~~To appear~~ In: Uri Zernik (ed): Proceedings of the 1989 IJCAI Workshop on Lexical Acquisition. AAAI Press 1989

Accommodating Complex Applications

Masayo Iida, John Nerbonne, Derek Proudian, and Diana Roberts
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, California 94304-1126

Abstract

APPLICATIONS in natural natural language processing (NLP) need large lexicons; NLP needs therefore to address the problem of providing lexicons quickly and easily. The present contribution focuses on a need to accommodate applications through specialized lexicons. We sketch the need for these and then report on one opportunity for building them: at least databases provide information which can be exploited to produce specialized lexicons. We maintain that the construction of such lexicons can be significantly automated, and demonstrate this on the basis of our own work in automated lexical acquisition; we sketch significant difficulties as well. Finally, we contrast the approach advocated here of exploiting an application for lexical information with the approach of extending and enhancing existing on-line lexicons. We regard this as insufficient by itself, but important and complementary to our own.

1 NLP Background

The development of lexicons for natural language processing (NLP) applications has always been important, and the turn of contemporary theory in linguistics and computational linguistics has made it even more important. Both in linguistics and in NLP the lexicon has traditionally been a locus of MORPHOLOGICAL and and SEMANTIC information. SYNTACTIC information was also present, but in relatively attenuated form. Modern models, on the other hand, focus increasingly on the lexicon as a major repository of syntactic information as well.[1] To cite a single example, whether a verb is used in the passive was once viewed as purely syntactic information, so that little thought was devoted to its representation outside of the grammar;[2] it is now generally accepted that such information requires lexical representation. This only increases the importance of lexicon development for NLP and NLP applications. The advent of structure-sharing in lexicons[3] has made the representation of extensive lexical information relatively efficient, but presupposes that the information (fine-grained word-class membership) is available.

How does one obtain all the lexical information needed for the effective processing of natural language? Here's a model that many researchers seem to assume effective: lexicon development for NLP applications is mainly a matter of building a large application-independent lexicon, and then selecting a subset of words and word senses that are used within the application domain. If this model were generally sufficient, we should all concentrate on developing a single, huge MEGADICTIONARY—a computational *Oxford Advanced Learner's* or *American Heritage Dictionary*, or perhaps something even larger. As the IJCAI '89 workshop on computational lexicons demonstrates, work has already begun in earnest on extracting information from existing hand-held dictionaries. We recognize the need for this effort. Indeed, some application types (e.g. grammar checkers) are served well here, and some sorts of information (e.g. part of speech) are provided reliably, though perhaps imprecisely. Furthermore, we see little (realistic) alternative to megadictionaries for obtaining morphological and syntactic information, especially if this is to be detailed and reliable.

We hold the megadictionary model to be insufficient, however. Exclusive reliance on this model of lexicon development has two shortcomings: First, applications involving specialized vocabulary do not fare well. We discuss an effort to interface to a database of electrical and electronic components below; a great deal of the vocabulary employed by users of this database would not be found in any standard dictionary. Some of the words—*J/K Flip Flop, OR-AND-INV, TTL LS*, or *Schmitt Trigger*—are unavailable in any dictionary, no matter how appropriately specialized or how comprehensive.[4] (Enlarging dictionaries to include all names and specialized vocabulary would result in a document too big to

[1] Cf. [Hoekstra *et al.*, 1981] and the references there for linguistic documentation; [Flickinger *et al.*, 1985] and [Flickinger, 1990] and references there for natural language processing.

[2] Nor does such information reliably appear in traditional dictionaries. Cf. the entry for *have* in [Webster, 1984], which omits any reference to the verb's normal failure to participate in passive constructions. *I have a pencil.* / **A pencil is had.*

[3] Cf. [Flickinger *et al.*, 1985] and [Evans and Gazdar, 1989]

[4] Many words are found in appropriate electronics catalogues, but this is of little help.

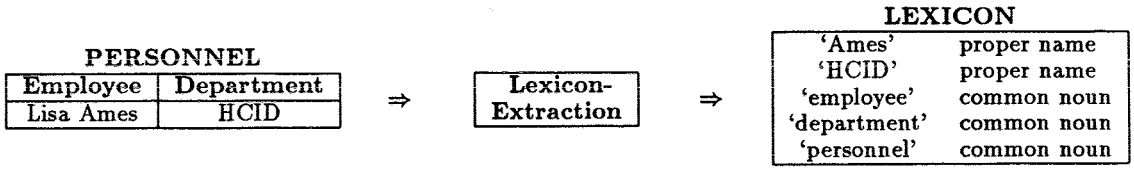

Figure 1: Extracting the Lexicon from the Application. DB table names, column headers and values are extracted and are recorded in the NLP Lexicon together with syntactic and semantic specifications.

be useful.) A megadictionary would require extensive supplementation in this and many other applications.

The second shortcoming of the megadictionary model arises in applications in which NLP serves as a user interface. These applications require a domain semantics mapping which can't be provided within the application-independent model, but which must be appended to it. In these applications it is insufficient to find e.g. *employee* in the dictionary, no matter how expertly and comprehensively it is represented. One must in addition link the lexical entry to information in the application; continuing the example above, we must link the word *employee* to the appropriate database relation (or relations). In Figure 1 this would be the `PERSONNEL.EMPLOYEE` relation. Megadictionaries don't help here.

