J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering*

NATURAL LANGUAGE UNDERSTANDING

Introduction

While reading this sentence, remarkably you are shaping an understanding of what I mean it to say, and you will probably succeed. Your achievement and success in understanding is most impressive. The speaker's task is much simpler—to generate an utterance that conveys a presumably preexisting thought. Your task as listener is to decide what the speaker must have been thinking in order to motivate his utterance in the particular context in which he uttered it. In general, understanding a natural language is simply miraculous.

In this article, we explain some of the major problems inherent in automating natural language understanding by computer, from the need to understand the domain of discourse for both understanding and answering questions to the need to engage in user modeling for correct interpretation and generation of natural language. We examine several uses to which automated natural language understanding has been put. Designing a general *question-answering system* may be the most difficult. The casual use of language with which humans engage—filling in missing information, making plausible and other subtle inferences, disambiguating equivocal language, etc.—is difficult, and a general question answerer must match this feat of human language interpretation and appropriate language generation.

By contrast, a natural language interface (*NLI*), say, to a relational database, is simpler to comprehend. Although one ultimately encounters similar problems to the unconstrained natural language understanding problem, the domain of discourse, and thereby the context, is highly constrained to the "world" posited by the database schema. General analysis of language phenomena and much of the ambiguity inherent in natural language understanding is limited.

One initial motive for understanding language by computer was *machine translation*, that is, the use of computers to translate one natural language into another. The long history of machine translation has yielded no fundamental breakthroughs, but progress has been impressive nonetheless, and use of machine translation systems is growing.

More recently, and with the growing popularity and use of the Internet, natural language processing techniques have been used for information retrieval and text categorization. In these approaches, reliance on knowledge bases and grammars is replaced by large corpora, often Internet-based, from which requisite knowledge is extracted. We will only categorize them briefly in this article, since current efforts are yielding results almost daily.

The rest of this article will be organized as follows. In the next section we enunciate a number of concerns that will have to be taken into account to process and understand natural language by computer. We illustrate these concerns using numerous examples and show how important it is to incorporate knowledge of the domain, knowledge of the user, and knowledge of discourse structure into an automated system.

In the section after, we discuss the need for machine translation systems, and what such systems amount to. We do so by examining past and current machine translation efforts and explaining the major categories of machine translation systems.

We then describe the general approach to constructing sophisticated *NLIs* to (relational) databases. A major practical issue of importance is the desirability of making, *LIs portable*. We address many portability aspects. There are a number of ways in which a *NLI* can be made portable, the most important of which is to promote easy attachment to databases representing different domains, that is, portability between applications. A closely related aspect is portability within different versions of the same database; the *NLI* must adapt easily when the content and structure of the database change. This type of portability has received little attention to date. Another important portability aspect is the ability to connect the *NLI* to different types of database management systems. Equally important is the ability to transport the *NLI* interface to different types of computers. Finally, the ease with which a *NLI* can be ported from one language to another (e.g., from English to French) can also be considered important. Language portability bears an obvious connection with machine translation.

A subsequent section reviews some material of more historical interest, describing the early goals of question-answering systems and their approaches and shortcomings. This review is followed by a exposition of some current efforts to utilize natural language techniques in less demanding enterprises such as information retrieval and text summarization. Included in this exposition is a brief introduction of empirical, corpus-based approaches to natural language processing.

Throughout the exposition of this article we will use numerous examples and provide some detail of one natural language interface system, SystemX, built in the late 1980s and used by a large Canadian cable company to provide better customer service. SystemX utilized many ideas from earlier systems but also embodied several modern theories, one of which is an evolving theory of parsing.

We conclude the article with a summary of ideas from natural language understanding by computer literature and provide an extensive set of references.

Understanding Natural Language

Contemporary methods for automated natural language processing depend, to a large extent, on the use to which each application of the processing is to be put. Consider the following excerpt from Erle Stanley Gardner's *The Case of the Demure Defendant* (1):

> "Cross-examine," Hamilton Burger snapped at Perry Mason.
>
> Mason said, "Mr. Dayton, when you described your occupation you gave it as that of a police expert technician. Is that correct?"
>
> "Yes sir."
>
> "What is an expert technician?"
>
> "Well, I have studied extensively on certain fields of science that are frequently called upon in the science of criminology."
>
> "That is what you meant by an expert technician?"
>
> "Yes sir."
>
> "Now what is a police expert technician?"
>
> "Well that means that . . . well, it all means the same thing."
>
> "What means the same thing?"
>
> "An expert technician."
>
> "An expert technician is the same as a police expert technician?"
>
> "Well I am in the employ of the police department."

"Oh the police employ you as an expert witness, do they?"
"Yes sir, . . . I mean no, sir. I am an expert investigator, not an expert witness."
"You are testifying now as an expert witness are you not?"
"Yes sir."
"Then what did you mean by saying you were an expert technician but not an expert witness?"
"I am employed as a technician but not as a witness."
"You draw a monthly salary?"
"Yes."
"And you are being paid for your time while you are on the stand as an expert witness?"
"Well, I'm paid for being a technician."
"Then you won't accept any pay for being a witness?"
"I can't divide my salary."
"So you are being paid?"
"Of course—as part of my employment."
"And are you now employed by the police?"
"Yes."
"And are you an expert witness?"
"Yes."
"Then you are now being employed as an expert witness."
"I guess so. Have it your own way."
"When you described yourself as a police expert technician that means your testimony is always called by the police. Isn't that so?"
"No, sir."
"Who else calls you?"
"Well, I . . . I could be called by either party."
"How many times have you been on the witness stand?"
"Oh, I don't know. I couldn't begin to tell you."
"Dozens of times?"
"Yes."
"Hundreds of times?"
"Probably."
"Have you ever been called by the defense as a defense witness?"
"I have not been directly subpoenaed by the defense. No, sir."
"So that you have always testified for the police, for the prosecution?"
"Yes, sir. That's my business."
"That was what I was trying to bring out," Mason said . . ..

Imagine, if you will, the processing required to emulate the part of Mr. Dayton. Mr. Dayton needs to understand the subtleties of noun phrases such as "police expert technician," in order to answer Mr. Mason's questions. Understanding such phrases is particularly troublesome to automate, since "police," "expert," and "technician" are all normally nouns. Generalizing semantic considerations for such constructions has proven evasive to date. The compositional approach to natural language understanding favored by the logic

grammarians becomes combinatorially explosive, and many researchers have opted to represent noun–noun constructions as single lexical entries instead, thus constraining the computation required to disambiguate such constructions, and circumventing an annoying semantics problem. When the domain of discourse is well specified and the number of such phrases is small, this approach works adequately. But is this covert procedure generally practical? Consider the noun–noun phrase "western region outage log" employed by telecommunications service personnel. Would the designer of a system designed to automatically process language of this nature resort to separate lexical entries for "eastern region outage log," "southern region outage log," "northern region outage log," "northeastern region outage log," . . ., "western district outage log," . . ., "western prefecture outage log," . . ., "western region service log," . . ., "western region outage record," . . .?

Imagine further the processing required by our automated language understanding system to emulate the part of Perry Mason. Not only must the subtleties of language understanding realized by Mr. Dayton be mastered, but also the reasoning capabilities of Mr. Mason and the extraction of relevant and salient features of the conversation must be identified in order to generate the appropriate next question. Actually, Mr. Mason's task is much simpler than Mr. Dayton's—to generate an utterance that conveys a presumably preexisting thought. Mr. Dayton's task as listener is to decide what Mr. Mason must have been thinking in order to motivate his utterance in the particular context in which he uttered it.

Speculating only on this abbreviated introduction, it would appear that the automation of ordinary natural language capabilities using computers is a formidable task indeed. To provide some focus to this discussion, let us assume that you are trying to obtain academic advice from an automated system. Many existing systems explore dialog and representation issues that arise in this domain (2,3,4,5). Thus considerable background exists to draw upon in the academic domain.

Complexities arise in building natural language capabilities, consider the sentence:

$$[2-1] \quad \text{I want the marks in CS110.}$$

The interpretation of this query would seem straightforward. The sentence structure is simple: almost any parser would easily be able to recognize the subject "I," the main verb "want," and the object "marks," modified by the prepositional phrase "in CS110." It would not be difficult to map the word "marks" onto the corresponding database attribute, and to require that only marks in the course Computing Science 110 be accessed. A list of these marks could easily be produced and returned to the user. Most natural language front ends could handle this without undue difficulty.

However, there are many assumptions underlying this interpretation that may, in fact, not correspond to what the user intended. The first problem which must be resolved is how to even recognize [2-1] as a question. Its surface form is declarative, not interrogative. Presumably recognizing the underlying *request speech act* is not too difficult for current systems, since almost any input of this nature is likely to be a request. We are interested, however, in the next generation of natural language systems, where it is possible to imagine the user informing the system of his/her desires ("I want to change all marks of 49% to 50%," "I want to add something to what I just told you," etc.). Moreover, other phraseologies are more problematic (for example, "Can you give me the marks in CS110?" where a yes–no answer is inappropriate). The development of computational models that satisfactorily explain how to recognize the underlying user intention despite the surface form of the natural language has begun (6).

Recognizing the underlying speech act(s) is just one item which must be inferred about the user. The system has to have some idea of what the user wants to do with the marks in order to figure out what exactly to present to him/her. For example, it is unlikely that anybody wants *just* the marks; presumably, names and student numbers should accompany them. Moreover, if the user is the registrar, who wants to mail the marks out to the students, it would be a good idea to include addresses with the names. If the user is the university statistician, it may be sufficient to give the distribution of the marks, rather than the list of students and their

marks. Certain types of users may not be even be allowed access to the marks (for example, another CS 110 student).

There are many other unspecified aspects of [2-1] that must be clarified if an appropriate response is to be given. Does the user want the marks for all sections of CS 110, or just one section? If just one section, might it be possible to infer which section without asking the user (for example, Prof. Smith would want the marks of his own section)? Does the user want the marks for last year's CS 110 course? Next year's? The course currently underway? If s/he wants the marks of the course currently in progress, should the system hold the request in abeyance and respond at the end of the term when the course is done? How about holding the request if the marks are for next year's course? Once the course and the section are determined, the system still has to figure out whether all marks are required, or just final examination marks, or the final grade for each student. The system must also decide what to do about the request if it is somehow inappropriate. For example, CS 110 may not have been offered, or there may not be any course named CS 110, or the marks may not have been assigned as yet.

One possible route to sorting out ambiguities like these is to force the user to use a formal language where the ambiguities do not occur, although it is still possible for some types of ambiguity to arise—the *multiple access path problem* (7). Learning and remembering the formal language is awkward if the database is to be accessed by computer-naive users. Another possible route is to engage the user in an (extended) *clarification dialog* as suggested early on by Codd et al. (8). The problem with this approach is that the user is forced to spend inordinate amounts of effort simply to make a simple query understandable to an inflexible computer system. This type of engagement is unlikely to impress naive computer users, and would militate against the usefulness of the natural language system. The only alternative left is to make the natural language system behave much more like a human, who would use knowledge of the user and of the subject to accurately infer answers to most of the questions raised above (see Ref. 9 for an in depth discussion).

Domain Knowledge Is Reqred to Interpret Queries. There are many places where it is necessary to use domain knowledge in order to interpret the queries. The place where the need for domain knowledge is most evident is in *resolving references* made in the input query. A particular reference problem that arises in *NLIs* to databases is the problem of anaphoric references, especially pronoun references. For example, the following sequence of sentences could easily arise in the academic advice domain:

[2 – 2] Could I have a report on the students who are registered in Computer Science?

[2 – 3] Would you summarize *it* for me?

[2 – 4] Which students dropped out?

[2 – 5] Which of *them* are still mentioned in *the report*?

A natural language system would have to resolve the pronouns *it* in [2-3] as referring to the concept represented by the *report* in [2-2], and *them* in [2-5] as referring to the *students* in [2-4]. Note that the reference is actually to the answer produced by the system to the user's queries, rather than to the concept in the query itself. A nonpronominal anaphoric reference which must be resolved is that *report* in [2-5] refers to the report produced in response to [2-2]. Many other kinds of anaphoric references are possible, some of them much more subtle than these, such as the so-called *forward references* (see Ref. 10 for examples).

Another place where the need for domain knowledge is evident is in discovering and *resolving ambiguities* which arise in the interpretation of a query. *Ellipsis* frequently occurs during database access, especially given the tediousness of typing questions rather than verbalizing them. Ellipsis can often be handled by syntactic mechanisms. *Tracking the focus of attention* of the user as he/she asks a series of questions is sometimes

important in natural language database interfaces, as the following sequence of queries illustrates:

[2 – 6] Who had and honors average last year?
[2 – 7] Which of them had the highest average?
[2 – 8] Was there anybody else close?

To answer query [2-6], the system must search through the set of all students. In sentence [2-7] the focus is narrowed to the set of students who had an honors average last year. Query [2-8] also assumes a focus set of students with an honors average last year. Note how differently [2-8] would be interpreted if it were asked after [2-6] without the intervention of [2-7] to change the focus set.

Domain Knowledge Is Required to Answer Queries. Domain knowledge is also required to answer queries. One category of queries that requires domain knowledge beyond data consists of questions asking about things that reflect *general rules* that apply in the domain, rather than things that happen to be accidentally true of the current data. Even with such general rules, sometimes it is not possible to respond definitively "yes" or "no." Knowledge of any realistically large domain (the kind that might really benefit by being automated) will not be complete. Facts will be missing, errors will occur in the data, some rules simply will not be known, and some topics not covered.

There are several kinds of ignorance a system can have. One kind arises when the system has not got all of the facts. Consider the following example:

[2 – 9] Have any students ever failed CS 882?

The system may be able to look back and see that no students have failed CS 882 according to its records, but it might also realize that its records only go back three years. Thus, in accordance with the principle of giving the user as much information about its limits as possible, the system should respond with something like "Not according to my records, but they only go back three years" rather than with "No" or "I don't know." It is important that the system be able to encode knowledge about what it knows and does not know.

A related issue is to keep track of exceptions to general rules. For example, to answer the query

[2 – 10] Do all professors in the Computer Science department have a Ph.D. degree?

the system may find a general rule $(\forall x)(\text{professor}(x) \Rightarrow \text{has-PhD}(x))$, which would allow it to respond "Yes." Such universal rules are fine in theory, but in knowledge bases representing real world domains, there are bound to be exceptions. Some of these exceptions will represent special cases; others may represent errors in the data. In either case, the system must discover these exceptions and produce an extensive response based on them; for example, "All professors do except Professor Jones." It may even be useful to try to explain why the exception exists in order to help the user understand the subtleties of the general rule. Thus, an answer like "All professors do except Professor Jones. She was hired before the requirement that all professors have Ph.D. degrees came into effect" might be more appropriate. A system good at producing such explanations may help the user distinguish special case exceptions from exceptions that are errors in data.

Another place where a knowledge base may be needed rather than simply a database in order to answer a query successfully is in the area of *summary response generation*. Instead of a long enumerative response to the query "Which courses may a computer science student take in addition to the courses in his or her major?" (for example, "Classics 100, Classics 101, Biology 110, . . ."), a summarized version of the response (for example, "Other courses in Arts and Science") would be more appropriate. It is fairly easy to imagine how this might be done without recourse to anything beyond a relational database. In this case the database could be searched for

all courses taken by computer science students, and then the faculty that offers any noncomputer courses read. If the entry for the faculty attribute for each such course is "Arts and Science," then the summary response can be given.

Other considerations requiring domain knowledge to answer queries include hidden *time dependencies* (for example, "Is Professor Jones available for advising?" may require a monitored response such as "No; shall I tell you when he is?"; answers related to *hypothetical queries* (for example, "If I gave Mark an 85% what would the class average be?"); references to *system-generated concepts* (for example, the concept "members of the graduate faculty" may be a system-generated concept, subsequently referenced, which answers the query "Who is capable of supervising graduate students?") practical considerations regarding *updates*; requests concerning the *database structure* (for example, "Who is teaching which courses this fall?"); and the *generation of extended responses*.

Modeling the User Is Important as Well. The need for domain knowledge in order to interpret queries and generate responses is unquestionable. In fact, many of the queries used as examples thus far demand that the system be able to infer the intentions of the user in order to interpret what the user is asking for, and hence to produce an appropriate response. Domain knowledge is not enough; we also need to be able to *model the user*.

This point is fairly obvious. For example, the database community has recognized for some time the need for different user views corresponding to different kinds of users. These user views allow the database management system to restrict the kinds of access available to a given class of users. Similar use should be made of user models. Moreover, a user model would be helpful in disambiguating the user's utterances in several ways: it would restrict the number of possible interpretations of the user's query; it would generally cut down on the combinatorics involved in accessing the knowledge base; and it would make it far easier to decide how to phrase the response for a class of users (for example, managers may need summary responses, clerks extensional information, etc.).

Grice 11 elaborates four basic tenets of the cooperative principle which a speaker should obey: (i) *the maxim of quantity*: be as informative as required, but no more so; (ii) the *maxim of quality*: do not make a contribution that one believes to be false or for which adequate evidence is lacking; (iii) the *maxim of relation*: be relevant; and (iv) the *maxim of maer*: avoid obscurity of expression, avoid ambiguity, be brief. Grice formulated these principles for human discourse. Much of the recent work investigating natural language dialog and in particular natural language question answering has elaborated firm computational underpinnings to Grice's maxims. Directly or indirectly Grice's maxims are of central importance to current research in natural language understanding (5, 6).

One of the most obvious places the need for user modeling arises is in recognizing *presuppositions* the user has implied, and correcting any false presuppositions if necessary. Kaplan's CO-OP system (12) was among the first to demonstrate that such presupposition correction was a serious necessity even in the supposedly constrained world of practical natural language understanding. To illustrate, consider the query

[2 – 11] How many undergraduate passed Cs 859 last year?

There is a presupposition in [2-11] that undergraduates are allowed to take CS 859. Clearly, the user believes this presupposition, or s/he wouldn't have asked the question in this fashion. An answer of "None," therefore, does nothing to correct this presupposition in the mind of the user in case it is false. The system, therefore, violates the maxim of quality outlined above. A much better answer would be "Undergraduates are not allowed to take Cmpt. 859."

The standard procedure for handling presuppositions divides into two steps: first it is necessary to find the presuppositions, and then it is necessary to recognize any that are false so that they can be corrected. It is important in correcting the false presuppositions that other false presuppositions be not set up (for example, if CS 859 were not offered, the answer above would still contain a false presupposition). Thus, an answer-

generation–presupposition-correction cycle must be carried out before producing the response. Alternative methods whereby the presuppositions of the answer, rather than those of the question, are computed would allow the system to behave more reliably when confronted with presuppositions that it cannot prove true or false in the database.

False-presupposition correction needs domain knowledge in order to determine if a presupposition is consistent or not with the current data. Developing automatic procedures that generate concept hierarchies to help respond appropriately to null database responses resulting from false user presuppositions would be useful (13). For example, depending on which part of the query "What grade did John receive in Math 101?" resulted in the null response (grade not yet posted, Math 101 not a course or not offered, John not a student, etc.), an appropriate response is generated ("Grades have not yet been posted. Would you like me to inform you when they are?" etc.).

Recognizing the implication of multiple user goals, keeping track of user's knowledge, answering "why" questions, and making scalar implicatures all demand some degree of user modeling and extended database knowledge. For example, "Is Smedley a student?" is a question that could straightforwardly be answered "Yes," but more informatively "Yes. A graduate student." This extended response is an example of a *scalar implicature*, an inference (not necessarily based on logic) that can be made based on some scale. In this case, "graduate student" is a subtype of "student" and thus is a higher value on a scale ("higher," rather than "lower," because the truth of the subtype semantically entails the truth of the type)The scalar implicature is to augment the answer with a reference to the higher value.

Thus the need for knowledge to access information is clear. We need knowledge of the domain to both interpret queries and to answer queries, and we must allow for user knowledge in this process as well. Investigations of the problems we mention are taking place and solutions are being incrementally integrated in successive prototype natural language database systems. For example, a problem that has incorporated results is the generation of appropriate responses to null database answers resulting from false user presuppositions (mentioned above) (14, 15).

Without resorting to a sophisticated and wholly integrated knowledge-based system (which will be necessary in the longer term), systems have been constructed to access information in ordinary language that are better than the limited number of earlier products that were built [for example, Hendrix et al.'s *NLI* (16); English Wizard, based on the earlier INTELLECT system (17); SystemX (18), based in part on aspects of TQA (19)] or under development [for example, NLAISE (20)]. An appendix to this article provides a list of some commercially available natural language systems, and a critique of English Wizard.

In the next section, we consider the problem of translating from one natural language into another, surveying the major approaches taken and commenting on each approach. We discuss briefly the emerging new paradigms for machine translation and provide numerous examples throughout our discussion.

Machine Translation

Why Machine Translation?. Exchanges between countries in trade, technologies, politics, telecommunications, and so on continue to grow very rapidly. Language plays a significant role in communication between nations, and in order to understand what is communicated, a person needs to understand the language used. It is at this point that machine translation becomes paramount: to translate the communication from one language into another language. The need for translation is increasing as exchanges become more globalized. However, the need for translation from one language into another does not exist only internationally, but also within a country that uses more than one language, e.g., Canada (English–French), the United States (English–Spanish).

In countries where multiple official languages are used, translation is particularly necessary to enable people to express themselves in the way that they wish and obtain the type of information they desire. Being

able to express oneself in one's own language and to receive information that directly affects an individual in the same medium is very important, because the loss of a language often involves the disappearance of a distinctive culture, a way of thinking; that loss should matter to everyone.

It seems that successful machine translation (*MT*) becomes a social and political necessity for modern societies that do not wish to impose a common language on their members. With the increasing benefits of *MT*, further research on *MT* will prove to be useful and valuable for any community.

