TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces

Barbara J. Grosz, Douglas E. Appelt, Paul A. Martin and Fernando C.N. Pereira
Artificial Intelligence Center, SRI International, Menlo Park, CA 94025, U.S.A.

Recommended by Yorick Wilks and William Woods

ABSTRACT

This article describes TEAM, a transportable natural-language interface system. TEAM was constructed to test the feasibility of building a natural-language system that could be adapted to interface with new databases by users who are not experts in natural-language processing. An overview of the system design is presented, emphasizing those choices that were imposed by the demands of transportability. Several general problems of natural-language processing that were faced in constructing the system are discussed, including quantifier scoping, various pragmatic issues, and verb acquisition. TEAM is compared with several other transportable systems; this comparison includes a discussion of the range of natural language handled by each as well as a description of the approach taken to achieving transportability in each system.

1. Introduction

A natural-language interface (NLI) to a computer database provides users with the capability of obtaining information stored in the database by querying the system in a natural language (e.g., English). With a natural language as a means of communication with computer systems users can frame a question or a statement in the way they normally think about the information being discussed, freeing them from having to know how the computer stores or processes the information. However, most existing natural-language-interface systems have been designed specifically to treat queries that are constrained in two ways: (1) they are concerned with a single application domain, and (2) they pertain to information in a single database.[1] Construction of a system for a

Artificial Intelligence **32** (1987) 173–243
0004-3702/87/$3.50

new domain or database requires a sizeable new effort, almost equal in magnitude to the original one.

Transportable NLIs, i.e., those that can be easily adapted to new domains or databases, are potentially much more useful than domain- or database-specific systems. However, because many of the techniques developed for specialized systems preclude automatic adaptation of the systems to new domains, constructing transportable systems poses a number of technical and theoretical problems. In describing TEAM (for Transportable English database Access Medium), a transportable NLI system that has been the focus of a four-year project (started in 1980), this article discusses several problems of natural-language processing that were faced in its construction and emphasizes those system design choices that were imposed by the demands of transportability. For some of these problems, the design decisions used in TEAM are generally applicable to a broader range of natural-language-processing systems; for others, we were forced to take a more limited approach.

1.1. Transportability

A major challenge in building NLIs is to provide the information the system needs to bridge the gap between the way the user thinks about the domain of discourse and the way information about the domain is structured for computer processing. Existing databases employ different representational conventions, many of them devised to satisfy various database management criteria. For example, one might encode geographic information about mountain peaks in Switzerland as part of a file of information about the mountain peaks of the world, identifying them with an "SWZ" in a COUNTRY field, or using a SWISS? feature field for which a "Y" indicates that a peak is in Switzerland, while an "N" means that it is not. Or the information might be in a separate file on Switzerland, or in one on Swiss mountain peaks. The kinds of queries a user might pose—for example, "What is the highest Swiss peak?" "Are there any peaks in Switzerland higher than Mt. Whitney?" "Where is the Jungfrau?"—should not depend in any way on the representation chosen for the database. An NLI should be able to handle these queries for any of the encodings; the queries all appropriately request information available in the database. Although the English query input to the NLI is the same in all cases, the NLI's output (i.e., specific commands to a database system to retrieve the requested information), will be quite different for the different encodings. The output

[1] The systems are also constrained in a third way: they handle only a single task, namely, database query. This constraint is in many ways more limiting than the other two. For example, queries are typically treated largely in isolation; very few dialogue features are handled. Inasmuch as this third constraint remains a characteristic of TEAM it will not be discussed further in this article.

must reflect the database's actual structures and conventions. One of the main functions of the NLI is to make the necessary transformations and thus to insulate the user from the particularities of the database.

To provide this insulation and bridge the gap between the user's view and the system's data structures requires a combination of domain-specific and general information. In particular, the system must have a model of the application domain's subject matter, including information about the objects in the domain, the properties they possess and their interrelationships, and the words and phrases used to refer to each of these; the system must also know the connection between entities in that model and the information in the database. In constructing transportable systems, it is therefore important to provide a means for acquiring domain-specific information easily.

A major hypothesis underlying TEAM is that, if an NLI is constructed in a sufficiently well-principled manner, the information needed to adapt it to a new database and its corresponding domain can be acquired from users who have general expertise about computer systems and the particular database, but who do not possess any special knowledge about natural-language processing or the particular NLI. In testing this hypothesis, we also assumed that the database could not be restructured. Theoretical and practical motivations underlay this choice. From a theoretical standpoint, it is the most conservative assumption we could have made. It has forced us to adopt general solutions to certain system design issues because we could not restructure the data to alleviate problems of natural-language processing.[2] From a practical point of view, the choice reflected our desire to furnish techniques that could cope adequately with existing databases—some of which, because of their size and complexity, are too difficult to restructure.

1.2. A sample database

Thoughout the rest of the article, we will use the database shown schematically in Fig. 1 to help illustrate various aspects of TEAM. This database comprises four *files* (or *relations*) of geographic data. The first file, WORLDC, has five *fields*—NAME, CONTINENT, CAPITAL, AREA and POP—that together specify the continent, capital, area, and population for each country in the world. The second, BCITY, contains the country and population of some of the larger cities of the world. The third, named CONT, shows the hemisphere, area, and population of the continents. Various mountains in the world are represented in the fourth file, named PEAK, along with the country in which they are located, their height, and an indication as to whether they are volcanoes.

[2] Such restructuring can often result in a closer match between the way information is stored and the way it is referred to in NL expressions. For instance, in the previous example a database structure that includes the SWISS? feature field is more difficult to handle in a general manner than one that uses the COUNTRY field encoding.

WORLDC

NAME	CONTINENT	CAPITAL	AREA	POP
Afghanistan	Asia	Kabul	260,000	17,450,000
Albania	Europe	Tirana	11,100	2,620,000
Algeria	Africa	Algiers	919,951	18,510,000

BCITY

NAME	COUNTRY	POP
Brussels	Belgium	1,050,787
Buenos Aires	Argentina	8,925,000
Canberra	Australia	210,600

CONT

NAME	HEMI	AREA	POP
Africa	S	11,500,000	41,200,000
Antarctica	S	5,000,000	500
Asia	N	16,990,000	2,366,000,000

PEAK

NAME	COUNTRY	HEIGHT	VOL
Anocagua	Argentina	23,080	N
Annapurna	Nepal	26,504	N
Chimborazo	Ecuador	20,702	Y

FIG. 1. Sample database.

Because several files may have fields with the same names, TEAM prefixes file names to field names to form unique identifiers (e.g., WORLDC-NAME, PEAK-NAME, CONT-POP, BCITY-POP); we shall do likewise in the discussion that follows.

TEAM distinguishes among three different kinds of fields—feature, arithmetic, and symbolic. *Feature fields* (e.g., PEAK-VOL and CONT-HEMI) contain true/false values indicating whether or not some attribute is a property of the file subject. *Arithmetic fields* (e.g., WORLDC-AREA and PEAK-HEIGHT) contain numeric values on which computations (e.g., averaging) can be done. *Symbolic fields* (e.g., WORLDC-NAME and PEAK-COUNTRY) typically contain values that correspond to nouns or adjectives denoting the subtypes of the domain denoted by the field.

More information can be gleaned from a database than that directly encoded in its individual files. For instance, the continent on which a peak is located can be derived from two facts in the sample database: the country in which it is located (in field PEAK-COUNTRY) and the continent of which the country is a part (in field WORLDC-CONTINENT). Likewise, the hemisphere in which a country is located can be determined from the continent on which the country is located and the hemisphere of that continent. TEAM allows the database expert to specify *virtual relations* that capture such additional information. In Section 1.3.2 we show both the information that must be acquired to handle the two foregoing examples and the way TEAM accomplishes the acquisition.

1.3. Overview of TEAM

The design of TEAM reflects several constraints imposed by the requirement of transportability. In particular, the need to decouple the representation of what

an end user means by a query from the procedure for obtaining that information from the database affected the choice of system components; likewise, the need to separate the domain-dependent knowledge to be acquired for each new database from the domain-independent parts of the system inevitably influenced the design of specific data structures (or "knowledge sources") selected for encoding the information utilized by these components.

The TEAM system is designed to interact with two kinds of users: a *database expert* (DBE) and an *end user*. The DBE engages in an acquisition dialogue with TEAM to supply the information needed to adapt the system to a new database or to expand its capabilities in answering questions about information in a database to which it had previously been adapted (e.g., by adding new verbs or adjectives or synonyms for existing words). Once a DBE has provided TEAM with the information it needs about a database and domain, any number of end users can use the system to query the database.

The TEAM system thus has two major modes: acquisition and question-answering. The acquisition dialogue with the DBE is organized around the database structure. It is a menu-driven interaction through which the DBE provides information about the files and fields in the database,[3] the conceptual content they encode and how they encode it, and the words and phrases used to refer to these concepts. Hence the DBE must know about the particular database structure and the subject domain its information covers, but does not need to know how TEAM works or any special language-processing terminology.

The question-answering system divides into two major components: the DIALOGIC system [10] for mapping natural-language expressions onto formal logical representations of their meanings, and a schema translator that transforms these logical forms into statements of a database query language. Figure 2 illustrates the major components of this system, the knowledge sources they use, and the flow of language-processing tasks from analysis of an inputted English sentence to generation of a database query. The rectangular boxes represent the components, while the ovals to their right stand for the various knowledge sources. The acquisition box on the right points to those knowledge sources that are augmented through interaction with the DBE. All other modules and knowledge sources are built into TEAM; they are not changed by the acquisition process.

1.3.1. *System components affected by acquisition*

Lexicon. The lexicon is a repository of the information about each word that is essential for morphological, syntactic, and semantic analysis. There are two kinds of lexical items: closed class and open class. Closed classes (e.g.,

[3] TEAM currently assumes a relational database with a number of files. No difficult language-processing problems would result from conversion to other models. The query language into which it translates queries has been used to access other database models.

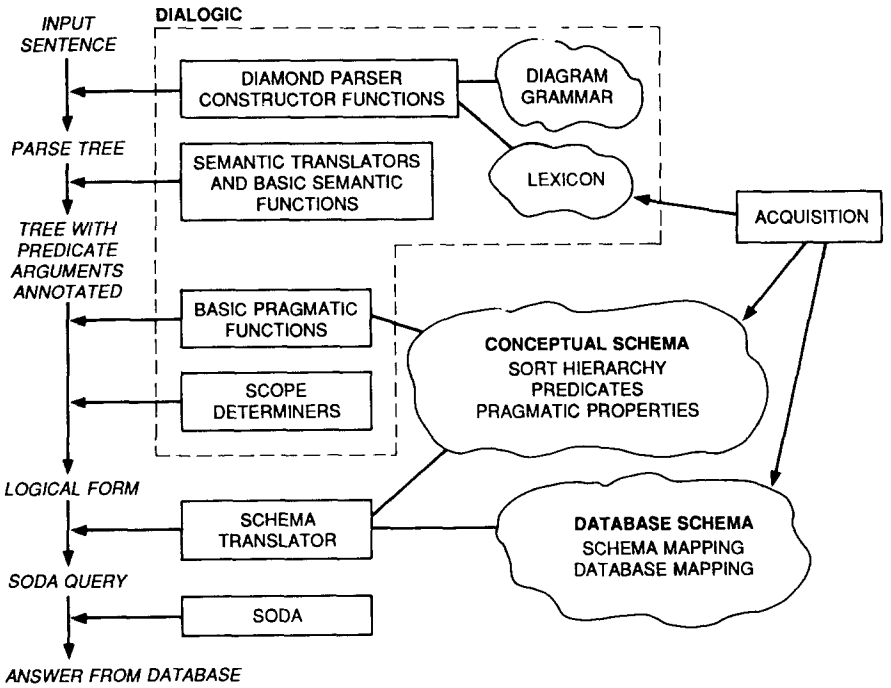

FIG. 2. TEAM system diagram.

pronouns, conjunctions, and determiners) contain only a finite, usually small number of lexical items. Typically, these words have complex and specialized grammatical functions, as well as (at least some) fixed meanings that are domain-independent. They are likely to occur with high frequency in queries to almost any database. Open classes (e.g., nouns, verbs, adjectives) are much larger and the meaning of their items tends to vary according to the given database and domain. Therefore, most closed-class words are built into the initial TEAM lexicon, while open-class words are acquired for each new domain. However, there are a number of open-class words—for example, words corresponding to concepts in the initial conceptual schema (see Section 1.3.1), and those for common measure units, such as meter and pound, that are so widely applicable to so many database domains that they are included in the initial lexicon as well.

The lexical entries acquired by TEAM include those for the names of fields, field values, and file subjects. (File subjects are those entities about which some relation contains information—e.g., peaks for PEAK, and countries for

WORLDC in the sample database illustrated in Fig. 1). In addition, DBEs can provide adjectives and verbs that correspond to various relations and properties represented in the database; they can also supply synonyms for words already acquired (see Section 1.3.2). Associated with each lexical entry is syntactic and semantic information for each of its senses. Syntactic information consists of a primary category (e.g., noun, verb, adjective), a subcategory (e.g., count, unit, mass for nouns; kinds of objects for verbs), and morphological information (e.g., irregular plurals and comparatives). Semantic information depends on the syntactic category. The entry for each noun includes the sort(s) or individual(s) in the conceptual schema (Section 1.3.1) to which that noun can refer. Entries for adjectives and verbs include the conceptual predicates to which they refer and information as to how the various (syntactic) constituents of a sentence map onto arguments of the predicate. Scalar adjectives (e.g., "high") also include an indication of direction on the scale (plus or minus).

Conceptual schema. The conceptual schema contains information about the objects, properties, and relations in the domain of the database. It includes sets of individuals, predicates, and constraints on the arguments of predicates, as well as information needed for certain pragmatic processing (see Section 3.4). The conceptual schema consists of a *sort hierarchy* and descriptions of various properties of nonsort predicates.

The sort hierarchy represents relationships among certain (monadic) *sort predicates* that play a primary role in categorizing individuals (represented here in italic type, as in *PERSON*). TEAM was designed with a considerable amount of this conceptual information built in. Figure 3 illustrates a portion of this initial built-in hierarchy with one new node, *peak-height*, added by acquisition. Each line connecting levels of the hierarchy represents a set-subset relationship between the categories of individuals. The sorts that are connected by the small arcs directly below the nodes are disjoint; that is, no individual can be in two sorts corresponding to nodes joined in this manner. The sort hierarchy grows as information about a database is acquired. The DBE is required to place newly acquired concepts in the appropriate place in the hierarchy and, if they represent sets that are disjoint, to specify accordingly.

Each field in the database is associated with the sort of objects that can appear in that field. Several additional properties are associated with the sorts derived from symbolic fields and from certain kinds of arithmetic fields.

With each sort arising from a symbolic field, TEAM associates a predicate that encodes the relationship between that sort and the sort of the file subject. For example, for the relation WORLDC in Section 1.2, which includes information about capitals and continents, it would link the sort *WORLDC-CAPITAL* with the predicate **WORLDC-CAPITAL-OF** (in this article, predicates will be represented in bold type) that takes two arguments, the first of sort *WORLDC-*

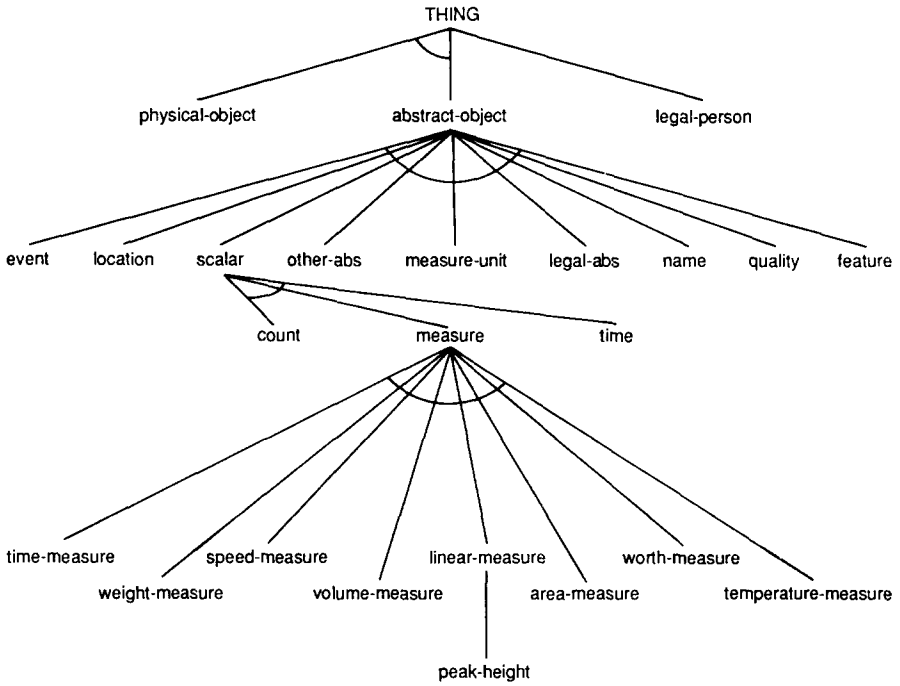

FIG. 3. A fragment of TEAM's sort hierarchy.

CAPITAL and second of sort *COUNTRY*. This link is used in handling queries like "What is the capital of each country in Europe?" In particular, it is used to determine what it means for a capital to be "of" a country or a country to be "in" Europe. Additional properties of the sort indicate whether individual instances of it can serve to modify or stand for instances of the sort of the file subject (e.g., "European countries," but not "Europeans," can be used to refer to the countries *c* satisfying the predication (CONTINENT-OF *c* EUROPE)). These possibilities are discussed in greater detail in Section 3.4.

Sorts that correspond to arithmetic fields containing measures (e.g., length, age) also include information about the implicit unit of measurement (e.g., feet, years), and the kind of measure (e.g., linear extent, temporal extent).

Several kinds of information are associated with nonsort predicates. A *delineation* specifies the constraints on the sorts for each of its arguments. (Multiple delineations are possible; see below and Section 3.5.2.) Predicates corresponding to comparative adjectives (e.g., "tall") have two additional properties: a link to the predicate that specifies the degree (e.g., some length predicate), and an indication of direction on the scale measured (e.g., plus for

TALL, minus for **SHORT**). Many general predicates have semantic and pragmatic specialists associated with them. These are used to determine more precise meanings of words denoting vague predicates (e.g., "have," "of" and genitive constructions) and to treat certain types of modification (e.g., nominal compounds) and coercion; they are discussed in more detail in Section 3.4.

Database schema. The translation from logical form to SODA [19] query requires knowing the exact structure of the target database and how the predicates appearing in the logical form are associated with the relations in the database. This information is provided by the *database schema*, which includes four types of information:[4] (1) definitions of sorts in terms of database relations (subject) or fields (and field value for sorts derived from feature fields); (2) a list of convenient identifying fields for each sort corresponding to a file subject or field; (3) definitions of predicates in terms of actual database relations and attributes; this is done for predicates generated for relation subjects and fields of both actual and virtual relations; (4) a list of the key fields of each relation.