Opposed to the megadictionary model is the model of APPLICATION-ACCOMMODATION: wherever possible, this model extracts vocabulary from applications, rather than predefined lexicons. This means that we define a LEXICON EXTRACTION procedure for an application. In database (DB) query, we have the scheme in Figure 1 above.

It is worth noting that even databases, which contain a good deal of text, would be poor sources of lexicons if relied on exclusively. Instead, we begin each application effort with the same BASE lexicon, whose task is to list and cross-classify grammatically crucial words—all of what linguists dub the "closed classes"—along with very common words and words which behave exceptionally. These include determiners such as *the*, *every*, and *five*; auxiliary verbs such as *have*, *is*, and *isn't*; pronouns; prepositions; question words; relative pronouns; etc. The application accommodation model then seeks "open class" elements in the application. For the database applications we've examined, moreover, few verbs are present; the open class elements are overwhelmingly common nouns, adjectives and proper names. We consider just the construction of the application-specific lexicon here, the process outlined in Figure 1.

This model of lexicon development uses the application itself as a source for vocabulary, encountering jargon and abbreviations along with standard words in the application, and therefore including them in the application lexicon. And because it uses the application as a source of vocabulary, it partially establishes the mapping from natural language (NL) vocabulary to domain semantics. We turn now to the description and analysis of the application-accommodation model in interfacing an NLP system to relational databases of technical and financial information.

2 Interface to Application Domains

The NLP system used in these application efforts was HP-NL.[5] It is a large LISP system, running in HP-UX (UNIX) on Hewlett-Packard workstations. The system was configured to function as a network server to client application requests. HP-NL had been earlier demonstrated as an interface to toy databases, but we report here on the first attempts to interface to an independent application.

Within the field of NLP interfaces to DB query, the difficulty of the tasks involved in interfacing between an NL system and an application domain is often underestimated. Textbook examples—paradigmatically, personnel databases—involve small DB's of well-understood information, which serve to illustrate some points well, but disguise significant problems. The problems that we concentrate on below arise because we are trying to interface software modules whose construction serves very different needs—databases are constructed to record information reliably and consistently, and NLP systems are constructed to process language. We could easily extend the list of problems below if we wished to gripe about poor database construction, since this can be a significant factor. We surely can't lose sight of the fact real databases have flaws that interfaces must accommodate. We risk belaboring the obvious to emphasize an important point: databases are artifacts of the imperfect collaboration of human engineers in a developing technology. Their interfaces will be leaky. But our focus below is interfaces problems that are likely to arise in any significant database interface effort.

We targeted two rather different databases. The first database we targeted contains information about 2.5×10^5 parts used in design, manufacturing, and inventory and quality control; it contains information such as price, supplier, usage, fault tolerance, size, and functional and physical characteristics. We focused on a subdatabase of digital integrated circuits (IC's) to exploit the availability of cooperative users. The database was an attractive target because its information was valuable; because non-NL access (available through some menus and through the database query language SQL) was incomplete and usually difficult; and because its users needed the information crucially but infrequently

[5] HP-NL is described in [Nerbonne and Proudian, 1987], and in [HP-NL, 1986].

(so that extensive training in the database query language was not an attractive option). Moreover, the database had design features that recommended it for NLP interface experimentation. These features included the support of SQL as a query language, availability of DB partitions, and the use of a data dictionary glossary.[6] In spite of these features, the NL interface was extremely difficult to construct.[7]

The second database contains information about software products and customers. The product information includes a generic name and several options; the customer information includes name, type of industry, etc. Even though this database contains familiar information and is relatively small and well-designed, it too proved to have significant and challenging problems.

For presentation purposes we divide these into problems of recognizing grammatical information and problems of understanding database semantics—grammatical and semantic underspecification problems. We discuss solutions to some of these problems where the problems are identified; for others, we propose general solutions in a separate section (Section 4.3).

3 Grammar

HP-NL supports the automatic acquisition of lexical entries within an elaborate word classification scheme, the word class hierarchy.[8] The sophisticated word class hierarchy aids in lexicon extension and acquisition by moving the task of the application lexicon creator from lexical entry specification to lexical entry classification within the lexical hierarchy.

In an application-accommodation model, fully automatic lexical acquisition would transform DB values into lexical entries by creating a lexical entry with the same spelling, and whose semantics is the database table entry. This lexical entry would be a member of an automatically determined word class. Outside this ideal world, several problems arise. Below, we sketch the problems encountered. Where the problems seem difficult but not insurmountable, we indicate some practical steps which will permit at least partial automation of the lexicon acquisition process.

Word Class Membership In some database applications, identifying the proper word class is a simple task: the column in which the record appears predictably indicates word class. For instance, in a personnel database, the records filling the employee column are all associated with proper names, records in the salary column are all numerals, etc. This convenient correspondence between word class membership and database column can not be expected from all databases, however. For instance, in the digital IC table, even experts admitted that the mapping of part of speech to entries within particular columns was inconsistent. Further, it was not clear to non-experts how the words associated with particular records were used in speech, or which word classes they belonged to. Only a few columns specified values using a single word class. Normally, more than one word class was represented. For example, both adjectives (`ASYNCHRO` and `TRANSPARENT`) and common nouns (`IDENTITY` and `LED`) were present within one column; within another column, both common nouns (`HYBRID`) and proper names (`CMOS` and `HTL`) were represented.

