A TOOL KIT FOR LEXICON BUILDING

Thomas E. Ahlswede
Computer Science Department
Illinois Institute of Technology
Chicago, Illinois 60616, USA

ABSTRACT

This paper describes a set of interactive routines that can be used to create, maintain, and update a computer lexicon. The routines are available to the user as a set of commands resembling a simple operating system. The lexicon produced by this system is based on lexical-semantic relations, but is compatible with a variety of other models of lexicon structure. The lexicon builder is suitable for the generation of moderate-sized vocabularies and has been used to construct a lexicon for a small medical expert system. A future version of the lexicon builder will create a much larger lexicon by parsing definitions from machine-readable dictionaries.

INTRODUCTION

Natural language processing systems need much larger lexicons than those available today. Furthermore, a good computer lexicon with semantic as well as syntactic information is elaborate and hard to construct. We have created a program which enables its user to interactively build and extend a lexicon. The program sets up a user environment similar to a simple interactive operating system; in this environment lexical entries can be produced through a small set of commands, combined with prompts specified by the user for the desired kind of lexicon.

The interactive lexicon builder is being used to help construct entries for a lexicon to be used to parse and generate stroke case reports. Many terms in this medical sublanguage either do not appear in standard dictionaries or are used in the sublanguage with special meanings. The design of the lexicon builder is intended to be general enough to make it useful for others building lexicons for large natural language processing systems involving different sublanguages.

The interactive lexicon builder will be the basis for a fully automatic lexicon builder which uses Sager's Linguistic String Parser (LSP) to parse machine-readable text into a relational network based on a modified version of Werner's MTQ (Modification-Taxonomy-Queueing) schema. Initially this program will be applied to Webster's Seventh Collegiate Dictionary and the Longman Dictionary of Contemporary English, both of which are available in machine-readable form.

LEXICAL-SEMANTIC RELATIONS

The semantic component of the lexicon produced by this system consists principally of a network of lexical-semantic relations. That is, the meaning of a word in the lexicon is indicated as far as possible by its relationships with other words. These relations often have semantic content themselves and thus contribute to the definition of the words they link.

The two most familiar such relations are synonymy and antonymy, but others are interesting and important. For instance, to take an example from the vocabulary of stroke reports, the carotid is a kind of artery and an artery is a kind of blood vessel. This "is a kind of" relation is taxonomy. We express the taxonomic relations of "carotid", "artery" and "blood vessel" with the relational arcs

carotid	T	artery
artery	T	blood vessel

Another important relation is that of the part to the whole:

ventricle	PART	heart
Broca's area	PART	brain

Note that taxonomy is transitive: if the carotid is an artery and an artery is a blood vessel, then the carotid is a blood vessel. The presence or absence of the properties of transitivity, reflexivity and symmetry are important in using relations to make inferences.

The part-whole relation is more complicated than taxonomy in its properties; some instances of it are transitive and others are not. From this and other criteria, Iris et al. (forthcoming) distinguish four different part-whole relations.

Taxonomy and part-whole are very common relations, by no means restricted to any particular sublanguage. Sublanguages may, however, use relations that are rare or nonexistent in the general language. In the stroke vocabulary, there are many words for pathological conditions involving the failure of some physical or mental function. We have invented a relation NNABLE to express the connection between the condition and the function:

aphasia NNABLE speech
amnesia NNABLE memory

Relations such as T, PART, and NNABLE are especially useful in making inferences. For instance, if we have another relation FUNC, describing the typical function of a body part, we might combine the relational arc

speech FUNC Broca's area

with the arc

aphasia NNABLE speech

to infer that when aphasia is present, the diagnostician should check for the possibility of damage to Broca's area (as well as to any other body part which has speech as a function).

