International Journal of Computer and Information Sciences, Vol. 6, No. 4, 1977

Simple-English for Data Base Communication

J. A. Moyne[1]

Received September 1976; revised January 1977

Three classes of the so-called natural languages for communication with data bases are defined: *English-like*, *pseudo-English*, and *simple-English*. It is argued that English-like and pseudo-English languages are normally more difficult to learn and use than artificial programming languages with no overt claim to English likeness. Simple-English is presented as a family of languages in which many restrictions (which hamper learning) are removed through interaction with, and drawing inferences from, the data base and the underlying system. It is concluded, however, that English likeness and ease of learning may be contradictory notions.

KEY WORDS: Ambiguity problems; data base communication; data base models; English-like and pseudo-English languages; learning problems; linguistics; narrative languages; semantics; simple-English.

1. INTRODUCTION

This paper is concerned with a language, or perhaps a family of languages, for communication with data base systems. *Simple-English* is not claimed to be a general-purpose programming language; it is too intimately tied with particular data bases and their applications to be called general or universal. We have taken the modest approach of developing models for specific applications. To develop a lingua franca or the so-called "English as a programming language," presupposes significant breakthroughs in both linguistics and computer science which, in the opinion of the author, are not forthcoming in the near future

[1] Department of Computer Science, Queens College of The City University of New York, Flushing, New York.

828/6/4-5

We have an abundance of literature in the form of manuals, journal articles, and technical reports on all aspects of information management systems including languages for communication with them, but as a point of departure we can take the CODASYL survey,[5] which describes 10 large data management systems, five self-contained and five host-language. The communicative devices for these systems are the following "functions": interrogation, update, and data administration Some systems have different types of languages for each of these functions; others use the same type for all the functions. However, all the languages for these systems are classified into the following types:

a. Narrative
b. Keyword
c. Separator
d. Fixed-position (1)

We are here interested in the *narrative* type, which is also referred to as *English-like*. This type of language is represented by seemingly English sentences with severe restrictions. Examples of the narrative language given for Data Base Task Group (DBTG) are the following[2]:

a. SCHEMA NAME IS EMPFILE
b. RECORD NAME IS EMPREC (2)
c. LOCATION MODE IS CALC USING EMPNO

Note the resemblance between these and COBOL statements, which is also classified as a narrative language. In narrative statements, one can often include additional *noise words* which play no role in the interpretation and understanding or execution of the statement; an example in DBTG is

THE PRIVACY LOCK FOR *THE* REMOVE *FUNCTION* IS AUTHENTICATE (3)

where the underlined words are noise words.

2. ENGLISH-LIKE AND PSEUDO-ENGLISH LANGUAGES

A striking characteristic of the narrative languages is that the more "narrative" they get, the more complex and cumbersome they become.

[2] The 10 systems described in the CODASYL report include five self-contained, GIS, MARK IV, NIPS/FFS, TDMS, UL/1, and five host-language systems, COBOL, DBTG, IDS, IMS, SC-1. DBTG is a proposal and has not been implemented. For a discussion of a number of on-line systems and their languages, see Ref. 9.

The CODASYL report has some examples of procedures written in the different languages of the 10 systems, and a cursory examination of these may convince the reader. The author believes that, in general, it is easier for a person to learn a new simple programming language than to learn to use his own language (e.g., English) in unfamiliar and artificial ways. It should be pointed out, however, that by this statement we are not taking any position concerning the usefulness of narrative languages for various purposes; we are simply making an observation about complexity of structure and problems of learning.

Attempts have been made to alleviate learning problems by incorporating functional procedures into the meaning of English words. Thus, a word can represent an underlying function in the sense of implicit functions in FORTRAN, for example. The author developed an experimental programming language system several years ago on this principle.[12] The following is a program written in this language for generating a concordance of an input natural language text[3]:

```
Read the text from the Input.
Make concordance.                                    (4)
Write on the Output Tape.
Stop.
```

This programming language had a built-in procedure for concordance generation that could be used in the above manner. If we did not want to use this procedure, we could write a concordance program as follows:

```
"Input" is the next file on the Input Tape,
    "List" is an array.
"Text" is "—" followed by the input.
Repeat the next three sentences until there are less than seven
  words left in the text, with index I starting at 1 and
  increasing by 1 each time:                         (5)
Move the first seven words of the text to List(I).
Erase the first word in the text.
Set heading "Concordance before sort".
Output "Print the heading on a new page and print the List."
Execute output.
Sort the List on the 9th word of each item.
Set heading "concordance after sort" and execute output.
End.
```

More successful attempts have been made by limiting the applications of such languages to some narrow fields. Many such special-purpose

[3] Examples (4) and (5) are copied from pp. 5–7 of Ref. 13.

languages have been developed with various degrees of success.[4] A further extension of these attempts is in the development of general-purpose macro generators for programming languages such as PL/I, FORTRAN, and COBOL.

