Lex and Yacc is required to comprehend the nine passes. Following is a table of commonly found codes and their respective translations.

Note: Often relations are mentioned without including the prefix c_

CED Text	Result
#h	c_HEADWORD
#H	c_HEADWORD
@m?-#1	c_MORPH
@m#1?-	c_MORPH
@m&-#1	c_MORPH
@m#1&-	c_MORPH
@m#5?-#1	c_MORPH
@m#5&-#1	c_MORPH
@m#5#1?-	c_MORPH
@m#5#1&-	c_MORPH
#5@m&-#1	c_MORPH
#5@m?-#1	c_MORPH
#5@m#1?-	c_MORPH
#5@m#1&-	c_MORPH
@m#1?-#6	c_POS
@m#1&eqv.#6	c_POS
@n#6	c_POS
@f#6	c_POS
#6	c_POS
@m#1&eqv.#1$D. #6	c_DEF
@n#1$D. #5	c_DEF
#1$D.	c_DEF
@n$D.	c_DEF

CED Text	Result
@n#5	c_DEF
#5	c_DEF, c_ALSO_CALLED
: #6	c_SAMP
@n#1$D. #6	c_CATEGORY
#6	c_CATEGORY
#+	syllabification break
#!	syllabification break
@.	syllabification break
?%	degrees
@t	comma
#5(	pronunciation
@n#5[	etymology
@m#5[	etymology
@n[	etymology
@m[	etymology
@n	subdefinition, population, c_ALSO_CALLED
#1	subdefinition, italicized word, c_NLAST
@=	minus
@t#m@t	times
@t#M@t	times
@t#+@t	plus
@t=@t	equals
#5 %	homograph number
?&	ampersand

CED Text	Result
*'	the accent grave
&!	exclamation mark
*%	Swedish A

Table 3. Table of Common Translations

PASS1

This pass is clearly the most difficult to follow, in addition to being the pass which produces the most results. It incorporates the use of a grammar, written in Yacc, as well as many lines of Lex code. I will not explain the usage of large numbers of flags within pass1, but will nonetheless try to explain pass1 with as much clarity as possible.

First of all, there are four variables that were declared globally to be accessed by the yyerror() function within Yacc. The variable byt_offset is the offset (in bytes) within the input stream that is the present location. The variable stor_frst_byt is the position of the beginning of the headword entry that is currently being processed. The position of the end of the headword entry is found in the stor_last_byt variable, and the headword itself is stored in the array called wordarry.

1-32 Most of the first thirty-two rules in the Lex program of pass1 match some of the many typesetting codes to produce special characters.

- The second rule eliminates all etymologies.

- Occasionally, a cross-reference is separated from other entry text by braces. The third rule accepts all text from a left curly brace to its corresponding right brace and writes it out. This text will be processed in further passes.

- The # when followed by a number signals a change in font, and when the font changes within a definition, for example, these font characters should not be visible. Occasionally words or word phrases are italicized within definitions and these words will be printed as will all other text.

- @t#8 followed by a number (0001, 0002, ...) refers the typesetting machine to a table of special characters (unprintable?), the pattern of which should not be printed.

33 #H or #h signals a new headword, and A1 is the token that is returned to the parser generated by Yacc to mean either of these codes.

34 b20, b21, and b22 are the three different syllabification codes. These codes also occur (and are ignored) within morphological variants, plural endings, a person's first name, and in the cross-reference 'also called:'. The minus token is returned to the parser generated by Yacc and the underscore is printed.

35 '#3' is found immediately preceding a variant spelling, and in this rule the token A3 is returned.

36-44 The next nine rules recognize up to nine homograph numbers of a headword. The number nine was chosen to be larger than any occurring headword group, and the results were consistent with this assumption. A single quote was inserted between the headword and the homograph number to separate the headword text from the homograph number. A token named NUMB is returned to the parser generated by Yacc to indicate the presence of this homograph number.

45-46 The next rule when given the regular expression '#5)' simply prints out the right parenthesis. A left parenthesis under font five needs to close a right parenthesis within font five. The GARBAGE token is simply unused information that is not printed out but which can be found within a definition. An example is the pronunciation of a headword that is occasionally found at the sense or sub-sense level.

47-50 The next four regular expressions are commonly delimited by a headword and a variant spelling or two different morphological variants. The strings 'or sometimes before a vowel', 'or before a vowel', 'or', 'or esp. U.S.', 'or esp.', 'or U.S.', 'or U.S. fem.', 'or fem.', 'or U.S. sometimes', 'or sometimes', when found in these contexts, results in the printing of 'or'.

51-87 The next thirty-seven rules match all of the parts of speech, and many combinations of parts of speech. The actions are to print the part of speech out in the form: **POS adj.**, for example. Any further combinations would be recognized in future passes.

88 The expression ': #6' signals the end of a definition and the beginning of a sample usage, so a period is printed to end the definition.

90-91 The next two rules recognize the introduction of singular and plural forms and returns the PL token to the parser generated by Yacc.

92 The '#6' regular expression that follows would normally signal a part of speech, but it often signals a category. If no match is made in the 37 parts of speech remarked about above, a category is defaulted to (given certain conditions).

93-94 The next rule eliminates most pronunciations. If a pronunciation follows a headword this pronunciation is ignored. If it is the pronunciation of a plural, title, first name, etc., it will also be ignored. If this regular expression is matched within a definition the text is printed. Sometimes the '#5' does not precede the left parenthesis of a pronunciation. In the cases where a headword has a homograph number, the pronunciation will follow without the '#5' to signal it.

95-96 The next two expressions 'b10#5' and 'b10' simply toggle certain flags signalling the beginning of a new definition in one case and printing a blank in the latter.

97-102 The six rules to follow will recognize subdefinitions, and notify Yacc of the occurrence of this by returning LETT. I did not expect (from inspection of the files) there to be many occurrences of subdefinitions, so I imposed the restriction of the maximum subdefinition letter (subsense) to the letter f. Table 11 on page 104 is a table of occurrences of these subsenses, showing only three occurrences of the subdefinition f.

103-104 'also called:' and 'official name:' are recognized in the next two rules. These eventually become the basis for the ALSO_CALLED relation.

105 The font number #1 commonly signals a first name, but in the cases where #1 is found in a definition or a category, this is not true. If first1_flg is turned on (only by previously having a noun as its part of speech) then we expect to see a first name (to become the NLAST relation) next in the input stream.

106-107 The next two regular expressions signal the beginning of a new definition, so flags are appropriately toggled.

108 After these two rules we have a regular expression that matches the introduction of a morphological variant.

110 Finally I will explain the rule associated with the definition of alpha. This variable is a set of all possible characters that could comprise a headword.

- First of all, if the text is a first name, print it and return FIRST.

- Otherwise, if it is a c_ALSO, print that.

- If the text is a morphological variant, return MORPH.

- If it is a headword (where numh_flg equals one), we want to store the headword in an array called wordarry.

- If it is a sample usage, acknowledge that fact by printing the text and SAMP if it is a new sample usage.

- If it is an inflection print the ending and return END to the parser generated by Yacc.

- If it is a definition, print the text and return DEF to the parser generated by Yacc.

- Finally, if it is a category, print CATEGORY and the text and return CATEGORY to the parser generated by Yacc.

114 If there are any other characters that do not match any of these previous regular expressions (newline characters were removed), the characters are ignored. This is evident in the last rule of pass1 where the period (.) matches any character but the newline and the action does not include printing any text.

Yacc Grammar

The other half of pass1 is the Yacc program. There is not a lot of detail to be reviewed concerning this grammar, but I will now proceed to discuss the origins of the Yacc program of pass1.

Initially I had only an outdated version of the CED, and with this I wrote a regular expression of a dictionary entry in the Collins English Dic-

tionary. Based upon this regular expression I wrote a grammar for Yacc to handle. The grammar, which is slightly modified for the Lex and Yacc interaction, is written according to the Yacc syntax and can be found in Appendix A.

Explanation of Yacc Grammar

The first two rules recursively define a dictionary beginning with the 'A1' token (the #h or #H tokens) and continuing with the rest of the dictionary entry 'restdic'. The third rule is provided in the event of the occurrence of an error, in which case the input is flushed to the beginning of 'restdic'. SYLL is printed each time any of these rules are applied so that at the end of an entry we will prepare for the next headword. The rule beginning with 'restdic' indicates headwords followed by parts of speech followed by definitions, and finally morphological variants.

Skipping down to the rule beginning with 'hdwds', 'hdwds' is made up of a 'word' followed by more 'word's. The 'word' is comprised of syllables linked by MINUSes, or followed by more syllables. This is evident in the rules beginning with 'word'. Each syllable is printed in the action of the rule beginning with 'syllab', since it was not printed in the Lex program of pass1. 'morwds' recursively calls itelf, or can be empty as the first rule indicates. 'mwds' may be either a 'word' itself, an 'A3' (#3) followed by a 'word', an OR followed by 'A3' and a 'word', a cate-

gory, a homograph number, a homograph number followed by pronunciation, or just the pronunciation.

The 'ptspch' rules can accept a part of speech (POS) followed by 'en' and 'mcat', or a category followed by a part of speech, 'en', and 'mcat'. 'mcat' is an optional category. 'en' is the empty string, or an optional plural (PL) followed by 'een' and 'moren', or 'fir' optionally followed by a pronunciation (APRON). 'een' is one or more endings (END). 'moren' is the token OR followed by 'een' and an optional pronunciation, or a pronunciation followed by 'moren'. 'fir' is one or more first name tokens (FIRST).

The definitions section begins with the rule that has 'defins' on the left hand side, which may be the empty string, or a part of speech and recurring (calling itself- 'defins'), or 'garb' followed by 'defpart', then 'sampart' and recurring. 'garb' is the empty string, or garbage and recurring, or a CATEGORY token and recurring, or an optional category followed by a LETT token and recurring. 'defpart' is a definition (DEF) followed by 'garb', and optionally recurring. 'sampart' is a sample usage token (SAMP) followed by 'garb' and optionally recurring.

The morphological variant section begins with 'morphs' that may be the empty string, or 'morm' followed by 'mormor' and recurring, or a part of speech token followed by 'morm' and recurring. 'morm' is the MORPH token optionally recurring. 'mormor' can be the empty string, a pronunciation, part of speech, category, or definition, all possibly recurring.

The output of my first pass made me aware of a key problem: there were several entries that did not follow my grammar. It was then that I decided to skip over these words and go to the next headword. This is where the yyerror() function of Yacc fits in. When an error occurs (that is, something returned from the lexical analyzer produced by Lex that is not expected by Yacc) the yyerror function is called.

In the following yyerror function, the input is flushed to the next headword (signalled by #H or #h). The routine also writes to a file (call it badfil) the following information:

1. The byte offset within the dictionary of the beginning of the headword.

2. The byte offset within the dictionary of the end of the headword.

3. The approximate position within the entry where the error occurred.

4. The headword itself. (wordarry)

```
yyerror()
{
fprintf(stderr, "%u", stor_frst_byt);   /* beginning of headword */
```

```
fprintf(stderr, " ");
spend = 0; /* flag- have not reached next headword */
temp_pos = byt_offset - stor_frst_byt;  /* pos. within entry of err */
stor_last_byt = byt_offset;  /* position within file of error */
while (spend ==0){  /* flush to next #h or #H */
  c = getchar();    /* get next character */
  stor_last_byt = stor_last_byt + 1; /* incr. position in file */
  cc = getchar();    /* get next character */
  stor_last_byt = stor_last_byt + 1; /* incr. position in file */
  if ((c == '#')&&((cc == 'H')||(cc == 'h'))) spend = 1; /* #h or #H */
  else if (cc == '#') {    /* if cc is # next char may be h or H */
     ccc = getchar();    /* get next character */
     stor_last_byt = stor_last_byt + 1;  /* incr. pos. in file */
     if ((ccc == 'H')||(ccc == 'h')) spend = 1; }  /* quit if h or H */
  }
byt_offset = stor_last_byt;  /* present position stored */
stor_last_byt = stor_last_byt - 2;  /* pos. of next entry is bef. #h */
fprintf(stderr, "%u", stor_last_byt);  /* end of headword */
fprintf(stderr, "%u", (temp_pos));  /* pos. of err */
fprintf(stderr, "%s\n", wordarry);  /* headword of error */
}
```

PASS2

As an overview, the second pass handles formatting cross-reference homograph numbers and sense numbers; it also takes care of doubling single

quotes that occur within parentheses. This pass creates the c_RELADJ, c_ABBREV, c_USAGE, and c_COMPARE relations, and it prepares the c_DEF relation to include definitions beginning with a left parenthesis. An in depth look at the second pass follows:

1-30 The first thirty rules print special characters.

31 If two single quotes are seen, they are printed out. If a single quote is on the input stream and within parentheses, two single quotes are printed. Only when outside of parentheses does pass1 print two single quotes when one single quote is on the input stream.

32 When a single quote is to be printed, two single quotes are printed for MU-PROLOG syntax compatibility.

34-42 The nine rules to follow recognize homograph numbers for cross-references, while within parentheses.

43 The next rule skips over the first of a list of senses, and only prints the last sense.

44 The following rule turns some flags off and may print the right parenthesis.

45-73 The next twenty-nine rules (it probably should have been fifty) default the homograph number to one, if one is not supplied, and print the sense number. It has a homograph number supplied if it has matched one of the nine previous rules that refer to homograph numbers.

74-75 The next two rules eliminate etymologies.

76-77 The following two rules eliminate braces.

78-81 These four rules indicate the end of sample usages and simply print a period followed by the new line character.

82 The next rule will print a new line character before the left parenthesis if we have seen #5(#6 garbage #5).

83-85 The next three rules will function similarly except that since a period is in this regular expression, the text that will follow the left parenthesis will become part of the following definition.

86-97 The following twelve rules will produce subdefinitions if they have not previously been produced.

98 The next rule produces the c_RELADJ relation. The senses of the related adjective will be taken care of by the twenty-nine "sense" rules previously mentioned.

99 The c_ABBREV relation is produced by this rule.

100-101 The abbreviation part of speech is recognized in these two rules.

102 The usage relation is produced using the next rule.

103-109 These seven rules determine the c_COMPARE relation from **see**, **see also**, or **compare**.

110-113 Verb transitive (vt) and intransitive (vi) are produced by the next four rules.

114-116 Adjective postpositive (adjp) is recognized by these three rules.

117-118 The next two rules skip over any pronunciations left behind.

119-121 These three rules recognize an adjective within parentheses, the first rule recognizing a sample to follow.

122-125 These rules recognize sentence modifier text that is found within parentheses.

126-127 The next two rules ignore all previously unmatched text within parentheses. The text that this ignores is syntactical and collocational information that could not always be separated from one definition and placed in its appropriate relation.

128 This rule recognizes DEF and sets some flags.

129 The second-to-last rule prints the syllabification of the word two times. The reason for this is to produce a headword in the next pass which will remove the underscore syllable markers.

130 Occasionally a plural was followed by a category or definition. The final rule will split these apart by printing the plural form on one line, and following it with a new line character to leave the category or definition on a line of its own.

PASS3

This pass yields headwords without syllabification marks and separates lists of items in a relation (more than one plural ending for example) into more than one relation. It also extracts categories from the definition text. It does some basic cleaning up of definitions, exclamation marks, and syllabifications. Finally, it converts one of the SYLL relations into a variant spelling by producing the OWORD (other word) relation.

2-4 The second, third, and fourth rules of pass3 insert the temporary word DDEF whenever a new line begins with a small letter, a capital letter followed by a small letter, or a number.

5-30 The next twenty-six rules change a letter followed by an ampersand to that letter followed by an exclamation mark. The reason for this change is that in pass1 the pattern that signalled an exclamation mark was the same as that of an ampersand. However, all ampersands are surrounded by blanks, so it is truly the exclamation mark which is identified in these cases.

31 The next rule will produce a headword without syllabification on the first occurence of SYLL. On the second occurrence, the syllabification marks will remain. The headword was originally stored in syllabified form, and duplicated. The syllabification marks will be removed from the headword but will remain to produce the c_SYLL relation.

32 The next rule cleans up any case in which the relation is neither a COMPARE nor a HEADWORD and will just print out the comma.

33 Double quotes are produced in the next rule for instances of single quotes (for example, in headwords).

34-35 The next two rules separate items in a list into individual relations. For example, **PL es, s** translates into the two relations **PL es** and **PL s.**

36 The plurals always end in a period, so the next rule eliminates this period.

37 The homograph numbers are removed from the SYLL relation in the next rule.

38-39 Anything in parentheses on the same line as a plural is removed by the next two rules.

40 In this rule, any time a left parenthesis is found at the beginning of a new line, the DEF relation is produced.

41-42 If the right parenthesis is followed on a new line with DEF, the DEF and new line is removed and only the right parenthesis is printed. This rule and the one preceding produce the DEF relation which begins with a parenthesis.

43-47 The following five rules change flags for the purpose of converting the lists of plurals, morphological variants, etc. to individual relations.

48 The next rule prints HDWORD for the first occurrence of SYLL and prints SYLL for the second. It is within the first occurrence of SYLL that the syllabification separators are removed.

49 ALSO was occasionally found to be in the wrong position, sometimes followed by another relation name. This is corrected in this pass and the next.

50 The word 'usage' is sometimes found alone on a line, and this line is removed by the action of this rule.

51-53 The next three rules help to take care of the COMPARE relation by separating it into two relations when necessary.

54 The FIRST relation is separated from any other relations in this next rule.

55-67 The rest of the rules attempt to separate categories from definitions. Before pass3, many definitions begin with categories. The final two rules of this group selectively insert a D before the word CATEGORY. This D indicates in successive passes that the word DCATEGORY should be deleted, that is, it is not really a category.

PASS4

Pass4 will separate other lists of relations (as in pass3) that were not previously in correct form. Definitions are also prefixed with the DEF relation name. Subdefinitions are put in order, and cross-referenced headword homograph numbers are defaulted to 1. When a cross-reference is made to a word with only one homograph number, the authors of the CED opted not to include the homograph number. More cleaning up is done to the categories.

1-3 The first three rules (as in pass3) insert the word DDEF on lines that begin with a number, a small letter, or a capital letter followed by a small letter.

4-5 The next two rules split up the lists following: MORPH, ABBREV, CATEGORY, ALSO, and POS.

6-8 Some general cleaning up is done by the next three rules.

9 The word DEF on a single line is removed.

10-12 The next two rules turn off some flags to end the lists described above. When DDEF is seen followed by a new line and definition, the new line is removed and DEF is printed.

13-19 The next seven rules organize subdefinitions.

20-21 The first of these rules turns flags on and off for the SYLL relation. Headwords are defaulted to homograph number one in the following rule.

22-24 These three rules turn certain flags on or off.

25-26 Cleaning up is the purpose of the next rule, and the rule following that will print a ',1 to indicate a homograph number of 1. The next pass will delete this if a homograph number already exists.

27-29 The following three rules turn on and off certain flags.

30-34 The last five rules take care of the CATEGORY relation. If DCATEGORY is recognized and it is time for a definition, DEF is printed. If the part of speech of a morphological variant is appropriate, POS is printed.

PASS5

This pass helps to clean up homograph numbers and sense numbers. Additionally, it prints a quote before the headword and puts the headword relation into correct syntax. The syllabification is also cleaned up somewhat.

1-2 The first two rules print special characters.

3-5 These three rules again print DDEF before the text.

6-8 The next three rules are a bit complicated. In each of them, if within a syllabification, it is necessary to remove the homograph numbers. If within a headword a homograph number is seen, the first one recognized should be kept. In the previous pass when the homograph number was defaulted to 1, this text was appended to the end whether a homograph number was present or not. If a headword already has a homograph number, the second one is deleted. If within a COMPARE relation without the blank in the expression, a quote should be appended after the word COMPARE; otherwise, print nothing. If there is a blank in this expression (a true homograph number), and the position is within the COMPARE relation, the homograph number should be printed.

9-10 These next two rules turn off some flags, with one of these actions printing DEF.

11-12 These rules flush the input up to the first comma within a syllabification. This will be used to eliminate the homograph numbers of a syllabification.

13 The next rule prints a single quote before each headword.

14-22 The last nine rules do some general cleaning up that occasionally is needed. Flags are also changed within many of these rules.

PASS6

Pass6 is the final pass before putting the relations into true MU-PROLOG form. It tidies up many things such as eliminating blanks occurring at the end of a headword, removing periods and blanks that occur in cross-references before and after single quotes. It also removes periods and commas from the end of some relations.

1 Any DDEF's that are left are changed by the first rule to the DEF relation.

2-3 These next two rules turn flags on (and others off).

4 In the following rule, if within a COMPARE and a period is followed by an optional blank and end-quote, the period and blank are removed.

5 If a blank precedes an end-quote in a headword, that blank is removed.

6-7 These two rules remove other blanks before and after end-quotes.

8-9 These rules are simply for the purpose of setting flags.

10 The following rule removes a comma (if one occurs) from the end of a relation; if a comma is within an abbreviation, a period is printed.

11 The next to last rule removes any blanks from the ends of lines.

12 Finally, in the relations ABBREVIATION, ALSO_CALLED, and CATEGORY the period is left at the end of the relation. This action is taken because all three of these relations commonly have abbreviations within their text. The period is removed from any other relations.

PASS7

Pass7 is a rather lengthy pass that takes relations of the form: **RELATION text** and produces relations in the form of MU-PROLOG facts. Each relation is prefixed with c_ to characterize that relation as being obtained from the CED. This pass could easily be modified and used for processing another dictionary.