What is Machine Translation? *MT* is one of the practical applications of natural language understanding research. *MT* is defined in several ways: (1) systems that perform a syntactic analysis of a source text and then generate a target language rendering thereof, with the intent of preserving and reconstituting its semantic and stylistic elements, are described as *Machine translation systems* (21); (2) *MT* is the name for computerized systems responsible for the production of translations from one natural language into another, with or without human assistance (22); and (3) *MT* is the transfer of meaning from one natural (human) language to another with the aid of a computer (23).

However, all definitions lead to the same conclusion, which is that *MT* attempts to translate from one language to another (or others in multilingual system) with a computer. The term *MT* is used to refer to a fully automatic machine translation system. When a computer collaborates with a human in some fashion during the translation process, the system is referred to as *machine-aided translation* (*MAT*) or *computer-aided translation* (*CAT*). *MAT* can be classified into two categories: (1) *machine-aided human translation* (*MAHT*), in which the human performs the translation with assistance from a machine, e.g., consulting on-line or automatic dictionaries or terminological databases, using multilingual word processors; and (2) *human-aided machine translation* (*HAMT*), in which the machine performs the translation and consults a human user at many stages to complete the process, e.g., asking a user to make a choice in case of lexical ambiguity.

History of Machine Translation. The details of history of *MT* can be found in many books (22, 24, 25). A brief chronology follows:

Petr Smirnov-Troyanskii (a Russian) envisaged the three stages of mechanical translation: First, analyze input words in the source language (*SL*), to get their base forms and syntactic functions. Next, transform sequences of those base forms and syntactic functions into equivalent sequences in the target language (*TL*). Then convert this output into the normal form in the TL.

A word-by-word French-to-Spanish program was implemented by Booth and Ritchens as reported in (26).

Warren Weaver (Rockefeller Foundation) and Andrew D. Booth (a British crystallographer) discussed the possibility of using computers for translation. This was the first discussion of *MT*. Weaver brought the idea of *MT* to general notice and suggested some techniques for developing *MT* systems, e.g., wartime cryptography techniques, statistic analysis, Shannon's information theory, and exploration of the underlying logic and universal features of language. *MT* research began at a number of research institutions in the United States.

Yehoshua Bar-Hillel (MIT) convened the first *MT* conference, outlining the future research in *MT*.

Georgetown University's Leon Dostert collaborated with IBM on an *MT* project, demonstrating an *MT* system that translated from Russian into English. It is regarded as the first generation of *MT*. The journal *Machine Translation* was founded.

MT projects were initiated elsewhere in the world. Many approaches were adopted in developing *MT* systems, for example: the empirical trial-and-error approach, the statistics-based approach with an immediate working system as the goal, brute-force techniques, the direct approach, and early versions of the interlingual and the transfer approachs.

MT progress was first reviewed by Bar-Hillel, with his conclusions as follows:

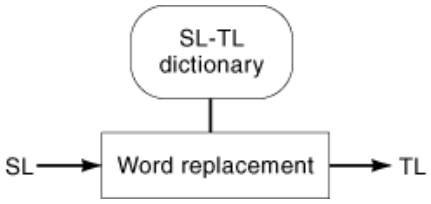

Fig. 1. Direct MT.

- The fully automatic, high quality translation was not really attainable in the near future so that a less ambitious goal is definitely indicated.
- The empirical approach which stressed statistical and distributional analyses of texts seemed somewhat wasteful in practice and not sufficiently justified in theory.

Erwin Reifler pointed out some obstacles to *MT* having to do with the interaction between syntax and semantic

The government sponsors of *MT* in the United States formed the Automatic Language Processing Advisory Committee (*ALPAC*) to provide directed technical assistance as well as contribute independent observation in *MT* (27). *MT* research was initiated in Japan.

An *ALPAC* report concluded that *MT* was slower and less accurate than human translation, and twice as expensive. The report also stated that "there is no immediate of predictable prospect of useful *MT*." This report brought a virtual end to *MT* research in the United States for over a decade and damaged the public perception of *MT* for many years afterward. *MT* research was initiated in Canada and Western Europe.

MT research projects revived in the United States.

The beginnings of AI-oriented research on *MT* (27).

The Commission of the European Communities (*CEC*) purchased the Systran system for development of translating systems for languages of the community (29).

The first multilingual *MT*, Eurotra Project, was developed. This system aimed at translating back and forth between nine languages: Danish, Dutch, English, French, German, Greek, Italian, Portuguese, and Spanish. The first public *MT* system Météo, developed by university of Montreal, for translating weather reports from English to French was used. It was one of the most successful *MT* systems.

The first commercial system, Automated Language Processing System (*ALPS*), was developed.

A number of *MT* research projects took place in Asia. Speech translation began in Great Britain and Japan. A number of commercial *MT* systems began.

Machine Translation Methodologies. The *MT* methodologies used for developing *MT* systems can be classified into three different categories: the three classic strategies, the nonlinguistic information strategies, and the hybrid strategies.

Three Classic Strategies. The first of the three classic strategies (Fig. 1), *Direct MT*, is a word-to-word replacement approach, replacing a SL word with a TL word. The accuracy of the translation is based entirely thbilingual dictionary. The following examples, the translations of Russian (SL) into English (TL) and the correct translation (*CT*), show the poor translation accuracy of this approach. (These examples were taken from Ref. 22.)

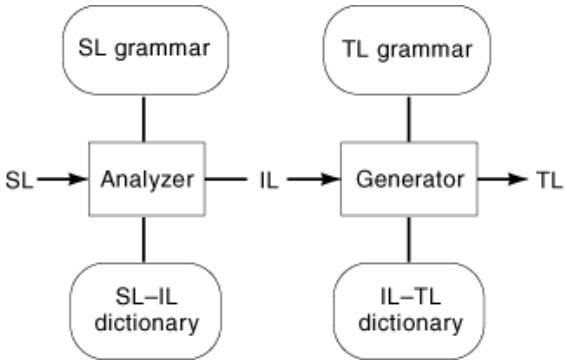

Fig. 2. Interlingual MT.

Example:

My trebuem mira.
We require world.
We want peace.

Example:

On dopisal stranitsu i otlozˇil rucˇku v storonu.
It wrote a page and put off a knob to the side.
He finished writing the page and laid his pen aside.

This approach was adopted by most *MT* systems in the first generation of *MT*, for example, the Georgetown system of Garvin in 1967 (23). The translation accuracy of this approach is limited because the linguistic analysis of the SL is inadequate. However, it is still convincing in certain circumstances. In addition, traces of the direct approach are found in indirect approaches such as those of Météo (22), Systran (22), and GRMT (31).

To generate more accurate translations, the second approach, *interlingual MT* (Fig. 2) was proposed. The treatment of linguistic information became more sophisticated in this approach. The interlingual approach uses an *interlingua* (*IL*) as an intermediate representation between SL and TL, analyzing SL and mapping it to a language-independent representation form, then generating the TL from this *IL* without looking back at SL. Evidently, the accuracy of this approach is based on the *IL*. A number of interlingual systems were developed for example, ATLAS of Fujitsu (32), PIVOT of NEC (33), Rosetta of Phillips (34), KANT (35), and CICC (36).

Since the analysis and generation modules are totally independent of each other, the interlingual approach is attractive for multilingual lingual *MT* systems. The all-ways translation between n languages requires n analysis and n generation modules, whereas the direct approach requires n $(n-1)$ translation modules. The other advantage of interlingual approach is its extendibility.

The interlingua must capture all necessary information of the analyzed SL so that the generator can produce an accurate translation result. However, defining neutral concepts for different languages is rather a chimera. Some of difficulties in defining neutral concepts are as follows:

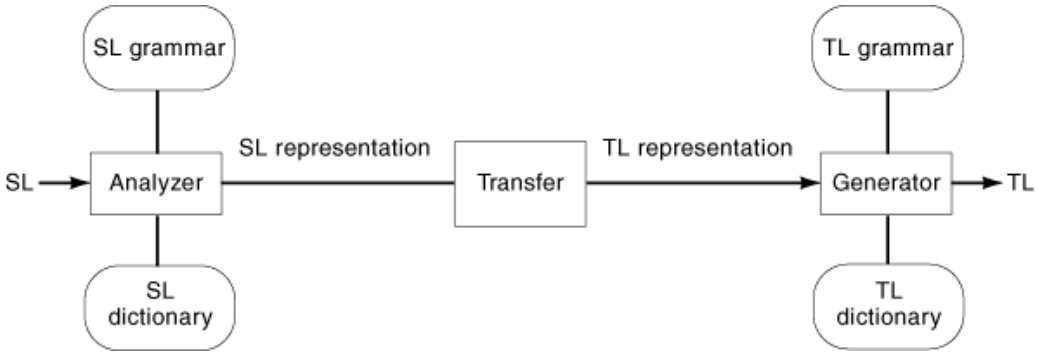

Fig. 3. Transfer MT.

- The scopes and concept classification of each language are different for example, the concept of culture, the concept of the supernatural world, and the concept of unit.
- A single concept is one language can be mapped into many concepts in another language.
- More than one concept can be mapped into one concept in another language.
- Some concepts do not exist in some languages.

Because of these difficulties, a compromise between the direct approach and the interlingual approach led to the third approach, *transfer MT* (Fig. 3). The transfer approach involves *transferring* the linguistic information of the SL to the TL; analyzing the SL, representing it with a machine-readable form, and mapping this form to a machine-readable form of TL; and then generating the TL. The accuracy of this approach depends mostly on the transfer module.

Unlike the intermediate representation in the interlingual system, the intermediate representation in the transfer approach strongly depends on the language pair. Therefore, the all-ways translation of n languages requires n analysis, n generation, and $n(n-1)$ transfer modules. It is clear that the transfer approach is not attractive for a multilingual *MT* system. However, it is still preferred to the interlingual approach among *MT* researchers because of the difficulties in defining the neutral concepts as mentioned earlier and because the intermediate representation of the transfer approach depends on language pairs, so the analysis and generation processes are less complicated than in the interlingual approach. The complexity of the analysis and generation modules in a transfer system is much reduced because the intermediate representations involved are still language-dependent. Some examples of *MT* systems that are based on transfer approach are Eurotra (22), Susy (22), and METAL (22).

In more general terms we can say that in the direct approach information is in dictionaries, phrases, and words, in the transfer approach it is in grammars, lexicons, and transfer rules, and in the interlingual approach it is in interlingual representations and lexicons.

Nonlinguistic Information Strategies. Different languages require different amounts of information to be expressed. The spanish word "Ilegó" means either "he arrived" or "she arrived," but in order to translate it into English, we have to choose "he" or "she" on the basis of context and background knowledge. Japanese and Thai have "polite" markers. To translate from English to these languages, the "polite" marker need to be added, based on the knowledge of culture. This observation has led to the nonlinguistic information strategies. Such strategies have recently become popular, in example-based *MT* (37), statistical strategies (38), and so on.

In *example-based MT* (EBMT, Fig. 4), there are three main steps: establish the bilingual corpus, retrieve the best match, and produce the translation. The first step is to collect a number of translation pairs (for example, English–French) in parallel, to produce what is called the *bilingual corpus*. Next, one maps the input

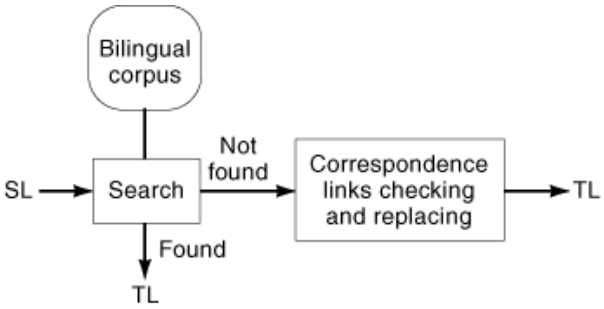

Fig. 4. Example-based MT.

sentence to an example in the collected corpus. If the entire input sentence can be matched, then it will be translated to the translation pair of that matched example. If not, then it takes two steps to translate such a sentence: *finding the closest mapping* and *recombining the matched segments*.

The first step is to look for the closest example. If it is found, the system will use the translation pair as a template and replace the SL word with the TL word. All matched segments will be translated and recombined to constitute the translation of the input sentence. The measure of similarity between a sentence segment and a set of examples can be based on both appearance and content. However, the system will have to determine the best match for each segment if there are more than one match. This selection can be done by using the clustering method (39). EBMT requires a large collection of translation (pair) examples to produce an accurate output. One advantage of EBMT is that the translations come with *scores*, which are measures of similarity to known examples. These scores could be useful to a posteditor.

Statistics-based MT tries to do away with formulating linguistic knowledge, but applies statistical techniques instead. This method requires a corpus that consists of aligned translated text. Alignment is established by a technique widely used in speech recognition. The translation technique is to calculate the probabilities that any one word in thSL corresponds to various words in the TL. In other words, the translation technique is to regard the occurrence of the TL as conditioned by the occurrence of the SL.

One sample system of this approach has been designed by an IBM research group. The researcher had access the three million sentence pairs from the Canadian (English–French) Hansard. Two sets of probabilities are calculated. First, for each individual English word, the probabilities of its correspondences to a set of French words; for example (22), the word “the” corresponds to

“le” with probability 0.610
“la” with probability 0.178
“l’” with probability 0.083
“les” with probability 0.023
“ce” with probability 0.013
“il” with probability 0.012

Second, the probabilities that two, one, or zero French words correspond to a single English word; for example, the word “the” corresponds to

two French words with probability 0.004
one French word with probability 0.871
no French word with probability 0.124

Therefore, the translation of "the" is most likely to be "le."

This approach, while interesting, requires for producing an accurate translation a very large corpus, which is difficult to compute and use, In addition, there is no linguistic information. Further details of the statistics approach can be found in Refs. 24, 38).

Many other approaches to *MT* have been (or are being) sought. In the *neural-net-based* (*connectionist-based*) approach the network is *trained* to identify the sense of words or to learn from past mistakes and from correction by humans (post-editors). Still other work has taken constraint-based (*CB*), lexicalist-based (*LB*), rule-based (etc.) considerations into account. Many attempts at *MT* today employ the intralingual or transfer approaches or some combination of the nonlinguistic approaches mentioned above.

Each *MT* approach needs a dictionary, either bilingual or monolingual. The dictionary has an effect on translation accuracy. An *MT* system with a perfect dictionary has a better chance of producing a better translation. A perfect dictionary means one containing a large number of words with complete information for each. One way to develop such a dictionary is by manual coding, but that is difficult, tedious, and error-prone. At this point, machine learning (*ML*) research can be used to facilitate this task. For example, the system can learn to guess the meaning and other information for an unknown word and add that learned word to the dictionary.

Fthermore, the translation task is not simple. To translate the SL sentence, the input sentence is analyzed in terms of morphology, syntax, semantics, and pragmatics, and then the target language is generated from the analyzed sentence. Again, in the generation process, the pragmatics, semantics, syntax, and morphology of the TL will be considered. If the system receives the same input sentence that was translated before, it will repeat the same process, which wastes time and other resources. However, if there is a collection of translated sentences (so-called *translation memory*), once the system receives new input, it will check with the collection first to see if the same sentence occurs there. That way, the translated sentence can be obtained easily without performing any analysis. If there is no exact same sentence, it can still be translated based on an example that has the same structure, by using example-based machine translation.

Hybrid Strategies. *Knowledge-based machine translation* (*KBMT*) (40) is based on the interlingual approach and an attempt to use Al techniques (reasoning—induction and abduction) in interpreting the SL text by reference to world knowledge, not only a knowledge of language. The meaning representation should be (merely) sufficient for translation to a number of languages, rather than sufficient for total understanding.

One advantage of this approach is in solving some ambiguity problems. However, there are some disadvantages: abductive or inductive reasoning is expensive, even on small domains, and building the knowledge representation is time-consuming.

Generate-and-repair machine translation (*GRMT*) (31) begins by generating the translation candidate (*TC*) for the SL by *quick and dirty MT* (QDMT). Next, *GRMT* evaluates the accuracy of that *TC* during the second phase, *translation candidate evaluation*. If necessary, *GRMT* repairs the inaccurate *TC* in the *repair-and-literate* phase.

GRMT integrates the best features of each previous approach. The process is simple, as in the direct approach. However, *GRMT* utilizes linguistic information to produce an accurate translation, like the transfer approach. *GRMT* also treats the SL and TL separately for easy management in multilingual *MT* systems, like the interlingual approach.

Knowledge Required, and Problems Inherent, in Machine Translation. Two types of primary knowledge are required for machine translation systems: language knowledge and world knowledge.

Language knowledge is required to produce a sensible translation output and is embodied in the *dictionary* and the *grammar*. The *lexicon* is needed for relating the words in the SL to the words in the TL language with the same meaning. However, word information alone is inadequate for translation because languages differ in structure—grammars are required.

World knowledge is required to solve ambiguity problems on various levels. For example, in the sentences

The two teams were trained by the same coach.

The coach was heading for the station.

The boy followed the man with the camcorder.

"coach" can denote either a vehicle or a person. For the first sentence, the coach denotes a person, as indicated by world knowledge (34). For the second sentence, it requires the context to solve the ambiguity problem. Both show the lexical ambiguity problem, while the third shows the structural ambiguity problem: it is not clear whether the camcorder was handled by the boy or the man. This ambiguity also can be solved using context. World knowledge is normally formalized by means of semantic features, logic, and so on.

Major problems face the designer of a *MT* system. Perhaps the most crucial one is how to represent a large amount of world knowledge. Another problem could be referred to as the *linguistic problem*: how to interpret the SL statement. Other major problem areas in the design of a multilingual *MT* system include:

- *Portability* (discussed in the next section):
- Linguistic knowledge must be encoded in a special-purpose formalism and be clearly independent of any programs that actually use such knowledge.
- A single grammar must be usable by both analysis and generation algorithms, for bidirectionality.
- Linguistic purity must be preserved: the content of a monolingual grammar of any language in an *MT* system must not be influenced by the other languages in the system.
- *Acquisition of linguistic knowledge* can be helped by machine learning and statistical techniques.
- *Ambiguity* is always a problem.
- *Robustness* refers to how the *MT* system responds to unexpected phenomena.

Some Contemporary Machine Translation Research Projects. *MT* research is a recognized field with a worldwide community of researchers, primarily in Europe, North America, and Asia. Some contemporary *MT* research projects around the world include:

GAZELLE. GAZELLE has been developed by the research group at the Information Sciences Institute, University of Southern California, to translate Japanese, Arabic, and Spanish texts into English. These programs operate over unrestricted newspaper text. The group is investigating the use of large-scale semantic representations and methods for automatically gathering linguistic knowledge inductively (statistically) from large on line text collections. GAZELLE is also deployed in a prototype translating copy machine that converts paper documents from one language to another. See http://www.isi.edu/natural-language/projects/GAZELLE.html.

KANT. The KANT project, part of the Center for Machine Translation (*CMT*) at Carnegie-Mellon University (*CMU*), was founded in 1989 for the research and development of large-scale, practical translation systems for technical documentation. KANT uses a controlled vocabulary and grammar for each source language, and explicit yet focused semantic models for each technical domain, to achieve very high accuracy in translation. Designed for multilingual document production, KANT has been applied to the domains of electric power utility management and heavy-equipment techni- cal documentation. See http://www.lti.cs.cmu.edu/Research/Kant.

SPANAM and ENGSPAN. *SPANAM* and *ENGSPAN* are fully automatic *MT* systems, developed and maintained by computational linguists, translators, and systems programmers at Pan American Health Organization (*PAHO*) in Washington. The translation unit has used *SPANAM* (Spanish to English) and *ENGSPAN* (English to Spanish) to process over 25 million words since 1980. Staff and free-lance translators postedit the raw output to produce high-quality translations with a 30% to 50% gain in productivity. The system is installed on a local area network at PAHO Headquarters and is used regularly by staff in the technical and administrative units. The software is also installed in a number of PAHO field offices and has been licensed to public and nonprofit institutions in the United States, Latin America, and Spain. See http://www.paho.org/.

Verbmobil. Vermobil, the speech *MT* system, has been developed under the sponsorship of the Federal Ministry for Education, Science, Research and Technology (BMBF), Germany, since 1993. The Verbmobil system recognizes spoken language and translates it into spoken English. The first fully integrated system, called the Verbmobil Demonstrator, was unveiled to the public in 1995 by Dr. Juergen Ruettgers, the German Federal Minister for Research. The Verbmobil Demonstrator recognizes spoken German input within the context of appointment negotiation (vocabulary 1292 words), analyzes and translates it, and finally utters the English translion. The Verbmobil research prototype (vocabulary 2500 words), to be introduced in 1996 towards the end of the first phase, additionally recognizes Japanese input and translates it into English. See http://www.dfki.uni-sb.de/verbmobil/VM.English.Mail.30.10.96.html.

Some Contemporary Commercial Machine Translation Projects. A number of translation software packages are available in the marketplace. Some of them are available for free translation online. Most packages are able to translate documents, Web pages, electronic mail, newsgroups, and even chat. Some of them are configured to be compatible with most Windows applications, including the leading office suites, email packages, Web browsers, and groupware packages. A sample of such programs is given below:

We consider SYSTRAN in some greater detail. As a system with over 30 years, experience in building translation software products, it cannot be overlooked. SYSTRAN, founded in 1968, develops and markets the leading *MT* technology, and provides a full range of automatic translation software products and services. SYSTRAN *MT* systems run on many different platforms, including Windows 95 and 98, NT, Linux, HP UX, Sun Solaris, Sun OS, and DEC Unix. To date, 16 language pairs are commercially available. In late 1997, SYSTRAN cooperated with AltaVista's Translation Service and pushed *MT* awareness to the forefront of the Internet community by offering any Internet user free real-time translations of Web content (BABELFISH).

The methodology used in SYSTRAN is generally characterized as based on the transfer approach. SYSTRAN translates sentence by sentence, concentrates on individual words and their dictionary data, then on parsing of the sentence unit, followed by the translation of the parsed sentence.