The database schema relates all the predicates in the conceptual schema to their representation in a particular database. For each predicate, the database schema gives a logic formula defining the predicate in terms of database relations. For example, the predicate **WORLDC-CAPITAL-OF** has as its associated database schema a formula representing the fact that its first argument is taken from the WORLDC-CAPITAL field of a tuple of the WORLDC relation, and the second argument comes from the WORLDC-NAME field of the same relation. If a predicate has multiple delineations—i.e., if it applies to different sorts of arguments—(e.g., a **HEMISPHERE-OF** predicate could apply to both *COUNTRIES* and *CONTINENTS*), the schema will include a different definition for each set of arguments. In some cases (e.g., predicates corresponding to some verbs and adjectives), the mapping associated with a predicate indicates its equivalence to another (conceptual schema) predicate with certain arguments fixed; the predicate **NORTHERN** in the second example of Section 2 illustrates this case.

1.3.2. *Acquisition*

The acquisition component of TEAM is crucial to its success as a transportable system. Recall that one constraint on TEAM is that the DBE not be required to have any knowledge of the system's internal workings, nor about the intricacies of the grammar, nor of computational linguistics in general. Yet it is necessary to somehow extract detailed and, frequently, linguistically oriented information from the DBE during acquisition. Furthermore, it is desirable that the acquisition component be designed to allow a DBE to

[4] As described in Section 3.6, the schema translator also uses certain information in the conceptual schema, including taxonomic information in the sort hierarchy and delineation information associated with nonsort predicates.

change answers to questions and add information as he gains experience with TEAM and the types of questions that are asked by end users.

In an attempt to satisfy all these constraints, the menu-oriented system depicted in Fig. 4 was developed. The acquisition system consists of a menu of general commands at the very top, three menus associated with relations, fields, and lexical items respectively, and, at the bottom, a window for replies to questions. The fonts used to display the icons encode some information about the objects they represent. Actual relations are represented in roman type, while virtual relations are represented by italics. An icon appears in boldface when the minimal amount of information about the associated object that is required for responding to queries has been furnished by the DBE. When the DBE mouse selects one of the items from the three menus, a set of questions appears in the question-answering area at the bottom of the display

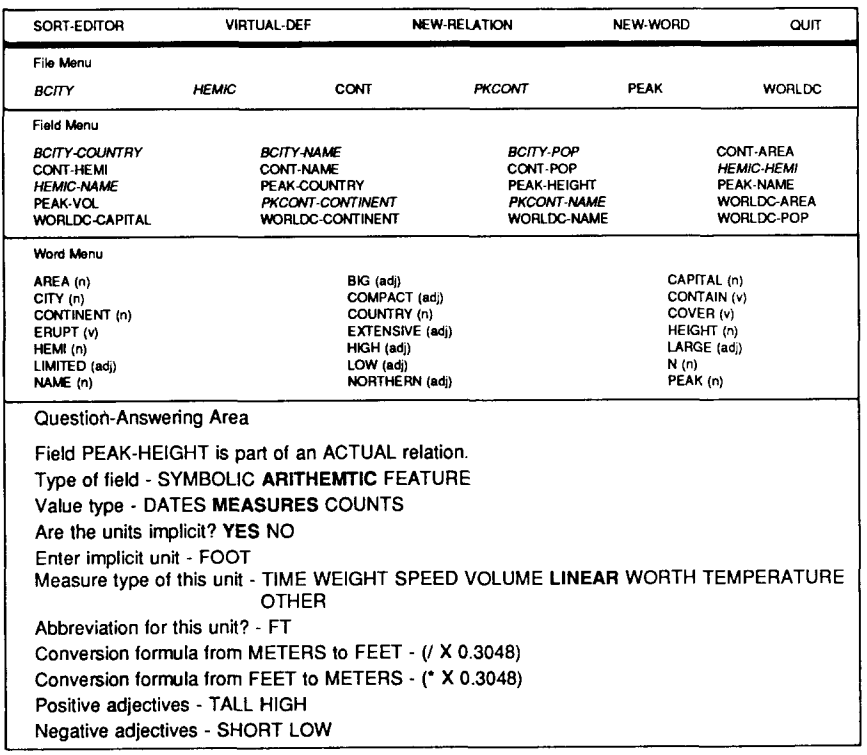

FIG. 4. The acquisition menu.

to which he can then respond. Appendix A lists all of the questions asked by TEAM along with a brief description of the information derived from replies to each.

One of the general principles of acquisition is evident from this display, namely, that the acquisition is centered on the relations and fields in the database, because this is the information most familiar to the DBE. The answers to each question can affect the lexicon, the conceptual schema, and the database schema. The DBE need not be aware of exactly why TEAM asks the questions it does—he need only answer them correctly. Even the entries displayed in the word menu owe their presence to questions about the database. The DBE volunteers additions to this menu only in the case of verbs (described in Section 3.1), to provide an adjective corresponding to some noun already in TEAM's lexicon, or to enter a synonym for some word that is already in the lexicon.

The DBE is assumed not to have any knowledge of formal linguistics or of natural-language-processing methods. He is assumed to know some general facts about English—for example, what proper nouns, verbs, plurals, and tense are—but nothing more detailed than that. If more sophisticated linguistic information is required, as in the case of verb acquisition, TEAM proceeds by asking questions about sample sentences, allowing the DBE to rely on his intuition as a native speaker, and extracting the information it needs from his responses.

Virtual relations are specified iconically. Figure 5 shows the acquisition of a virtual relation encoding the continent (PKCONT-CONTINENT, identified with WORLDC-CONTINENT) of a peak (PKCONT-NAME, identified with PEAK-NAME) by joining the PEAK-COUNTRY and WORLDC-CONTINENT fields. This join captures the information that peaks are on the continent of the country in which they are located. By specifying the join, the DBE enables TEAM to answer questions concerning peaks and the continents on which they are located without requiring that a query include anything explicit about the country in which the peak is located.

The DBE can change previous answers. Incremental updates are possible because most of the methods for updating the various TEAM structures (lexicon, schemata) were written to undo the effects of previous answers before asserting the effects of new ones. Help is always available to assist the DBE when he is unsure how to answer a question. Selecting the question text with the mouse produces a more elaborate description of the information TEAM is trying to elicit, usually including illustrative examples.

Finally, the acquisition component keeps track of what information remains to be supplied before TEAM has the minimum it requires to handle queries. The DBE does not need to figure out what information is sufficient, but only to recognize whether any of the acquisition windows indicate that there are still unanswered questions. Of course, the DBE can always provide information

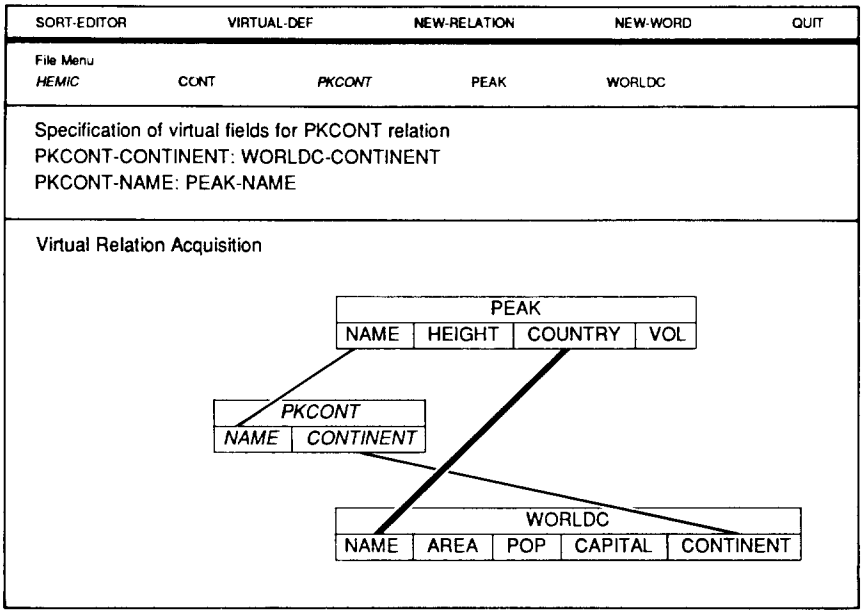

FIG. 5. Acquiring the virtual relation PKCONT.

beyond the minimum required—for example, by supplying additional verbs or synonyms.

1.3.3. *Natural-language processing component*

The flow of control during TEAM's translation of a natural-language query into a formal query to the database becomes evident if the left side of Fig. 2 is scanned from top to bottom. The transformation takes place in two steps: first, the DIALOGIC system constructs a representation of the query's literal meaning, or *logical form* (*LF*) (Section 1.3.4); second, the schema translator transforms this logical form into a database query (Section 3.6).

The DIALOGIC system comprises the following components (shown enclosed within the dotted box in Fig. 2): the DIAMOND parser, the DIAGRAM grammar, the lexicon, semantic-interpretation functions, basic pragmatic functions, and procedures for determining the scope of quantifiers. DIALOGIC is described in more detail in [10], and we can mention here only some of its main features. Various key issues addressed in developing the semantic-interpretation functions, basic pragmatic functions, and quantification procedures are discussed in Section 3. The two data structures in DIALOGIC that are affected by TEAM's acquisition process are the *lexicon* and the *conceptual schema*.

The DIAGRAM grammar [26] is an augmented phrase structure grammar with rules that cover a broad spectrum of constructions, including all common sentence types, complex auxiliaries and modals, complex noun phrases, all the common quantifiers, comparative and measure expressions, relative clauses, and subordinate clauses and other adverbial modifiers along with various constructions that are less common in the database query task[5] (e.g., sentential complements); it also handles a limited range of conjunction. The DIAMOND parser is a bottom-up parser based on Paxton's executive system for the SRI speech-understanding project [21].

DIALOGIC separates semantic-interpretation operations into two main classes: *translators*, which define how the interpretations of the constituents of a phrase are combined into the phrase's overall interpretation, and *basic semantic functions*, which are called by the translators to construct the particular representations of the interpretations of phrases. The translators are tied to the grammar rules (and change as those rules change), but not to the underlying representation (e.g., logical form, network) so that the representation can be changed without affecting the basic translation process.[6]

In brief, when the end user asks a query, DIALOGIC parses the sentence, producing one or more trees that represent possible syntactic structures. The "best" parse tree, based on a priori syntactic criteria, is selected and annotated with semantic information [18, 26]. Next, *pragmatic analysis* (Section 3.4) is applied to assign specific meanings, of relevance for the current domain, to noun-noun combinations and to "vague" predicates like **HAVE** and **OF**.[7] Finally, the quantifier-scope determination process (Section 3.3), after considering all possible alternatives, determines the best relative scope for the quantifiers in the query. The logical form thus constructed constitutes a completely unambiguous representation of the English query, using a set of predicates that are meaningful with respect to the given domain and database.

The sequential processing in DIALOGIC (completing syntactic analysis before semantic processing, etc.) represents a practical rather than a theoretical choice. Although in principle it seems advantageous to bring different types of information to bear throughout processing and, moreover, techniques have been developed for doing so [21, 36], there is as yet no evidence that, given current techniques for applying the different types of knowledge, it is actually more efficient to build a system this way. At present syntactic processing is

[5] DIAGRAM has been used in a number of systems, including text (e.g., the Tacitus project currently under way at SRI) and dialogue systems (e.g., [25]).

[6] Experience with moving from partitioned (semantic) networks to logical form motivated this design [12, 16].

[7] We consider these predicates vague, because their semantic import is highly context-sensitive; the reasons for introducing them, as well as the mechanisms used to handle them, are discussed in Sections 3.4 and 3.5.

sufficiently less costly than semantic analysis as to make sequential processing appear more efficient in practice.

The logical form produced by DIALOGIC is translated into a query in the SODA [19] database query language by the *schema translator* (Section 3.6). In addition to the conceptual schema, the schema translator uses a database schema that provides information about the actual structures used in the particular database being accessed. This schema (discussed in more detail in Section 1.3.1) is also affected by the acquisition process.

Finally, the database query produced by the schema translator is given to the SODA interpreter, which executes the query and displays the answer. SODA was not developed as part of TEAM, but has features consistent with TEAM's overall goal of transportability. It was designed for querying distributed databases and is capable of interfacing simultaneously with several types of database management systems.

The processes TEAM executes in replying to an end user's query are similar to those that any custom-designed NLI would execute. What is different in the case of TEAM is that the modules must be carefully designed to allow for maximum generality, which precludes many of the shortcuts that are common to custom NLI systems (e.g., LADDER [11], PLANES [34]). Two techniques that are consequently ruled out are the use of a *semantic grammar* and the conflation of the meaning of a query with a specification of the procedure for retrieving its answer from the database.

Semantic grammars are based on constituent categories that are chosen not for their ability to represent grammatical regularities, but for the simplification of parsing and interpretation achieved when the grammar reflects the conceptual structure of the database domain. For example, instead of the general categories of "noun" and "verb phrase," semantic grammars may have such categories as "country" and "location specification." Because they do not capture syntactic regularities, such grammars often provide uneven coverage (e.g., an inability to handle passive forms of all queries). Furthermore, adding conceptual coverage (e.g., to cover a new domain) typically requires extensive revision of the grammar.

Efficiency can also be achieved by mapping a natural-language query directly into the code required for retrieving an answer from the database, but at the cost of being locked into a particular database. A number of database query systems (e.g., LADDER) construct a database query directly while parsing the input with semantic grammar rules; they do this without building any other representation of what the query means.

1.3.4. *Logical form*

Logical form plays a central role in TEAM: it mediates between the way an end user thinks about the information in a database, as revealed in queries to the

system, and the way information can be retrieved from the database through queries expressed in a formal database query language. The predicates and terms in the logical form for a particular query are derived from information in the lexicon and conceptual schema;[8] hence the choice of logical form affects the design of those system components indirectly and determines to some extent the information the DBE must supply.

The logical form employed by TEAM is first-order logic extended by some intensional and higher-order operators, and augmented with special quantifiers for definite determiners and WH determiners (e.g., "which" and "what"). Figure 6 gives a BNF-like specification of the logical form used in TEAM. Much research has been done to determine appropriate logical forms for many kinds of sentences [20], but such issues lie beyond the scope of this article.

Although the SODA query that results from the analysis of an English query represents, at least in some sense, the intended meaning of the query, it does so in a way that directly reflects the structure of the database being queried. The reflection of database structure becomes quite evident if one compares the logical forms of queries with the respective SODA forms for the examples of Section 2. As a result, two different databases encoding the same information, but in different structures, will result in two different database queries for the same English question. For example, if an end user asks "How many Swiss mountains are there?" the database queries generated in response to his

```
logical-form :: = = (connective . conditions) | condition
connective :: = = AND | OR | NOT
condition : : = = quantification | predication
quantification : : = = (quantifier variable range scope)
predication : : = = (predicate . arguments)
range : : = = logical-form
scope : : = = logical-form
quantifier : : = = determiner | count
determiner : : = = ALL | SOME | THE | WHAT
count : : = = (NUM number)
predicate : : = = basic-predicate | comparative | superlative
comparative : : = = (comparative-operator adjective)
superlative : : = = ((SUPER comparative) variable range)
comparative-operator : : = = LESS | AS | MORE
  (and minor variations)
argument : : = = variable | constant
```

FIG 6. BNF definition of logical form.

[8] As noted previously, the specific form depends also on general syntactic, semantic, and pragmatic rules for English encoded in the various components of DIALOGIC.

question can look very different, depending on whether the tuples representing Swiss peaks are distinguished from those representing other peaks by their membership in a different relation or by the presence of the character string "SWZ" in a COUNTRY field.

The problem this creates is more than an aesthetic one: for TEAM to obtain through the acquisition process the semantic and pragmatic rules necessary for producing a database query directly from an English query (or information sufficient for generating such rules automatically), it would have to ask the DBE about much more than the structure and contents of the database. Such acquisition would require the kind of expertise in natural-language processing that TEAM is meant to obviate. Thus, the demands of transportability preclude using the SODA language for representing the meaning of queries.[9]

In Section 2 we illustrate the use of some particular logical-form constructions. By showing their use in processing actual queries presented to TEAM, we are able to demonstrate how they relate both to various English expressions and to the SODA query language.

1.4. Outline

In Section 2 we describe how TEAM processes various queries put to our sample database.

Section 3 contains discussions of the major conceptual problems addressed in building TEAM. These fall into three groups. The first includes general features of natural language that are prevalent in database queries (e.g., quantifier scoping, noun-noun combinations, lexical ambiguity, and coordination) and require sufficiently general treatment to permit TEAM to handle queries over a wide range of domains, acquiring any domain-specific information it needs for this purpose. A second group of problems has to do with providing adequate tools for extracting the right kinds of information (e.g., about verbs) from a DBE who has no special knowledge of AI or linguistics. The third group is related to the transformation of the logical representation of a query's meaning into an appropriate and efficient database query; these problems arise in part from the requirement that TEAM's natural-language processing be independent of the details of a particular database structure.

TEAM is one of several recent attempts to build transportable systems. Different approaches to transportable systems reflect diverse conceptions as to what kinds of skills and knowledge might be required of the people doing the adaptations (in particular, whether expertise in natural-language processing is essential), and what parts of the system might be changed (in particular,

[9] In addition, DIALOGIC was designed to be a general language-understanding system that could be (and has been) applied to tasks other than database question-answering. Therefore, it was undesirable to restrict its application by choosing a less general semantic representation.

whether the database could be restructured). In Section 4, we compare several other systems with TEAM from the standpoint of the choices they embody and the effects of these choices on system design.

Section 5 presents our conclusions and suggests several areas for further research.

2. Examples of TEAM in Operation

To illustrate TEAM's processing of natural-language queries, we will consider two queries posed about the information in the sample database given in Fig. 1. These queries also illustrate the role of the various kinds of information utilized by TEAM—whatever has been entered through the acquisition process as well as information built into the core system. The two examples cover only some of the constructions TEAM handles. Section 3 gives more details on the way those and other constructions are treated.

The queries to be discussed are:

Show each continent's highest peak . (1)
What northern countries contain peaks higher than Fuji? (2)

Although (1) is an imperative, in this setting it functions as a question; the initial interpretation preserves the imperative form, but a simplified logical form constructed for the database component converts this to a query. Sentence (1) also illustrates the use of the genitive interpreted to mean containment and the need to determine quantifier scope. Answering it requires information from three of the files—CONT, WORLDC, and PEAK. Note, also, that the schema translator must use the virtual relation PKCONT, generated by joining PEAK and WORLDC, to provide the appropriate database query for this example.

The second query, (2) is a WH question that illustrates TEAM's handling of verbs and comparatives. It too requires that a virtual relation, HEMIC, be used by the schema translator, because "northern" is associated with continents in the actual database, not with countries.

For each of these queries, we will show the sequential steps of TEAM's processing—deriving the parse trees and constructing the logical form, including determining quantifier scope, resolving various pragmatic problems and forming the SODA query.

TEAM initially gets four parses for example (1); these are ranked according to a priori scores given to certain constructions [21, 26], and factors that weigh the likelihoods of various combinations of constituents. The top-ranked parse is shown in Fig. 7. Two parses have the next higher ranking; they correspond to structures that would make sense for queries like the following (elided material is given in square brackets to make the meaning of each more clear):

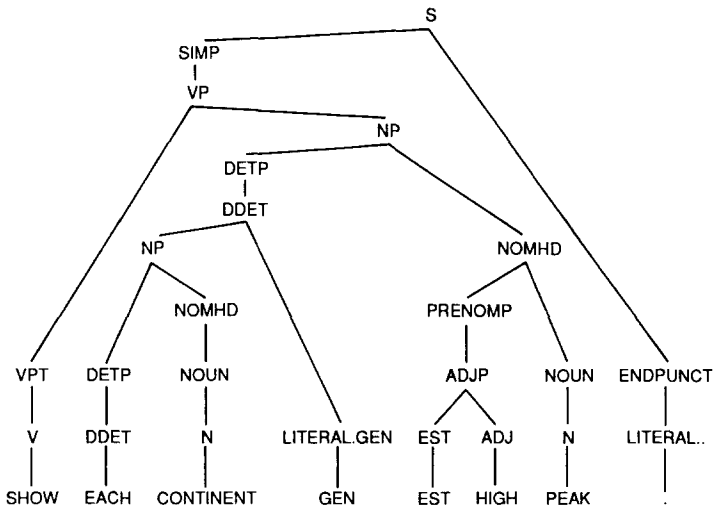

FIG. 7. First parse for example (1).