1 When a new headword is recognized, all counts become zero and all arrays are cleared.

2-23 The next twenty-two rules simply set flags based on the relation encountered. Some rules keep counts of the occurrences (of the

given relation) within the headword. None of the original CED text is recognized in these rules.

24 Since each relation that has been produced lies on a single line, the regular expression [¬\n]+ recognizes all text up to and not including the new line character.

- If within a headword, store the headword in an array (wordarry), and print out the c_HEADWORD relation.

  Note: Wordarry contains the headword and homograph number.

- If within a variant spelling (OWORD), print out the headword and the variant spellings based on the number of occurrences (o_ct) of OWORD at a given point. Store this variant spelling in an array.

- The syllabification of the headword is printed. If there are variant spellings of the headword, the c_VAR_SYLL relation is created.

- The c_COMPARE relation is produced containing the headword with its homograph number; the posnum, which is the syntax level; and the defnum, which is the sense level (if that level is zero, assume it indicates sense one).

- The c_PAST, c_SINGULAR, c_ALSO_CALLED, c_RELADJ, c_NLAST, c_ABBREV, and c_CATEGORY relations are produced similarly. These relations are found at the sense level, and so are produced in this context.

- The c_USAGE and c_PLURAL relations operate at the headword level, and so are generated at that level.

- There are two cases for a part of speech. The first case is the part of speech of a morphological variant. In this instance we write the headword, the array holding the morphological variant (see next rule), and the part of speech. If the given part of speech relates to more than one morphological variant (mm_ct > 1), the part of speech is printed along with each consecutive morphological variant. The second case is when the part of speech relates to the headword; the c_POS relation is produced in this instance.

- Definitions are recognized in the next statement. If no part of speech is given for the headword, the number corresponding to the syntax level is set to one. If the definition is actually a subdefinition, (sub_flg = 1), then the subsense number (subdefnum) is appended and the text printed.

- Otherwise, if it is a sample usage, the text is printed accordingly. It may be a sample usage of a sub-sense, so if sub_flg = 1 the sub-sense number is appended.

25 The final rule double checks to see that if anything does not match up (other than new lines) it will be printed to standard error. It is worth noting that when the entire dictionary was processed, no text was ever printed here.

PASS8

Pass8 divides definitions and usage notes into blocks of 80 characters or less. MU-PROLOG requires that all relations be less than 160 characters in length. Eighty characters of text, in addition to the text of the relation name and the text of the headword, should not exceed this 160 character limit of a rule. Another reason for the 80 character blocks is because of the standard 80 character-width screen. This pass also sets up the c_DEF_NUM and c_USAGE_NUM relations by printing the pseudo-relation NUM which is followed by DEF or USAGE.

1 The first rule recognizes the c_DEF relation. This relation as well as the c_USAGE relation has the textual matter enclosed in square brackets. The entire relation name through the second left square bracket is stored in begdefarry.

Note: This array has the headword and sense numbers.

2 The second rule similarly stores the c_USAGE text as well as the headword text in the array called begusearry.

3 The next rule turns off the usage and definition flags.

4 The next to last rule recognizes the end of the c_DEF and the c_USAGE relations.

- If this pattern is within the c_DEF relation, print the begdefarry. If n_first flag is one it prints a single quote at the beginning of the second and successive relations. The defarry (definition text) is printed followed by the number of the block. NUM is printed at the end to be followed by c_DEF and the headword and the number of blocks. This will become the c_DEF_NUM relation in the final pass.

- If this pattern is within the c_USAGE relation, print the begusearry. The rest of this rule is defined just as in the c_DEF relation described above.

5 The final rule recognizes non-blank and non-newline characters. If the definition (respective, usage note) length (found by adding the length of the current word to the counter def_ct (usage_ct)) is greater than 80 characters, a new relation instance is created. The text of the current word is then added to the array.

PASS9

This final pass cleans up the c_DEF_NUM and c_USAGE_NUM relations. It also removes square brackets from the c_DEF, c_SAMP, c_USAGE, c_DEF_NUM, and the c_USAGE_NUM relations.

1-2 Rules one and two simply rewrite the c_DEF_NUM and c_USAGE_NUM relations by eliminating the newline character in the middle of the expression.

3 The third rule eliminates the second left square bracket from the c_DEF, c_SAMP, c_USAGE, c_DEF_NUM, and the c_USAGE_NUM relations.

4 This rule removes the final right square bracket from these relations.

5 The second right square bracket is removed from the c_DEF, c_SAMP, c_USAGE relations.

6-9 The final four rules simply turn flags on or off for the previous rules.

DICTIONARY PROCESSING APPROACH

All nine passes were created and were called pass1, pass2, ... pass9. The first step in the actual processing of the Collins English Dictionary was to translate (using tr) all of the newline characters to blanks. This was decided upon in order to simplify the nine pass approach. The regular expressions would not need to recognize the newline characters within each expression. The dictionary, being a file of over 21 megabytes, was too large to process as a whole. I wrote a small lex program to split the dictionary approximately into thirds - part1 being A-F, part2 being G-O, and part3 being P-Z. There were four entries that were too long - go, run, take, and time. I could not determine the reason for the difficulty with these words, so I just removed them from the dictionary. These four words would be cleaned up by hand at a later time. Figure 2 on page 80 shows the basic approach that was taken with all three parts of the dictionary.

Part1 was directed into pass1 to create the file of words that caused yyerror to be called. This is badfil.11, and is used in conjunction with part1 and the program 'reprint' to create the new files part1.good1 that contains all entries that can be processed, and part1.bad1 that contains all entries that cannot be processed. The program reprint.c, written in the C programming language, takes badfil as a reference file and the dictionary as input, and produce the entire entries of each headword in badfil. This would then be cleaned up at a later date. Part1.good1 was

directed into pass1 to create another file of words that caused yyerror to be called. The reason for this pass is that when yyerror is called, it flushes to the next headword, that is, past a #H or #h which signal new headwords. Occasionally, there may be a problem between the #H or #h signal and the headword. Badfil.12 contained a few entries, most of them with a problem between the signal and the headword. Badfil.12 was used with part1.good1 and reprint to create part1.good2 and part1.bad2. Just for safety's sake, I ran pass1 a third time to be certain badfil.13 was empty. It was empty so I could send part1.good3 through the rest of the passes.

It was my first intention that with badfil, I could recover any words that were necessary elements of the lexicon, and process them by hand. Only about 3.5% of the entries in the CED caused yyerror to be called. This still amounted to over 3000 headwords that were missing.

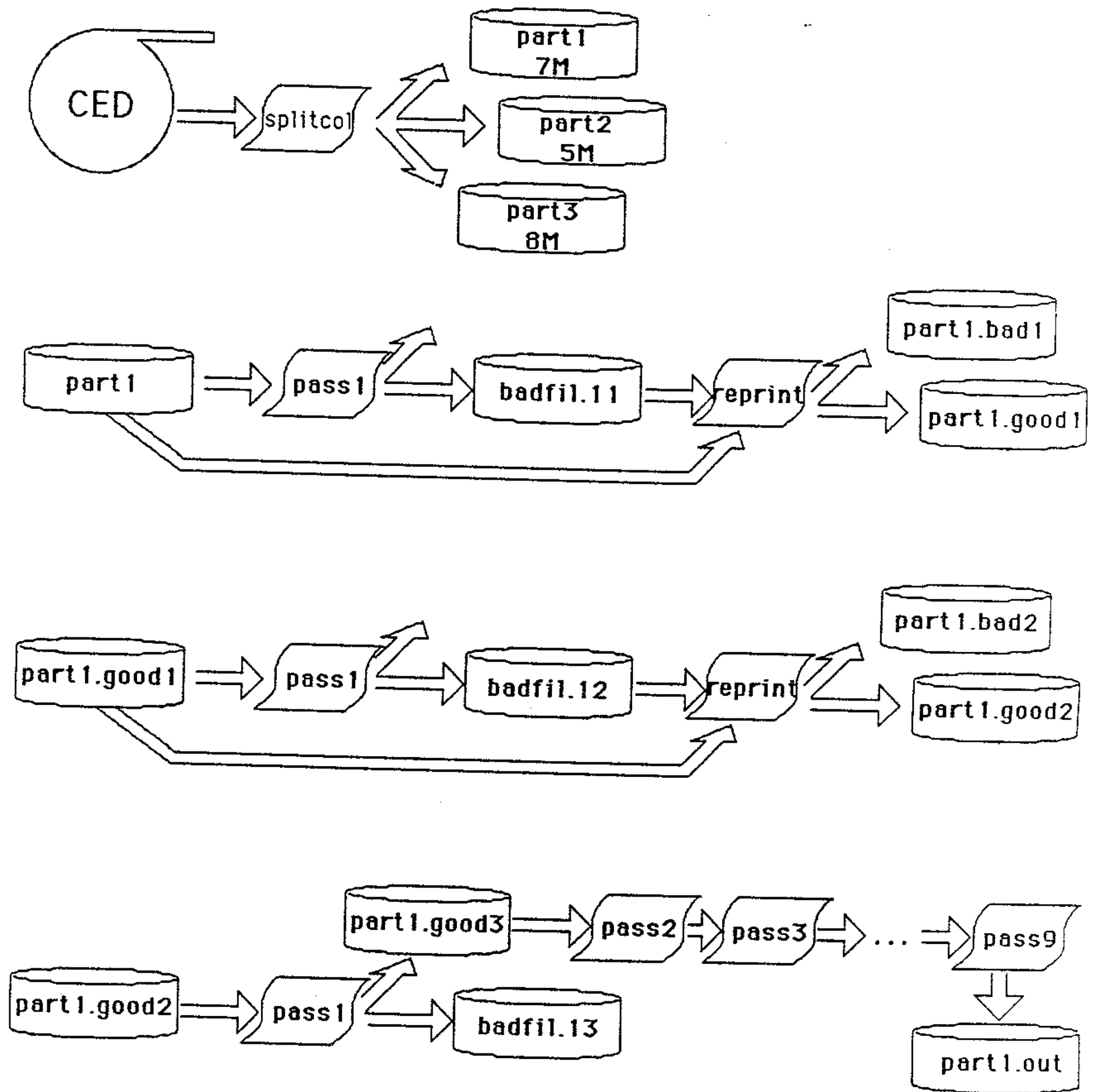

Figure 2. Dictionary Processing Approach

CLEANING UP

The following is a table of the number of words in all badfil's.:

1013	in part1, 1st pass.
10	in part1, 2nd pass.
807	in part2, 1st pass.
5	in part2, 2nd pass.
1203	in part3, 1st pass.
14	in part3, 2nd pass.
3052	total.

Table 4. Table of Non-parseable Entries

There were a total of 3052 entries that could not be handled by my first pass.

These words, along with the four previous (go, run, take, time) would manually be put into the form of the output of pass1. From this form they would be sent through the eight other passes, cleaned up further, and become part of the lexicon.

The following is a sample list of some of the words that have entries which need to be cleaned up further. These are words that are found in the lexicon.

'bone up'

'Checheno-Ingush Autonomous Soviet Socialist Republic'

'International Bank for Reconstruction and Development'

'Kabardino-Balkar Autonomous Soviet Socialist Republic'

'mute',2

'North Ossetian Autonomous Soviet Socialist Republic'

'round off'

'gaffer'

Using the **awk** utility I made frequency counts for all of the parts of speech and all of the categories. There were 263 parts of speech that were obviously erroneous, so the **awk** program printed the headword associated with the errors. I manually corrected these errors by changing the output of pass1, to allow the future passes to respond in an appropriate manner. Similarly there were 1420 categories that were in error that I had to correct by hand. This is a large number of errors to correct manually, but after spending a large amount of time trying to correct these problems in pass1 and making little progress, I decided that the manual route was appropriate. The reason for the errors in the categories is that it was very difficult (using Lex and Yacc) to determine what is a category, since there was little consistency in the syntax. Sometimes these categories began with a capital letter; often they did not. Sometimes they were abbreviated, other times they were not, and still other times they were abbreviated differently. Table 6 on page 88 is a list of categories that occurred more than 30 times in the CED. It was often

difficult to determine when a given part of speech ended and another entity, such as a definition, began. An exhaustive list of all possible parts of speech would have been an extremely helpful item previous to the implementation of my project. Table 8 on page 94 is a list of all of the parts of speech that were found in the Collins English Dictionary.

STATISTICS AND OBSERVATIONS

The purpose of this section is to compare some of the results of my findings with the results of previous efforts. In 1980 Amsler published his findings regarding the **Merriam-Webster's Pocket Dictionary** (MWPD) [AMSL 80]. His work involved finding kernel words within dictionary definitions, and included frequencies of parts of speech. Peterson analyzed the **Webster's Seventh** (W7) dictionary and reported his findings in [PETE 82]. Both of these results are compared with the results of analysis of the CED, and observations are made concerning similarities and differences between the dictionaries, and possible reasons for these conclusions. Note that data from the MWPD was not readily available for all comparisons.

FREQUENCIES OF RELATIONS IN THE CED.

The following table lists the relations that were found in the CED, and corresponding relations observed in the W7.

	CED			W7		
ITEM	FREQ	FREQ/HD	% USAGE	FREQ	FREQ/HD	% USAGE
DEFINITIONS	161693	1.98	36.8	140500	2.04	45.0
PARTS OF SPEECH	96218	1.18	21.9	67809	0.99	21.7
HEADWORDS	81561	1.00	18.6	68766	1.00	22.0
MORPHOLOGICAL VARIANTS	27219	0.33	6.2	9957	0.14	3.2
CATEGORIES	26765	0.33	6.1	11990	0.17	3.8
SAMPLE USAGES	16779	0.21	3.8	-	-	-
PLURALS	9366	0.11	2.1	6320	0.09	2.0
COMPARISONS	7301	0.09	1.7	4025	0.06	1.3
ALTERNATIVE NAMES	4562	0.06	1.0	572	0.01	0.2
VARIANT SPELLINGS	4540	0.06	1.0	2460	0.04	0.8
FIRST NAMES	2445	0.03	0.6	-	-	-
ABBREVIATIONS	525	0.01	0.1	-	-	-
RELATED ADJECTIVES	141	*	x	-	-	-
USAGE NOTES	84	*	x	-	-	-
SINGULAR FORMS	43	*	x	-	-	-
PAST FORMS	7	*	x	-	-	-
total	439494	5.39	100.0	312399	4.54	100.0

- no occurrences
x less than .05
* less than .005

Table 5. Comparative Frequencies of Items in the CED

The heading 'CED freq' refers to the number of times the given relation occurred in the CED. 'CED freq/hd' reflects the average number of times the given relation is found in each headword. The heading 'CED % usage' is the relative frequency of a relation compared with all other relations. The columns are likewise described for the results of the **Webster's Seventh** (W7). See [PETE 82] for explanation of W7 data.

It is interesting to note that there are on the average almost two definitions for each headword in the CED, whereas the W7 contains just over two definitions per headword. 45% of the relations in the W7 are definitions, but only 36.8% of the relations are definitions in the CED.

The CED has a larger number of parts of speech per headword than the W7. It is also interesting to note that the W7 has a smaller number of parts of speech than it has headwords. Apparently the W7 does not insist on a part of speech for each headword, and few headwords have more than one part of speech; instead, a new headword is created.

The CED has an appreciably larger number of headwords than the W7. There are over twice as many morphological variants per headword in the CED than in the W7. It appears that the CED goes into much more detail concerning morphological variants than the W7.

The CATEGORY relation also provides an interesting comparison. Almost twice as many categories per headword are found in the CED. This relation will be very valuable in a document retrieval environment.

The CED also includes sample usages, approximately 1 for every 5 headwords. At first this appears to be a small ratio, but if one excludes the large number of proper nouns that are in this dictionary, this ratio may approach one sample usage for every 4 headwords. The W7 does not include example usages of the headwords.

The rest of the list provides no surprising results until the totals are examined. The CED provides on the average about five and a half relations per headword compared to about four and a half in the W7. It appears (but this observation may be unfounded) that the CED may cover dictionary entries slightly more thoroughly than the W7. It appears, at least, to be a more than adequate dictionary.

FREQUENCIES OF CATEGORIES

CATEGORY	FREQ	CATEGORY	FREQ
Informal	2457	Linguistics	93
Archaic	1531	History	90
Brit.	1162	Property law	90
U. S.	1056	Bible	86
Slang	796	Statistics	83
Rare	780	Meteorol.	78
Chiefly U. S.	757	Judaism	76
Law	717	Brit. dialect	75
Music	596	Commerce	70
Obsolete	550	Psychiatry	70
Nautical	547	Finance	67
Chiefly Brit.	488	New Testament	67
Physics	457	Chess	65
Maths.	411	Economics	63
Pathol.	408	Geom.	62
Anatomy	379	Formal	61
Austral.	369	Now rare	60
Biology	354	Bridge	59
Chem.	348	Golf	58
Botany	345	Canadian	56
Greek myth.	296	Derogatory	56
Med.	276	Surgery	56

CATEGORY	FREQ	CATEGORY	FREQ
French	254	Angling	53
Grammar	226	Tennis	50
Brit. informal	221	Ecology	49
Printing	218	Genetics	48
Psychol.	214	Theatre	48
Military	199	Psychoanal.	47
Old Testament	186	Cards	46
U.S. slang	186	Taboo slang	46
R.C. Church	182	Astrology	43
Poetic	177	Criminal law	43
Austral. informal	175	Films	43
U.S. informal	175	Hinduism	43
Zoology	175	Metallurgy	43
Brit. slang	172	Chiefly R.C. Church	42
Logic	172	Chiefly Scot.	42
Literary	164	German	42
Phonetics	161	Mining	41
Philosophy	160	Vet. science	40
dialect	159	Rhetoric	38
Trademark	154	Billiards	37
Austral. slang	151	Norse myth.	37
Architecture	146	Archaeology	36
Electronics	144	Boxing	36

CATEGORY	FREQ	CATEGORY	FREQ
Scot.	144	Chiefly Austral.	36
Christianity	143	Caribbean	35
Cricket	141	Baseball	34
Astronomy	131	English history	34
Physiol.	127	S. African	34
Latin	121	Soccer	34
Geology	120	Stock Exchange	34
Prosody	120	Not standard	33
Sport	120	Classical myth.	32
Computer technol.	107	Rugby	32
Photog.	103	Sociol.	32
Northern Brit.	101	Ecclesiast.	30
Heraldry	97	Embryol.	30
Theol.	95	Surveying	30
Biochem.	94		

Table 6. Comparative Frequencies of Categories in the CED

This table indicates the semantic category and number of occurrences for all categories occurring at least 30 times. The most common category of the CED is the Informal category. Following this in frequency are the Archaic, British, and U.S. categories. There are a number of categories that were not expected to occur quite so frequently. These include: Chess, Angling, Cards, Billiards, Norse Mythology, Computer Technology,

Bridge, Surgery, Cricket, and Golf. The sciences encompassed a very wide range of frequently occurring categories, including: Music, Physics, Mathematics, Pathology, Anatomy, etc. Semantic information such as this may be quite helpful from a computational linguistics point of view. The category is defined at the sense level, so that an informal sense of a word, for example, will relate to a definition, sample usage, etc., at that level.

FREQUENCIES OF HOMOGRAPH NUMBERS

	CED		W7	
HOMOGRAPH #	FREQ	FREQ/HD	FREQ	FREQ/HD
1	79222	0.97	60079	0.87
2	1928	0.02	6542	0.10
3	313	*	1427	0.02
4	81	*	475	0.01
5	15	*	164	*
6	2	*	54	*
7	-	-	17	*
8	-	-	5	*
9	-	-	3	*
total	81561	1.00	68766	1.00

Table 7. Comparative Frequencies of Homograph Numbers in the CED

This table attempts to indicate the approach that different editors take in dealing with homograph numbers. The authors of the W7 tended to create a new entry when the part of speech of the lexeme changed. This is evident in the more frequent use of larger homograph numbers in the W7 compared with the CED.

