PROGRAMMING IN NATURAL LANGUAGE:

"NLC" AS A PROTOTYPE

Bruce W. Ballard
Alan W. Biermann

Department of Computer Science
Duke University
Durham, North Carolina 27706

1.0 Introduction

The state of the art in computational linguistics has progressed to the point where it is now possible to process simple programs written in natural language. This report describes a natural language programming system called NLC which enables a computer user to type English commands into a display terminal and watch them executed on example data shown on the screen. The system is designed to process data stored in matrices or tables, and any problem which can be represented in such structures can be handled if the total storage requirements are not excessive.

As an example of a natural language program that can be processed by NLC, consider the few sentences given below. Assume an instructor has kept his or her grades on the machine and wishes to compute final averages. Further assume that the grades are kept in a matrix with the students' names appearing along the left side and that for each student there are four quiz grades, an examination grade, and a sixth entry which is to hold the final average. The final average is to be based one third on the quizzes and two thirds on the examination. The instructor might request that the grades be displayed and then type the following:

```
"Choose a row in the matrix."
"Put the average of the first four
     entries in that row into its
     last entry."
"Double its fifth entry and add that
     to the last entry of that row."
"Divide its last entry by 3."
"Repeat for the other rows."
```

Three things should be noticed about this short program. First, no reference is made to problem domain concepts such as students or grades. The problem domain for the system is the world of matrices, and all references are to related concepts: rows, columns, entries, labels, and simple operations on them. Second, all input sentences are imperatives. It was found that users tended to use this form and that the system design could be greatly improved if the main verb could be assumed to begin the sentence. Thus users are told to begin every input with an imperative verb. The final and most important point concerns the style of programming. While entering the above program, the user could observe the system performing the calculation on a sample row and had the opportunity to verify the correctness of each action. NLC clearly marks each operand when executing a computation so that its operation can be checked. Then the user asked that the series of operations which had been observed to be correctly executed on the example row be executed on the remaining rows. This style of programming provides the user with continuous feedback concerning program correctness while the computation is proceeding and continues a repetitive computation only after one pass through the loop has been verified to be correct.

Examples of much more complicated natural language programs appear in Biermann and Ballard[78] and Ballard[79]. The reader may find other discussions of natural language programming in Green et al[78], Heidorn[74,76], Hobbs[77], Petrick[76], Sammet[66], and Woods[77]. Section 2 provides an overview of the NLC system. Section 3 discusses the kinds of knowledge it must be able to handle: (1) linguistic; (2) domain-specific; and (3) computational. The paper concludes by

---

This material is based upon work supported by the National Science Foundation under Grant Number MCS74-14445-A01.

considering questions related to the generality and extensibility of the NLC design.

2.0 Overview of NLC

To sharpen the focus of the material that follows, the representative input

"Add five te the 2nd positive entry in col 2." (sic)

will be traced through each of the system's four stages of processing: (1) morphological; (2) syntactic; (3) semantic; and (4) computational. Each of these modules is discussed separately in the following subsections.

2.1 The "Scanner"

The role assigned to the scanner is to identify, from the characters of the input, the individual "tokens" to be used during the syntactic phase. The example contains eleven of these (see Table I below). With the exception of the punctuation ".", the tokens are precisely the maximal substrings containing no spaces. Methods of dealing with morphologically complex languages, where tokens actually need to be "parsed" into subunits, are presented in Kay[77].

Although locating individual tokens is a relatively simple matter, determining their meaning(s) is more complicated. In the example, none of the tokens "five", "te", "2nd", or "2" appears in the dictionary. Additional facilities are needed to recognize special classes of tokens such as:

1. integers ("2", "1234")
2. ordinals ("1st", "2nd", "3rd", "4th")
3. spelled-out numbers and ordinals ("five", "fifth")
4. misspelled words ("teh", "sixxth")
5. names ("pivot", "sally")
6. floating-point constants ("3.1", "3.456", "-3.456")
7. abbreviations ("col", "max", "neg")

The present dictionary of about 450 entries includes 60 imperative verbs, 12 domain nouns (e.g. "column"), about 20 functional nouns (like "product" or "square root"), and several adjectives, comparatives and their superlative cognates. For tokens found in the dictionary, multiple meanings occur frequently. For example, "that" can introduce a relative clause ("THAT are positive"), appear as a pronominal article (or so-called demonstrative adjective) as in "THAT entry", or serve as a pronoun ("add THAT to x").