SYSTRAN architecture comprises dictionaries, systems software, and linguistic software. Details can be found at http://www.systransoft.com. A short summary is given below:

- *Dictionaries* The system traditionally employs three distinct, but interconnected types of dictionaries of all languages. The *stem dictionary* contains words in a root form with codes to indicate inflectional patterns, part of speech, syntactic behavior, semantic properties, and TL meanings together with codes needed for the target word generation. The *expression dictionary* is the dictionary of multiple-word expressions. The *customer specific dictionary* allows a user to add terms that were not found in the main dictionaries. This dictionary is available in PC Windows format.
- *System Software* The system software includes the general programs for handling input, the user interface, and dictionary look up procedures. It also controls the flow of linguistic modules and creates final formatted output.
- *Linguistic Software* The linguistic software includes a parser, TL translation modules, and a synthesis module. SYSTRAN's parser is composed of procedural modules that analyze the sentences in terms of both syntactic and semantic relationships. TL translation modules contain algorithms for constructing a translation. Translation information is derived for transfer and synthesis modules. The transfer component performs situation-specific restructuring, depending on the degree of difference between source and target languages. The synthesis module is to generate the TL strings that correspond to the information provided by all previous modules. The synthesis modules contain algorithms for creating specialized TL constructs, such as negation, questions, verbs with complete morphology, and articles.

It is clear that *MT* systems have been developing throughout the world and the expansion of *MT* actity has occurred around the world. For example, in Japan, "A rough guess would indicate that 800 to 900 people are presently engaged in research and development of *MT* systems (quoted from Ref. 29). *MT* works best if the

subject matter is specific or restricted. The results are even better when the original text is straightforward and devoid of ambiguities.

Evaluation. As an *MT* user, the first thing that you should be concerned with is your translation requirements; this concern includes the size of the translation task, the text type of the material, and its form. The time needed in updating the dictionary and preparing the input is a second concern. System performance (speed) is also a concern, as are the translation quality and the time required for postediting of the translation. The final concern is cost. The *MT* developer must take into account suitability to the user's needs. This includes the best and the most economical way to comply with the user's translation requirements, the facilities that must be provided, and the extendibility desired. *MT* researchers are normally concerned with particular types of errors connected with the specific linguistic matter. These concerns lead to strategies for evaluating the translation process to meet translation requirements.

To evaluate the translation quality, the most common aspects that must be addressed are:

- *Intelligibility.* The ease with which a reader can understand the translation
- *Accuracy.* The extent to which the translation conveys the same meaning as the original
- *Style.* The extent to which the translation uses language appropriate to its content and intention (22)

We now turn our attention to another major venue for natural language processing research—that of natural language access to databases.

Natural Language Interfaces to Databases

Many natural language problems arise when constructing *NLIs* to a database. For a more elegant system that would access knowledge in a knowledge base, rather than simply data in a database, additional problems arise. Such an elegant system would be truly sophisticated and useful to decision makers. For example, if a system responded to the question "Is model R2D2 for sale?" with "Yes, in limited quantities and by special order," more appropriate information would be available to the interrogator (assuming some user modeling) upon which to base decisions. Automatic methods for including knowledge of the domain, knowledge of the user, and knowledge of discourse structure in the *NLI* must be considered, as mentioned earlier.

The widespread use of databases by nonexpert users has made the development of sophisticated *NLIs* increasingly important and desirable, if not essential. In this section we describe some of issues surrounding the construction of *NLIs* to databases in greater detail by discussing *SystemX*, a *NLI* that translates English into the de facto standard relational database language SQL (41). We provide a thumbnail sketch of SystemX's evolution, including a discussion of its advantages, which include its *portability* and the ability to handle *arbitrarily quantified queries* and to *learn* the meanings of new words.

For many years the Natural Language Group at Simon Fraser University has been engaged in a long-term project entitled "Assessing Information with Ordinary Language," which has found its realization in several versions of SystemX. The initial SystemX *NLI* prototypes (English-to-SQL translation systems) were designed modularly utilizing proven technology (42), for-example, extended augmented transition network grammars (*ATNs*). The design of the interface thus served as an umbrella project for new ideas and technologies to be incorporated (41) as a testbed for various techniques espoused by our graduate students and visitors (3, 4, 15, 20, 43), and for experimenting with nonstandard or not yet completely specified theories (44, 45). We now present a thumbnail sketch of how this initial version of SystemX was constructed.

A Thumbnail Sketch of the Early SystemX. SystemX consists of a set of modules that creates a canonical query representation and a set of modules that translates this query representation into a (logical) form and then into SQL. The canonical query representation reflects the join path implicit in the query, any

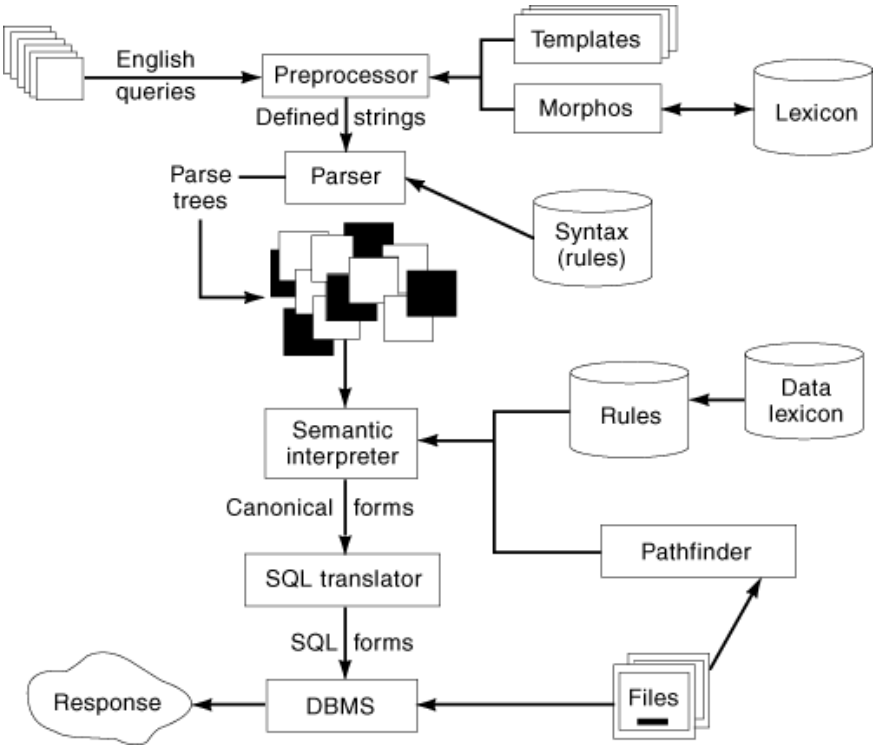

Fig. 5. A highly simplified graphical representation of SystemX.

predicates (such as *greaterthan*) that are to be applied against database values, and any operations (such as *average*) that are to generate values from the database, as well as quantifiers and their scope.

The lexicon, parser, and semantic interpreter construct the canonical query representation. Figure 5 portrays a simplified SystemX. Each module is independent and replaceable, provided that replacements accept input and provide output of the appropriate sort.

The lexicon consists of syntactic and semantic dictionaries. The analyzer contains a set of morphological rules that define the set of inflectional endings and permit inferences about the grammatical category of strings, and a set of respelling rules that describe how strings are to be transformed after inflectional endings have been removed. The morphological analyzer substantially reduces the amount of storage required for the syntactic dictionary, since only the root form of a word need be stored. Moreover, since the rules do not require a root dictionary to check against, the analyzer is used to generate new lexical entries, querying the user when an inflectional ending is used by more than one grammatical category and automatically updating the lexicon when a new word is unambiguously analyzed.

The semantic dictionary is divided into two parts: a domain-independent lexicon of predicates, operations, quantifiers, and so on, which are portable from application to application, and an application-dependent lexicon. The latter is largely generated automatically from the database schema. Customization is limited to a synonym dictionary. Further reducing storage requirements is a preprocessor, which examines queries for strings whose lexical representation, both syntactic and semantic, is defined by their form. These include proper names, standard code names and numbers, and so on. These strings are not represented in the lexicon, but are defined automatically when encountered.

SystemX's parser is a top-down breadth-first parser. Grammar rules are complied into Lisp code, which is compiled and applied to the input query by an interpreter. The set of grammar rules to be applied against any input string is determined by the intersection of the set of possible rules that will parse a candidate node and the set of rules triggered by the grammatical category of the first word in the input stream as well as the rules triggered by that word itself. Thus, unlike many top-down parsers, our parser will parse sentence fragments. With many parsers, particularly top-down parsers, the same input string will be parsed many times as the parser seeks the correct path to the end of the sentence. In our parser, when selected nodes are parsed, the resulting parse tree is placed in memory. Before an input string is parsed, this memory is first checked. If the input string has been previously parsed by the same rule, the parse tree is retrieved from the memory, resulting in significantly reduced parse times.

It is possible for a grammar rule to invoke the semantic interpreter to choose among competing parses. This flexibility enables the parser to reject syntactically possible but semantically anomalous trees at the source of the anomaly, instead of maintaining competing parses and rejecting anomalous trees after parsing is completed. Grammar rules may be assigned a rank relative to other that parse the same node, so that the order in which they are applied to a string is controlled. This ordering ensures that rules most likely to succeed are attempted first. A set of semantic interpretation rules is associated with each grammar rule; these rules are complied into compiled Lisp code. When the semantic interpreter is presented with a parse tree, it retrieves from the tree the name of the grammar rule that produced the tree. This name provides access to the appropriate set of translation rules.

Translation rules consist of three constituents. A set of transformation rules create an underlying form from the parse tree. Different underlying forms may have the same parse trees; parse trees of the same structure may have different underlying forms depending on their semantic content. The second set of rules are peculiar to the application. The final set of rules take the underlying form, perhaps augmented by the domain-specific rules, and pass them in the appropriate order to the appropriate functions, which create the canonical query representations. These functions are assisted by a set of hand-coded selectional restrictions, which block possible joins, and a small set of frames that describe the encoded database world, for example, "students take courses from professors."

Underlying the semantic interpreter is *Pathfinder*, which generates join paths given a set of two or more column names. Since terms are treated as values in the database, the semantic representation of an expression of two or more terms or expressions is the join path between them. The task of the semantic interpreter is to pass the column names represented by the terms in an expression to this system and build up a representation using the join path, which it returns. Since there may be more than one possible join path involving a given set of attributes, Pathfinder must solve a much-studied problem, the multiple-access-path problem (*MAPP*) (7, 46). Hall (42, 47) addressed this part of our prototype system. Treating semantic interpretation in this way results in a very flexible system, which requires little customization. An improved approach can be found in Refs. 48 and 49, in which a heuristic is developed that gives the same outcome in schemas designed by different database administrators.

Prior to semantic interpretation, information from the semantic dictionary is attached to terms that form the leaves of the parse tree. This information specifies the names of the database relations and attributes to which the terms correspond. The interpretation of the tree then proceeds in a bottom-up fashion from the leaves. The interpretation of a nonterminal node often requires establishing the relationship between attributes corresponding to the heads of the branches of the subtree rooted at that node. In other words, the access path to the database relation corresponding to the subtree must be found. Consider the N^A node in the left main branch of Fig. 6, which shows a parse tree in which the triangles indicate that information not relevant to our discussion has been deleted from the tree. Interpretation of this node requires derivation of the relationship between *math* and *students*.

Since a given set of attributes may belong to different database relations derived by different access paths, interpreting this nonterminal node requires a *MAPP* solution. The semantic translation rules responsible for

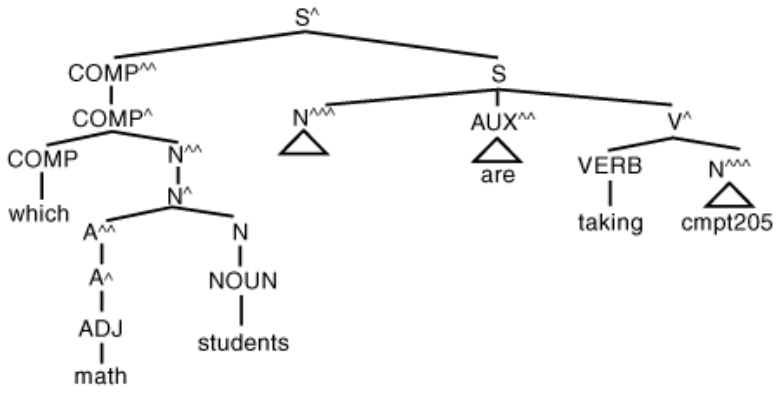

Fig. 6. A parse tree for "Which math students are taking CMPT205?"

interpreting this type of node employ Pathfinder to generate a *best guess* of the correct relation, which is assumed to be the relation containing the most cohesive relationship between the given set of attributes and is derived using a semantic model derived from the database scheme.

The modules that translate the canonical form to logical form and thence to SQL successively convert canonical forms containing any combination of unirsal quantifiers, existential quantifiers, and negation operators into complex SQL expressions that experienced SQL programmers would have great difficulty in composing. Routines in the translator perform some optimizations on the access paths specified in the canonical form so the SQL expression will more efficiently retrieve the data from the database.

Advantages of Early SystemX. For practicality, it is desirable to make *NLIs* portable. Many authors have written about portable *NLIs* (12,50,51,52,53,54,55). The most important portability issue is to promote easy attachment to databases representing different domains, that is, portability between applications. Closely related is portability within versions of the same database. Another important portability aspect is to connect the *NLI* to different types of database management systems. Also important is the ability to transport the *NLI* to different types of computers.

Two main approaches make *NLIs* portable between applications. One approach modularizes the system so that domain-specific modules can be exchanged; this approach is emphasized by Datalog (55). Another approach provides interface software for a database administrator or skilled user to furnish required domain-specific information; this approach is taken by TEAM (54). These systems require nontrivial human intervention, resulting in loss of portability.

We focus on *portability among different databases*. SystemX minimizes the amount of knowledge that humans must provide by optimizing the knowledge that the system can discover for itself by analyzing the database. The ultimate semantic interpretation of many natural language queries to relational databases consists of access paths through the logical structure of the database. Pathfinder automatically generates most of the access paths required to interpret the natural language input, thus reducing the effort required to build the semantic lexicon.

Much information to be learned is linguistic. The system must learn all of the words that users employ to refer to objects and relationships in the new database. The interface must also have knowledge of the contents and structure of the database. Most queries involve database navigation: values in different tables must be extracted and compared. The *NLI* must construct the navigation path though the database that will answer the query. In commercially available systems, this path must be created by experts before the system can be

used. Consider a simple query, In which room is math344 held? with SystemX's SQL translation:

```
select distinct c.room, a.class#, a.offer#
    from class a, offering b, schedule c
        where a.offer#=b.offer#
            and b.cname='math344'
            and a.class#=c.class#
```

To translate this query most systems would require knowledge that the word room referred to the attribute room#, that math344 was a noun that referred to the attribute cname, and that the database navigation proceeds from the offering relation through the class relation to the schedule relation. In contrast, SystemX requires only knowledge that room refers to room# and that course names consist of four characters and three digits. All other information (grammatical categories, and semantics of new words, and navigation paths) required to answer the query is generated by the system as required. New words are learned by the interaction of modules, which assign grammatical and meaning representations to a new word on the basis of its context.

This ability to learn considerably reduces the amount of information that must be created initially for each new database. Our approach departs from that of commercially available systems, which require extensive customization before they can operate, and avoids interruptions to update the knowledge base when users present them with new words. Because of this learning ability, SystemX requires a short, largely automated customization period. More importantly, it can often assess database-dependent information such as the meanings of new words and navigation paths for itself as they are required.

SystemX is able to handle a greater range of *quantifiers*, such as every and only, than all other natural language interface systems. Quantifiers are problematic in part because SQL is not able to express queries involving quantification in an obvious fashion. The example query Has every cmpt major taken at least 3 math courses? combines the problem of quantification with those of data organization and calculation:

```
ctr
 select a, student#
     from student a
         where a.major='cmpt'
         ***
             where b.dept='math'
                 and b.cname=d.cname
                 and c.offer#=d.offer#
                 and a.student#=e.student#
                 and a.class#=e.class#
                 and 883 > d.semester
             group by e.student#
                 having 3 ⇐ count (distinct d.cname))
```

The SQL query may translated as "The answer is no if there is a student who is a computer science major and it is not the case that student is a member of the set of students who have taken at least three math courses."

The highly modular architecture of SystemX permits the solution of many problems by simply adding more rules to the rulebase. Datalog can correctly interpret the query, What is the average of the salaries of the magers?, but not the similar query What is the salary of the managers?. Datalog's solution required modification of the representation scheme and a new module for handling the new representations. This situation could be handled in SystemX by someone without such expert knowledge.

Semantics may also be customized by creating selectional restrictions and frames. As part of our proposal and our invitation to discuss design specifications, we are examining how to provide SystemX with the capability to automatically generate its required frames.

More complete reports of SystemX have appeared (41, 42).

The New SystemX. At Rogers Cablesystems Ltd., executive vice president for customer service Ted Hotzak enters the following into his computer terminal: Give me the Western region outage log for June. Within seconds SystemX presents him with a neat table (or graph) of the data retrieved from the relational database. Ted could have said What's the outage log for the Western region for June?, or Tell me the June regional outage log for the West, or Find the Western outages for June.", or the like. SystemX can tell that whichever version Ted uses, he means the same thing. Such flexibility in parsing (applying the logical rules of grammar to determine meaning) is nontrivial. SystemX's parsing techniques will be described below. After parsing, another part of SystemX reformulates the question in SQL, and data are extracted for presentation from Roger's large central database.

The nontrivial problem described in the preceding paragraph is but one of a large number of very difficult problems of understanding natural language by computer, as discussed earlier. General analysis of language phenomena and much of the ambiguity inherent in unconstrained natural language understanding is limited, but complexities arise when building natural language capabilities into database interfaces. Without resorting to a sophisticated and wholly integrated knowledge-based system, which will be necessary in the longer term, we can construct systems to access information in ordinary language that are superior to the limited number of such products currently available. SystemX is one such advanced prototype. Although the original prototype SystemX functioned well, particularly in delivering in delivering fast parse times, its ad hoc design made extension to sophisticated linguistic constructions difficult. Thus we replaced the parser.

Head-Driven Phrase Structure Grammars—the Choice for the SystemX Grammar. As a grammar formalism we chose the head-driven phrase structure grammar (*HPSG*) (44, 45), which uses unification (56) as its primary control mechanism. The *HPSG* expresses grammatical knowledge in the lexicon rather than in rules. Our current implementation of the *HPSG* (57) contains only seven grammar rules. Extending the coverage of the grammar consists in building the lexicon, not in adding more rules.

Many contemporary linguistic theories, including lexical functional grammars (*LFGs*), situation semantics, generalized phrase structure grammars (*GPSGs*) government-binding (*GB*) theory, systemic grammar, and so on, have all been implemented within unification-based frameworks. A consequence of this embodiment within a unification-based formalism is the development of a expressive and formally precise lingua franca. Thus, many of the constructs and hypotheses of *HPSG* are borrowed or adapted from elsewhere.

HPSG is an information-based theory of natural language syntax and semantics. It was developed by synthesizing a number of theories mentioned above. In these theories syntactic features are classified as head features, binding features, and the subcategorization feature; thus *HPSG* uses several principles of universal grammar, three principles of which are (from Ref. 44):

Head Feature Principle. This principle is similar to the head feature convention (*HFC*) of *GPSG*. It states that the head features (e.g., part of speech, case of nouns, inflection form of verbs) of a phrasal sign are shared with its head daughter; thus the case of a noun phrase (*NP*) is determined by the case of its head noun, and the inflectional form of a verb phrase (*VP*) is determined by its head verb.

Binding Inheritance Principle. Binding features encode syntactic dependencies of signs such as the presence of gaps, relative pronouns, and interrogative elements. The binding principle is similar to the foot feature principle (*FFP*) of *GPSG*. Information about syntactic dependencies is transmitted upward through the constituent structure of signs until the dependency in question can be *bound* or *discharged*.

Subcategorization Principle. This principle is a generalization of the *argument cancellation* employed in categorial grammar. Subcategorization is described by a subcategorization feature (SUBCAT). SUBCAT's value is simply a list of the kinds of signs with which the sign in question must combine to become saturated. This principle states that in any phrasal sign, each complement daughter must satisfy a member of the head daughter's SUBCAT list, and that the SUBCAT list of the mother must consist of those elements on the head daughter's SUBCAT list that remain to be satisfied. For example, the SUBCAT value of the past-tense intransitive verb *walked* is the list ⟨*NP*[NOM]⟩, since *walked* must combine with a single *NP* in the nominative case (the subject) to become saturated; the past-tense transitive verb *liked* has the SUBCAT value ⟨(*NP*[ACC], *NP*[NOM]⟩, since *liked* requires both an accusative-case *NP* (the direct object) and a nominative-case *NP* (the subject).