FREQUENCIES OF PARTS OF SPEECH

An entry can have more than one part of speech. When the statistics were gathered, the primary parts of speech as well as any secondary parts of speech were counted. Amsler's results are tabulated in this same way. Peterson distinguished between the primary and the secondary parts of speech, so his results were combined to allow consistency. The table indicating the frequencies of all parts of speech found in the CED, W7, and MWPD follows:

	CED		W7		MWPD	
PART OF SPEECH	FREQ	% USE	FREQ	% USE	FREQ	% USE
noun	61820	64.2	42610	62.8	15166	58.8
plural_noun	1226	1.3	-	-	-	-
noun_total	63046	65.5	42610	62.8	15166	58.8
verb_transitive	7078	7.4	4976	7.3	3	x
verb	4303	4.5	2425	3.6	5029	19.5
verb_intransitive	3247	3.4	1363	2.0	-	-
verb_imperative	-	-	10	x	-	-
verbal_auxiliary	-	-	4	x	-	-
verb_impersonal	-	-	2	x	-	-
verb_phrase	-	-	-	-	2	x
verb_total	14628	15.2	8780	12.9	5034	19.5
adjective	12661	13.2	13435	19.8	4677	18.1
adjective_postpos	291	0.3	-	-	-	-
adjective_prenominal	82	0.1	-	-	-	-
adjective_total	13034	13.5	13435	19.8	4677	18.1
combining_form	503	0.5	516	0.8	-	-
noun_combining_form	134	0.1	150	0.2	-	-
adj_combining_form	44	x	-	-	-	-
adv_combining_form	1	x	-	-	-	-
verb_combining_form	1	x	2	x	-	-

	CED		W7		MWPD	
PART OF SPEECH	FREQ	% USE	FREQ	% USE	FREQ	% USE
combining_form_total	683	0.7	668	1.0	-	-
suffix_forming_nouns	83	0.1	107	0.2	36	0.1
suff_form_pl_prop_nns	6	x	1	x	-	-
suffix	60	0.1	2	x	1	x
suffix_forming_adjs	30	x	50	0.1	26	0.1
suffix_forming_advs	5	x	7	x	4	x
suffix_forming_verbs	-	-	12	x	10	x
suffix_form_interjs	-	-	1	x	-	-
suffix_total	184	0.2	180	0.3	77	0.3
abbreviation	2423	2.5	-	-	-	-
adverb	1247	1.3	1468	2.2	549	2.1
interjection	336	0.3	94	0.1	22	0.1
preposition	130	0.1	164	0.2	120	0.5
prefix	116	0.1	74	0.1	6	x
determiner	96	0.1	-	-	-	-
pronoun	85	0.1	108	0.2	82	0.3
conjunction	62	0.1	96	0.1	55	0.2
symbol_for	61	0.1	-	-	-	-
sentence_connector	42	x	-	-	-	-
sentence_substitute	40	x	-	-	-	-
connecting_vowel	2	x	-	-	-	-
sentence_modifier	1	x	-	-	-	-

	CED		W7		MWPD	
PART OF SPEECH	FREQ	% USE	FREQ	% USE	FREQ	% USE
modifier	1	x	-	-	-	-
trademark	-	-	121	0.2	-	-
indefinate_article	-	-	3	x	-	-
definate_article	-	-	2	x	2	x
overall total	96242	100.0	67809	100.0	25790	100.0

- no occurrences.
x less than .05

Note: Sum of primary and secondary parts of speech is used for W7 data.

Table 8. Comparative Frequencies of Parts of Speech in the CED

By far the most frequent part of speech is the noun. This is consistent both with Amsler's work with the Merriam Webster Pocket Dictionary and with Peterson's efforts with the W7. In the CED, almost two-thirds of all parts of speech are nouns. Over 15% of the parts of speech in the CED are verbs, while Amsler's results indicate 19.5%, and Peterson's indicate almost 13%. 13.5% of all parts of speech in the CED are adjectives, compared with almost 20% in the W7, and 18% in the MWPD. It is interesting to note that there are a larger (absolute) number of adjectives and adverbs found in the smaller W7.

Amsler's results show that 96.4% of the parts of speech are nouns, verbs, or adjectives. Peterson's same top 3 parts of speech account for 95.5% of all parts of speech. The corresponding results in the CED indicate

that 96.7% of all parts of speech are nouns, verbs, adjectives, or abbreviations. Two and a half percent of the parts of speech in the CED are abbreviations, the distinction of which is not made in either of the other two dictionaries. Abbreviations are most likely considered to be nouns in the MWPD and the W7, so for comparison purposes these are included in the top three.

It appears that a wider range of parts of speech may be found in the CED. This list indicates that there is a larger variety of parts of speech in the CED as compared with the W7 and MWPD.

CED PARTS OF SPEECH WITH EXAMPLES

PART OF SPEECH	FREQ	EXAMPLE
n	61820	Aaron's beard
adj	12661	Aaronic
vt	7078	abandon
vb	4303	abdicate
vi	3247	abide
abbrev	2423	ABC
adv	1247	abeam
npl	1226	abuttals
comb	503	acantho-
interj	336	action
adjp	291	ablaze
ncomb	134	-aemia (as in leucaemia)
prep	130	across
prefix	116	a- (as in atonal)
det	96	a
pron	85	allyou
suffn	83	-ad (as in triad)
adjpren	82	all-around
conj	62	albeit
symbol	61	AA (an AA film in Brit.)
suffix	60	-an (as in European)

PART OF SPEECH	FREQ	EXAMPLE
adjcomb	44	-agogic
sentcon	42	also
sentsub	40	affirmative (in military context)
suffadj	30	-al (as in functional)
suffadv	30	-ad (as in cephalad)
suffppn	6	-acea (as in Crustacea)
connvowl	2	-o-, -i- (connect elem. in cpd. word)
advcomb	1	-graphically (adverb combining form)
modifier	1	bread and butter
sentmod	1	doubtless (sentence modifier)
vcomb	1	-sect (verb comb. form e.g. trisect)

Table 9. Frequencies of Parts of Speech in the CED

FREQUENCIES OF SENSES

	CED			W7		
SENSE	FREQ	FREQ/HD	% USE	FREQ	FREQ/HD	% USE
1	94780	1.16	61.9	82604	1.20	58.8
2	29249	0.36	19.1	33867	0.49	24.1
3	11841	0.15	7.7	11956	0.17	8.5
4	5788	0.07	3.8	5204	0.08	3.7
5	3309	0.04	2.2	2682	0.04	1.9
6	2117	0.03	1.4	1559	0.02	1.1
7	1407	0.02	0.9	872	0.01	0.6
8	984	0.01	0.6	532	0.01	0.4
9	718	0.01	0.5	335	*	0.2
10	553	0.01	0.4	233	*	0.2
11	423	0.01	0.3	183	*	0.1
12	340	*	0.2	143	*	0.1
13	274	*	0.2	97	*	0.1
14	221	*	0.1	66	*	x
15	178	*	0.1	47	*	x
16	147	*	0.1	29	*	x
17	121	*	0.1	22	*	x
18	92	*	0.1	19	*	x
19	77	*	0.1	13	*	x
20	67	*	x	5	*	x

	CED			W7		
SENSE	FREQ	FREQ/HD	% USE	FREQ	FREQ/HD	% USE
21	53	*	x	9	*	x
22	42	*	x	5	*	x
23	35	*	x	5	*	x
24	31	*	x	4	*	x
25	27	*	x	1	*	x
26	25	*	x	1	*	x
27	18	*	x	-	-	-
28	15	*	x	-	-	-
29	15	*	x	-	-	-
30	12	*	x	-	-	-
31	11	*	x	-	-	-
32	11	*	x	-	-	-
33	9	*	x	-	-	-
34	9	*	x	-	-	-
35	7	*	x	-	-	-
36	7	*	x	-	-	-
37	7	*	x	-	-	-
38	6	*	x	-	-	-
39	5	*	x	-	-	-
40	5	*	x	-	-	-

	CED			W7		
SENSE	FREQ	FREQ/HD	% USE	FREQ	FREQ/HD	% USE
41	4	*	x	-	-	-
42	3	*	x	-	-	-
43	3	*	x	-	-	-
44	3	*	x	-	-	-
45	3	*	x	-	-	-
46	3	*	x	-	-	-
47	3	*	x	-	-	-
48	1	*	x	-	-	-
49	1	*	x	-	-	-
50	1	*	x	-	-	-
51	-	-	-	-	-	-
totals	153061	1.88	100.0	140493	2.04	100.0

- no occurrences
* less than 0.005
x less than 0.05

Table 10. Comparative Frequencies of Sense Numbers in the CED

Peterson's work with the W7 provided comparable information regarding senses and sub-senses. This information lists the number of times that a sense or sub-sense number occurs. His data also provided an insignificant number of sub-sub-senses, which are not compatible with the structure of the CED. Comparatively the results for the sense numbers are very similar.

The largest difference occurs in the second sense of a word where the CED frequency per headword is .36 whereas that of the W7 is .49. The CED tends to have fewer senses overall for a given word, as is evident in the totals, where the CED has 1.88 senses per headword, and W7 has 2.04. The difference is that the CED can have a larger number of senses for a given word; one entry even had 50 senses.

FREQUENCIES OF SUB-SENSES

	CED			W7		
SUB-SENSE #	FREQ	FREQ/HD	% USE	FREQ	FREQ/HD	% USE
1	4945	0.06	47.3	16659	0.24	42.2
2	4950	0.06	47.4	16578	0.24	42.0
3	470	0.01	4.5	4307	0.06	10.9
4	58	*	0.6	1214	0.02	3.1
5	18	*	0.2	409	0.01	1.0
6	3	*	x	153	*	0.4
7	-	-	-	65	*	0.2
8	-	-	-	30	*	0.1
9	-	-	-	12	*	x
10	-	-	-	10	*	x
11	-	-	-	4	*	x
12	-	-	-	4	*	x
13	-	-	-	1	*	x
14	-	-	-	2	*	x
15	-	-	-	-	-	-
totals	10444	0.13	100.0	39448	0.57	100.0

- no occurrences
x less than .05
* less than .005

Table 11. Comparative Frequencies of Sub-sense Numbers in the CED

There is a marked difference in the way the CED uses sub-senses as compared to the way W7 uses them. The authors of the CED use sub-senses much less frequently than do the authors of the W7. In the W7 a total of .57 sub-sense numbers per headword are found, whereas only .13 are found in the CED. The CED has no sub-senses beyond sub-sense 6, whereas the W7 sub-sense numbers become as large as 14.

CONCLUSIONS

One of my greatest accomplishments with this project was in learning to use some of the UNIX utilities. I have developed a broad understanding of Lex, Yacc, and the Awk facility, as well as an increased understanding of programming in C. My project as a whole began as a treasure hunt with a map that was written in a foreign language. The typesetting codes were at times quite irregular, and without the book to refer to, it made the project very challenging. My intentions at the outset were far greater than what I accomplished. I intended to do a large amount of definition parsing as well, but time would not permit it.

By studying other similar efforts in the midst of this project I gained an appreciation for the art of constructing a relational lexicon. I also learned about different types of grammars and various parsing methods. Before my project began I had little knowledge of information storage and retrieval, but by reading many articles and being a part of the CODER project I have gained an insight into this field. I have also gained an understanding of many A.I. techniques through my CODER involvement.

The single largest drawback of this project was the fact that I did not initially have the CED available as a reference. The only version that was available for quite some time was a thirty year old text. If I were to do the project again, I would not commence until the hard copy of the dictionary were in my hand. Another thing that I would change is that

when I wrote my passes I would not put quite so much emphasis on the interaction between Lex and Yacc, but more on Lex. I could not have easily solved all of my problems with Lex alone, but many of the problems arose from my Yacc program. Another problem that I frequently encountered was a system problem. Often times one thing or another, ranging from video boards to mouse wires to electrical shocks, would cause system malfunctions.

Work is currently underway to clean up the roughly 3000 entries that could not be handled by my programs. At this point, I would like to make some suggestions for future work with this lexicon and lexicons in general.

The lexicon should be divided by part of speech into five files, nouns, verbs, adjectives, adverbs, and others. Vickie Klick's LSP grammar for adverb definitions [EVEN 82] can be very useful for parsing the adverb definitions. I ran some preliminary tests of adverbs in the OALDCE and found that of the first thirty adverb definitions, twenty-four (80%) were able to be parsed by this grammar or a slight modification of it. If the definitions of morphological variants are desired the adverbial suffix algorithm [EVEN 82] may be a useful tool. This algorithm deduces the definition depending on the part of speech of the headword.

A list of the most frequent verb kernels in verb definitions, and a kernel grammar for parsing verb definitions is found in [AMSL 80]. [AMSL 80] published a frequency list of words in verb definitions. I suggest using

a modification of Amsler's verb grammar and his taxonomy [ISA] relation for parsing verb definitions.

Amsler also worked with noun kernels and word frequencies in noun definitions [AMSL 80]. Evens [EVEN 82] completed some work with the part - whole relation (p. 30). I suggest the use of Amsler's list of noun relations and defining formulae, and I believe the LSP grammar for noun relations will handle a large subset of noun phrases.

Ahlswede [AHLS 83] has done the most work with adjective relations. He wrote an adjective definition grammar to parse adjective definitions in the W7. I recommend using Ahlswede's list of adjective relations and a modification of his LSP grammar or another grammar.

There are a number of other sources that may be applicable for further study. The "un-" algorithm [EVEN 82] may be quite useful to construct definitions of words with the prefix 'un'. Given certain relations, other new relations may be implied. Finally, Apresyan et al. [APRE 69] include a list of relations and their descriptions (in Russian). Fox [FOXE 80] also lists several relations that have been shown to be useful in information retrieval systems.

I was successful in creating an MU-PROLOG fact base from the original typesetting codes. This process took several passes, since the codes were not entirely consistent. Several interesting results concerning fre-

quencies of various elements of the CED were discussed. However, this is but the first step in creating a true relational lexicon.

APPENDIX A. PASS1

C PROGRAM FOR PASS1

```
#include <stdio.h>
#include <ctype.h>
#include "y.tab.c"
#include "lex.yy.c"
char *progname;     /* for error messages */
int  lineno = 1;

main(argc,argv)     /* dictionary */
     char *argv[];

{
     progname = argv[0];
     printf("\n");
     printf("\nSYLL ");
     yyparse();
}
```

YACC PROGRAM FOR PASS1

```
%{
int last=1;
int spend;
unsigned long temp_pos;
char c, cc, ccc;
extern unsigned long stor_frst_byt;
extern unsigned long stor_last_byt;
extern unsigned long byt_offset;
extern int numh_flg;
extern int morph_flg;
extern int pl_flg;
extern int first1_flg;
extern int first2_flg;
extern int first3_flg;
extern int also_flg;
extern int five_flg;
extern int six_flg;
extern int one_flg;
extern int samp_flg;
extern int n_flg;
extern int dee_flg;
extern int cat_flg;
extern int def_flg;
extern int begd_flg;
extern int begm_flg;
extern int dflg;
extern int yyleng;
```

```
extern char yytext[];
extern char wordarry[50];
%}
%token A1
%token A3
%token A5
%token FIRST
%token POS
%token LETT
%token APRON
%token MORPH
%token GARBAGE
%token OR
%token PL
%token TRINTRPP
%token NUMB
%token DEF
%token SAMP
%token END
%token CATEGORY
%token SYLL
%token MINUS
%start dict
%%
dict     :   dict A1 restdic
            {printf("\n");
             printf("\n");
             printf("SYLL ");}
         |   A1 restdic
            {printf("\n");
             printf("\n");
             printf("SYLL ");}
         |   error restdic
            {printf("\n");
             printf("\n");
             printf("SYLL ");}
         ;

restdic :    hdwds ptspch defins morphs
         ;

ptspch   :
         |  POS en mcat
         |  CATEGORY POS en mcat
         ;

mcat     :
         |  CATEGORY
         ;

en       :
```

```
          |  PL een moren      /*  only if ", pl. #1" is found */
          |  een moren         /*  if vb. -s, -ing, -ed endings */
          |  fir APRON
          |  fir
          ;

fir       :  FIRST
          |  FIRST fir
          ;

een       :  END
          |  END een
          ;

moren     :
          |  OR een
          |  OR een APRON
          |  APRON moren
          ;

hdwds     :   word morwds
          ;

word      :   syllab
          |   word  MINUS syllab
          |   word syllab
          ;

morwds    :
          |   mwds morwds
          ;

syllab    :   SYLL
              { printf("%s",yytext);
              }
          ;

mwds      :  word
          |  A3 word
          |  OR A3 word
          |  CATEGORY
          |  NUMB
          |  NUMB APRON
          |  APRON
          ;

defins    :
          |  POS defins
          |  garb defpart sampart defins
          ;
```

```
defpart :   DEF garb
        |   DEF garb defpart
        ;

sampart :
        |   SAMP garb sampart
        ;

garb    :
        |   GARBAGE garb
        |   CATEGORY garb
        |   mcat LETT garb
        ;

morphs  :
        |   morm mormor morphs
        |   POS morm morphs
        ;

morm    :   mm
        |   morm mm
        ;

mm      :   MORPH
             {printf("%s", yytext);}
        ;

mormor  :
        |   APRON mormor
        |   POS mormor
        |   CATEGORY mormor
        |   DEF mormor
        ;
%%
yyerror()
{
if (last == 0) printf("******error******"); /* last pass it must be 1 */
fprintf(stderr, "%u", stor_frst_byt);  /* beginning of headword */
fprintf(stderr, " ");
spend = 0;
temp_pos = byt_offset - stor_frst_byt;  /* pos. of err */
stor_last_byt = byt_offset;
while (spend ==0){
  c = getchar();
  stor_last_byt = stor_last_byt + 1;
  cc = getchar();
  stor_last_byt = stor_last_byt + 1;
  if ((c == '#')&&((cc == 'H')||(cc == 'h'))) spend = 1;
  else if (cc == '#') {
     ccc = getchar();
     stor_last_byt = stor_last_byt + 1;
```

```
        if ((ccc == 'H')||(ccc == 'h')) spend = 1; }
     }
   byt_offset = stor_last_byt;
   stor_last_byt = stor_last_byt - 2;
   fprintf(stderr, "%u", stor_last_byt);  /* end of headword */
   fprintf(stderr, " ");
   fprintf(stderr, "%u", (temp_pos));  /* pos. of err */
   fprintf(stderr, " ");
   fprintf(stderr, "%s\n", wordarry);  /* headword of error */
   stor_frst_byt = byt_offset - 1;
   numh_flg = 1;
   morph_flg = 0;
   pl_flg = 0;
   first1_flg = 0;
   first2_flg = 0;
   first3_flg = 0;
   also_flg = 0;
   five_flg = 0;
   six_flg = 0;
   one_flg = 0;
   samp_flg = 0;
   n_flg = 0;
   dee_flg = 0;
   cat_flg = 0;
   def_flg = 0;
   begd_flg = 0;
   begm_flg = 0;
   dflg = 0;
   printf("\n");
   printf("\n");
   printf("\n");
   printf("SYLL ");
   yyclearin;
   yyerrok;
   }
```