Toward Some Solutions: We suggest two ways through which word classification could be made somewhat more automatic.

First, we can identify word class somewhat reliably based on morphology. Words ending in *-ent* (*transparent*) and *-able* (*encodable*) are normally adjectives; words ending in *-er* (*decoder*), or *-or* (*generator*) are normally nouns, etc.

Second, the lexicon acquisition facility could use an on-line dictionary to identify the word class of words originating from the database. The word class found in the dictionary would serve as the word class of the new lexical entry. This approach will still be unable to assign a word class to specialized vocabulary such as *CMOS* and *HTL*. As well, words with the same spelling but a different meaning would cause problems for this approach, such as the database-specific word *LED* (light-emitting diode) and the English past tense verb *led*.

Syntactically Structured Values Some entries consist of more than a single linguistically atomic piece of information; in both the digital IC table and the software products database, entries composed of more than one element were common. These entries thus have internal grammatical structure. Examples are: `TTL-LS compatible`, `COMMON RESET`, `FILTER (ACTIVE)`, and `PLASTIC PIN GRID ARRAY`, from the digital IC table; and `ADVANCELINK-ARABIC`, and `END USER` from the software products database.

Toward Some Solutions: Any simple heuristic for parsing syntactically complex database entries will have a noticeable failure rate.

If a dictionary were used to identify the component, words we could parse some syntactically complex database entries. However, because of potential parse ambiguity and again because not all of the domain-specific words would be found in the on-line dictionary, this approach is also limited.

Spelling The database entries do not always correspond to the commonly-used term. Simple record-to-lexical-entry conversion in this case would create an apparent lexical gap. For instance, the common spelling for an exclusive-nor gate is "X-NOR"; in the digital IC table, the record string is `EXCL-NOR`.

Punctuation Many entries in both the digital IC table and the software products database include arbitrary punctuation. For instance, in the digital IC table we find: `EXCL-NOR`, `R-S/T`, `LATCH INPUT`,

[6] Cf. [Loomis, 1987], pp.367-9 for an explication of glossary functions.

[7] In fact, this interface effort was ultimately abandoned, partly because of its difficulty.

[8] The lexical hierarchy of HP-NL is the special subject of [Flickinger *et al.*, 1985].

METAL PACKAGE (1W OR LESS), and in the software products database are ADVANCELINK-ARABIC and ADVWRITE PL/GERM. A lexical acquisition plan must include some appropriate treatment of this punctuation which will assure that few queries will fail because of a trivial punctuation mismatch, such as an input *latch-input* rather than the database entry's spelling *latch input*.

Toward Some Solutions: The solution to this problem would involve treating all punctuation within a word equivalently. One approach would be to incorporate a spelling checker into the natural language system which could be loaded with entries from the database. The spelling checker could then be instructed to respond not only to common English misspellings, but also to common variations in punctuation (for instance, substituting a "–" for a "/"), so that *excl-nor* would be considered to be the same as *excl nor* or *excl/nor*.

Numeric and Mixed Numeric Records The digital IC table and the software products database both use numerals in two distinct fashions. First, the numeral may actually indicate a number; for these entries, a lexical entry for the number should already exist in the BASE lexicon (perhaps generated by rule). Second, the numeral may be a code, and may thus be used as a lexical item; for instance,

Does AdvanceLink run on a 350?

(where *350* is a kind of computer). In this case, the database entry 350 should appear in the lexicon as the word *350*.

For those columns in which numeric entries are used as numbers, the numeric entries sometimes contain both numbers and words; for instance, a column in the digital IC table includes the values .5-.99K cells, divide-by-12, 10 X 8, and 12K AND UP. These entries contain some information which should not be introduced into the lexicon: the numerals themselves. However, there is information in these entries which should be introduced into the lexicon. First, the non-numeric parts of mixed numeric records contain useful lexical items such as *cell* and *divide*. We want to be able to extract non-numeric words from the database entries. And second, we may want to use patterns found in the database entries to recognize similar new words, such as deriving the *divide-by-n* relation based on the pattern of divide-by-12 and divide-by-2, divide-by-8, etc.

Accidental Gaps Even a very large database only encodes values for items it contains, while users may query about absent items.

4 Semantics

We continue to suppose that we have a software application—here, a database— for which we'd like to construct a natural language interface. We turn now to the opportunities for extracting lexical semantic information from the application. We first sketch two types of lexical semantic information that can be extracted from database applications—the domain semantics mapping and disambiguating sortal information. Neither of these is at all available from dictionaries. We then turn to difficulties that arose in attempting to automate the extraction of lexical semantics from a complex database application, and consider solutions to some of these difficulties.