Contrasting with these developments are systems in which natural languages play a more significant role in a natural way. For example, in the systems for English, there are nontrivial grammars of the English language, and a sentence representing a command or a request, or conveying a piece of information is fully analyzed and understood before it is translated into machine instructions. Thus, there are no "noise words": every word used in a sentence plays a role in its analysis and understanding. To avoid confusion, let us call these systems *pseudo-English*. We use the term *pseudo-* because of the practical needs for severe restrictions and because, if the language is used for more than simple interactive communication with a data base, the user must write a data management task in a highly procedural manner. Many such systems have been developed during the past decade.[5] For example, the author[14,10] developed a language for communication with a data base consisting of the catalogue of a small library of 4000 items. The system would process queries put to the library such as the following:

Who wrote "Advances in Computers"?
Find all the books by Altman and list them.
If you have any books on automata theory, list the authors. (6)
Have any books on compilers been received?
What documents do we have pertaining to graph theory?

Another example of what we have called pseudo-English in this paper is Woods's model for airline flight information:[28]

Does American Airlines have a flight that goes from Boston to Chicago?
What meals do I get on Flight AA-57? (7)
What is the departure time from Boston of every American Airlines flight that goes from Boston to Chicago?

Note that the differences between English-like languages [Examples (2)–(5)] and pseudo-English languages [examples (6) and (7)] are not merely in some outward appearances, but in the underlying processors for these languages. In English-like languages, sentences must be written and ordered as rigid programming language statements, with noise words added as palatable paddings. The existence of library procedures [example (4)] can

[4] For details, see Ref. 21; also see the Annual Roster of Programming Languages published by Jean Sammet in *Computing Reviews*.

[5] For details and recent surveys of such systems, see Refs. 11, 17, 22, and 26, and the references given in those documents.

help make a program even more deceptively "English." The principal processor for such languages is usually a compiler with additional preprocessors supplied to handle uncommon operations. In pseudo-English languages, on the other hand, the English sentences, albeit highly restricted, must undergo extensive linguistic analysis. The processors for these are normally written as interpreters in high-level languages (PL/I, LISP, etc.), requiring additional memory size and costs in processing speeds.

To recapitulate, many of the so-called English-like and pseudo-English languages suffer from two fundamental drawbacks: first, our claim that learning to use the English language in a highly restricted and unnatural way is often more difficult than learning an artificial programming language, and, second, the fact that underlying such languages are often large systems, ranging from complex preprocessors to interpreters and other programs requiring double compilation, lots of memory, and auxiliary storage, and having relatively slow speeds.

The second objection is being met by technological developments in higher speeds, larger memories and more efficient storage techniques, multiprocessors, building of high level functions into the hardware, and other relevant advancements in computer technology. We further discuss the first drawback in a later section. Before leaving this section, however, let us note that there is another situation in which the data base is unstructured or semistructured and the data consists of a running text in a natural language (e.g., English). An interesting and worthwhile task would be to develop processors that would analyze and structure the data base for retrieval purposes. Many attempts in this direction have been made; for some important results, see Woods *et al.*[28] and Salton.[20] The surveys noted under footnote 4 contain reviews of these efforts as well. In this paper, however, we are concerned with the communicative devices for structured data bases, and we do not consider the problems of text processing in connection with the structuring of the data base.

3. A MODEL DATA BASE

In order to describe a generalized language that is independent of any data base, we must assume a generalized logical data structure. The actual storage structure can be varied as long as we can map it into a standard logical data structure. Let us then define a very simple logical data structure.