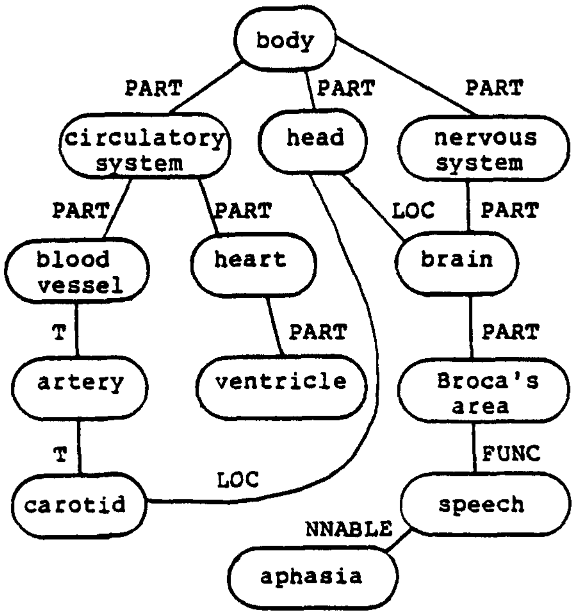

Figure 1. Part of a relational network

Another kind of relation is the "collocational relation", which governs the combining of words. These are particularly useful for generating idiomatic text. Consider the "typical preposition" relation PREP:

on PREP list

which says that an item may be "on a list" as opposed to "in a list" or "at a list."

Although the lexicon builder is based on a relational model, it can be adapted for use in connection with a variety of models of lexicon structure. A semantic-field approach can be handled by the same mechanism as relations; the lexicon builder also recognizes unary attributes of words, and these attributes can be treated as semantic features if one wishes to build a feature-based lexicon.

APPLICATIONS FOR THE LEXICON BUILDER

This project was motivated partly by theoretical questions of lexicon design and partly by projects which required the use of a lexicon.

For instance, the Michael Reese Hospital Stroke Registry includes a text generation module powered by a relational lexicon (Evens et al., 1984). This application provided a framework of goals within which the interactive lexicon builder was developed. The vocabulary required for the Stroke Registry text generator is of moderate size, about 2000 words and phrases. This is small enough that a lexicon for it can be built interactively.

One can imagine many applications for a large lexicon such as the automatic lexicon builder will construct. Question answering is one of our original areas of interest; a large, densely connected vocabulary will greatly add to the variety of inferences a question answering system can make. Another area is information retrieval, where experiments (Evens et al., forthcoming) have shown that the use of a relational thesaurus leads to improvements in both recall and precision.

On a more theoretical level, the automatic lexicon builder will add greatly to our understanding of sublanguages, notably that of the dictionary itself. We have noted that a specialized relation such as NNABLE, unusual in the general language, may be important in a sublanguage. We believe that such specific relations are part of the distinctive character of every sublanguage. The very possibility of creating a large, general-language lexicon points toward a time when sublanguages will be obsolete for many of the purposes for which they are now used; but they will still be useful and interesting for a long time to come, and the automatic lexicon builder gives us a new tool for analyzing them.

THE INTERACTIVE LEXICON BUILDER

Commands

The interactive lexicon builder consists of an operating-system-like environment in which the user may invoke the following commands:

HELP displays a set of one-line summaries of the commands, or a paragraph-length description of a specified command. This paragraph describes the command-line arguments, optional or required, for the given command, and briefly explains the function of the command.

ADDENTRY provides a series of prompts to enable the user to create a lexical entry. Some of these prompts are hard coded; others can be set up in advance by the user so that the lexicon can be tailored to the user's needs.

EDIT enables the user to modify an existing entry. It displays the existing contents of the entry item by item, prompting for changes or additions. If the desired entry is not already in the lexicon, EDIT behaves in the same way as ADDENTRY.

DELETE lets the user delete one or more entries. An entry is not physically deleted; it is removed from the directory, and all entries with arcs pointing to it are modified to eliminate those arcs. (This is simple to do, since for every such arc there is an inverse arc pointing to that entry from the deleted one.) On the next PACK operation (see below) the deleted entry will not be preserved in the lexicon.

This command can also be used to delete the defective entries that are occasionally caused by unresolved bugs in the entry-creating routines, or which might arise from other circumstances. A special option with this command searches the directory for a variety of "illegal" conditions such as nonprinting characters, zero-length names, etc.