HPSG theory borrows the notion of deep grammatical relation from categorial grammar. Rather than consider the surface order of argument structure like many linguistic theories, or even the semantic argument order, *HPSG* embodies a hierarchical theory of grammatical relations. *HPSG orders* according to those signs that satisfy (unify with) the elements, respectively, of the verb's SUBCAT list. Thus *HPSG* employs an order-free notion of *semantic role* similar, to the one employed by situation semantics. Since words (lexical signs) in *HPSG* are highly structured, in view of the principles mentioned above, the flow (sharing) of information is constrained between lexical signs and the phrasal signs they head ("projections"—the projection principle of *GB* theories). *HPSG* principles are explicitly formulated, and thus their implementations are more likely to be faithful to theory. There is less work for language-specific rules of grammar. In Pollard and Sag 44 only four highly schematic *HPSG* rules accounted for a substantial fragment of English. One rule, informally written as

$$[\text{SUBCAT}\langle\ \ \rangle] \rightarrow \text{H}[\text{LEX}-], \text{C}$$

subsumes a number of conventional phrase structure rules, such as those below:

$$\text{S} \rightarrow \text{NP VP}$$
$$\text{NP} \rightarrow \text{DET NOM}$$
$$\text{NP} \rightarrow \text{NP's NOM}$$

In the *HPSG* rule, one possibility is that the English phrase is a saturated sign [SUBCAT ⟨⟩] (with ⟨ ⟩ denoting the empty list) and has for its constituents that a phrasal head (H[LEX -]) and a single complement (C). Another *HPSG* rule, informally expressed, is

$$[\text{SUBCAT}\langle[\ \ ]\rangle] \rightarrow \text{H}[\text{LEX}\ \ +], \text{C}^*$$

This rule states that another option for English phrases is to be a sign that subcategorizes for exactly one complement [SUBCAT ⟨[ ]⟩] with "⟨[⟩]⟩" standing for any list of length one, and whose daughters are a lexical head (H[LEX +]) and any number of complement daughters. This rule subsumes a number of conventional

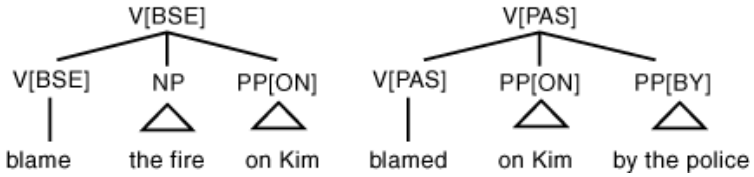

Fig. 7. Same phrase structure rule for lexical signs.

phrase structure rules, such as those below:

$$\begin{gathered}\mathrm{VP} \rightarrow \mathrm{V}; \mathrm{VP} \rightarrow \mathrm{VS'}; \mathrm{AP} \rightarrow \mathrm{A};\\ \mathrm{VP} \rightarrow \mathrm{V\ NP}; \mathrm{AP} \rightarrow \mathrm{A\ PP}; \mathrm{PP} \rightarrow \mathrm{P\ NP};\\ \mathrm{VP} \rightarrow \mathrm{V\ PP}; \mathrm{VP} \rightarrow \mathrm{V\ VP}; \mathrm{VP} \rightarrow \mathrm{V\ AP};\\ \mathrm{VP} \rightarrow \mathrm{V\ NP\ NP}; \mathrm{VP} \rightarrow \mathrm{V\ NP\ PP}; \text{etc.}\end{gathered}$$

HPSG rules determine constituency only; this follows *GPSG* theory, where generalizations about relative order of constituents are factored out of phrase structure rules and expressed in independent language-specific *linear precedence* (*LP*) constraints. Unlike *GPSGs*, some *LP* constraints may refer not only to syntactic categories but also to their grammatical relations. An example of a hierarchical *LP* constraint is

$$\mathrm{COMPLEMENT[MAJ} \sim \mathrm{V]} \ll \mathrm{COMPLEMENT}$$

Roughly interpreted, this constraint says that in English any complement with major feature (part of speech) other than V (that is, any complement whose head is not a verb) must linearly precede its more oblique complement sisters. There are many consequents of this constraint; for example, direct objects precede second objects.

Additional lexicalization of linguistic information and further simplification of the grammar are achieved in *HPSG* by *lexical rules* (similar to those of LFG). Lexical rules operate upon lexical signs of a given *input* class, systematically affecting their phonology, semantics, and syntax to produce lexical signs of a certain *output* class. For example, English passivization uses the lexical rule (stated informally) below:

$$\begin{gathered}\langle \phi, \mathrm{V[BSE, SUBCAT}\langle \ldots, \mathrm{NP, NP}\rangle\rangle \Rightarrow\\ \langle \mathrm{passive}\phi, \mathrm{V[PAS, SUBCAT}\langle (\mathrm{PP[BY]}, \ldots, \mathrm{NB}]\rangle.\end{gathered}$$

The input word, a base-form verb (V[BSE]) with phonological form ϕ that subcategorizes at least for a subject *NP* and a direct object *NP* (SUBCAT ⟨ . . ., *NP*, *NP*⟩) is mapped to a passive-form verb (V[PAS]) with phonological form passive(ϕ), where passive is a morphological operation that systematically produces the regular passive morphology. The example below illustrates the effect of the lexical rule given above:

$$\begin{gathered}\langle \mathrm{blame, V[BSE, SUBCAT}\langle \mathrm{PP[ON], NP, NP}\rangle]\rangle\\ \langle \mathrm{blamed, V[PAS, SUBCAT}\langle (\mathrm{PP[BY]},) \mathrm{PP, ON], NP}\rangle]\rangle\end{gathered}$$

Phrases headed by lexical signs like these can now be produced from the same phrase structure rule, e.g., [SUBCAT ⟨[ ]⟩] → H[LEX +], C∗, as indicated in Fig. 7.

HPSG is *not* a theory of syntax. Researchers into *GB*, *GPSG*, and LFG have focused on syntax, relying mainly on a Montague-style system of model-theoretic interpretation. In contrast, *HPSG* theory inextricably intertwines syntax and semantics. Syntactic and semantic aspects of grammar are built up in an integrated way from the start. Thus *HPSG* is closer in spirit to situation semantics, and this closeness is reflected in the choice of ontological categories in terms of which the semantic contents of signs are analyzed: *individuals, relations, roles, situations*, and *circumstances*. The semantic content of a simple declarative sentence is a circumstance, a situation-theoretic object composed of individuals playing roles in a relation. This formulation is a more precise account of the earlier conceptual dependency theory and preference semantics formalisms of the early 1970s, based in part on Fillmore's case grammars (58). Thus, the semantic content of the sentence John admires Jane is [an attribute–value matrix (*AVM*)] as follows:

$$\begin{bmatrix} \text{RELATION} & \text{ADMIRE} \\ \text{ADMIRER} & \text{JOHN} \\ \text{ADMIREE} & \text{JANE} \end{bmatrix}$$

The semantic content of sentences, and of phrases generally, is determined by various pieces of syntactic and semantic information associated with their constituents, in conjunction with universal linguistic principles (and contextual factors). In the example above, John and Jane are part of the semantic contents of the subject *NP John* and the direct object *NP Jane*. The relation ADMIRE and the assignment of the ADMIRER and ADMIREE roles to the subject and direct object come from the head verb *admires*, which has the form

$$\left\langle \text{admires, V[SUBCAT}\langle \text{MP}i, \text{NP}j \rangle \text{]}, \begin{bmatrix} \text{RELATION} & \text{ADMIRE} \\ \text{ADMIRER} & j \\ \text{ADMIREE} & i \end{bmatrix} \right\rangle$$

Note that the lexical sign consists of phonological, syntactic, and semantic information. The crucial assignment of semantic roles to grammatical relations is mediated by the SUBCAT feature. i and j are variables in the semantic sense of classical mathematical analysis. The specification "NPi" demands a noun phrase whose variable is to be identified (unified) with the filler of the ADMIREE role. The subcategorization principle ensures that the variables j and i are then unified with John and Jane. Finally the semantic content of the sentence now follows by an additional universal *semantic principle*, which requires that the semantic content of a phrase be unified with that of its head daughter. Whereas Montague-style semantics are determined by syntax-directed model-theoretic interpretation, in *HPSG* theory the semantic content of a sentence's lexical constituents "falls out" by virtue of the general linguistic constraints, which require that information pieces associated with signs be unified with other pieces.

These few paragraphs are intended only as the shortest of introductions to *HPSG* formalisms. Other linguistic aspects, such as agreement, interpretation of "understood subjects," gaps, and analysis of reflexive and nonreflexive pronouns, are generally treated in a manner consistent with unifying principles just described.

Considering the new SystemX, representations as described are built as queries are parsed. As words and phrases are combined into larger phrases, the representations associated with each constituent are combined. In simple cases, the process by which they are combined is unification. For example, in the phrase

$$[4-1] \quad \text{the outages for June}$$

the representation of "outages" contributes an *AVM* that specifies a table and a set of columns from that table. The representation of "June" contributes a matrix that specifies a column and its value (the database representation of "June"). The representation of "June" does not specify a table, because many of the tables

in the domain contain the column represented by "June." The preposition "for" does not have a database representation. Consequently, the representation of "for June," that is, the unification of the representations of "for" and "June," is the representation of "June." The representation of the entire phrase in [4-1] is created by unifying the constituents "outages" and "for June." The effect of this is to supply a value for the column representing dates in the table represented by "outages."

The example in [4-1] illustrates cases in which adjunct modifiers further specify restrictions on a particular table in the database. Of course, in a relational database this is not sufficient, as it is often necessary to build up virtual relations by joins on different tables. For example, in the database developed for Rogers Cablesystems locations are represented by numeric codes. As is the case for dates, location codes appear in many tables in the database. However, the description of each code resides in a table that is joined with any table containing a location code. Thus, in the phrase

$$[4-2] \quad \text{the Vancourver outages for June}$$

the term "Vancouver" contributes an *AVM* representing a column in the table containing descriptions of locations. Notice that because the *AVM* representing "Vancouver" specifies a table, it cannot unify with that representing "outages," the *AVM* for which also specifies a table, one different from that specified by "Vancouver." Instead, the representation of the phrase will contain two tables, the join between them being represented by identical variables in the columns on which they are joined. The column on which tables are joined is determined dynamically by the module Pathfinder described in Ref. 47. This facility determines for any set of columns the minimal path through the database that will join the set into a virtual relation. This module eliminates the need for customizing joins.

Frequently in natural language processing (*NLP*) systems, researchers encounter problems in determining the meaning of complex constituents. In many cases, the meaning of a complex phrase cannot be viewed as a simple composition of the meaning of its parts. One such example from the Rogers Cable Systems database domain is the phrase "trouble call ratio." When one asks for the "trouble call ratio," one is interested in information from a specific database table. However, when asking for the "trouble call ratio total," not only is one interested in a different table, but one is interested in different columns as well. Furthermore, a join is not necessary either. In this case, the simple unification of the *AVMs* associated with each of the constituents will not be able to produce the kind of information structure that we need. The operation of unification, by definition, incorporates all of the information of the initial structures—it never throws information away.

In arriving at a solution to this problem, we notice that the information from the *AVM* for "trouble call ratio" that is used in the *AVM* for "trouble call ratio total" is limited to one column specification. Within *HPSG*, it is possible for a word like "total" to *examine* the *AVM* that it is to be combined with (in our case, the *AVM* for "trouble call ratio") and extract specific information. Based on this examination, it can then determine what the *AVM* for the complex phrase should look like. So the *AVM* associated with the complex phrase "trouble call ratio total" can be obtained from the *AVM* for "total." The *AVM* for "trouble call ratio" thus does not play a direct role in determining the *AVM* for the complex phrase; it contributes only information that "total" selects from it.

Cases like this, where one distinguished constituent selects information from its neighboring constituents and then imposes its own *AVM* for that of the complex phrase, are handled by introducing a feature into the *AVM* that states that it has *priority* over another *AVM* in unification. When two constituents with priority are to be combined, then no single constituent is assumed to have priority, and the usual unification is attempted. There is a related mechanism, similar to the priority mechanism, by which an *AVM* can be specified to contain *default* information. Thus, when a default *AVM* is unified with some other *AVM*, it is the information from the other *AVM* that is used. Essentially, specifying an *AVM* as default is equivalent to saying that any *AVM* that it is combined with is given priority.

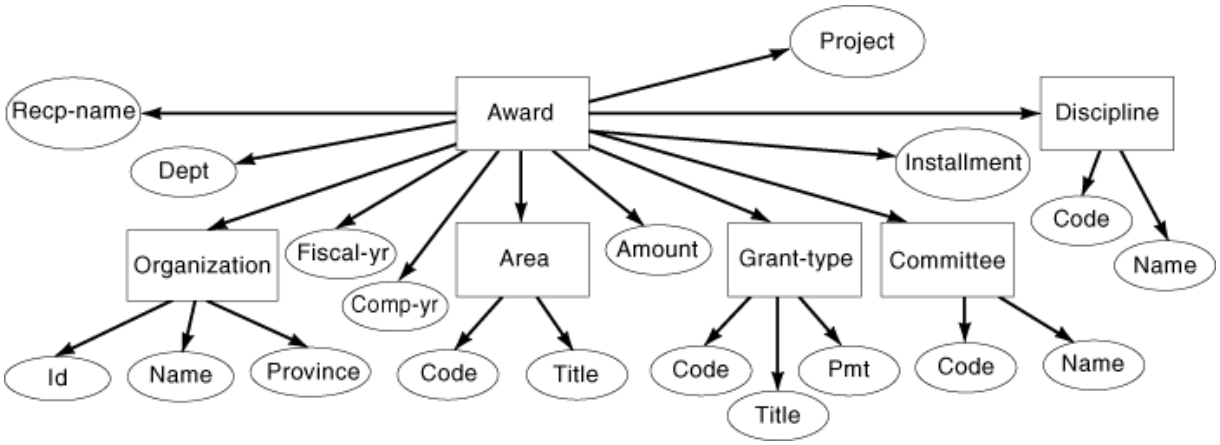

Fig. 8. Schema diagram for *NSERC* Grants Information System.

This interpretation differs from default unification and overriding as usually discussed in unification-based formalisms (59). Default unification merges part of one *AVM* with another in such a way that incompatibilities between different *AVMs* are avoided; default unification never fails. When we specify an *AVM* as having priority, it overrides all of the information from the other *AVM*. Other mechanisms are used to extract the relevant information from the *overridden AVM*. In cases where both *AVMs* are specified as having priority, unification may fail. These examples of compositionality, as in "Vancouver outages," and noncompositionality, as in "trouble call ratio total," are prevalent in the database for Rogers Cablesystems. They serve as illustration that the principles of natural language interface design must be sufficiently robust to handle accidental properties of the database schema.

We now turn to some examples of SystemX in action.

Examples of the New SystemX. We were privileged to receive the Grants Information System (*NGIS*) of the Natural Sciences and Engineering Research Council (*NSERC*) of Canada as a testbed for our experiments. Since the NGIS was implemented as a self-contained dBase system, we reproduced its database on both the Oracle and Sybase relational database systems, preserving the relations inherent in the original implementation, and ran tests on them with SystemX. Our experiments demonstrated that our system showed great promise for providing users of the NGIS and similar systems with valuable capabilities for ad hoc querying that they currently lack.

The NSERC Grants Information System. The NGIS is a software package consisting of a database of information about grants that are awarded by *NSERC*, and a menu-based interface to that database. It is intended to be used by individuals in "universities, government agencies and industry . . . to search for grants that are of particular interest." In addition to the detail about individual grants, the system provides summary statistics and reports.

The central table in the database is made up of rows, each of which describes an award by *NSERC* to a researcher. ("Award" is used here in the sense of a particular installment of a particular grant.) The values constituting each row (i.e., the columns constituting the table) specify the different properties of the award, including the name of the recipient, the amount of the award, and so on. In the schema diagram in Fig. 8, nodes representing the properties of awards are represented by nodes linked to the "Award" node.

There are a number of subsidiary tables, which record details about some of the properties of awards (e.g., the province of the organization in which the recipient is to carry out the research). Most subsidiary tables are

used simply to associate (English and French) phrases describing the entity to a code denoting a particular entity. In the schema diagram, tables are specified by rectangular nodes.

The Interface. The interface to the NGIS is menu-based. Using the menus, users may search for sets of awards or may request certain summary reports. Searches may be based on any one of the properties of awards alone, or on a combination selected from a given subset of the properties (i.e., on one or more of university, department, grant type, committee, and area of application). The interface automatically provides summary statistics (e.g., total amount, average amount) for the award set forming the result of a given search. Searches retrieve all awards that have user-specified values for each of the properties on which the search is based. Users may specify a set or range of values for some properties (e.g., for discipline), but are limited to a single value for most properties. This limitation forces users who wish to associate data describing different sets of awards, (e.g., to bring together information describing awards in two universities) to conduct separate searches for each set and make the association manually.

There are six different summary reports available. They categorize awards based on a particular property (e.g., committee) and give a table of summary statistics measuring these categories of awards for a user-specified grant type and university. Some reports include a bar chart illustrating one statistical measure. For example, one report provides a table detailing the number, total amount, and average amount of awards for each major area of application for a given grant type and university. In addition, a bar chart shows the percentage of money awarded by area to the university for the specified type of grant. The system limits users to making comparisons with one university for a single grant type in a single query. In addition, these reports take a long time to generate. Thus, a user who wishes to see summary data for only two or three of the areas of application, for example, must wait until the statistics have been generated for all areas and pick out the statistics in which (s)he is interested from the resulting report.

The Need for Ad Hoc Querying. While the NGIS is largely successful in providing the "easy and effective" access that its designers intended, it is still somewhat limited in that access to certain information (particularly information involving the comparison of data involving different elements of the same domain) requires a significant investment of time and effort on the part of the users. A *NLI* to the database could provide ready access to this kind of information to sophisticated users who desire it. In other words, the NGIS is limited in its usability by the lack of ability to query the database in an ad hoc manner. While the menu-based interface does permit access to all of the data contained in the database in one form or another, it places much of the burden on the user to derive the information (s)he is seeking. For example, suppose a user wishes to know the answer to the following question: "What types of grants were received by SFU Computing Science?"

The best way to access this information via the existing interface is to request a search on University ("SFU") and Department ("Computing Science") and then page through set of awards that are returned, one at a time, noting the different grant types represented. If an ad hoc query capability were available, the system could return immediately the simple list the user desires. (See example below.) There are many examples of this type of situation, such as "Which universities won major equipment grants?" or "Which departments won grants for research into expert systems?"

Not only is the NGIS user required to put time and effort into extracting information from the response, but also (s)he is forced to wait while the system retrieves, calculates, and formats the noise that obscures it from immediate view.

As noted, the system requires users who wish to compare information about different categories to access the information related to each category separately and make the comparison manually. For example, to learn whether the CS Department at SFU was awarded more operating grant funds than was the CS Department at UVic, the user would need to conduct two separate searches, record the results and make the comparison. Again, this query is typical of many that a considerable number of users would likely wish to submit.

The version of the NGIS we used contains data from a single year only. Undoubtedly these problems will be compounded when data from multiple years are present, since users will wish to make comparisons over time. One useful way to present data across time is to make a trend of the type provided by our natural language

inteface to users of the RIAS executive information system (for Rogers Cablesystems). Research indicates that executives and others frequently access data in a top-down fashion, first examining high-level summaries before "drilling down" to the more detailed level. Adding the capability to graphically display historical trends will allow NGIS users to quickly access the information they require.

Experiments with the Natural Language Interface.

```
Enter sentence to be parsed:
what types of grants were received by sfu computing
     science in 1990
Parsing Completed.
1 New query              6 Consult Oracle
2 Display parse tree     7 Display a Trend
3 Display semantics      8 Load Lexicon
4 Display Logical Form   9 Toggle Parse (LISP or PROLOG)
5 Display SQL            10 stop
?4
((SETX X1
        (RELATION AWARD
                (GRANT CODE FISCAL_YR
                   ORG_CODE DEPT)
                (X1 '1990 '1030 "COMPUTING SCIENCE"
                  (= = = =))))
1 New query              6 Consult Oracle
2 Display parse tree     7 Display a Trend
3 Display semantics      8 Load Lexicon
4 Display Logical Form   9 Toggle Parse (LISP or PROLOG)
5 Display SQL            10 stop
?5
SELECT DISTINCT B.GRANT_TITLE
  FROM AWARD A, GRANT_TYPE B
    WHERE A.FISCAL_YR = 1990
        AND A.ORG_CODE = 1030
        AND UPPER (A.DEPT) = 'COMPUTING SCIENCE'
        AND A.GRAND_CODE = B. GRANT_CODE
1 New query              6 Consult Oracle
2 Display parse tree     7 Display a Trend
3 Display semantics      8 Load Lexicon
4 Display Logical Form   9 Toggle Parse (LISP or PROLOG)
5 Display SQL            10 stop
?6
Consulting Oracle.
what type of grants were received by sfu computing
     science in 1990
GRANT_ TITLE
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
Individual Operating
Infrastructure
Microelectronics Fund

3 row(s) selected
```

With a slightly modified interface module we have the following (conceptually simpler) example:

```
Enter sentence to be parsed:
which universities won grants for expert
   system research in 1990
Parsing Completed.
?Display SQL
SELECT DISTINCT B.ORG_NAME
FROM AWARD A, ORGANIZATION B
WHERE A.AREA_CODE = 865
AND A.COMP_YR = 1990
AND A.ORG_CODE = B.ORG_CODE

?Consult Oracle
Consulting Oracle.
which universities won grants for expert systems research in
1990
ORG_NAME
- - - - - - - - - - -
Acadia
Alberta
British Columbia
Calgary
Carleton
Guelph
***
McMaster
Regina
Ryerson Polytechnic Institute
Simon Fraser
Toronto
Victoria
Ecole Polytechnique

   14 row(s) selected
```

System Design. SystemX, outlined graphically in Fig. 9, is implemented on a Sun SPARCstation running SunOS, OpenWindows, Sicstus Prolog, and Common Lisp. The interface accepts queries from the user (on the right side of Fig. 9) specified wholly or partly in English. Requests for tabular displays of data are completely specified in natural language. When requesting graphic displays, users specify, in natural language, the statistic to be graphed and specify the graph parameters by selecting items from system-generated menus.