LEX PROGRAM FOR PASS1

```
%{
# define YYLMAX 4096
char wordarry[50];
unsigned long byt_offset=0;    /* total byte offset */
unsigned long stor_frst_byt=0;  /* byte offset of beginning of entry */
unsigned long stor_last_byt=0;  /* byte offset of end of entry */
int alpha_flg=0;   /* within a headword  */
int n_flg=0;       /* homonym number seen */
int numh_flg=0;    /* have seen a #h or #H */
int bfirst_flg=0;   /* beginning of a new first name */
int first1_flg=0;   /* first name flag -after a noun */
int first2_flg=0;   /* in the first name section  */
int first3_flg=0;   /* within a first name  */
```

```
int also_flg=0;    /* also called: is seen */
int pl_flg=0;      /* pl., is found- a plural ending */
int samp_flg=0;    /* : #6 found- in sample section */
int begs_flg=0;    /* beginning of a new sample usage */
int dee_flg=0;     /* a $D. is found  */
int cat_flg=0;     /* in a category  */
int morph_flg=0;   /* in the morph. var. section */
int begm_flg = 1;  /* beginning of a new morph */
int begd_flg=0;    /* beginning of a new defn */
int def_flg=0;     /* within a defn */
int dflg=0;        /* in defn section - after pronunc */
int one_flg=0;     /* when a #1 is found */
int three_flg=0; /* when a #3 is found */
int five_flg=0; /* when a #5 is found */
int six_flg=0; /* when a #6 is found */
int b2012_flg=0;   /* have seen definitions b20, b21, or b22 */
%}
%n 5000
%a 5000
%o 5000
%e 5000
%p 5000
%k 5000
alpha    [a-zA-Z0-9\'\'\.\,\-\¬\/\&]+
b9       \@m
b10      \@n
b13      \?\-
b14      \&\-
b15      \@f
b20      \@\.
b21      \#\!
b22      \#\+
num     [0-9]+
bl      [\ ]+
a       [aA]
b       [bB]
c       [cC]
d       [dD]
e       [eE]
f       [fF]
g       [gG]
h       [hH]
i       [iI]
j       [jJ]
k       [kK]
l       [lL]
m       [mM]
n       [nN]
o       [oO]
p       [pP]
r       [rR]
```

```
s       [sS]
t       [tT]
u       [uU]
v       [vV]
w       [wW]
x       [xX]
y       [yY]
sa      [a]
sb      [b]
sc      [c]
sd      [d]
%%
1. [\ \t]+   byt_offset = byt_offset + yyleng;
2. ({b9}|{b10})(#5)?[\[][¬\]]*[\]]    { also_flg = 0;
                               byt_offset = byt_offset + yyleng;
                               }
3. (\#1)?[\{][¬\}&¬\.]*[\}]       { printf("%s", yytext);
                               byt_offset = byt_offset + yyleng; }
4. \?\&      {printf("&");
          byt_offset = byt_offset + yyleng;
          if (alpha_flg == 1)
             numh_flg = 1; }
5. \?\%      {printf(" degrees ");
          byt_offset = byt_offset + yyleng;  }
6. \@t       {printf(",");
          byt_offset = byt_offset + yyleng;  }
7. \@\=      {printf("-");
          byt_offset = byt_offset + yyleng;  }
8. \&\!      {printf("!");
          byt_offset = byt_offset + yyleng;  }
9. \#2       {printf("");
          byt_offset = byt_offset + yyleng;  }
10. \#4(\*)? {printf("");
          byt_offset = byt_offset + yyleng;  }
11. \#7      {printf("");
          byt_offset = byt_offset + yyleng;  }
12. \#8      {printf("");
          byt_offset = byt_offset + yyleng;  }
13. \#9      {printf("");
          byt_offset = byt_offset + yyleng;  }
14. \#[a-gi-zA-GI-Z]    {printf("");
          byt_offset = byt_offset + yyleng;  }
15. \@f      {printf("");
          byt_offset = byt_offset + yyleng;  }
16. \@b      {printf("");
          byt_offset = byt_offset + yyleng;  }
17. \@m      {printf("");
          byt_offset = byt_offset + yyleng;  }
18. \@[NT]   {printf("");
          byt_offset = byt_offset + yyleng;  }
19. \*[\,\¬\"\'\'\¬]  {printf("");  /* funny accents */
```

```
          byt_offset = byt_offset + yyleng;
          if (def_flg == 1) return(GARBAGE); }
20. \@t\#{m}\@t   {printf(" X ");
              byt_offset = byt_offset + yyleng; }
21. \@t\#\+\@t   {printf(" + ");
              byt_offset = byt_offset + yyleng; }
22. \@t\=\@t     {printf(" = ");
              byt_offset = byt_offset + yyleng; }
23. \@[0-9]      {printf("");
              byt_offset = byt_offset + yyleng; }
24. \@\'         {printf("''");
              byt_offset = byt_offset + yyleng; }
25. \@\"         {printf("\"");
              byt_offset = byt_offset + yyleng; }
26. \@           {printf("");
              byt_offset = byt_offset + yyleng; }
27. \$\/         {printf(" sqrt ");
              byt_offset = byt_offset + yyleng; }
28. \%\-         {printf("-");
              byt_offset = byt_offset + yyleng; }
29. \%           {printf("¬");
              byt_offset = byt_offset + yyleng; }
30. \@t\#8[0-9]*(\@t)?(\#5)?   {printf("");
              byt_offset = byt_offset + yyleng; }
31. (\@8)?\#8[0-9]*(\@t)?(\#5)?   {printf("");
              byt_offset = byt_offset + yyleng; }
32. \?\!       {byt_offset = byt_offset + yyleng;
           if (dflg == 1) return(GARBAGE); }
33. \#[hH]       {  /* this section resets all flags to 0 except numh (#H) */
              byt_offset = byt_offset + yyleng;
              stor_frst_byt = byt_offset - yyleng;
              first1_flg = 0;
              first2_flg = 0;
              first3_flg = 0;
              also_flg=0;
              pl_flg= 0;
              five_flg=0;
              samp_flg=0;
              six_flg=0;
              one_flg=0;
              n_flg=0;
              dee_flg=0;
              cat_flg=0;
              morph_flg=0;
              begm_flg = 0;
              numh_flg=1;
              def_flg=0;
              begd_flg=0;
              dflg=0;
              return(A1);
```

```
                }
34. ({b20}|{b21}|{b22})  {  /*  repl. with '-' for syll if in hdwd. */
                          byt_offset = byt_offset + yyleng;
                          if ((morph_flg == 1)||(pl_flg == 1)||
                 (first2_flg == 1)||(also_flg == 1)||(one_flg == 1))
                             printf("");
                          else if(alpha_flg == 1){
                             alpha_flg=0;
                             b2012_flg =1;
                             printf("_");
                             return(MINUS); }
                          else if (def_flg == 1)  {
                             printf("");
                             return(GARBAGE); }
                        }
35.  \#3           {
             three_flg =1;
             one_flg = 0;
             five_flg = 0;
             six_flg = 0;
             byt_offset = byt_offset + yyleng;
             return(A3);
             }
36.  \#5\%1        {  /* homonym #1 is found for a headword */
             three_flg =0;
             one_flg = 0;
             five_flg = 1;
             six_flg = 0;
             n_flg = 1;
             printf(" \' ,1 ");
             byt_offset = byt_offset + yyleng;
             if (def_flg == 0) return(NUMB);
             }
37.  \#5\%2        {  /* homonym #2 is found for a headword */
             three_flg =0;
             one_flg = 0;
             five_flg = 1;
             six_flg = 0;
             n_flg = 1;
             printf(" \' ,2 ");
             byt_offset = byt_offset + yyleng;
             if (def_flg == 0) return(NUMB);
             }
38.  \#5\%3        {  /* homonym #3 is found for a headword */
             three_flg =0;
             one_flg = 0;
             five_flg = 1;
             six_flg = 0;
             n_flg = 1;
             printf(" \' ,3 ");
             byt_offset = byt_offset + yyleng;
```

```
            if (def_flg == 0) return(NUMB);
            }
39. \#5\%4      { /* homonym #4 is found for a headword */
            three_flg =0;
            one_flg = 0;
            five_flg = 1;
            six_flg = 0;
            n_flg = 1;
            printf(" \' ,4 ");
            byt_offset = byt_offset + yyleng;
            if (def_flg == 0) return(NUMB);
            }
40. \#5\%5      { /* homonym #5 is found for a headword */
            three_flg =0;
            one_flg = 0;
            five_flg = 1;
            six_flg = 0;
            n_flg = 1;
            printf(" \' ,5 ");
            byt_offset = byt_offset + yyleng;
            if (def_flg == 0) return(NUMB);
            }
41. \#5\%6      { /* homonym #6 is found for a headword */
            three_flg =0;
            one_flg = 0;
            five_flg = 1;
            six_flg = 0;
            n_flg = 1;
            printf(" \' ,6 ");
            byt_offset = byt_offset + yyleng;
            if (def_flg == 0) return(NUMB);
            }
42. \#5\%7      { /* homonym #7 is found for a headword */
            three_flg =0;
            one_flg = 0;
            five_flg = 1;
            six_flg = 0;
            n_flg = 1;
            printf(" \' ,7 ");
            byt_offset = byt_offset + yyleng;
            if (def_flg == 0) return(NUMB);
            }
43. \#5\%8      { /* homonym #8 is found for a headword */
            three_flg =0;
            one_flg = 0;
            five_flg = 1;
            six_flg = 0;
            n_flg = 1;
            printf(" \' ,8 ");
            byt_offset = byt_offset + yyleng;
            if (def_flg == 0) return(NUMB);
```

```
                  }
44. \#5\%9            {  /* homonym #9 is found for a headword */
                  three_flg =0;
                  one_flg = 0;
                  five_flg = 1;
                  six_flg = 0;
                  n_flg = 1;
                  printf(" \' ,9 ");
                  byt_offset = byt_offset + yyleng;
                  if (def_flg == 0) return(NUMB);
                  }
45. \#5\)             {
                  printf(")");
                  five_flg = 1;
                  three_flg =0;
                  one_flg = 0;
                  byt_offset = byt_offset + yyleng;
                  six_flg = 0;}
46. \#5               {
                  pl_flg = 0;
                  also_flg = 0;
                  first1_flg = 0;
                  first2_flg = 0;
                  first3_flg = 0;
                  five_flg = 1;
                  three_flg =0;
                  one_flg = 0;
                  six_flg = 0;
                  dee_flg = 0;
                  byt_offset = byt_offset + yyleng;
                  if (dflg == 1){     /* if in def section next is definition */
                     def_flg = 1;
                     return(GARBAGE);}
                  }
47. (\,)?({bl})?\#6{o}{r}({bl})+({s}{o}{m}{e}{t}{i}{m}{e}{s}({bl})+)?
                  {b}{e}{f}{o}{r}{e}({bl})+{a}({bl})+{v}{o}{w}{e}{l}      {
                  byt_offset = byt_offset + yyleng;
                  printf("or ");  /* or before a vowel ->  'or'  */
                     }
48. (\,)?({bl})?\#6{o}{r}{bl}({e}{s}{p}\.{bl})?({u}\.{s}\.{bl})?
                {  /* or, or U.S. -> 'or'  */
                  five_flg = 0;
                  three_flg =0;
                  one_flg = 0;
                  six_flg = 1;
                  printf("or ");        /* within a morph don't do anything */
                  byt_offset = byt_offset + yyleng;
                  if (morph_flg == 0) return(OR);
                  }
49. (\,)?({bl})?\#6{o}{r}\@N      {  /* or  within morph  */
                  five_flg = 0;  /* within morph don't print or */
```

```
            three_flg =0;
            one_flg = 0;
            six_flg = 1;
            byt_offset = byt_offset + yyleng;
            }
50.  (\,)?({bl})?\#6{o}{r}{bl}({u}\.{s}\.{bl})?(\#1|\#5\(\#6({f}{e}{m}\.|
            {s}{o}{m}{e}{t}{i}{m}{e}{s})\#5       {
            five_flg = 0;     /* or #1, or U.S. #1 -> 'or'  */
            three_flg =0;
            one_flg = 1;
            six_flg = 0;
            printf("or ");
            byt_offset = byt_offset + yyleng;
            if (def_flg == 1) return(GARBAGE);
            else if (morph_flg == 0) {  /* #1 sig. ending within pl. */
               pl_flg = 1;
               return(OR); }
            }
51.  \#6{n}\.     {  /* Parts of Speech. sig. beg. of defs, end of pron. */
            first1_flg = 1;      /* nouns have first names */
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS n.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
52.  \#6{p}{l}\.{bl}{n}\.     {
            first1_flg = 1;      /* nouns have first names */
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS npl.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
53.  \#6({p}{l}\.{bl})?{f}{e}{m}\.({bl})?(\@f)?{n}\.     { /* pl. opt. */
            first1_flg = 1;      /* nouns have first names */
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
```

```
            five_flg = 0;
            printf("\n");
            printf("POS nfem.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
54. \#6{p}{r}{o}{n}({o}{u}{n}|\.)     {
            first1_flg = 1;     /* pronouns may have first names */
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS pron.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
55. \#6{a}{d}{j}(\.|{e}{c}{t}{i}{v}{e}){bl}{c}{o}{m}{b}{i}{n}{i}{n}{g}
            {bl}{f}{o}{r}{m}\.     {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS adjcomb.");
            printf("\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
56. \#6{n}(\.|{o}{u}{n}){bl}{c}{o}{m}{b}{i}{n}{i}{n}{g}{bl}{f}{o}{r}{m}\.
{
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS ncomb.");
            printf("\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
57. \#6{c}{o}{m}{b}{i}{n}{i}{n}{g}{bl}{f}{o}{r}{m}\.     {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
```

```
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS comb.");
            printf("\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
58. \#6{n}\.\,({bl}|{b15}){a}{d}{j}\.     {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS n.");
            printf("\n");
            printf("POS adj.");
            printf("\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
59. \#6{n}\.\,({bl}|{b15}){a}{d}{v}\.     {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS n.");
            printf("\n");
            printf("POS adv.");
            printf("\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
60. \#6{n}\.\,({bl}|{b15}){v}{b}\.     {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS n.");
            printf("\n");
            printf("POS vb.");
            printf("\n");
```

```
             byt_offset = byt_offset + yyleng;
             return(POS);
             }
 61. \#6{v}{b}\.      {
             dflg = 1;
             n_flg = 0;
             six_flg = 1;
             three_flg =0;
             one_flg = 0;
             five_flg = 0;
             printf("\n");
             printf("POS vb.\n");
             byt_offset = byt_offset + yyleng;
             return(POS);
             }
 62. \#6{v}{b}\.{bl}{p}{a}{s}{t}{bl}      {
             dflg = 1;
             n_flg = 0;
             six_flg = 1;
             three_flg =0;
             one_flg = 0;
             five_flg = 0;
             printf("\n");
             printf("POS vb.\n");
             printf("PAST ");
             byt_offset = byt_offset + yyleng;
             return(POS);
             }
 63. \#6{v}{b}\.{bl}\#1      {
             dflg = 1;
             n_flg = 0;
             six_flg = 0;
             three_flg =0;
             one_flg = 1;
             five_flg = 0;
             pl_flg = 1;       /* vb. -s, -ing, -ed endings */
             printf("\n");
             printf("POS vb.\n");
             printf("PL ");
             byt_offset = byt_offset + yyleng;
             return(POS);
             }
 64. \#6{v}{b}\.\,({bl}|{b15}){n}\.      {
             dflg = 1;
             n_flg = 0;
             six_flg = 1;
             three_flg =0;
             one_flg = 0;
             five_flg = 0;
             printf("\n");
             printf("POS vb.");
```

```
            printf("\n");
            printf("POS n.");
            printf("\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
65. \#6{d}{e}{t}{e}{r}{m}{i}{n}{e}{r}\.          {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS det.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
66. \#6{c}{o}{n}{j}(\.|{u}{n}{c}{t}{i}{o}{n})          {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS conj.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
67. \#6{p}{r}{e}{p}({o}{s}{i}{t}{i}{o}{n}|\.)          {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS prep.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
68. \#6{a}{d}{j}\.          {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS adj.\n");
```

```
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
69. \#6{a}{d}{j}\.{bl}(\#5\(\#6|\(){p}{r}{e}{n}
            {o}{m}{i}{n}{a}{l}(\#5)?\)  {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS adjpren.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
70. \#6{a}{d}{j}\.\,({bl}|{b15}){n}\.       {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS adj.");
            printf("\n");
            printf("POS n.");
            printf("\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
71. \#6{a}{d}{j}\.\,({bl}|{b15}){a}{d}{v}\.       {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS adj.");
            printf("\n");
            printf("POS adv.");
            printf("\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
72. \#6{a}{d}{v}\.        {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
```

```
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS adv.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
73.  \#6{a}{d}{v}\.\,({bl}|{b15}){a}{d}{j}\.        {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS adv.");
            printf("\n");
            printf("POS adj.");
            printf("\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
74.  \#6{a}{d}{v}\.\,({bl}|{b15}){n}\.        {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS adv.");
            printf("\n");
            printf("POS n.");
            printf("\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
75.  \#6{p}{r}{e}{f}{i}{x}[¬\.]*\.        {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS prefix.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
76.  \#6{s}{u}{f}{f}{i}{x}{bl}{f}{o}{r}{m}{i}{n}{g}{bl}
            {a}{d}{j}{e}{c}{t}{i}{v}{e}{s}\.     {
```

```
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS suffadj.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
77. \#6{s}{u}{f}{f}{i}{x}{bl}{f}{o}{r}{m}{i}{n}{g}{bl}
            {a}{d}{v}{e}{r}{b}{s}\.
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS suffadv.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
78.
\#6{s}{u}{f}{f}{i}{x}{bl}{f}{o}{r}{m}{i}{n}{g}{bl}{p}{l}{u}{r}{a}{l}
            {bl}{p}{r}{o}{p}{e}{r}{bl}{n}{o}{u}{n}{s}\.
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS suffppn.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
79. \#6{s}{u}{f}{f}{i}{x}{bl}{f}{o}{r}{m}{i}{n}{g}{bl}
            {n}{o}{u}{n}{s}\.        {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS suffn.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
```

```
         }
80. \#6{s}{u}{f}{f}{i}{x}[¬\.]*\.          {
         dflg = 1;
         n_flg = 0;
         six_flg = 1;
         three_flg =0;
         one_flg = 0;
         five_flg = 0;
         printf("\n");
         printf("POS suffix.\n");
         byt_offset = byt_offset + yyleng;
         return(POS);
         }
81. \#6{i}{n}{t}{e}{r}{j}({e}{c}{t}{i}{o}{n}|\.)          {
         dflg = 1;
         n_flg = 0;
         six_flg = 1;
         three_flg =0;
         one_flg = 0;
         five_flg = 0;
         printf("\n");
         printf("POS interj.\n");
         byt_offset = byt_offset + yyleng;
         return(POS);
         }
82. \#6{i}{n}{t}{e}{r}{j}({e}{c}{t}{i}{o}{n}|\.)\,{bl}{n}\.          {
         dflg = 1;
         n_flg = 0;
         six_flg = 1;
         three_flg =0;
         one_flg = 0;
         five_flg = 0;
         printf("\n");
         printf("POS interj.\n");
         printf("POS n.\n");
         byt_offset = byt_offset + yyleng;
         return(POS);
         }
83. \#6{s}{e}{n}{t}{e}{n}{c}{e}{bl}{c}{o}{n}{n}{e}{c}{t}{o}{r}\.     {
         dflg = 1;
         n_flg = 0;
         six_flg = 1;
         three_flg =0;
         one_flg = 0;
         five_flg = 0;
         printf("\n");
         printf("POS sentcon.\n");
         byt_offset = byt_offset + yyleng;
         return(POS);
         }
84. \#6{s}{e}{n}{t}{e}{n}{c}{e}{bl}{s}{u}{b}{s}{t}{i}{t}{u}{t}{e}\.  {
```

```
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS sentsub.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
85.  \#6{s}{e}{n}{t}{e}{n}{c}{e}{bl}{m}{o}{d}{i}{f}{i}{e}{r}\.      {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            printf("\n");
            printf("POS sentmod.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);
            }
86.  \#6{s}{y}{m}{b}{o}{l}\ {f}{o}{r}(\:)?    {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            def_flg = 1;
            printf("\n");
            printf("POS symbol.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);   }
87.  \#6{a}{b}{b}{r}{e}{v}\.\ {f}{o}{r}(\:)?    {
            dflg = 1;
            n_flg = 0;
            six_flg = 1;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            def_flg = 1;
            printf("\n");
            printf("POS abbrev.\n");
            byt_offset = byt_offset + yyleng;
            return(POS);   }
88.  \:{bl}\#6      {  /* beginning of sample, end of def. */
            printf(".");
            samp_flg = 1;
            def_flg = 0;
```

```
            begd_flg = 0;
            begs_flg = 1;
            byt_offset = byt_offset + yyleng;
            three_flg =0;
            one_flg = 0;
            five_flg = 0;
            six_flg = 1; }
89.  \:{bl}      {
            byt_offset = byt_offset + yyleng;
            printf(": ");
            }
90.  \,{bl}{s}{i}{n}{g}\.{bl}(\#1|\#2)    {   /* beginning of plurals  */
            byt_offset = byt_offset + yyleng;
            if (six_flg == 1){
               printf("\nSING ");
               one_flg = 1;
               six_flg = 0;
               five_flg = 0;
               three_flg =0;
               pl_flg = 1;
               return(PL); }
            }
91.  \,{bl}{p}{l}\.{bl}(\#1|\#2)    {  /* beginning of plurals  */
            byt_offset = byt_offset + yyleng;
            if (six_flg == 1){
               printf("PL ");
               one_flg = 1;
               six_flg = 0;
               five_flg = 0;
               three_flg =0;
               pl_flg = 1;
               return(PL); }
            }
92.  \#6             {
              six_flg = 1;
              five_flg = 0;
              three_flg =0;
              one_flg = 0;
              byt_offset = byt_offset + yyleng;
              if ((dee_flg == 1)||(morph_flg == 1))  {
                 cat_flg = 1;  /* possibility of categ before part of spch. */
                 def_flg = 0;  }
              if (def_flg == 1) return(GARBAGE);
              }
93.  \#5[\(][¬\)]*[\)](\.)?  { /* if first fol by pron. ret pron. */
              first2_flg = 0; /* if not pl and in def sect print it. */
              first1_flg = 0;
              byt_offset = byt_offset + yyleng;
              six_flg = 0;
              one_flg = 0;
              three_flg = 0;
```

```
                five_flg = 1;
                if (also_flg == 1) {
                   printf("\n");
                   also_flg = 0; }
                else if (first3_flg == 1) {
                   printf("\n");
                   return(APRON); }
                else if ((dflg == 1)&&(pl_flg == 0)) {
                      printf("%s", yytext);
                      return(GARBAGE);   }
                else  {
                   printf("\n");
                   return(APRON); }
                   }
94. [\(][¬\)]*[\)]   {  /* if see a hom number, pron follows. */
                byt_offset = byt_offset + yyleng;
                if (n_flg == 1)
                   return(APRON);
                else  {
                   printf("%s", yytext);
                   return(GARBAGE); }
                }
95. {b10}\#5       {
             first2_flg = 0;
             first1_flg = 0;
             byt_offset = byt_offset + yyleng;
             also_flg = 0;
             five_flg = 1;
             three_flg =0;
             one_flg = 0;
             six_flg = 0;
             pl_flg = 0;
             if (def_flg == 1) /* if w/in def and see @n#5m - ignore */
                return(GARBAGE);
             else {       /* @n#5 signals def. so turn on flags. */
                def_flg =1;
                begd_flg = 1;
                dflg = 1;
                return(GARBAGE);
                }
             }
96. {b10}        {
             pl_flg = 0;
             byt_offset = byt_offset + yyleng;
             if (def_flg == 1) {
                printf(" ");
                return(GARBAGE); }
             }
97. ({b10}|\#1|{b1})+{sa}\.{b1}    {
               printf("\n"); /* a subsense letter */
               six_flg = 0;
```

```
                 one_flg = 1;
                 three_flg = 0;
                 five_flg = 0;
                 byt_offset = byt_offset + yyleng;
                 pl_flg = 0;
                 printf("\n");
                 printf("SUBDEFA ");
                 return(LETT); }
98.  ({b10}|\#1|{bl})+{sb}\.{bl}    {printf("\n");
                 byt_offset = byt_offset + yyleng;
                 six_flg = 0;
                 one_flg = 1;
                 three_flg = 0;
                 five_flg = 0;
                 pl_flg = 0;
                 printf("\n");
                 printf("SUBDEFB ");
                 return(LETT); }
99.  ({b10}|\#1|{bl})+{sc}\.{bl}    {printf("\n");
                 byt_offset = byt_offset + yyleng;
                 six_flg = 0;
                 one_flg = 1;
                 three_flg = 0;
                 five_flg = 0;
                 pl_flg = 0;
                 printf("\n");
                 printf("SUBDEFC ");
                 return(LETT); }
100.  ({b10}|\#1|{bl})+{sd}\.{bl}    {printf("\n");
                 byt_offset = byt_offset + yyleng;
                 six_flg = 0;
                 one_flg = 1;
                 three_flg = 0;
                 five_flg = 0;
                 pl_flg = 0;
                 printf("\n");
                 printf("SUBDEFD ");
                 return(LETT); }
101.  ({b10}|\#1|{bl})+[e]\.{bl}    {printf("\n");
                 byt_offset = byt_offset + yyleng;
                 six_flg = 0;
                 one_flg = 1;
                 three_flg = 0;
                 five_flg = 0;
                 pl_flg = 0;
                 printf("\n");
                 printf("SUBDEFE ");
                 return(LETT); }
102.  ({b10}|\#1|{bl})+[f]\.{bl}    {printf("\n");
                 byt_offset = byt_offset + yyleng;
                 six_flg = 0;
```

```
                    one_flg = 1;
                    three_flg = 0;
                    five_flg = 0;
                    pl_flg = 0;
                    printf("\n");
                    printf("SUBDEFF ");
                    return(LETT); }
103.  ({b10}|\#5){a}{l}{s}{o}({bl}{c}{a}{l}{l}{e}{d})?\:{bl}\#1        {
                    byt_offset = byt_offset + yyleng;
                    printf("\n");
                    printf("ALSO ");
                    also_flg = 1;
                    one_flg = 1;
                    def_flg = 0;
                    return(GARBAGE);
                    }
104.  ({b10}|\#5){o}{f}{f}{i}{c}{i}{a}{l}{bl}{n}{a}{m}{e}\:{bl}\#1        {
                    byt_offset = byt_offset + yyleng;
                    printf("\n");
                    printf("ALSO ");
                    also_flg = 1;
                    one_flg = 1;
                    return(GARBAGE);
                    }
105.  \#1   { /*  can have first1 name (a noun) so turn on flags */
                    one_flg = 1;
                    six_flg = 0;
                    three_flg = 0;
                    five_flg = 0;
                    byt_offset = byt_offset + yyleng;
                    if (first1_flg == 1) {
                          first2_flg = 1;
                          bfirst_flg = 1;
                          first1_flg = 0;
                          }
                    if ((def_flg == 1)||(cat_flg == 1))
                       return(GARBAGE);  /* if def or cat, ignore */
                    }
106.  {b9}\#1\¬\#1\$D\.{bl}\#6  {
                    byt_offset = byt_offset + yyleng;
                    six_flg = 1;
                    one_flg = 0;
                    three_flg = 0;
                    five_flg = 0;
                    n_flg = 0;
                    pl_flg = 0;
                    samp_flg = 0;
                    dee_flg = 1;
                    begd_flg = 1;
                    def_flg = 1;
                    dflg = 1;
```

```
                }
107.  ({b10})?(\#1)?(\$D)\.     {    /* signal of beginning of def. */
                byt_offset = byt_offset + yyleng;
                six_flg = 0;
                one_flg = 1;
                three_flg = 0;
                five_flg = 0;
                n_flg = 0;
                pl_flg = 0;
                samp_flg = 0;
                dee_flg = 1;
                begd_flg = 1;
                def_flg = 1;
                dflg = 1;  }
108.  ({b9}|\#5)*({b1})?({b13}|{b14}|\#1)({b1})?
                ({b13}|{b14}|\#1) { /* signal begin of morph. */
                byt_offset = byt_offset + yyleng;
                morph_flg = 1;
                one_flg = 1;
                six_flg = 0;
                three_flg = 0;
                five_flg = 0;
                samp_flg = 0;  /* end of pl, defs, sample  */
                pl_flg = 0;
                dflg = 0;
                def_flg = 0;
                alpha_flg = 1;
                begm_flg = 1;
                }
109.  ({b10}|{b9})({b13}|{b14}|\#1\¬)  {
                byt_offset = byt_offset + yyleng;
                }
110.  {alpha}         {
                byt_offset = byt_offset + yyleng;
                if (first2_flg == 1) {  /* if it is a first name */
                   if (bfirst_flg == 1) {  /* print new 1st name */
                      bfirst_flg = 0;
                      printf("\n");
                      printf("FIRST "); }
                   printf("%s", yytext);
                   first3_flg = 1;
                   return(FIRST); }
                else if (also_flg == 1) {  /* if also: is found */
                   printf("%s", yytext);
                   return(GARBAGE);
                   }
                else if ((five_flg == 0)&&(samp_flg == 0)&&(pl_flg == 0)
                   &&(six_flg == 0)&&(def_flg == 0)){
                   if (morph_flg == 1) {
                      if (begm_flg == 1) {  /* if beginning of new morph */
                         begm_flg = 0;
```

```
                printf("\n");
                printf("MORPH "); }
             return(MORPH);  }
          else if ((numh_flg == 1)||(b2012_flg == 1)
             ||(three_flg == 1)){
             if (numh_flg == 1)
                wordarry[0] ='\0';
             numh_flg = 0;    /* are w/in word */
             three_flg = 0;   /* or after #H or #3 */
             alpha_flg = 1;
             b2012_flg = 0;
             strcat(wordarry, yytext);   /* store headword */
             return(SYLL); }
          else if (one_flg == 1) printf("%s", yytext);
             }
       else if (samp_flg == 1) {   /* if a sample usage */
          if (begs_flg == 1) {   /* if beginning of new samp */
             begs_flg = 0;
             printf("\n");
             printf("SAMP "); }
          printf("%s", yytext);
          def_flg = 0;
          return(SAMP); }
       else if ((pl_flg == 1)&&((one_flg == 1)
          ||(b2012_flg == 1)))  {
          printf("%s", yytext);   /* plural ending to print */
          return(END); }
       else if ((five_flg == 1)||(def_flg == 1)) {
          if (begd_flg == 1) {   /* if beginning of new def */
             begd_flg = 0;
             printf("\n");
             printf("DEF "); }
          dee_flg = 0;
          morph_flg = 0;
          printf("%s", yytext);
          return(DEF); }
       else if ((six_flg == 1)||(cat_flg == 1)) {
          cat_flg = 0;          /* cat to print */
          dee_flg = 0;
          printf("\n");
          printf("CATEGORY ");
          printf("%s", yytext);
          return(CATEGORY); }
       }
111. \(           {
       byt_offset = byt_offset + yyleng;
       if (def_flg == 1) printf("("); }
112. \)           {
       byt_offset = byt_offset + yyleng;
       if (def_flg == 1) printf(")"); }
113. \;           {
```

```
                byt_offset = byt_offset + yyleng;
                if ((def_flg == 1)||(samp_flg == 1)) printf(";"); }
114. .               {
                byt_offset = byt_offset + yyleng;
                alpha_flg = 0;
                b2012_flg = 0;
                if (def_flg == 1) return(GARBAGE); }
```