Output from the scanner is sent directly to the parser (syntax analyzer). For the example sentence, eleven tokens are found with the possible meanings indicated in Table I.

	Token	Possible Meanings
1	"Add"	imperative verb
2	"five"	integer(5) float-constant(5.0)
3	"te"	name verb(be) article(the) prep(to) integer(10) float-constant(10.0)
4	"the"	article
5	"2nd"	ordinal
6	"positive"	adjective
7	"entry"	noun
8	"in"	preposition
9	"col"	noun (abbreviated)
10	"2"	integer(2), float-constant(2.0)
11	"."	punctuation

Table I - Output from the Scanner

If "te" were a name previously assigned by the user to a matrix entity, then spelling correction would not be attempted.

2.2 The "Parser"

After the scanner has completed its job of identifying and providing syntactic information about each of the tokens of the input, it is necessary to determine the structure of the user's request. For instance, the example being considered exhibits the overall structure

"add" y1 "to" y2 .

where the y's represent the noun phrase "arguments" of the "add" command. Of the two noun phrases, one is simple ("five") while the other is moderately complex ("the second positive entry in column 2"). Of course, it is impossible to tell immediately that "te" is really "to", which accompanies the word "add", instead of heading an infinitive or a prepositional phrase.

2.2.1 Verbs and Their Operands

The example sentence is typical in that the major effort of parsing in NLC is to recognize the imperative verb and its associated operands. For this reason, noun phrases play an extremely important role. In the domain of matrices, a large proportion of imperative verbs fall into one of 6 categories. Verbs like "double" and "negate" take exactly one operand. "Add" and "subtract" receive two operands which are distinguished by a preposition: "to" for "add", "from" for "subtract". The term we coined to denote a preposition serving this function is verbicle. A related form is the class of one-operand, particle-taking verbs such as "add up" or "round off". The three remaining forms are the two-operand imperative verb without verbicle ("call column 5 terry") and the two zero-operand forms ("quit", "back up"). With verbicles and particles, a number of paraphrases may arise. Thus both "add x to y" and "add to y x" are accepted, as are "add row 1 up", "add up row 1", and "add the numbers up in row 1".

2.2.2 Grammar Networks

The grammar of NLC is expressed in transition networks similar to those described by Woods[70]. An example network is given below for processing sentences of the forms

```
add y1 to y2 .
add to y2 y1 .
```

For the sake of printing the network, the following convention has been adopted: an indentation of two spaces represents a transition to the next (daughter) node.

```
 1  START
 2    PARSE imperative-verb
 3      PARSE verbicle
 4        INVOKE <noun-group> network
 5          INVOKE <noun-group> network
 6            SUCCEED
 7      INVOKE <noun-group> network
 8        PARSE verbicle
 9          INVOKE <noun-group> network
10            SUCCEED
```

Thus the START node has one transition leading to "PARSE imperative". This node has the two transitions to nodes 3 and 7. The remaining nodes each have a unique outgoing transition, except for the SUCCEED nodes, which have none.

A successful traversal of this network involves an "executable" path from the START node to some SUCCEED node. For the example sentence, parsing begins by recognizing the imperative verb "add". Then the transition from 2 to 3 is taken and an attempt is made to parse a verbicle such as "to". This failure causes backup to the alternate transition (7) from the previous node (2), which invokes a sub-network to parse a noun group. At this point "five" is accepted and a return is made to the daughter (8) of the "INVOKE <noun-group> network" node, which successfully parses the verbicle "to". Finally the second noun group (9) is parsed and the input is accepted.

The networks actually used are more complicated than the one shown here because of the variety of verb types and because of complications related to conjunctions.