Show each class' brightest [student the] pictures. (3)
Show each [visitor] Susan's best photograph. (4)

The fourth analysis corresponds to a structure that would account for

Give the student's [puppy] highest marks. (5)

Such utterances presume a discourse context that fills in the material elided from the genitive noun phrase. In this example, the context must supply a set of puppies, one of which belongs to the student.

Several factors contribute to scores for each query that result in these particular rankings. Some of the problematic constructions in the lower-ranked analyses are as follows: the three lower-ranked analyses include determiner phrases that do not contain nouns; the third and fourth analyses use *continent* without a determiner; the fourth also uses *highest peak* with no determiner.

Figure 8 shows the final logical form derived for this query (following the correct syntactic analysis) and an English paraphrase that reflects the chosen quantifier scoping. All the single predications (e.g., (*PEAK* PEAK1)) and relations (e.g., (**PKCONT-CONTINENT-OF** PEAK5 CONTINENT2)) were originally associated with nodes of the parse tree. The pragmatic functions (described in more detail in Section 3.4) change the genitive *continent's* into the relation **PKCONT-CONTINENT-OF**, which encodes the relationship between a peak and the continent containing that peak. The quantifier-scoping algorithm (see Section

```
TEAM > Show each continent's highest peak.

SHOW EACH CONTINENT GEN EST HIGH PEAK.
four parses were found
(IMPERATIVE
  (ALL +WORLDC-CONTINENT+2
       (+WORLDC-CONTINENT+ +WORLDC-CONTINENT+2)
       (THE +PEAK+1
            (AND
              (+PEAK+ +PEAK+1)
              (((*SUPER* (*MORE* *HIGH))
                +PEAK+5
                (AND (+PEAK+ +PEAK+5)
                     (*PKCONT-CONTINENT-OF +PEAK+5 +WORLDC-CONTINENT+2)))
                +PEAK+1))
            (*SHOW +YOU+ +SPEAKER+ +PEAK+1))))

FOR EVERY CONTINENT
  FOR THE UNIQUE PEAK
  SUCH THAT THE PEAK IS THE HIGHEST PEAK SUCH THAT
    THE CONTINENT IS CONTINENT OF THE PEAK
  YOU SHOW SPEAKER THE PEAK.
```

FIG. 8. Logical form of example (1).

3.3) gives the highest ranking to the reading in which *each continent* outscopes *the highest peak* (so that Everest is not the only answer).

Figure 9 shows the simplified logical form that is sent to the database component, its English paraphrase, and the SODA query derived from the logical form by the schema translator (described in detail in Section 3.6). The logical form must be simplified because the language component can represent more kinds of information than the database query language can handle. Thus, as previously noted, imperatives are turned into queries. The simplifier also turns all definites into indefinites; since TEAM does not have a discourse component, indefinites and definites are treated alike.

There is only one parse for example (2), shown in Fig. 10. The predicate adjective phrase *higher than Fuji* is treated as *higher than Fuji [is]*. Whereas *show* in example (1) is a verb that is built into TEAM, *contain* in example (2) is a verb that was learned during acquisition. Such syntactic properties as transitivity and possibility of use in the passive form were derived from answers supplied in the verb acquisition dialogue. Section 3.1 describes this acquisition process in detail, discussing the various kinds of information that need to be acquired.

The logical form for this query is shown in Fig. 11. It illustrates several aspects of semantic and pragmatic processing. First, note that the predicate **NORTHERN** applied to a country uses the virtual relation HEMIC linking coun-

```
Logical form transformed for DB:

(QUERY
  (ALL +WORLDC-CONTINENT+2
       (+WORLDC-CONTINENT+ +WORLDC-CONTINENT+2)
       (WH +PEAK+1
           (AND (+PEAK+ +PEAK+1)
                (((*SUPER* (*MORE* *HIGH))
                  +PEAK+5
                  (AND (+PEAK+ +PEAK+5)
                       (*PKCONT-CONTINENT-OF +PEAK+5 +WORLDC-CONTINENT+2)))
                 +PEAK+1))
           T)))

FOR EVERY CONTINENT
  WHAT IS EACH PEAK
  SUCH THAT THE PEAK IS THE HIGHEST PEAK SUCH THAT
    THE CONTINENT IS CONTINENT OF THE PEAK?

SODA query:
((IN #: $1 CONT) (MAX (#: $3 PEAK-HEIGHT)
                      (IN #: $2 WORLDC)
                      (IN #: $3 PEAK)
                      ((#: $3 PEAK-COUNTRY) EQ (#: $2 WORLDC-NAME))
                      ((#: $2 WORLDC-CONTINENT) EQ (#: $1 CONT-NAME)))
                (? (#: $1 CONT-NAME))
                (? (#: $3 PEAK-HEIGHT))
                (? (#: $3 PEAK-NAME)))
```

FIG. 9. Simplified logical form and SODA query for example (1).

tries and hemispheres through an implicit join of the CONTINENT field of WORLDC and the NAME field of CONT. Whether this predicate applies to countries or to continents is ambiguous, but the decision as to which interpretation is appropriate can be made in one of two places—by DIALOGIC or by the schema translator. Because the decision does not affect the logical form for the query (the semantic and pragmatic processes require only that the argument be one of the appropriate sorts), it is left to the schema translator.

The definition of the predicate **NORTHERN** in the database schema includes the information that the predication (**NORTHERN** *x*) is equivalent to the predication (**HEMIC-HEMI-OF** *x* N) if the sort of *x* is *COUNTRY* and to the predication (**CONT-HEMI-OF** *x* N) if the sort of *x* is *CONTINENT*. In this case, because **NORTHERN** is applied to a country, the first equivalence holds. The predicate definition of **HEMIC-HEMI-OF** states that the predication (**HEMIC-HEMI-OF** *x y*) corresponds to the (virtual) database relation (HEMIC *x y*). The database

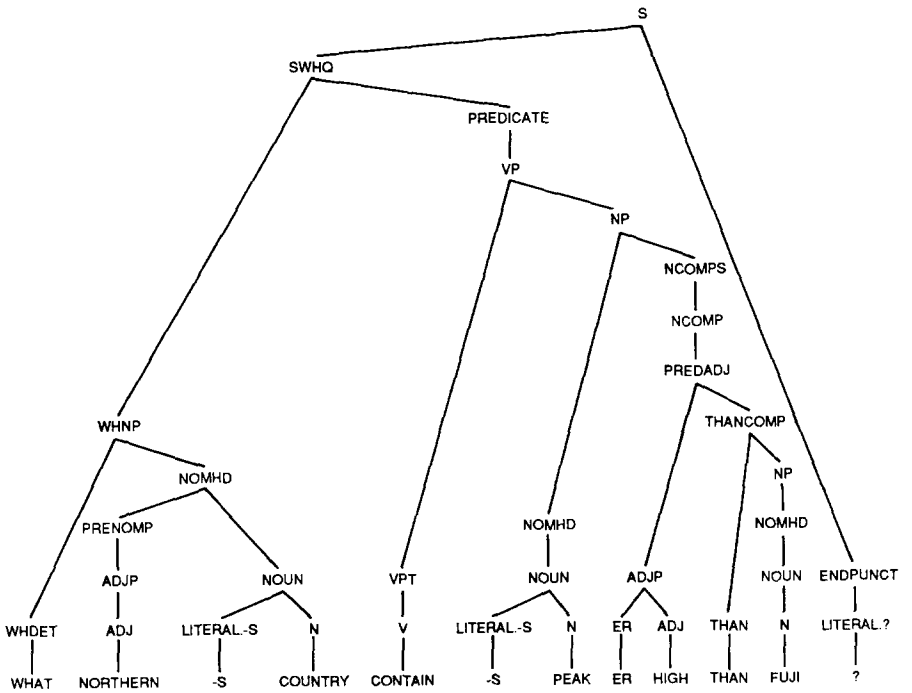

FIG. 10. Parse tree for example (2).

schema also includes the information that this virtual database relation corresponds to the join of (WORLDC $x \ldots z \ldots$) and (CONT $z \ldots y$). Together this information specifies the translation of the logical form into the database query shown in Fig. 12.

Second, the logical-form fragment

(AND(*PEAK FUJI4*)(**PEAK-NAME-OF** *FUJI4* FUJI))

illustrates the coercion (described in Section 3.4) of a name (Fuji) into the thing it names (a mountain); this is necessary because the height predicate is to be applied to the mountain, not its name. We cannot treat the name as being identical to the mountain itself, as it would then not be possible to answer queries like

What is the name of the highest volcano in Japan?

```
TEAM > What northern countries contain peaks higher than Fuji?

WHAT NORTHERN -S COUNTRY CONTAIN -S PEAK ER HIGH THAN FUJI?
exactly one parse was found
(QUERY
  (WH +COUNTRY+1
      (AND (+COUNTRY+ +COUNTRY+1)
           (*NORTHERN +COUNTRY+1))
      (THE +FUJI-4
           (AND (+PEAK+ +FUJI-4)
                (*PEAK-NAME-OF +FUJI-4 +FUJI+))
           (SOME THING3
                (+PEAK-HEIGHT+ THING3)
                (SOME +PEAK+2
                      (+PEAK+ +PEAK+2)
                      (AND (*PEAK-HEIGHT-OF +FUJI-4 THING3)
                           ((*MORE12 *HIGH) +PEAK+2 THING3)
                           (*CONTAIN +COUNTRY+1 +PEAK+2)))))))

WHAT IS EACH COUNTRY
SUCH THAT
  BOTH THE COUNTRY IS NORTHERN
  AND
    FOR SOME HEIGHT AND PEAK
      BOTH THE HEIGHT IS HEIGHT OF THE PEAK NAMED FUJI
      AND THE PEAK IS HIGHER THAN THE HEIGHT
      AND THE COUNTRY CONTAINS THE PEAK?
```

FIG. 11. Logical form for example (2).

Finally, from the information about *contain* gained during acquisition, TEAM determines that the query is asking for peaks that are in countries and not vice versa. The datails of translating the logical form for this example to a SODA query are given in Section 3.6.

3. Specific Problems

This section includes more detailed discussions of several technical problems addressed in the course of constructing the TEAM system. We begin with two constructs that cause problems for acquisition: verbs (Section 3.1) and feature fields (Section 3.2). Verbs are an essential part of natural language; adequate acquisition is an especially difficult task because the DBE cannot be assumed to know explicitly linguistic-theoretical information about their subcategorization. Feature fields encode complex relations between objects and properties;

```
Logical form transformed for DB:

(QUERY
  (WH +COUNTRY+1
      (AND (+COUNTRY+ +COUNTRY1)
           (*NORTHERN +COUNTRY+1))
      (SOME +FUJI-4
            (AND (+PEAK+ +FUJI-4)
                 (*PEAK-NAME-OF +FUJI-4 +FUJI+))
            (SOME THING3
                  (+PEAK-HEIGHT+ THING3)
                  (SOME +PEAK+2
                        (+PEAK+ +PEAK+2)
                        (AND (*PEAK-HEIGHT-OF +FUJI-4 THING3)
                             ((*MORE12 *HIGH) +PEAK+2 THING3)
                             (*CONTAIN COUNTRY+1 +PEAK+2)))))))

WHAT IS EACH COUNTRY
SUCH THAT
  BOTH THE COUNTRY IS NORTHERN
  AND
    FOR SOME HEIGHT AND PEAK
      BOTH THE HEIGHT IS HEIGHT OF PEAK NAMED FUJI
      AND THE PEAK IS HIGHER THAN THE HEIGHT
      AND THE COUNTRY CONTAINS THE PEAK?

SODA query:
((IN #: $1 CONT) ((#: $1 CONT-HEMI) EQ N)
                 (IN #: $2 WORLDC)
                 ((#: $2 WORLDC-CONTINENT) EQ (#: $1 CONT-NAME))
                 (IN #: $3 PEAK)
                 ((#: $3 PEAK-NAME) EQ FUJI)
                 (IN #: $4 PEAK)
                 ((#: $4 PEAK-HEIGHT) GT (#: $3 PEAK-HEIGHT))
                 ((#: $4 PEAK-COUNTRY) EQ (#: $2 WORLDC-NAME))
                 (? (#: $3 PEAK-HEIGHT))
                 (? (#: $4 PEAK-NAME))
                 (? (#: $4 PEAK-HEIGHT))
                 (? (#: $2 WORLDC-NAME)))
```

FIG. 12. SODA query for example (2).

they represent a case in which the gap between the user's conception of the world is quite different from the database encoding of information. Next we discuss some general problems in natural-language processing—e.g., quantifier scoping (Section 3.3), noun-noun modification (Section 3.4), lexical ambiguity (Section 3.5). The methods used for handling these constructions differ from previous work, and were specifically designed to depend only on the kinds of

domain-specific information that could be acquired from a DBE. Finally, we look at a group of problems that arise in the transformation of the logical form for a natural-language query into an appropriate and efficient database query (Section 3.6); in showing how these problems are handled, we provide a detailed description of the schema translator.

3.1. Verb acquisition

Some of the most interesting aspects of the TEAM acquisition process are illustrated effectively by examining the process of verb acquisition, which contributes information to the lexicon, the conceptual schema, and the database schema.

The capability of using a wide range of verbs in queries is essential to a good natural-language interface. Most NLI systems offer the capability of using the two general verbs "have" and "be," thereby making it possible to pose such queries as "What countries have an area greater than 3 million square miles?" and "What peaks are more than 10,000 feet high?" However, there are other (in many cases much more natural) ways of asking the same question that require the use of different verbs. For example, an end user may ask "What countries cover more than 3 million square miles?" If an NLI is capable of answering only the two former queries, then the end user must learn that the latter is not allowed; more important, such constraints frequently reflect the need for the end user to know some details of how information is structured in the database. If the range of "acceptable English" is so small as to exclude most verbs, then the end user is better off learning a formal query language; natural language will not be very useful. Therefore, the ability to acquire and use arbitrary verbs is absolutely essential to TEAM's objectives, regardless of the difficulties involved.

The behavior of English verbs is very complex, a fact reflected in the difficulty of acquiring new ones. For each verb, TEAM must discover how many arguments it has and whether or not these arguments are optional; how the arguments map onto various cases (e.g., subject, object); the kinds of prepositional phrases that can be used with the verb; whether passive, unaccusative, and dative constructions are permitted; and if there are any particles and whether they are separable. The acquisition component asks a series of questions based on a classification of verbs developed by Levin [17]. In addition, if there are any irregular forms of the verb, they must be added to the lexicon. Finally, the database schema must be updated with information that relates the verb predicate to the underlying database.

Fortunately, the fact that the verbs are being used to answer questions about a relational database makes the acquisition problem somewhat simpler. For example, the arguments to the verb predicate will always be values of some field in some relation and therefore refer to objects and not propositions.

Therefore, no verb will ever be used that requires a sentential complement, and TEAM does not acquire such verbs.

To describe how TEAM determines the properties of a verb, we will consider the database illustrated in Fig. 13 and the aquisition of the verb *cover* to relate countries and their areas, as in the question "What countries cover more than 1 million square miles?" Proceeding as illustrated in Fig. 14, the database expert

WORLDC

NAME	CONTINENT	CAPITAL	AREA	POP
Afghanistan	Asia	Kabul	260,000	17,450,000
Albania	Europe	Tirana	11,100	2,620,000
Algeria	Africa	Algiers	919,951	18,510,000

FIG. 13. The WORLDC relation.

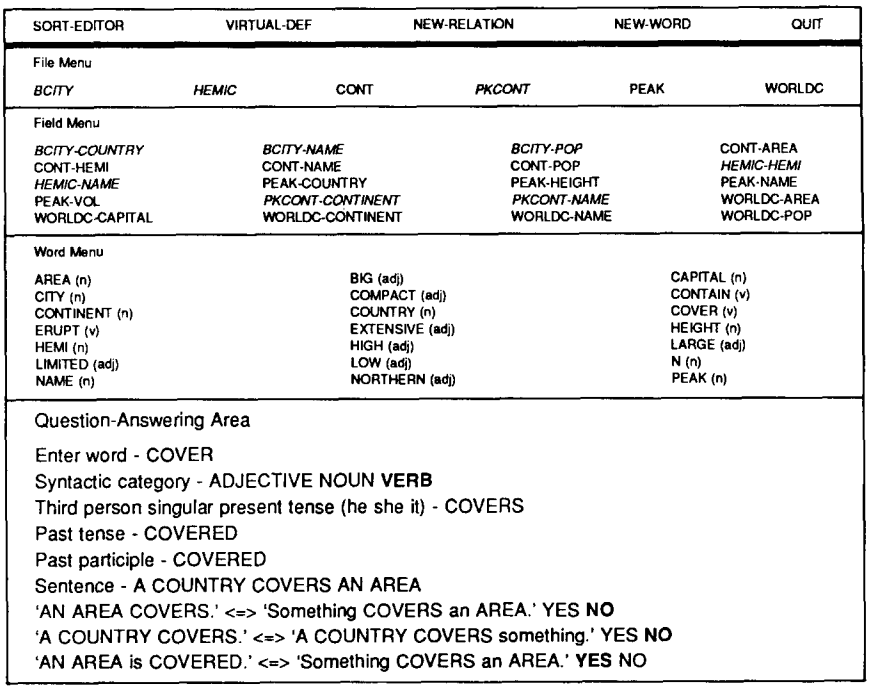

FIG. 14. Acquiring the verb "cover."

enters a new word "cover" and answers the question about its syntactic category by saying that it is a verb. Next TEAM asks three questions to determine the verb's third-person singular, past, and past-participle forms. In this case, TEAM's defaults of "covers," "covered," and "covered" are the correct answers to the respective questions. Any irregularities supplied by the answers are inserted into the lexicon for morphological analysis.

The next question asked is the most important, since TEAM extracts a great deal of both syntactic and semantic information from the answer. TEAM asks the database expert to use the verb in a declarative sentence with indefinite noun phrases in the most general manner in which it can ever be used in a query. In the case of the example under consideration, the database expert uses the active-form sentence "A country covers an area." This response tells TEAM several things: (1) *Cover* is a transitive verb. (2) A new predicate, **COVER**, must be added to the conceptual schema. It has two arguments, the first of sort *COUNTRY* and the second of sort *AREA-MEASURE*. (3) The subject of the active-form sentence is mapped onto the first argument, and the object is mapped onto the second argument.

In a geography database with a relation about rivers, their country of origin, and the body of water into which they ultimately flow, one might acquire the verb *flow* by specifying the sentence "A river flows from a country into a sea." This question would tell TEAM that the conceptual schema is to be augmented by the predicate **FLOW** of three arguments, a *RIVER*, a *COUNTRY* and a *SEA*, that the second and third arguments to the predicate are indicated by the prepositions *from* and *to*, and that *flow* is an intransitive verb.