4.1 Extracting Lexical Semantics

Our design strategy is to assume that DB names—table, column, and value identifiers—would be used in natural language queries.[9] Let us call this the CORRESPONDENCE ASSUMPTION. It's worth noting that there is no reason why the assumption MUST hold. It's quite possible to fill a database with representations of arbitrary form from which no lexical extraction would be possible. The empirical fact seems to be that database designers prefer words (and phrases) as DB names, however. It isn't required that we explain this preference here, but we suspect that using words is simply more direct (which IS related to a software engineering dictum which is explicitly taught—that code should be "self-documenting"). If arbitrary forms were used, they'd have to be interpreted somewhere; what's more, they'd probably have to be interpreted via words and phrases. It's also worth noting that our proposal doesn't depend on the correspondence assumption being valid across databases, but only that it holds for some. In any case, where it holds, we can systematize, sometimes even automate a great deal of the mapping between English and the application domain. Let us examine a sketch of the lexicon extraction procedure in some more detail.

Domain Semantics Mapping

In Figure 2 we see a felicitous instance of the correspondence assumption: the column header **employee** corresponds to a common noun whose application semantics, in turn, is just the (virtual) db-relation consisting of all the individuals in that column. (We include other relations in Figure 2 in order to suggest generality in treating e.g. the semantic relations specified by compound nouns such as *HCID employee*.)

Roughly this same scheme works for several classes of db-identifiers in the digital IC's application. For example, the functional family column contained the values **multiplexer** and **adder**; in this case, the values, rather than the column headers, correspond to common nouns, but the semantic relation is quite similar:

Relations

'multiplexer'	multiplexer(x)	DIGITAL-IC where part-number=x & function-family= multiplexer

[9] Cf. [Harris, 1978] for suggestions on how to exploit common database facilities in order to obtain further semantic information effectively.

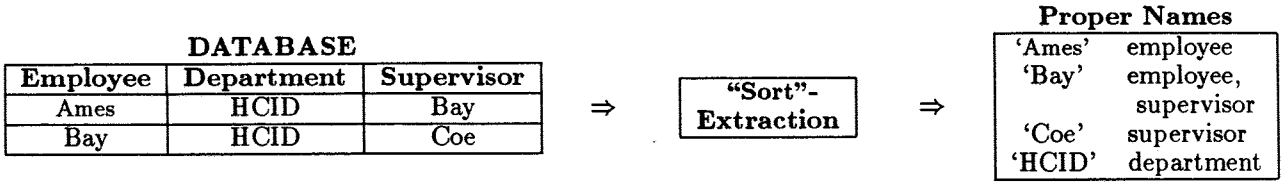

Figure 3: Extracting Disambiguation Information

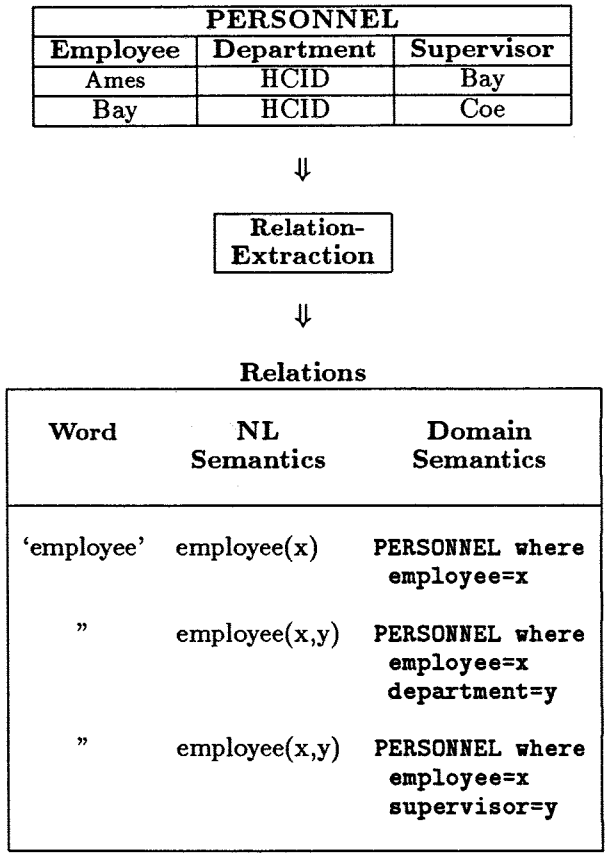

Figure 2: Extracting Relational Information. Some words are represented as logical relations, which in turn map to database relations. NL Semantics and domain semantics are in pseudo-notation.

Disambiguating Information

It turns out that some of the lexical semantics information we extract from the application isn't simply the extension of a term in the application, but rather DISAMBIGUATING information. The key insight exploited here is simple: if we know that HCID is a kind of department, then predications involving HCID must refer to relations involving departments. Since this is effectively noting that HCID belongs to a SORT appropriate for particular relations, we refer to this as the extraction of disambiguating sortal information.