2.3 The "Semantics" Processor

Given the parse trees of the noun phrases recognized during syntactic analysis, the semantics processor is responsible for determining the specific matrix entities referred to. Since many of the functional aspects of semantics will be dealt with in Sections 3.1 and 3.2, this brief section concentrates on the order in which to perform resolutions. Returning to the example input, "add five to the second positive entry in column 2", the resolution to the constant "5.0" is obviously trivial. However, the processing of the second noun phrase, "the 2nd positive entry in column 2", is more interesting.

The words occurring before the main noun in a typical English noun phrase often require semantic analysis from right to left. To resolve the phrase in question, the following steps are performed:

1. Resolve the noun phrase "column 2".
2. Compute the set of entries in it.
3. Remove from consideration the entries that are not positive.
4. Select the second member remaining. If there are fewer than 2 members, the user's phrase being processed is not meaningful.
5. Apply "the" by checking to see that the most recent value computed agrees in number with the main noun.

An important aspect of the semantics processor is that it can handle arbitrarily deep nestings. For instance, the formidable noun phrase

> "the first 2 and last 3 entries less than 5 in the last row containing an odd number that are negative"

is dealt with as follows:

Order Processed	Token
14	"the"
11	"first 2"
13	"and"
12	"last 3"
7	"entries"
9	"less than"
8	"5"
7	"in"
6	"the"
5	"last"
4	"row"
4	"containing"
3	"an"
2	"odd"
1	"number"
10	"that are negative"

An interesting consideration is the order in which the 3 qualifiers of "entries" should be processed. NLC begins by locating the positional modifiers such as "in". When none are present, context is used to supply a default location.

2.4 The "Matrix Computer"

The example sentence requires only the simplest type of computation: addition of a constant to an entry. After carrying out this command, however, it is necessary to update the screen and to create a new "world" file for semantic processing of the next command. Whenever operations take place, the screen update routine places a "*" beside the values that have changed and another mark (depending upon terminal type) beside operands used as a source. Thus the command "add 5 to the second entry in column 2" results in starring entry (2 2) after removing marks remaining from the previous command.

The most frequent operations performed in the matrix domain are arithmetic. For this reason, the matrix computer consists largely of code to perform 2-operand arithmetic. Some of the non-arithmetic commands include the naming of entries ("name", "call", "label") and meta-commands ("create", "back up"). Knowledge of the various imperatives is represented by a table, so that only a small number of primitives are required. Thus, "double" is treated as "multiply by 2", "increment" as "add 1 to", and so forth. The normal method of handling the verb-operand triple <operator;source;dest> is to execute what in a high-level programming language would be written as "dest = dest <op> source" where <op> is "+", "*", and so on. Of the two operands, the one called "dest" is updated while the source merely participates in computing a value.

Before actual computations can be carried out, it is necessary to decide how the operands are to be treated. Implicit actions upon (for example) entries resemble the way in which containment information is utilized during semantic processing. Thus, "add 5 to row 2" entails adding 5 to each entry in row 2. The situation is similar if "5" is replaced by an entry or by a (scalar) variable. With "add row 1 to row 2", however, the computation involves performing individual additions after pairing the entries in the rows. Still more elaborate behavior is called for when sets of objects are received as an operand. Thus, "add rows 1 and 2 to row 3" results in first pairing the entries of rows 1 and 3 and then of rows 2 and 3. In the case of "add rows 1 and 2 to rows 3 and 4", the operands are both sets of 2 members, and so a pairing yields the effect of "add row 1 to row 3 and add row 2 to row 4". If the user intends a cross-product, the command "add each of the first 2 rows to rows 3 and 4" will suffice, as will "add rows 1 and 2 to row 3 and to row 4" and several other paraphrases.

3.0 Three Types of Knowledge

In devising mechanisms for use in the NLC matrix prototype, a continuing concern has been to try to identify general principles wherever possible, rather than to treat minor issues in isolation. As discussed in Section 4, this increases the possibility of extending the current work into new domains. The following sections discuss some of the ways in which NLC embeds (1) linguistic, (2) domain-specific, and (3) computational knowledge.

3.1 Linguistic Knowledge

Virtually all designers of natural language processors have found it tremendously valuable, if not essential, to begin by restricting the domain of discourse. This serves to limit the kinds of structures to be recognized as well as the actual words to be considered. Thoughtful investigation nevertheless benefits from incorporating features in as general a manner as possible. This increases the likelihood that the techniques developed for the prototype can be transported to other domains.