We assume a data base D consisting of an arbitrary number of files F:

$$D = \{F_1, F_2, F_3, ..., F_n\} \tag{8}$$

Let us assume further that each file F_i is stored as a matrix in which each row $R \in F_i$ can be represented as

$$R = (I, N, \{P_1, P_2, ..., P_k \mid k \geqslant 0\}) \tag{9}$$

where I is a unique identification number, possibly a pointer, N is the name or *head* of a data group, and each P_i is a property of N. Thus, each row R is a *data group* or record, and each element E in the group can be referred to as a *data element*. If the file is, for example, an employee file, then each R_i would be an employee record, where I might be his person number, N his name, and P_i his age, education, salary, etc. Note, incidentally, that while each data element E is a variable-value pair, the variables need be recorded or "understood" only once in each file as the column headings of the file matrix. An employee file may, then, conceptually look something like the following:

MAN-#	NAME	BIRTH	SALARY	SKILLS	...
12345	Jones, TD	110237	25670	A, B, C	...
54672	Smith, JB	052346	17850	B, X, D	...
⋮	⋮	⋮	⋮	⋮	⋮

(10)

Obviously, some of the column headings such as NAME and SKILLS may have complex structures and may contain subheadings such as FIRST, MIDDLE, LAST or CODE, DESCRIPTION, etc., but these details are easy to detect and implement and we need not be concerned about them here.

Note that any data group G in an entire data base with the above logical structure can be represented or referenced as

$$G = (F_i, I_j, \{P_{1_j}, P_{2_j}, ..., P_{k_j}\} \mid i, j \geqslant 1; k \geqslant 0) \tag{11}$$

However, for the simple identification of any $G \in D$, we need only the pair (F_i, I_j). Let us represent this pair as ψ; then any data element E in the data base can be represented as

$$E = (\psi, P_m) \tag{12}$$

This element can represent, for example, the salary of a man in an employee file whose person number is I_j. We can then think of this data base structure as a collection of data elements each of which can be referenced or accessed as noted above. This will, incidentally, give us a multifile access device.

We must now reduce each statement for communication with the data base into a number of E-pairs. The statement might give some Boolean function of E-pairs and request the retrieval of others. For example, the

sentence, "What is the salary and birth data of employee 12345 with skill A?" (assuming that employee file is file No. 15) might be reduced to

$$\begin{aligned} \psi &= F_{15}, \ 12345 \\ E &= \psi, \ \text{SKILLS(A)} \\ E &= \psi, \ \text{SALARY} \\ E &= \psi, \ \text{BRITH DATE} \\ &\text{Given: Person-No} \wedge \text{Skills} \\ &\text{Requested: Salary} \wedge \text{Birthday} \end{aligned} \tag{13}$$

The learning problems notwithstanding, the language types listed under (1) as well as what we have called pseudo-English languages can generally undergo this sort of analysis. For example, although the linguistic analysis of sentence (14), as described in ref. 10, is complicated, the final result can be easily stated in terms of our present analysis[6]:

$$\begin{aligned} &\text{Have any books on automata been written?} \\ &\text{Given: A} = \text{Type of document (book)} \wedge \text{Topic (automata)} \\ &\text{Requested: All A in D} \end{aligned} \tag{14}$$

Within the confines of this paper, we cannot engage in detailed discussion of the analysis and translation of pseudo-English languages. It has been demonstrated that under the present state of the art English-like and pseudo-English languages can be processed with reasonable success.[10,14,19] The fundamental difficulty in their application remains the problem of learning, and we discuss this problem in the next section. We also argue that the problems of ambiguity inherent in pseudo-English languages are learning problems.

4. SIMPLE-ENGLISH LANGUAGES

We use the term *simple-English* to distinguish a family of languages from the *English-like* and *pseudo-English* systems described in this paper. The objective of simple-English and its underlying processors is to reduce to the minimum the learning drudgery of the user. If we want these languages to be general and relatively independent of specific applications or storage structures, we must assume a most general data structure such as the model described in this paper.

[6] Actual analysis of (14) in Ref. 10 involves processing through a transformational grammar of English which we cannot describe in this paper. For details see Refs. 6 and 10; see also Ref. 15 for an introduction to transformational grammars written for computer scientists. At this point, a vigilant reader may object to the interpretation of (14) as "All A in D" and say that (14) is a yes/no question or, at most, that it must be interpreted as "Exists A in D." Later in this paper, under "Problems of Ambiguity," we make a further observation concerning this interpretation.

Let us recall that we want simple-English to be used for writing tasks (programs) for transactions with a data base, and not just for individual statements for interactive on-line processing. Tasks for computer processing are normally written as procedures or algorithms; writing procedures in a highly restricted natural language is difficult and, as noted before, presents ample learning problems for the user.