Given the sketch in Figure 2, we would assume that the phrase *HCID employees* could refer to the objects satisfying the DB relation `DEPARTMENT.EMPLOYEE`, where `DEPARTMENT` is bound to `HCID` in that table. But an NLP program must reason through several steps in order to arrive at this domain semantics. To appreciate the difficulty of selecting the correct domain semantics, recall that *employee* can be translated into several distinct database relations (cf. Figure 2 for a simple illustration). For example, in a more complete version of the Personnel Database, it may involve any of the following relations: `SUPERVISOR.EMPLOYEE` (*Jones's employee*, cf. Figure 2), `SEX.EMPLOYEE` (*female employee*), `NATIONALITY.EMPLOYEE` (*foreign employees*), `EMPLOYER.EMPLOYEE` (*Hewlett-Packard employees*), `BIRTHDATE.EMPLOYEE`, `HIREDATE.EMPLOYEE`, etc. To return to the *HCID employees* example, it is only by identifying `HCID` as a possible department (and rejecting it as a possible supervisor) that we are able to settle on the right relation for *employee*. Thus we pick out the `DEPARTMENT.EMPLOYEE` relation and reject e.g. the `SUPERVISOR.EMPLOYEE` relation, even though that relation is instantiated in the database. The extraction of sortal information is sketched in Figure 3.

These two classes of lexical semantic information—domain semantics mapping on the one hand, and potentially disambiguating sortal information on the other—may be systematically extracted from databases as a part of a standard lexical extraction procedure. Since they are both domain-dependent, there would seem no way of obtaining similar information from any standard dictionary, no matter how elaborate or comprehensive.

4.2 Semantics Underspecification

We turn now to difficulties we encountered. Some of these indicated that the correspondence assumption was too simplistic; others arose where the data model used in the database was not explicit or systematic. In each case below, we characterize the difficulty briefly, then provide examples, usually from the Digital IC database.

We postpone discussing solutions until Section 4.3.

Implicit Categorizations Some information is poorly labeled. E.g. several columns in the digital IC table were simply labeled FUNCTIONAL_CHAR_1 ("functional characteristic 1"), etc. The meaning of these categories was obscure not only to the NL interfacers, but also to DB users. Since some values in these columns were also obscure, it was not clear how to interpret them, and therefore not clear how questions concerning them would be phrased.

Other categorizations were implicit but recognizable; this presents difficulties when the categorization involves knowledge not easily accessible to an NLP system. Example: organizational structure in the software products database followed geographical lines closely. Human database users could combine DB information with geographical knowledge to answer questions which NLP could not, such as "Which offices outside of California had the highest sales?". The database simply has no notion of "California".

Incomplete Categorization It was difficult to interpret information when the information label didn't mesh with the range of alternatives available. E.g. FUNCTION_FAMILY included **multiplexer**, but nothing corresponding to *demultiplexer*. The information that a component was a demultiplexer was not represented explicitly, although it could be inferred from values such as 2-TO-4-LINE in one of the functional characteristics columns. (But note that this occurred beside e.g. J-K, EIA, ADDRESSABLE, and 360/370, so that there was no immediate indication that 2-TO-4-LINE referred to input/output specs, rather than a range.)

Semantically Structured Values The values for a given attribute may be semantically structured in many ways, some obvious, some quite subtle. Numerical values, dates and locations have obvious semantic structure; we find less obviously structured fields as well, e.g. a field (in the software products database) specifying a generic name for the software: **DB2, Word-Star, AdvanceLink**, etc. But several values specified what were clearly subtypes of other values, e.g. **AdvanceLink-Arabic**, which is clearly a subtype of **AdvanceLink**, etc. All the various subtypes should clearly be counted as one (together with the simple **AdvanceLink**) if we are to answer certain queries correctly (e.g. "How many AdvanceLink programs were sold in 1988?", "Which AdvanceLink ... ?").

A second type of semantic structure in values is evident in examples such as **TTL-LS compatible**, discussed in another connection in Section 3 above. Here TTL-LS specifies a parameter for the relation *compatible*, disguising a recognized semantics problem, which we sketch now in outline. While English would normally represent a part as standing in the **compatible** relation with TTL-LS, the Digital IC database represents the part as standing in a FUNCTIONAL_CHARACTERISTIC_3 relation to the enumerated value **TTL-LS compatible**. The database represents the relation together with one argument as a single value (effectively "Currying" at this one instance), thus representing this chunk of information more tersely than English does—in contrast to the normal pattern, in which natural language formulations are more terse than database formulations. The normal pattern of translating from natural language to database queries thus involves the expansion of concepts, which is easily reduced to routine. In those cases where the database chooses simpler representations, we encounter a new problem, that of recognizing the more verbose formulations that have to be reduced to database terseness. The problem has been discussed by Moore and others at an ACL Panel where they considered a database of college applicants with a boolean CHILD-OF-ALUMNUS field.[10]

As a third, and distinct, sort of structure, consider the common practice of coding several distinct sorts of information into a single database field. In our software products database, we encountered "standard industrial code" (for customers' business categorization). This is a four-digit field, whose first two digits indicate general field of business—electrical vs. mechanical, etc—and whose following two single-digit fields indicate further specializations. A similar example is the usual practice of coding dates as "MMDDYY" strings, using familiar abbreviations.

"Other" Values This is a special case of the semantically structured values, but one which occurs frequently enough to warrant special mention. Example: If part A's function family is **MISCELLANEOUS**, then it certainly is not an **ACCUMULATOR**, which would be specified as such. It might be a demultiplexer or an optical encoder, however. The use of these values means that other value terms have to be evaluated against the range of all the values used. We found **MISCELLANEOUS**, **OTHER**, and **N/A**.