As a specific example, we felt that declaratives (statements) and interrogatives (questions) would not be needed in the computational matrix domain. For this reason, the system expects each input to be an imperative (command). Placing an additional restriction upon the user, namely, requiring each input to

begin with the imperative verb, has greatly reduced the tasks of the system designers without, it is believed, significantly altering the user's expressive power. Some examples of accepted and unaccepted inputs follow.

ACCEPTED
  Add row 1 to row 3.
  Add to row 3 row 1.
  Add the first row to row 3.
  Add the first row to the third row.

NOT ACCEPTED
  To row 3 add row 1.
  I want row 3 to have row 1 added to it.
  Row 1 is to be added to row 3.
  Row 3 equals its old value plus row 1.

In addition to categorizing verb types within the matrix domain (Section 2.2.1), a number of more general linguistic features demand treatment. A few of these are conjunction, pronominalization, and ambiguity, which are discussed below.

3.1.1 Conjunction

A few ways of incorporating conjunctions into the input "add the 1st row to row 3" are

1. Add the 1st and 2nd rows to row 3.
2. Add the 1st and the 2nd rows to row 3.
3. Add the 1st row and the 2nd row to row 3.
4. Add the 1st, 2nd and 4th rows to row 3.
5. Add the 1st row to rows 3 and 4.
6. Add row 1 and row 2 to row 3.
7. Add row 1 to row 3 and row 4.
8. Add row 1 to row 3 and to row 4.
9. Add row 1 to row 3 and row 2 to row 4.
10. Add row 1 to and subtract row 2 from row 3.

Analysis of these sentences reveals that natural conjunctions need not be confined to well-defined groups of words. (The earlier work of Winograd[72] recognized 3 such units.) For instance, "the first" (sentence 2) consists of an article followed by an ordinal; "to row 3" (sentence 9) is composed of a verbicle and its noun operand; and "add row 1 to" (sentence 10) represents suppression of an operand from the original sentence. Each of the examples is correctly processed by the system.

After parsing conjunctions, semantic analysis and computational actions are necessary. What is the correct handling of "x (y1 and y2) z", given a way to resolve "x y z"? As a specific example, consider the conjoining of ordinals, where the meaning of "the k-th column" is found by applying the ordinal "k-th" to the data structure obtained from processing "column". To process the phrase "the i-th and j-th columns", "i-th" and "j-th" are independently applied to "column" and the values combined, in this case by set union. Since NLC derives meanings independently of not only the specific x, y and z involved but typically of their parts of speech as well, we feel justified in speaking of having incorporated linguistic "knowledge" about conjunctions.

3.1.2 Pronominalization

In order to process pronominal units, NLC utilizes some general principles such as those mentioned below. The overall strategy consists of (1) enumerating referents that are syntactically acceptable and (2) choosing the first one which satisfies semantic constraints. Thus, "the entries in IT" requires a referent which is able to "contain" entries (the inverse of "in"). This involves communication with the domain-specific components to be discussed in Section 3.2.

When interpreting an utterance, the need arises to discern which of several possible referents was intended. Both syntactic and semantic clues aid in reaching a decision. For instance, in the sentence

  "Add column 4 to it."

some conditions sufficient for disallowing possible referents are:

  (syntactic violation)
1. "rows 2 and 3"
   ("it" must refer to a singular noun.)

2. "column 4"
   (In order for a sister operand to be the referent, the reflexive "itself" is required.)

  (semantic or "pragmatic" violation)
3. "entry x"
   (The operation of adding a column to an entry is not meaningful.)

Beyond this level, a few fairly subtle clues come into play. For instance, without further information, the more likely referent of "it" in the sequence

  "Add x to y."
  "Add it to z."

is "x": although "y" is more recent, parallelism is maintained by selecting "x" (see Caramazza[78]).

In addition to the one-word pronouns "it" and "them", NLC accepts such pronominal occurrences as

"Subtract 4 from THAT entry."
"Double THOSE rows."
"Multiply the last row by ITSELF."
"Add the odd entries to THEMSELVES."
"Square the NEXT entry."
"Triple the OTHER numbers in row 5."