A characteristic of simple-English in the context of this paper is that it must be nonprocedural. In cases where we need a procedural language for some application, the existing programming languages would be more appropriate. If there is any difficulty in learning such languages, then it seems that our efforts would best be spent in inventing simple artificial languages for layman applications. This simplicity cannot, however, be achieved by making such artificial languages English-like or pseudo-English; in fact, it may be achieved by moving in the opposite direction, as in APL. The crux of our argument is that *ease of learning* and *English-likeness* do not presuppose each other; on the contrary, empirical evidence has shown them to be contradictory terms in many cases.[7]

By nonprocedural we mean this[8]: The language should have no explicit or implicit GOTO statements, DO-loops, or the need for referring explicitly to any statement that is to follow a current statement. On the other hand, the underlying system must have the capability to record and remember the relevant results obtained from previous statements contained in a task. This specification should not, however, be confused with the current concerns about structured programming and GOTO-less programs (cf. Knuth[8]). The processor for simple-English, whether a compiler or an interpreter, must process the "task" and translate it into suitable machine instructions. In processing each statement, the processor should have the capability to draw inferences from the previously processed statements or from a universal semantic component, and perhaps to make some predictions about what statements may follow. None of this should, however, require an overt reference (GOTO) to actual individual statements in the task.

Before any further discussion of simple-English, we should perhaps point out that other models have been developed by the author as well as by other scholars for communication in English with computer systems. Discussions and surveys of such systems are available in the literature.[3,10,17,20,23,29] A survey of various proposals for the semantic component

[7] To take an example from programming languages, experiments by the author and his colleagues at Queens College have shown that beginning students, irrespective of their background, learn and master APL and FORTRAN faster than COBOL and also make many fewer errors in the former two languages.

[8] For recent discussions and definitions of nonprocedural languages, and other articles in the proceedings also relevant for review, see Ref. 24.

of such natural language systems appears in Pacak and Pratt,[18] and Winograd's[27] limited but significant model for using heuristic procedures for language analysis and understanding is relevant for our purposes. There are also programming languages and systems that provide powerful tools for developing natural language systems: starting with COMIT,[30] developed for computational linguistics and machine translation, to special versions of LISP and CONNIVER and PLANNER[7,25] as tools for heuristic processing and the kind of inferential procedures involved in language "understanding" (see Bobrow and Raphael[2] for a survey of such languages).

Most of the other models are concerned with using restricted English in query and question-answering systems, generally involving the processing and execution of individual questions or commands. Our concern with simple-English is to develop a family of languages for writing complete tasks or programs in a nonprocedural manner.

5. PROBLEMS OF AMBIGUITY

The simple-English languages proposed in this paper must have reasonable grammars of English in their processors and, in order for these languages to be most general and require the least effort in learning, the grammars must recognize and account for the various ambiguities that can occur even in the most simple English sentences. In fact, for the English-like and many of the existing pseudo-English languages developed for practical purposes, a major problem of learning lies in the artificial removing of the inherent ambiguities from sentences. This practice requires the user to be constantly conscious of some particular meaning of words and phrases, and, in general, of the use of common English sentences in restricted and unfamiliar ways. On the other hand, in practical data base communication, the computer system cannot have nondeterministic procedures resulting in more than one analysis and interpretation for an input statement. This, then, is the dilemma of using English as a programming language. The problem remains unresolved in any practical sense. In this section, however, we propose a pragmatic approach for making simple-English a reasonable language family for data base management. The examples of ambiguous sentences used in this section are all related to a model of a pseudo-English system developed by the author[14] and a model of simple-English outlined in the following section.

We can divide the ambiguities in English into three types: lexical, syntactic, and semantic. Examples of lexical ambiguity are found in the following sentences:

a. Do you have any books on computers?
b. I saw the leg of lamb. (15)

In (15a) the preposition *on* could mean "on the top of" or "about," "concerning," etc.; in (15b) *saw* could be the past tense of *to see* or it could be the verb meaning "to cut with a saw." For structural ambiguity, we can cite the following example:

We subscribe to many journals in the library. (16)

Here the prepositional phrase "in the library" can modify *we* with the reading that we who are in the library subscribe to many journals (which may or may not be in the library). Compare this with

We write to many subscribers in the library. (17)

