COMPILER-INTERPRETER FOR SURVEY DATA ANALYSIS LANGUAGE

Babubhai V. Shah, Research Triangle Institute
P. O. Box 12194, Research Triangle Park, NC 27709

KEY WORDS: survey data analysis, statistical software, compiler, interpreter

ABSTRACT

A large number of statistical software packages are available. There is still a considerable time gap between the discovery of new statistical theory and creation of a procedure in a software package. RTI has developed SUrvey DAta ANalysis (SUDAAN) language that permits a statistician to translate statistical formulae to a computer program. SUDAAN language has a functional syntax with algebraic operators known to all statisticians. The arguments of the functions are algebraic abstraction of data structures for most statistical analyses. The abstract objects include input and output data sets and other intermediate calculations. The user needs to input only functional definitions of the needed results. This paper describes the current status of SUDAAN implementation. Once the complete system has been implemented, the procedures will be written in SUDAAN language.

1. INTRODUCTION

In recent years, there has been a tremendous growth in statistical software. However, the development of a new program that incorporates recent advances in statistical theory requires several man months. If the number of potential users is not large, development of such software is not commercially viable. Currently, a few programs for computing sampling error of a mean (the simplest of statistics) for data collected in a multistage sample survey are available.

Scientists at the Research Triangle Institute (RTI) have been developing and maintaining software for survey data analysis for the past 18 years. An application language that will permit rapid development of statistical software (primarily aimed at analyzing multistage nested survey data) was proposed by Shah (1978 and 1979). Over the years, RTI has continued its research effort to investigate ways in which the proposed concepts may become practically useful.

Since 1987, RTI has been under a Public Health Service (PHS) contract to develop a comprehensive software package for analyzing survey data. Rather than modify the "old" software, we decided to redesign the software, based on our past experiences. In this process, the SUDAAN language concepts are becoming a practical reality. The new procedures are being written in a SUDAAN prototype language and are discussed by LaVange et al. (1988).

The SUDAAN language is taking shape as an independent application language that will permit rapid development of new software. This paper presents the language in its current test version. Comments and criticisms will greatly influence the future versions of the language.

In Sections 2 and 3, we present data abstraction and functions that are basic to SUDAAN language. The syntax and semantics for the language are presented in Sections 4 and 5. An example of a procedure written in SUDAAN language is presented in Section 6. The remaining sections highlight implementation, advantages, and limitations.

2. DATA ABSTRACTION

The expressive power of very high-level languages is a result of special functions (operators) and data objects (operands). Most statistical languages have functions (procedures) that accept as argument a data set that is a sequence of identically structured vectors. APL by Iverson (1962) supports functions (like inner product, outer product, and reduce) whose arguments are multidimensional arrays, as well as other primitive functions. Backus (1978) proposed a formal system of functional programming language. The concept of data structure used in SUDAAN is a generalization of the data structure in APL and most statistical languages such as SAS.

The basic data structures, which are described as sequences of multidimensional arrays each having hierarchically nested identifiers, may be thought of as trees whose root segments are multidimensional arrays. These special data structures are trees whose terminal nodes are all multidimensional arrays that have the same description and dimension. Secondly, each of the data structures is nested up to the same depth or has an identical number of nesting identifiers. To avoid confusion with the general term data structure, a data structure for SUDAAN is referred to as a Balanced Array Tree (BAT).

Hierarchical ordering within each hierarchy may be chronological as in year, month, and day; or alphabetical as in states, counties, and enumeration districts; or arbitrary as needed by the user. The basic data structure provides unified representation for all entities (scalars, vectors, arrays, matrices, and sequences of these and other most commonly used data sets) in statistical analysis. The concept of a data structure may be regarded as a generalization of a similar concept in APL (Iverson 1962). The basic entity of APL, for instance, is a sequence of "one" multidimensional array with "zero" identifiers, and treated as a special case of BATs.