The words "next" and "other" (called "special ordinals" in NLC) are included in this list since they entail the identical semantic processing as pronouns.

3.1.3 Ambiguity

One of the more frequent objections to the use of English as a programming language relates to its supposed vagueness and ambiguity (Simmons[78], Petrick[76]). Much attention has been devoted to the problems of ambiguity in NLC in an attempt to develop techniques for overcoming this concern.

3.1.3.1 Distinction Between Structural and Semantic Ambiguity

For our purposes, it is useful to distinguish between structural and semantic ambiguity. In case of structural ambiguity, multiple meanings arise when there are two or more ways of parsing a phrase or sentence. Some typical examples are:

"the first and last odd entries"
"Increment the row incremented by 3."

In the first example, the adjective "odd" could apply to both ordinals or to just the latter. The issue in the second example is which "increment" the modifier "by 3" should be attached to.

Semantic ambiguity, on the other hand, refers to a word or unit with meanings that cannot be distinguished positionally. Thus, the fact that "sum" can be either an imperative verb or a noun presents no problem since "sum row 2" and "the sum of row 2" each have just one parse. However, the multiple meanings for "above", namely "greater than" and "on top of", cannot be structurally distinguished in the phrase "entries above entry x": in both cases, the word is being used as a preposition.

In the matrix domain of NLC, "semantic" ambiguity is of less concern than the "structural" variety for two reasons. First, it appears to arise much less frequently. Second, it can be easily flagged and reported to the user. Structural ambiguity, on the other hand, is a more serious matter, since inputs which are ambiguous with respect to the grammar of NLC may be perfectly clear to human users. Thus the following discussion concerns only structural ambiguity.

3.1.3.2 Resolving Some Specific Structural Ambiguities

Examination of a number of ambiguous sentences reveals that many of them can be shown to possess one of a small number of properties. This suggests formulating some general principles to aid in the design of a disambiguation strategy.

An open question at this time is whether a carefully formulated a priori set of principles can compete with strategies which (expensively) find all parses and then attempt to select the correct one based upon some selection function. Experience with NLC suggests that a careful ordering of grammar rules, seeking to determine the correct parse without ever seeing the others, may sometimes be preferable.

3.1.3.2.1 Ambiguity Through Conjunction

When two units connected by "and" are encountered, the word preceding "and" obviously marks the end of the left conjunct. Similarly, the word following "and" begins the right conjunct. For example:

```
x1  x2  x3  x4 ) and ( x5  x6  x7  x8
```

where the x's represent the tokens of the input sentence. Four of the 16 possible ways to complete the parse are:

```
x1  x2  x3  [ (x4) & (x5) ]  x6  x7  x8
x1  x2  [ (x3  x4) & (x5) ]  x6  x7  x8
x1  x2  [ (x3  x4) & (x5  x6) ]  x7  x8
x1  x2  x3  [ (x4) & (x5  x6) ]  x7  x8
```

Due to syntactic and semantic constraints, few English phrases present such multiplicities. For instance, the ambiguous 8-token command

"Add up the first and last odd positive entries."

presents just 3 legitimate interpretations of the conjunction:

```
... [(first) & (last)] ...
... [(first) & (last odd)] ...
... [(first) & (last odd positive)] ...
```

Recognizing that the pattern

<ordinal> "and" <ordinal> <adjective>

is responsible for the ambiguity here, it is possible to formulate the grammar rules so that the "preferred" parse is the one first found by the parser. Adhering to the general principle that conjuncts tend to have similar type in natural language, NLC favors the first parse given above. This strategy differs sharply from parsing in most high-level programming languages, where the precedence of operators, rather than the type of operands, governs.

3.1.3.2.2 Ambiguity Through Nesting

A common instance where levels of nesting contribute to structural ambiguity involves prepositional phrases modifying nouns. Since these modifiers are optional, the phrase can conceivably be attached to any of several previous nouns. Thus

```
a in b in c
```

presents a typical problem: should "in c" be attached to "b", in a nested fashion (i.e. modifying another modifier), or should it be seen instead in parallel with "in b", modifying "a"? The alternatives are

```
(a (in b (in c)))
(a (in b) (in c))
```

A useful general principle is to attach the prepositional phrase to the closest possible noun when no other factors are involved (Kimball[73]). In the situation at hand, then, the first parse, (a (in b (in c))), will be chosen unless "b in c" is unacceptable, for instance if entities of type b cannot occur "in" entities of type c.