APPENDIX B. PASS2

```
%{
# define YYLMAX 4096
int senses_flg=0;  /* looking for word senses (after a compare, see)*/
int g_flg=0;  /* have seen #5(#6 garbage #5)  already  */
int num_flg=0;  /* have seen a homonym number */
int lparn_flg=0;  /* left paren. allow ( equation ) with sup.script */
%}
%a 5000
%n 5000
%e 5000
%k 5000
%p 9000
nl      [\n]+
bl      [\ ]+
num     [0-9]+
a       [aA]
b       [bB]
c       [cC]
d       [dD]
e       [eE]
f       [fF]
g       [gG]
h       [hH]
i       [iI]
j       [jJ]
k       [kK]
l       [lL]
m       [mM]
n       [nN]
o       [oO]
p       [pP]
r       [rR]
s       [sS]
t       [tT]
u       [uU]
v       [vV]
w       [wW]
x       [xX]
y       [yY]
%%
1. \?\&         printf("&");
2. \&\!         printf("!");
3. \?\!         printf("");
4. \?\%         printf(" degrees ");
5. \@\=         printf("-");
6. \*\'         printf("''");
7. \*\'         printf("'");
8. \@t          printf(",");
```

```
9. \@\'          printf("''");
10. \@\"          printf("\"");
11. \@[a-su-zA-Z]  printf("");
12. \#1           printf("");
13. \#2           printf("");
14. \#3           printf("");
15. \#4           printf("");
16. \#5           printf("");
17. \#6           printf("");
18. \#7           printf("");
19. \#8           printf("");
20. \#9           printf("");
21. \}            printf("");
22. \{            printf("");
23. \#8[0-9][0-9][0-9][0-9][0-9]      printf("");
24. \#\#          printf("");
25. \*[\.]        printf(".");
26. \*[u\,\¬\"\¬\_]   printf("");
27. \@t\#{m}\@t    printf(" X ");
28. \@t\#\+\@t    printf(" + ");
29. \@t\#\-\@t    printf(" - ");
30. \@t\=\@t      printf(" = ");
31. \'\'          printf("%s", yytext);
32. \'           { if (lparn_flg == 1) {  /* print double quotes */
              printf("%s", yytext);
              printf("%s", yytext);
              }
            else printf("%s", yytext); }
33. {nl}\.[0-9]+({bl})?{nl}{d}{e}{f}     { printf("\nDEF");
                                 g_flg = 0; }
34. (\#5)?\%1({bl}\#5)?            { if (lparn_flg == 0)
                                 printf(" \' ,1,");
                               else printf("%s", yytext);
                               num_flg = 1; }
35. (\#5)?\%2({bl}\#5)?            { if (lparn_flg == 0)
                                 printf(" \' ,2,");
                               else printf("%s", yytext);
                               num_flg = 1; }
36. (\#5)?\%3({bl}\#5)?            { if (lparn_flg == 0)
                                 printf(" \' ,3,");
                               else printf("%s", yytext);
                               num_flg = 1; }
37. (\#5)?\%4({bl}\#5)?            { if (lparn_flg == 0)
                                 printf(" \' ,4,");
                               else printf("%s", yytext);
                               num_flg = 1; }
38. (\#5)?\%5({bl}\#5)?            { if (lparn_flg == 0)
                                 printf(" \' ,5,");
                               else printf("%s", yytext);
                               num_flg = 1; }
39. (\#5)?\%6({bl}\#5)?            { if (lparn_flg == 0)
```

```
                                  printf(" \' ,6,");
                                else printf("%s", yytext);
                                num_flg = 1; }
40. (\#5)?\%7({bl}\#5)?           { if (lparn_flg == 0)
                                  printf(" \' ,7,");
                                else printf("%s", yytext);
                                num_flg = 1; }
41. (\#5)?\%8({bl}\#5)?           { if (lparn_flg == 0)
                                  printf(" \' ,8,");
                                else printf("%s", yytext);
                                num_flg = 1; }
42. (\#5)?\%9({bl}\#5)?           { if (lparn_flg == 0)
                                  printf(" \' ,9,");
                                else printf("%s", yytext);
                                num_flg = 1; }
43. ({bl})?\({s}{e}{n}{s}{e}{s}{bl}({num}(\,|\@\=)({bl})?)+     {
                                senses_flg = 1;
                                }
44. \)                  { if (senses_flg == 1)  senses_flg = 0;
                       else printf("%s", yytext);
                       lparn_flg = 0;
                      }
45. ({bl})?\({s}{e}{n}{s}{e}{bl}1\) { /* if no hom # deflt to 1*/
                                  if (num_flg == 0) printf(" \' ,1,");
                                      num_flg = 0;
                                      printf("1");  }
46. ({bl})?\({s}{e}{n}{s}{e}{bl}2\) { if (num_flg == 0)
                                         printf(" \' ,1,");
                                      num_flg = 0;
                                      printf("2");  }
47. ({bl})?\({s}{e}{n}{s}{e}{bl}3\) { if (num_flg == 0)
                                      printf(" \' ,1,");
                                      num_flg = 0;
                                      printf("3");  }
48. ({bl})?\({s}{e}{n}{s}{e}{bl}4\)    { if (num_flg == 0)
                                      printf(" \' ,1,");
                                      num_flg = 0;
                                      printf("4");  }
49. ({bl})?\({s}{e}{n}{s}{e}{bl}5\)    { if (num_flg == 0)
                                      printf(" \' ,1,");
                                      num_flg = 0;
                                      printf("5");  }
50. ({bl})?\({s}{e}{n}{s}{e}{bl}6\)    { if (num_flg == 0)
                                      printf(" \' ,1,");
                                      num_flg = 0;
                                      printf("6");  }
51. ({bl})?\({s}{e}{n}{s}{e}{bl}7\)    { if (num_flg == 0)
                                      printf(" \' ,1,");
                                      num_flg = 0;
                                      printf("7");  }
52. ({bl})?\({s}{e}{n}{s}{e}{bl}8\)    { if (num_flg == 0)
```

```
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("8");  }
53. ({bl})?\({s}{e}{n}{s}{e}{bl}9\)     { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("9");  }
54. ({bl})?\({s}{e}{n}{s}{e}{bl}10\)    { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("10");  }
55. ({bl})?\({s}{e}{n}{s}{e}{bl}11\)    { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("11");  }
56. ({bl})?\({s}{e}{n}{s}{e}{bl}12\)    { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("12");  }
57. ({bl})?\({s}{e}{n}{s}{e}{bl}13\)    { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("13");  }
58. ({bl})?\({s}{e}{n}{s}{e}{bl}14\)    { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("14");  }
59. ({bl})?\({s}{e}{n}{s}{e}{bl}15\)    { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("15");  }
60. ({bl})?\({s}{e}{n}{s}{e}{bl}16\)    { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("16");  }
61. ({bl})?\({s}{e}{n}{s}{e}{bl}17\)    { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("17");  }
62. ({bl})?\({s}{e}{n}{s}{e}{bl}18\)    { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("18");  }
63. ({bl})?\({s}{e}{n}{s}{e}{bl}19\)    { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
                                         printf("19");  }
64. ({bl})?\({s}{e}{n}{s}{e}{bl}20\)    { if (num_flg == 0)
                                         printf(" \' ,1,");
                                         num_flg = 0;
```

```
                                          printf("20");  }
65. ({bl})?\({s}{e}{n}{s}{e}{bl}21\)       { if (num_flg == 0)
                                          printf(" \' ,1,");
                                          num_flg = 0;
                                          printf("21");  }
66. ({bl})?\({s}{e}{n}{s}{e}{bl}22\)       { if (num_flg == 0)
                                          printf(" \' ,1,");
                                          num_flg = 0;
                                          printf("22");  }
67. ({bl})?\({s}{e}{n}{s}{e}{bl}23\)       { if (num_flg == 0)
                                          printf(" \' ,1,");
                                          num_flg = 0;
                                          printf("23");  }
68. ({bl})?\({s}{e}{n}{s}{e}{bl}24\)       { if (num_flg == 0)
                                          printf(" \' ,1,");
                                          num_flg = 0;
                                          printf("24");  }
69. ({bl})?\({s}{e}{n}{s}{e}{bl}25\)       { if (num_flg == 0)
                                          printf(" \' ,1,");
                                          num_flg = 0;
                                          printf("25");  }
70. ({bl})?\({s}{e}{n}{s}{e}{bl}26\)       { if (num_flg == 0)
                                          printf(" \' ,1,");
                                          num_flg = 0;
                                          printf("26");  }
71. ({bl})?\({s}{e}{n}{s}{e}{bl}27\)       { if (num_flg == 0)
                                          printf(" \' ,1,");
                                          num_flg = 0;
                                          printf("27");  }
72. ({bl})?\({s}{e}{n}{s}{e}{bl}28\)       { if (num_flg == 0)
                                          printf(" \' ,1,");
                                          num_flg = 0;
                                          printf("28");  }
73. ({bl})?\({s}{e}{n}{s}{e}{bl}29\)       { if (num_flg == 0)
                                          printf(" \' ,1,");
                                          num_flg = 0;
                                          printf("29");  }
74. [C][1-9]([0-9])?\:[¬\n]*\n         printf("\n");
75. \[[¬\n]*\n                     printf("\n");
76. \{\#1                          printf("");
77. \#1\{                          printf("");
78. \.{bl}\#5                      printf(".\n");
79. \.\#5                          printf(".\n");
80. \;{bl}\#5                      printf(".\n");
81. \;\#5                          printf(".\n");
82. \(                             { if (g_flg == 1)
                                    printf("\n(");
                                 else printf("(");
                                 lparn_flg = 1; }
83. \.\(                           { printf(".\n(");
```

```
                                    lparn_flg = 1;  }
84.  \.{bl}\#5\(                      { printf(".\n(");
                                    lparn_flg = 1;  }
85.  \.\#5\(                          { printf(".\n(");
                                    lparn_flg = 1;  }
86.  [S][U][B][D][E][F][A]({bl})?\n[D][E][F]{bl}       printf("\nSUBDEFA ");
87.  [S][U][B][D][E][F][B]({bl})?\n[D][E][F]{bl}       printf("\nSUBDEFB ");
88.  [S][U][B][D][E][F][C]({bl})?\n[D][E][F]{bl}       printf("\nSUBDEFC ");
89.  [S][U][B][D][E][F][D]({bl})?\n[D][E][F]{bl}       printf("\nSUBDEFD ");
90.  [S][U][B][D][E][F][E]({bl})?\n[D][E][F]{bl}       printf("\nSUBDEFE ");
91.  [S][U][B][D][E][F][F]({bl})?\n[D][E][F]{bl}       printf("\nSUBDEFF ");
92.  \.{bl}[a]\.{bl}                    printf("\nSUBDEFA ");
93.  \.{bl}[b]\.{bl}                    printf("\nSUBDEFB ");
94.  \.{bl}[c]\.{bl}                    printf("\nSUBDEFC ");
95.  \.{bl}[d]\.{bl}                    printf("\nSUBDEFD ");
96.  \.{bl}[e]\.{bl}                    printf("\nSUBDEFE ");
97.  \.{bl}[f]\.{bl}                    printf("\nSUBDEFF ");
98.  {r}{e}{l}{a}{t}{e}{d}{bl}{a}{d}{j}\.\:({bl})?      printf("\nRELADJ ");
99.  {a}{b}{b}{r}{e}{v}({s})?\.({bl})?\:               printf("\nABBREV ");
100.  {a}{b}{b}{r}{e}{v}({i}{a}{t}{i}{o}{n}|\.){bl}{f}{o}{r}({bl})?
        printf("\nPOS abbrev.\nDEF ");
101.  {a}{b}{b}{r}{e}{v}({i}{a}{t}{i}{o}{n}|\.){bl}{f}{o}{r}({bl})?\n
        printf("\nPOS abbrev.\n");
102.  [U]{s}{a}{g}{e}\.({bl})?                        printf("\nUSAGE ");
103.  \.({bl})?{c}{o}{m}{p}{a}{r}{e}{bl}              printf(".\nCOMPARE ");
104.  \.({bl})?{c}{o}{m}{p}{a}{r}{e}{bl}\{\#1         printf(".\nCOMPARE ");
105.  \.({bl})?{c}{o}{m}{p}{a}{r}{e}{bl}\#1\{         printf(".\nCOMPARE ");
106.  \.({bl})?{c}{o}{m}{p}{a}{r}{e}{bl}\([¬\)]*\)(\{|\#1)*
        printf(".\nCOMPARE ");
107.  [S]{e}{e}{bl}({a}{l}{s}{o}{bl})?                printf("\nCOMPARE ");
108.  [S]{e}{e}{bl}({a}{l}{s}{o}{bl})?\{\#1           printf("\nCOMPARE ");
109.  [S]{e}{e}{bl}({a}{l}{s}{o}{bl})?\#1\{           printf("\nCOMPARE ");
110.  {p}{o}{s}{bl}{v}{b}\.{nl}\#5\(\#6{t}{r}\.[¬\)]*\)
        printf("\nPOS vt.\n");
111.  \.({bl})?(\#5)?\(\#6{t}{r}\.[¬\)]*\)      printf(".\nPOS vt.\n");
112.  {p}{o}{s}{bl}{v}{b}\.{nl}\#5\(\#6{i}{n}{t}{r}\.[¬\)]*\)
        printf("\nPOS vi.\n");
113.  \.({bl})?(\#5)?\(\#6{i}{n}{t}{r}\.[¬\)]*\)
        printf(".\nPOS vi.\n");
114.  {p}{o}{s}{bl}{a}{d}{v}\.\,{bl}{a}{d}{j}\.{nl}\#5\(\#6{p}{o}{s}{t}
        {p}{o}{s}{i}{t}{i}{v}{e}  printf("\nPOS adv.\nPOS adjp.\n");
115.  {p}{o}{s}{bl}{a}{d}{j}\.{nl}\#5\(\#6{p}{o}{s}{t}
        {p}{o}{s}{i}{t}{i}{v}{e}[¬\)]*\)
116.  \.({bl})?(\#5)?\(\#6{p}{o}{s}{t}{p}{o}{s}{i}{t}{i}{v}{e}[¬\)]*\)
        printf("\nPOS adjp.\n");
117.  \#5\((\#\!|\@|\#\+|\?\ )[¬\)]*\)          printf("");
118.  \.({bl})?\#5\((\#\!|\@|\#\+)[¬\)]*\)         printf(".");
119.  \#5\(\#6{a}{s}{bl}{a}{d}{j}\.(\#5)?\)(\#6)?\:
        printf("as adj.\nSAMP ");
120.  \#5\(\#6{a}{s}{bl}{a}{d}{j}\.(\#5)?\)(\#6)?\.        printf("as adj.");
```

```
121. \#5\(\#6{a}{s}{bl}{a}{d}{j}\.(\#5)?\)(\#6)?          printf("as adj.");
122. \#5\(\#6({a}{s}{bl})?({s}{e}{n}{t}{e}{n}{c}{e}{bl})?{m}{o}{d}{i}{f}
        {i}{e}{r}(\#5)?\)(\#6)?
123. \#5\(\#6({a}{s}{bl})?({s}{e}{n}{t}{e}{n}{c}{e}{bl})?{m}{o}{d}{i}{f}
        {i}{e}{r}(\#5)?\)(\#6)?
124. \#5\(\#6({a}{s}{bl})?({s}{e}{n}{t}{e}{n}{c}{e}{bl})?{m}{o}{d}{i}{f}
        {i}{e}{r}(\#5)?\)(\#6)?
125. \(\#6({a}{s}{bl})?({s}{e}{n}{t}{e}{n}{c}{e}{bl})?{m}{o}{d}{i}{f}
        {i}{e}{r}(\#5)?\)(\#6)?
                        printf("as modifier");  }
126. \#5\((\#1|\#6)[¬\)]*\)(\:|\.)?       { printf("");
                            g_flg = 1;  }
127. \.({bl})?\#5\(\#6[¬\)]*\)(\:|\.)?     { printf(".");
                                g_flg = 1;  }
128. [D][E][F]        { printf("%s", yytext);
                    senses_flg = 0;
                    lparn_flg = 0;
                    num_flg = 0;
                    g_flg = 0;  }
129. {nl}{s}{y}{l}{l}[¬\n]*{nl}      { printf("\n");
                    printf("%s", yytext);
                    printf("%s", yytext);
                    num_flg = 0;
                    g_flg = 0;  }
130. {nl}[P][L]{bl}[¬\.\n]*(\.|\n)({bl})?      {
                    printf("\n");
                    printf("%s", yytext);
                    printf("\n");}
```

