CREATING AND QUERYING LEXICAL DATA BASES

Mary S. Neff, Roy J. Byrd, and Omneya A. Rizk
IBM T. J. Watson Research Center
P. O. Box 704
Yorktown Heights, New York 10598

ABSTRACT

Users of computerized dictionaries require powerful and flexible tools for analyzing and manipulating the information in them. This paper discusses a system for grammatically describing and parsing entries from machine-readable dictionary tapes and a lexical data base representation for storing the dictionary information. It also describes a language for querying, formatting, and maintaining dictionaries and other lexical data stored with that representation.

1. Introduction.

Computer resident lexical information is required in a large variety of applications, including on-line dictionary reference systems (Neff and Byrd(1987)), lexicographic systems for creating new dictionaries (Tompa(1986)), systems for performing lexicological analyses of existing dictionaries (Zampolli and Calzolari(1985), Chodorow et al.(1985)), and various natural language processing systems (Byrd(1986)). Establishment of standard representations and access mechanisms for lexical information will help us to better meet these requirements.

The Lexical Systems project at the IBM T. J. Watson Research Center has defined a lexical data base representation (LDB) for lexical information, a set of tools and grammars for converting machine readable dictionary (MRD) typesetting tapes to the LDB representation, and a Lexical Query Language (LQL) for specifying storage and retrieval operations on LDBs. This paper briefly discusses the LDB structure, the tools and grammar for creating LDBs from existing sources, and then describes the facilities and implementation of LQL.

Of the many machine-readable dictionary (MRD) resources available, current research concerns exploitation of the information from one dictionary (for example, Michiels (1982), Ahlswede, et. al. (1986)) or, in the case of multiple dictionaries, a limited subset of information from each (cf. Chodorow, et. al. (1985)). This exploitation is usually via a special extraction program which gets only the information needed by the project or application. Although in some cases the literature mentions the term "data base", the term is frequently not used in any strict sense, as it is here.

The work reported in the present paper is related to work reported by Calzolari (1984a, 1984b) and Calzolari and Picchi(1986). Some of it, particularly the development of tools and grammars for parsing entries in MRD tapes to LDB format, is most related in spirit to some of the work reported by the New Oxford English Dictionary project (Tompa (1986a), Gonnet and Tompa(1986), Benbow(1986)). Whereas the new OED project has defined a markup language for dictionary entries before data entry and a grammar based on that markup language, this work attempts to parse and convert to a standard form a number of different raw type-setting tapes without any pre-editing.

2. Lexical Data Bases.

Dictionary entries are typically organized as shallow hierarchies of information with a variable number of instances of certain items at each level, e.g. multiple homographs within an entry or multiple senses within a homograph. More formally, they can be characterized as finite depth hierarchies of attribute-value pairs with a variable number of instances of specific nodes at each level. At at each node there is an attribute name and a value, which can be a simple value, a list of attribute-value pairs, or an entire subtree. A node may have an arbitrary number of daughter nodes of the same attribute type, just as a dictionary entry can have an arbitrary number of senses per homograph, for example. Thus, in the entry for *quack* from the Collins English-Italian dictionary, shown in Figure 1, the "entry" node has two "superhom" (superscript homograph) nodes as daughters; the first of these, in turn, dominates two "homograph" nodes, one for a noun and the other for an intransitive verb. Lexicographers are familiar with the notion of hierarchically structured entries of this type.

It is important to distinguish between the type of hierarchies found in dictionary entries and those, for example, that characterize the syntactic structure of English sentences. Both types of hierarchy are, in principle, of un-

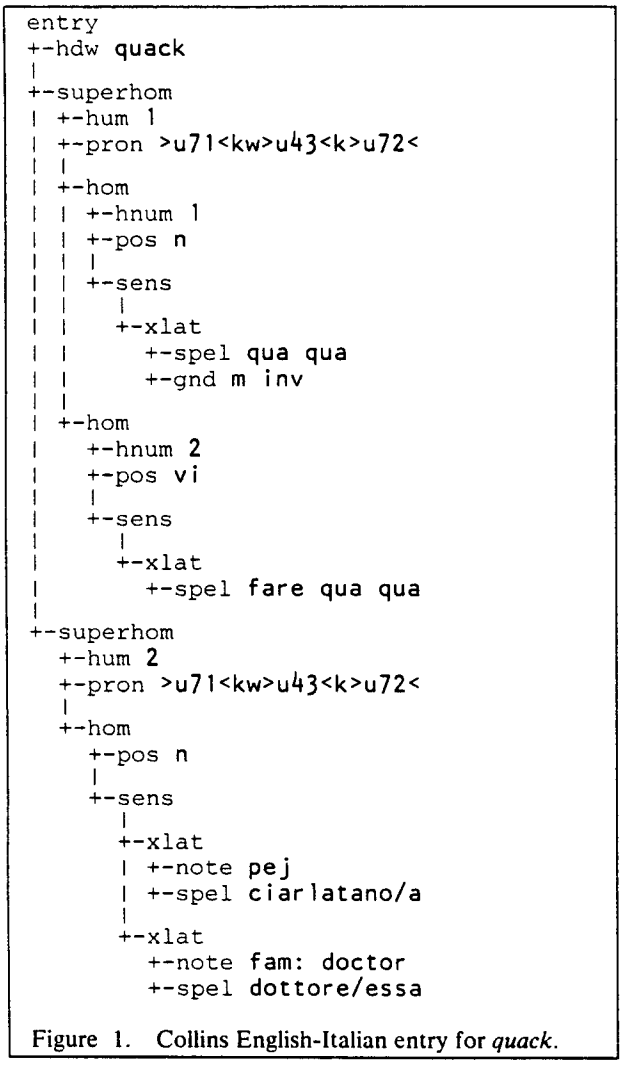

Figure 1. Collins English-Italian entry for *quack*.