As an example of BAT, consider data consisting of a student's test scores. The sample selection was done in four stages: states, school districts, schools, and students. The survey data file can be looked upon as a sequence of vectors of test scores. Each vector (corresponding to a student) is uniquely identified by the four hierarchical identifiers (names or code numbers) of the state, school district, school, and student, respectively. Now consider within each school the variance-covariance matrix of test scores between the

students. Data consisting of all "within school between students variance-covariance matrices" are sequences of matrices with only three hierarchical identifiers: state, school district, and school, for each matrix. The pooled variance-covariance matrix (across all schools) can be regarded as a sequence of only one matrix with zero (no) identifiers. All three conceptual files are "hierarchically consistent" BATs because they have the same hierarchical frame.

Complex data can be handled efficiently by defining multiple BATs. For example, a transportation survey of households may contain information about: (a) households, (b) persons in each household, (c) trips made by each person as a driver, and (d) cars owned by each household. The entire set of data can be conveniently divided into four files, each constituting one data structure: (a) households, (b) persons, (c) trips, and (d) cars. These data structure definitions require any language implementation to deal with several BATs that may be hierarchically consistent. The SUDAAN language also includes objects that are scalars and vectors as in most languages.

3. FUNCTIONS

A program written in SUDAAN language permits a user to define a set of output data sets from a given set of input data sets. The program is a sequence of functional specifications leading from input to output. The definitions of the functions and objects are general enough to include almost all input, intermediate computations, and output encountered in survey data analysis. More functions may be added as needed.

A basic function is an operator that defines a new BAT from one or more BATs. For example, C = A + B defines a new BAT, C, such that it has the same structure as A and B and every element of C is a sum of the corresponding elements of B. The function is not valid if A and B do not have identical structure. The design of SUDAAN can permit definition of new functions and, hence, it is not necessary to define a complete class of all possible functions at the outset. Initial definitions need to include only a limited number of basic functions needed in performing most of the computations in statistics. A list of a few basic functions is presented in Appendices A and B. Many more functions can be defined using the general concepts of Backus (1978) where arguments of the functions may be other functions.

There are three different types of functions defined for this system: (a) scalar functions, (b) input/output functions, and (c) functions on BATs.

Most scalar functions can be used in the same fashion as the operators. For example, Z = F(a,b) defines an array Z whose elements are formed by applying function F to the corresponding elements of arrays a and b.

Then there are functions that define new BATs as a function of other BATs. The most useful one for statistical computation is "sigma." The key argument for sigma is the number of nest values retained in the output. For example, if X represents a BAT that has three nest levels, region, state, and county, the data consists of population totals by age, sex, and race for each of the counties. Then

```
Y = sigma(X,2);
```

defines a new BAT that has two nest levels, region and state, and its elements are obtained by summing all the corresponding elements of all counties in each state, thus resulting in population totals by age, sex, and race for each state. Similarly,

```
Y = sigma(X,0);
```

represents one table for the nation.

4. SYNTAX

A program written in SUDAAN language consists of statements. Each statement has the form:

```
a = f(x,y,z);
```

where "a" is a new identifier, "f" is a function known to SUDAAN, and x, y, z are the arguments to which function "f" is applied. The argument x, y, or z may be an expression. For example:

```
a = f(p*q+r, y, z);
```

or one may define

```
B = M*n-k;
```

These are most commonly used algebraic expressions, except for the ending ";" semicolon, which is used by SUDAAN language to recognize the end of a statement.

The right-hand side and "=" signs are absent when an output function is used. For example:

```
Print1(A);
```

```
Outfile("Bfile", B, "Means", SASfile);
```

Iverson (1962) has proposed syntax that has no precedence and interprets statements from right to left. Backus (1978) has proposed functional syntax that is similar to LISP. SUDAAN is an application language for statisticians. The syntax for SUDAAN is the functional syntax used by most statisticians to write down mathematical formulae. A complete formal syntax using diagrams is presented in Appendix C.