On the other hand, the phrase "in the library" in (16) can modify *journals*, meaning that many of the journals in the library are subscribed to by us. Under semantic ambiguity, we lump together various ambiguities ranging from those that can be accounted for by formal semantic features and selectional restrictions to those statements whose truth value can only be determined from extralinguistic knowledge of the world sources. Current debate on semantic theory is too controversial and the state of the art too fluid to draw any dividing lines among semantic subcategories. Furthermore, if we push semantics to its logical conclusion, all other ambiguities also fall under this category. For example, the following sentences may be considered to have structural ambiguities similar to (16):

a. The cook bought the vegetables in the bag.
b. The cook bought the vegetables in the store. (18)

However, the nouns *bag* and *store* can have certain features or attributes attached to them in the lexicon that rule out the reading in (18a) in which the cook goes into a bag to purchase vegetables. Thus, theoretically it is feasible to envisage a grammar system with a lexicon and semantic base including features, semantic markers, and other components that can fully account for a language, including ambiguities and anomalies at all levels.[4] In practice, however, the construction of such a grammar for computer applications is not feasible. Apart from a number of unresolved crucial questions about the nature of languages, the construction of a dictionary containing this sort of semantic marker and extralinguistic information amounts to, as Bar-Hillel[1] pointed out over a decade ago, the building of a universal encyclopedia with unlimited bounds, for "the number of facts we human beings know is, in a certain very pregnant sense, infinite" (p. 177).

Human beings disambiguate sentences through contextual references and by inference from their encyclopedic knowledge. If we place restrictions on these devices to make the language manageable for computer applications,

we impose severe learning problems. The author[17] has proposed an approach in which the burden of restriction is not on the language user but on the comprehension of the computer system. This will then allow the user to use the language in a relatively unrestricted fashion, but the computer will understand the language in its own limited ways. The principle behind this proposal is very simple:

> The computer system shall accept only what it can understand and process. (19)

By *computer system* in this context, we mean the language processor plus the ability to carry out certain instructions (i.e., compilation and execution). We can perhaps best illustrate this principle by giving some examples. Consider first the sentence in (15a), "Do you have any books on computers?" Following our previous simplified examples, this sentence might be analyzed as follows:

$$\begin{aligned} \text{Given:}\quad & A' = \text{books on the top of computers} \\ & A'' = \text{books about computers} \\ & A = A' \vee A'' \\ \text{Requested:}\quad & \text{All } A \text{ in } D \end{aligned} \tag{20}$$

At the time of execution, the computer system will realize that it has no capability to reach on the top of any computer; the A' interpretation is, therefore, rejected and we have $A = A''$ unambiguously. As another example, consider the sentence in (14), "Have any books on automata been written?"

$$\begin{aligned} \text{Given:}\quad & A' = \text{book on the top of automata} \wedge \text{written} \\ & A'' = \text{book about automata} \wedge \text{written} \\ & A = A' \vee A'' \\ \text{Requested:}\quad & \text{All } A \text{ in } D \end{aligned} \tag{21}$$

Here again the system will be unable to verify the truth value of A' and will reject it.

Our second example in (21) raises another question which can be better illustrated with the following example:

> Can you tell me if you have any books about automata? (22)

Queries such as (14) and (22) are, strictly speaking, yes/no questions; but the user more often wants for the answer not just a *yes* or a *no* but a list of the books. This is automatically achieved by the principle in (19) if the system has no way of determining about its own capabilities.

$$\begin{aligned} &\text{Given: } A = \text{book about automata} \\ &\text{Requested: Q(CAN TELL}(A \text{ in } D)) \vee (\text{all } A \text{ in } D) \end{aligned} \tag{23}$$

Similar constraints can be imposed by interaction with the data base and/or the information contained in other statements in a task description. For example, the sentence

List the names of all employees in the Accounting Department with five dependents. (24)

has structural ambiguity in that the phrase *five dependents* can modify *employees* or *Accounting Department*. Let us represent this sentence as "List all X in Y with Z; we then have

$$\begin{aligned} \text{Given:} \quad & A' = X(Z) \subset Y \\ & A'' = X \subset Y(Z) \\ & A = A' \vee A'' \\ \text{Requested:} \quad & \text{All } A \end{aligned} \tag{25}$$

If, however, the data base has dependents listed for employees but not for the Accounting Department, then the A'' interpretation will be rejected.[9]

6. SAMPLE IMPLEMENTATION

Complete detailing of the specifications and implementation of a simple-English language, like any other language, requires much more space than can be afforded by a section in a short, general article. In this section, therefore, we try to describe some of the main features of a sample implementation in order to give the reader an overview of the language and its processor.