bounded size. Dictionary entries achieve their unboundedness by iteration: a dictionary entry may have any number of homographs, which may, in turn, have an arbitrary number of senses, and so forth. Syntax trees achieve their unboundedness both by iteration (e.g., coordination) and by recursion (e.g., self-embedding). In fact, while the nodes of a dictionary entry hierarchy may only iterate, the data values contained in the tree may have their own--possibly recursive--structures. This is the case for definition text in monolingual dictionaries, or for etymologies (as was pointed out by Martin Chodorow).

The observation that dictionary entry hierarchies may only iterate is useful for defining lexical data bases. All entries in an LDB can be characterized by a "design" or "template" which is stored once for that data base. The design is a single hierarchy which serves as a grammar for any instance of an entry by naming the possible parents, siblings, and children of each node type that may occur. Figure 2 shows a simplified design for an LDB containing the Collins English-Italian bilingual dictionary. Comments are given to describe the lexicographic functions of the various nodes. The entry for *quack* shown in Figure 1 is an instance of the design shown in Figure 2. Different dictionaries may have different structures and hence different designs. The notion of representing dictionary entries as instances of a hierarchical design derived from a grammar is not unique to our work; Gonnet and Tompa (1987) use the same notion when they create hierarchical "p-strings" over the strings that they analyze with their grammatical formalism.

Actual LDBs are created by storing hierarchically formatted entries in a direct access file with the design stored as control information (the creation of these formatted entries is the subject of the next section). The files are managed by the Dictionary Access Method (DAM), described in Byrd (1986) and Byrd, et al.(1986b). Each entry is stored with its headword as the key. Alternate access paths can be established by building indexes on attributes other than the headword.

It may be useful to point out reasons for not using traditional data base mechanisms for storing and accessing LDBs. We believe that lexical data bases cannot be correctly viewed as relational data bases. Within the taxonomy of "normal forms" defined by relational data base theory, dictionary entries are "unnormalized relations" in which attributes can contain other relations, rather than simple scalar values. As a result, no efficient data base tools or attractive languages have been developed for accessing unnormalized relations. In order to use existing relational tools to store and access dictionary entries, we should first re-cast the entries into one of the normal forms defined by the theory. This could be achieved by defining relations over sets of siblings in our LDB designs, creating or selecting unique key attributes among those siblings, and performing join operations on those attributes. However, to do so would force us to sacrifice the intuitive advantage that hierarchically organized dictionary entries offer. We have therefore chosen not to.

Similarly, traditional hierarchical data bases do not provide an appropriate model for lexical data. In those systems, as typified by IBM's IMS (Date (1986)), hierarchical relationships are represented among separate entities which are stored as "segments". This metaphor of separate entities does not apply to lexical data. There is no sense, for example, in which a translation in the Collins English-Italian dictionary is a separate entity to be related to other entities. Rather, each lexical entry is itself a complex entity

```
entry                   /*root of the tree*/
+-hdw                   /*English headword; key to the DAM file*/
|
+-superhom              /*superscript homograph*/
  +-hum                 /*superscript number from printed dictionary*/
  +-pron                /*encoded pronunciations*/
  |
  +-altspel             /*alternate spelling*/
  | +-note              /*e.g."U.S. English"*/
  | +-spel              /*the alternate spelling string*/
  |
  +-hom                 /*homograph*/
    +-hnum              /*homograph number*/
    +-pos               /*part-of-speech*/
    +-morph             /*irregular inflections, etc.*/
    |
    +-sens              /*sense*/
      +-snum            /*sense number*/
      |
      +-xlat            /*translation information for headword*/
      | +-note          /*usage note for the English term*/
      | +-spel          /*the Italian translation*/
      | +-gnd           /*grammatical information about the Italian*/
      |
      +-xmp             /*"example" phrases containing headword*/
        +-note          /*usage note for the English phrase*/
        +-gloss         /*the English phrase*/
        +-pos           /*part-of-speech*/
        +-expl          /*more usage information*/
        +-tran          /*Italian translation of phrase*/
        +-gnd           /*grammatical information about the Italian*/
```

Figure 2. LDB design for the Collins English-Italian dictionary.

with a hierarchical relationship among its internal components. Our LDB strategy preserves this intuition.