LIST gives one-line listings of some or all of the entries in the lexicon. The listing for each entry includes the name (the word itself), sense number, part of speech, and the first forty characters of the definition if there is one.

SHOW displays the full contents of one or more entries.

RELATIONS displays a table of the lexical-semantic relations used by the lexicon builder. This table is created by the user in a separate operation.

UNDEF is a special form of EDIT. In creating an entry, the user may create relational arcs from the current word to other words that are not in the lexicon. The system keeps a queue of undefined words. UNDEF invokes EDIT for the word at the head of the queue, thus saving the user the trouble of looking up undefined words.

PACK performs file management on the lexicon, sorting the entries and eliminating space left by deleted ones.

This routine works in two passes. In the first pass, the entries are copied from the existing lexicon file to a new file in lexicographic order and a table is created that maps the entries from their old locations to their new ones. At this stage, a relational arc from one entry to another still points to the other entry's old location. The second pass updates the new lexicon, modifying all relational arcs to point to the correct new locations.

QUIT exits from the lexicon builder environment. Any new entries or changes made during the lexicon building session are incorporated and the directory is updated.

Extensions to the commands

All of the commands can be abbreviated; so far they all have distinctive initials and can thus be called with a single keystroke.

Each command may be accompanied by command-line arguments to define its action more precisely. Display commands, such as HELP or SHOW, allow the user to get a printout of the display. Where an entry name is to be specified, the user can get more than one entry by means of "wild cards." For instance, the command "LIST produc" might yield a list showing entries for "produce", "produced", "produces", "producing", "product", and "production."

Additional commands are currently being developed to help the user manage the relation table and the attribute list from within the lexicon builder environment.

The design of the user interface took into account both the available facilities and the expected users. The lexicon builder runs on a VAX 11-750, normally accessed with line-editing terminals. This suggests that a single-line command format is most appropriate. Since much of the work with the system is done over 300 baud telephone lines, conciseness is also important. The users have all had some programming experience (though not necessarily very much) so an operating-system-like interface is easy for them to get used to. If the lexicon builder becomes popular, we hope to have the opportunity to develop a more sophisticated interface, perhaps with a combination of features for beginners and more experienced users.

Structure of a lexical entry

A complete lexical entry consists of:

1. The "name" of the entry -- its character-string form.

2. Its sense. We represent senses by simple numbers, not attempting to formally distinguish polysemy and homonymy, or any other degree of semantic difference. The system leaves to the user the problem of distinguishing different senses from extensions of a single sense: that is, where a word has already been entered in some sense, the user must decide whether to modify the entry for that sense or create a new entry for a new sense.

3. Part of speech, or "class." Our classification of parts of speech is basically the traditional classification with some convenient additions, largely drawn from the classification used by Sager in the LSP (Sager, 1981). Most of the additions are to the category of verbs: "verb" to the lexicon builder denotes the stem form, while the third person and past tense are distinguished as "finite verb", and the past and present participles are classified separately.

4. The text of the definition, entered by the user.

At this stage in our work, the definition is not parsed or otherwise analyzed, so its presence is more for purposes of documentation than anything else. In future versions of the lexicon builder, the definition will play an important role in constructing the entry but in the entry itself will be replaced by information derived from its analysis.

5. A list of attributes (or semantic features), each with its value, which may be binary or scalar.

6. A predicate calculus definition. For example, for the most common sense of the verb "promise", the predicate calculus definition is expressed as

```
promise(x,y,z) = say(x,w,z)
   _event(y) => w = will happen(y)
   _thing(y) => w = will receive(z,y)
```

or, in freer form,

```
(x promises y to z) = (x says w to z)
    where w =
        (y will happen)
                if y is an event
        (z will receive y)
                if y is a physical object.
```

This is entered by the user.

We have been inclined to think of the relational lexicon as a network, since the network representation vividly brings out the interconnected quality which the relational model gives to the lexicon. Predicate calculus is better in other respects; for instance, it expresses the above definition of "promise" much more elegantly than any network notation could. The two methods of representation have traditionally been seen as alternatives rather than as supplementing each other; we believe that predicate calculus has an important supplementary role to play in defining the core vocabulary of the lexicon, although we are not sure yet how to use it.