If a verb can be followed by prepositional phrases, TEAM must discover whether the preposition behaves as a particle or is supposed to mark a prepositional phrase. Verb-particle combinations are quite common in English. For example, one can say "A man drove down the street," but one cannot say "A man drove it down," so in this case, *down the street* is a prepositional phrase. On the other hand, if "A driver ran down a pedestrian," one can say "A driver ran him down;" here *down* behaves as a particle, and *a pedestrian* is the object of the verb rather than the object of a prepositional phrase. TEAM gathers this information by presenting the database expert with a sentence that uses the preposition following a suitable pronoun for the noun phrase referent, and then asking him if the sentence sounds acceptable.

There is still information that TEAM needs to acquire about the verb *cover* before its acquisition is complete. With many verbs, some of the arguments to the underlying predicate are optional and may be omitted in the sentence. In such situations, the mapping of the remaining arguments onto the underlying predicate may be different. TEAM needs to discover what these possible differences are. For example, in a database about wines and their producers, one can acquire the verb *produce* with the sample sentence "A vintner produces a wine in a region?" and later ask "What regions produce Chardon-

nay?" In this case, the region appears in subject position, but it is not the agent in the underlying predicate. The logical form for that query is paraphrased as "What is each region such that, for some vintner, the vintner produces Chardonnay in the region?"

To acquire this information for *cover*, TEAM presents the DBE with some sample sentences and asks if they are correct. TEAM asks whether "A country covers" can mean the same thing as "A country covers something," just as one can say "A man eats" to mean "A man eats something." *Cover* is always transitive, and must have two arguments, so in this case the answer is no. The next question, can "An area covers" mean "Something covers an area," just as "A crop grows" can mean "Someone grows a crop," is answered no for the same reason. The final question, can "An area is covered," mean "Something covers an area," seeks to find out whether the verb can be used in the passive form.

This example of verb acquisition provides one with insight into some of the considerations that were taken into account in adapting the acquisition component to the database expert. The primary strategy is, so far as possible, to focus the interaction on the structure of the database, to extract as much information from each answer, and to avoid confronting the database expert with questions that require any sophisticated knowledge of linguistics or the inner workings of DIALOGIC or TEAM.

3.2. Feature fields

Much of the effort put into developing TEAM has concentrated on the problem of how to bridge the gap between the end user's conceptual view of the domain and the actual structure of the relational database. This problem is especially severe with *feature fields*.

Feature fields occur in relations in which each tuple describes some individual object in the world. A binary feature field can contain two distinct values, depending on whether or not some particular property of the object described by the tuple holds. For example, in the relation describing various mountains of the world depicted in Fig. 1, the field VOL is a feature field that can contain the character Y if the mountain is volcanic, N if it is not.

Feature fields present a particularly difficult problem for a transportable system because of the very large number of alternatives that are possible in English for stating that an object has or does not have a certain property. As an example of an especially troublesome case, one database about military ships had a field DOB, with values Y or N, whose conceptual interpretation was whether the ship described by the tuple did or did not have a doctor on board. In this case the gap between the conceptual schema and the database schema is particularly wide because the database collapses two conceptual predicates—that of being a doctor, and that of being on board a ship—into a single bit of information in the DOB field.

There are quite a few syntactically complex questions that one might ask, such as "Is there a doctor on board each ship in the North Atlantic?" In this query, "a doctor on board" is not a constituent noun phrase; therefore, simply associating a phrase with the conceptual predicate is insufficient. Other queries, like "Is there a doctor within 500 miles of Naples?" require separating the doctor concept from the on board one, and involve more complex reasoning capabilities than are available in most NLIs.

The difficulties this example illustrates are also problems for designers of custom NLIs [23]; they do not arise from transportability. In TEAM, our approach has been to identify as many regularities as possible in the use of feature fields and devise an overall strategy that will bridge the gap between the conceptual and database schemata in as many cases as possible. Four alternative ways of linking conceptual predicates to feature fields have been identified—by means of adjectives, abstract nouns, common nouns, or intransitive verbs—and these seem to suffice for many references to feature fields.

Figure 15 illustrates an acquisition of a feature field. TEAM first prompts to find out the two possible values that can occur in this field. Next the DBE must

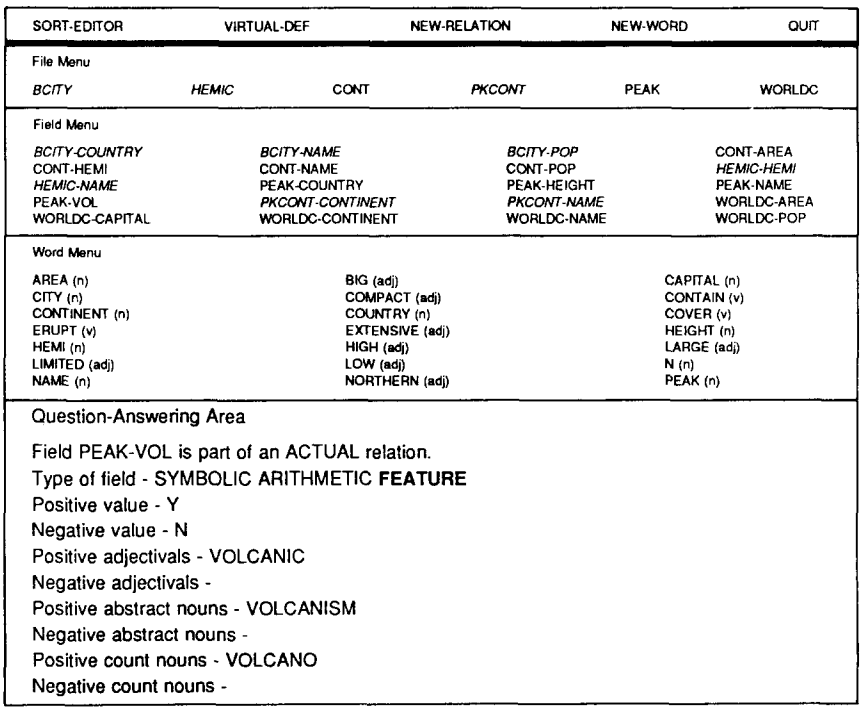

FIG. 15. Feature field acquisition.

specify how end users are going to refer to the feature, which of the four forms of expression are possible.

Adjectivals are used to modify a noun phrase referring to members of the relation's subject sort. The modification is designed to restrict the reference to those members who possess (or do not possess) the property. For example, the adjective *volcanic* applies to any mountain that is described by a tuple with a Y in the VOL field of the PEAK relation.

Abstract nouns refer to some abstract property or quality that members of the relation's subject sort possess if they have the appropriate value in the feature field. A property possessed by a mountain described by a tuple with a Y in the VOL field is *volcanism*, which is supplied in Fig. 15 as an answer to the question about positive abstract nouns.

Each feature field divides the relation's subject sort into two disjoint subsorts, one for each value of the feature. The database expert can specify *common nouns* that refer to individuals of one of these two sorts. For example, if a tuple has a Y in the VOL field, it describes a *volcano*.

Finally, feature fields can be referred to by *intransitive verbs* whose subject is a member of one of the two subsorts induced by the feature field. The option of specifying an intransitive verb is handled by verb acquisition, rather than as part of acquiring the information associated with the feature field itself. Figure 16 illustrates the process of acquiring the verb *erupt*. The database expert first

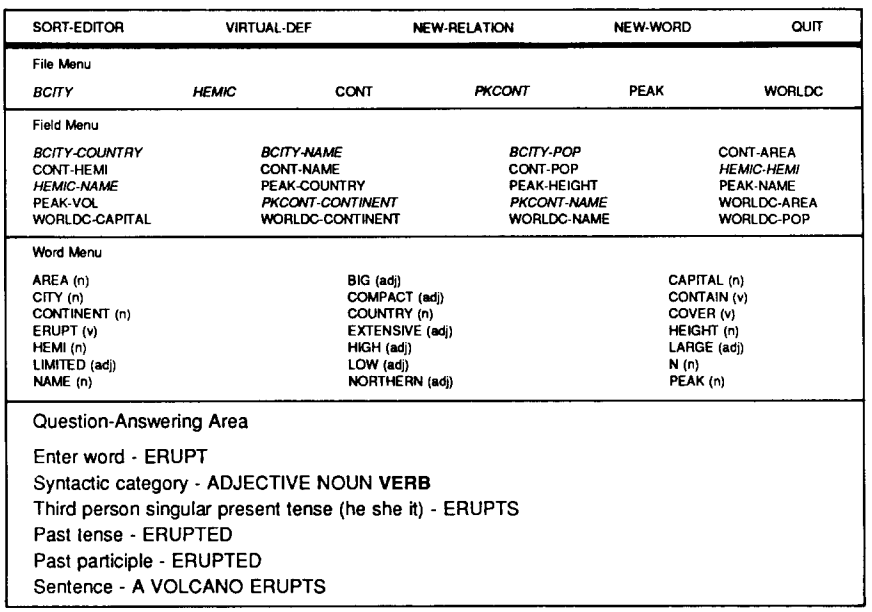

FIG. 16. Acquisition of an intransitive verb.

acquires *volcano* as a common noun for the subsort of mountains represented by Y feature field values. Then, when asked for a sentence for the verb *erupt*, the database expert says "A volcano erupts." From this information TEAM deduces that all things that erupt are volcanos and can then answer questions like "What mountains in Europe erupt?" by listing all the volcanos in Europe.

3.3. Quantifier scoping

One of the problems TEAM must resolve is what relative scope to give to the quantifiers in logical form corresponding to determiners in natural language (e.g., *each*, *every*, *some*, *all*, ...). In fact, scoping applies not only to quantifiers, but also to operators (including negation, tense, and modal markers) and superlatives. While the scope of the LF quantifiers often corresponds to the left-to-right order of the determiners in the surface sentence, other orders are at times intended. Because English is not entirely compositional, the obvious plan of composing the logical form as a result of recursive calls from the translators (and thus rigidly following the tree structure of the syntactic parse) is not adequate to the task. For example, sentences (6) and (7) below have the same parse tree structure, but require different relative scope for the quantifiers of "country."

Show the highest peak in each Asian country. (6)
Show the highest peak in an Asian country. (7)

To determine the appropriate scoping, a method must be employed that uses the information about the tree structure, but balances it with other considerations.

3.3.1. *How TEAM determines quantifier scope*

TEAM uses the syntactic parse tree as the framework from which to derive all the possible scopings; it then ranks these, using additional considerations. The basic strategy follows Hendrix [12], but includes several modifications. TEAM associates two kinds of information with each determiner: a quantifier strength and an associated logical-form quantifier. The overall algorithm is of the classic generate-and-test type; possibilities are generated based on the structure of the parse, and a combined score from specialists (called *scope critics*) provides the test. The test produces a ranking of all candidates rather than rejections of some. The combined score allows any critic to "overrule" any other critic whenever the first critic has a stronger judgment to apply.

During semantic translation, calls by the translators to the basic semantic functions cause logical-form fragments (LFFs) to be associated with certain nodes of the parse tree. These LFF markings include predicates for verbs, adjectives, and prepositions; sort assertions for variables derived from nomi-

nals; quantifier markings derived from determiners, and propositions associated with verb phrases. The quantifier marking includes both a logical-form quantifier and a quantifier strength (see below). LFFs are deposited on the relevant tree nodes rather than being collected into a completed logical form; collecting them compositionally would defeat our efforts to support the non-compositional determiner uses in English.

Throughout semantic processing, about one quarter of the nodes in a typical syntax tree will be marked with an LFF. While the remaining nodes have served as "glue" in the original sentences, only the marked nodes carry information important for semantic processing from this point on. After semantic translation and the first pass of pragmatics (called *first-stage pragmatics*) have been applied, the parse tree is traversed by a process that links up the marked nodes to produce the subtree of interest to the remaining semantic and pragmatic processes, and assigns an ordering to the nodes in this subtree. The traversal (and hence the ordering) is done in a top-down, depth-first, left-to-right fashion; the result is an ordering that corresponds to the default left-to-right interpretation of the determiners in the sentence.

The nodes of the subtree are collected into a list and divided into two categories: those that require relative scoping computations and those whose scoping is determined entirely by their surroundings. The first group of nodes, called *mobile quantifiers*, includes all the nodes derived from determiners. The second set contains all the sort specifications of nominals, all the predications, and all quantifiers that are known to be allowed only the closest possible scoping (e.g., those generated to go with variables associated with proper names). Some nodes can appear in both collected node lists. This is the case for WHNP nodes, like the one for "what" in the sentence "What is the highest volcano?" The WH quantifier and the source of the implicit "thing" from "what thing" are both associated with the same node in the tree.

All the potential alternative scopings are generated by permuting the mobile nodes in the node list and generating the logical form that would correspond to each permutation. Because this method yields $n!$ possible alternative scopings for any query having n quantifiers, restricting consideration to the mobile ones reduces the computational effort significantly. The scoping procedure tries to generate logical forms corresponding to all $n!$ node orderings, but exceptions are taken into account to further constrain the number of possibilities.

The first exception occurs when there is a sequence of existentially quantified variables with nothing intervening (e.g., SOME x . . . SOME y . . .). Because all the permutations would be logically equivalent,[10] the generation routines are designed to generate only one of these equivalent scopings. The other excep-

[10] This is equally true for a sequence of universally quantified variables, and TEAM treats them the same way; however, such sequences occur infrequently.

tion is to rule out those variations in node orderings that cause variables to appear outside the range of their scoping.

An additional filter is applied to any query in which the determiner "any" appears. This filter allows TEAM to handle one of the many difficult behaviors "any" exhibits. The mapping of "any" to a predicate calculus quantifier depends on its immediate context. Either a negation or a query marker within the same clause as "any" changes its mapping. Typically "any" maps into predicate calculus ALL, but in this situation it maps to predicate calculus SOME. This difference is illustrated by the sentence pair:

"I can lick any man in the bar!"→
For ALL man in bar . . . (8)

"Can I lick any man in this bar?"→
Does there exist SOME man . . . (9)

This is a hard rule rather than a preference, so TEAM enforces it at the point of generating the logical form, not in the any-rule critic (discussed in the next section).

3.3.2. *The scope critics*

As a result of the generation process, a set of alternative logical forms is available; all are logically possible interpretations, and TEAM uses other kinds of information to select more closely among them. In particular, each of the relevant scope critics assigns a score reflecting the critic's assessment of the particular proposed ordering of quantifiers (and other operators). Thus, the scope critics are essentially functions which embody TEAM's heuristics for "correct" scoping of English.

The scope critics are specialists; if a particular logical form does not contain an instance of whatever the critic specializes in, it assigns a score of zero. Positive scores indicate increasing approval, negative scores disapproval. The critics currently used by TEAM are *left-right*, *quantifier strength*, *any-rule*, *what-some*, and *superlative*.

The *left-right* critic is relevant to all candidate logical forms. It uses the node ordering assigned earlier by the recursive tree traversal to compute the degree to which the proposed ordering deviates from the default left-right ordering. This critic deducts points for each deviation from the order of the original node ordering; if no other considerations arise, this ordering will be the one preferred.

The *quantifier strength* critic is also relevant to most logical forms. Associated with each determiner is a value that represents its relative strength (with stronger determiners tending to outscope weaker ones). The quantifier

strengths in TEAM are similar to those used by Hendrix [12]. The values are all relative; the ranking imposed by them is illustrated by considering those of some of the most common determiners: bare plural is 0; plural numbered, possessive adjective, and "some" are 5; "the," "a," "either," "every" and "none" are 10; "which," "neither," and "no" are 15; "who" and "what" are 25; "any" is 35. The strength critic uses these values to judge the appropriateness of the relative scoping strength of the quantifiers. It looks for cases in which the scope of one quantifier encloses the scope of another, and it adds points to the overall score if the determiner corresponding to the outer quantifier is stronger than that of the determiner corresponding to the inner one (it deducts points in the converse case). This metric serves to encode the tendency of strong determiners to extend outside the scope that a strictly compositional reading of a sentence would have produced. For instance, although the English determiners *each* and *all* produce the same logical-form quantifier ALL, *each* has a strength of 30 and *all* has a strength of 10. Thus, the quantified variable associated with "each" is more likely to outscope a lexically surrounding expression than if the word used had been "all." Likewise, the quantifier strength critic supplies the knowledge to differentiate the following two queries:

"Who is the boss of all buyers"
(One boss for many buyers implied.) (10)

"Who is the boss of each buyer?"
(Perhaps as many bosses as buyers implied.) (11)

The *what-some* critic is specialized for the database application served by TEAM. Logically the predicate calculus quantifiers WH and SOME are the same. However, the values of a variable bound by a WH quantifier must be shown in the answer to the query, whereas the values of variables bound by SOME quantifiers should not be shown. Therefore, it is preferable to have the WH variable outscope the SOME variable for cases that would otherwise be equivalent; such cases can arise when the left-right critic and the strength critic have competing preferences. This critic ensures that the plain existentials are inside the database restrictions—rather than outside, where they might implicitly cause additional search passes over the database.

The what-some critic is useful in sentences like:

"What peaks have a height greater than 10,000 feet?" (12)

Although other parts of the database-query-generation system might intervene to optimize the query later, the what-some critic endeavors to choose a

logical form close to the form of the optimized database query.[11] The two possible logical forms for this query would differ considerably in their efficiency. In the preferred logical form, peaks are selected and tested to see if their height exceeds 10,000 feet; in the other one, heights above 10,000 feet are generated, and then all peaks are tested to determine whether the generated height coincides with theirs.

Opposed:
(SOME h (*HEIGHT* h)
(WH p (*PEAK* p) (**PEAK-HEIGHT** p h))) (13)

Preferred:
(WH p (*PEAK* p)
(SOME h (*HEIGHT* h) (**PEAK-HEIGHT** p h))) (14)

The *any-rule* critic embodies one of the special rules concerning the English determiner *any*; the correct scoping for a query including this determiner generally requires at least one degree of perturbation of the default left-right scoping order of the sentence. The quantifier in logical form derived from "any" should outscope at least one quantifier that would otherwise surround it. The any-critic tests for this condition in any logical form derived from a sentence using English "any" and adds its vote to the overall scoping score—positively if at least one other quantifier has been outscoped by the quantifier derived from the "any," negatively otherwise.

To see the any-rule critic at work, contrast the rephrasing done by TEAM for the two sentences below. In both cases the quantification of "capital" is existential, but, in the sentence using "any," the variable for "capital" is forced to outscope the variable for "country."

"Is every country governed by any capital?"
For some capital, does the capital govern every country? (15)

"Is every country governed by some capital?"
For every country, does some capital govern the country? (16)

The *superlative* critic applies only to sentences that contain superlative expressions. The critic simply tries to keep the scope of the quantifier of the

[11] This is important, since TEAM provides an English rendering of logical form so that the end user can determine whether the query has been analyzed correctly. Because optimizations by the database query generator will not be shown to the user, queries that seem very inefficient may be reasonable to run despite their apparently poor design.

superlative's head variable (the LF quantifier corresponding to the determiner "the" in a phrase like "the biggest one I ever saw") outside, but as close as possible to, the scope of the superlative itself (the scope that includes all the selectional criteria for the domain within which the superlative applies). This expresses the heuristic that we wish to encourage all the restrictions applying to the head variable of a superlative to reside within the scope of the superlative's domain specification. As an example, consider the following sentence:

"What is the largest country that has a peak taller than Fuji?" (17)

The superlative scope critic will prefer scopings where the pool of countries from which the largest is to be chosen will be exactly those countries containing peaks higher than Fuji. With the superlative critic disabled, TEAM prefers a scoping in which a randomly selected peak that is taller than Fuji is found, and then the largest country that contains that peak is chosen. With the critic, TEAM will give (17) the reading:

What is each country such that
the country is the largest country such that
for some peak that is taller than the height of the peak named Fuji
the country is the country of the peak?