3. From typesetting tape to data base.

Among the resources available for making lexical data bases, we have typesetting tapes of Webster's Seventh, Longman's Dictionary of Contemporary English (LDOCE), several Collins bilingual dictionaries, and two synonym dictionaries. Creation of lexical data bases began from a number of direct access b-tree (DAM) files which had been created from the tapes for the WordSmith on-line dictionary reference system (Neff and Byrd(1987)). These files, keyed on headword and containing the otherwise unnormalized body of the entry, complete with original font codes, were the result of a by-entry segmentation program, an idiosyncratic process reflecting the diverse origin and formatting conventions of the source dictionaries. To meet the requirements of computerized random access, some entries (e.g., those with superscripted homograph numbers) were combined into one entry; words with alternate spellings (such as "whisk(e)y") became multiple entries with cross references; still others (compound entries from the German-English bilingual dictionary) were broken up so that each compound word listed could be accessed by its own key.

The dictionary entry parser. The hierarchical structure of a dictionary entry is implicit in its syntax. Major signposts are the homograph numbers, the sense numbers, and the consistent alternation of fonts; also useful are punctuation, position of swung dashes (indicating repetition of the head word in a collocation or example), and membership of an item in a closed set (part of speech, for example). By way of illustration, we reproduce here for the entry *quack* -- both the raw version from the tape of the Collins English-Italian dictionary and the formatted entry as it appears in the printed dictionary.

```
>u1<quack >u123<1 >u155<>u71<kw>u43<k>u72<
>u2<1 >u6<n >u5<qua qua >u6<m inv. >u2<2
>u6<vi >u5<fare qua qua. >u1<quack >u123<2
>u155<>u71<kw>u43<k>u72<  >u6<n  >u6<(pej)
>u5<ciarlatano/a; >u6<(fam: doctor)
>u5<dottore/essa.
```

quack[1] [kwaek] **1** *n* qua qua *m inv.* **2** *vi* fare qua qua.
quack[2] [kwaek] *n (pej)* ciarlatano/a; *(fam: doctor)* dottore/essa.

To parse the dictionary entries, we constructed a general parsing engine and a specific grammar for each one of se-

veral MRD's. Because dictionary entries may only iterate, the level of sophistication required of an entry parser is not as high as that required for a sentence parser. Nevertheless, two technologies for sentence parsers were readily available to us: a bottom-up, all-paths strategy offered by PLNLP (Langendoen and Barnett(1986)), and a top-down depth-first approach offered by logic grammars in Prolog (McCord(1986)). Using either would significantly reduce the effort required to parse our more than a dozen dictionaries, because each new dictionary would only require a new grammar. Preliminary versions in PLNLP and Prolog were both adequate and contribute nothing to the theoretical issues surrounding parsing strategies. The choice of Prolog for continued development was largely due to the somewhat deterministic nature of parsing dictionary entries vis-à-vis the extravagance of the bottom-up, all-paths strategy. Nevertheless, we are studying the possibility of implementing a partial bottom-up strategy analogous to parse-fitting (cf. Jensen, et al.(1983)) when it becomes necessary to process input with missing or corrupt font codes, there being few recovery strategies available to a top-down parser.

Grammars for entries and grammars for sentences differ in three important ways: (1) entries can be huge: some are longer than 5000 bytes; (2) tokens, the smallest unit handled by the grammar are larger and defined differently; and (3) a dictionary grammar does not have to produce recursive structures, so can be to a large extent deterministic. A consequence of (1) is that it takes a large amount of storage to parse an entry. To process extremely long entries, we use an 8-megabyte virtual machine under VM/CMS. The motivation for (2) comes from the fact that the entries consist of text interspersed with font codes, which are not required to be delimited with blanks. The token string for an entry is therefore usually an alternating string of font codes and characters, with some semicolons or periods also defined as tokens.

The grammar is a formal description of the structure of entries for a particular dictionary; its formalism is a modification and extension of McCord's (1986) modular logic grammar (MLG), which is in turn derived from the definite clause grammar (DCG) of Pereira and Warren (1980). We illustrate with some sample rules from the grammar of the Collins English-German dictionary.

```
(1) body ==> opt(prehom) : opt(homlist(*)).
(2) body ==> alt.
(3) prehom ==>  opt(alt) :
    opt(pronunc) : opt(vbmorph) :
    opt(note) : opt(abb).
(4) homlist(num) ==> hom(num) :
    opt(opt(-sc) : homlist(num)).
(5) homlist(nonum) ==> hom(nonum).
```

Rule (1) says that the body of an entry consists of optional prehomograph material (*prehom*) and an optional homograph list of any type(*). A significant addition to the MLG formalism was a mechanism for treatment of large numbers of optional elements in a single rule to prevent proliferation of rules. Tests added by the compiler enforce the convention that to succeed, a rule consisting solely of optional elements must contain at least one. Rule (2) says that the body of an entry consists of an alternate spelling. This rule is an alternate to (1); normal Prolog backtracking conventions apply. Rule (3) says that the prehomograph may contain any of the following, in this order: alternate spelling, pronunciation, verb morphology, a usage note, a long form of which this is an abbreviation. Rule (4) says that a numbered homograph list consists of a numbered homograph followed optionally by a semicolon (to be discarded from the resulting structure) and a numbered homograph list. This rule illustrates the mechanism for defining multiple sister nodes of the same type by means of a recursive rule. Though it appears that each successive homograph node is one level deeper in the tree, the use of McCord's definition of some nodes as "weak nonterminals" ensures that nodes so defined (like *homlist*, and, incidentally, *prehom*) disappear from the final structure, thus flattening the tree and eliminating recursion. Rule (5) handles the case of a single unnumbered homograph.