7. Case structure (for verbs). This is a table describing, for each syntactic slot associated with the verb (subject, direct object, etc.) the semantic case or cases that may be used in that slot ("agent", "experiencer", etc.), whether it is required, optional, or may be expressed elliptically (as with the direct and indirect object in "I promise!" referring to an earlier statement).

Space is reserved in this structure for selection restrictions. A relational model gives us the much more powerful option of indicating through relations such as "permissible subject", "permissible object", etc., not only what words may go with what others, but whether the usage is literal, a conventional figure of speech, fanciful, or whatever. Selection restrictions do, however, have the virtue of conciseness, and they permit us to make generalizations. Relational arcs may then be used to mark exceptions.

8. A list of zero or more relations, each with one or more pointers to other entries, to which the current entry is connected by that relation.

We find it convenient to treat morphological derivations such as plural of nouns, tenses and participles of verbs, as relations connecting separate entries. The entry for a regularly derived form such as a noun plural is a minimal one, consisting of name, sense, part of speech, and one relational arc, linking the entry to the stem form. The lexicon builder generates these regular forms automatically. It also distinguishes these "regular" entries from "undefined" entries, which have been entered indirectly as target words of relational arcs and which are on the queue accessed by UNDEF, as well as from "defined" entries.

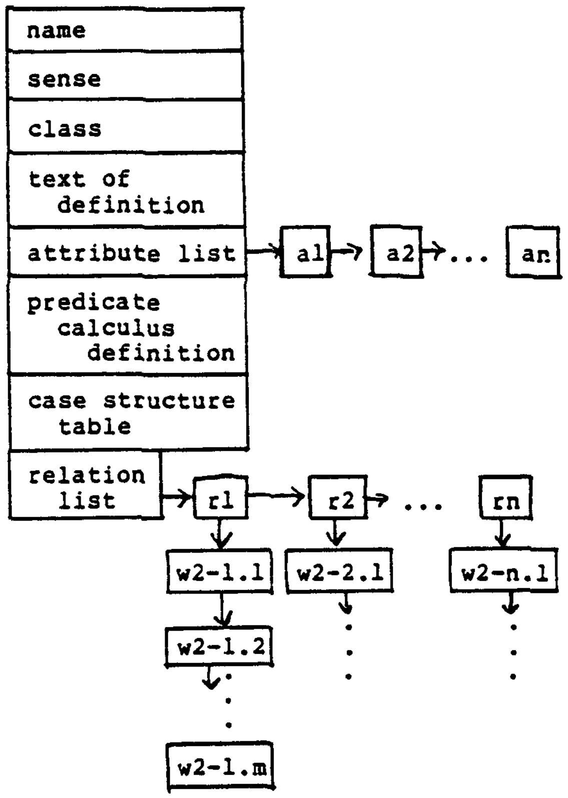

Figure 2. Structure of a lexical entry

File structure of the lexicon

There are four data files associated with the lexicon.

The first is the lexicon proper. The biggest complicating factor in the design of the lexicon is the extremely interconnected nature of the data; a change in one portion of the file may necessitate changes in many other places in the file. Each entry is linked through relational arcs to many other entries; and for every arc pointing from word1 to word2, there must be an inverse arc from word2 to word1. This means that whenever we create a new arc in the course of building or modifying an entry for word1, we must update the entry for word2 so that it will contain the appropriate inverse arc back to word1. Word2's entry has to be updated or created from scratch; we need to structure the lexicon file so that this updating process, which may take place anywhere in the file, can be done with the least possible dislocation.

```
  aphasia (1) n.
definition
  a disorder of language due to injury
to the brain
attributes
  nonhuman
  collective
predicate calculus
  have(x, aphasia) = ~able(speak(x))
relations
  TAX
[aphasia is a kind of x]
      deficit
      disorder
      loss
      inability
  ~TAX
[x is a kind of aphasia]
      anomic
      global
      gerstmann's
      semantic
      Wernicke's
      Broca's
      conduction
      transcortical
  SYMPTOM
[aphasia is a symptom of x]
      stroke
      TIA
  ASSOC
[aphasia may be associated with x]
      apraxia
  _CAUSE
[x is a cause of aphasia]
      injury
      lesion
  NNABLE
[aphasia is the inability to do x]
      speech
      language
```