In contrast, without the superlative critic the reading would be:

What is each country such that
for some peak that is taller than the height of the peak named Fuji
the country is the largest country such that
the country is the country of the peak?

Any new rules to determine correct scoping would be added as additional scope critics. The scores from all the critics are summed with a weighting (which is currently neutral) to produce an overall scoping score for each candidate logical form. The alternatives are ranked and presented to the end user, with the highest-ranked one representing the best guess as to correct scoping.

3.3.3. *Scoping problems*

Even the fairly elaborate and comprehensive mechanism in TEAM fails to generate the correct scoping for some sentences. For the hardest of these, pragmatic information specific to the domain may be essential to deriving the correct reading; in some cases pragmatic considerations may override the rules of thumb that are embodied in the TEAM scoping mechanism. For example, if

an end user asks TEAM:

"What is the height of all the peaks in Nepal?" (18)

TEAM will build a query that searches for some unique height that happens to be the height of all the peaks in Nepal. While this is a correct scoping for similar sentences (consider "Who is the commander of all the troops?"), a human would never produce such an interpretation because of the pragmatic information that peaks each have their own individual heights. The acquisition and use of such pragmatic information are beyond the goals of this effort.

3.3.4. *Other scoping algorithms*

Although an exhaustive comparision of quantifier-scoping algorithms lies outside the purview of this article, we can nevertheless compare some aspects of the TEAM algorithm with those of LUNAR [37, 38] and CHAT-80 [22, 35]. These are two other systems that incorporate detailed analyses of the quantifier-scoping problem and in which the problem of selecting plausible scopings has been addressed.

The main difference between the TEAM algorithm and those of both LUNAR and CHAT-80 is that TEAM generates and compares all possible scopings, whereas LUNAR and CHAT-80 apply some set of scoping rules repeatedly to an intermediate representation to produce a final logical form that satisfies all the relevant scoping rules. In other words, LUNAR and CHAT-80 use "first fit" algorithms, whereas TEAM uses a "best fit" algorithm.[12] Unfortunately, the complexity of both kinds of algorithms precludes any analytic comparison, nor is there enough controlled data for a full empirical comparison. It is possible, however, to discuss how certain specific problems are handled by the three algorithms.

As explained above, the TEAM scoping algorithm generates a set of possible scopings and then applies a system of scoring rules uniformly to the scopings to choose the best candidate. In contrast, the scoping mechanisms of LUNAR and CHAT-80 are part of their logical-form construction rules. In LUNAR, scope preferences are encoded in specific rewrite rules that look for certain intermediate representation patterns, such as that corresponding to the interaction between a general determiner and negation. The CHAT-80 scoping algorithm takes an intermediate representation tree with determiners in their original syntax-determined positions, and "percolates" each determiner up the tree as far as it will go, given a table of scope preferences between individual determiners. The order of tree traversal and determiner percolation achieve an effect similar to that of the bubble sort algorithm (with the ordering governed by the relative strengths of the determiners).

[12] By analogy with dynamic storage allocation algorithms [15, Section 2.5].

Syntactic structure and empirical quantifier strength information are insufficient to determine the preferred scoping in all cases. Given this limitation, which is shared by all the algorithms under consideration, the next best comparison criterion for scoping algorithms seems to be the freedom of expression allowed by the algorithm. That is, given an intended meaning, would there be a sentence to which the analysis and interpretation processes, including the scoping algorithm, would assign that intended meaning.

For instance, LUNAR uses the functional nesting of phrases to decide scope (function arguments have wider scope) whereas TEAM and CHAT-80 use left-to-right ordering and quantifier strength to determine scope. Consider the query

Who is the owner of *every* grocery store in Palo Alto? (19)

A system relying on functional nesting will assign the same wide scope to the universal quantifier, independently of the determiner used in that position. However, the intended meaning could be the one in which one is asking whether there is a grocery monopoly in Palo Alto. In systems like TEAM or CHAT-80, it would be possible to convey either meaning by choosing determiners of different strengths:

Who is the owner of *each* grocery store in Palo Alto? (20)
Who is the owner of *all* grocery stores in Palo Alto? (21)

LUNAR and CHAT-80 are stricter than TEAM in the treatment of determiners in relative clauses. TEAM allows orderings in which determiners are moved outside a relative clause to be considered, provided that the ordering does not leave dangling (unbound) variables; LUNAR and CHAT-80 restrict the scope of any determiner from a relative clause to that clause. Both approaches have been proposed in the linguistic literature, but the pertinent evidence is very difficult to evaluate [33].

The three algorithms also differ in their treatment of specific determiners. For instance, TEAM usually translates "any" as an existential quantifier that outscopes at least one other quantifier, whereas CHAT-80 always translates it as a wide-scope universal quantifier. Although the two solutions seem very different on the surface, for most examples they produce the same results. This is because in CHAT-80 a question is translated as the antecedent of an implication. The universal quantifier translating "any" outscopes the implication and binds no variables in the consequent of the implication, which makes the logical form equivalent to one in which the quantifier for "any" is an existential quantifier with just the antecedent of the implication as its scope. The logical form produced by TEAM for the same question will correspond to just the antecedent of the one produced by CHAT-80 and, in most cases the quantifiers will appear in the same order.

LUNAR and CHAT-80 have specialized rules to deal with aggregation words such as "average" and "total" by turning their arguments into constructors of sets on which the aggregation operation is to be performed. This problem was not addressed in TEAM; in fact, the specialized rules it requires do not fit well in a generate-and-test mechanism with uniform scoring.

3.4. Pragmatics and coercion

Although the pragmatic component of TEAM addresses but a small set of issues, it nonetheless does solve a number of problems that are both interesting and essential to the goal of practical language understanding.

3.4.1. *Resolving vague predicates*

English abounds with predicate-producing words (verbs, adjectives, prepositions, and perhaps adverbs) that have a clear predicate-argument structure but do not completely determine a domain-specific predicate. The words "in," "with," "has," and "of" in the following sample queries illustrate this:

"Show the country *in* Europe *with* the highest peak." (22)
"Which continent *has* a peak *with* a height *of* 15,500 feet?" (23)

In TEAM, "of" (and other prepositions that have not been given domain-specific meanings by acquisition), "is," "have" and "do" (the generic verbs), noun-noun pairs, and the genitive all produce predicates that are processed in the syntactic and early semantic modules in the same manner as those predicates that have a domain-specific meaning. These *vague* predicates are then replaced by more specific predicates when later semantic or pragmatic processing succeeds in determining how they are used in a particular query. The predicate chosen is determined by specialist routines that may be called at one of three times: at the end of semantic processing for a clause (*semantic specialists*), before the LFFs are collected from the tree (*first-stage pragmatics*) or after the logical form is collected (*second-stage pragmatics*). The complete list of specialist functions is given in Fig. 17; currently all of the pragmatic specialists are invoked as first-stage pragmatics.

The Is-Semantics specialist is associated with the predicate IS and propagates sort restrictions across all the variables that are being equated by the IS assertion. This specialist is invoked before pragmatic processing (hence the "semantics" label); it attempts to reconcile any conflicts it detects, and may revise some sort predications as a result. For example, it is used in processing the query, "What is the area of Nepal?" to determine that the variable corresponding to the "what" is a *WORLDC-AREA*, not a *CONT-AREA*.

The Degree-Semantics specialist replaces the general predicate DEGREE-OF with a more specific one. This predicate is associated with phrases that query

Semantic specialists:	Is-Semantics
	Degree-Semantics
Pragmatic specialists:	Genitive
	Noun-Noun
	Have
	Of
	General-Preposition
	Time
	Location
	Do
	Comparative

FIG. 17. Semantic and pragmatic specialist in TEAM.

measures (e.g., "How tall is Fuji?" "Is Fuji taller than Everest?"). For example, by determining that the predication (**DEGREE-OF** *PEAK1 x*) refers to the predicate **PEAK-HEIGHT**—i.e., that it is equivalent to the predication (**PEAK-HEIGHT-OF** *PEAK1 x*)—the specialist allows TEAM to further constrain the sort of *x* to be a linear measure, allowing the comparative specialist invoked during pragmatic processing to make the right decisions about comparing the heights of two objects versus comparing an object's height with a given height value.

The Genitive, Noun-Noun, Have, and Of specialists replace the vague predicates **GENITIVE**, **NN** (for noun-noun combinations), **HAVE**, and **OF** with more specific ones. The individual specialists differ only slightly from one another, the differences reflecting special restrictions associated with each construction. The role of these specialists is illustrated in the examples of Section 2; their operation is described below.

The General-Preposition specialist is associated with **ON**, **FROM**, **WITH**, and **IN**, and converts these predicates into appropriate domain-specific predicates. For example, the In specialist determines that the phrase "countries in Asia" means those countries *c* for which (**WORLDC-CONTINENT-OF** *c* ASIA).

The Time specialist and Location specialist serve to map **TIME-OF** and **LOCATION-OF** into predicates that are specifically pertinent to the given database. They can be invoked obliquely by the WH constructions "when" and "where."

The Do specialist replaces the predicate **DO** (from the verb "do") with a specific verb chosen from those acquired for a domain. Although in the database query task "do" does not occur explicitly as the main verb very often, the translators deduce its presence in some queries, as in "What countries cover more area than Peru [does]?"

The Comparative specialist examines the two arguments of a comparison to determine whether it is to be made between two attribute values (e.g., Jack's height and seven feet) or between an entity and some value (e.g., Jack and seven feet). In the latter case, TEAM tries to identify the appropriate attribute of the entity (e.g., Jack's height).

3.4.2. *Acquired relations for pragmatics*

In unrestricted discourse, a vague predicate can stand for a wide variety of specific relations, depending on discourse context. Fortunately, TEAM does not have to generate all these possibilities; the acquisition routines extract from the database expert the specific relations that can be used to replace a vague predicate. Three different kinds of relations are treated: *of-relations*, *modification-relations*, and *coercion-relations*. All these relations are associated with sorts that correspond to various fields in the database.

An of-relation is associated with each sort that corresponds to a field of a file. Such relations are used to interpret phrases that fit the schema: "the ⟨field name⟩ of a ⟨file subject⟩" as corresponding to the LFF (⟨field-name predicate⟩ ⟨file-subject variable⟩ ⟨field-name variable⟩). For example, "height" is the name of a field in the PEAK file given in Section 2; the sort it corresponds to, *PEAK-HEIGHT*, has associated with it as an of-relation the predicate **PEAK-HEIGHT-OF**. As a result, the phrase "the height of Everest" is represented in logical form as (PEAK-HEIGHT-OF EVEREST *height1*). Of course, of-relations provide only one possible interpretation for "of"; if our example were "a height of 10,000 feet," the correct LFF would be (**EQ** *height1* (FOOT 10000)).

This description is somewhat oversimplified in two ways. First, "height" need not be the real name of the field; it could be a synonym for the first name (which, for example, might be HGT). Second, and more important, if the actual field names were always used, the name of the of-relation predicate could always be generated by simply tacking the string "-OF" onto the field name. In most databases containing multiple files, the same sort actually appears in more than one field. Virtual relations always give rise to such cases. The TEAM acquisition for sorts includes facilities for merging multiple names for the same sort; such a merged sort would actually have a list of acquired of-relations rather than the unique value found in the simple case.

Modification-relations are associated with those sorts (derived from symbolic fields) for which the DBE has asserted that an element of the sort *may* be used to directly modify an instance of the sort corresponding to the file subject. These relations are especially important for processing noun-noun pairs and genitive relations. For example, in the WORLDC database, the DBE had to answer a question for the WORLDC-CAPITAL field that asked approximately "Will you want to be able to say 'Paris countries' to mean those countries whose WORLDC-CAPITAL is Paris?" An affirmative answer here would have

caused TEAM to associate with the sort WORLDC-CAPITAL a modification-relation similar to the of-relation. TEAM uses such information to rule out inappropriate uses of modifiers (both explicit and implicit) that would create ambiguities if they were allowed to remain. Because the database expert would refuse to allow the above interpretation of "Paris countries," TEAM is able to reject any parse that assigns a noun-noun construction to that two word sequence. Section 3.4.4 includes more detail on the treatment of such phrases, while Section 3.4.7 discusses some of the limitations of this approach.

Coercion-relations are a variant form of modification-relations; they are not necessarily appropriate for noun-noun substitution, but represent a useful source of relations for genitive, general preposition, and vague verb relations. For example, in a hypothetical automobile registration file, CARS, a DBE might have answered affirmatively the question about implicit modification of the file subject by the MANUFACTURER field, i.e., "Will you want to be able to say 'Fords' to mean 'cars whose CAR-MANUFACTURER is Ford'?" Doing so would cause a coercion-relation to be associated with the sort *CAR-MANUFACTURER* that linked it to the predicate **CAR-MANUFACTURER-OF** and the sort of the file subject *CAR*. This coercion-relation would allow a genitive construction like "Ford's cars" to be interpreted as an LFF like

(SOME *c* (AND (*CAR c*) (**CAR-MANUFACTURER-OF** *c* FORD)) . . .) .

3.4.3. *Changing the sort of a variable*

Some nominals used in English can be forced by their context to take on a meaning related to but different from the one most immediately apparent; we refer to this change as *coercion* from the surface meaning to the semantically required meaning. Coercion occurs whenever some property of an object is used to refer indirectly to the object; naturally this happens most with those properties that tend to select a single object from among the larger range of possibilities in the domain of discourse.

For example, consider a database about employees in which one of the fields for each employee is JOB-TITLE, and one of the job titles is SECRETARY. The word "secretary" is acquired by reading the database, so it is known to be a job title. The query "Which secretaries earn more than their bosses?" can be understood by treating "secretary" as meaning not a job title but an employee whose job title is SECRETARY. This change requires the coercion of the variable representing this nominal from one sort *JOB-TITLE* to another *EMPLOYEE*; the original sort provides a further restriction on the variable.

Rather than record the meaning of such words as ambiguous among this potentially large set of related meanings, TEAM includes specific mechanisms to

support the transformation. These include questions in the acquisition phase to identify legitimate coercions, rules about names and unique key fields, information about syntactic clues that indicate the need for coercion, pragmatic tests to verify that it is indeed required, and a process capable of effecting the coercion on demand. These mechanisms serve the same role, albeit in a less general but more easily customizable way, as the network of known concepts in Ginsparg's NLP [8].

3.4.4. *Acquisition of modification- and coercion-relations*

As was mentioned in Section 1.3.1, TEAM's acquisition asks the database expert whether the values in a symbolic field can be used to modify the file subject either explicitly (e.g., "European countries" to mean countries on the European continent in the sample database) or implicitly (e.g., "Fords" to refer to cars manufactured by Ford in a database of automobiles and their makers). TEAM takes this explicit approach rather than simply allowing all such fields' values to serve as possible modifiers; doing so enables it to rule out some alternative interpretations arising from fields whose values overlap. For example, a file about ships might include fields containing the registry and the destination of each ship. Without explicit acquisition information, TEAM would be unable to decide whether the phrase *U.S. ships* meant "ships of U.S. registry" or "ships whose destination is the U.S." A human could reason that registry is more truly a property of the ship whereas its destination is a more ephemeral one, but TEAM is incapable of such sophisticated reasoning; in the absence of the explicitly acquired information, TEAM would have to treat this phrase as ambiguous.

Despite the need for explicit information about implicit and explicit modifier fields, TEAM does use the natural-language convention that allows names and other similar descriptors to be utilized as either kind of modifier. For example, we can use "John" to mean the person whose name is John, and we can use "1040 form" to mean the form whose identification number is 1040. Any symbolic field that serves as a unique key field for a file is treated by TEAM in the same manner as such names. However, in cases in which more than one field must be conjoined to form the key, no such assumptions are made about any of the key fields. In practice, this rule means that no questions need be answered concerning a symbolic field that is the unique key of a file.

3.4.5. *When does TEAM coerce?*

TEAM recognizes a variety of circumstances requiring (or allowing) coercion.

Syntactic clues may indicate the need for coercion; the case that TEAM can recognize is the use of a proper noun with any indefinite determiner. Using determiners like *what*, *10*, *both*, *some*, or even just the bare plural with a proper noun flags the need to coerce the sort of the noun to something else that is being modified by the proper noun. For example, "both Fords," "what

Boeing," or even "the Joneses" refer to two cars made by Ford, some aircraft made by Boeing, and some group of people who are all named Jones, respectively. TEAM applies this coercion in the basic semantic functions at the time the determiner is combined with the proper noun. The phrase "ten Fords" will produce the logical-form fragment:

((NUM 10) *c1* (AND ((*CAR c1*)(**CAR-MAKER-OF** *c1* FORD))))

Semantic clues may arise at the time a predication is constructed one argument of which is a variable corresponding to a nominal; if the semantics of the predicate clash with the sort of the variable (i.e., if the variable is not of an appropriate sort for the given argument position), coercion may be tried as a means of resolving the conflict. Although "France" is of sort *COUNTRY-NAME* and is used that way in a query like "How large is the country named France?" it must be coerced for semantic reasons in a queries like "How large is France?" and "What area does France cover?" Such coercion will be necessary because the restrictions on the argument for the predicate **COVER** associated with the verb "to cover" require a country, not a country name.

Words that are associated with subsorts of the sort *NAME* are always allowed to be coerced into the things named by them. Although it might seem appealing to assume from the start that a name is linked directly to the thing named, such a direct association makes it difficult to treat correctly expressions in which the name itself is important, e.g., those including the verb "name." By distinguishing a name from the thing named, TEAM can handle queries like "Show the country whose name is the name of a peak." (Which, for the sample database, would yield the response Kenya.) This treatment also enables "name" to be acquired as a normal verb.[13]

Pragmatic clues indicating the need for coercion can arise during pragmatic-processing when there is a clash between the arguments of the predicate chosen to replace a vague predicate and the sort of a filler nominal. This is the same kind of coercion as that triggered by the semantic clues, but for vague rather than domain-specific predicates. However, this coercion must be revocable, because the particular predicate being tried as a replacement for the vague predicate may be an incorrect interpretation. In this case, it must be possible to seek another replacement predicate. Moreover, since the possible replacement predicates are chosen on the basis of the sorts of the variables corresponding to nominals, coercion may be essential to find the correct predicate. This too is necessarily a very tentative step, one done in such a manner as to allow it to be revoked if it fails to yield an acceptable predication.

[13] A name field may be the key field for a file, making such identification of a name with the object named even more appealing. However, this too would cause problems, because multiple-field keys and nonname keys could not be treated uniformly with name field keys.