5. SEMANTICS

All functions have syntax of the form name (arg1, arg2,...), where "name" is the name of the function and arg1, arg2,... are expressions to be taken as arguments of the function. For example, SIGMA(x) defines the sum over all members of a BAT named "x".

While arguments must be given in a fixed order, SUDAAN language permits a great deal of flexibility to the user. Typically, functions will have a few basic arguments (supplying the data on which the function operates). Such arguments must always be supplied by the user. There may also be several optional arguments, which usually give special features or options to control the function in more detail and may be omitted. The function will assume some prescribed default values for the omitted arguments. For example, SIGMA (x) is equivalent to SIGMA (x,0,"+"). To obtain the maximum of x, one must say SIGMA (x,0,"max").

The list function denotes sequences or collectives, i.e.:

```
vlist = list (age, sex, race)
alpha = list (2, 3, list (4, 3, 2, 5), 7, 3).
```

The operators +, -, *, /, and ** when used between two arguments will be interpreted as a term-by-term operation. For example, the equation

```
C = a+b;
```

is equivalent to

$$C_{ij} = a_{ij} + b_{ij} \text{ for all } i, j.$$

The logical operators will be denoted by the following symbols

$$<,\ >,\ <=,\ >=,\ ==,\ !=,$$

for less than, greater than, less than or equal to, greater than or equal to, equal to or not equal to. The Boolean operations "not," "and," and "or" are denoted by "!", "&", and "|" respectively.

The operator precedence order from highest to the lowest is as follows:

1. ** : exponentiation
2. ! - : not, and unary minus
3. * / : multiplication, and division
4. + - : plus, and minus
5. <> <= >= != == : all relational operations
6. | & : Boolean or, and "and" operations.

The functions are defined with arguments that are either a BAT, scalar, or a scalar function and usually define a new BAT. If one of the arguments is a list, then the function is applied for each value in the vector for that argument while keeping other arguments unchanged. The result is a vector of BATs. The number of BATs is equal to the size of the vector. For example:

```
y = SIGMA (X, LIST (0, 1, 2))
```

is equivalent to

```
y1 = SIGMA (X, 0)
y2 = SIGMA (X, 1)
y3 = SIGMA (X, 2)
y  = list (y1, y2, y3).
```

If two or more arguments are vectors (or trees), they must conform with each other and the result is similar to term-by-term operation. For example:

```
y = SIGMA (list(x1, x2, x3), list (0, 1, 2))
```

is equivalent to

```
y = list (SIGMA (x1, 0), SIGMA (x2, 1), SIGMA
    (x3, 2)).
```

Furthermore, the user will be able to define higher level functions using functions already defined. Macro definitions will enhance the facility available to the user. These will be explained in sections on user interface.

A function or operation on two bats is defined for two BATs, with different nest values, by (logically) replicating the members of the BAT with smaller numbers of nest values as needed. As an example consider two BATs. "A" has three nest values: region, state, and county. "B" has two nest values: region and state. Each member of the BATs represent two-dimensional table of total population by sex and race. The equation

```
C = A*100/B;
```

defines for each county the proportion of the state's population by age and sex group. This definition implies replication of a two-dimensional table for a state in BAT "B" for each county in BAT "C" belonging to the same state.

SUDAAN semantic interpreter makes such extensions, when needed. It also converts Booleans to integers, integers to reals and conversely as needed.