Figure 3. Lexical entry for "aphasia"

The size of an entry can vary enormously. Regular derived forms contain only the name, sense, class and one relational arc (to the stem form), as well as a certain amount of overhead for the definition, predicate calculus definition and attribute list although these are not used. The smallest possible entry takes up about thirty bytes. At the other extreme, a word may have an extensive attribute list, elaborate text and predicate calculus definitions, and dozens or even (eventually) hundreds of relational arcs. "Aphasia", a moderately large entry with 19 arcs, occupies 322 bytes. Like all entries in the current lexicon, it will be subject to updating and will certainly become much larger.

With this range of entry sizes, the choice between fixed-size and variable-size records becomes somewhat painful. Variable-size records would be highly convenient as well as efficient except for the fact that when we add a new entry that is related to existing entries, we must add new arcs to those entries. The existing entries thus no longer fit into their previous space and must be either broken up or moved to a new space. The former option creates problems of identifying the various pieces of the entry; the latter requires that yet more existing entries be modified.

Because of these problems, we have opted for a fixed-size record. Some space is wasted, either in empty space if the record is too large or through proliferation of pointers if the record is too small; but the amount of necessary updating is much less, and the file can be kept in order through frequent use of the PACK command. The choice of record size is conditioned by many factors, system requirements as well as the range of entry sizes. We are currently working on determining the best record size for the MRH application.

So far the user does not have the option of saving or rejecting the results of a lexicon building session, since entries are written to the file as soon as they are created. We are studying ways of providing this option. A brute force way would be to keep the entire lexicon in memory and rewrite it at the end of the session. This is feasible if the host computer is large and the lexicon is small. The 2000-word lexicon for the Michael Reese stroke database takes up about a third of a megabyte, so this approach would work on a mainframe or a large minicomputer such as our Vax 750, but could not readily be ported to a smaller machine; nor could we handle a much larger vocabulary such as we plan to create with the automatic lexicon builder.

The second file is a directory, showing each entry's name, sense, and status (defined, undefined or regular derivative), with a pointer to the appropriate entry in the lexicon proper. The directory entries are linked in lexicographic order. When the lexicon builder is invoked, the entire directory is read into a buffer in memory, and this buffer is updated as entries are created, modified or deleted. At the end of a lexicon building session, the updated directory is written out to disk.

The third (optional) file is a table of attributes, with pointers into the lexicon proper. This can be extended into a feature matrix.

The fourth (also optional) is a table of pre-defined relations. This table includes, for each relation:

(1) its mnemonic name.

(2) its properties. A relation may be reflexive, symmetric or transitive; there may be other properties worth including.

(3) a pointer to the relation's inverse. If x REL y, then we can define some REL such that y REL x. If REL is reflexive or symmetric, then REL = REL.

(4) the appropriate parts of speech for the words linked by the relation. For instance, the NNABLE relation links two nouns, while the collocational PREP relation links a preposition to a noun. Taxonomy can link any two words (apart from prepositions, conjunctions, etc.) as long as they are of the same part of speech: nouns to nouns, verbs to verbs, etc.

(5) the text of a prompt. ADDENTRY uses this prompt when querying the user for the occurrence of relational arcs involving this relation. For instance, if we are entering the word "promise" and our application uses the taxonomy relation, we might choose a short prompt, in which case the query for taxonomy might take the form

"promise" T: [user enters word2 here]

or we could use something more explicit:

"promise" is a kind of:

Users familiar with lexical-semantic relations might prefer the shorter mnemonic prompt, whereas other users might prefer a prompt that better expressed the significance of the relation.