APPENDIX C. PASS3

```
%{
# define YYLMAX 4096
int divsyl_flg=0;  /* divide syllabificated words, 1 word/line */
int pl_flg=0;  /* in plural section   */
int morph_flg=0;  /* in morpholog. var section   */
int abbrev_flg=0;  /* in abbreviation section   */
int syll_flg=0;    /*  in syllabification section  */
int comp_flg=0;    /*  in comparison section  */
int nlparn_flg=0;     /*  left paren, following a newline */
%}
%e 5000
%a 5000
%k 5000
%p 5000
%o 5000
nl      [\n]+
bl      [\ ]+
a       [aA]
b       [bB]
c       [cC]
d       [dD]
e       [eE]
f       [fF]
g       [gG]
h       [hH]
i       [iI]
j       [jJ]
k       [kK]
l       [lL]
m       [mM]
n       [nN]
o       [oO]
p       [pP]
r       [rR]
s       [sS]
t       [tT]
u       [uU]
v       [vV]
w       [wW]
x       [xX]
y       [yY]
%%
1. \%           printf(" ¬ ");
2. \n({bl})?[a-z]             { printf("\nDDEF");
                             printf("%s", yytext);
                             syll_flg = 0;
                             divsyl_flg = 0;
```

```
                                  pl_flg = 0;  }
3.  \n({bl})?[A-Z]/[a-z]            { printf("\nDDEF");
                                  printf("%s", yytext);
                                  syll_flg = 0;
                                  divsyl_flg = 0;
                                  pl_flg = 0;  }
4.  \n({bl})?[0-9]                 { printf("\nDDEF");
                                  printf("%s", yytext);
                                  syll_flg = 0;
                                  divsyl_flg = 0;
                                  pl_flg = 0;  }
5.  [a]\&                printf("a!");
6.  [b]\&                printf("b!");
7.  [c]\&                printf("c!");
8.  [d]\&                printf("d!");
9.  [e]\&                printf("e!");
10.  [f]\&               printf("f!");
11.  [g]\&               printf("g!");
12.  [h]\&               printf("h!");
13.  [i]\&               printf("i!");
14.  [j]\&               printf("j!");
15.  [k]\&               printf("k!");
16.  [l]\&               printf("l!");
17.  [m]\&               printf("m!");
18.  [n]\&               printf("n!");
19.  [o]\&               printf("o!");
20.  [p]\&               printf("p!");
21.  [q]\&               printf("q!");
22.  [r]\&               printf("r!");
23.  [s]\&               printf("s!");
24.  [t]\&               printf("t!");
25.  [u]\&               printf("u!");
26.  [v]\&               printf("v!");
27.  [w]\&               printf("w!");
28.  [x]\&               printf("x!");
29.  [y]\&               printf("y!");
30.  [z]\&               printf("z!");
31.  \_                  {  if (syll_flg == 1)
                            printf("");
                         else printf("%s", yytext);
                      }
32.  \ \'\ \,             {   /* leave ' with ones having homonym # */
                        if ((syll_flg == 1)||(comp_flg == 1))
                           printf("%s", yytext);
                        else printf(" ,");
                      }
33.  \'                   {
                        printf("\'\'");
                      }
34.  ({nl})?\,{bl}        { if (pl_flg == 1)
                           printf("\nPL ");
```

```
                    else if (divsyl_flg == 1)
                       printf("\nSYLL ");
                    else if (comp_flg == 1)
                       printf("\nCOMPARE ");
                    else if (syll_flg == 1)
                       printf("\nOWORD ");
                    else if (morph_flg == 1)
                       printf("\nMORPH ");
                    else if (abbrev_flg == 1)
                       printf("\nABBREV ");
                    else printf("%s", yytext);
                  }
35. (\,)?({nl}|{bl}){o}{r}{bl}  { if (pl_flg == 1)
                       printf("\nPL ");
                    else if (divsyl_flg == 1)
                       printf("\nSYLL ");
                    else if (comp_flg == 1)
                       printf("\nCOMPARE ");
                    else if (syll_flg == 1)
                       printf("\nOWORD ");
                    else if (morph_flg == 1)
                       printf("\nMORPH ");
                    else if (abbrev_flg == 1)
                       printf("\nABBREV ");
                    else printf("%s", yytext);
                  }
36. \.               { if (pl_flg == 1) pl_flg = 0;
                    else printf("%s", yytext);
                  }
37. {bl}[0-9]        { if (divsyl_flg == 1)
                       printf("");
                    else printf("%s", yytext);
                  }
38. {bl}\([¬\)]+\)   { if (pl_flg == 1)
                       printf("\n");
                    printf("%s", yytext);
                  }
39. (\.)?\([¬\)]+\)       { if (pl_flg == 1)
                       printf("");
                    else printf("%s", yytext);
                  }
40. {nl}\(           { printf("\nDEF (");
                    divsyl_flg = 0;
                    syll_flg = 0;
                    comp_flg = 0;
                    pl_flg = 0;
                    abbrev_flg = 0;
                    morph_flg = 0;
                    nlparn_flg = 1;   }
41. \){nl}[D][E][F]{bl}     {
                    comp_flg = 0;
```

```
                    syll_flg = 0;
                    divsyl_flg = 0;
                    pl_flg = 0;
                    abbrev_flg = 0;
                    morph_flg = 0;
                    if (nlparn_flg == 1){
                       printf(") ");
                       nlparn_flg = 0;   }
                    else  {
                       printf(")\nDEF ");
                       nlparn_flg = 0;   }
                 }
42.  {nl}[D][E][F]{bl}      {
                      comp_flg = 0;
                      syll_flg = 0;
                      divsyl_flg = 0;
                      pl_flg = 0;
                      morph_flg = 0;
                      abbrev_flg = 0;
                      printf("\nDEF ");
                      }
43.  {nl}[S][U][B][D][E][F][A-F]{bl}      {
                      comp_flg = 0;
                      syll_flg = 0;
                      divsyl_flg = 0;
                      pl_flg = 0;
                      morph_flg = 0;
                      abbrev_flg = 0;
                      printf("%s", yytext);
                      }
44.  {nl}[P][L]         { pl_flg = 1;
                  printf("%s", yytext);
                  divsyl_flg = 0;
               }
45.  {nl}[P][O][S]        { printf("%s", yytext);
                    comp_flg = 0;
                    syll_flg = 0;
                    divsyl_flg = 0;   }
46.  {nl}[M][O][R][P][H]   { printf("%s", yytext);
                      morph_flg = 1;
                    }
47.  {nl}[A][B][B][R][E][V]   { printf("%s", yytext);
                         abbrev_flg = 1;
                       }
48.  {nl}[S][Y][L][L]     { nlparn_flg = 0;
                     comp_flg = 0;
                     abbrev_flg = 0;
                     morph_flg = 0;
                     if (syll_flg == 0) {
                       syll_flg = 1;
                       printf("\n");
```

```
                        printf("\nHDWORD"); }
                     else if (syll_flg == 1) {
                        syll_flg = 0;
                        divsyl_flg = 1;
                        printf("%s", yytext); }
                   }
49.  {nl}[A][L][S][O]{bl}[¬\,\.\n]*(\,)    { printf("\n");
                                  printf("%s", yytext);
                                  printf("\n");
                                  printf("ALSO"); }
50.  {nl}[U][S][A][G][E]({bl})?\n       {
                                  printf("");
                                 }
51.  {nl}[C][O][M][P][A][R][E]{bl}      {
                                  comp_flg = 1;
                                  printf("%s", yytext);
                                 }
52.  {nl}[C][O][M][P][A][R][E]{bl}[¬\,\n]*(\,)[1-9][\,][1-9][\,]
                                 { printf("\n");
                                  comp_flg = 1;
                                  printf("%s", yytext);
                                  printf("\n");
                                  printf("COMPARE"); }
53.  {nl}[C][O][M][P][A][R][E]{bl}[¬\,\n]*(\,)[1-9][\,][¬1-9]    {
printf("\n");
                                  comp_flg = 1;
                                  printf("%s", yytext);
                                  printf("\n");
                                  printf("COMPARE"); }
54.  {nl}[F][I][R][S][T]{bl}[¬\.\,\n]*(\.|\,|\n)({bl})?
{printf("\n");
                                  printf("%s", yytext);
                                  printf("\n");  }
55.  (\ |\n)+[D][C][A][T][E][G][O][R][Y]   {
                                  printf("");
                                  }
56.  {nl}[C][A][T][E][G][O][R][Y]{bl}[a-z]\.    { /* rid of sense lett. */
                                  printf("\n");
                                  }
57.  {nl}[C][A][T][E][G][O][R][Y]{bl}\,{bl}    { printf("\n");
                                      printf("POS ");
                                     }
58.  {nl}[C][A][T][E][G][O][R][Y]{bl}([a-zA-Z]+\ )?
                                  (\,{bl})?([A-Z]\.)([A-Z]\.) {
                                  comp_flg = 0;
                                  printf("\n");
                                  printf("%s", yytext);
                                  printf("\n");  }
59.  {nl}[C][A][T][E][G][O][R][Y]{bl}([a-zA-Z]+\ )?([A-Z]\.)?([A-Z]\.\
                                  [a-zA-Z]+\.) {
                                  comp_flg = 0;
```

```
                                     printf("\n");
                                     printf("%s", yytext);
                                     printf("\n");  }
60. {nl}[C][A][T][E][G][O][R][Y]{bl}([a-zA-Z]+\ )?(\,\ )?([A-Z]\.)?
                                     ([A-Z]\.\ [a-zA-Z]+\.)\, {
                                     comp_flg = 0;
                                     printf("\n"); /* more than 1 cat.*/
                                     printf("%s", yytext);
                                     printf("\n");
                                     printf("CATEGORY");  }
61. {nl}[C][A][T][E][G][O][R][Y]{bl}([a-zA-Z]+\.)(\ [a-zA-Z]+\.)? {
                                     comp_flg = 0;
                                     printf("\n");
                                     printf("%s", yytext);
                                     printf("\n");  }
62. {nl}[C][A][T][E][G][O][R][Y]{bl}([a-zA-Z]+\.)(\ [a-zA-Z]+\.)?\, {
                                     comp_flg = 0;
                                     printf("\n"); /* more than 1 cat.*/
                                     printf("%s", yytext);
                                     printf("\n");
                                     printf("CATEGORY");  }
63. [C][A][T][E][G][O][R][Y]{bl}[A-Z][a-zA-Z\-]*{bl}[a-zA-Z\-]+
                              (\.)?({bl})?
                              {
                                comp_flg = 0;
                                printf("%s", yytext);
                                printf("\n");
                              }
64. [C][A][T][E][G][O][R][Y]{bl}[A-Z][a-zA-Z\-]*{bl}[a-zA-Z\-]+(\,)?{bl}
                              [a-zA-Z\-]+(\.)?({bl})?
                              {
                                comp_flg = 0;
                                printf("%s", yytext);
                                printf("\n");
                              }
65. [C][A][T][E][G][O][R][Y]{bl}[A-Z][a-zA-Z\-]*{bl}[a-zA-Z\-]+{bl}
                              [a-zA-Z\-]+{bl}[a-zA-Z]+(\.)?({bl})?
                              {
                                comp_flg = 0;
                                printf("%s", yytext);
                                printf("\n");
                              }
66. [C][A][T][E][G][O][R][Y]{bl}[a-z][a-zA-Z\-]*{bl}[a-zA-Z\-]+{bl}
                              [a-zA-Z\-]+(\.)?({bl})?
                              { printf("D");
                                printf("%s", yytext);
                                printf("\n");
                                comp_flg = 0;
                              }
67. [C][A][T][E][G][O][R][Y]{bl}[a-zA-Z\-]+{bl}[a-zA-Z\-]+{bl}[a-zA-Z\-]+
                              ({bl}[a-zA-Z\-]+)+(\.)?({bl})?
```

```
{ printf("D");
  printf("%s", yytext);
  printf("\n");
  comp_flg = 0;
}
```

APPENDIX D. PASS4

```
%{
# define YYLMAX 4096
int syll_flg=0;    /*  in syllabification section  */
int morph_flg=0;    /*  in morphological section  */
int cat_flg=0;    /*  in category section  */
int dcat_flg=0;    /*  in dcategory section  */
int also_flg=0;    /*  in also section  */
int pos_flg=0;    /*  in part of speech section  */
int abbrev_flg=0;    /*  in abbreviation section  */
%}
%e 5000
%a 5000
%n 5000
%k 5000
%p 5000
%o 5000
nl      [\n]+
bl      [\ ]+
%%
1.  \n({bl})?[0-9]          { printf("\nDDEF");
                             printf("%s", yytext);
                             morph_flg = 0;
                             abbrev_flg = 0;
                             cat_flg = 0;
                             also_flg = 0;
                             pos_flg = 0;
                            }
2.  \n({bl})?[a-z]          { printf("\nDDEF");
                             printf("%s", yytext);
                             morph_flg = 0;
                             abbrev_flg = 0;
                             cat_flg = 0;
                             also_flg = 0;
                             pos_flg = 0;
                            }
3.  \n({bl})?[A-Z]/[a-z]    { printf("\nDDEF");
                             printf("%s", yytext);
                             morph_flg = 0;
                             abbrev_flg = 0;
                             cat_flg = 0;
                             also_flg = 0;
                             pos_flg = 0;
                            }
4.  ({nl})?\,{bl}       {
                        if (morph_flg == 1)    {
                           printf("\nMORPH ");    }
                        else if (abbrev_flg == 1)    {
                           printf("\nABBREV ");    }
```

```
                     else if (cat_flg == 1)    {
                        printf("\nCATEGORY ");    }
                     else if (also_flg == 1)   {
                        printf("\nALSO ");    }
                     else if (dcat_flg == 1)    {
                        printf(", ");    }
                     else if (pos_flg == 1)    {
                        printf("\nPOS ");    }
                     else printf("%s", yytext);
                     morph_flg = 0;
                     abbrev_flg = 0;
                     cat_flg = 0;
                     also_flg = 0;
                     dcat_flg = 0;
                     pos_flg = 0;
                    }
5. (\,)?({nl}|{bl})[o][r]{bl}    {
                     if (morph_flg == 1)   {
                        printf("\nMORPH ");    }
                     else if (abbrev_flg == 1)   {
                        printf("\nABBREV ");    }
                     else if (cat_flg == 1)    {
                        printf("\nCATEGORY ");    }
                     else if (also_flg == 1)   {
                        printf("\nALSO ");    }
                     else if (pos_flg == 1)    {
                        printf("\nPOS ");    }
                     else printf("%s", yytext);
                     pos_flg = 0;
                     morph_flg = 0;
                     abbrev_flg = 0;
                     cat_flg = 0;
                     also_flg = 0;
                    }
6. {nl}\;              {
                     printf(";");
                    }
7. \.({bl})?\n\)       printf(".)");
8. ({bl})?\n\)         printf(")");
9. {nl}[D][E][F]({bl}|\.)*\n  {
                         printf("\n");
                         syll_flg = 0;
                         morph_flg = 0;
                         abbrev_flg = 0;
                         cat_flg = 0;
                         also_flg = 0;
                         }
10. {nl}[S][U][B][D][E][F][A-F]{bl}     {
                         syll_flg = 0;
                         pos_flg = 0;
                         morph_flg = 0;
```

```
                         abbrev_flg = 0;
                         cat_flg = 0;
                         also_flg = 0;
                         printf("%s", yytext);
                         }
11.  {nl}[D][E][F]{bl}      {
                         syll_flg = 0;
                         pos_flg = 0;
                         morph_flg = 0;
                         abbrev_flg = 0;
                         cat_flg = 0;
                         also_flg = 0;
                         printf("%s", yytext);
                         }
12.  {nl}[D][D][E][F]({bl})?{nl}     {
                         syll_flg = 0;
                         pos_flg = 0;
                         morph_flg = 0;
                         abbrev_flg = 0;
                         cat_flg = 0;
                         also_flg = 0;
                         printf("\nDEF ");
                         }
13.  {nl}[S][U][B][D][E][F][A]({bl})?{nl}([A][L][S][O]|[D][E][F]){bl}        {
                         syll_flg = 0;
                         pos_flg = 0;
                         morph_flg = 0;
                         abbrev_flg = 0;
                         cat_flg = 0;
                         also_flg = 0;
                         printf("\nSUBDEFA ");
                         }
14.  {nl}[S][U][B][D][E][F][B]({bl})?{nl}([A][L][S][O]|[D][E][F]){bl}        {
                         syll_flg = 0;
                         pos_flg = 0;
                         morph_flg = 0;
                         abbrev_flg = 0;
                         cat_flg = 0;
                         also_flg = 0;
                         printf("\nSUBDEFB ");
                         }
15.  {nl}[S][U][B][D][E][F][C]({bl})?{nl}([A][L][S][O]|[D][E][F]){bl}        {
                         syll_flg = 0;
                         pos_flg = 0;
                         morph_flg = 0;
                         abbrev_flg = 0;
                         cat_flg = 0;
                         also_flg = 0;
                         printf("\nSUBDEFC ");
                         }
16.  {nl}[S][U][B][D][E][F][D]({bl})?{nl}([A][L][S][O]|[D][E][F]){bl}        {
```

```
                    syll_flg = 0;
                    pos_flg = 0;
                    morph_flg = 0;
                    abbrev_flg = 0;
                    cat_flg = 0;
                    also_flg = 0;
                    printf("\nSUBDEFD ");
                    }
17.  {nl}[S][U][B][D][E][F][E]({bl})?{nl}([A][L][S][O]|[D][E][F]){bl}      {
                    syll_flg = 0;
                    morph_flg = 0;
                    pos_flg = 0;
                    abbrev_flg = 0;
                    cat_flg = 0;
                    also_flg = 0;
                    printf("\nSUBDEFE ");
                    }
18.  {nl}[S][U][B][D][E][F][F]({bl})?{nl}([A][L][S][O]|[D][E][F]){bl}      {
                    syll_flg = 0;
                    pos_flg = 0;
                    morph_flg = 0;
                    abbrev_flg = 0;
                    cat_flg = 0;
                    also_flg = 0;
                    printf("\nSUBDEFF ");
                    }
19.  {nl}[S][U][B][D][E][F][A]({bl})?\n  {
                    syll_flg = 0;
                    pos_flg = 0;
                    morph_flg = 0;
                    abbrev_flg = 0;
                    cat_flg = 0;
                    also_flg = 0;
                    }
20.  {nl}[S][Y][L][L]        {
                       syll_flg = 1;
                       printf("%s", yytext);
                       morph_flg = 0;
                       abbrev_flg = 0;
                       cat_flg = 0;
                       also_flg = 0;
                    }
21.  {nl}[H][D][W][O][R][D]{bl}[¬\n]*     {  /* default to sense 1 */
                       printf("%s", yytext);
                       printf("',1\n");
                       morph_flg = 0;
                       abbrev_flg = 0;
                       cat_flg = 0;
                       also_flg = 0;
                       pos_flg = 0;
```

```
                              }
22. [P][O][S]({bl})?\n    {
                        syll_flg = 0;
                        printf("");
                        }
23. [P][O][S]{bl}[a-z]+\.({bl})?     {
                        pos_flg = 1;
                        syll_flg = 0;
                        printf("%s", yytext);
                        printf("\n");
                        }
24. [M][O][R][P][H]     {
                        printf("%s", yytext);
                        morph_flg = 1;
                        }
25. [C][O][M][P][A][R][E]({bl})?(\.|\,)?({bl})?\n  {
                                   syll_flg = 0;
                                   printf("");
                                  }
26. [C][O][M][P][A][R][E][¬\n]*    {  /* default to always put ',1 */
                            syll_flg = 0;
                            printf("%s", yytext);
                            printf("',1");  /* to delete ,1 later */
                           }
27. [A][B][B][R][E][V]      {
                            abbrev_flg = 1;
                            printf("%s", yytext);
                           }
28. [A][L][S][O]      {
                       also_flg = 1;
                       printf("%s", yytext);
                      }
29. {nl}[A][L][S][O]({bl})*{nl}   {
                        printf("\n");
                        syll_flg = 0;
                        morph_flg = 0;
                        abbrev_flg = 0;
                        cat_flg = 0;
                        also_flg = 0;
                        }
30. (\n)+[D][C][A][T][E][G][O][R][Y]{bl}   {
                                abbrev_flg = 0;
                                cat_flg = 0;
                                dcat_flg = 1;
                                also_flg = 0;
                                if (syll_flg == 1) printf("\nDEF ");
                                else if (morph_flg == 1)
                                   printf("\nPOS ");
                                else printf("\n");
                                }
31. [C][A][T][E][G][O][R][Y]{bl}[a-z][a-zA-Z\-]*{bl}[a-zA-Z\-]+{bl}
```

```
                         [a-zA-Z\-]+(\.)?({bl})?
                         { printf("D");
                           printf("%s", yytext);
                           printf("\n");
                           syll_flg = 0;
                           morph_flg = 0;
                           cat_flg = 1;
                         }
32. [C][A][T][E][G][O][R][Y]{bl}[a-zA-Z\-]+{bl}[a-zA-Z\-]+{bl}[a-zA-Z\-]+
                         ({bl}[a-zA-Z\-]+)+(\.)?({bl})?
                         { printf("D");
                           printf("%s", yytext);
                           printf("\n");
                           syll_flg = 0;
                           morph_flg = 0;
                           cat_flg = 1;
                         }
33. {nl}[C][A][T][E][G][O][R][Y]({bl})?\.({bl})?\n  {
                           pos_flg = 0;
                           syll_flg = 0;
                           morph_flg = 0;
                           printf("\n");
                           }
34. {nl}[C][A][T][E][G][O][R][Y]  {
                            syll_flg = 0;
                            morph_flg = 0;
                            cat_flg = 1;
                            printf("%s", yytext);
                           }
```