6. PROCEDURE FOR RATIO ESTIMATES

This example illustrates a program that requires several hundred lines in most conventional languages such as FORTRAN or PASCAL yet requires only 24 lines in SUDAAN. Furthermore, the program is easy to understand and maintain. The words denoted by capital letters are key words or internally defined objects and the words in lower-case letters are names defined by the user.

```
 1.  nestvec = LIST ("STRATUM", "PSU");
 2.  data = INFILE ("myfile", nestvec);
 3.  X = SELECT (data, LIST ("x1", "x2", "x3",
     "x4", "x5"));
 4.  Y = SELECT (data, LIST ("y1", "y2", "y3",
     "y4", "y5"));
 5.  valid = x!=MISSVAL & y!=MISSVAL;
 6.  subgroup = EFFECTS (LIST ("race", "sex",
     "age"), LIST (3, 2, 5));
 7.  tab1 = LIST (1, 2);
 8.  tab2 = LIST (1, 3);
 9.  tab3 = LIST (2, 3);
10.  tablist = LIST (tab1, tab2, tab3);
11.  tab = CROSSP (subgroup, tablist);
12.  nsum = SIGMA (CROSMUL (valid, tab));
13.  wsum = SIGMA (CROSMUL (valid*w, tab));
14.  xsum = SIGMA (CROSMUL (valid*w*x, tab));
15.  ysum = SIGMA (CROSMUL (valid*w*y, tab));
16.  ratio = ysum/xsum;
17.  zhij = (y-ratio*x)*w/xsum;
18.  npsu = SIGMA (CONSTANT(1,2),1);
19.  zhi = SIGMA (zhij, 2);
20.  zh = SIGMA (zhi, 1);
21.  var = SIGMA ((npsu*zhi-zh*zh)/(npsu-1));
22.  se = sqrt(var);
23.  names = list ("sample size", "Popn.
     Est.", "Ratio", "Std. Err.");
24.  print1 (list(nsum, wsum, ratio, se),
     names);
```

Statements 1 through 4 define data, numerator, and denominator variables. Line 5 is a test for valid data; it is a vector of five (0,1)s for each record. Statement 6 defines three dummy vectors for race, sex, and age, respectively. Statements 7 through 10 create a list of three tables. The function CROSSP defines three table cells for each record in data. Lines 12 through 15 produce sums for each of these table cells for each of 5 variables. Note that the result of the function CROSMUL is the argument of SIGMA. Statement 16 is the ratio estimate; 17

represents Taylorized deviation. Statements 18 through 22 are direct translation of formulae for computing sampling variance for data from stratified random samples where PSUs are sampled with replacement. The last two statements request the printout of the results.

7. EXECUTION STRATEGY

Because there are no control specifications in SUDAAN, there are many other alternative ways of executing the program. A simple way is to execute each function independently by reading in arguments and storing results on a disk or a tape. This approach is very inefficient but it makes it feasible to execute the program on a machine with limited memory size, although it may require a huge amount of disk and tape storage. Alternately, all data and computations may be kept in memory but most machines may not have a huge memory.

The practical approach is to read one data record at a time and do all the data processing pertinent to that record. Under this approach, it may be necessary to read data twice in some cases. In rare cases, one may need to read data three or more times.

The implemented strategy is based on the following rules.

1. If the arguments of a function are two (or more) BATs with a different number of nesting levels, then the BAT with a smaller number of nesting levels must be saved and read for execution of this function, unless the BAT to be saved is an input structure or is dependent on input structures whose number of nesting levels is the same.
2. The phases will be constructed collecting output that require common inputs after substitution described in Rule 1.
3. If some other BATs (which are not saved) are required in more than one phase, the set of function calls to generate them will be repeated in both phases.
4. The allocation of in-core memory for data will be limited to that needed to store, at most, one member of a BAT.
5. Some BATS are needed only temporarily to generate other BATS. Such intermediate results may not require any core.

The analysis of data flow is carried out to implement the strategy.

8. DATA FLOW ANALYSIS

The syntax analyzer will convert user input to an internal table of functional specifications with results and arguments.

The data flow analysis involves various steps.

1. Identify each data structure to determine if it is a constant (or self defining), an input to be read from file, or an output data structure.
2. Performing a backward scan from output to input structure will reveal if there are any redundant structures being defined. If so, the function specifications for generating these may be deleted from the list.
3. If there are any specifications of the same function with an identical set of arguments, then the results of the later function may be equated to those of the former and the corresponding function call eliminated.
4. Identify alternate ways to develop blocks of functions or phases for execution of the tasks.
5. Determine blocks of functions that can be performed simultaneously and those that require separate phases.
6. Identify data to be passed from one phase or block to another phase or block and add instructions to write and read these data.