Leaf node rules are more numerous than those that describe higher structures, and these may contain Prolog-like tests. Unlike syntax parsers, dictionary entry parsers discard some segments (e.g. font codes) after use in parsing; the "-" operator, added to the original rule formalism, allows any segment to disappear from the resulting tree.

```
(6) pos ==> -font(ital) : +Seg :
    $not(stconc("(",*,Seg)) : opt(-sc).
```

This rule, one of four handling part of speech segments, says that the *pos* segment (Seg) is preceded by an italic font code (discarded), does not begin with a left parenthesis, and is followed optionally by a semicolon (also discarded).

Font code definition rules and retokenization rules are needed before parsing rules are applied. Font code definition rules are simply Prolog unit clauses, which define font codes and delimiters, as in the following simple example, where the string ">u4<" is both a boldface font token and a delimiter.

```
delim(">u4<",font(bold)).
```

Delimiters and strings between delimiters are the tokens. After initial tokenization, retokenization rules in a formalism similar to that of the grammar rules make adjustments to the token string because of the inexact mapping of font changes to segments, as in

```
...>u5<(>u4<word>u5<) something ...
```

which is tokenized as

```
font(roman)."(".font(bold)."word".
   font(roman).") something"...
```

but which is modified to:

```
font(bold)."(word)".font(roman).
"something" ...
```

The grammar and retokenization rules are compiled into Prolog clauses with a compiler that is a modification and extension of McCord's (1986) MLG rule compiler. Extensions include the "opt" operator, the "-" operator, and the "++" operator, which allows rules to *insert* tokens into the string during parsing, thus allowing for breakup and analysis of two different data items not separated by a font code.

The compiler and rules are supported by a Prolog shell which offers the rule developer a variety of tools for tracing, debugging, and selecting among a large number of options, including input and output data and file formats, thus supporting both development and batch processing. Tools for analyzing failures assist in the often tedious process of combing through a long entry to determine why it failed; in particular, there is a mechanism optionally compiled into the rules which marks the right-most frontier (RMF) reached in the parsing process. Other tools analyze the RMF environments resulting from a batch run to help the developer determine which rule modifications are required by the largest number of entries.

The parse trees are compacted and encoded in LDB format and stored in a DAM file for access by analysis programs, such as the Lexical Query Language (LQL), described in the next section. The rule compiler, as part of the compilation process, produces the entry design for the dictionary that is required by LQL and encodes it as control information with the LDB.

4. The Lexical Query Language.

The LQL programming language must satisfy several requirements. First, it must be possible to formulate an unrestricted variety of queries against any LDB. Second, it must be possible to flexibly format the answers to queries, either for display or for use in creating other computer files. Third, LQL must allow users to specify modification operations on LDBs; this will allow for LDB maintenance. The style of LQL programs must be natural and easy to remember. See Byrd(1986) for a discussion of these requirements on computerized dictionaries.

Data base query. The Lexical Query Language allows the user to specify conditions on the attributes of LDB entries. Only those entries which satisfy all conditions become part of the query answer. Further, the user specifies which attributes of the successful entries are part of the answer and what their output format will be. The query is stated as entries in the nodes of a two-dimensional representation of an LDB's design (see Figure 2), using a syntax reminiscent of the Query-by-Example (QBE) data base query language (Zloof(1974)). Example-elements, denoted by a leading underscore, are used to relate values of attributes in a query tree to conditions in a **condition box**, to display positions in an **output box**, and to values in other dictionary entry trees (for "join" operations).

Part (a) of Figure 3 shows a query which will list all words which are both nouns and verbs in English together with their translations in the Collins English-Italian dictionary. The condition on the noun part-of-speech attribute is simple (it must be "n" for this data base) and is entered directly in the tree. The condition on the verb part of speech is more complex, and the example-element __VPOS is used to relate those attribute values to the condition box, where they are tested for equality with either "vi" or "vt". The example-elements __WORD, __NTRAN, and __VTRAN are used to map answers from the hierarchy into the output box where their eventual output format is schematically represented. Part (b) of Figure 5 shows sample entries from the answer to this query when applied to the Collins English-Italian dictionary. Such a query might be useful, for example, in a study of the relation between homography in English and derivational morphology in Italian.

As in the query tree itself, the output box may contain both constants and example-elements. The constants define boilerplate material used to label the variable instantiations of the example-elements, as in Figure 3. Such labelling is perhaps more useful when dictionary entries are presented one at a time, as in the WordSmith on-line dictionary system (Neff and Byrd(1987)). Alternately, a **p.** (for "print") operator can be used in the terminal or non-terminal nodes of the query tree to specify which data should be given in a hierarchically formatted display of the query answers. Figure Figure 4(c) illustrates the use of the **p.** operator.