APPENDIX E. PASS5

```
%{
# define YYLMAX 4096
int divsyl_flg=0;  /* divide syllabificated words, 1 word/line */
int syll_flg=0;    /* in syllabification section */
int comp_flg=0;    /* in compare section */
int also_flg=0;    /* in also section */
int abbrev_flg=0;  /* in abbreviation section */
int cat_flg=0;     /* in category section */
int com_num=0;     /* number of commas in headword part */
int comp_num=0;    /* number of commas in compare part */
int hdword_flg=0;  /* in headword section */
%}
%e 5000
%a 20000
%n 5000
%k 5000
%p 20000
%o 50000
nl      [\n]+
bl      [\ ]+
%%
1. \#\!                 {
                        printf("!");
                       }
2. \@\!                 {
                        printf(" ");
                       }
3. \n({bl})?[0-9]             { printf("\nDDEF");
                              printf("%s", yytext);
                             }
4. \n({bl})?[A-Z]/[a-z]       { printf("\nDDEF");
                              printf("%s", yytext);
                             }
5. \n({bl})?[a-z]             { printf("\nDDEF");
                              printf("%s", yytext);
                             }
6. \'\,[0-9]            { if (divsyl_flg == 1) {
                          printf("");
                          divsyl_flg = 0; }
                        else if (hdword_flg == 1) {
                          if (com_num == 1)
                             printf("");
                          else   {
                             printf("%s", yytext);
                             com_num = 1; }
                          }
                        else if (comp_flg == 1) {
                          if (comp_num == 1)
```

```
                    printf("");
                 else   {
                    printf("\'");
                    comp_num = 1;  }
                 }
              else
                 printf("%s", yytext);
           }
7. \'\ \,[0-9]      { if (divsyl_flg == 1) {
                 printf("");
                 divsyl_flg = 0;  }
              else if (hdword_flg == 1) {
                    printf("%s", yytext);
                    com_num = 1;  }
              else if (comp_flg == 1) {
                    printf("%s", yytext);
                    comp_num = 1;  }
              else
                 printf("%s", yytext);
           }
8. \,[0-9]          { if (divsyl_flg == 1) {
                 printf("");
                 divsyl_flg = 0;
                 }
              else
                 printf("%s", yytext);
           }
9. {nl}[D][E][F]{bl}     {
                 syll_flg = 0;
                 divsyl_flg = 0;
                 printf("%s", yytext);
                 }
10. [D][D][E][F]({bl})?{nl}     {
                 syll_flg = 0;
                 divsyl_flg = 0;
                 printf("\nDEF ");
                 }
11. {nl}[S][Y][L][L]{bl}[¬\,/[0-9]]* { /* get rid of syll word 1-9 */
                    divsyl_flg = 1;
                    hdword_flg = 0;
                    com_num = 0;
                    comp_num = 0;
                    printf("%s", yytext);
              }
12. {nl}[S][Y][L][L]{bl}[¬\-\,/[0-9]]* { /* get rid of syll word 1-9 */
                    divsyl_flg = 1;
                    hdword_flg = 0;
                    com_num = 0;
                    comp_num = 0;
                    printf("%s", yytext);
```

```
                            }
13. {nl}[H][D][W][O][R][D]{bl}     {
                                hdword_flg = 1;
                                divsyl_flg = 0;
                                abbrev_flg = 0;
                                cat_flg = 0;
                                comp_flg = 0;
                                com_num = 0;
                                comp_num = 0;
                                printf("%s", yytext);
                                printf("'");
                            }
14. {nl}[A][B][B][R][E][V]({bl})?\n[M][O][R][P][H]({bl})?     {
                                divsyl_flg = 0;
                                printf("\nABBREV ");
                            }
15. {nl}[C][O][M][P][A][R][E]({bl})?\n[M][O][R][P][H]({bl})?     {
                                comp_flg = 1;
                                comp_num = 0;
                                divsyl_flg = 0;
                                printf("\nCOMPARE '");
                            }
16. {nl}[C][O][M][P][A][R][E]{bl}     {
                                comp_flg = 1;
                                comp_num = 0;
                                divsyl_flg = 0;
                                printf("%s", yytext);
                                printf("'");
                            }
17. {nl}[C][O][M][P][A][R][E]{bl}\n     {
                                printf("");
                            }
18. {nl}[M][O][R][P][H]{bl}(\.|{bl})*\n     {
                                printf("\n");
                            }
19. {nl}[A][L][S][O]{bl}     {
                                also_flg = 1;
                                divsyl_flg = 0;
                                printf("%s", yytext);
                            }
20. {nl}[C][A][T][E][G][O][R][Y]{bl}[Cc][h][i][e][f][l][y]({bl})?\n
                            [U]\.[S]\.     {
                                cat_flg = 1;
                                divsyl_flg = 0;
                                printf("\nCATEGORY Chiefly U.S.");
                            }
21. {nl}[C][A][T][E][G][O][R][Y]{bl}     {
                                cat_flg = 1;
                                divsyl_flg = 0;
                                printf("%s", yytext);
```

```
                                }
22.  {nl}[A][B][B][R][E][V]{bl}      {
                                 abbrev_flg = 1;
                                 divsyl_flg = 0;
                                 printf("%s", yytext);
                                }
```

APPENDIX F. PASS6

```
%{
# define YYLMAX 4096
int comp_flg=0;    /* in compare section */
int also_flg=0;    /* in also section */
int abbrev_flg=0;    /* in abbreviation section */
int cat_flg=0;    /* in category section */
int hdword_flg=0;    /* in headword section */
%}
%e 5000
%a 20000
%n 5000
%k 5000
%p 20000
%o 50000
nl      [\n]+
bl      [\ ]+
%%
1. [D][D][E][F]({bl})?{nl}      {
                     printf("\nDEF ");
                     }
2. [C][O][M][P][A][R][E]{bl}    {
                     comp_flg = 1;
                     hdword_flg = 0;
                     printf("%s", yytext);
                     }
3. [H][D][W][O][R][D]{bl}       {
                     hdword_flg = 1;
                     comp_flg = 0;
                     abbrev_flg=0;
                     cat_flg=0;
                     also_flg=0;
                     printf("%s", yytext);
                     }
4. \.({bl})?\'
                       {
                     if (comp_flg == 1) printf("'");
                     else printf("%s", yytext);
                     }
5. {bl}\'\,
                       {
                     if (hdword_flg == 1) printf("',");
                     else printf("%s", yytext);
                     }
6. \'{bl}
                       {
                     if (comp_flg == 1) printf("'");
                     else printf("%s", yytext);
                     }
7. {bl}\'{bl}
                       {
                     if ((comp_flg == 1)||(hdword_flg == 1))   {
                        printf("'");
```

```
                        }
                     else printf("%s", yytext);
                     }
8. [A][B][B][R][E][V]{bl}    {
                     abbrev_flg = 1;
                     printf("%s", yytext);
                     }
9. [C][A][T][E][G][O][R][Y]{bl}    {
                     cat_flg = 1;
                     printf("%s", yytext);
                     }
10. \,{nl}               {
                     if (abbrev_flg == 1)  {
                        abbrev_flg = 0;
                        printf(".");
                        }
                     printf("\n");
                     }
11. ({bl})?{nl}          {
                     printf("\n");
                     }
12. (\,)?\.({bl})?{nl}          {
                     if (abbrev_flg == 1)  {
                        printf(".\n");
                        }
                     else if (also_flg == 1)  {
                        printf(".\n");
                        }
                     else if (cat_flg == 1)  {
                        printf(".\n");
                        }
                     else printf("\n");
                     abbrev_flg = 0;
                     cat_flg = 0;
                     also_flg = 0;
                     hdword_flg = 0;
                     comp_flg = 0;
                     }
```

```
%{
# define YYLMAX 4096
int defnum=0;  /* definition number */
int subdefnum=0;  /* sub definition number */
int posnum=0;  /* sense number, second number, changes w/ next POS */
int hdword_flg=0;  /* flag that headword has been seen */
int oword_flg=0;  /* flag that other spell. of headword has been seen */
int morph_flg=0;  /* flag that a morphological var. has been seen */
int o_ct=0;       /* count of the number of owords */
int s_ct=0;       /* count of the number of syllabifications */
int m_ct=0;       /* count of the number of morphological vars. */
int mm_ct=0;       /* count of continuous morphological vars. */
int syll_flg=0;  /* flag that a syllabification has been seen */
int abbrev_flg=0;  /* flag that an abbreviation has been seen */
int pos_flg=0;  /* flag that a part of speech has been seen */
int category_flg=0;  /* flag that a category has been seen */
int def_flg=0;  /* flag that a definition has been seen */
int samp_flg=0;  /* flag that a sample usage has been seen */
int past_flg=0;  /* flag that a past relation has been seen */
int sing_flg=0;  /* flag that a singular relation has been seen */
int also_flg=0;  /* flag that an also has been seen */
int sub_flg=0;  /* flag that a sub definition has been seen */
int usage_flg=0;  /* flag that a usage has been seen */
int reladj_flg=0;  /* flag that a related adjective has been seen */
int first_flg=0;  /* flag that a first has been seen */
int pl_flg=0;  /* flag that a plural has been seen */
int compare_flg=0;  /* flag that a compare has been seen */
char wordarry[50];  /* headword stored in this array */
char owordarry[50];  /* other headword stored in this array */
char oword2arry[50];  /* second other headword stored in this array */
char oword3arry[50];  /* third other headword stored in this array */
char oword4arry[50];  /* fourth other headword stored in this array */
char oword5arry[50];  /* fifth other headword stored in this array */
char oword6arry[50];  /* sixth other headword stored in this array */
char oword7arry[50]; /* seventh other headword stored in this array */
char oword8arry[50];  /* eighth other headword stored in this array */
char oword9arry[50];  /* ninth other headword stored in this array */
char morpharry[50];  /* morphological variant stored in this array */
char morph2arry[50];  /* second morph. stored in this array */
char morph3arry[50];  /* third morph. stored in this array */
char morph4arry[50];  /* fourth morph. stored in this array */
char morph5arry[50];  /* fifth morph. stored in this array */
char morph6arry[50];  /* sixth morph. stored in this array */
char morph7arry[50];  /* seventh morph. stored in this array */
char morph8arry[50];  /* eighth morph. stored in this array */
char morph9arry[50];  /* ninth morph. stored in this array */
%}
%e 5000
```

```
%k 5000
%p 5000
bl  [\ ]+
nl  [\n]+
%%
1. {nl}[H][D][W][O][R][D]{bl}          {
                                    defnum = 0;
                                    posnum = 0;
                                    hdword_flg = 1;
                                    morph_flg = 0;
                                    sub_flg = 0;
                                    o_ct = 0;
                                    m_ct = 0;
                                    mm_ct = 0;
                                    s_ct = 0;
                                    wordarry[0] = '\0';
                                    owordarry[0] = '\0';
                                    oword2arry[0] = '\0';
                                    oword3arry[0] = '\0';
                                    oword4arry[0] = '\0';
                                    oword5arry[0] = '\0';
                                    oword6arry[0] = '\0';
                                    oword7arry[0] = '\0';
                                    oword8arry[0] = '\0';
                                    oword9arry[0] = '\0';
                                    morpharry[0] = '\0';
                                    morph2arry[0] = '\0';
                                    morph3arry[0] = '\0';
                                    morph4arry[0] = '\0';
                                    morph5arry[0] = '\0';
                                    morph6arry[0] = '\0';
                                    morph7arry[0] = '\0';
                                    morph8arry[0] = '\0';
                                    morph9arry[0] = '\0';
                                    }
2. {nl}[O][W][O][R][D]{bl}             {
                                    oword_flg = 1;
                                    o_ct = o_ct + 1;
                                    }
3. {nl}[S][Y][L][L]{bl}                {
                                    syll_flg = 1;
                                    s_ct = s_ct + 1;
                                    }
4. {nl}[C][A][T][E][G][O][R][Y]{bl} {
                                    category_flg = 1;
                                    sub_flg = 0;
                                    }
5. {nl}[P][O][S]{bl}                   {
                                    pos_flg = 1;
                                    sub_flg = 0;
                                    defnum = 0;
```

```
                                        posnum = posnum + 1;
                                        }
6.  {nl}[D][E][F]{bl}                     {
                                        def_flg = 1;
                                        sub_flg = 0;
                                        defnum = defnum + 1;
                                        }
7.  {nl}[S][U][B][D][E][F][A]{bl}         {
                                        def_flg = 1;
                                        sub_flg = 1;
                                        subdefnum = 1;
                                        if (defnum == 0) defnum = 1;
                                        }
8.  {nl}[S][U][B][D][E][F][B]{bl}         {
                                        def_flg = 1;
                                        sub_flg = 1;
                                        subdefnum = 2;
                                        }
9.  {nl}[S][U][B][D][E][F][C]{bl}         {
                                        def_flg = 1;
                                        sub_flg = 1;
                                        subdefnum = 3;
                                        }
10. {nl}[S][U][B][D][E][F][D]{bl}         {
                                        def_flg = 1;
                                        sub_flg = 1;
                                        subdefnum = 4;
                                        }
11. {nl}[S][U][B][D][E][F][E]{bl}         {
                                        def_flg = 1;
                                        sub_flg = 1;
                                        subdefnum = 5;
                                        }
12. {nl}[S][U][B][D][E][F][F]{bl}         {
                                        def_flg = 1;
                                        sub_flg = 1;
                                        subdefnum = 6;
                                        }
13. {nl}[S][A][M][P]{bl}                  {
                                        samp_flg = 1;
                                        }
14. {nl}[M][O][R][P][H]{bl}               {
                                        morph_flg = 1;
                                        defnum = 0;
                                        sub_flg = 0;
                                        posnum = 0;
                                        m_ct = m_ct + 1;
                                        mm_ct = mm_ct + 1;
                                        }
15. {nl}[C][O][M][P][A][R][E]{bl}         {
                                        compare_flg = 1;
```

```
                                            sub_flg = 0;
                                            }
16. {nl}[F][I][R][S][T]{bl}                     {
                                            first_flg = 1;
                                            }
17. {nl}[R][E][L][A][D][J]{bl}                  {
                                            reladj_flg = 1;
                                            }
18. {nl}[P][A][S][T]{bl}                        {
                                            past_flg = 1;
                                            }
19. {nl}[S][I][N][G]{bl}                        {
                                            sing_flg = 1;
                                            }
20. {nl}[U][S][A][G][E]{bl}                     {
                                            usage_flg = 1;
                                            }
21. {nl}[A][L][S][O]{bl}                        {
                                            also_flg = 1;
                                            sub_flg = 0;
                                            }
22. {nl}[A][B][B][R][E][V]{bl}                  {
                                            abbrev_flg = 1;
                                            sub_flg = 0;
                                            }
23. {nl}[P][L]{bl}                              {
                                            pl_flg = 1;
                                            }
24. [¬\n]+                                  {
                                        if (hdword_flg == 1)  {
                                           hdword_flg = 0;
                                           wordarry[0] = '\0';
                                           strcat(wordarry, yytext);
                                           printf("c_HEADWORD([ ");
                                           printf("%s", wordarry);
                                           printf(" ]).\n");   }
                                        else if (oword_flg == 1)  {
                                           oword_flg = 0;
                                           printf("c_VAR_SPELL([ ");
                                           printf("%s", wordarry);
                                           printf(" ], '");
                                           if (o_ct == 1) {
                                              owordarry[0] = '\0';
                                              strcat(owordarry, yytext);
                                              printf("%s", owordarry);
                                              }
                                           else if (o_ct == 2) {
                                              oword2arry[0] = '\0';
                                              strcat(oword2arry, yytext);
                                              printf("%s", oword2arry);
                                              }
```

```
        else if (o_ct == 3) {
           oword3arry[0] = '\0';
           strcat(oword3arry, yytext);
           printf("%s", oword3arry);
           }
        else if (o_ct == 4) {
           oword4arry[0] = '\0';
           strcat(oword4arry, yytext);
           printf("%s", oword4arry);
           }
        else if (o_ct == 5) {
           oword5arry[0] = '\0';
           strcat(oword5arry, yytext);
           printf("%s", oword5arry);
           }
        else if (o_ct == 6) {
           oword6arry[0] = '\0';
           strcat(oword6arry, yytext);
           printf("%s", oword6arry);
           }
        else if (o_ct == 7) {
           oword7arry[0] = '\0';
           strcat(oword7arry, yytext);
           printf("%s", oword7arry);
           }
        else if (o_ct == 8) {
           oword8arry[0] = '\0';
           strcat(oword8arry, yytext);
           printf("%s", oword8arry);
           }
        else if (o_ct == 9) {
           oword9arry[0] = '\0';
           strcat(oword9arry, yytext);
           printf("%s", oword9arry);
           }
        printf("' ).\n");
        }     /* end of else if oword_flg */
     else if (syll_flg == 1)  {
        syll_flg = 0;
        if (s_ct == 1) {
           printf("c_SYLL([ ");
           printf("%s", wordarry);
           }
        else if (s_ct == 2) {
           printf("c_VAR_SYLL([ '");
           printf("%s", owordarry);
           }
        else if (s_ct == 3) {
           printf("c_VAR_SYLL([ '");
           printf("%s", oword2arry);
           }
```

```
    else if (s_ct == 4) {
       printf("c_VAR_SYLL([ '");
       printf("%s", oword3arry);
       }
    else if (s_ct == 5) {
       printf("c_VAR_SYLL([ '");
       printf("%s", oword4arry);
       }
    else if (s_ct == 6) {
       printf("c_VAR_SYLL([ '");
       printf("%s", oword5arry);
       }
    else if (s_ct == 7) {
       printf("c_VAR_SYLL([ '");
       printf("%s", oword6arry);
       }
    else if (s_ct == 8) {
       printf("c_VAR_SYLL([ '");
       printf("%s", oword7arry);
       }
    else if (s_ct == 9) {
       printf("c_VAR_SYLL([ '");
       printf("%s", oword8arry);
       }
    else if (s_ct == 10) {
       printf("c_VAR_SYLL([ '");
       printf("%s", oword9arry);
       }
    if (s_ct == 1) printf(" ],[ '");
    else printf("' ],[ '");
    printf("%s", yytext);
    printf("' ]).\n");
    }   /* end of if syll_flg */
 else if (compare_flg == 1)  {
    compare_flg = 0;
    printf("c_COMPARE([ ");
    printf("%s", wordarry);
    printf(",");
    printf("%d", posnum);
    printf(",");
    if (defnum == 0)
       printf("1");
    else
       printf("%d", defnum);
    printf(" ],[ ");
    printf("%s", yytext);
    printf(" ]).\n");
    }   /* end of if compare_flg */
 else if (past_flg == 1)  {
    past_flg = 0;
    printf("c_PAST([ ");
```

```
printf("%s", wordarry);
printf(",");
printf("%d", posnum);
printf(",");
if (defnum == 0)
   printf("1");
else
   printf("%d", defnum);
printf(" ], '");
printf("%s", yytext);
printf("' ).\n");
}   /* end of if past_flg */
else if (sing_flg == 1)  {
   sing_flg = 0;
   printf("c_SINGULAR([ ");
   printf("%s", wordarry);
   printf(",");
   printf("%d", posnum);
   printf(",");
   if (defnum == 0)
      printf("1");
   else
      printf("%d", defnum);
   printf(" ], '");
   printf("%s", yytext);
   printf("' ).\n");
   }   /* end of if sing_flg */
else if (also_flg == 1)  {
   also_flg = 0;
   printf("c_ALSO_CALLED([ ");
   printf("%s", wordarry);
   printf(",");
   printf("%d", posnum);
   printf(",");
   if (defnum == 0)
      printf("1");
   else
      printf("%d", defnum);
   printf(" ], '");
   printf("%s", yytext);
   printf("' ).\n");
   }   /* end of if also_flg */
else if (usage_flg == 1)  {
   usage_flg = 0;
   printf("c_USAGE([ ");
   printf("%s", wordarry);
   printf(" ],[ '");
   printf("%s", yytext);
   printf("' ]).\n");
   }   /* end of if usage_flg */
else if (reladj_flg == 1)  {
```

```
      reladj_flg = 0;
      printf("c_RELADJ([ ");
      printf("%s", wordarry);
      printf(",");
      printf("%d", posnum);
      printf(",");
      if (defnum == 0)
         printf("1");
      else
         printf("%d", defnum);
      printf(" ],[ '");
      printf("%s", yytext);
      printf("',1 ]).\n");
      }   /* end of if reladj_flg */
   else if (first_flg == 1)  {
      first_flg = 0;
      printf("c_NLAST([ ");
      printf("%s", wordarry);
      printf(",");
      printf("%d", posnum);
      printf(",");
      if (defnum == 0)
         printf("1");
      else
         printf("%d", defnum);
      printf(" ], '");
      printf("%s", yytext);
      printf("' ).\n");
      }   /* end of if first_flg */
   else if (abbrev_flg == 1)  {
      abbrev_flg = 0;
      printf("c_ABBREV([ ");
      printf("%s", wordarry);
      printf(",");
      printf("%d", posnum);
      printf(",");
      if (defnum == 0)
         printf("1");
      else
         printf("%d", defnum);
      printf(" ], '");
      printf("%s", yytext);
      printf("' ).\n");
      }   /* end of if abbrev_flg */
   else if (pl_flg == 1)  {
      pl_flg = 0;
      printf("c_PLURAL([ ");
      printf("%s", wordarry);
      printf(" ], '");
      printf("%s", yytext);
      printf("' ).\n");
```

```
    }   /* end of if pl_flg */
  else if (category_flg == 1)  {
    category_flg = 0;
    printf("c_CATEGORY([ ");
    printf("%s", wordarry);
    printf(",");
    printf("%d", posnum);
    printf(",");
    if (defnum == 0)
      printf("1");
    else
      printf("%d", defnum+1);
    printf(" ], '");
    printf("%s", yytext);
    printf("' ).\n");
    }   /* end of if category_flg */
else if (pos_flg == 1)  {
   pos_flg = 0;
   if (morph_flg == 1)  {
      printf("c_MORPH([ ");
      printf("%s", wordarry);
      printf(" ], '");
      printf("%s", morpharry);
      printf("' , ");
      printf("%s", yytext);
      printf(" ).\n");
      if (mm_ct >= 2) {  /* print consec. */
         printf("c_MORPH([ ");
         printf("%s", wordarry);
         printf(" ], '");
         printf("%s", morph2arry);
         printf("' , ");
         printf("%s", yytext);
         printf(" ).\n");
         }
     if (mm_ct >= 3) {
        printf("c_MORPH([ ");
        printf("%s", wordarry);
        printf(" ], '");
        printf("%s", morph3arry);
        printf("' , ");
        printf("%s", yytext);
        printf(" ).\n");
        }
     if (mm_ct >= 4) {
        printf("c_MORPH([ ");
        printf("%s", wordarry);
        printf(" ], '");
        printf("%s", morph4arry);
        printf("' , ");
        printf("%s", yytext);
```

```
   printf(" ).\n");
   }
if (mm_ct >= 5) {
   printf("c_MORPH([ ");
   printf("%s", wordarry);
   printf(" ], '");
   printf("%s", morph5arry);
   printf("' , ");
   printf("%s", yytext);
   printf(" ).\n");
   }
if (mm_ct >= 6) {
   printf("c_MORPH([ ");
   printf("%s", wordarry);
   printf(" ], '");
   printf("%s", morph6arry);
   printf("' , ");
   printf("%s", yytext);
   printf(" ).\n");
   }
if (mm_ct >= 7) {
   printf("c_MORPH([ ");
   printf("%s", wordarry);
   printf(" ], '");
   printf("%s", morph7arry);
   printf("' , ");
   printf("%s", yytext);
   printf(" ).\n");
   }
if (mm_ct >= 8) {
   printf("c_MORPH([ ");
   printf("%s", wordarry);
   printf(" ], '");
   printf("%s", morph8arry);
   printf("' , ");
   printf("%s", yytext);
   printf(" ).\n");
   }
if (mm_ct >= 9) {
   printf("c_MORPH([ ");
   printf("%s", wordarry);
   printf(" ], '");
   printf("%s", morph9arry);
   printf("' , ");
   printf("%s", yytext);
   printf(" ).\n");
   }
morpharry[0] = '\0';
morph2arry[0] = '\0';
morph3arry[0] = '\0';
morph4arry[0] = '\0';
```

```
          morph5arry[0] = '\0';
          morph6arry[0] = '\0';
          morph7arry[0] = '\0';
          morph8arry[0] = '\0';
          morph9arry[0] = '\0';
          mm_ct = 0;
          m_ct = 0;
          morph_flg = 0;
          }  /* end of if morph_flg */
      else {
            printf("c_POS([ ");
            printf("%s", wordarry);
            printf(",");
            printf("%d", posnum);
            printf(" ], ");
            printf("%s", yytext);
            printf(" ).\n");
            }  /* end of else (not morph) */
      }  /* end of if pos_flg  */
  else if (morph_flg == 1)  {
      if (m_ct == 1)  {
         morpharry[0] = '\0';
         strcat(morpharry, yytext); }
      else if (m_ct == 2)  {
         morph2arry[0] = '\0';
         strcat(morph2arry, yytext); }
      else if (m_ct == 3)  {
         morph3arry[0] = '\0';
         strcat(morph3arry, yytext); }
      else if (m_ct == 4)  {
         morph4arry[0] = '\0';
         strcat(morph4arry, yytext); }
      else if (m_ct == 5)  {
         morph5arry[0] = '\0';
         strcat(morph5arry, yytext); }
      else if (m_ct == 6)  {
         morph6arry[0] = '\0';
         strcat(morph6arry, yytext); }
      else if (m_ct == 7)  {
         morph7arry[0] = '\0';
         strcat(morph7arry, yytext); }
      else if (m_ct == 8)  {
         morph8arry[0] = '\0';
         strcat(morph8arry, yytext); }
      else if (m_ct == 9)  {
         morph9arry[0] = '\0';
         strcat(morph9arry, yytext); }
      }  /* end of if morph_flg */
  else if (def_flg == 1)  {
     def_flg = 0;
     if (posnum == 0)
```

```
                        posnum = 1;  /* in case no POS */
                     printf("c_DEF([ ");
                     printf("%s", wordarry);
                     printf(",");
                     printf("%d", posnum);
                     printf(",");
                     printf("%d", defnum);
                     if (sub_flg == 1) {
                        printf(",");
                        printf("%d", subdefnum); }
                     printf(" ],[ '");
                     printf("%s", yytext);
                     printf("' ]).\n");
                     }  /* end of if def_flg  */
                  else if (samp_flg == 1)  {
                     samp_flg = 0;
                     printf("c_SAMP([ ");
                     printf("%s", wordarry);
                     printf(",");
                     printf("%d", posnum);
                     printf(",");
                     printf("%d", defnum);
                     if (sub_flg == 1) {
                        printf(",");
                        printf("%d", subdefnum); }
                     printf(" ],[ '");
                     printf("%s", yytext);
                     printf("' ]).\n");
                     }  /* end of if samp_flg  */
                  }
25. .          {
             fprintf(stderr, yytext);
            }
```