9. MISSING VALUES

In SUDAAN users can explicitly check for missing values by using comparison operations. For example: x ! = MISSVAL. The constant MISSVAL is always defined by SUDAAN. Its internal value is not relevant.

If arguments are invalid, all scalar functions or operations result in missing values. The division by zero results in a missing value. The square root of a negative number is a missing value.

All scalar functions or operations result in missing if one of the arguments is missing. For example, x+y, x*y, or x/y is missing if either x, or y has a missing value.

The scalar functions or operations are treated as missing values when used as an argument of generalized sigma. The generalized sigma is defined by induction. The generalized sigma for $x_1, x_2, \ldots x_n$ with respect to the function max is defined as follows: Let Y_i be the result after processing $x_1, x_2, \ldots, x_i$ then

$$Y_i = \max(y_{i-1}, x_i);$$
$$Y_0 = \text{MISSVAL};$$

If one of the arguments to max is missing, then the function returns the other argument. Thus, the result is missing only if all arguments are missing. In general, the cumulative result is maximum X value among the x_i's that have a valid value.

The standard sigma is defined with the operation "+", or sum, and results in total over valid values of x.

In summary, the missing values are consistently treated and follow the following rules:

1. Explicit constant MISSVAL is defined and is used for comparing or assigning missing values explicitly.
2. Invalid arguments or missing value arguments result in missing values.
3. In sigma operations, if one of the arguments is missing, the result is the value of the other argument.

With these simple rules, it is easy to check what each SUDAAN program does with missing values.

10. ADVANTAGES

The SUDAAN source code will be provided as part of the documentation. By referring to the SUDAAN program for a statistical procedure, analysts will be able to determine exactly what underlying mathematical formulas are employed. When statistical research indicates new or improved formulas for a given analysis, the

formulas can be readily translated into the SUDAAN language to yield a working computer program.

In writing a SUDAAN program, users define only the computational objects through function calls. Users need not specify how the calculations will be carried out. Increased reliability is achieved through checks built into the system to ensure that the definitions are meaningful and that the objects required as output can be generated. The system is free to select the optimal computational algorithm given the dataset and available computing resources. The system also provides tremendous feasibility with respect to input and output. All input/output objects are defined as BATs. Any intermediate BAT can be saved or displayed at any time in the program.

One advantage of SUDAAN is its modular independence. Because the system functions (including binary scalar functions) are only interpretations of the user's definitions of data structures, their implementation is independent of the supervisor; more functions can be added without rewriting any other components of the system. SUDAAN functions, as well as the supervisor, are totally independent of all binary functions. Thus, the system has total freedom for expansion in terms of its vocabulary and users. All improvements to the supervisor will be available to all SUDAAN programs without any rewriting.

The ability to handle several data sets simultaneously will eliminate the need for complex data base management systems. This may be a considerable advantage when dealing with large volumes of data.

The requirements for conformity among arguments for each function or operation are checked and errors are reported. Such error checking is not possible in conventional programming languages.

11. LIMITATIONS

The SUDAAN programmer is confined to the available vocabulary of functions. Computations that require new functions of BATs may have to wait until such functions are implemented in SUDAAN. Over time, as more and more functions are implemented, this limitation will not be critical. The modular design of the system permits new functions to be added to the system without recompiling the entire system. Only the new function and its interface need to be compiled. It also is possible to test the new function independently before incorporating it into the system, thereby greatly facilitating the addition of functions to the system library on an as-needed basis.

Computations that cannot be represented as BATs will not be feasible in SUDAAN. The user must define all computations as functions of input BATs or data objects. The SUDAAN system, therefore, is not for general programming use but should prove reasonably adequate for statistical analysis and, in particular, survey data analysis.