LQL conditions specify tests to be performed on the values of attributes in the data tree to which nodes in the query tree may be mapped during query processing. Terminal nodes may be tested using a variety of string and arithmetic operations. The current prototype implementation includes the built-in functions of the REXX programming language (IBM (1984)). Non-terminal nodes may only be tested for equality or inequality with other nodes having the same attribute name. All nodes may have aggregate functions (e.g., count, maximum, minimum, etc.) applied to them and the results may either be tested in conditions or be output as part of the query answer. Nodes may also be

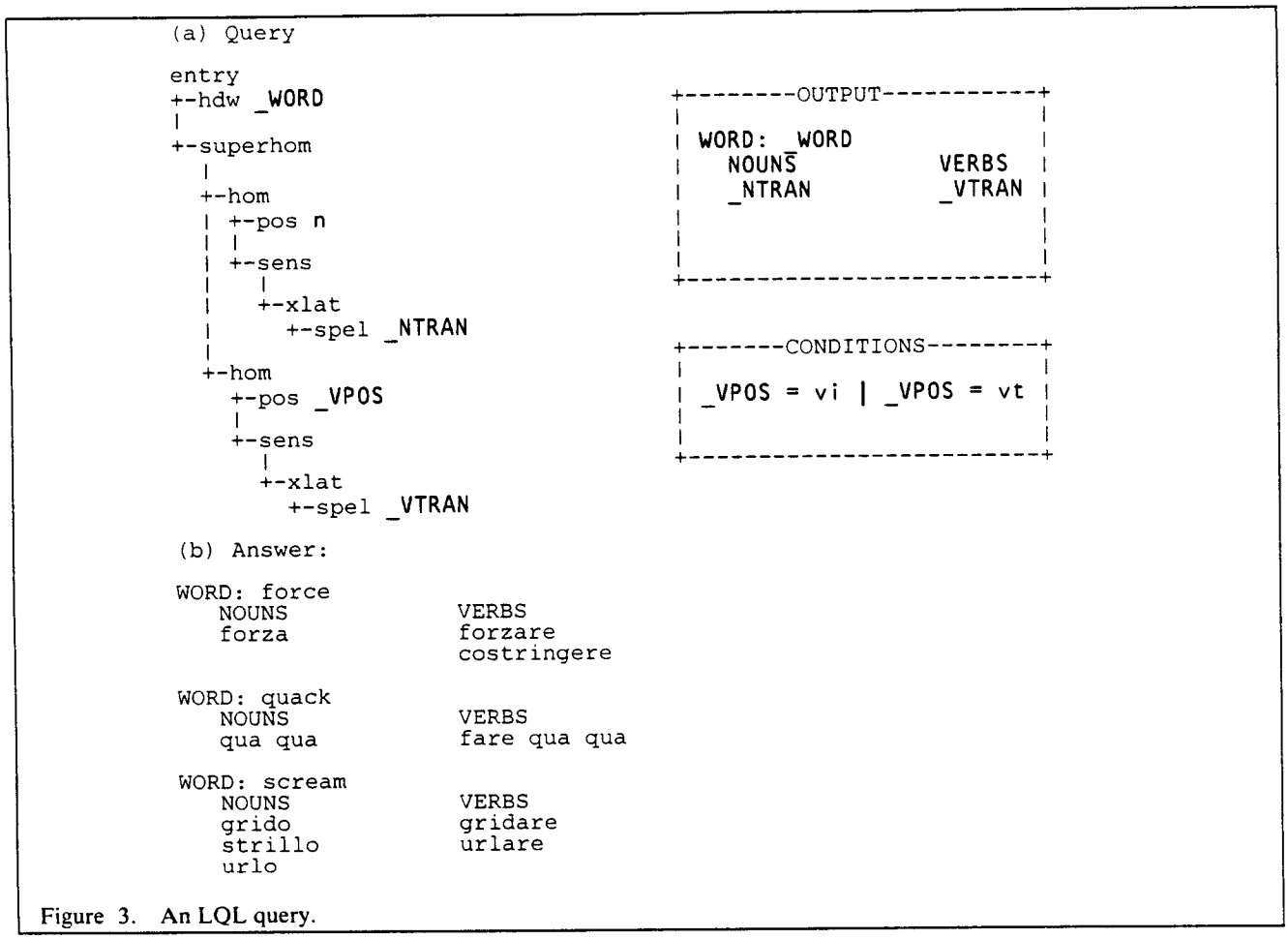

Figure 3. An LQL query.

tested for a null value (i.e., non-occurrence in a particular entry).

In addition to producing the formatted output specified in the output box, the process of answering a query also yields, as a by-product, an **answer LDB**. This is a new LDB containing just those entries from the original LDB which satisfied all conditions, and containing just those attributes which were either part of the conditions or the output specification. The design for the answer LDB is derived from the query tree and is a subset of the design for the original LDB.

Data base modification. A realistic data base system must provide facilities for creating and maintaining data bases. LDB creation is usually a bulk operation and has been discussed in section 3. LDB maintenance, on the other hand, can benefit from the flexibility provided by combining a powerful query processor with the capability to insert, delete, and update parts of the LDB entries. LQL offers this flexibility by providing the operators **i.** (for "insert"), **d.** (for "delete), and **u.** (for "update"). These are also familiar QBE operators and are described (in a relational context) in IBM(1978).