THE AUTOMATIC LEXICON BUILDER

Building a very large lexicon

There are numerous logistical problems in implementing the sort of very large lexicon that would result from analysis of an entire dictionary, as the work of Amsler and White (1979) or Kelly and Stone (1975) shows. Integrating the lexicon builder with the LSP, and writing preprocessors for dictionary data, will also be big jobs. Fully automatic analysis of dictionary material, then, is a long-range goal.

A major problem in the relational analysis of the dictionary is that of determining what relations to use. Noun and verb definitions rely on taxonomy to a great extent (e.g. Amsler and White, 1979) but there are definitions that do not clearly fit this pattern; furthermore, even in a taxonomic definition, much semantic information is contained in the qualifying or differentiating part of the definition.

Adjective definitions are another problem area. Adjectives are usually defined in terms of nouns or verbs rather than other adjectives, so simple taxonomy does not work neatly. In a sample of about 7,000 definitions from W7, we identified nineteen major relations unique to adjective definitions, and these covered only half of the sample. The remaining definitions were much more varied and would probably require far more then nineteen additional relations. And for each relation, we had to identify words or phrases (the "defining formulas") that signaled the presence of the relation.

The MTQ model

For these reasons as well as theoretical ones, we need a simplifying model of relations, a model that enables us either to avoid the endless identification of new relations or to conduct the identification within an orderly framework. Werner's MTQ schema (Werner, 1978; Werner and Topper, 1976) seems to provide the basis for such a model.

Werner identifies only three relations: modification, taxonomy and queueing. He asserts that all other relations can be expressed as compounds of these relations and of lexical items -- for instance, the PART relation can be expressed, with the help of the lexical item "part", by the relational arcs

Broca's area	T	part
brain	M	part

which say in effect that Broca's area is a kind of part, specifically a "brain-part."

Werner's concept of modification and taxonomy reflects Aristotle's model of the definition as consisting of species, genus and differentiae -- taxonomy links the species to the genus and modification links the differentiae to the genus. A study of definitions in W7 and LDOCE shows that they do indeed follow this pattern, although (as in adjective definitions) the pattern is not always obvious.

The special power of MTQ in the analysis of definitions is that in a definition following the Aristotelian structure, taxonomy and modification can be identified by purely syntactic means. One (or occasionally more than one) word in the definition is modified directly or indirectly by all the other words. The core word is linked to the defined word by taxonomy; all the others are linked to the core word by modification. (Queueing so far does not seem to be important in the analysis of definitions.)

In order to avoid certain ambiguities that arise in a very elaborate network such as that generated from a large dictionary, we have replaced the separate modification and taxonomy arcs with a single, ternary relational arc that keeps the species, genus and differentiating items of any particular definition linked to each other.

The problem of identifying "higher level" relations such as PART and NNABLE in an MTQ network still remains. At this point it seems to be similar to the problem of identifying higher level relations from defining formulas.

Another pleasant discovery is that the Linguistic String Parser, which we have used successfully for some years, is exceptionally well suited for this strategy, since it is geared toward an analysis of sentences and phrases in terms of "centers" or "cores" with their modifying "adjuncts", which is exactly the kind of analysis we need to do.

Design of the automatic lexicon builder

The automatic lexicon builder will contain at least the following subsystems:

1. The standard data structure for the lexical entry, as described for the interactive lexicon builder, with slight changes to adjust to the use of MTQ.

The relation list is presently structured as a linked list of relations, each pointing to a linked list of word2s. ("Word2" refers to any word related to the word ("word1") we are currently investigating.) Incorporating the ternary MTQ model, we would have two relation lists: a T list and an M list. The T list would be a linked list of words connected to word1 by the T relation; its structure would be identical to the present relation list except that its nodes would be lexical entry pointers instead of relations. Each of these lexical entry pointers would, like the relation nodes in the existing implementation, point to a linked list of word2s. The word2s in the T list would be connected to the T words by an inverse-modification relation (~M) and the word2s in the M list would be connected to the M words by inverse taxonomy (~T).

2. Preprocessors to convert preexisting data to the standard form. The preprocessor need not be intelligent; its job is to identify and decode part-of-speech and other such information, separating this from the definition proper.