For this implementation, we have developed an abstract robot-cook that receives instructions for meals and responds by generating a procedure for the preparation and serving of the meals. Let us trace through an example:

I want a rare hamburger for lunch. (26)

This sentence, as we see below, may or may not involve a large number of operations depending on what is stored in the various files at the time of the processing of the sentence. Without getting into detailed discussion of the linguistic analysis of (26), we note that there is a lexicon in the processor that includes "rules" as definitions of words. These rules generally represent words as operand–operator pairs in which the operator is a meta symbol, often a function or macro name with one or more arguments as its collective

[9] For a variant of this approach in which the resolution of ambiguity is at the level of linguistic analysis resulting in a unique parse, see Ref. 16.

operand. Thus, in processing the sentence in (26), the following rules, among others, may be used.

1. hamburger → BROIL (meat patty) ∧ COOK (mode)
2. meat patty → MOLD (ground meat) ∧ AMOUNT ($\frac{1}{4}$)
3. ground meat → GRIND (meat) ∧ TYPE (cut) ∧ AMOUNT ($\frac{1}{4}$, 5) (27)
4. mode → rare | medium | well done
5. meat → beef | pork | lamb | ...
6. cut → sirloin | chuck | round | ...

The operators BROIL, MOLD, etc., as indicated above, are complex symbols and would expand to other operator–operand pairs. For example, BROIL is defined as: SELECT (oven); SET TEM[6] (broil). Furthermore, primitive words in the lexicon have features or attributes associated with them; for example, associated with the word *rare* is a time feature of 3 min (these attributes are in addition to the standard syntactic and semantic features required for the analysis of sentences).

The structure of the data base is conceptually identical with the model that we have described in this paper, except that items and their attributes are coded and stored in a compact way for efficiency. We have a data base called kitchen (K) which contains three files: refrigerator (R), kitchen table (T), and pantry shelves (S). In addition to the rules, such as those in (27), entries in the lexicon have other features associated with them; for example, for the attributes of hamburger, we have its ingredients: chuck beef, salt, pepper,.... Every entry has also storage class attributes indicated for primary and secondary storage; for example, meat has refrigerator as its primary and kitchen table as its secondary storage. Thus, when meat is called for, first the R file is searched and then the T file. If both the searches fail, there is no meat in storage. Meat items in the R file conceptually are organized as in the following table:

Item no. (I)	Name (N)	Kind (P_1)	Cut (P_2)	Weight (P_3)	Quantity (P_4)
000162	meat	beef	sirloin	2	1
000163	meat	beef	chuck	6	1
⋮	⋮	⋮	⋮	⋮	⋮

(28)

Returning to the sentence in (26), let us assume that the sentence is used for the first time and we have meat items stored in the data base (file R) as shown in (28). Beef chuck will be selected along with Rule 3 in (27) which invokes the GRIND operator. Note that the AMOUNT operator in this rule has lower and upper bounds ranging from $\frac{1}{4}$ to 5 lb. The selection of

amount is random. A random number generator will select a number between $\frac{1}{4}$ and 5 in increments of $\frac{1}{4}$. Suppose 3 is selected. Thus, 3 lb of beef chuck is taken [i.e., 3 is subtracted from the original storage of 6 lb in the *R* file (28) leaving 3 lbs of chuck behind]. After the "execution" of Rule 3, we store in the data base 3 lb of ground meat. Rule 2 will then apply. In Rule 2 the AMOUNT is invariant; that is, each time a meat patty is made from $\frac{1}{4}$ lb of ground meat. However, there is no restriction to the number of times that this rule can reapply.[10] Potentially, the rule may apply and reapply until all the 3 lb of ground beef has been made into 12 meat patties. Actually, the number of times that this rule applies is again governed by the output of the random number generator whose lower bound at this time is one, and its upper bound is the amount of ground meat in storage divided by the required amount for each patty. Let us assume number 4 is generated for this decision; we will then make four meat patties and store them in the file *R* and deduct 1 lb from the abount of ground meat in storage. Next, Rule 1 will apply. The number of times that this rule will apply is governed by the number of hamburgers asked for in the original input sentence. In the present case, we will make one hamburger. The data in the *R* file [cf. (28)] will now look like the following:

000162	meat	beef	sirloin	2	1	
000163	meat	beef	chuck	3	1	
⋮	⋮	⋮	⋮	⋮		
000195	ground meat	beef	ground	2	1	(29)
000196	meat patty	beef	patty	$\frac{3}{4}$	3	

If the sentence in (26), with the storage items as shown in (29), is used again, only Rule 1 in (27) will apply, since meat patty is available in storage and there is no need for Rules 2 and 3. After the sentence in (26) has been processed again, the entry for meat patty will change to

000196 meat patty beef patty $\frac{1}{2}$ 2 (30)

In tracing through the processing of (26), we have left out many of the details, but we have given enough to give the reader a general view. One

[10] Some of the rules in the lexicon are marked as repetitive; these rules can reapply to the same construction as long as the conditions for their application hold. Other rules, not marked as repetitive, can apply only once in each cycle. The details of this are not important for our present discussion.

further point about (26), however, bears some further explanation. The word *lunch* in (26) is defined by the following "rule":

$$\text{lunch} \rightarrow (\text{first course} \wedge \text{main course} \wedge \text{desert} \wedge \text{beverage}) \wedge \text{TIME (hour)} \tag{31}$$

The word *hamburger* in the lexicon has an attribute indicating that it is a "main course," and *tea* and *coffee*, for example, are marked as beverages. Since the sentence in (26) does not refer to any first course, desert, or beverage item, these will be marked as nill in the final analysis of this sentence.

We can update the files in the data base by "shopping instructions"; that is, write and execute a task (program) in the same laguage, for example:

> Check the refrigerator; if we are short of meat, buy 5 lb of steaks, 2 lb of stew meat, and 2 lb of ground beef. We may also need lemon juice, green peas, and flour. Get two loaves of white bread from the bakery. Etc. (32)

Notions such as "short of" are arbitrarily defined over a certain weight range. We can also automate the file updating by establishing a minimum threshold for each item and automatically increase the storage quantity of the item by some arbitrary increment when it reaches the specific threshold.

Two other implementations of simple-English languages are underway at present. One is in the area of on-line high school geometry exercises and the other is in propositional calculus. But descriptions of these will have to be subjects of other communications after sufficient work has been done in them.

The preliminary implementation of the "robot-cook" has been done by a number of students of the author. The programs written in PL/1 are highly modular and run on IBM 370/168. In the lexicon, so far there are about 150 words and 50 rules. The size of the data base is variable, and grows or diminishes in the course of a run. At present, we have allowed a matrix with a maximum dimension of 200×12. Perhaps the most difficult and somewhat messy modules are those dealing with linguistic analysis, disambiguation, etc. While study and experiments continue in linguistic analysis, we use models previously developed by the author and his colleagues.[6,10,13] A number of new procedures for disambiguation have been written in PL/1 and FORTRAN and these are now being tested.

The implementation of geometry exercises is also being done in PL/1, while for the propositional calculus project LISP is being used.

7. CONCLUSIONS

Some significant achievements notwithstanding, we have argued that English-like and pseudo-English languages may not be as desirable as more

formal programming languages for writing tasks for data base communication. We have proposed a family of *simple-English* languages which may diminish the learning problems in the English-like and pseudo-English languages. This is done through continuous contextual analysis, interaction with the data base, and drawing inferences. We have also introduced special ways for treating ambiguities, thus reducing the needs for placing restrictions on the language. By citing various models and developments and outlining an implementation, we have suggested that the state of the art permits the construction of a simple-English processor.

We should hasten to point out that we do not in the least claim that simple-English as described in this paper has no major unresolved problems. It is perhaps only relatively more expedient than English-like and pseudo-English languages. The reader must have noticed that even some of the examples we have given in this paper do not readily fall within the purview of our ambiguity treatment, and it is easy to make up other counterexamples that defy any deterministic analysis for practical purposes.

We believe that continued investigations in processing natural languages by computers have important theoretical and practical consequences for both linguistics and information sciences, but the notion that ease of learning and facility in usage can be had by moving superficially in the direction of natural languages for programming computers is an illusion. If we need simple languages for layman applications, we should invent artificial languages that do not superficially resemble natural languages and will not, therefore, confuse the casual user by similarities with his native language. On the other hand, when natural language is appropriate (particularly in data base communication), then the resemblance must be much more than surface deep.