As described in Section 2 of this paper, the NLC design is committed to a firm distinction between modules. Our solution to the problem of nesting-induced ambiguity was to notice that in order to parse a phrase such as

```
"the entries in column 3 in the first
 2 rows"
```

it is not necessary to perform a full semantic analysis (i.e. to determine the referents) of the phrases "column 3" and "the first 2 rows" before checking compatability. More simply, the noun itself, in our domain, is all that need be considered, since the particular rows and column do not affect illegality of column-in-row. Thus, a simple traversal of the syntax tree is sufficient to avoid parses which would be accepted if only purely syntactic criteria were considered.

3.2 Domain-Specific Knowledge

The set of domain entities, or principle nouns of a domain, is particularly important since it determines the accompanying adjectives, relational words, and so forth. Mechanisms which have been devised to handle certain aspects of natural language processing are as varied as the domains themselves. For instance, domain entities for some of the well-known systems have included toy blocks (Winograd[72]), people, places and events (Schank[73]), vehicles and service facilities (Heidorn[72]), ships and ports (Hendrix[77]), circuit components (Brown and Burton[75]), and lunar rocks (Woods et al[72]).

3.2.1 Entities of NLC

A characteristic feature of the NLC matrix domain is the hierarchical organization of its entities. Matrices contain rows and columns, which are in turn comprised of elements. Since containment is a transitive relation, matrices contain entries as well. Scalar variables may be created by users. For convenience, entries, rows, columns, and matrices may receive and later be referred to by names. Because of the hierarchy, entries within a row are often affected when commands referring to the row are carried out. There is also an implicit sequential numbering of rows and columns available for users.

Fortunately, it is possible to display on the terminal screen the current state of the data being manipulated, which tends to minimize the discrepency between the user's view and that of the system.

3.2.2 Properties within NLC

Most commands refer to operands by either position ("row 2") or value ("the positive entries"). Complicated phrases such as "the second greatest negative number in the row that was doubled by the last command" may involve both value ("greatest", "negative") and position ("in the row ..."), and also rely upon the system's memory of previous actions ("doubled by the last command"). Because of the special hierarchical nature of the matrix domain, the "in" relationship is so basic that its semantics are usually involved in resolving simple noun phrases like "the positive entries", even in the absence of an explicit "in".

The most frequent classes of noun phrase units processed by NLC are articles, quantifiers, ordinals, superlatives, adjectives, classifiers, nouns and pronouns, prepositional phrases,

comparatives, and relative clauses. A few other qualifier forms are recognized. Each of these categories involves a specific kind of processing in NLC. For example, the role of an adjective is to perform a subsetting operation. To resolve the noun phrase "the non-negative entries in column 4", the system first finds "entries in column 4" and then applies the predicate "non-negative" to each member to form a subset. Comparatives ("greater than", "equal to") are semantically identical to adjectives but take an operand (and hence follow the noun in English as a multi-word unit). The ordinal and superlative each select one or more members of a set. Processing of ordinals ("first", "second", "last", "first 2") involves only the position of elements in the (ordered) set acted upon. Therefore, ordinal processing is completely domain-independent. For superlatives ("greatest", "greatest 2", "second greatest"), however, the decision involves examining the entire set and appealing to the associated comparative routine ("greater" for "greatest", "less" for "least").

3.3 Computational Knowledge

The term "computational" is chosen to suggest an understanding of some of the ways in which individual commands are employed in sequence to solve problems. Primary considerations are (1) the interpretation of verbs whose scope is a single sentence; (2) various control structures; and (3) provisions for separately defined procedures.

Section 2.4 has discussed the handling of ordinary sentence-level imperatives such as "add" and "double". It is reasonable to regard such commands as primitives within a procedure or looping pattern. In this way, the flow of control at the topmost level is largely independent of the specific semantics of the individual operations occurring within. Of course, the "environment" or context must be carefully maintained for proper pronominal functions and context references (which are too detailed for consideration here).