Part of the preprocessing phase is to generate a "dictionary" for the LSP. This dictionary need only contain part-of-speech information for all the words that will be used in definitions; other information such as part- of-speech subclass and selection restrictions is helpful but not necessary. Sager and her associates (1980) have created programs to do this.

3. Batch and interactive input modules. The batch input reads a data file in standard form, perhaps optionally noting where further information would be especially desirable. The interactive input is preserved from the interactive version of the system and allows the user to "improve" on dictionary data as well as to observe the results of the dictionary parse.

4. Definition analyzer. In this module, the LSP will parse the definition to produce a parse tree, which will then be converted into an MTQ network to be linked into the overall lexical network.

5. Entry generator. This module, like the preprocessor, can be tailored to the user's needs.

SUMMARY

A program has been written that enables a user interested in creating a lexicon for natural language processing to generate lexical entries interactively and link them automatically to other lexical entries through lexical-semantic relations. The program provides a small set of commands that allow the user to create, modify, delete, and display lexical entries, among other operations.

The immediate motivation for the program was to produce a relational lexicon for text generation of clinical reports by a diagnostic expert system. It is now being used for that purpose. It can equally well be used in any other sublanguage environment; in addition, it is intended to be compatible, as far as possible, with models of lexicon structure other than the relational model on which it is based.

The interactive lexicon builder is further intended as the starting point for a fully automatic lexicon building program which will create a large, general purpose relational lexicon from machine readable dictionary text, using a slightly modified form of Werner's Modification-Taxonomy-Queueing relational model.

REFERENCES

Ahlswede, Thomas E., and Evens, Martha W., 1983. "Generating a Relational Lexicon from a Machine-Readable Dictionary." Proceedings of the Conference on Artificial Intelligence, Oakland University, Rochester, Michigan.

Ahlswede, Thomas E., and Evens, Martha W., 1984. "A Lexicon for a Medical Expert System." Presented at the Workshop on Relational Models, Coling '84, Stanford University, Palo Alto, California.

Ahlswede, Thomas E., in press. "A Linguistic String Grammar of Adjective Definitions." In S. WIlliams, ed. Humans and Machines: The Interface Through Language, Ablex.

Amsler, Robert A., and White, John S., 1979. Development of a Computational Methodology for Deriving Natural Language Semantic Structures via Analysis of Machine Readable Dictionaries. Linguistics Research Center, University of Texas.

Evens, Martha W., Ahlswede, Thomas E., Hill, Howard, and Li, Ping-Yang, 1984. "Generating Case Reports from the Michael Reese Stroke Database." Proc. 1984 Conference on Intelligent Systems and Machines, Oakland University, Rochester, Michigan, April.

Evens, Martha W., Vandendorpe, James, and Wang, Yih-Chen, in press. "Lexical-Semantic Relations in Information Retrieval," In S. WIlliams, ed. Humans and Machines: The Interface Through Language, Ablex.

Iris, Madelyn, Litowitz, Bonnie, and Evens, Martha W., unpublished. "The Part-Whole Relation in the Lexicon: an Investigation of Semantic Primitives."

Kelly, Edward F., and Stone. Philip J., 1975. Computer Recognition of English Word Senses. North-Holland, Amsterdam.

Sager, Naomi, 1981. Natural Language Information Processing. Addison-Wesley, New York.

Sager Naomi, Hirschman, Lynette, White, Carolyn, Foster, Carol, Wolff, Susanne, Grad, Robert, and Fitzpatrick, Eileen, 1980. Research into Methods for Automatic Classification and Fact Retrieval in Science Subfields. String Reports No. 13, New York University.

Werner, Oswald, 1978. "The Synthetic Informant Model: the Simulation of Large Lexical/Semantic Fields." In M. Loflin and J. Silverberg, eds., Discourse and Difference in Cognitive Anthropology. Mouton, The Hague.

Werner, Oswald, and Topper, Martin D., 1976. "On the Theoretical Unity of Ethnoscience Lexicography and Ethnoscience Ethnographies." In C. Rameh, ed., Semantics, Theory and Application, Proc. Georgetown University Round Table on Language and Linguistics.