REFERENCES

1. Y. Bar-Hillel, *Language and Information* (Addison-Wesley, Reading, Mass., 1964).
2. D. G. Bobrow and B. Raphael, "New programming languages for artificial intelligence research," *ACM Comput. Rev.* 6(3):153–174 (September 1974).
3. Center for Applied Linguistics, *Research Trends in Computational Linguistics* (Center for Applied Linguistics, Washington, D.C., 1972).
4. N. Chomsky, *Aspects of the Theory of Syntax* (MIT Press, Cambridge, Mass., 1965).
5. *Feature Analysis of Generalized Data Base Management Systems,* CODASYL Systems Committee, ACM, New York (May 1971).
6. P. Culicover, J. Kimball, D. Lewis, D. Loveman, and J. Moyne, "An Automated Recognition Grammar for English," Technical Report FSC 69-5007, IBM, Cambridge, Mass. (1969).
7. C. Hewitt, "PLANNER, A Project MAC report," MAC-M-386, MIT, Cambridge, Mass. (October 1968); revised August 1970.

8. D. E. Knuth, "Structured programming with go to statements," *ACM Comput. Surv.* 6(4):261–301 (December 1974).
9. F. W. Lancaster and E. G. Fayen, *Information Retrieval On-Line* (Melville, Los Angeles, 1973).
10. D. B. Loveman, J. A. Moyne, and R. G. Tobey, "Cue: A Processor System for Restricted Natural English," *Proceedings of the Symposium on Information Storage and Retrieval*, University of Maryland (1971), pp. 47–59.
11. C. A. Montgomery, "Linguistics and information science," *J. Am. Soc. Inf. Sci.* (May–June 1972), pp. 195–219.
12. J. A. Moyne, "A Simulated Computer for Natural Language Processing," Technical Report TR 00.1463, IBM (1966).
13. J. A. Moyne, "Introduction to an Operational RELADES," Technical Report TR 00.1442, IBM (1966).
14. J. A. Moyne, "Proto-RELADES: A Restrictive Natural Language System," Technical Report BPC 3, IBM, Cambridge, Mass. (1967).
15. J. A. Moyne, "An introduction to transformational grammars," *Int. J. Comput. Math.* 2:169–181 (1968).
16. J. A. Moyne, "Informational retrieval and natural language," *Proc. Am. Soc. Inf. Sci.* 6:259–263 (1969).
17. J. A. Moyne, "Some Grammars and Recognizers for Formal and Natural Languages," in *Advances in Information Systems Science*, Julius T. Tou, Ed. (Plenum, New York, 1974).
18. M. Pacak and A. W. Pratt, "The Function of Semantics in Automated Language Processing, *Proceedings of the Symposium on Information Storage and Retrieval*, University of Maryland (1971), pp. 5–18.
19. R. Rustin, Ed., *Natural Language Processing* (Algorithmics Press, New York, 1973).
20. G. Salton, *Automatic Information Organization and Retrieval* (McGraw-Hill, New York, 1968).
21. J. E. Sammet, *Programming Languages: History and Fundamentals* (Prentice-Hall, Englewood Cliffs, N.J., 1969).
22. S. Y. Sedelow and W. A. Sedelow, *Language Research and the Computer* (The University of Kansas, Lawrence, 1972).
23. R. M. Schwarcz, J. F. Burger, and R. F. Simmons, "A deductive question-answering for natural language inference," *Commun. ACM*, 13(3):167–183 (March 1970).
24. "SIGPLAN, Proceedings of a symposium on very high level languages," *SIGPLAN Notices, ACM*:9(4) (April 1974).
25. G. J. Sussman, and D. V. McDermott, "From conniver to planner, a genetic approach," *Proc. FJCC* (1972).
26. D. E. Walker, "Automated Language Processing," in *Annual Review of Information Science and Technology*, Vol. 8, Carlos A. Cuadra, Ed. (American Society for Information Science, Washington, D.C., 1973).
27. T. Winograd, *Understanding Natural Languages* (Academic Press, New York, 1972).
28. W. A. Woods, "Semantics for a Question-Answering System," Report No. NSF 19 to National Science Foundation, Harvard University, Cambridge, Mass. (1967).
29. W. A. Woods, R. M. Kaplan, and B. Nash-Webber, "The Lunar Sciences Natural Language Information System: Final Report," BBN Report No. 2378, Bolt Beranek and Newman, Cambridge, Mass. (1972).
30. V. Yngve, *COMIT*, 2nd ed. (MIT Press, Cambridge, Mass.).

Printed in Belgium