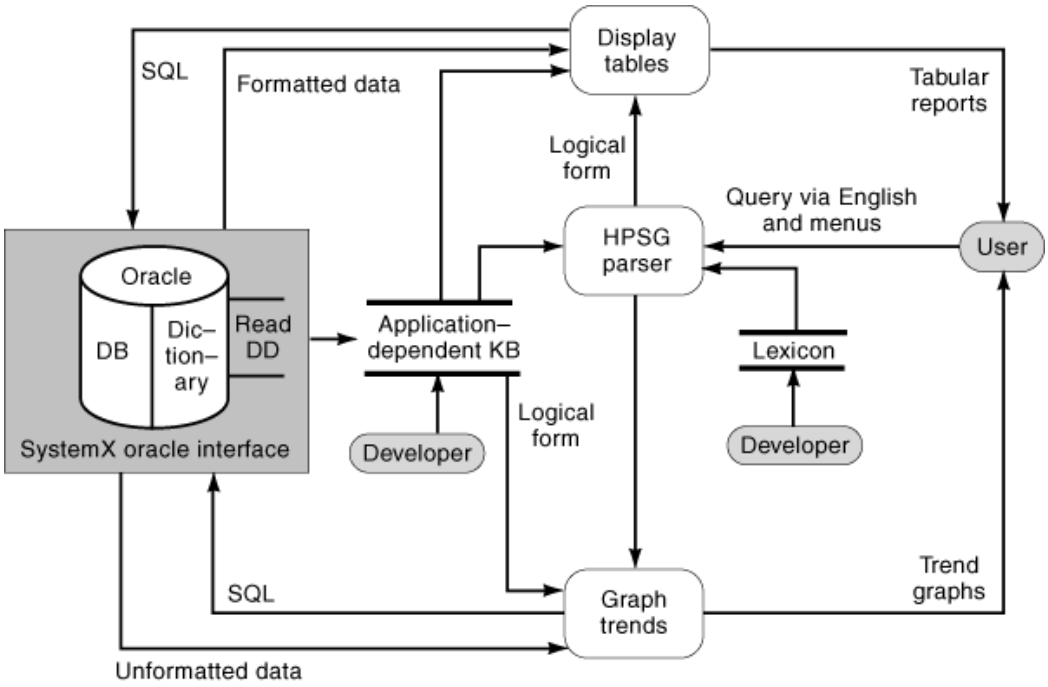

Fig. 9. Overview of SystemX.

Natural language is translated into a semantic representation and is converted into a logical form (*LF*) and finally to SQL by an *HPSG* chart parser (60, 61) as illustrated in Fig. 10.

Data are retrieved from the database and displayed in the format (table or graphed trend) specified by the user in the query. The "Display Tables" or "Graph Trends" routine passes the logical form received from the parser as SQL to the database and returns the table(s) or graph to the user. The interface is customized to a given application via the creation of the parser's lexicon and an *application-dependent knowledge base* (*ADKB*) containing information about the target database. The *ADKB* is constructed with information extracted automatically from the data dictionary (*DD*) of the database as well as with information supplied by individuals familiar with the domain.

The parser (Fig. 10) uses a grammar of four rules and a lexicon of approximately 500 words, acronyms, abbreviations, and common misspellings (such as "eastern"). The lexicon is organized as a multiple inheritance hierarchy of classes called *lexical types* (adopting the terminology of Ref. 12). An inheritance mechanism operates on the lexicon to produce an *expanded* lexicon of entries, which is then used by the parser (processing is done offline, before parsing is attempted). The parser can handle most of the sentences and sentence fragments from the second corpus described above, including the compound nouns listed.

SystemX permits the user to generate printed copy. This functionality is currently accessed by a menu interface (see the sub-subsection "Experments with Natural Language Interfaces" above). The user has the choice of inputting his trend request using English, using menus (*canned* trends), or using combination of English and menu responses. The various input modalities are provided as a convenience to users. Canned trends display data that are predictably desired on a reasonably frequent basis. They may be accessed with a minimum of keystrokes. The canned trends are those available through the first eight menu items in the trend menu in Fig. 11, which illustrates the new SystemX.

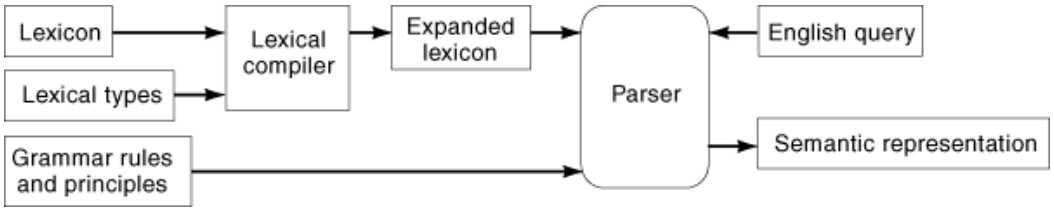

Fig. 10. The SystemX parser.

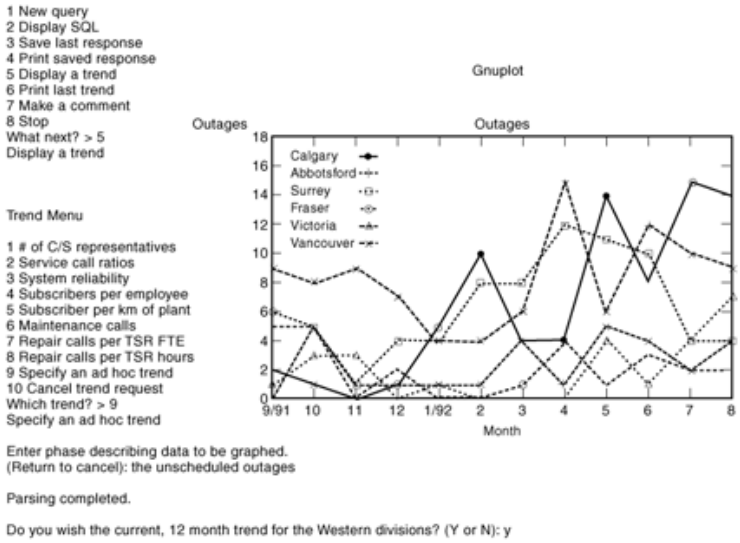

Fig. 11. An example of the latest incarnation of SystemX.

Field Test, Evaluation Methods, and Main Findings. Real-world user interface evaluation is currently an active area of research (62). We prepared a real-world field test using both formative and summative evaluation methods (63). Formative evaluation provides information about how to refine and improve the design of a system; summative evaluation provides information to assess the system's impact, usability, and effectiveness. Qualitative (formative) and quantitative (summative) measures were therefore required. Minimally intrusive evaluation methods were employed, including logging, on-line commenting, questionnaire/interviewing, and videotaping. The user's interactions with the system were automatically logged in a file and timestamped. An on-line commenting facility accepted and stored typed, time stamped comments. The user was videotaped using the system twice during the month-long field test. A follow-up interview was guided by a questionnaire.

Full results from our evaluation are presented in Ref. 64. The main findings are that 68% of requests for data were partially or completely specified in natural language. The overall success rate for answering queries (for tables) entirely in natural language, eliminating clearly unintended queries, was 82.1% (46 of 56 queries). The success rate in parsing requests to display a trend, in which natural language was used to query for a base statistic, eliminating unintended queries, was 95.2% (139 of 146 queries). Note, however, that although a request to display a trend may have been successfully parsed, it was not always possible to display a graph. This is because the requested statistic sometimes referred to several data, and GnuPlot, the graphing package used, could only graph one item of data.

Even including unintended queries, our success rate for answering queries entirely in natural language, 69.7% (46 of 66 queries) compares favorably with the 65.1% of queries successfully answered by the TQA *NLI*, while on field trial (65) a percentage Petrick reported as "consistent with but slightly smaller than that of such other investigators" such as Refs. 66,67,68. The 95.2% success rate with partial natural language queries for a base statistic only, and the popularity of this form of querying with the Rogers executives [of the 219 requests that used natural language (*NL*), including unintended queries, 153 (69.8%) were partial *NL* queries; this popularity may be due in part to the user's preference for requesting data in graphical form rather than tables], suggests that natural language may well be very useful for filling in parts of queries (perhaps as a form to be completed), as well as whole queries.

Petrick (65) notes that "extreme care should be exercised in interpreting any such overall numbers, however, and even more care must be exercised in comparing numbers from different studies." We agree that care must be taken in interpreting results. Only one Rogers Cablevision executive was involved in the field test. Nonetheless, the results from the evaluation have been used to improve SystemX, especially its interface, in ways we likely would not have anticipated without input from real-world users.

Having examined two of the major goals of natural language understanding at some length, let us turn our attention to a major goal, that of question-answering systems. We consider their approaches and shortcomings and the current efforts to utilize natural language techniques in a less demanding enterprise such as information retrieval and text summarization.

Question Answering, Information Retrieval, and Text Summarization

Question-Answering Systems. But while we are confined to books, though the most select and classic, and read only particular written languages, which are themselves but dialects and provincial, we are in danger of forgetting the language which all things and events speak without metaphor, which alone is copious and standard. Much is published but little printed. The rays which stream through the shutter will be no longer remembered when the shutter is wholly removed. No method or discipline can supersede the necessity of being forever on the alert. What is a course of history or philosophy, or poetry, no matter how well selected, or the best society, or the most admirable routine of life, compared with the discipline of looking at what is always to be seen? Will you be a reader, a student merely, or a seer? Read you fate, see what is before you, and walk on into futurity.

Henry David Thoreau, Walden

Early Question-Answering Systems. Interest began to grow in the late 1950s in natural language question-answering systems. These designs contemplated machines that engage in conversation rather than machines that generate sentences in response to pictures, retrieve documents, or convert English into logical calculi.

It has been well recognized that most statements can be transformed into questions through the use of intonation, the question mark, and the rearrangement of the subject and the verb. Often clues such as the words "who," "why", end "what", also provide delineations between questions and declarative statements. While linguistic differences between questions and declarations are well known, logical differences are not. A number of the earlier question-answering systems dealt with logical differences by adopting the point of view that a question is a special subclass of assertions whose propositional truth or falsity can be determined.

Most early question-answering systems made use of a dictionary as well as a small predetermined domain of discourse, and the *Conversation Machine* of Green, Berkeley, and Gotlaeb (69) is no exception. For that machine, the domain in which conversation is carried out is the weather. Their purpose in choosing such a conversational topic was to exploit Turing's notions of machine thinking.

Meaning, experience, and the environment were considered by the Conversion Machine. Environment was defined as the day's weather; experience was the knowledge of weather types that characterized various parts

of the year. The dictionary represented meaning, where words were stored along with their meanings. Words were classified as *ordinary* (snow, rain, hail, etc.), *time* (today, yesterday, etc.), and *operator* (change, stop, etc.). Meanings of the words were stored as attribute–value pairs. For example, the attribute of a time-classified word is a type of time (calendar or relative), and the associated value is a code for the amount. Operator words have attributes of functions to be accomplished and values of the degree to which the associated functions are to be executed. For example, the functions of change and stop are the same, but the degree code for stop is greater than for change. Ordinary words are coded similarly to operator words.

The meaning of a declaration or assertion is represented by the set of codes for the constituent words. Words not in the dictionary are considered meaningless and are assigned a zero code. Each word is found via table lookup, and coding is assigned by its attribute–value pair. The Conversation Machine then compares this "meaning" with its own store of coded declarations and selects an appropriate response frame from its storage in which there are blanks to be filled in with words from the original declaration or from experience.

An example from Ref. 66 is the sentence "I do not enjoy rain during July." The words "not" and "enjoy" yield the word "dislike," which is meaningful to the conversation machine. The set of words {dislike, rain, July} gives the resultant meaning to the sentence. July is associated with heat and blue skies, however. This gives rise to an essential disagreement, and the conversation machine chooses a reply on this basis, such as "Well, we don't usually have *rainy* weather in *July*, so you will probably not be disappointed."

The conversation machine limits syntactic analysis to only a few constructions and the problem of selecting the right response. Its principle of coding words, however, based on the attribute–value pair, has been used in subsequent and to a degree in current question-answering systems.

The BASEBALL program of Green et al. 70 is an expert on answering certain kinds of queries about baseball facts and figures from a stylized baseball yearbook stored in computer memory. The program consists of two parts. The linguistic part reads input, syntactically analyzes it, and determines what information is given about the data being requested. The processor part searches through the data for the appropriate information, processes it, and prints the answer. Questions are limited to single independent clauses. Logical connectives such as "and," "not," and "or" are prohibited, as are constructions implying the relations highest, most and so on.

The concept of a *specification list* (spec list) is fundamental to the operation of BASEBALL. The spec list represents in formation in the question in the form of attribute—value pairs. The spec list is generated by the linguistic part of the program and governs the processor part. The spec list for the question "Where did the Red Sox play on July 7?" is {Place =?; Team = Red-Sox; Month = July}.

Dictionary definitions are also stored as attribute–value pairs and are used to generate the spec list. Separate dictionaries are kept for words and idioms. Meanings of words may take several forms. They may have main or derived attributes and also may designate a subroutine that will be executed by the content analysis. Some words ("the," "did," "play") have no meaning attached. Data are represented as a hierarchical list structure. Any one path through the data, however, must contain all the facts for a single game.

The program includes linguistic routines with input, dictionary lookup, syntactic analysis, and content analysis routines, while the processor includes the processor and responder routines. Syntactic analysis is based on the parts of speech (stored in the dictionary along with the attribute–value pairs), which are syntactic categories assigned to words for use by the syntax routine. Fourteen parts of speech and ambiguity markers are used. Ambiguities are resolved by looking at adjoining words and resolving through context. The noun phrases and the prepositional and adverbial phrases are next located and bracketed. For example,

((How many games) did (the Yankees) Play (in (July)))

The verb is then checked to see whether it is active or passive, and the subject and object located.

Content analysis uses the results of the syntax analysis along with the dictionary to set up the spec list for the processing routine. Subroutines attached to the meanings of words or idioms are executed. These

subroutines deal with either the meaning of the word itself or another word whose meaning is somehow affected by the present word. Attribute–value pairs then may be consolidated as necessary. For example, "Team =?" and "Team(winning) = (blank)" are collapsed into "Team(winning) =?"

The processor determines, using the specification list, what part of the stored data is relevant for answering the input question. Output from the processor is the answer in the form of a list structure. The main task of the processor is, given a spec list, to find a match on each path of the given data structure for all the attribute–value pairs of the spec list. Matching is not always simple. Derived attributes are computed before values are matched. The output becomes the *found list*; no attempt is made to print grammatical English sentences.

"Infential Memory as the Basis of Machines Which Understand Natural Language" is the title of Lindsay's (71) work in which he describes SAD SAM. This program uses a subset of English (known as Basic English, it is limited to simple constructions and a vocabulary of around 1700 words) and answers questions about family kinship relations by making inferences. Two list structures are constructed in memory for SAD SAM. The first is similar to high school English diagrams, and the second is the family tree. The implementation consists of the sentence-parsing program and the semantic analysis program. The parsing program utilizes phrase structure grammar rules as outlined by Chomsky 72.

While it is clear that in general different sequences of rewriting rules produce different sentences, it is also clear that they may produce the same sentences. SAD SAM makes two assumptions:

- It is assumed that almost all sentences encountered in actual text may be parsed by a procedure that works from left to right, making decisions about the disposition of earlier phrases without considering the entire sentence (in a limited sense this idea is employed by Winograd 73 for Lindsay it reduces the number of rewriting rule combinations which must be searched).
- It is assumed that a very limited amount of memory is available to remember intermediate results during the parsing of even extremely long sentences. This severely restricts the complexity of sentences that will be handled.
- The parser of a sentence associatively organizes memory into a structure reflecting interrelations among words in the sentence. How the inferential machinery works in parsing is best shown by example (71):

The first word is "the." Now I need to find a nounlike word. The second word is "very," so now I need an adjective or adverb. The third word is "big," which is the adjective I needed, so combine these two words into the structure "very big." Now I need a nounlike word. The fourth word is "man," which is the noun I needed. Now all words are combined into the structure "(the ((very) big)) man." But now we have a subject, so look for a verb. The fifth word is "bit" which can act as the verb, so create the structure "((the ((very) big)) man) bit." Now another nounlike structure could serve as object. The sixth word is "the," so save it to modify a nounlike word; now we have two things saved, both looking for a nounlike word. The final word is "dog" which will serve both needed functions. We now have the complete sentence, ...

After the sentence is parsed, the semantic analyzer considers the meanings of the words. The parse tree is used not as a syntactic entity alone, but also to relate words. Subject–object combinations linked by a form of the verb "to be" are marked as equivalent, and their modifiers grouped together. SAD SAM next searches for any of the eight allowable kinship relations—father, mother, brother, sister, offspring, married, brother-in-law, and sister-in-law. Any proper name possessive adjective found modifying one of these words is paired with all others associated with the same occurrence of the relation word. From the elementary relations found, a family tree is constructed in an associative manner. The family tree, or trees, continue to grow as additional information is input, as shown:

Family Unit 1

Family Unit 3

Family Unit 2
HUSBAND = A
HUSBAND = E
HUSBAND = C
WIFE = B
WIFE = F
WIFE = D
OFFSPRING = Husband,
OFFSPRING = ?
OFFSPRING = Wife,

Family Unit 3
HUSBAND'S PARENTS = ?
HUSBAND'S PARENTS = Family Unit 1
HUSBAND'S PARENTS = ?
WIFE'S PARENTS = ?
WIFE'S PARENTS = Family Unit 2
WIFE'S PARENTS = ?

The diagram shows the structure storing definite implications implicity; however SAD SAM also stores possible implications explicitly to solve the problem of connotative meaning. The offspring of family unit 1 might include the wife of family unit 4 or the like; this is stored explicitly, an example of an implication. At the very least, SAD SAM deals with the problem of English comprehension and understanding through the construction of data structures from input text and then extracts semantic implications from them.

A number of other early works must be mentioned here, if only in passing, since they are at least historically are important.

The *DEACON* (Direct English Access and CONtrol) breadboard, written by F. B. Thompson of General Electric along with J. A. Craig, is a data-based question answerer that is part of a man–machine communication system. It uses a list-structured data base.

NAMER is a system that was developed by R. F. Simmons and D. Londe of the Systems Development Corporation. It is a graphical data-based system that generates natural language sentences from line drawings displayed on a matrix. Pattern recognition aspects of NAMER were derived from Vossler and Uhr's work.

Cheatham and Warshall 74 developed a retrieval language called QUERY. In their work they describe a grammar and a procedure for translating requests formulated in English-like prose into a program for searching a well-structured database.

Bohnert (75) designed project LOGOS. Its objective is the approximation of the logic of English grammar, and it resulted in *LAMB* (Language for an Automated Marriage Bureau).

Kirsch (76) has tent support to the notion of inference making (71) with the Picture Language Machine. Simmons, Klein, and McConlogue (77) developed a system named PROTOSYNTHEX that attempts to answer questions from an encyclopedia.

The Automatic Language Analyzer (*ALA*) came from Indiana University. F. W. Householder, J. Lyons, and J. P. Thorne were responsible for progress on this rather complicated language analysis system. The *ALA* was designed to handle the breadth and complexity of language found in a book on astronomy. It is a text-based system that translates English to intermediate forms.

The General Inquirer of P. J. Stone, R. F. Bales, J. Z. Namerwirth, and D. M. Ogilvie is another text-based system useful for analyzing the content of text. The most interesting features of the General Inquirer include its dictionary and thesaurus operation.

Many sophistications have been added to question-answering systems since these early systems were devised. The complexity of the Protosynthex programs stand out, along with the inherent elegance of Lindsay's inferential SAD SAM program, as truly significant in this early period of question-answering development. Though it is true that the other systems mentioned contributed a great deal to the understanding of natural language as we know it today, the two systems cited above stand a cut above the rest. The far-reaching implications of Lindsay's system seem to have influenced the direction of work today.

The Next Family of Question-Answering Systems. The question-answering systems described in this section are characterized by the problem of representation of knowledge, however acquired, and by the related problem of breaking through the formality and narrowness of the older systems. Heuristic search efficiency remains a problem but serves more as a constraint.

The development of a computer system that "understands," has certain cognitive abilities, and exhibits some humanlike conversational behavior is realized in *SIR* (Semantic Information Retrieval) of Raphael (78). *SIR* is a prototype of an "understanding" machine demonstrating how conversational and deductive abilities can be obtained through the use of a suitable model. The following facts and assumptions are basic to *SIR*:

- Understanding can be demonstrated by dialog.
- The most important prerequisite for the ability to understand is a suitable internal representation.
- A machine that understands must be able to recognize logical implications as to which facts are relevant to a particular question.

SIR is semantic in the sense that the actual information extracted from text and stored by the program is intended to approximate the linguistic "semantic content" or "meaning" of the material, and the computer representation of information used in *SIR* is derived from the "semantic" model structures of formal mathematical logic. Raphael is concerned with what he terms descriptive semantics, an empirical search for rules governing truth and meaningfulness of sentences in natural language, rather than pure semantics, dealing with properties of artificially constructed formal systems with respect to rules for sentence formation.

SIR uses a data structure called a *model* of an entity x, with the following properties: (1) certain features of the model correspond in some well-defined way to x; (2) changes in the model represent changes in x in some well-defined way; and (3) there is some distinct advantage to studying the model rather than x. Here x may represent a wide class of entities, such as objects, English statements, or mathematical concepts. The *SIR* model is dynamic and semantic, and is the collection of data to which the *SIR* programs can refer in the course of question answering. The model consists of words associated with each other through particular relations. A descriptor list represents associations, is a sequence of a pair of elements, and is associated with a particular object (similarly to the attribute–value pairs used before). The formalization of the model used by *SIR* is that of a limited relational calculus, and it takes advantage of the property list feature in Lisp.