Figure 4 shows three examples of how these operators might be used to modify an LDB containing the Collins English-Italian dictionary. In (a), a new entry for the term "lexical data base" is added to the dictionary; notice that the **i.** operator applies to the entire hierarchy, so that a new record will be added to the DAM file containing the LDB. Similarly, program (b) will delete entire entries, if any have headwords which satisfy the condition that they not be alphabetic (determined by using the REXX **datatype** function, which - in this case - checks whether the value consists of entirely mixed case alphabetic characters). Finally, (c) locates entries where a "superhom" (superscript homograph) node dominates exactly one "hom" (homograph) node (as determined by the **cnt.** operator). The headwords of such entries are printed by the **p.** operator and the "hom" nodes receive a new "hno" (homograph number) attribute with the value "1" as a result of the **i.** operator.

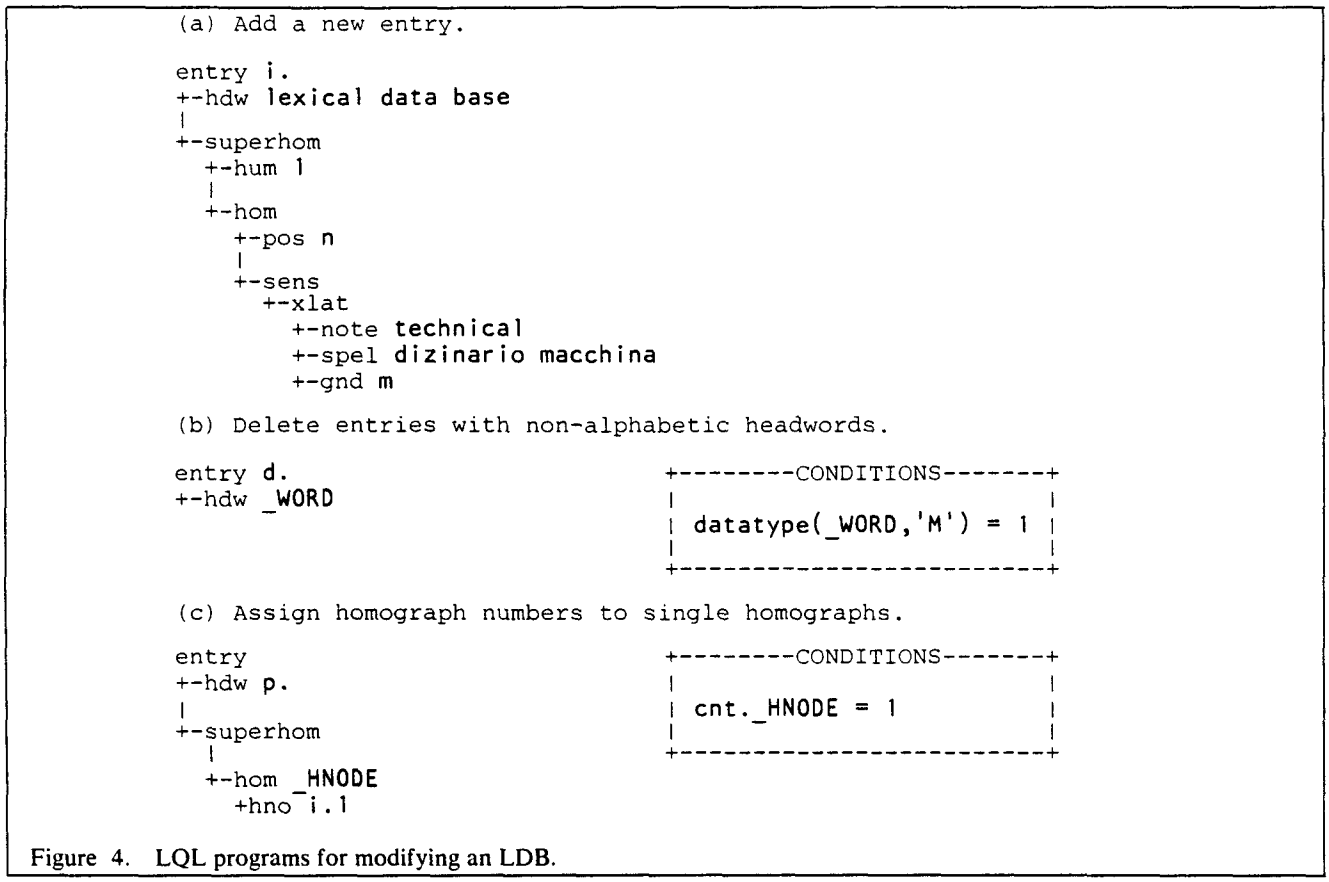

Figure 4. LQL programs for modifying an LDB.

Figure 5 shows how an initial attempt to build an Italian synonym dictionary from an English synonym dictionary and an English-Italian bilingual dictionary might be programmed using LQL. This program creates the Italian synonym dictionary by simply translating the English synonym dictionary into Italian while retaining the English sense numbers. The program uses example elements (e.g., __ESYN which maps English synonyms to headwords in the English Italian dictionary) to specify join operations among the input dictionaries and to map results into the output dictionary. Clearly, this procedure is lexicographically naive and inadequate; the point of the example is to show the ease which which lexical exploration can be performed using LQL.