Records in Mixed Modes In Section 3 we discussed grammatical difficulties these caused, but we also encountered semantic problems wherever essentially numeric information was entered in nonnumeric fashion (usually for a reason). For example, gate arrays' logic size included **.5-.99K CELLS**, **10 X 10**, **16 X 45 X 12**, and **12K AND UP**. String information must be converted and normalized for correct processing.

Unclear Semantics Some information in databases is unclear, and some is poorly organized. We lump these two difficulties together, because both are valuable as cautions, and because they tend to be difficult to distinguish in practice. The interpretation of many fields in the digital IC database was opaque to expert users, let alone NL interfacers. Example: **NAND** was a value in the logic field,

[10] Cf. [Moore, 1982] who sets the problems for the panel; our own approach falls somewhere between [Warren, 1982] and [Scha, 1982].

NL–Domain Correspondences

Type of Semantic Underspecification	NL Semantics	Domain Semantics
Structured Values	advancelink(x)	PRODUCTS where generic-name='AdvLink' OR generic-name='AdvLink-Arabic' OR generic-name='AdvLink-Spanish' OR etc.
Structured Values	june(x)	PRODUCTS where date=y AND substring(y,1,2)= ''06''
Mixed Modes	logic-size(x)>4K	DIGITAL-IC where logic-size=y AND string2numeric(y)> 4000

Figure 4: Some Complex Domain Semantics Correspondences. The appeal to the specialized functions "substring" and "string2numeric" probably disguises a mix of domain and interface processing.

and the number of inputs could be specified. But there was a special ITEM_TYPE field that included the value 2-IN NAND, apparently redundantly. We suspect here that the information was simply poorly recorded, but it is hard to be certain.

4.3 Toward Some Solutions

We feel that some of the problems above will be with us for some time: the problems caused by implicit categorization and unclear semantics, and the problem of simply not obtaining all the terms that "other" may stand for in a given context. But we are more sanguine about addressing other issues. What we next report is essentially work in progress; we are just now implementing some of the ideas below. The crucial point here is that we recognize semantic structure where it exists, and that we allow for complex relations between natural language expressions and database relations. We illustrate this general thesis with some formulations of the complex mappings required to handle the examples in Section 4.2 above. These are shown in Figure 4. In each case we identify a natural language semantic relation and a domain semantics correspondence. Our research tack is to suppose that we can identify a limited number of correspondence TYPES, so that a mapping can be systematically specified, and perhaps partially automated.

The important points here are: first, apparent difficulties in obtaining domain semantics mappings are not insuperable; and second, that genuinely complex domain semantics mappings are required. These are available when lexical semantics are obtained in tandem with interfacing to an application, and they may be systematically available. But they require something like the application-accommodation model in order to be established at all.

5 Incomplete Lexicons

A lexicon acquired automatically will be incomplete. Some information about lexical entries will be incorrect, and some lexical entries may be underspecified. Additionally, the user may use a domain-appropriate word which, because of an accidental gap in the database, does not exist in the derived lexicon. There must be some facility within the NLP system which can accommodate unrecognized words to allow graceful failure and possibly even to provide a sensible answer from the application.

ERRORS in lexical acquisition must be corrected by hand. Underspecified lexical entries may be accommodated in processing by exploiting syntactic and semantic well-formedness constraints; in fact, the technique may be generalized to unknown words, the maximally underspecified lexical entries. In order to parse a sentence containing an underspecified word, we invoke unification as employed in grammar rules and subcategorization restrictions; this may succeed in providing more exact specifications for a word.

Given a domain model, we can similarly assign a partial semantic interpretation to the unknown word based on sortal restrictions imposed on it by the semantic properties of the other known words in the sentence.

6 Application Accommodation

Before turning to the general question of models for lexical acquisition in NLP, it is worth summarizing our view of the prospects for automating lexical acquisition using application information. When—and to what extent—can vocabulary acquisition be automated in natural language interfaces to DB query applications? This is clearly easier with small databases of familiar information than with large databases of specialized information. But we can isolate other factors as well:

1. Vocabulary acquisition is easier where the database uses real terms in common parlance (abbreviations in common use are unproblematic). This simplifies determining information such as spelling and punctuation. Alternatively, a systematic specification of such terms is required.
2. Internally consistent databases with explicit data models eliminate problems of unclear semantics and implicit or incomplete categorization; they reduce reliance on implicit information.
3. An elaborated view of DB/NL correspondence is required to handle problems of syntactically and semantically structured values, including "other" values, mixed modes in value specifications, and the problems of accidental gaps.

Some of the properties which NL requires of DBs would also facilitate the construction of other user interfaces, e.g. menu systems or "forms" interfaces.