3.4.6. *When should pragmatics be done?*

The choice as to when a particular pragmatic specialist will run is based on whether quantifier scoping must be done before the specialist can make its decision. If not, the specialist can be invoked when logical-form fragments (*LFFs*) are still in the tree; as a result, it does not have to be reevaluated for each possible scope assignment. The design of TEAM allows some of the specialists to be run during first-stage pragmatics and then, if scoping information is needed to resolve a choice, again during late pragmatics. Although the current pragmatic processing in TEAM is not able to take advantage of this design feature, such a capability is clearly needed. Consider, for example, a database and task in which two interpretations for the term "commander" are possible, namely "captain of a ship" and "chief of the Navy." The query "Who is the commander of every ship?" has two possible interpretations corresponding to the logical forms

(ALL *s* (*SHIP s*)
(WH *c* (*COMMANDER c*) (**SHIP-CAPTAIN** *c s*))) (24)

(WH *c* (*COMMANDER c*)
(ALL *s* (SHIP *s*) (**NAVY-COMMANDER** *c s*))) (25)

Note that the vague predicate **OF** in the LFF (**OF** *commander1 ship1*) is refined into a different predicate in each case. In (24), "ship" has wide scope, so the **SHIP-CAPTAIN** (which pairs a commander with a single ship) predicate is preferred. On the other hand, **NAVY-COMMANDER** (which pairs a commander with a set of ships) is preferred in (25), because commander has wide scope.

3.4.7. *Shortcomings of the pragmatics*

Vague-predicate resolution and sort coercion can be influenced strongly by the context built up during a discourse; TEAM cannot model these changing rules and so cannot resolve such cases. For example, in the query pair below

What planes are in trouble? (26)
Where are the TWA planes? (27)

a human reader will have no difficulty in determining that the planes referred to by the phrase "the TWA planes" in (27) are those TWA planes identified in the response to (26). However, TEAM would retrieve the locations of all TWA planes, because it does not have a model of dialogue context.

If the semantic interpretation of a word needs to be coerced, there are usually clues within the sentence to the effect that its "face value" interpretation is inappropriate; because TEAM uses only the clash of conflicting sort requirements within the sentence to test the interpretation, it will fail to

perform coercion when the clues originate in the broader context and not explicitly in the individual query. For example, in the GEO database the sentence "What is Everest?" would produce a query paraphrased as "What peak name is Everest?" which would show the peak name "Everest" as the answer. Since no other predicate (besides the one derived from *show*) is applied to *Everest*, there is no semantic conflict to tell TEAM that *Everest* should be coerced from a peak name to the peak with that name. This is a rarely seen problem because, in a more natural query like "How tall is Everest?" a conflict of sorts occurs (derived from asking the height of a peak name) inducing the necessary coercion.

3.5. Lexical ambiguity in TEAM

TEAM must handle a wide range of ambiguities that result from the union of its initial vocabulary with the vocabulary of its acquired domain. Because the total vocabulary is not built in, general mechanisms are included in the acquisition, parsing, and semantic code to support various forms of lexical ambiguity.

3.5.1. *Semantic versus syntactic lexical ambiguity*

The simplest form of lexical ambiguity arises when a word has two interpretations that are not the same syntactically—namely, when different parses are needed for the different choices. TEAM represents this case by assigning (during acquisition) to the lexical form a unique *wordalt* (for "word alternative") for each syntactically unique meaning. As an example from TEAM's initial domain-independent vocabulary, the word *who* has two wordalts: one as an interrogative pronoun ("who is he"?) and one as a relative pronoun ("the man who did it"). The morphological analyzer converts the input string into a chart in which the alternatives for a given word correspond to different arcs joining the same pair of nodes. This representation is general and could be used for all the forms of lexical ambiguity, but it can cause the generation of a potentially explosive number of parses; the number of superfluous parses approaches the product of all the lexical ambiguities in the sentence.

For ambiguous nouns that share the same syntactic properties, TEAM produces for the ambiguous word only a single terminal node (thus spawning no additional parses) in which an explicit set of semantic alternatives is spelled out. For example, in the geography database example, the word "Kenya" is triply ambiguous: it can be used to refer to a country learned from the peaks file, a country name from the countries file, and a peak name from the peaks file. The basic semantic functions use the first listed alternative as the semantics until something fails to work smoothly, then reach back to the node for additional possibilities until one that is compatible with the interpretation up to that point is found. Whenever a change is made in the semantic choice, previously established constraints are checked and any semantic alternative that is incompatible with them is rejected.

This approach reduces the parsing load to a minimum. However, it applies a strict level of acceptability rather than assessing the relative merits of every possible interpretation; if it finds an acceptable semantic alternative for each word, it does not try additional possibilities to see if any are better. Other approaches involving more sophisticated control schemes could seek a minimal cost interpretation for which cost depended on a wider range of factors, including such things as number of coercions required, kinds of coercions required, and rating of the parse. Because of the complexity of such approaches, they remain problems for future research in natural-language processing.

3.5.2. *Semantic alternatives versus multiple delineations*

For ambiguous verbs and adjectives, an additional alternative exists; there are predicate-producing words that are logically related to one another but must produce slightly different predicates in the database domain. As an example, consider a domain in which both aircraft and ships have a length property, but the property is represented differently for the two kinds of objects. The sentence "How *long* are the ships and planes in the Blue Army?" supplies the generic **LENGTH-OF** predicate, but, because it applies to a set composed of both kinds of objects, it cannot be differentiated into either **SHIP-LENGTH-OF** or **PLANE-LENGTH-OF** at the time of semantic translation.

TEAM handles such cases by constructing *multiple-delineation predicates* whose specifications allow different corresponding sorts for their fields. In our fleet example, the generic **LENGTH-OF** predicate would have two delineations, one calling for an airplane and an airplane length, and the other calling for a ship and a ship length as the role fillers. In the original design of TEAM, the database access generator determined whether to use one or the other specific predicate, or to use both for cases like the example of ships and planes. Unfortunately, the SODA query language does not currently support the disjunctive case. Since the language component produces such predicates, the database query generator is forced to use the sorts of the variables either to choose one of the possible interpretations or to produce multiple SODA queries, each addressing part of the problem.

Multiple-delineation predicates are created by acquisition at the point at which a verb or adjective is discovered to have a second, closely related meaning. The original, unambiguous interpretation is also replaced with the new multiple-delineation interpretation; it does not get treated specially just because it was encountered first.

3.5.3. *Pronouns and coreference resolution*

TEAM uses the mechanisms developed for semantically ambiguous nominals to handle a limited range of cases of coreference. A pronoun is treated as a proper noun that has a set of semantic alternatives; members of the set are

derived from the semantics associated with the various nodes that are syntactically acceptable as referents. The determination of a node's syntactic acceptability is done by Hobbs' algorithm [13]. Although this algorithm can be applied not only to the current sentence, but to previous ones as well, because of other limitations, TEAM does not currently support coreference outside the current sentence. Membership in the set is further constrained by requiring number and gender compatibility, and by applying rules about reflexive usage (the difference between "he cut himself shaving" and "he cut him shaving"). To support applying these constraints, gender markings for acquired vocabulary are explicitly learned during acquisition.

A general solution to the problems of coreference goes far beyond the scope of the current work. Although its importance in general is well-known, the role of discourse in determining coreference was not addressed in building TEAM. However, [9] contains a sketch of the need to consider discourse constraints seriously even in the restricted case of using natural language for database querying.

3.5.4. *Interaction with pragmatics*

Lexical ambiguity complicates the pragmatics processes significantly. Recall that these processes handle such tasks as replacing vague predications like (**OF** *EVEREST1 HEIGHT1*) with appropriately chosen specific ones like (**PEAK-HEIGHT-OF** *EVEREST1 HEIGHT1*). In this example, there are no relations that tie the initial interpretation of *EVEREST1* (a peak name) to *HEIGHT1* (a peak height). Coercion is applied to make *EVEREST1* become the variable representing "a peak whose name is Everest" rather than "the peak name Everest." The situation becomes even messier if a lexical item is semantically ambiguous; not only coercion, but alternative semantics and coercions of the alternatives, will be attempted to determine whether any relation might replace **OF** in the predication. TEAM's pragmatic processing tries all these possibilities before rejecting a parse. This is a case in which overly liberal acquisition combined with TEAM's limited capabilities in pragmatic processing can result in constructing a bizarre interpretation based on a parse that should have been rejected.

3.6. The schema translator

The schema translator maps a logical form that is independent of the database structure into a database query. To do so, it must solve the following major problems:

- map logical-form quantifiers to database query operators,
- determine how logical-form predicates and arguments map to database relations, fields, and values,
- determine what information to include in a reply to a query,

– remove redundant constraints from the logical form to produce more efficient queries.

The schema translator also deals with minor changes of representation and with peculiarities of the database query language.

As discussed before, the information described by a given English word or phrase can be represented in different ways in different databases. For example, the verb "to manage" (as in "Who manages Smith?") could be represented in one database by the value of a *manager* field of an *employee* relation, in another by the virtual relation that is defined by

$$x \text{ manages an employee } y \text{ if}$$
$$y \text{ is in department } z \text{ and } x \text{ is the manager of } z \tag{28}$$

The decoupling of logical form and database structure is essential for the modularity and transportability of TEAM. Nevertheless, a straightforward translation of logical form to database query will produce a highly redundant query. This redundancy stems from the fact that different predicates in the logical form for a given query may well correspond to accesses to one and the same database relation.

The schema translator can be seen as a rewrite system that takes the initial logical form and rewrites it into a simpler form that will produce the same answers as the initial logical form when evaluated against the database. This rewriting involves two kinds of processes:

(1) *Expansion*. A construct (connective, predicate) is replaced by its definition.

(2) *Simplification*. A subformula implied by the rest of the logical form is eliminated.

The schema translator operates by means of successive transformations of the logical form supplied by DIALOGIC. It is simplest to look at these transformations as operations upon logic formulas that preserve logical equivalence (or at least equivalence under the *closed-world assumption* [24]). In this section, we will use predicate notation for the database relations, although, both in acquisition and in the final relational queries, the actual notations used are those of relations and fields. For example, an occurrence of the virtual relation PKCONT will be written as (PKCONT n c). "Don't care" argument positions will be denoted by a question mark.

The intermediate forms of a query are represented in the schema translator by implications of the following form:

$$(\text{ANSWER } x_1 \ldots x_m) \leftarrow C \tag{29}$$

where the *condition* C is a conjunction of formulas

$$P_1 \wedge \cdots \wedge P_n \tag{30}$$

and each P_i is either an atomic *literal* $(P\, x_{i_1} \ldots x_{i_k})$ or the application of some operator to a condition of the same form. The possible operators are the negation NOT and *aggregation* operators like MAXIMUM, MINIMUM and COUNT. A query in the general form shown in (29) specifies the answer as the set of tuples $x_1, \ldots, x_m$ that satisfy the condition C.

The observant reader will notice that the general form in (29) is just that of a certain kind of PROLOG clause [5] in which goals may involve the metalevel operators for aggregation and for negation as failure. In fact, if the database were itself in PROLOG, the user query could be answered by executing the goal (ANSWER $x_1 \ldots x_m$) with respect to the PROLOG program comprising the database, the schema predicate definitions (discussed below) and the definitions of the metalevel operators. The query processing in the schema translator may thus be seen as a form of symbolic evaluation and optimization of the original query with respect to the schema predicate definitions.

3.6.1. *Quantifiers and database operators*

The component of the schema translator that transforms the incoming logical form into an implication of the form shown in (29) transforms certain logical-form quantifiers into their equivalents (e.g., it changes universals into double negations), selects the predicates that extract values being compared in comparative and superlative constructions, and converts superlatives into appropriate forms of the generic aggregation operator AGGR. Redundant equalities are removed. Finally, the EACH and WH logical-form quantifiers are processed to insure that the corresponding bound variables appear in the answer tuple. This is clearly required for the WH quantifier that binds the object of the query, but, as will be discussed in Section 3.6.3, it is also necessary for the wide-scope EACH quantifier.

The generic aggregation operator AGGR has the form

$$(\text{AGGR } op\ v\ x\ C\ r)\,,$$

which applies an aggregation operator *op* over the set of x, v pairs that satisfy condition C, producing result r. For example, "the largest country" will translate into the aggregation

$$(\text{AGGR MAX } a\ c\ (\textbf{COUNTRY}\ c) \wedge (\textbf{AREA-OF}\ c\ a)\ r)\,, \tag{31}$$

given the appropriate logical form and schema. Aggregation (31) holds when r is *the* country c whose area a is the largest among the areas of countries.

It should be noted that superlatives like "the largest country" (discussed in more detail in the next subsection) are represented in logical form not by *aggr*

conditions, but rather by special quantifiers, for example:

$$((\text{SUPER (MORE LARGE)})\ c\ (\text{COUNTRY}\ c)\ \ldots)\ ,$$

where (SUPER (MORE **LARGE**)) represents "the largest."

3.6.2. *Information sources and their use*

The schema translator uses two main kinds of information: *predicate definitions* and *constraints*. Predicate definitions specify how predicates appearing in logical form are to be mapped onto database relations. The constraints give the domain information that will be used in simplifying the logical form or choosing among alternative definitions. Stated informally, predicate definitions provide *active* information that tells the system how to query the database, whereas constraints provide *passive* information that determines the form of a correct and efficient query (cf. integrity and redundancy constraints in relational databases).

Both predicate definitions and constraints are expressed in the schema translator by logic formulas that correspond directly to well-known logical representations of database views and integrity constraints [6].

A predicate definition has the form

$$P \leftarrow S_1 \wedge \cdots \wedge S_k \wedge R_1 \wedge \cdots \wedge R_m\ , \tag{32}$$

in which the S_i are sort predications and the R_j are the predicates associated with real database relations. For example, the definition associated with the virtual relation PKCONT might be

$$\begin{aligned}&(\text{PKCONT}\ pk\ cont) \leftarrow \\ &\quad (PEAK\ pk) \wedge (CONTINENT\ cont) \wedge \\ &\quad (\text{PEAK}\ pk\ height\ country\ vol) \wedge \\ &\quad (\text{WORLDC}\ country\ area\ pop\ cap\ cont)\end{aligned}$$

Of the constraints used by the schema translator to simplify queries, the most important types express relationships among sorts and sortal constraints on arguments of logical-form predicates and on constants. Such constraints are represented here by implications of the forms

$$(S\ x) \leftarrow (S'\ x) \tag{33}$$

$$(S\ x) \leftarrow (P\ \ldots\ x\ \ldots) \tag{34}$$

$$(S\ c) \tag{35}$$

Constraints of type (33) state that S' is a subsort of S. Constraints of type (34)

state that the named argument of P is necessarily of sort S. Constraints of type (35) give the sorts of constants c appearing in queries—measures, for example. In practice, some of this information is not represented explicitly by logical formulas, but is rather extracted as needed from the sort hierarchy and other acquisition data structures.

Common nouns are normally translated into sorts; therefore, most variables in a logical form are restricted by a nontrivial sort. However, nonsort predicates in the logical form are also sources of implicit sort information, as any nonsort predicate has delineations that give the possible sort combinations for its arguments. For the purposes of the schema translator, delineation information is represented by implications of type (34), which are used in the schema translator to remove redundant noun-phrase predications.

Although proper nouns and other constants are not explicitly given sorts in English (in contrast with the entities introduced by common nouns), DIALOGIC needs to give them sorts so that it can choose among competing interpretations of other words in queries. Those sorts applied to constants appear in the logical form, but most can be removed by using constraints of type (35) (see below for a special case). Actually, facts of type (35) are not represented explicitly but rather are calculated as needed from information in the conceptual schema.

The sorts S_i in predicate definition (32), which are actually implied by the R_j, are used to select the right definition for a predicate with multiple definitions for arguments of different sorts (*multiple delineations*). This selection played a role in the interpretation of the predicate **NORTHERN** in (2) of Section 2. The appropriate definition for expanding a particular predication is selected by comparing the sorts of the arguments in the predication with the sorts of the predicate arguments given in each of its definitions. A definition is not suitable for expanding a predicate occurrence if any one of the sorts in the predicate occurrence is disjoint from the corresponding sort in the definition. Thus, the schema translator uses the disjointness information encoded in the sort hierarchy.

The final query simplification step uses *functional* constraints derived from acquired information. A functional constraint states that some field(s) of a relation is a function of other fields. In predicate terms, we have, for example, the constraint

$$c = c' \leftarrow (\text{PKCONT } n\ c) \wedge (\text{PKCONT } n'\ c') \wedge n = n' \tag{36}$$

In other words, peak continent is a function of peak name.

Functional constraints like (36) allow the schema translator to transform a query of the form

$$(\text{ANSWER} \ldots) \leftarrow \ldots (\text{PKCONT } n\ c_1) \wedge \cdots \wedge (\text{PKCONT } n\ c_2) \ldots$$

into the simpler query

$$(\text{ANSWER} \ldots) \leftarrow \ldots (\text{PKCONT } n\ c_1) \wedge c_1 = c_2 \ldots,$$

thereby eliminating one access to the PKCONT relation. We call this transformation *merging*.

It is clear from the above how functional constraints can be used to merge conjoined literals. However, it is possible to go further and, in certain special cases, merge pairs of literals in which one of the literals appears outside and the other one inside *conjunctive* aggregations; the special cases are those aggregations like MAX and MIN (and unlike COUNT), whose aggregation condition is true of the aggregation's result (e.g., the largest country is a country).

In practice, the schema translator does not use functional constraints in their full generality. Instead, it uses only the information on the key fields of relations obtained during acquisition; if a set of fields is the key of a relation, then (by definition) all other fields are functions of them.

Besides definitions and constraints, the schema translator requires some extra information before it can translate comparative and superlative constructions. The central element of the logical form of comparatives and superlatives is an *extremal form*

$$\Delta(\alpha) \tag{37}$$

in which Δ is an operator such as MORE or LESS and α is the translation of the uninflected adjective in the comparative or superlative (e.g., **HIGH** for "higher than"). Any adjective that can appear in an extremal form is linked (by the acquisition process) with a binary predicate that associates with the subject of a relation one of its numeric fields. In the sample database, the adjective *high* with translation **HIGH** is linked to the predicate **PEAK-HEIGHT-OF**. Acquisition also determines whether the adjective is *positive* like "high" or *negative* like "low." This metalevel information is recorded by two types of assertions:

$$\begin{array}{l}(\text{DEGREEPRED } \alpha\ P) \\ (\text{DIRECTION } \alpha\ \delta)\end{array} \tag{38}$$

In translating the comparative or superlative, *degreepred* is used to build a literal that extracts the numeric attribute used for comparison, maximization, or minimization; the direction indicator δ from *direction* is combined with the operator Δ from the extremal form (37) to determine the direction of comparison of numerical values (i.e., greater or less than).

3.6.3. *Informative answers*

In a query such as "What is the area of each country?" a strict interpretation of

the logical form would lead to the simplified query

$$(\text{ANSWER } a) \leftarrow (\text{COUNTRY } ? \; a \; ? \; ? \; ?) \tag{39}$$

resulting in a database query that lists just the areas of countries, without showing which country covers which area. A more appropriate answer would be a table of country-area pairs, as provided by the query

$$(\text{ANSWER } c \; a) \leftarrow (\text{COUNTRY } c \; a \; ? \; ? \; ?) \tag{40}$$

We say in this case that the country name argument c is related to the strict answer argument a. More generally, the related arguments of an answer are those on whose values the answer depends.

It would be possible to determine exactly the related arguments of an answer by examining the functional properties of the database relations (encoded in the specification of key fields). However, this would in many cases result in an overly verbose answer, because not all argument positions are mentioned explicitly in the natural-language query.

We opted instead for the simpler heuristic of taking as related arguments those variables explicitly quantified in the logical form whose quantifiers outscope the quantifier for the answer argument. Of course, variables that appear outside a negation are never related to arguments inside it.