5. Status and Plans.

The entire Collins English-German dictionary, consisting of 46,600 entries, was recently submitted to the parser and E-G grammar with a parsing rate of 80% of the entries. Parts of the Collins Italian-English and English-Italian dictionary were parsed with their respective grammars with a success rate of about 95%; some of the entries were quite large. The high success rate on the Italian dictionaries is partly due to the consistency in the formatting of these dictionaries and the integrity of the tapes. The rules for both grammars are still being improved to account for the residue. Among the remaining problems are the following: some contiguous data items appear in the same font without an intervening font code, some discrete data items use more than one font, some kinds of data items are discontiguous, and some new lower level structures used in only a few entries remain to be discovered.

Unfortunately, the residue often contains long entries associated with high-frequency words, making partial results less immediately usable. However, unparsable entries often have large sections of parsable material, which could be made available in LDB format for analysis or applications in spite of its partial nature, if only the top-down parser wouldn't fail. Because a dictionary entry has identifiable signposts like homograph numbers and sense numbers that can be possible recovery points, we plan to implement and constrain a "junk collecting" rule in the English-German grammar that will pick up and blandly label everything at or just before the failure point up to the next recovery point.

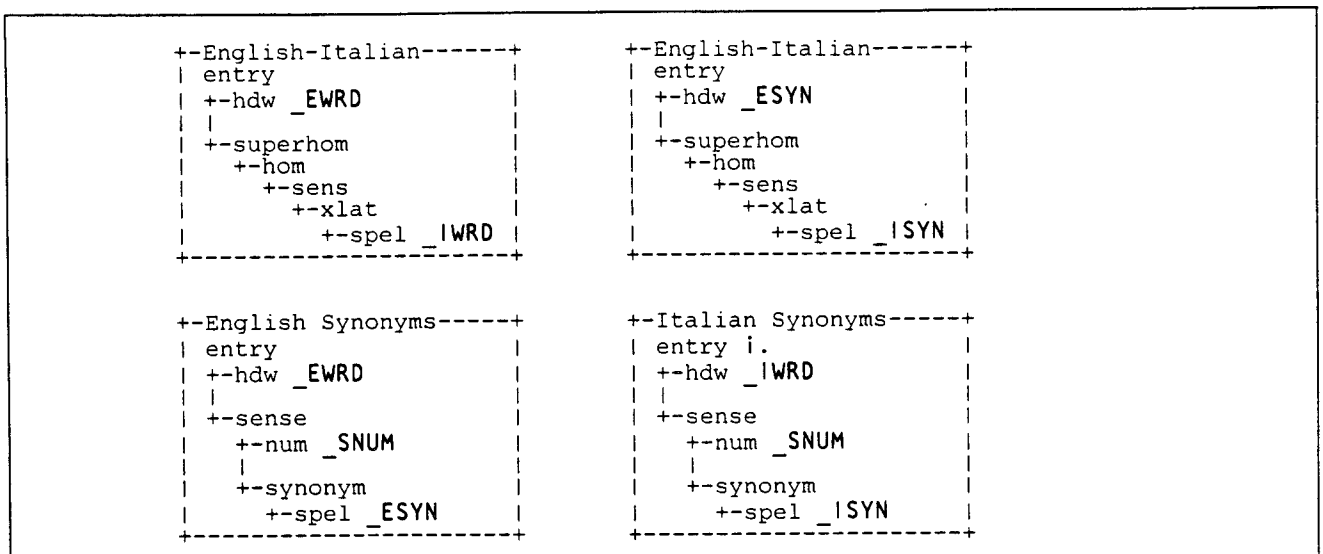

Figure 5. LQL program to create an Italian synonym dictionary.

Recently we parsed a small random sample from the Collins French-English dictionary using a slightly modified English-German grammar, including some junk collection. The success rate was about 50%. The time required to modify the English-German grammar for the French dictionary was less than an hour, confirming our belief in the efficacy of our approach to dealing with multiple dictionaries.

We intend to build grammars and LDBs for all our dictionary resources. The uses to which these LDBs can be put are numerous; we have identified the following initial projects. Analysis of the English-Italian and Italian-English dictionaries has allowed us to transfer semantic information from an English monolingual dictionary to an Italian one (see Byrd, et al.(1986)). The English-German LDB will soon be used for analysis as part of the development of LMT, an English-German machine translation system (McCord (1986)). The English-French and French-English dictionaries will be used as part of a study of asymmetrical references in bilingual dictionaries (see Byrd, et al.(1986)).

Many other kinds of applications and analyses will be possible with the LDBs created from printed dictionaries. One could easily imagine a lexical data base as the common source for several different variants of printed dictionaries. Indeed, publishers are beginning to use the notion of an LDB for maintenance and enhancement of their products (cf. New OED, Gonnet and Tompa(1986)). On-line reference systems need not be limited to the information available in printed dictionaries, as they are today: users of such systems can define their own views of the on-line data. Natural language processing systems can use parts of a common LDB, extracted for their requirements. Further exciting possibilities include more inter-dictionary investigations or even the creation of a combined LDB and dealing with the so-called "mapping" problem: how to map information from one dictionary onto another.