7 Models of Lexical Acquisition

We posed the question above in Section 1: How does one obtain all the lexical information needed for the effective processing of natural language? In asking this question, it is important to distinguish methods that promise SOME INCREMENT OF useful lexical information from those which promise some sort of CLOSURE on the lexical acquisition problem—in the case of database interfaces, for example, we'd like to provide the common NL formulations of each significant database relation. This would guarantee that users could ask whatever they liked. Several sources for this lexicon immediately suggest themselves:

Potential Resources for Lexical Acquisition

- users (through corpus studies)
- database experts (through interviews)
- linguistics
- dictionaries

Any corpus of appropriate material will be useful to NLP application builders—whether it be sessions logged from similar work or the results of specific studies. Indeed, we have benefited from user studies performed by colleagues; Hewlett-Packard Laboratories has reported on this work elsewhere.[11] It is unclear how one should systematically obtain any semantics from such studies, and how much material is required for closure. But the studies are useful.

At early stages of the interface effort we relied practically on interviewing database experts—meaning not necessarily database administrators, but rather "power" users, i.e. the *de facto* consultants for typical users. They were able to provide a useful sense of the sort of information normal users wanted and the way they wished to ask for it. But interviews with database experts are even less capable (than user studies) of providing more than anecdotal information about the required application lexicon. Experts navigate adroitly in restricted waters, but they don't pretend to know the entire map.

The usefulness of theoretical linguistics is precisely its systematic view of the lexicon within grammar and vis-à-vis semantics representation. But linguistics sacrifices breadth for depth. Linguistic studies do not provide large lexicons, even if we may be encouraged by recent efforts to create more general purpose lexicons.[12]

The contribution of dictionaries, once they become available in a fashion usable to NLP systems, is potentially enormous: they are the repository of millenia of scholarly work. By "usable" we mean that the dictionaries should have the functionality, say, of databases. If an NLP system encounters the word "cathodic" for the first time, then it needs to know at least that "cathodic" is an attributive adjective, preferably also that it is unlikely to form a comparative or combine with a degree specifier, and that it refers to an electrical process. On-line dictionaries are just now approaching this level of functionality. There is, moreover, still work to be done to link them systematically to other language facilities—it isn't clear what sorts of grammars, parsers, etc. they are compatible with. Furthermore, and this has been the focus of the present paper, DICTIONARIES NECESSARILY EXCLUDE ALMOST ALL SPECIALIZED TERMS AND PROPER NAMES; AND THEY PROVIDE NO SYSTEMATIC CONNECTION TO ANY REFERENCE OF LEXICAL ITEMS—DOMAIN SEMANTICS.

We hasten to reiterate that we don't advocate avoiding the use of any good resource for lexical acquisition, and we believe that user studies, linguistics and lexicography may each have a valuable role to play. Since they appear to be insufficient, even in combination, it is encouraging that another source of lexical information is available for many NLP applications—the application itself. Databases are good examples of the possibilities here, since they normally contain an abundance of textual material.

A Further Resource for Lexical Acquisition

- applications

As Sections 3 and 4.2 demonstrate, the recommendation to employ applications in lexical acquisition cannot be made without qualification. But the digital IC database example is valuable as a "worst-case" scenario. It was a difficult interface task. Even there, however, there was no question but that lexical extraction from the database was the only realistic possibility. Lexical information was wrung from the application at some cost, but there was no realistic alternative.

We've argued above for two positive benefits of extracting lexicons from applications: first, this procedure provides specialized terminology that CANNOT be obtained elsewhere; and second, the procedure links vocab-

[11] Cf. [Whittaker and Stenton, 1989] for an explanation of their work on NLP using the "Wizard of Oz" methodology. Our own natural language system was befuddled by the etiquette vocabulary found in users in Whittaker and Stenton's study—we hadn't thought to provide "please" and "thank you".

[12] Cf. [Russell *et al.*, 1986], which reports on work within the Alvey IKBS project. But note that even this project reached only a lexicon size of approximately $3,500 - 10,000$.