Since *SIR* communicates interactively and uses a relational model, it is faced with the problem of extracting relational information from natural language text. *SIR* is primarily concerned with the ability of a computer to store and utilize relational information in order to produce intelligent behavior; therefore it avoids the complexity of syntactic analysis (necessary for *NLP*) by limiting itself to a small number of fixed formats for sentences. *SIR* is limited to about 20 simple sentence formats. The following sample conversation from *SIR* illustrates its deductive capabilities, and also shows Raphael's model testing a sentence for consistency before

accepting it for data:

```
INPUT: Every boy is a person
FORMALIZATION: SETR (Boy,Person)
SIR: I understand

INPUT: A finger is part of a hand
FORMALIZATION: PARTR (Finger,Hand)
SIR: I understand

INPUT: Each person has two hands
SIR: The above sentence is ambiguous, please rephrase it

INPUT: There are two hands on each person
FORMALIZATION: PARTRN (Hand,Person,2)
SIR: I understand

INPUT: How many fingers does John have
SIR: The above sentence is ambiguous, but I assume (has)
     means (Has as parts). I don't know whether finger is a
     part of John.

INPUT: John is a boy
FORMALIZATION: SETR (John,Boy)
SIR: I understand

INPUT: How many fingers does John have
SIR: The above sentence is ambiguous, but I
     assume (has as parts). How many finger
     per hand

INPUT: Every hand has 5 fingers
FORMALIZATION: PARTRN (Finger,Hand,5)
SIR: The above sentence is ambiguous, but I
     assume (has) means (has as parts).
     I understand

INPUT: How many fingers does John have
SIR: The above sentence is ambiguous, but I
     assume (has) means (has as parts).
     The answer is 10

INPUT: Who is president of the United States
SIR: Statement form not recognized
```

STUDENT, a system devised by D. Bobrow (79), accepts a restricted subset of English that expressed a wide variety of algebra problems. Programmed in Lisp, STUDENT is based on a relational model that utilizes an expandable store of general knowledge to build a model of a situation. The method of discourse analysis is formal, depending only on the occurrence of morphemes as distinguishable elements, and not on the analyst's knowledge of the particular meaning of each morpheme. Bobrow assumes that a listener understands

a discourse by transforming it into an equivalent sequence of kernel sentences, that is, sentences that one can understand directly.

The model used by Bobrow satisfies the following properties that he believes must be contained in all models: (1) a set of n objects, $\{O\}$; (2) a set of n functions, $\{F\}$, a function being a mapping from ordered sets of n objects, called arguments of Fi, into the set of objects; (3) a set of n relations, $\{R\}$, a relation being a special type of object in the model consisting of a unique identifier and an ordered set of n conditions for the relation; (4) a set of propositions, $\{P\}$, being either elementary or complex, that is, either a label associated with some relation and an ordered set of n objects satisfying the relation, or logical combination's of elementary relations; and (5) a set of semantic deductive rules, that is, rules giving procedures for adding new propositions to the model based on the propositions now in the model.

STUDENT operates as follows: It is asked to solve a particular problem. We assume that all necessary global information has been stored previously. STUDENT will now transform the English input statement of this problem into expressions in its limited deductive model, and through appropriate deductive procedures attempt to find a solution. More specifically, STUDENT finds the kernel of sentences of the input discourse and transforms this sequence of kernels into a set of simulataneous equations, keeping a list of the answers required, a list of the units involved in the problem (e.g. dollars, pounds), and a list of all the variables (simple names) in the equations. Then STUDENT invokes the SOLVE program to solve this set of equations for the desired unknowns. If a solution is found, STUDENT prints the values of the unknowns requested in a fixed format, substituting in "(variable IS value)" the appropriate phrases for variable and value. If a solution cannot be found, various heuristics are used to identify the variables (i.e., find two slightly different phrases that refer to the same object in the model). If two variables, A and B, are identified, the equation $A = B$ is added to the set of equations. In addition, the store of global information is searched to find any equations that may be useful in finding the solution to this problem. STUDENT prints out any assumptions it makes about the identity of the two variables, and also any equations that it retrieves because it thinks they may be relevant. If the use of global equations or of equations from identifications leads to a solution, the answers are printed out in the format described above.

Let us follow through an example solved by STUDENT:

(THE PROBLEM TO BE SOLVED IS)
(IF THE NUMBER OF CUSTOMERS TOM GETS
IS TWICE THE SQUARE OF 20 PERCENT OF
THE NUMBER OF ADVERTISEMENTS HE RUNS,
AND THE NUMBER OF ADVERTISEMENTS
HE RUNS IS 45, WHAT IS THE NUMBER OF
CUSTOMERS TOM GETS Q'.)

The first step in the transformation is to make all mandatory substitutions:

(WITH MANDATORY SUBSTITUTIONS THE PROBLEM IS)
(IF THE NUMBER OF CUSTOMERS TOM GETS
IS 2 TIMES THE SQUARE 20 PERCENT OF
THE NUMBER OF ADVERTISEMENTS HE RUNS,
AND THE NUMBER OF ADVERTISEMENTS
HE RUNS IS 45, WHAT IS THE NUMBER OF
CUSTOMERS TOM GETS Q.)

Words are then tagged by their function:

```
(WITH WORDS TAGGED BY THEIR FUNCTION THE
    PROBLEM IS) (IF THE NUMBER (OF/OP) CUSTOMERS
      TOM (GETS/VERB) IS 2 (TIMES/OP 1) THE
    (SQUARE/ OP 1) 20 (PERCENT/OP 2)
      (OF/OP) THE NUMBER (OF/OP) ADVERTISEMENTS
    (HE/PRO) RUNS, AND THE NUMBER (OF/OP)
      ADVERTISEMENTS (HE/PRO) RUNS IS 45,
    (WHAT/QMARK) IS THE NUMBER (OF/OP)
      CUSTOMERS TOM (GETS/VERB) (QMARK/DLM))
```

Using two inverse syntactic transformations, the problem statement is resolved into simple kernel sentences:

```
(THE SIMPLE SENTENCES ARE)
    (THE NUMBER (OF/OP) CUSTOMERS TOM
      (GET/VERB) IS 2 (TIMES/OP 1) THE (SQUARE/
    OP1) 25 (PERCENT/OP 2) (OF/OP) THE NUMBER
      (OF/OP) ADVERTISEMENTS (HE/PRO)
    RUNS (PERIOD/DLM).(THE NUMBER (OF/OP)
      ADVETISEMENTS (HE/PRO) RUNS IS 45
    (PERIOD/DLM)).
    ((WHAT/QWORD)IS THE NUMBER (OF/OP)
      CUSTOMERS TOM (GETS/VERB) (QMARK/DLM)).
```

The transformations from simple kernel sentences to equations uses three levels of precedence for operators, represented by (OP), (OP1), and (OP2). Finally the (list form of) equations shown below are solved:

```
(THE EQUATIONS TO BE SOLVED ARE)
    (EQUAL X00001 (NUMBER OF
      CUSTOMERS TOM (GETS/VERB)))
    (EQUAL (NUMBER OF ADVERTISEMENTS
      (HE/PRO) RUNS 45)
    (EQUAL (NUMBER OF CUSTOMERS TOM
      (GETS/VERB)) (TIMES 2 (EXPT
    (TIMES 2000 (NUMBER OF ADVERTISEMENTS
      (HE/PRO) RUNS)) 2)))
    (THE NUMBER OF CUSTOMERS TOM GETS IS 162)
```

In some cases, whenever necessary information is missing from the store of information, or variables which name the same object cannot be identified by the heuristics of the program, STUDENT turns to the questioner for help; for example: (DO YOU KNOW ANY MORE RELATIONSHIPS BETWEEN THESE VARIABLES).

Although the system is limited in some ways, it is sufficiently versatile to solve a large number of high school algebra word problems. It significantly contributes to language processing in its heuristic approach to syntactic analysis, the use of background information, and its direct approach.

C. Green (80) shows how a question-answering system can be constructed using first-order logic as its language and a resolution-type theorem prover as its deductive mechanism in a program called QA3. This work is a follow-up of an earlier program reported by Raphael.

Chamiak (81) generalized and expanded techniques used by Bobrow in CARPS, a program that solves calculus word problems. He also uses a heuristic approach to the analysis of English sentences using pattern-operation rules for both syntactic and semantic operators. Charniak concludes however that an incremental left to right parse would be more efficient than pattern-operation rules that syntactically analyze the input text.

Early question-answering systems were not only handicapped by lack of adequate linguistic models but also were often written in low-level languages. Significant progress was made in the systems described in this section. Developments in high-level languages progressed, and syntactic processing became well understood. For certain well-defined subsets of English, operational semantic analysis progressed satisfactorily.

Significant weaknesses were still prominent, however, Systems were experimental in nature, small, and memory-bound. Inductive (inference) procedures had not been incorporated. Complexity grew out of bounds when systems tried to enlarge their vocabularies. The promise for practical question answerers was in the future.

The Boom of the Seventies. The importance of effective task representations in the design of programs intended to exhibit sophisticated behavior manifested itself in the systems and techniques developed for language comprehension in the 1970s. This is not to imply that the problems are solved or even that the approaches are correct, but that indeed some progress was made.

Kaplan 82 described the operation of an augmented recursive transition network parser and demonstrated the natural way in which perceptual strategies based on the results of psycholinguistic experimentation could be represented in the transition network notation. Kaplan did not propose a transition network model as a complete and sufficient representation of all aspects of language behavior, but aimed rather at simulating the syntactic analysis component of performance. The semantic and cognitive interpretation were ignored.

Kaplan concluded that the formal model based on transformational grammars was not intended to give accurate accounts of the psychological processes involved in the human use of language. Despite this, psycholinguists have used transformational theory because it provided the most intricate and compelling explanation at that time of basic linguistic intuitions.

The heart of the augmented recursive transition network is the familiar finite-state grammar consisting of a finite set of nodes (states) connected by labeled directed arcs representing allowable transitions. By adding a recursive control mechanism to the basic strategy, Kaplan avoided a well-known inadequacy of finite state grammars, in expressing statements about hierarchical structure. All states are given names, which are allowed as labels on arcs as well. When an arc with a state name is encountered and control is transferred to that labeled state, the name of the state at the head of the arc is saved on top of a pushdown store. When a final state is reached in this new part of the grammar, the stack is popped and a sentence is then accepted only if a final state, the end of an input string, and an empty stack are all reached concurrently. This finite state transition network with recursion is still inadequate for natural language (consider immediate constituent grammars with discontinuous constituents or cross-serial dependencies). Permitting sequences of actions and conditions on arcs provides additional resolution to give the network the generative power of a Turing machine. Actions provide facilities for building tree structures, while conditions furnish more sensitive controls on the admissibility of transitions.

Humans use a small number of perceptual strategies to process sentences. These strategies account in part for the perceptual complexity of sentences. They can be naturally represented in transition network grammars. Validation of transition network models depends on the correlation between experimentally observed

complexity and complexity measured in the transition network. The complexity of a sentence analyzed by a transition network is proportional to the number of transitions made or attempted during its analysis. Perceptual strategies, however, express generalizations that are not necessarily always true. Thus the strategies serve as heuristic guidelines to the listener and do not directly reflect his or her abstract appreciation of the structure of the language.

Concluding, Kaplan remarks that the augmented recursive transition network described is a natural medium for expressing and explaining a wide variety of facts about the psychological processes of sentence comprehension. One important issue not discussed is the role and representation of semantic information in sentence production.

Truly the most spectacular system put together at that time was that of Winograd 73. Incorporation of the concepts of human syntactic, semantic, and problem-solving abilities as well as their interactions in understanding natural language has been achieved by Winograd as by no one else to date. Believing that a computer should, as humans do, take advantage of available knowledge in solving problems and comprehending dialog, Winograd gave his system a detailed model of a particular domain. Knowledge is represented in the system procedurally and factually, as opposed to traditional methods (lists of patterns or rules), which adds flexibility, since syntax, semantics, and inferences may now be represented thus. Winograd had three goals in mind when he designed his system: (1) the practical desire to have a usable language-understanding system; (2) the attainment of better understanding of what language is and how it is put together; and (3) the understanding of intelligence and how to make it amenable to computers.

The system was written in Lisp and was organized as shown in Fig. 12. MONITOR is a small Lisp program that calls the basic parts of the system. INPUT accepts normal English input, performs morphemic analysis, modifying the dictionary definitions accordingly, and returns a string of words with their definitions to be used as input to GRAMMAR, the main coordinator of the language understanding process. GRAMMAR handles noun clauses, noun groups, prepositional groups, and so on, and is written in PROGRAMMAR. SEMANTICS is a collection of Lisp programs that work with GRAMMAR to interpret sentences. ANSWER contls the response of the system, and also remembers discourse for future reference. PROGRAMMAR is a parsing system that interprets grammars written in the form of programs. DICTIONARY consists of two parts: a set of syntactic features associated with each word, and a semantic definition for each word. SEMANTIC FEATURES is a network of property lists used for the initial phase of semantic analysis. BLOCKS is a collection of PLANNER theorems that contain the system's knowledge about the properties of a particular world. MOVER is a set of display routines that simulate the behavior of SHRDLU (the "robot" and the environment. PLANNER is a deductive system used at all stages of analysis. DATA consists of facts about the current scene in the form of PLANNER assertions describing objects.

SHRDLU's vocabulary consists of a few hundred words relevant to the robot's world. Within this narrow framework, the system's capacity for engaging in intelligible discourse is remarkable. SHRDLU accepts virtually any reasonable imperative or interrogative sentences (including "why . . .," "how . . .," "what . . .," "where . . .," and "when . . ." questions, as well as questions requiring a yes or no answer), and some declarative sentences (including simple word definitions and e user's "likes" and "dislikes"). SHRDLU's capacity for using context to aid syntactic and semantic analysis extends not only to words within the same sentence, but to other sentences and to the entire discourse, enabling comprehension of such sentences as "Why did you do it?"

Design features included:

- *Systemic Grammar*.Winograd felt that traditional syntactic analysis using formal grammars was inappropriate for finding meaning-related structural units in utterances. He chose instead to work with systemic grammar, a nonformal grammar proposed by Halliday. In this grammar the only syntactic types are the clause, the noun group, the verb group, the adjective group, the preposition group, and 18 word types. A group may contain a large number of words and may be modified by a variety of syntactic features. For example, a noun group might have the features DET, DEF, OBJ, PREPOBJ, NS, indicating that it contains

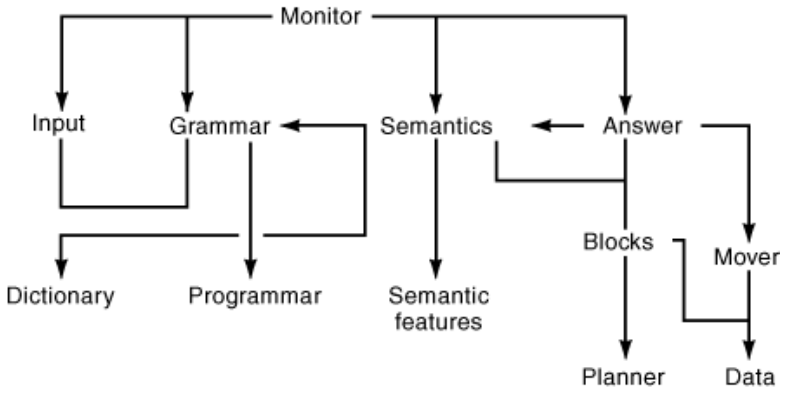

Fig. 12. A System written in Lisp by Winograd.

a determiner that is definite, that it serves as object of a preposition, and that its number is singular. Essentially syntactic analysis consists of attaching features to syntactic units (or deleting them) and using constraints on how these features may combine to steer the search for additional units and features. The constraints are expressed by AND–OR trees of permissible feature combinations for each syntactic type (these trees are not actually stored anywhere—they are built into the parsing programs). A group of mutually exclusive features is called a *system*; hence the term *systemic grammar*. The parse trees in systemic grammar thus have few but complex nodes. The parsing is strictly top-down. That is, instead of first grouping words into noun groups (*NGs*) and verb groups (*VGs*), then seeking to form preposition groups, clauses, and higher-level noun groups (*NGs*) and verb groups (*VGs*), we start by looking for a top-level clause; this means invoking the CLAUSE program, which then looks for its constituents (basically a NG and a VG), and so on; eventually we are looking at a particular word, expecting it to be of a particular type; if it is not, something else is tried (as will be seen, failure is program-controlled). Whenever a new unit is successfully parsed, a new node is put on the parse tree. Failure may lead to deletions of nodes. Concurrently with the formation of the syntactic structure, parts of a semantic structure are formed. As soon as the main part of any NG (i.e., a NG minus its qualifying clauses or preposition groups) has been parsed an *object semantic structure* (*OSS*) is built for it; such *OSSs* generally describe physical entities of some sort. The partial structure is later expanded to include the qualifiers. First, however, as soon as any relative clause or preposition group has been parsed, a *relation semantic structure* (*RSS*) is formed, using knowledge about the kinds of semantic subjects and objects particular verbs and prepositions require. *RSSs* describe events or relations between things or events. Finally, a *RSS* is formed for the complete clause (sentence) on the basis of the main verb (plus its modifiers) and the top-level *NGs*, or rather their *OSSs*. This final *RSS* is not yet the desired PLANNER program, which is the end product of the syntactic and semantic analysis process. However, it contains all the places required in that program, for example, all the THGOAL statements.

- *Procedural Embedding* As already indicated, syntactic units are defined by programs that look for them. The language PROGRAMMAR is used to write these procedural definitions. Thus syntactic knowledge has been procedurally embedded. In addition, the meaning structure for an input utterance is a PLANNER program. Knowledge about the "world" consists of the PLANNER database and theorem base. The database, which describes the current state or hypothetical states of the simulated toy world, is the only nonprocedural part of SHRDLU's knowledge. It properly corresponds to the subsystem called the "knowledge base" in our general schema. The theorems, which express know-how about the world and SHRDLU's relation to it, belong with the "routines to carry out the intent of an utterance"; they participate in the execution of the PLANNER program, which is the end product of semantic analysis.

- *Heterarchy* As has already been seen, semantic analysis begins before syntactic analysis has been completed. In fact, the semantic routines are called by the syntactic routines. The importance of this is that semantic incongruitles found early in the parsing process can quickly redirect false starts. For example, at one point in the parsing of "the wicked support injustice," an attempt would be made to parse "the wicked support" as a NG. This would quickly be rejected by the *semantic noun group specialist* SMNG1 (invoked by the NG), which notes that the semantic definition of "wicked" has the marker "animate," while "support," as a noun, has the marker "inanimate" (say). Winograd also states that inference routines may be called during the parsing process, presumably to deal with problematical sentences such as "I saw the Grand Canyon flying to New York."
- *PROGRAMMAR.* The parsing language PROGRAMMAR is discussed in some detail below, since it is the tool that enabled Winograd to put Halliday's very complex grammar into usable form.

Winograd's program endorses the procedural meaning representation for knowledge. In this way factual and heuristic information are combined to promote more efficient (w.r.t. processing time) use of the factual information. Nevertheless, the visual suggestiveness inherent in network-oriented representations, which leads to efficient processing algorithms, is lost. More importantly, the procedural embedding of knowledge runs into difficulties when contexts shift. Different heuristics and processing methods are then appropriate. Winograd's system combines many fascinating techniques to produce spectacular results. In his system, semantic analysis takes place before syntactic processing is complete. If semantic analysis fails for the structure built thus far, a backtrack mechanism allows the parsing to be redirected along more promising lines. This was the first true integration of syntactic and semantic analyses with good results.

PROGRAMMAR, a language for writing grammars, was developed in order to put systemic grammar into a usable form. Winograd found systemic grammar more amenable to finding meaning-related structural units in natural language than the traditional syntactic use of formal grammars (e.g., phrase structure and transformational grammars). Systemic grammar is a nonformal grammar that produces parse trees with few, but complex, nodes.

Criticisms leveled at Winograd's system include the following. His semantics are tied to the simple referential blocks world, without a method to make them extensible to any general real-world situation. Winograd's system would be unable to decide correctly between alternative word meanings in any given context. It is difficult to set up new "microworlds" in which understanding would take place, even if one were so inclined. The use of MICROPLANNER as a deductive mechanism can be criticized because of MICROPLANNER's use of blind backtracking to handle errors (note that no better working implementation of a deductive system was available then). Finally, the use of procedures as meaning representations is acceptable in some cases but awkward in many cases where a "static" representation would be sufficient.

While Winograd is very much a "heuristic programmer," Schank (83) saw himself as doing cognitive simulation, that is, modeling the mind in some sense. Thus, he is more interested in constructing a viable theory of language processing and meaning representation than in building high-performance programs. He has, however, built paraphrase programs embodying his theories.

Schank's basic contentions are:

- The central problem is to discover how meaning should be represented internally. The problem of translating from English to appropriate meaning representations cannot be posed until this has been done.
- Many of the ideas that seem to be conveyed by an utterance are actually inferred by the listener. Thus an essential part of a theory of language processing must be a theory of how such discourse inferences are made.
- We should seek simple conceptual building blocks from which to construct meaning representations. We should not assume that there is a one-to-one correspondence between these building blocks and English words. In particular, reasoning about actions would be phenomenally complex if we used as many action

primitives as there are English verbs. Schank proposes 14 primitive building blocks for expressing everyday actions.

- A guiding principle in the semantic analysis of language should be that two utterances of identical meaning—however different they may be in structure—should be mapped into identical meaning representations. The fact that most ideas can in fact be expressed in many ways, of widely varying syntactic structure, indicates that syntactic analysis is of little value in extracting the meaning of an utterance. Instead, knowing the kinds of semantic structures we are looking for, we should seek "fillers" for the "slots" in the semantic structures.

Schank's conceptual dependency theory is the basis of a system developed over about six years, culminating in the MARGIE system (84). It is a system rich in semantic representation, designed to provide a representation of meaning in terms of which paraphrase, inference, and *MT* could be carried out. Schank has succeeded to some extent with the first of these two aims. MARGIE operates in two modes: the paraphrase mode and the inference mode. Following is a sample output from those two modes:Inference mode:

INPUT: John tol Mary that Bill wants a book.
OUTPUT: A book about what?
Mary knows that Bill wants a book.
Bill wants to come to have a book.
Bill wants someone to cease to have a book.
Bill wants to read a book.