A prototype LQL query processor and output formatter have been built at IBM Research. The prototype processes single dictionary queries with conditions and produces formatted and hierarchical output as well as answer LDBs. Current work is aimed at implementing the full language (including joins, updates, and aggregate operations) and improving performance in order to make LQL attractive for use in a wide variety of applications. The first large scale applications will be a stand-alone query processor for use in lexicological research and an entry storage, filtering, and formatting mechanism for the WordSmith on-line dictionary system.

LQL queries would be tedious and difficult to create if they had to be entered all by hand. Fortunately, the LDB design can be used to build a user interface which presents a query tree template to be filled in by the terminal user. The prototype implementation does this and provides further facilities for moving, copying, and deleting subtrees during query preparation. A new one-dimensional representation must be defined for use in storing LQL programs and to provide a mechanism for issuing LQL requests from other programming languages.

Beyond dictionaries. The tools described here can be used for parsing and querying other kinds of data. The notion of creating a data base from data parsed from text ("text-dominated databases," cf. Gonnet and Tompa(1987)) can

be applied to other collections of structured data, such as almanacs, encyclopedias, abstracts, legal documents, or bibliographies.

References.

Ahlswede, Thomas, Martha Evens, Kay Rossi, and Judith Markowitz (1986) "Building a Lexical Database by Parsing Webster's Seventh New Collegiate Dictionary," *Advances in Lexicology*, Second Annual Conference of the UW Centre for the New Oxford English Dictionary, 65-78.

Benbow, Tim (1986) "Status Report on the New OED Project", Oxford University Press, unpublished.

Byrd, Roy J. (1986) "Dictionary Systems for Office Practice," Proceedings of the Grosseto Workshop "On Automating the Lexicon", also available as IBM Research Report RC 11872.

Byrd, Roy J., Nicoletta Calzolari, Martin S. Chodorow, Judith L. Klavans, Mary S. Neff, Omneya A. Rizk (1987) "Tools and Methods for Computational Lexicology," *Computational Linguistics*.

Byrd, Roy J., Gustaf Neumann, and Karl Seved B. Andersson (1986b) "DAM - A Dictionary Access Method," IBM Research Report, in preparation.

Calzolari, Nicoletta (1984a) "Detecting patterns in a lexical data base," Proceedings of COLING 84, 170-173.

Calzolari, Nicoletta (1984b) "Machine-readable dictionaries, lexical data bases, and the lexical system," Proceedings of COLING 84, 460.

Calzolari, Nicoletta and Eugenio Picchi, (1986) "A Project for a Bilingual Lexical Database System", *Advances in Lexicology*, Second Annual Conference of the UW Centre for the New Oxford English Dictionary, 79-92.

Chodorow, Martin S., Roy J. Byrd, and George E. Heidorn (1985) "Extracting Semantic Hierarchies from a Large On-line Dictionary," *Proceedings of the Association for Computational Linguistics*, 299-304.

Collins (1980) *Collins German Dictionary: German-English, English-German*, Collins Publishers, Glasgow.

Collins (1980) *Collins Sansoni Italian Dictionary: Italian-English, English-Italian*, Collins Publishers, Glasgow.

Date, Christopher J. (1986) *An Introduction to Data Base Systems*, Addison-Wesley.

Gonnet, Gaston H. and Frank Wm. Tompa (1986) "Status Report on University of Waterloo Technical Activities for the New OED Project", University of Waterloo, unpublished.

Gonnet, Gaston H. and Frank Wm. Tompa (1987) "Mind Your Grammar: a New Approach to Modelling Text," University of Waterloo Centre for the New Oxford English Dictionary, Report OED-87-01.

IBM (1978) *Query-by-Example: Terminal Users Guide*, IBM form no. SH20-2078.

IBM (1984) *System Product Interpreter (REXX) Reference Manual*, IBM form no. SC24-5239.

Jensen, Karen, George E. Heidorn, Lance A. Miller, and Yael Ravin (1983) "Parse Fitting and prose Fixing: Getting a Hold on Ill-formedness," AJCL 9.3-4.123-36.

Langendoen, D. Terence and H. Michael Barnett (1986) "PLNLP: A Linguist's Introduction," IBM Research Report.

Longman(1978) *Longman Dictionary of Contemporary English*, Longman Group, London.

McCord, Michael C. (1986) "Design of a Prolog-Based Machine Translation System", *Proc. Third International Conference on Logic Programming*, Springer-Verlag, 350-374.

McCord, Michael C. (1987) "Natural language processing and Prolog," *Knowledge Systems and Prolog*. in Adrian Walker, Michael McCord, John Sowa, and Walter Wilson, ed. Addison-Wesley, Waltham, Massachusetts.

Michiels, Archibal (1982) *Exploiting a Large Dictionary Data Base*. Unpublished PhD Dissertation. University of Liege, Liege, Holland.

Neff, Mary S. and Roy J. Byrd (1987) "WordSmith Users Guide," IBM Research Report, in preparation.

Pereira, Fernando, and David Warren (1980) "Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks", *Artificial Intelligence*, 13, 231-178.

Tompa, Frank (1986) "Database design for a dictionary of the future," University of Waterloo, unpublished.

Zampolli, Antonio and Nicoletta Calzolari (1985) "Computational Lexicography and Lexicology," *AILA Bulletin*, pp. 59-78.

Zloof, Moshe M. (1974) "Query by Example," IBM Research Report RC 4917.