Some of the control structures common to many algorithms are (a) loops and (b) conditional execution (if-then-else). Although conditionals have not yet been incorporated into the NLC design, the present program is capable of handling a few forms of looping patterns. For instance, a user may type a sequence of commands to be executed on a sample member of the data structure and then instruct the processor to treat those commands as a loop to be applied to further data. Thus, NLC will currently accept

```
"Pick a row."
...
"Repeat for the other rows."
```

Procedure definitions in the NLC environment are thought of as word definitions for the natural language processor. That is, suppose that the user wants to define a procedure for the simple grade averaging task of Section 1. Suppose the user wishes to name the procedure "finalaverage" and has called the grades matrix CS105. Then he or she would create the procedure by defining "finalaverage" as a new imperative verb. This is accomplished by typing the following:

```
"Define a way to finalaverage CS105."
... (procedure body from Section 1) ...
"End the definition."
```

This new imperative verb "finalaverage" will be subsequently available and processed (from the user's point of view) like any previously known imperative. Thus the user might say

```
"Finalaverage CS51 and CS52."
```

or, if read and print imperatives are available,

```
"Read file COURSE, finalaverage it,
 and print the results."
```

More complicated word definitions appear in Biermann and Ballard[78] and Ballard[79].

Mechanisms for defining words other than imperatives are being designed. Procedural facilities affect linguistic considerations because the existence of user-defined procedures requires recognizing user-defined words and some special phrasings. For example, a procedure called "glob" which is to return a numeric value may be invoked by the noun phrase "the glob of ..." and also requires that NLC be able to parse the relative clauses "which was globbed" and "which the last command globbed".

4.0 Conclusions

Natural language programming has seemed a remote possibility in recent years because of difficulties related to problem representation and computational linguistics. The NLC design attempts to bypass some of these problems by (1) making the world of data structures the sole domain of discourse and thus requiring users to define the desired computations in terms of them; and (2) presenting the user with a high-quality linguistic facility and an environment

within which system response can be continuously observed to verify correct action or to discover and correct errors.

This paper has presented an overview of the NLC system and discussed issues related to the knowledge base required. When evaluating efforts in computational linguistics, two questions are particularly important. First, does the system correctly respond to inputs from users not familiar with the design of the system? This is especially important for NLC, which purports to allow users to solve problems in their natural language. In a recent test of the system (Biermann, Ballard and Holler[79]), subjects drawn from a first-semester programming course were asked to solve two problems of intermediate difficulty after only 50 minutes of training, which involved reading a tutorial and doing some practice exercises. Of the 1581 sentences typed during the subsequent testing session, 81 percent were correctly processed. Only one of the 12 subjects was unable to use the system to solve the linear equations problem assigned.

A second question of interest in evaluating a natural language system is whether the design is sufficiently general to allow for (a) adding new capabilities within the selected domain; and (b) extending the techniques developed for the prototype to new domains. A primary goal in designing NLC was achieving a clear modularization of the features being implemented. In this way, the interactions which result when new features are introduced are minimized, thus avoiding the potential exponential increase in processing time as further capabilities are provided. One of the principle directions of research deals with providing control structures (e. g. looping) and word definition capabilities (namely through procedures), each of which is orthogonal to the sentence-by-sentence processing needs of the existing software and therefore unlikely to affect current arithmetic functions.

Another direction of research deals with examining the linguistic structures and semantics of new domains to which the overall NLC design might be applicable. The parser and most of the scanner are entirely domain-independent and thus should be immediately effective for a second prototype. Extensions to the grammar to allow new types of phrases possibly required by new domains will be made by writing further networks as described in Section 2.2.2. The present semantics module, together with its specialty routines for word classes, should also be extremely valuable for new domains. Of course, low-level routines representing precise knowledge of the meanings of the new modifiers will be necessary. A more pervasive modification will be in designing new internal data structures corresponding to the nouns and relationships of new domains. Finally, the matrix computer module, including most of the output display routines, will necessarily be entirely rewritten to accomodate the new entities.