3.6.4. *Operation*

We will now describe the main steps of the schema translator, which operates in almost sequential fashion. The effect of each step will be exemplified by showing the resulting intermediate and final query representations of query (2) from Section 2, "What northern countries contain peaks higher than Fuji?" Recall that this query is translated by earlier steps of TEAM into the logical form of Fig. 18.

First, the logical form of the query is put into clause form and the variables related to the answer are identified, resulting in the implication in Fig. 19. The answer variable c of sort *COUNTRY* is preceded in the answer literal by the related variables. Besides the variable p that corresponds to the noun phrase "peaks," we have f for the quantification associated with the constant FUJI (an artifact of the logical form that will be removed later), s for the height of Fuji, and h for the height of the peak p. The variables s and h do not come from explicit quantifications in the English query, but it is often convenient to show these implicit comparison variables to justify to the user the selection made by the comparison. Note also the transformation of the term ((MORE12 **HIGH**) p s) into the conjunction ((**PEAK-HEIGHT-OF** p h) $\wedge$ ($> h$ s)). The extremal operator MORE12 indicates a comparison in the increasing direction between the value of the attribute of p (indicated by the predicate **HIGH** and accessed by predicate **PEAK-HEIGHT-OF**) and a height value s.

In the next step all nonsort literals are expanded as specified in their predicate definitions. The expanded predicates are **PEAK-NAME-OF**, **PEAK-HEIGHT-OF**, **NORTHERN**, and **CONTAIN**. As the relation PEAK has a single key field, peak and peak name can be identified by replacing variable *f* with the constant FUJI and replacing the literal (**PEAK-NAME-OF** *f* FUJI) with (PEAK FUJI ? ? ?). More interestingly, the predicate **NORTHERN** for countries (objects of sort *COUNTRY*) is defined by a join between relations WORLDC and CONT; another definition of **NORTHERN**, for continents, will be used only for objects of sort *CONTINENT*. The resulting query (Fig. 20) contains only database relations and sort literals.

This first expansion step is followed by a simplification step in which sort

```
(QUERY
  (WH c
      (AND (COUNTRY c)
           (NORTHERN c))
      (SOME f
           (AND (PEAK f)
                (PEAK-NAME-OF f FUJI))
           (SOME h
                (PEAK-HEIGHT h)
                (SOME p)
                     (PEAK p)
                     (AND (PEAK-HEIGHT-OF f h)
                          ((MORE12 (HIGH)) p h)
                          (CONTAIN c p)))))))
```

FIG. 18. Logical form.

```
(ANSWER f s p h c) ←
  (COUNTRY c) ∧ (NORTHERN c) ∧
  (PEAK f) ∧ (PEAK-NAME-OF f FUJI) ∧
  (PEAK-HEIGHT s) ∧
  (PEAK p) ∧
  (PEAK-HEIGHT-OF f s) ∧
  (PEAK-HEIGHT-OF p h) ∧
  (> h s) ∧
  (CONTAIN c p)
```

FIG. 19. Query in clause form.

(ANSWER FUJI s p h c) ←
  (*COUNTRY* c) ∧
  (CONT n ? N ?) ∧
  (WORLDC c ? ? ? n) ∧
  (*PEAK* FUJI) ∧
  (PEAK FUJI ? ? ?) ∧
  (*PEAK-HEIGHT* s) ∧
  (*PEAK* p) ∧
  (PEAK FUJI s ? ?) ∧
  (PEAK p h ? ?) ∧ ($>$ h s) ∧
  (PEAK p ? c ?)

FIG. 20. Expanded query.

constraints are used to remove all sort literals implied by other literals. If there were any sort literals left (which is not the case in this example) it would be because they contribute constraints to the query. This occurs in particular with the sorts derived from feature fields. The result for our example is shown in Fig. 21. At this point, another expansion step would replace the remaining sort literals with their definitions. Since there are none in our example, the query is unchanged.

(ANSWER FUJI s p h c) ←
  (CONT n ? N ?) ∧
  (WORLDC c ? ? ? n) ∧
  (PEAK FUJI ? ? ?) ∧
  (PEAK FUJI s ? ?) ∧
  (PEAK p h ? ?) ∧ ($>$ h s) ∧
  (PEAK p ? c ?)

FIG. 21. Simplified query (no postexpansion needed).

At this point, the query contains only actual database relations. However, because of the decoupling of logical form and database predicates, it contains many redundant accesses to the database relations. The merging transformation discussed earlier is now invoked to remove those redundant accesses, resulting in the query of Fig. 22. The body of this query is as simple as it could be, given the initial English query and the structure of the database.

The final step is to transform the implication form query into a query in the target database query language, SODA. Answer variables identified either with

```
(ANSWER FUJI s p h c) ←
  (CONT n ? N ?) ∧
  (WORLDC c ? ? ? n) ∧
  (PEAK FUJI s ? ?) ∧
  ( > h s) ∧
  (PEAK p h c ?)
```

FIG. 22. Merged query.

constants or with other answer variables are also removed at this point to eliminate redundant information from the answer vector.

In SODA, as in more familiar relational query languages, such as SQL, equalities between relation fields play the same role as shared variables in logic, while tuple variables play the same role as literals. The final transformation is essentially a compilation from clause form to SODA that takes into account those notational differences as well as the scope rules for tuple variables and aggregations in SODA. The result for our example is shown in Fig. 23.

Of all the steps in the schema translator, only the last depends on the particular database query language used and would have to be changed for a database system with a different query language. If the database query language were PROLOG, the last step would not be needed at all.

```
((IN n CONT)
 (n CONT-HEMI EQ N)
 (IN c WORLDC)
 ((c WORLDC-CONTINENT) EQ (n CONT-NAME))
 (IN f PEAK)
 ((f PEAK-NAME) EQ FUJI)
 (IN p PEAK)
 ((p PEAK-HEIGHT) GT (f PEAK-HEIGHT))
 ((p PEAK-COUNTRY) EQ (c WORLDC-NAME))
 (? (p PEAK-NAME))
 (? (p PEAK-HEIGHT))
 (? (c WORLDC-NAME))
 (? (f PEAK-HEIGHT)))
```

FIG. 23. SODA query.

4. Other Approaches to Transportability

Interest in developing natural-language interfaces for a wide range of databases has spurred a number of attempts to create transportable systems. TEAM itself grew out of an earlier attempt [16] to make LADDER [11] more easily adaptable to new domains by providing a system designer with functions making it relatively easy to create new entries in the lexicon, conceptual schema, and database schema. Figure 24 summarizes the properties of six systems that have taken approaches to transportability that differ in a number of ways from TEAM's. The six systems are: ASK [30], EUFID [28, 29], Ginsparg's system [8], IRUS [4], CHAT-80 [35] and LDC-1 [2, 3].

We will compare these systems with TEAM and with each other from several standpoints.[14] Because of the diverse goals of these research efforts, it is important to restate the obvious: any single set of criteria can serve as a basis for comparison only with respect to particular issues, and should not be taken

	TEAM	Ginsparg	IRUS	CHAT-80	ASK	LDC-1	EUFID
TYPES OF TRANSPORT-ABILITY	linguistic domain, db, DBMS	linguistic domain, db, DBMS(rel'l)	linguistic domain, db, DBMS	linguistic domain, db	linguistic domain, db	linguistic domain, db	linguistic domain, db, DBMS
EXPERTISE OF TRANSPORTER	database expert	system designer	system designer	system designer	range: user, superuser, system designer	superuser	system designer + domain expert
SYSTEM DESIGN	general NLP (+ acquisition)	general NLP	general NLP	general NLP	db-specific NLP (+ acquisition)	semantic grammar-db NLP (+ acquisition)	semantic grammer specific to db
INFORMATION ACQUIRED	lexical, conceptual, db schema	lexical, semantic network, db schema	lexical, domain semantics, db schema	lexical, logical form to db predicate mappings	? (lexical, conceptual, db)	lexical, conceptual, conceptual to db mappings	lexical, semantic graph, database
CONTROL OF ACQUISITION	system	system designer	system designer	system designer	superuser	superuser	transport-ability staff
RANGE OF NL EXPRESSIONS	general [verbs, quantifiers, limited anaphora]	general [verbs, quantifiers, limited anaphora]	general [verbs, quantifiers, limited anaphora]	general [verbs, quantifiers, no anaphora]	general-database-query	restricted	grammar-dependent
TIME TO ADAPT	minutes - hours	hours - day	weeks	days - weeks	?	minutes - hours	months

FIG. 24. Summary of comparison of transportable systems.

[14] Discussions with the builders of the other systems were helpful in refining these criteria as well as in correcting the presentations of each system.

as a means of ranking systems above or below one another globally. The criteria of comparison we will use are the following:

(1) The kinds of transportability that are accommodated: to other linguistic domains, to other specific databases, to other database management systems.[15]

(2) The kinds of expertise required for the person doing the transporting: expertise in the particular system (*system designer*), natural-language processing (*computational linguist*), databases (*database expert*), experience or training with the particular system (*superuser*), ordinary user (*end user*).

(3) Basic system design. How general is the natural-language processing system? Is it intended for handling natural language generally (*general NLP*), the subset of NL that arises in database querying (*db-specific NLP*), or the subset of NL specific to a particular database (*specific to db*)? We explicitly note if a system uses a semantic grammar because, as discussed previously, this affects transportability. In addition, we note those systems that include automatic acquisition capabilities of some sort.

(4) The kinds of information that must be supplied to adapt the system to a new database and domain. This differs with the particular system design, more details in regard to which are given below.

(5) Control of acquisition. Does the user (of whatever type) or system determine when the minimum necessary information has been supplied?

(6) Range of natural-language expressions handled. For example, how general a capability for adding new verbs is provided? Are quantifiers handled adequately? What kinds of anaphoric reference are supported?

(7) How long does it take to adapt the system to a new database?

In the following discussion, when considering system design, we will also note whether the particular system (1) separates the representation of the meaning of the query from the retrieval statement in the database query language, and (2) separates domain- and database-specific information from domain- and database-independent information.

With regard to the range of natural-language expressions handled, we will focus our attention on three important constructions: verbs, because they are one of the prime components of natural language and, to a great extent, are what distinguishes fluent natural-language querying from querying in formal database query languages; quantifiers, because they are essential to database query; and anaphoric expressions, because they give some indication as to whether the system is making any attempt to handle extended discourse.

We have not included runtime for processing a query as one of the points of comparison, because the systems run on different types of computer systems

[15] We did not consider transportability to other types of computer systems, but looked only at transportability for interfacing to a database. Some systems (e.g., IRUS, the DIALOGIC component of TEAM) have been developed with this more broad type of transportability in mind, but this opens many questions beyond the scope of this paper (cf. [23]).

making such comparisons difficult and of dubious value (in fact, doing this was resisted quite strongly by the other systems' designers as well). TEAM typically takes between one and two seconds on a Symbolics 3600 series machine to translate an English query into the corresponding SODA query; the time to produce an answer depends, of course, on the size of the database. Further details on TEAM are in the preceding sections of the present paper. In the remainder of this section we would like to elaborate on the other systems in sufficient detail to convey their basic premises and techniques, as well as to emphasize their differentiating characteristics.

Ginsparg's system. The core of Ginsparg's system is a general natural-language-processing system that includes an extensive semantic network relating various concepts to one another. Lexical and network information are used to derive a formal representation of the meaning of a query; this representation is also in the form of a semantic network. A database application program maps the network representation into a query in an augmented relational algebra.

The system handles a wide range of natural-language expressions, including noun-noun constructions and some other "pragmatic" constructions. The system also includes extensive capabilities for handling quantifiers and conjunction. One of its interesting features is a general mechanism to compute coercions on the basis of preferences for case fillers and comparison of path-lengths in the semantic network.

Ginsparg's system was designed to be moved to new linguistic domains, new file structures, and new relational DBMSs. The mapping to a new DBMS is done by writing a translator program between the relational-algebra query language and the target DBMS language. The system includes certain tools that aid in acquiring the information needed for moving to a new domain or database, but the adaptation presupposes sufficient knowledge of how the system works that a system designer is required to do it. In particular, it is assumed that the transporter understands the target application program and is familiar with the layout of the existing semantic network.

The core system contains an elaborate semantic network for concepts related to a range of common English words, so that much of the language acquisition is concerned with providing links between the existing vocabulary and database-specific concepts. This involves supplying implicit case information about file attributes and creating virtual relations to support attributes as different synonyms in different contexts. Connection to a database proceeds by first connecting nodes of the semantic net to the relations in the database schema, then connecting the relations to the actual database.

The user determines when enough has been acquired (another factor contributing to the need for a system designer to do the transporting). This system has been moved to several "toy" and one real database (the Bell Labs

Company Directory). It requires only a few hours to move to a similar domain, or a day or so for a first attempt at adapting the system to a domain further removed from its existing network [7].

IRUS. Like TEAM and Ginsparg's system, IRUS (for Information Retrieval using the RUS parsing system) is based on a general natural-language-processing system. It translates an English query into a higher-order predicate calculus representation of its meaning, which is then mapped to a language reflecting the database structure, and then to an appropriate DBMS query. IRUS has been interfaced to System 1022 and to the IDM 5000 database machine; interfaces to two other DBMSs are currently being developed.

IRUS includes an extensive grammar of English, including a wide range of verb types (e.g., IRUS, like TEAM and Ginsparg's system, can handle sentential complements). It covers a broad range of natural-language determiners, including some not common in predicate calculus. It deals with a good portion of the quantifier-scoping problem by restricting the quantification possibilities for variables associated with database fields to those that would make sense for generating database values.

IRUS is modularized for adaptation to different domains, databases, and DBMSs. Currently the adaption requires a system designer, although (as with Ginsparg's system) IRUS includes several tools for aiding in acquisition; these tools may well evolve to the point that a superuser will be able to do the adaptation. Responsibility for deciding when enough information has been supplied rests with the user (i.e., the system designer at present).

To adapt IRUS to a new domain requires adding lexical information (the system has closed-category words already), providing the semantics for domain concepts, and creating file structure tables for the particular DBMS. Staff weeks are required for moving IRUS to a new database if the domain is one to which IRUS has previously been adapted and the DBMS is one that is already supported. More time is required if a new DBMS is involved or a more radical shift in domain is entailed. However, some of the vocabulary can be carried over between applications; the connection between the semantics and the database can be changed without any need to reacquire the complete lexical entry.

CHAT-80. CHAT-80 also includes a general natural-language-processing system, but differs from the previously described interfaces in its reliance on PROLOG. CHAT has a separate syntactic grammar, semantic interpretation, scoping and query planning components. These components are domain-independent and use domain information in tables represented by logic clauses. The grammar has extensive treatment of gaps and movement, some forms of topicalization, comparatives, superlatives, and aggregation words (e.g., "average" and "proportion"). The verb phrase grammar is restricted to noun phrase, prepositional

phrase, and predicative complement. Coordination is allowed only between predicative noun or verb complements. Semantic interpretation uses the sorts of predicate arguments to select acceptable complement attachments. Scoping generates the most closely scoped reading compatible with scoping rules and heuristics; differences among determiners such as "each," "any," and "every" are handled reasonably. The result of the scoping component is a logical form that is further simplified and optimized for PROLOG execution by query planning.

CHAT can be transported to new linguistic domains and, to the extent that they can be mapped onto PROLOG concepts, to new file structures. Adaptation requires supplying PROLOG clauses that give dictionary entries and specify mappings between logical-form predicates and database predicates. All the current applications of CHAT use the internal PROLOG database; adaptation to any other database system would require an interface between it and PROLOG. The current version of CHAT has been transported by the original designers as well as by competent PROLOG users with some understanding of the system's design concepts. It is the user who decides whether an acquisition is complete.

The time to transport CHAT has ranged from a few days to a few weeks, depending on the complexity of the new application; the longest adaptation times include substantial new PROLOG code to provide new functionality, such as graphical presentation of answers.

The ASK system. ASK handles a broad range of linguistic forms within the task of database querying. It also can treat a limited range of intersentential anaphora. Because little information is available on the internal structures and processes of ASK, it is difficult to determine exactly what the system design is, or what information must be supplied to adapt ASK to a new domain or database. ASK is based on a previous system, REL [32], which used a case grammar as the basis for parsing.

ASK, like REL, in transforming an English question into a database query, does not use an intermediate representation of the query's meaning. This design decision was made for efficiency reasons, but one of its effects is to make it necessary for a superuser or system designer to make certain kinds of adaptations. This decision also makes it difficult to assess the general capabilities of the natural-language processing system; because the meaning of a query is taken to be identical to the code for retrieving an answer, the system's coverage of natural language is restricted to database querying.

ASK uses an entity/attribute database model; to adapt it to a new domain the user has to specify explicitly the individuals, classes, attributes (single-valued) and relations for the domain. Some of these may be explicit in the database structure, but others may not be. Adapting to a new database can be done in two ways: using the ASK internal database structure, or hooking directly to a "foreign" database. The first is done by providing a text file with all the

database information and using one of ASK's acquisition dialogues, the "bulk data input dialogue," to provide information about the contents of the database so that ASK can appropriately configure its internal database. This dialogue seems to assume a combination of knowledge about natural-language processing concepts and database concepts (including details of the text file format). The second method for connecting to a new database, directly connecting to that database, involves providing ASK with details of how information is stored in the database. ASK includes an acquisition dialogue, the "foreign access dialogue," that aids in providing this information. However, to provide this kind of access, one needs to know the details of the implementation of the foreign database.

There are special stylized English-like inputs for adding new definitions (e.g., to specify that child-of is, roughly, the inverse of parent-of) as well as new individuals, classes, attributes, and relations. The addition of verbs (the core system includes "be" and "have") also is done by using one of these stylized patterns [31]. Typical verb definitions depend on prior definition of all the appropriate individuals, classes, relations and attributes. ASK acquires new verbs as specialized instances of a general verb form. It can handle passives, datives, and prepositional arguments, but does not provide for verbs with features such as unaccusative as opposed to object-omitted constructions,[16] prohibited passives,[17] and sentential complements.

ASK also includes some capabilities for natural-language access to additional kinds of software; text messages and graphics are two examples often cited. Adapting ASK to such systems requires a system designer however, and is beyond the scope of concepts dealt with in this paper.

In summary, end users can easily add new word definitions and new data (using the stylized input format for addition definitions), but adapting to a new domain and database seems to require at least a superuser or someone with combined database and natural-language-processing expertise. No data are available on how much time is required to transport ASK.

LDC-1. LDC-1 is a transportable system that serves to interface with a database consisting of text files rather than with the more structured form of database considered in the other systems described here. These files do not consist of natural-language sentences expressing information, but are rather just a textual form of database. We are including it because it does offer reasonable linguistic coverage and some degree of transportability. LDC-1 provides for transport-

[16] For example, it cannot distinguish verbs like "grow" which does not allow for object-omission (i.e., "A farmer grows" cannot be used to mean a farmer grows something, although "Wheat grows" can be used to mean that wheat is grown by someone) from those like "cook" which do (i.e., "A farmer cooks" can be used to mean a farmer cooks something, and "A turkey cooks" to mean a turkey is cooked).

[17] Certain senses of some verbs cannot be passivized; e.g., "The turkey weighs twelve pounds" cannot be passivized to "Twelve pounds are weighed by the turkey."

ability to new linguistic domains and new databases; the actual database is internal to LDC and is constructed from raw text files.