Paraphrase mode:

INPUT: John killed Mary by choking Mary.
OUTPUT: John strangled Mary.
John choked Mary and she died
beacuse she could not breathe.
Mary died beacuse she was unable
to inhale some air and she was unable to
inhale some air because John
grabbed her neck.

Schank uses a graphical notation consisting of items from four conceptual categories to represent meaning structures. These categories are picture producers (*PP*), picture aiders (*PA*), action aiders (*AA*), and actions (ACTs), and they correspond closely to nouns, adjectives, adverbs, and verbs, respectively.

The smallest structural unit that Schank deals with is the *conceptualization*: a graphical structure that links together conceptual categories using a variety of symbols, conceptual tense markers conceptual cases [analogous to and certainly influenced by the linguistic case structure of Fillmore (58)], and primitive actions. Schank uses four cases that serve as subgraphs in conceptualizations. These cases include the objective case, which relates an objective *PP* to an ACT; the recipient case, which relates a donor *PP* and a recipient *PP* to an ACT; the directive case, which relates direction (to or from) to an ACT; and the instrumental case, which links conceptualizations instrumental to an ACT to a conceptualization containing the ACT. In addition to conceptual cases, Schank makes use of only fourteen primitive actions through which he expresses all other actions. These

primitive actions are PROPEL, MOVE, INGEST, EXPEL, GRASP, PTRANS, MTRANS, ATRANS, SMELL, LOOK-AT, LISTEN-TO, CONC, and MBUILD. Forgoing further discussion of Schank's system, we remark that it can be considered as one of the very few natural language processing systems that may fall into the category of understanding systems.

The final system we discuss might also fall into this category: Wilks's (85) preference semantics system. Wilks's system, like Schank's, has a uniform representation in terms of structures of primitives for representing the meaning content of natural language. Unlike Schank, Wilks has concentrated on *MT*, from English to French, of small input paragraphs, and he has reported reasonably good translation. His system does not operate in paraphrase or inference modes. It makes use of formulas, one for each meaning (sense) of a word. These formulas are based on the (binary)ecomposition trees developed by Lakoff 86. The formula is a tree structure of semantic primitives interpreted formally using dependency relations. A typical formula, for the action of *drinking*, is as follows:

((ANI SUBJ)((FLOW STUFF) OBJ)(SELF IN)
((THI(ANI(THRU PART))) TO)(MOVE CAUSE))

The rightmost element is called the *head*. Template structures that actually represent sentences are built up as networks of formulas. These templates always contain an agent node, an action node, and an object node, and may include other nodes, which may depend on these three formulas. Formulas dictate how other places in the template should be filled. Thus "drink" would prefer a FLOW STUFF as object and an ANIM as subject. "Prefer" is the correct word to use, since if either a non-ANIM subject or a non-FLOW STUFF object is the only choice available, the utterance will still be understood (metaphorically). The template finally established for a fragment of text is the one in which most formulas have their preferences satisfied. This very simple device is able to do most of the work of a syntax and word sense ambiguity-resolving program.

After the agent–action–object templates have been set up locally for fragments of input text, Wilks's system attempts to tie these templates together to provide an overall meaning structure for the input. To accomplish this, Wilks makes use of *paraplates* attached to formulas for English prepositions. These paraplates range across two (not necessarily contiguous) templates. Thus far the structure of mutually connected templates comprises a semantic block in this basic mode. Whenever sentences cannot be successfully resolved into a semantic block in the basic mode, Wilks employs another mode, the *extended* mode, which makes use of commonsense inference rules, attempting, by a simple strategy, to construct the shortest possible chain of rule-linked template forms from previous text containing one of its possible referents. This chain then represents the solution to the ambiguity problem. After constructing a semantic block, French generation proceeds by "unwrapping" the block. There is no deepening of the representation by the generation routines.

The conceptual dependency approach of Schank and the preference semantics approach of Wilks exemplify what we believe to be the correct approach to representing the conceptual content of natural language utterances in terms of meaning structures. Specific criticisms of these approaches have been given elsewhere (87). Nevertheless, these two related approaches have the following desirable features regarding knowledge representation. The meaning structures corresponding to natural language utterances are formed according to simple structural rules. Powerful heuristic criteria, based on the central role of verbs and on preferred semantic categories for the subjects and objects of verbs, take on a "slot and filler" nature. Syntax is deemphasized in favor of meaning analysis. Paraphrase relations are made clearer. Similarity relations are made clearer. Inferences that are true of various classes of verbs can be treated as coming from the individual (primitive) ACTs. The inferences come from ACTs and states rather than from words. And finally, the organization in memory is simplified because much information need not be duplicated. The primitive ACTs provide focal points under which information is organized.

We now turn our attention forward about 20 years to the activities that are claiming significant current interest.

Empirical, Corpus-Based Approaches to NaturalLanguage Processing. Information retrieval systems have been around for as long as natural systems. Contemporary information retrieval systems have moved to a vector space model in which every list of words in both the document and the query is treated as a vector in an n-dimensional space (n being the number of symbols in the document collection). Thus a three-word query such as "water meter cover" would be treated as a vector with a value of 1 for these three terms and a value of 0 for all other terms. Documents are then found by comparing this vector with all others in the collection and reporting ones that match or are close. This more contemporary method exhibits flexibility, since documents are ranked by their distance to the query.

The use of *NLP* in information retrieval is still novel, but already some natural language tools are appearing in information retrieval applications (morphological analyzers, word sense disambiguators, etc.). *NLP* has proven successful in text categorization, summarization, and extracting data from text. Since the categories are fixed, text categorization appears a better candidate for *NLP* techniques than information retrieval, where categories are not fixed.

Perhaps due to the tremendous interest in the WWW, there has been a revival of empirical and non-linguistic methods of *NLP*. Essentially these methods use machine learning techniques to extract linguistic knowledge automatically from natural language corpora for *NLP* applications. All of the efforts described in the previous section required the system designer to encode requisite knowledge manually (albeit Winograd's system could deduce new knowledge once a core of facts was predefined). The reliance on manual encoding of domain knowledge gave rise to the term *knowledge engineering*. This made these earlier systems quite *brittle*: they simply did not function well outside of the domain of discourse for which they were designed.

Prior to the influence Chomsky (72) exerted on linguistics, arguing that language was largely innate and could not be learned simply from data (natural language corpora), there was significant activity in the use of primarily statistical methods to determine the structure of languages. Such distributional information permitted clustering of words and phrases from which much could be learned about a language.

In the late 1980s, owing in part to the success of statistical methods applied to speech understanding, researchers in natural language understanding began to use similar methods, initially to recover original data from noisy data, then to perform part-of-speech tagging (useful as preprocessing for parsing systems) and thence parsing, speech synthesis, intelligent information retrieval, and *MT*. Conferences have sprung up as a mechanism for researchers to exchange ideas, and repositories have been developed on methods, corpora, and so on for researchers to share. An excellent overview article on empirical *NLP* was written recently by Brill and Mooney (88). More recently, other facets of *NLP* cessing have been approached by empirical, corpus-based learning methods, including semantic analysis (i.e., word sense disambiguation and semantic parsing), see Ng and Zelle (89) for an overview.

In addition to providing an excellent overview, those authors discuss CHILL, Zelle's 90 empirical *NLP* system. CHILL takes as input a set of training instances consisting of sentences paired with the desired queries. The output is a shift–reduce parser in Prolog that maps sentences into queries. CHILL was demonstrated on a US geograph, database encoded as a set of facts in Prolog. The query language is a logical form that is directly executable by a query interpreter, which subsequently receives appropriate answers from the database. For example, for the query "show me the morning flights from Boston to Denver," CHILL produces a parser which turn the sentence into the logical form

answer(F, time(F,t), flight(F), morning(T), origin(F,O),
equal(O, city(Boston)), destination(F,D), equal(D, city(Denver))

This form is directly interpretable in Prolog, and the general operators are directly inferable from the representations themselves.

Another interesting modern question-answering system is Hammond's FAQFinder program (91, 92). According to Hammond's Web page, FAQFinder is an automated question-answering system that uses the files of "frequently asked questions" (*FAQs*) associated with many Usenet newsgroups. These files are compendiums of the accumulated wisdom of the newsgroup on topics that are of frequent interest. FAQFinder will take a user's query on any topic, attempt to find the *FAQ* file most likely to yield an answer, search within that file for similar questions, and return the corresponding answers.

Unlike artificial intelligence question-answering systems that focus on the generation of new answers, FAQFinder retrieves existing answers found in *FAQ* files. Unlike information retrieval approaches that rely on a purely lexical metric of similarity between query and document, FAQFinder uses a semantic knowledge base (WordNet) to improve its ability to match question and answer. FAQFinder is based on four assumptions about *FAQ* files: (1) all of the information in a *FAQ* file is organized in question–answer (*Q&A*) format; (2) all of the information needed to determine the relevance of a *Q&A* pair can be found within that pair; (3) the question part of the *Q&A* pair is the most relevant for determining the match to a user's question; and (4) broad, shallow knowledge of language is sufficient for question matching. Essentially these assumptions lead to a case-based solution (view) of the *FAQ* retrieval problem.

Hammond's system is available on the WWW, and an example of using this extremely fast system follows. From an input configuration, shown in Fig. 13 with the input question "I want to schedule a trip to Japan." output in Fig. 14 was generated.

FAQFinder has demonstrated that when there is an existing collection of questions and answers, as found in the *FAQ* files, then question answering can be reduced to matching new questions against *Q&A* pairs. The authors rightly claim that this task is simpler than that of natural language understanding, but, as Figs. 13 and 14 illustrate, caution is nonetheless important in interpreting results.

Summary and Conclusions. Natural language understanding by computer is an important goal, and many uses of this technology have already been put to the test. In this article we have considered a number of concerns that must be confronted in the design of a sophisticated *NLI*, in building a machine translation system, and in constructing sophisticated question-answering systems. We argued the necessity of incorporating knowledge of the dain, knowledge of the user, and knowledge of discourse structure into the natural language system, and indicated many research efforts that investigate problems deriving from this necessity. Particular attention in the paper has been devoted to issues that arise when trying to represent these diverse kinds of knowledge.

Has research into these many concerns progressed to the point where it will actually be possible to begin to build the "ideal" natural language system? As knowledge representation ideas become more precisely formulated, such an evolution is happening. Work in the 1970s and 1980s took knowledge representation far from its ad hoc beginnings. General knowledge representation formalisms have been developed and honed. Investigations into fundamental knowledge representation principles have proceeded.

Despite these developments, however, the ability to incorporate knowledge is still a major source of difficulty confronting the designer of the ideal natural language system. There are a number of reasons for this. First, little of the work on knowledge representation is explicitly aimed at natural language understanding, and less yet at the problem of integrating knowledge into the interpretation processes. Second, much of the work is highly theoretical, and will require substantial reforlation to be applied. Finally, knowledge representation is a vast area of inquiry; current research has investigated only a relatively small subset of the potential issues lying in wait. New empirical, corpus-based methods appear promising, but are as yet incomplete.

It appears that the ideal natural language system is still some way off, at least in its full splendor. Nevertheless, the indicators are all positive. Almost every current research effort has dealt deeply with at least one or two issues of relevance to constructing the ideal system. The union of all such research covers a vast range of relevant issues. The next few years should see a distillation of the many techniques collectively provided by current and future research, and the slow development of more comprehensive theories. The effect on the sophistication of practical natural language systems will be incremental, but over the long run immense.

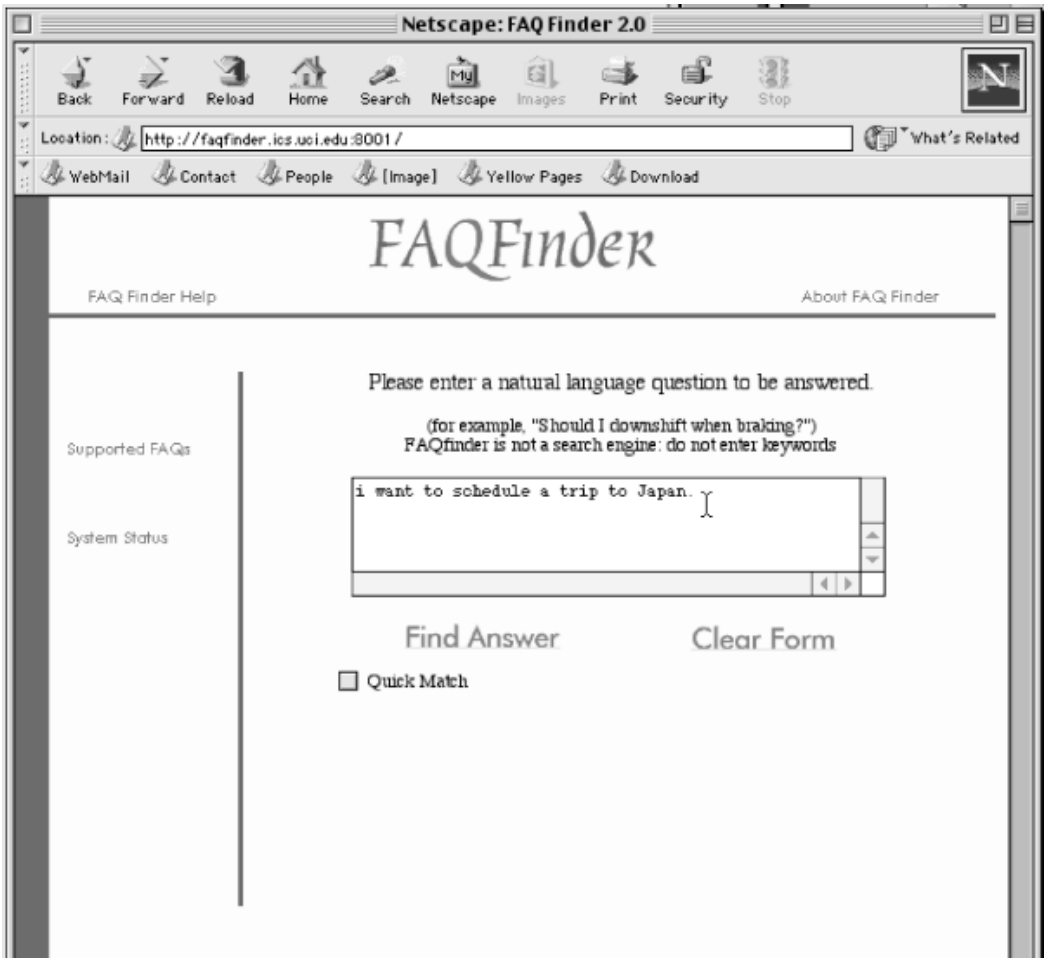

Fig. 13. FAQFinder screen with the input question "I want to schedule a trip to Japan."

The ideal natural language system may not be available for some time, but as we move into the future, closer and closer approximations to it should be.

Acknowledgments

The authors are members of the Institute for Robotics and Intelligent Systems (*IRIS*) and wish to acknowledge the support of the Networks of Centres of Excellence Program of the Government of Canada and the Natural Sciences and Engineering Research Council, as well as the participation of PRECARN Associates Inc. We are grateful to Canadian Cable Labs Fund for their financial assistance.

Appendix: Some Commercial Natural language Interfaces

The following is a list of commercial (*NLIs*) that either are well known or have long development histories or both. The final item in the list is not an *NLI* but an environment for constructing *NLIs*.

Fig. 14. FAQFinder screen with output for the input question "I want to schedule a trip to Japan."

- Natural Language, from Natural Language Inc., Berkeley, CA (415-841-3500), runs on Rdb, ingres, Oracle, Sybase, Informix and Sharebase, and works on VAX/VMS and Windows platforms. Another *NLI*—presumably an earlier one—is DataTalker, which "runs on Digital VAX, under VMS, ULTRIX and Berkeley UNIX 4.2, Sun-3 and Sun-4 workstations under UNIX, and the IBM PC/AT (and compatibles) equipped with a coprocessor board" (93, p. 5-14).
- Intellect, supplied by Al Corporation, Waltham, MA, is integrated within their knowledge-based system (*KBS*) shell product KBMS and runs against a number of database management systems (*DBMSs*). According to Miller and Walker (93, p. 5–2), Intellect "runs in IBM mainframe environments under MVS/TSO, VM/CMS and MVS/CICS." Intellect is based on Larry Harris's ROBOT system.
- *Q&A* from Symantec Corp., Cupertino, CA (408-253-9600), is based on the LIFER *NLI* developed by Gary Hendrix et al. *Q&A* can import and export data from "Lotus 1-2-3, dBASE II and III, Symphony pfs:File, IBM Filing Assistant and DIF and ASCII files" (93, p. 5-16). *Q&A* has an integrated word processor which "will work with documents from Lotus 1-2-3, pfs:Write, IBM Writing Assistant, WordStar, and ASCII files" (93). "Version 2.0 features include local area network support and tighter integration with Lotus 1-2-3. Several magazine awards have been given to this version." (93, p. 5-17).
- Parlance, from Bolt Beranek & Newman, Cambridge, MA (617-873-5412), is probably based on (and an improvement of) the LUNAR and IRUS systems, also developed at BBN by William Woods et al.
- EasyTalk, developed by Intelligent Business Systems of New Haven, CT. For more on this system see Steve Shwartz's (1987) book *Applied Natural Language Processing* (94). IBS appears to be an offspring of Roger Schank's company Cognitive Systems, also of New Haven, CT.
- IBM SAA LanguageAccess, developed at IBM's lab in Lidongo, Sweden, is written in Prolog, and permits NL queries against a relational DBMS. Based on TQA and other former IBM *NLIs*, its status as a product is unknown at this time. For more information, see Gregor Jonsson's paper in the 1991 Data Engineering conference (95).
- English Wizard, from Linguistic Technology Corporation, Acton, MA, is discussed in some depth below (96).
- ELFSoft has announced Access ELF, a shareware *NLI* for Microsoft Access.

- Language Craft, from the Carnegie Group, Pittsburgh, *PA*, is "an integrated open-architecture environment for constructing natural language interfaces to databases, operating systems, expert systems and conventional software applications. Language Craft combines a grammar building development environment with a domain-independent natural language interpreter and run-time module. Features include: PLUME run-time interpreter combining case–frame instantiation and pattern matching; Dialogue management for handling fragmentary input and ambiguities; Grammar Writer's Workbench containing a grammar editor, grammar compiler, tracer and stepper to debug grammars; and Domain translator. Language Craft can be adapted to a wide variety of applications. Users can develop their own domain-specific natural language interfaces without specialized Al or linguistic expertise" (93, pp. 5-3, 5-4).

97 list the following companies as *NLP* (text and speech) software developers: Al Corporation, ApTech, AT&T Bell Labs, *ALPS*, Battelle/NLQ, Bolt Beranek & Newman, Carnegie Group, Cognitive Systems, Dynamics Research Corp, Dragon Systems, Excalibur Technologies Corp, Gachot-Systran, GSI Erli, IBM, Intelligent Business Systems, Kurtzweil Applied Intelligence Inc., Lernout & Hauspie, Logos Corp., Natural Language Inc., Reference Software, Siemens Nixdorf, Smart Comm, Speech Systems, Symantec Corp.

Critique of English Wizard. English Wizard is a commercially available *NLI* to databases. The version reviewed, unless otherwise specified, is English Wizard 1.1.

The installation or setup time for *NLIs* is a major issue in their overall usability, as 98, creator of English Wizard, acknowledges. Harris says that the dictionary construction process must be fully automated—particularly in a personals computer setting where no technical support is available other than the user. Harris regards setup/installation as much more important than system coverage, which he calls "fluency":

Rather than trying to maximize fluency at all costs, taking the approach of maximizing fluency subject to the constraint of fully automatic dictionary construction is necessary. Since automating the setup process is an absolute requirement, the system must do the best it can, given this requirement. If the resulting level of fluency is comparable to the earlier systems (such as Intellect, a natural language database query tool introduced by Al Corp in 1981) without the setup time, then that would be sufficient.

English Wizard is installed from a dialog box called the Dictionary Construction Kit. No bugs were encountered during installation of our own trial application, which had 14 tables, 135 columns, and 646 column values that we deemed useful to represent with lexical entries. The installation process requires some manual intervention to decide which tables and columns to include. Lack of automatic join finding was a hindrance to automatic dictionary building, because English Wizard makes a few joins but the installer must manually create the majority of joins using the Link Editor. Once dictionary building is complete, the lexicon must be customized to the database it works against in order to make it usable. The same process is used to maintain the lexicon, that is, to add new vocabulary.

Users can review the configuration of the database: they can edit definitions of tables, columns, and values. English Wizard's root words are from its "root dictionary." *Ignored* words are those that can be omitted from a sentence without changing its meaning. Users can define their own ignored words, though the User Guide notes that "words should only be ignored as a last resort." English Wizard predefines 243 root or ignored words. Very few have user-accessible definitions, except for some mathematical ones. The ignored words are: about, also, an, at, cent, column, data, database, day, dollar, field, file, from, I, I'd, ignore, in, info, information, me, month, need, our, quarter, record, row, table, than, the, there, too, us, want, we, week, will, wonder, year, you, your. English Wizard predefines about 200 synonyms from its *thesaurus*. The *NLI* looks at the names of tables or columns or any word users define, consults its thesaurus, and supplies a large number of (largely unhelpful) synonyms, e.g., "English" → britain, british, england, englishman, great britain, uk, united kingdom; and "offering" → gave, giving, given, funding, handout, bestowal, appropriation. Predefined words and synonyms include US state names and their two-letter abbreviations (e.g., "MA"), which we have deleted for our (Canadian) application. Also predefined are "M" and "F" for male and female. Synonyms can be defined for tables, columns, values, root words, formulas, and groups. *Formulas* are words that represent a calculation and can involve the use of

columns, constants, other formulas, and English Wizard's built-in functions. For example, if the database stores "Salary," then "Monthly Salary" could be defined as "Salary/12". Groups are words that represent a group of columns, for example, "address" might be defined as the group of columns Street, City, State, and ZIP Code.