Probably the most immediate extentions of the present NLC semantics processor would be into similarly structured domains. In particular, the hierarchy of the matrix domain, where matrices "contain" rows and columns, which in turn "contain" elements, may be important in selecting the next domains for consideration, since the way in which modifiers of aggregate (or collective) objects depend upon their application to containing members is often independent of the particular nouns and modifiers involved. For instance, the tree-like structure of most corporations (employee-section-division), or of many database file systems (e.g. the sub-directories of MULTICS or UNIX), may be amenable to hierarchical treatment similar to that for the matrix domain.

References

Ballard B. W., Semantic and Procedural Processing for a Natural Language Programming System (Ph.D. Dissertation), Technical Report CS-1979-5, Duke University, April, 1979.

Biermann, A. W. and Ballard, B. W., "Toward Natural Language Programming", Technical Report CS-1978-11, Computer Science Department, Duke University, November, 1978.

Biermann, A. W., Ballard, B. W., and Holler, A. M., "An Experimental Study of Natural Language Programming", Technical Report CS-1979-9, Computer Science Department, Duke University, July, 1979.

Brown, J. S. and Burton, R. R., "Multiple Representations of Knowledge for Tutorial Reasoning" in Bobrow, D. G. and Collins, A., ed., Representation and Understanding, Academic Press, New York, 1975.

Caramazza, Alfonso, "Parallel Function Strategy in Pronoun Assignment", Cognition, June, 1978.

Green, C. C., Gabriel, R. P., Ginsparg, J. M., Kant, E., Ludlow, J. J., McCune, B. P., Phillips, J. V., Steinberg, L. I., Tappel, S. T., Westfold, S. J., "Progress Report on Knowledge Based Programming", Computer Science Division, Systems Control, Inc., Palo Alto, California, September, 1978.

Heidorn, George E., "Automatic Programming Through Natural Language Dialogue: A Survey", IBM J. Res. Develop., July, 1976.

Heidorn, George E., "English as a Very High Level Language for Simulation Programming", SIGPLAN, April, 1974.

Heidorn, George E., Natural Language Inputs to a Simulation Programming System. Naval Postgraduate School, October, 1972.

Hendrix, Gary G., "Human Engineering for Applied Natural Language Processing", Proceedings Of 5th International Conference On Artificial Intelligence, Volume I, August, 1977.

Hobbs, Jerry R., "From 'Well-Written' Algorithm Descriptionss Into Code", Research Report 77-1, Department of Computer Science, City College, CUNY, July, 1977.

Kay, M., "Morphological and Syntactic Processing", in Linguistic Structures Processing, A. Zampolli, ed.

Kimball, J., "Seven Principles of Surface Structure Processing in Natural Language", Cognition 2 (1), 1973, pp. 15-47.

Minsky, M., "A Framework for Representing Knowledge", in The Psychology Of Computer Vision, Winston, P. H. ed., McGraw-Hill, New York, 1975.

Petrick, S. R., "On Natural Language Based Computer Systems", in Linguistic Structures Processing, A. Zampolli, ed.

Sammet, J. E., "The Use of English as a Programming Language", CACM, March, 1966.

Schank, R. C., "Identification of Conceptualizations Underlying Natural Language" in Schank and Colby, Computer Models of Thought and Language. W. H. Freeman and Company, San Francisco, 1973.

Simmons, R. F., Personal Communication at TINLAP-2 Conference, Univ. of Illinois, July, 1978.

Winograd, T. Understanding Natural Language, Academic Press, New York, 1972.

Woods, W. A., "A Personal View of Natural Language Understanding", in "Natural Language Interfaces", Waltz, D. L., ed., SIGART, February, 1977.

Woods, W. A., Kaplan, R. M., and Nash-Weber, B., The Lunar Sciences Natural Language Information System: Final Report. Report Number 2378, Bolt, Beranek and Newman, Inc., Cambridge, Mass., 1972.

Woods, W. A., "Transition Network Grammars for Natural Language Analysis", CACM, October, 1970.

Zampolli, A., ed., Linguistic Structures Processing, North-Holland Publishing Company, Amsterdam, Holland, 1977.