REFERENCES

BACKUS, J. (1978), "Can Programming be Liberated from the Von Neumann Style? A Functional Style and Its Algebra of Programs," **Communications of the ACM**, 21:8, 613-641.

BARR, ANTHONY J.; GOODKNIGHT, JAMES H.; SALL, JOHN P.; AND HELWIG, JANE T. (1976), **A User's Guide to SAS 76**, North Carolina State University, Raleigh, North Carolina.

IVERSON, K. E. (1962), **A Programming Language**, Wiley and Sons Publishing Company.

LAVANGE, LISA M. AND SHAH, BABUBHAI V. (1988), "A Comprehensive Software Package for Survey Data Analysis," **Fourth Annual Research Conference Proceedings**, Bureau of the Census, 327.

SHAH, B. V. (1978), "SUDAAN: Survey Data Analysis Software," **Proceedings of the Statistical Computing Section, American Statistical Association**, 146-151.

APPENDIX A
Selected Operators and Scalar Functions

Syntax	Results
exp(x)	xth power of natural number "e"
ln(x)	Natural logorithm of x
-x	Negative of x
x_1+x_2	Sum
$x_1<=x_2$	1, if x_1 is less than x_2; 0 otherwise
x_1**x_2	x_1 to the power x_2; if $x>=0$; missing value, if $x_1<0$.

Note: if either x, x_1, or x_2 has missing value, then the result of any function has missing value.

APPENDIX B
Selected BAT Functions in SUDAAN

The function infile (filename, nestvar, filetyp) defines a BAT that is read form input "filename."

The function effects (data, varlist, levelist) define a list of BATS corresponding to varlist. EAch BAT is conversion of a categorical variable to a dummy (0,1) vector, according to the number of levels.

The function matmul (A,B) defines generalized matrix multiplication valid when A and B may have more than two dimensions. This is similar to the inner product in APL. The number of levels for the last dimension of every multidimensional array of A must be equal to the number of levels for the first dimension of the corresponding member of B.

The function pseudoinverse (A) defines the pseudoinverse of every matrix defined by the last two dimensions of every array in A.

The function Sigma (A, nestnum) defines the cumulative sum over arrays with the same values for the first nestnum nesting variables. The resulting BAT has nestnum nest variables. The input must have at least nestnum nest variables.

The function print (BATlist, namelist, declist, stylename) requests printing of all

BATs in the BATlist with reference names in "namelist," with the precision as in declist and format specified by style name.

APPENDIX C
SUDAAN Syntax

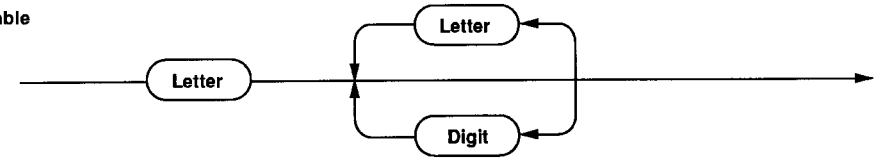
In SUDAAN syntax diagrams, a digit represents any one of the decimal characters 0 through 9. A letter can be any lower-case alphabet a through <, upper-case A to Z, or underscore "_".

The function identifier includes any SUDAAN function that has been implemented.

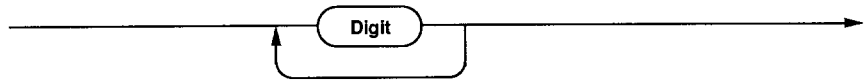
The syntax diagrams which follow define valid syntax for SUDAAN language.