There are two phases to customizing a dictionary after the initial build. The first phase is to prune away unwanted synonyms supplied by English Wizard. The second phase is to define new vocabulary. Items to be pruned consist of strange pluralizations and singularizations of redefined words and synonyms, such as "Arkansa" from "Arkansas" "Texa" from "Texas." Users must specify irregular word infections, since English Wizard cannot anticipate irregular word endings: all irregular inflections of words must also be defined separately as synonyms for the words. For example, "bough" should be defined as a synonym for "buy" Defining new vocabulary consists in naming synonyms of predefined words and names of tables columns and column values; specifying ignored words, groups, and requests; and assigning verbs as names of joins. Despite the User Guide's claim that users can specify their own ignored words, we were unable to and could only make a word ignored by defining it as a synonym of another ignored word.

English Wizard has zero tolerance for lexical ambiguity. Three cases of its intolerance are as follows:

(1) A word can not be defined as both a column value and as something else, presumably because English Wizard cannot distinguish between the two uses. For example, "French" is predefined but also is part of various phrases that refer to column values.
(2) A word with a verb definition cannot also be given a noun definition. For example, "show" is predefined with a verb definition, so one cannot create a noun definition for it as well.
(3) A word cannot be defined as both a column and table name, again presumably because English Wizard cannot distinguish between the usages. For example, I wanted to name the column "Offering Type" as "offering" but was told by English Wizard that The term "offering" is already defined. Column names cannot conflict with table names or other columns in the same table." So we are forced to define "offering" to refer to the table, thereby forcing inclusion of theable in the querywhich can result in a significant inefficiency. For example, if we ask for the restricted offerings, the OFFERING table does not need to be consulted, yet if we define "offering" as referring to the table, then the table will be consulted in any case.

Problems were encountered when defining requests: a limit was encountered when trying to impose a hierarchy on top of a set of database column values. We have used requests to (a) give names to groups of column values, (b) request names of type (a) that refer to groups of column values and (c) request names of type (b) that refer to groups of request names. However, in certain cases, English Wizard says that we are trying to define an item in terms of itself. We were told by Mike Groch of Linguistic Technology that trying to define such a request hits a limit that is in 1.1 beta but should be eliminated from the final 1.1.

In a first quick test drive of English Wizard 1.0, we found a number of problems. The system was frozen by the query "the customers from bonn and washington state." An SQL error state was caused by the query "the employees who sell meat products." The following phrases led to the failure of a built-in function: "in the year of 1994" and "month of August." The query "the employees who reside in bonn" was given an incorrect interpretation in terms of a person with the last name Bonn (a person with last name Bonn is in the dictionary, and there is no information that Bonn is a place, so presumably the system defaults to the only dictionary entry for Bonn that it has). The following words were not in the dictionary: "made," "Bonn," "1994" (though "August 1994" is O.K.), "first" as in "in the first quarter," and "second" as in "in the second quarter." The test drive dictionary contained entries for "next April" and "next August."

English Wizard has an attractive-looking interface, but it is not very usable in practice and could certainly be improved. It would be nice to search on the definitions of words that represent column values. In this way, one could select out particular types of definitions (to check for overlaps, etc.). The search facility in the Dictionary Editor is limited and awkward to use. It is awkward to define requests using the Query Builder window. One

has to transfer Uncommon Words into Common Words, so that those words can be used in requests. [What are (Un)Common Words? Is this the only difficulty?]

The verdict: Customization is a big stumbling block for practical use of *NLIs*. Presently, building a dictionary map is hard, and creating lexical entries is hard because one has to decide whether a word properly refers to a table, a column, or a link. Thus the usefulness of English Wizard is questionable for nontrivial applications.

The main problem with the performance of English Wizard is its intolerance of lexical ambiguity. It will allow database ambiguity (i.e., a column appearing in several different tables). Such ambiguity is handled by presenting a window showing the different tables in which a column appears. This is again a reflection of its primitive join-finding ability. In almost all cases the system should be able to figure such ambiguities out for itself.

No bugs were found during installation. The process requires some manual intervention to decide which tables and columns to include. Lack of automatic join finding was a hindrance to automatic dictionary building, because the installer had to supply some joins manually. English Wizard makes a few joins, but the majority of joins must be manually created using the Link Editor. According to a mailing from Linguistic Technology, they have been surprised to find that customers want to hire consultants to help them set up English Wizard in their environments. Linguistic Technology thought they were designing an end-user tool. They have now set up a technical consulting program in which consultants are trained and accredited before being recommended to customers.

The reason for the intolerance of ambiguity is fairly obvious: English Wizard provides no mechanism for restricting the context of individual word usages, so any ambiguity has to be handled by presenting multiple alternatives, which would prove very annoying in the presence of much lexical ambiguity.

English Wizard is really geared towards small databases. Due to limitations in dictionary size and in join-finding capabilities, it is hard to make English Wizard work with large databases.

MultiList Translation System Languages

Cybertrans http://www.epiuse.co.za/translator/default.htm

GLOBALINKhttp://www.globalink.com/pages/product-pwtrans6.htmlInTransNet
http://www.intransnet.bc.ca/ intrans/intrae.htm

LOGOS http://www.logos-ca.com/ SYSTRAN http://www.systransoft.com/

Afrikaans–English, German–English, French–English, Portuguese–English, Spanish–English

French, German, Spanish, Italian, Portuguese, English, and more Japanese–English

English to German, French, Spanish, Italian, Japanese; German to English, French, Italian

French, German, Spanish, Portuguese, Italian, Japanese, Chinese, English, and more End-MultiLis

BIBLIOGRAPHY

1. E. S. Gardner *The Case of the Demure Defendant*, New York: Morrow Company, 1956.
2. J. Mylopoulos *et al.* TORUS: A step toward bridging the gap between databases and the casual user, *Inf. Syst.*, **2**: 49–64, 1976.
3. T. Strzalkowski ENGRA—yet another parser for English, TR83-10, LCCR, Burnaby, BC: Department of Computing Science, Simon Fraser University, 1983.
4. R. Hadley SHADOW: A natural language query analyser, *Comput. Math.* **11** (5): 481–504, 1985.
5. B. Webber *et al.* Extended natural language database interactions, in N. Cercone, (ed.), *Computational Linguistics*, London, Pergamon, 1983, pp. 233–244.
6. J. F. Allen C. R. Perrault Analyzing intention in utterances, *Artif. Intell.*, **15** (3): 143–178, 1980.

7. J. Wald Problems in query inference, Ph.D. Thesis, Department of Computational Science, University of Saskatchewan, Saskatoon, 1985.
8. E. F. Codd *et al.* RENDEZVOUS Version 1: *An experimental English language query formulation system for casual users of relational databases*, Res. Rep. No. RJ2144, San Jose, CA: IBM Research Laboratory, 1978.
9. N. Cercone G. McCalla Accessing knowledge through natural language, invited chapter for M. Yovits 25th Anniversary Issue *Advances in Computers* series, New York: Academic Press, 1986, pp. 1–99.
10. T. Strzalkowski N. Cercone A framework for computing extra sentential references, *Comput. Intell.*, **2** (4): 159–180, 1988.
11. H. P. Grice Logic and conversation, in P. Cole, J. *Morgan (ed.) Syntax and Semantics*, New York: Academic Press, 1975.
12. S. J. Kaplan Designing a portable natural language database query system, *ACM Trans. Database Syst.*, **9** (1): 1–19, 1984.
13. M. Kao N. Cercone, W. S. Luk Providing quality responses with natural language interfaces; The null value problem, *IEEE Trans. Softw. Eng.*, **14**: 959–984, 1988.
14. W. Luk S. Kloster ELFS: English language from SQL, *ACM Trans. Database Syst.*, **11** (4): 447–472, 1986.
15. M. Kao Turning null responses into quality responses, M.Sc. Thesis, School of Computing Science, Simon Fraser University, Burnaby, BC, 1986.
16. G. Hendrix *et al.* Developing a natural language interface to complex data, *ACM Trans. Database Syst.*, **3** (2): 105–147, 1978.
17. L. R. Harris Experiences with intellect: Artificial intelligence technology transfer, *AI Mag.*, **5** (2): 43–55, 1984.
18. N. Cercone *et al.* SystemX and DBLean: Easily getting more from your relational database, *Integ. Comput. Aided Eng.*, **1** (4), F. Golshani (ed.) pp. 311–339, 1994.
19. S. Petrick Natural language database query systems, IBM Res. Rep. RC 10508, 1984.
20. G. Malingaham Natural language access to internet search engines, M.Sc. Thesis, Department of Computer Science, University of Regina, Saskatchewan, 1997.
21. J. Newton (ed.) *Computers in Translation*, London: Routledge, 1992.
22. W. J. Hutchins H. L. Somers *An Introduction to Machine Translation*, London: Academic Press, 1992.
23. W. Goshawke I. D. K. Kelly, J. D. Wigg *Computer Translation of Natural Language*, Wilmslow, United Kingdom: SIGMA Press, 1987.
24. D. J. Arnolds *Machine Translation: An Introduction Guide*, London: Blackwell-NCC, 1994.
25. J. Slocum *Machine Translation System*, Cambridge, UK: Cambridge University Press, 1988.
26. M. A. Covington *Natural Language Processing for Prolog Programmers*, Englewood CI: RRS, NJ: Prentice-Hall, 1994.
27. *ALPAC Language Machines: Computers in Translation and Linguistics*, A report by the Automatic Language Processing Advisory Committee, Washington, DC: Division of Behavioral Sciences, National Academy of Sciences, National Research Council, 1966.
28. Y. Wilks *An Artificial Intelligence Approach to Machine Translation*, Report CS-264, Computer Science Department, Stanford University, 1972.
29. W. J. Hutchins *Machine Translation: Past, Present, Future*, Chichester, UK: Ellis Horwood, 1986.
30. W. J. Hutchin *Recent Developments in Machine Translation: A Review of the Last Five Years In New Directions in Machine Translation*, Proc. Budapest, Foris Publications, 1988.
31. K. Naruedomkul N. Cercone Steps toward accurate machine translation, *Proc. 7th Int. Conf. Theor. Methodol. Issues Mach. Transl.*, Santa Fe, NM, 1997, pp. 63–75.
32. H. Uchida ATLAS-II: A machine translation system using conceptual structure as an interlingua, *Proc. 2nd Mach. Transl. Summit*, Tokyo, 1989.
33. K. Muraki PIVOT: Two-phase machine translation system, *Proc. 2nd Mach. Transl. Summit*, Tokyo, 1989.
34. M. T. Rosetta *Compositional Translation*, Dordrecht, The Netherlands: Kluwer Academic Publishers, 1994.
35. T. Mitamura E. H. Nyberg 3rd I. G. Carbonell An efficient interlingua translation system for multi-lingual document production, *Proc. Mach. Transl. Summit III*, Washington, DC, 1991.
36. Center of the International Cooperation for Computerization, *Thai Basic Dictionary*, Tech. Rep. 6-CICC-MT55, Tokyo: Machine Translation System Laboratory, CICC, 1995.
37. D. Jones *Analogical Natural Language Processing*, London: UCL Press Limited, 1996.

38. P. F. Brown *et al.* Analysis, statistical transfer, and synthesis in machine translation, *Proc. 4th Int. Conf. Theor. Methodol. Issues Mach. Transl. Nat. Lang.*, Montreal, 1992, pp. 83–100.
39. L. Cranias, H. Papageorgiou, S. Piperidis *A Matching Technique in Example-Based Machine Translation*, The Computation and Language E-Print Archive, 1995. Available: http://xxx.lanl.gov/abs/cmp-lg/9508005
40. S. Nirenburg *et al. Machine Translation: A Knowledge-Based Approach*, San Mateo, CA: Morgan Kaufmann, 1992.
41. P. McFetridge *et al.* SystemX: A portable natural language interface, *7th Canadian Society for the Computational Studies of Intelligence/ Société canadienne pour l' étude de l' intelligence par ordinateur (CSCSI/SCEIO*), Edmonton, Alberta, 1988, pp. 30–38.
42. N. Cercone *et al.* Natural language interfaces: Introducing SystemX, in T. Oren (ed.), *Advances in Artificial Intelligence in Software Engineering*, Greenwich, CT: JAI Press, 1990, pp. 169– 250.
43. G. Hall Querying cyclic databases in natural language, M.Sc. Thesis, School of Computing Science, Simon Fraser University, Burnaby, BC, 1986.
44. C. Pollard I. Sag *Information-based Syntax and Semantics: Fundamentals*, Stanford, CA: Center for the Study of Language and Information, 1987.
45. C. Pollard I. Sag *Head Driven Phrase Structure Grammar*, Chicago: University of Chicago Press, 1994.
46. J. A. Wald P. G. Sorenson Resolving the query inference problem using Steiner trees, *ACM Trans. Database Syst. (ACM-TODS*), **9** (3): 348–368, 1984.
47. G. Hall *et al.* A solution to the MAP problem in natural language interface construction, *Int. Comput. Sci. Conf. '88.* Hong Kong, 1988, pp. 351–359.
48. J. Johnson Semantic relatedness, *Comput. Math. Appl.*, **29** (5): 51–64, 1995.
49. J. Johnson R. Rosenberg A measure of semantic relatedness for resolving ambiguities in natural language database requests, *Data Knowl. Eng.*, **7** (3): 201–225, 1992.
50. B. W. Ballard J. C. Lusth, N. L. Tinkham LDC-1: A transportable, knowledge-base natural language processor for office environments, *ACM Trans. Off. Inf. Syst.*, **3** (2): 1–25, 1985.
51. M. Bates R. Bobrow A. transportable natural language interface, *Proc. 6th Annu. Inte. SIGIR Conf. Res. Dev. Inf. Ret. ACM*, 1983.
52. D. E. Johnson Design of a portable natural language interface grammar, Tech. Rep. 10767. Yorkton Heights, NY: IBM Thomas J. Watson Research Laboratory, 1984.
53. J. A. Johnson R. S. Rosenberg A data managent strategy for transportable natural language interfaces, *Int. J. Intell. Syst.*, **10** (9): 771–808, 1995.
54. P. Martin *et al.* TEAM: An experimental transportable natural language interface, *Database Eng.*, **8** (3): 10–22, 1985.
55. C. D. Hafner K. Godden Portability of syntax and semantics in datalog, *ACM Trans. Off. Inf. Syst.*, **3** (2): 141–164, 1985.
56. S. Shieber *An Introduction to Unification-Based Approaches to Grammar*, Stanford, CA: Center for the Study of Language and Information, 1986.
57. F. Popowich *et al.* Processing complex noun phrases in a natural language interface to a statistical database, *Proc. 14th Int. Conf. Comput. Linguistics*, Nantes, France, pp. 46–52, 1992.
58. C. J. Filimore The case for case, in E. Bach and R. Harms (eds.), *Universals in Linguistic Theory*, New York: Holt, Rinehart & Winston, 1968, pp. 1–88.
59. G. Bouma Feature structures and nonmonotonicity, *Comput. Linguistics*, **18** (2): 183–203, 1992.
60. P. McFetridge N. Cercone Installing an *HPSG* parser in a modular natural language interface, *Computational Intelligence III*, Amsterdam: North Holland, 1991, pp. 169–178.
61. F. Popowich C. Vogel A logio-based implementation of head-driven phrase structure grammar, in C. Brown and G. Koch (eds.), *Natural Language Understanding and Logic Programming, III*, Amsterdam: Elsevier, North Holland 1991, pp. 227–245.
62. R. Jeffries *et al.* User interface evaluation in the real world: A comparison of four techniques, *ACM CHI '91, Conference on Human Factors in Computing Systems*, New Orleans, LA, 1991, pp. 119–124.
63. E. Guba Y. Lincoln *Fourth Generation Evaluation*, Newbury Park, CA: Sage, 1989.
64. N. Cercone *et al.* The systemX natural language interface: Design, implementation and evaluation, Tech. Rep. CSS-IS 93-03, Burnaby, BC: Centre for Systems Science, Simon Fraser University, 1993.
65. S. Petrick Field testing the Transformational Question Answering (*TQA*) system, *Proc. 19th Annu. Meet. Assoc. Comput. Linguistics*, Stanford, CA, 1981, pp. 35–36.

66. W. Woods An experimental parsing system for transition network grammars, in R. Rustin (ed.), *Natural Language Processing*, New York: Algorithmics Press, 1973, pp. 112–154.
67. A. Biermann B. Ballard Toward natural language computation, *Am. J. Comput. Linguistics*,H **6** (2): 71–86, 1980.
68. R. Hershman R. Kelley, H. Miller User performance with a natural language system for command control, NPRDC TR 79-7, San Diego, CA: Navy Personnel Research & Development Center, 1979.
69. L. E. Green E. C. Berkeley, C. Gottleb Conversation with a computer, *Comput. Autom.*, **8** (10): 9–11, 1959.
70. B. F. Green *et al.* BASEBALL: An automatic question-answerer, in E. A. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, New York: McGraw-Hill, 1963, pp. 207–216.
71. R. K. Lindsay Inferential memory as the basis of machines which understand natural language, in E. A. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, New York: McGraw-Hill, 1963, pp. 217–233.
72. N. Chomsky *Aspects on the Theory of Syntax.* Cambridge, MA: MIT Press, 1965.
73. T. Winograd *Understanding Natural Language*, New York: Academic Press, 1972.
74. T. E. Cheatham S. Warshall Translation of retrieval requests couched in a semi-formal English-like language, *Coun. ACM*, **5** (1): 34–39, 1962.
75. H. G. Bohnert Research summary, Project LOGOS, *Commun. ACM*, July 1962.
76. R. A. Kirsch Computer interpretation of English text and picture patterns, *IEEE Trans. Electron. Comput.*, **13** (4): 363–376, 1964.
77. R. F. Simmons S. Klein, K. L. McConlogue Maximum depth indexing for computer retrieval of English language data, *Am. Doc.*, **15** (3): 196–204, 1964.
78. B. Raphael *SIR*: A computer program for semantic information retrieval, in M. Minsky (ed.), *Semantic Information Processing*, Cambridge, MA: MIT Press, 1968, pp. 33–145.
79. D. Bobrow Natural language input for a computer problem-solving system, in M. Minsky (ed.), *Semantic Information Processing*, Cambridge, MA: MIT Press, 1968, pp. 146–226.
80. C. C. Green Theorem proving by resolution as a basic for question-answering systems, in B. Metzer and D. Michie (eds.). *Machine Intelligence* Vol. 4, New York: American Elsevier, 1969, pp. 183–208.
81. E. Chamiak Computer solution of calculus word problems, *Proc. Int. J. Conf. Artif. Intell.*, Bedford, MA: 1969, pp. 303–316.
82. R. M. Kaplan Augmented transition networks as psychological models of sentence Comprehension. *Artif. Intell. J.*, **3**: 77–100, 1972.
83. R. Schank Conceptual dependency: A theory of natural language understanding, *Cogn. Psychol.* **3**: 552–631, 1972.
84. R. Sohank *et al.* Margie: Memory, analysis, response generation, and inference on English, *Third International Joint Conference on Artificial Intelligence.* Stanford Research Institute, Menlo Park, CA: 1973, pp. 255–261.
85. Y. Wilks Preference semantics, Tech. Rep. AIM-206, Stanford Al Project, Stanford, CA: Stanford University, 1973.
86. G. Lakoff Linguistics and natural logic, in D. Davidson and G. Harman (eds.), *Semantics of Natural Language*, Boston: Reidel, 1972, pp. 545–665.
87. L. Schubert N. Cerecone, R. Goebel The structure and organization of a semantic net for Comprehension and inference, in N. Findler (ed.), *Associative Networks: Representation and Use of Knowledge by Machine*, New York: Academic Press, 1979, pp. 121–175.
88. E. Brill R. Mooney An overview of empirical natural language processing, *AI Mag.*, **18** (4): 13–24, 1997.
89. H. Ng J. Zelle Corpus-based approaches to semantic interpretation in natural language processing, *AI Mag.*, **18** (4): 45–64, 1997.
90. J. Zelle Using inductive: logic programming to automate the construction of natural language parsers, Ph.D. Thesis, Department of Computer Science, University of Texas at Austin, 1995.
91. R. Burke *et al.* Question answering from frequently-asked question fites: Experiences with the *FAQ* finder system, Techn. Rept. TR-97-05, Chicago: Department of Computer Science, University of Chicago, 1997.
92. K. Hammond R. Burke, K. Schmitt A case-based approach to knowledge navigation, in *AAAI Workshop on Indexing and Reuse in Multimedia Systems*, American Association for Artificial Intelligence, 1994, pp. 46–57.
93. K. Miller C. Walker *Natural Language and Voice Processing: An Assessment of Technology and Applications.* Lilburn, GA: Fairmont Press, 1990.
94. S. P. Shwartz *Applied Natural Language Processing*, Princeton, NJ: Petrocelli Books, 1987.
95. G. I. Jonsson (1991). The development of IBM SAA Language Access: An experience report, *Proc. Suppl. 7th Int. Conf. on Data Engineering*, Kobe, Japan, April 8–12, 1991, Los Alamitos, CA: IEEE Computer Society Press, 1991, pp. 13–21.

96. L. R. Harris H. A. Harris, M. A. Groch *English Wizard User's Guide (Version 1.0)*, Linguistic Technology Corporation, Acton, MA: 1995.
97. B. Engelien R. McBryde *Natural Language Markets: Commercial Strategies* London: Ovum Ltd. 1991.
98. L. R. Harris Natural language: A brightened outlook, *AI Expert: The Hitchhiker's Guide to Artificial Intelligence*, San Francisco: Freeman, 1995, pp. 43–50.

NICK CERCONE
University of Waterloo
KANLAYA NARUEDOMKUL
Mahidol University