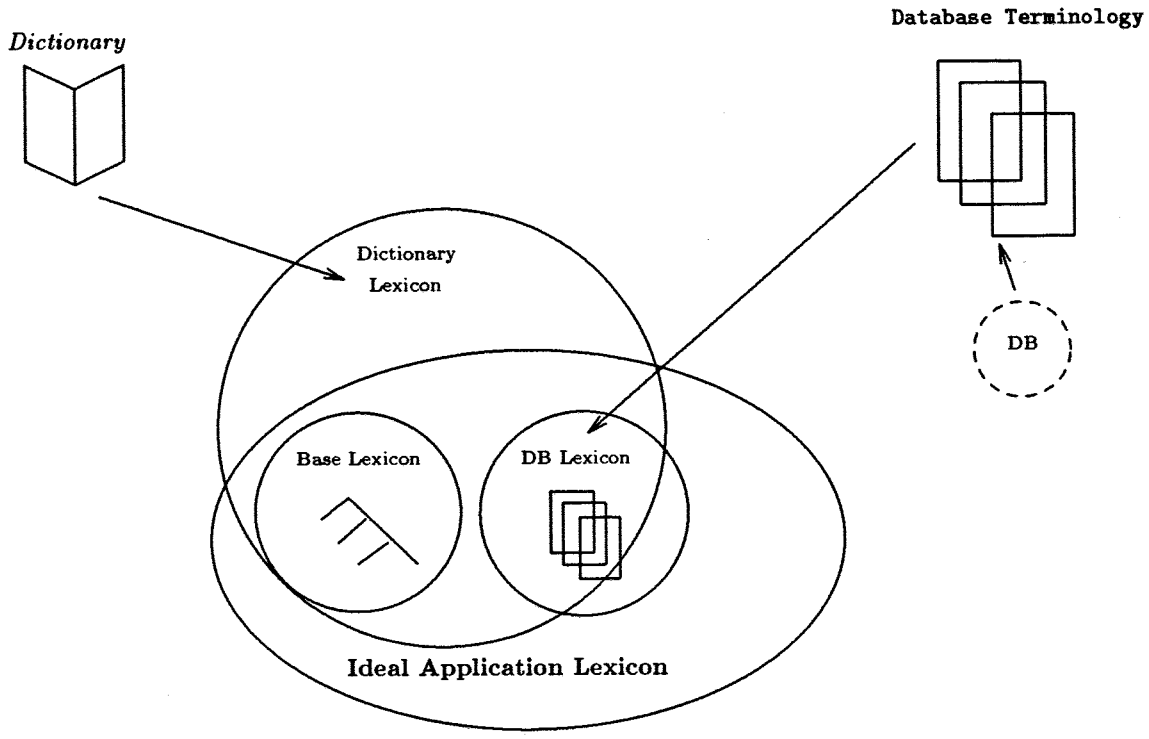

Figure 5: The Lexicon Required for Interfacing to Applications

ulary to application semantics—something no competing methodology can hope to accomplish.

Advantages of Application Accommodation

- includes specialized vocabulary, names
- provides domain semantics

The disadvantages of this methodology are also worth repeating: not all applications provide text from which to derive lexicons; the text wasn't provided for this purpose, and therefore is of variable utility; and there is no guarantee of completeness.

7.1 Lexical Acquisition for NLP Interfaces

Figure 5 sketches our view of the lexical acquisition process for NLP interfaces to databases. A base lexicon, the repository of extensive linguistic knowledge, is supplemented by a data base lexicon which has been extracted from a particular database application. These cover a good deal of the required vocabulary, but not all of it. Moore's "child-of-alumnus" example is useful here:[13] the base lexicon together with the database lexicon would not contain all the vocabulary users would likely employ. Our own base lexicon lacks ALL kinship terminology, for example, and no available linguistically capable lexicon contains all of the information needed to recognize semantic equivalences.[14] The ideal application lexicon—containing just that vocabulary that users will employ about a domain—must be further supplemented. We surely welcome the resources of computational lexicography in filling the gaps.

[13] See Section 4.2 above for discussion.

[14] Even though some may contain the required words—the semantic problem will remain.

Acknowledgements

We thank colleagues in the Natural Language Project and in the Human-Computer Interaction Department at Hewlett-Packard Laboratories for discussion and criticism of the ideas presented here, in particular Susan Brennan, Nancy Kendzierski, Brett Kessler, Joachim Laubsch and Tom Wasow.

References

[Evans and Gazdar, 1989] Roger Evans and Gerald Gazdar. Inference in datr. In *Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics*, pages 66–71, 1989.

[Flickinger *et al.*, 1985] Daniel Flickinger, Carl Pollard, and Thomas Wasow. Structure-sharing in lexical representation. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, 1985.

[Flickinger, 1990] Daniel Flickinger. Natural language processing: Lexical organization. In *Oxford Encyclopedia of Linguistics*, 1990.

[Harris, 1978] Larry R. Harris. Using the data base itself as a semantic component to aid in the parsing of natural language data base queries. Technical Report TR77-2, Department of Mathematics, Dartmouth College, Hanover, New Hampshire, 1978.

[Hoekstra *et al.*, 1981] Teun Hoekstra, Harry van der Hulst, and Michael Moortgaat. *Lexical Grammar*. Foris Publications, Dordrecht, 1981.

[HP-NL, 1986] Hewlett-Packard Laboratories. *HP-NL User's Guide*, 1986.

[Loomis, 1987] Mary E.S. Loomis. *The Database Book*. MacMillan, New York, 1987.

[Moore, 1982] Robert C. Moore. Natural language access to databases—theoretical/technical issues. In *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics*, pages 44–45, 1982.

[Nerbonne and Proudian, 1987] John Nerbonne and Derek Proudian. The hp-nl system. Technical report, Hewlett-Packard Labs, 1987.

[Russell *et al.*, 1986] Graham Russell, Steven Pulman, Graeme Ritchie, and A. Black. A dictionary and morphological analyzer for english. In *Proceedings of the 11th International Conference on Computational Linguistics*, pages 277–279, 1986.

[Scha, 1982] Remko J. H. Scha. English words and databases—how to bridge the gap. In *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics*, pages 57–59, 1982.

[Warren, 1982] David H. Warren. Issues in natural language access to databases from a logic programming perspective. In *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics*, pages 63–66, 1982.

[Webster, 1984] Webster's new world dictionary: Second college edition, 1984.

[Whittaker and Stenton, 1989] Steve Whittaker and Phil Stenton. User studies and the design of natural language systems. In *Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics*, pages 115–123, 1989.