APPENDIX H. PASS8

```
%{
# define YYLMAX 4096
int usage_flg=0;  /* signals that usage has been seen */
int def_flg=0;  /* signals that definition has been seen */
char begusearry[100];  /* store beginning of the usage line */
char begdefarry[100];  /* store beginning of the definition line */
char usearry[100];     /* store <= 80 char in this usage array */
char defarry[100];     /* store <= 80 char in this defin. array */
int def_num = 0;   /* store number of divisions of 80 char blocks */
int usage_num = 0;   /* store number of divisions of 80 char blocks */
int def_ct = 0;   /* store number of char <= 80 */
int usage_ct = 0;    /* store number of char <= 80 */
int nfirst_flg = 0;  /* done once to insert a ' at beginn. of 2,3.. */
%}
%%
1. \n[c][\_][D][E][F]\([\[]\ [¬\[]*[\[]\      {
                                    def_flg = 1;
                                    usage_flg = 0;
                                    def_num = 1;   /* first is 1 */
                                    def_ct = 0;
                                    nfirst_flg = 0;
                                    begdefarry[0] = '\0';
                                    defarry[0] = '\0';
                                    strcat(begdefarry, yytext);
                                    }
2. \n[c]\_[U][S][A][G][E]\([\[]\ [¬\[]*[\[]\      {
                                    usage_flg = 1;
                                    def_flg = 0;
                                    usage_num = 1;   /* first is 1 */
                                    usage_ct = 0;
                                    nfirst_flg = 0;
                                    begusearry[0] = '\0';
                                    usearry[0] = '\0';
                                    strcat(begusearry, yytext);
                                    }
3. \n[c]\_
                                        {
                                    usage_flg = 0;
                                    def_flg = 0;
                                    printf("%s", yytext);
                                    }
4. \ \]\)\.             {
                        if (def_flg == 1) {
                           printf("%s", begdefarry);
                           if (nfirst_flg == 1) printf("'");
                           printf("%s", defarry);
                           printf(" ],");
                           printf("%d", def_num);
                           printf(" ).\n");
```

```
   printf("NUM");
   printf("%s", begdefarry);
   printf("%d", def_num);
   printf(" ]).\n");
   }
else if (usage_flg == 1)  {
   printf("%s", begusearry);
   if (nfirst_flg == 1) printf("'");
   printf("%s", usearry);
   printf(" ],");
   printf("%d", usage_num);
   printf(" ).\n");
   printf("NUM");
   printf("%s", begusearry);
   printf("%d", usage_num);
   printf(" ]).\n");
   }
else printf("%s", yytext);
}
```

```
5. [¬ \n]*             {
 if (def_flg == 1) {
    def_ct = def_ct + yyleng;
    if (def_ct >= 80)  {
       def_ct = 0;
       printf("%s", begdefarry);
       if (nfirst_flg == 1) printf("'");
       printf("%s", defarry);
       printf("' ],");
       printf("%d", def_num);
       printf(" ).\n");
       def_num = def_num + 1;
       defarry[0] = '\0';
       strcat(defarry, yytext);
       nfirst_flg = 1;
       }
    else  {
       strcat(defarry, " ");
       strcat(defarry, yytext);
       def_ct = def_ct + 1;
       }
    }
 else if (usage_flg == 1)  {
    usage_ct = usage_ct + yyleng;
    if (usage_ct >= 80)  {
       usage_ct = 0;
       printf("%s", begusearry);
       if (nfirst_flg == 1) printf("'");
       printf("%s", usearry);
       printf("' ],");
       printf("%d", usage_num);
       printf(" ).\n");
```

```
            usage_num = usage_num + 1;
            usearry[0] = '\0';
            strcat(usearry, yytext);
            nfirst_flg = 1;
            }
        else  {
            strcat(usearry, " ");
            strcat(usearry, yytext);
            usage_ct = usage_ct + 1;
            }
        }
    else printf("%s", yytext);
}
```

APPENDIX I. PASS9

```
%{
# define YYLMAX 4096
int num_flg = 0;   /* after a defnum or usagenum */
int def_flg = 0;   /* after a def or a samp or usage */
%}
bl    [\ ]+
%%
1. \n[N][U][M]\n[c]\_[D][E][F]          {
                                    printf("\nc_DEF_NUM");
                                    num_flg = 1;
                                    }
2. \n[N][U][M]\n[c]\_[U][S][A][G][E]    {
                                    printf("\nc_USAGE_NUM");
                                    num_flg = 1;
                                    }
3. \]\,\[({bl})?                          {
                                    if ((num_flg == 1)|(def_flg == 1))
                                       printf("] , ");
                                    else printf("%s", yytext);
                                    }
4. \]\)\.                                 {
                                    if ((num_flg == 1)|(def_flg == 1))
                                       printf(").");
                                    else printf("%s", yytext);
                                    num_flg = 0;
                                    }
5. \]\,                                   {
                                    if (def_flg == 1)
                                       printf(",");
                                    else printf("%s", yytext);
                                    def_flg = 0;
                                    }
6. \n[c]\_[D][E][F]                       {
                                    printf("%s", yytext);
                                    def_flg = 1;
                                    }
7. \n[c]\_[S][A][M][P]                    {
                                    printf("%s", yytext);
                                    def_flg = 1;
                                    }
8. \n[c]\_[U][S][A][G][E]                 {
                                    printf("%s", yytext);
                                    def_flg = 1;
                                    }
9. \n[c]\_                                {
                                    printf("%s", yytext);
                                    def_flg = 0;
                                    num_flg = 0;
```

}

BIBLIOGRAPHY

[AHLS 83] Ahlswede, T.E., A Linguistic String Grammar of Adjective Definitions from Webster's Seventh Collegiate Dictionary, In Humans and Machines, S. Williams (ed.) 101-127, 1983.

[AHLS 84] Ahlswede, T.E. and Evens, M., A Lexicon for a Medical Expert System, Presented at the Workshop on Relational Models. In Coling 84.

[AHLS 85] Ahlswede, T. E., A Tool Kit for Lexicon Building, In Proc. of the 23rd Annual Meeting of the ACL, 268-276, July 1985.

[AHOA 74] A.V. Aho and S.C. Johnson, LR Parsing, Computing Surveys 6:2, pp.99-124 (June 1974).

[AHOA 75] A.V. Aho, S.C. Johnson, and J.D. Ullman, Deterministic Parsing of Ambiguous Grammars, Communications of the Association of Computing Machinery 18:8, pp. 441-452, August 1975.

[AHOA 77] A.V. Aho and J.D. Ullman, Principles of Compiler Design, Addison- Wesley, Reading, Mass. (1977).

[AMSL 80] Amsler, R. A., The Structure of the Merriam-Webster Pocket Dictionary, Dissertation. TR-164. Univ. of Texas at Austin, Dec. 1980.

[AMSL 81] Amsler, R. A., A Taxonomy for English Nouns and Verbs, In Proc. of the Assoc. for Comp. Ling. 133-138, June 29-July 1, 1981.

[AMSL 84] Amsler, R. A., Lexical Knowledge Bases, Proceedings of Coling 84 (Stamford, CA : July 2-6, 1984).

[AMSL 84] Amsler, R. A., Machine-Readable Dictionaries, ARIST, 19:161-209,1984.

[APRE 69] Apresyan, Yu. D., I. A. Mel'cuk, and A. K. Zolkovsky., Semantics and Lexicography: Towards a New Type of Unilingual Dictionary, In Studies in Syntax and Semantics, ed. F. Kiefer, 1-33. Dordrecht- Holland: D. Reidel, 1969.

[BABA 85] Babatz, R. and M. Bogen, Semantic Relations in Message Handling Systems: Referable Documents, In Proc. IFIP WG 6.5 Symposium, Sept. 1985.

[CASS 77] Cassidy, Frederic G., Computer-Aided Usage 'Labeling' in a Dictionary, In Computers and the Humanities, 11(1); 89-99, 1977.

[EVEN 79] Evens, M. W. and R. N. Smith, A Lexicon for Computer Question-Answering System, Am. J. Comp. Ling., Microfiche 83,1979.

[EVEN 82] Evens, M. W., Structuring the Lexicon and the Thesaurus with Lexical-Semantic Relations, Final Project Report. 1982.

[EVEN 85] Evens, Martha W., James Vandendorpe, and Yih-Chen Wang, Lexical Semantic Relations in Information Retrieval. In Humans and Machines: The Interface Through Language, Ablex, ed. S. Williams, 1985.

[FOXE 80] Fox, E. A., Lexical Relations: Enhancing Effectiveness of Information Retrieval Systems, ACM SIGIR Forum, Winter 1980,

15(3):5-36.

[FOXE 83a] Fox, E. A., Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types, Dissertation, Cornell University, Available from: UMI, Ann Arbor, Michigan, Aug. 1983.

[FOXE 83b] Fox, E. A., Some Considerations for Implementing the SMART Information Retrieval System under UNIX, TR 83-560, Cornell Univ., Dept. of Comp. Sci., Sept. 1983.

[FOXE 85a] Fox, E. A., Composite Document Search in a Networked Society, In _Coll. Papers 14th ASIS Mid-Year Mtg._, May 20-23, 1985.

[FOXE 85b] Fox, E. A., Composite Document Extended Retrieval: An Overview, In _Res. & Dev. in Inf. Ret., Eighth Annual Int. ACM SIGIR Conf., Montreal_, 42-53, June 1985.

[FOXE 85c] Fox, E. A., Analysis and Retrieval of Composite Documents, In _ASIS '85, Proc. 48th ASIS Ann. Mtg._, 54-58, Oct. 1985.

[FOXE 86] Fox, E. A., Information Retrieval: Research into New Capabilities, In _The New Papyrus: CD-ROM_, Suzanne Ropiequet and Steve Lambert (eds.), Microsoft Press, 1986 (to appear).

[FOXM 80] Fox, M. S., D. J. Bebel, and A. C. Parker, The Automated Dictionary, _IEEE Computer_, 35-48, July 1980.

[FRAN 82] Francis, W. Nelson and Henry Kucera, _Frequency Analysis of English Usage: Lexicon and Grammar_, Boston: Houghton Mifflin Company, 1982.

[HANK 79] Hanks, P. ed., _Collins Dictionary of the English Language_, William Collins Sons & Co., London, 1979.

[HART 81] Harter, Stephen P., and Kenneth F. Kister, Online Encyclopedias: The Potential. In _Library Journal_, pp. 1600-1604, Sept. 1, 1981.

[HOBB 84] Hobbs, Jerry R., Building a Large Knowledge Base for a Natural Language System. Proceedings of _Coling 84_ (Stamford, CA : July 2-6, 1984).

[HORN 74] Hornby, A. S. ed., _Oxford Advanced Dictionary of Current English_, Oxford University Press, Oxford, 1974.

[JOHN 75] Johnson, S. C., 'YACC: Yet Another Compiler Compiler', Computing Science Technical Report No. 32, 1975.

[KAYM 84] Kay, Martin, The Dictionary Server. Proceedings of _Coling 84_ (Stamford, CA : July 2-6, 1984).

[KERN 84] Kernighan, Brian W., and Pike, Rob, _The UNIX Programming Environment_. Prentice- Hall, Inc., New Jersey. 1984.

[LESK 75] Lesk, M.E., 'LEX: A Lexical Analyzer Generator', Computing Science Technical Report No. 39, Bell Laboratories, Murray Hill, New Jersey (Oct. 1975).

[MELC 73] Mel'cuk, I. A., Towards A Linguistic 'Meaning <=> Text' Model. In _Trends in Soviet Theoretical Linguistics_, ed. F. Kiefer. Dordrecht- Holland: D. Reidel, 33-57, 1973.

[MELC 79] Mel'cuk, I. A., _Studies in Dependency Syntax_. Ann Arbor: Karoma Publishers, Inc. 1979.

[MCIL 84] McIlroy, M. D., Personal communication on February 8, 1984.

[MITT 85] Mitton, Roger, 'A Description of the File OALD.DAT'.

Personal communication on July 18, 1985.

[NAIS 80] Naish, L., MU-Prolog 3.1db Reference Manual. Melbourne Univ., 1984.

[PETE 82] Peterson, James L., Webster's Seventh New Collegiate Dictionary: A Computer- Readable File Format. TR-196, Univ. of Texas at Austin, Dept of Comp. Sci., May 1982.

[RIEG 83] Rieger, Burghard B., Lexical Relevance and Semantic Disposition- On Stereotype Word Meaning Representation in Procedural Semantics. In Meaning and the Lexicon. 1983.

[SAGE 81] Sager, N., Natural Language Information Processing, Addison- Wesley, Reading, Mass. 1981.

[SHER 74] Sherman, D., A New Computer Format for Webster's Seventh Collegiate Dictionary. In Computers and the Humanities, 8:21-26, 1974.

[SIMM 84] Simmons, R. F., Computations from the English, Prentice-Hall, Englewood Cliffs, NJ, 1984.

[SPAR 73] Sparck Jones, K. and Martin Kay, Linguistics and Information Science, Academic Press, New York, 1973.

[WEBB 84] Webber, Howard R., Machine- Readable Components in a Variety of Information- System Applications. Proceedings of Coling 84 (Stamford, CA : July 2-6, 1984).

[WHIT 83] White, C., The Linguistic String Project Dictionary for Automatic Text Analysis. In Proc. Workshop on Machine Readable Dictionaries. SRI, Menlo Park, Ca. May 1983.