Syntax Diagrams

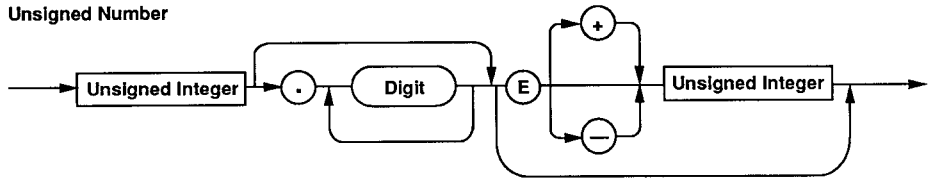
Variable



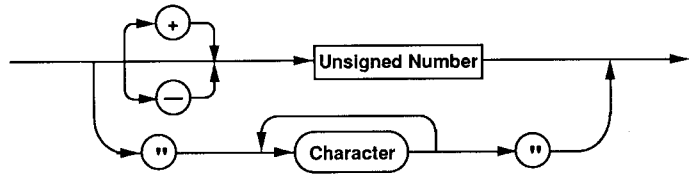
Unsigned Integer



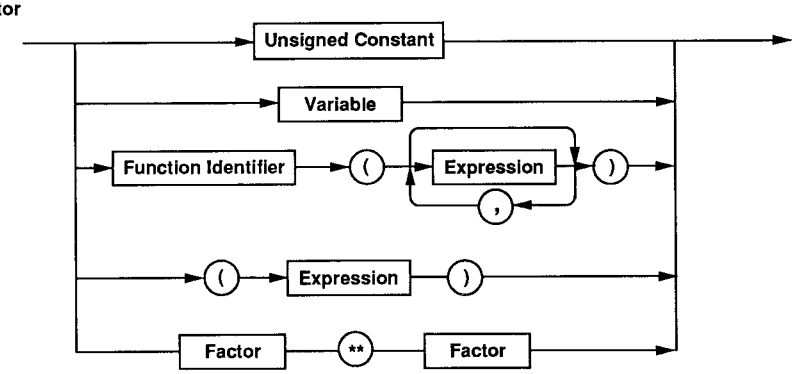
Unsigned Number



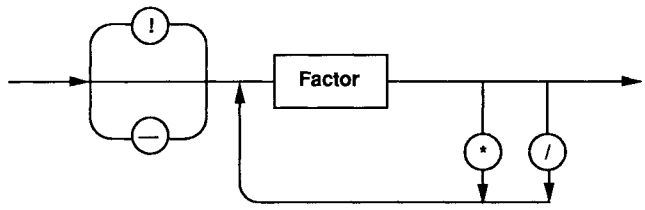
Constant



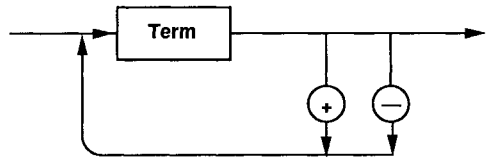
Factor



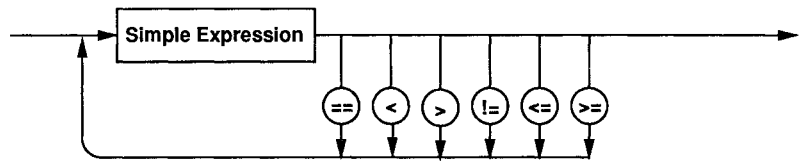
Term



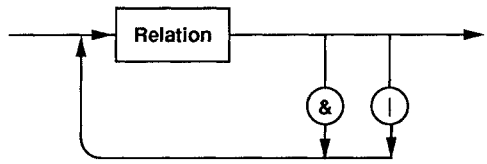
Simple Expression



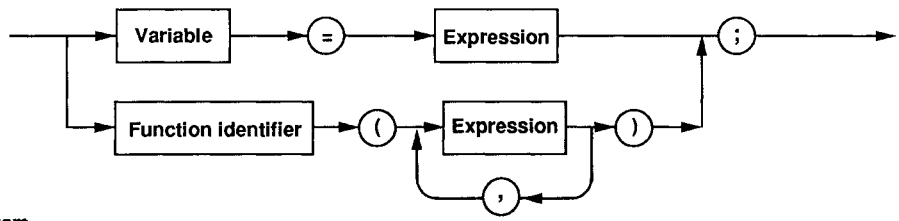
Relation



Expression



Statement



Program