The LDC parser produces a parse that is structured by semantic cases; the parse is converted directly into a high-level retrieval query. English modifiers in the original query are transformed into parameterized query procedures, which are then executed within the framework of the overall query.

LDC includes a phrase structure grammar that gives fairly broad NL coverage, including complex noun and verb phrases and limited coverage of noun-noun constructions and genitives, some discontinuous constituents (e.g., split "than" complements). LDC does not currently provide a general treatment of gapping, pronouns, or quantifier scoping.

To transport LDC to a new domain and database, the following be supplied: dictionary and compatibility files, including English words, their major categories, (verb, noun, adjective), their morphological properties, and selectional restrictions; entries in a data dictionary, including the names of each type of entity and the nature of the relations among them; macrodefinitions files that specify the relations between conceptual entities and the physical entities present in the raw lexical file, as well as providing a mapping from each English modifier to a corresponding retrieval expression. A preprocessor module provides assistance in supplying this information; it also supports restructuring of the raw data file for greater utility.

Adapting LDC requires a superuser (i.e., an experienced user who knows about the various system structures) for everything but entering data into the text file database. The superuser ascertains the completeness of each part of the acquisition. For certain portions, there is a preprocessor module that asks questions to elicit complete information. In addition, whenever the English processor encounters words whose lexical or semantic properties are not known, a run time customization module is invoked.

The form of the text file database can be changed quite fast; complete customization for a new domain has been reported as typically requiring from 30 minutes to four hours [1].

EUFID. EUFID's natural-language processing is based on a semantic grammar. A new one is written for each application; syntactic information is used only when needed to "resolve semantic ambiguities." Domain-independent linguistic information consists solely of the closed-set vocabulary (the "function words") in the dictionary. Semantic categories in the grammar, corresponding essentially to fields in the database, are derived from information about key fields and their joins.

EUFID processes queries in three steps. First, an analyzer produces a "parse" tree (the nodes of which include semantic categories). Next, a translator produces an intermediate-language (IL) representation of the meaning of the query; the IL is oriented to database structures. Finally, the IL representation is translated into the actual DBMS query language. The translator program is

written anew for each new DBMS language. (This has been done for both QUEL and WWDMS.)

EUFID uses a dictionary that includes both function words that are common across applications, and content words that pertain specifically to an application. The content words are linked into a large "semantic graph" in which the nodes' case-structure roles form the connecting links. Also associated with each content word are one or more mapping functions that specify how the nodes of a parse tree are to be transformed into IL; these mapping functions govern the translation from parse tree to IL.

The information for mapping from the concepts in the semantic graph to the IL consists of two tables, both of which must be supplied for each new domain and database: one has the database-specific information for each field of each relation (e.g., field name, formats, conversion functions) while the other lists each database "group" (similar to, but not identical with, a sort), each group-to-field mapping, and the logical joins needed to form each possible group-to-group connection.

The range of natural language that can be handled varies because a different grammar is constructed for each new domain/database. Some kinds of conjunction and simple negation are handled; quantifiers are not treated generally.

EUFID was designed to be transportable to new linguistic domains, new file structures, and new DBMSs. It has been adapted to three domains and two DBMSs. Adaptation requires both a EUFID expert and a domain expert. These experts decide when sufficient information has been given to EUFID. To facilitate their decision, a test set of about 300 sentences is devised before the transport attempt and, as it proceeds, used as a gauge of progress.

Accounts of experiences with EUFID [29] report that in adaptations to three new domains, each successive effort was faster than its predecessors, but that a typical transport still takes "several staff-months."

5. Conclusions

Previous sections of the paper have discussed the basic mechanisms used in TEAM to provide transportability in a natural-language interface system and have examined several specific research problems that had to be dealt with as TEAM was being developed.

Although the TEAM system has not undergone the kind of testing an actual product is normally subjected to, it has been used by a fair number of people in various organizations. Sufficient experience has been gained to suggest that the TEAM experiment has had essentially positive results. It does appear possible to build a natural-language interface that is general enough to allow adaptation of the system to a new database by a user (DBE) who is familiar with the database itself, but not an expert in natural-language processing and does not know the details of the system itself.

However, this accomplishment must be placed in the proper perspective. TEAM cannot exceed the limitations of natural-language-processing technology. It shares such constraints of customized interfaces as being restricted to single queries and being able only to retrieve the facts from a database, not to reason about them.

TEAM handles a wide range of verbs, a capability that is absolutely essential for fluent natural-language communication. Its limitations in treating determiners of all sorts are evident in those cases where discourse and pragmatic constraints are central to obtaining the correct interpretation. The system handles limited uses of pronouns, but here too the lack of discourse knowledge is a critical limiting factor.

Work on TEAM might be extended in a number of ways—some practical, others research-oriented. To construct a useful application system, the techniques developed for TEAM must be integrated with other features that are important to natural-language communication (e.g., techniques for generating cooperative responses [14]) and to human-computer interactions generally (e.g., error correction). In addition, since a grammatical formalism for encoding syntactic information in TEAM was decided upon, several more perspicuous ones have been developed and analyzed (e.g., [27]); however, grammars with coverage as extensive as TEAM's have yet to be developed in such formalisms. We are now investigating ways of providing transportability in natural-language systems that enable communication with other kinds of software and that include more discourse capabilities than TEAM and other transportable systems currently possess.

Appendix A. TEAM Questions and Information Extracted

This appendix lists all of the questions that are asked of the database expert by the TEAM acquisition component, and indicates what information is extracted from each one. Answers to questions potentially affect the lexicon, the conceptual schema, and the database schema, as well as determining what additional questions are presented to the DBE.

A.1. Relation objects

File name:

Gets name of relation. Used by both conceptual and database schemata.

Relation's status in database—VIRTUAL, ACTUAL:

Determines what questions need to be asked.

Database pathname:

Gets internal file name of database for use by LISP machine file system.

What is this relation about? ENTITIES RELATIONSHIPS:
  Determines what additional questions need to be asked.
  Conceptual schema: Determines whether OF-predicates will be created for fields.
Subject (*asked only for* ENTITIES *relations*):
  Lexicon: Adds noun for subject.
  Conceptual schema: Adds subject sort.
Fields:
  Lexicon: Adds noun for each field name.
  Conceptual schema: Adds sort predicate for each field sort.
    Adds OF-predicate for each field if ENTITIES relation.
Primary key set:
  Database schema: Provides information needed to specify attachment of OF-predicates, field sort predicates and subject sort to the database.
Identifying fields:
  Database schema: Provides information needed to answer queries properly.
Pronouns for file subject—HE, SHE, IT, THEY:
  Lexicon: Provides plural and gender information for subject noun.

A.2. Field objects

Type of field—SYMBOLIC, ARITHMETIC, FEATURE:
  Determines what questions need to be asked.

Symbolic fields

Edit lexicon for words in this field—YES, NO:
  If yes, when database is read, TEAM will have access to all entries so the DBE can specify irregular plurals, synonyms, etc.
Are field values units of measure:
  If answered yes, it wants to know what arithmetic field this field units for.
Database nouns subcategory—PROPER, COUNT, ABSTRACT, MASS, UNIT:
  Lexicon: Information required to create lexical entries for words in database.
  Conceptual Schema: Information required to associate the proper semantic predicate with the words in database.
Typical value:
  Used to phrase help information for the following two questions.
Will the values in this field be used as classifiers of the file subject? YES, NO:
  Conceptual schema: Pragmatic information required to support "How many Ford cars . . . ?" type of questions.
Will the values in this field be used alone as implicit classifiers? YES, NO:
  Conceptual schema: Pragmatic information required to support coercion. Used in "How many Fords . . . . ?" type of questions.

Arithmetic fields

Value type—DATES, MEASURES, COUNTS:
  Determines what questions need to be asked.
Date format—MM/DD/YY, DD/MM/YY, YY/MM/DD, MONTH, DAY, JULIAN:
  Database schema: Determines conversion factor to canonical time representation for date fields.
Are the units implicit—YES NO:
  Companion question to "Are field values units of measure?" question.
Enter implicit unit:
  Lexicon: Adds noun for unit of measure.
  Conceptual schema: Adds predicate for measure unit.
Measure type of this unit:
  Conceptual schema: Determines where to place field sort predicate in the sort hierarchy.
Abbreviation for this unit:
  Lexicon: Enters noun for abbreviation.
Conversion formula for ⟨canonical⟩ to ⟨new-unit⟩:
  Database schema: Provides conversion factor for printing the answer to query.
Conversion formula for ⟨new-unit⟩ to ⟨canonical⟩:
  Database schema: Provides conversion factor for constructing query.
Type of object counted (*asked for* COUNT *fields*):
  Lexicon: Enters noun for type of object.
  Conceptual schema: Enters sort predicate for type of object.
Positive adjectives:
  Lexicon: Enters adjectives.
  Conceptual schema: Enters adjective predicate, related field, and direction for scale information.
Negative adjectives:
  Lexicon: Enters adjectives.
  Conceptual schema: Enters adjective predicate, related field, and direction on scale information.

Feature fields

Positive value:
  Lexicon: Enters noun for value.
  Conceptual schema: Enters semantic predicate for value.
    Creates subsort of field subject sort.
  Database schema: Specifies database attachment of subsort.
Negative value:
  Lexicon: Enters noun for value.
  Conceptual schema: Enters semantic predicate for value.
    Creates subsort of field subject sort.

Database schema: Specifies database attachment of subsort.
Positive adjectivals:
Lexicon: Makes adjective entry.
Conceptual schema: Enters predicate for adjective.
Database schema: Specifies attachment of adjective predicate to database
Negative adjectivals:
Lexicon: Makes adjective entry.
Conceptual schema: Enters predicate for adjective.
Database schema: Specifies attachment of adjective predicate to database
Positive abstract nouns:
Lexicon: Makes noun entry.
Conceptual schema: Creates sort for noun, places in sort hierarchy.
Database schema: Attaches sort predicate to the database.
Negative abstract nouns:
Lexicon: Makes noun entry.
Conceptual schema: Creates sort for noun, places in sort hierarchy.
Database schema: Attaches sort predicate to the database.
Positive count nouns:
Lexicon: Makes noun entry with appropriate semantics.
Negative count nouns:
Lexicon: Makes noun entry with appropriate semantics.

A.3. Word objects

Enter word:
Gets the name of the word.
Synonym:
Specifies synonym for newly added word.
Lexicon: Copies all syntactic and semantic properties from synonym.
Syntactic category—NOUN, ADJECTIVE, VERB:
Lexicon: Determines major syntactic category.
Noun subcategory—PROPER, COMMON, ABSTRACT, MASS:
Lexicon: Determines subcategory information for nouns.
Plural:
Lexicon: Makes entry for irregular plurals.
Comparative:
Lexicon: Makes entry for irregular comparative form of adjective.
Superlative:
Lexicon: Makes entry for irregular superlative form of adjective.
Third person singular present tense:
Lexicon: Makes entry for irregular third person singular of verbs.
Past tense:
Lexicon: Makes entry for irregular ED form.

Past participle:
  Lexicon: Makes entry for irregular EN form.
Sentence:
  Determines what questions get asked next.
  Conceptual schema: Creates verb predicate and determines the number of arguments and the sort of each one.
A subj Vs P obj ⇔ A subj Vs it P;
  Lexicon: Is preposition a case marker or a particle?

For transitive verbs: A subj Vs an obj + PPs.

An obj Vs ⇔ Something Vs an obj:
  Lexicon: Can object appear as subject when deep subj omitted?
A subj Vs ⇔ A subj Vs something:
  Lexicon: Is the object optional?
An obj is Ven ⇔ Something Vs an obj:
  Lexicon: Can sentence containing verb be passivized?

For transitive verbs: A subj Vs an obj1 obj2 + PPs.

A subj Vs and obj1 obj2 ⇔ A subj Vs an obj2 {to | for} an obj1:
  Lexicon: Determines preposition for dative.
A subj Vs and obj2 ⇔ A subj Vs someone an obj2:
  Lexicon: Determines if object is optional.
A subj Vs ⇔ A subj Vs someone something:
  Lexicon: Determines if both beneficiary and object are optional.
An obj2 is Ven ⇔ Something Vs someone an obj2:
  Lexicon: Determines if verb can be passivized.

ACKNOWLEDGMENT

Jerry R. Hobbs and Jane J. Robinson made major contributions to the DIALOGIC system. They, Robert C. Moore, and Daniel Sagalowicz played important roles in the design of TEAM. Armar Archbold, Norman Haas, Gary Hendrix, Lorna Shinkle, Mark Stickel, and David Warren also contributed to the project. We have profited from extensive comments made by Madeleine Bates, Jerry Hobbs, C. Ray Perrault, Martha Pollack, Jane Robinson, Daniel Sagalowicz and Ralph Weischedel on earlier drafts of this article.

We also wish to thank Bruce Bollard (LDC-1), Madeleine Bates and Robert Bobrow (IRUS), Jerrold Ginsparg, Marjorie Templeton (EUFID), and Fred Thompson (ASK) for their help in modifying the criteria of comparison and in providing accurate descriptions of their systems' capabilities. We are especially grateful for their supplying information not available (yet) in published papers, but crucial to the comparison.

The development of TEAM was supported by DARPA under contracts N00039-80-C-0645 and N00039-83-C-0109, monitored by the Naval Electronic Systems Command. Development of DIALOGIC received additional support from DARPA under contract N00039-80-C-0575, from the National Library of Medicine under NIH Grant LM03611 and from the National Science Foundation under Grant IST8209346.

REFERENCES

1. Ballard, B., Personal communication, 1985.
2. Ballard, B., The syntax and semantics of user-defined modifiers in a transportable natural-language processor, in: *Proceedings COLING*, Stanford, CA (1984) 52–56.
3. Ballard, B. and Tinkham, N., A grammatical framework for transportable natural-language processing, *Comput. Linguistics* **10** (2) (1984) 81–96.
4. Bates, M. and Bobrow, R., A transportable natural language interface, in: *Proceedings Sixth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Bethesda, MD, 1983.
5. Clocksin, W.F. and Mellish, C.S., *Programming in Prolog* (Springer, Berlin, 1981).
6. Gallaire, H., Minker, J. and Nicolas, J., Logic and databases: a deductive approach, *ACM Comput. Surv.* **16** (2)(1984) 153–185.
7. Ginsparg, J., Personal communication, February 1985.
8. Ginsparg, J., A robust portable natural language data base interface, in: *Conference on Applied Natural Language Processing*, Santa Monica, CA (1983) 25–30.
9. Grosz, B.J., Transportable natural-language interfaces: Problems and techniques, in: *Proceedings of the 20th Annual Meeting of the ACL*, Toronto, Ont. (1982) 46–50.
10. Grosz, B.J., Haas, N., Hendrix, G.G., Hobbs, J., Martin, P., Moore, R., Robinson, J. and Rosenschein, S., Dialogic: A core natural-language processing system, in: *Proceedings of Ninth International Conference on Computational Linguistics*, Prague, Czechoslovakia, 1982.
11. Hendrix, G.G., Human engineering for applied natural language processing, in: *Proceedings IJCAI-77*, Cambridge, MA (1977) 183–191.
12. Hendrix, G.G., Semantic aspects of translation, in: D.E. Walker (Ed.), *Understanding Spoken Language* (Elsevier, New York, 1978).
13. Hobbs, J.R., Pronoun resolution, Research Rept. 76-1, Department of Computer Sciences, City College, City University of New York, New York, 1976.
14. Kaplan, S.J., Cooperative responses from a portable natural language database query system, in: M. Brady and R. Berwick (Eds.), *Computational Models of Discourse* (MIT Press, Cambridge, MA, 1983).
15. Knuth, D.E., *The Art of Computer Programming, Fundamental Algorithms* **1** (Addison-Wesley, Reading, MA, 1973).
16. Konolige, K., A framework for a portable natural language interface to large data bases, Tech. Note 197, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1979.
17. Levin, B.C., English verb complementation patterns, Unpublished manuscript, 1979.
18. Martin, P., Appelt, D. and Pereira, F., Transportability and generality in a natural-language interface system, in: A. Bundy (Ed.), *Proceedings IJCAI-83*, Karlsruhe, F.R.G. (1983) 573–581.
19. Moore, R.C., Handling complex queries in a distributed database, Tech. Note 170, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1979.
20. Moore, R.C., Problems in logical form, in: *Proceedings 19th Annual Meeting of the Association for Computational Linguistics*, Stanford, CA, 1981.
21. Paxton, W., A best-first parser, in: *Proceedings IEEE Speech Symposium*, Pittsburgh, PA, 1974.
22. Pereira, F.C.N., Logic for natural language analysis, Ph.D. Thesis, University of Edinburgh, U.K., 1982; Reprinted as: Tech. Note 275, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1983.
23. Perrault, C.R. and Grosz, B.J., Natural-language interfaces, *Annu. Rev. Comput. Sci.* **1** (1986) 47–82.
24. Reiter, R., On closed world data bases, in: H. Gallaire and J. Minker (Eds.), *Logic and Databases* (Plenum Press, New York, 1978).
25. Robinson, A., Appelt, D.E., Grosz, B., Hendrix, G. and Robinson, J., Interpreting natural-

language utterances in dialogs about tasks, Tech. Note 210, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1980.
26. Robinson, J.J., Diagram: A grammar for dialogues, *Commun. ACM* **25** (1) (1982) 27–47.
27. Shieber, S.M., The design of a computer language for linguistic information, in: *Proceedings of COLING*, Stanford, CA (1984) 362–366.
28. Templeton, M., Practical natural language processing, in: M. Rubinof and M.C. Yovits (Eds.), *Advances in Computers* **13** (Academic Press, New York, 1975).
29. Templeton, M., and Burger, J., Problems in natural language interface to DMBS with examples from EUFID, in: *Conference on Applied Natural Language Processing*, Santa Monica, CA (1983) 3–16.
30. Thompson, B. and Thompson, F., Introducing ASK, a simple knowledgeable system, in: *Conference on Applied Natural Language Processing*, Santa Monica, CA (1983) 17–24.
31. Thompson, B. and Thompson, F., Rapidly extensible natural language, in: *Annual ACM Conference*, Washington, DC, 1978.
32. Thompson, F.B. and Thompson, B.H., Practical natural language processing: The REL system as prototype, in: *Advances in Computers* **13** (Academic Press, New York, 1975).
33. VanLehn, K.A., Determining the scope of English quantifiers, Master's Thesis, MIT Rept. AI-TR-483, Cambridge, MA, 1978.
34. Waltz, D., Natural-language access to a large data base: An engineering approach, in: *Proceedings IJCAI-75*, Tblisi, U.S.S.R. (1975) 868–872.
35. Warren, D.H.D. and Pereira, F.C.N., An efficient easily adaptable system for interpreting natural language queries, *Am. J. Comput. Linguistics* **8** (3–4) (1982) 110–122.
36. Woods, W.A., Cascaded ATN grammars, *Am. J. Comput. Linguistics* **6** (1) (1980) 1–12.
37. Woods, W.A., Semantics and quantification in natural language question answering, in: M.C. Yovits (Ed.), *Advances in Computers* **17** (Academic Press, New York, 1978).
38. Woods, W.A., Kaplan, R.M. and Nash-Webber, B., The lunar sciences natural language information system: Final report, Rept. 3438, Bolt Beranek and Newman, Cambridge, MA, 1972.

Received June 1985; revised version received September 1986