VENUS: A Small Interactive Nonprocedural Language

HOWARD F. MATTHEWS

The Boeing Company, Seattle, Washington

For this paper, the subtitle terms are defined as follows:

INTERACTIVE: The language processor and the user are engaged in constant dialogue throughout the solution process.

NONPROCEDURAL: The input to the processor consists of a problem definition and a request for the solution, rather than a solution procedure.

The guidelines used in the development of VENUS were chosen to give the scientifically oriented nonprogrammer a tool which could be quickly learned and easily used. The syntax was designed for incremental learning, so that what the user did not know would not hurt him.

Since the predicated users are well acquainted with the notation and meaning of algebra, it was used as the model. Most computer languages use considerably modified algebra syntax, but most *do not retain the semantics of an equation*; that is, "$A = 2 * B$" usually means "when, in the sequential execution of statements, this statement is executed, then set the computed value $2 * B$ into storage cell A." It is usually assumed that somewhere previously B as acquired a meaningful value. VENUS accepts "$A = 2 * B$" as meaning "the value of *A at all times* should be twice the value of B at that time." *Should the value of B change, the value of A will be automatically updated.*

There are two types of statements in VENUS. A *definition* is an equation defining a variable, and is terminated by an "!". A *query* is an expression terminated by a "?" and generally requests VENUS to evaluate and print out the current value of the expression.

The following is a short conversation with VENUS. The "←" is a cue character indicating that the system is ready for the next statement, and thus it identifies the user's input. Notice how the value of *A* changes as the variables *B* and *Z* are redefined.

```
←2+2?
   !(2+2)=4

←A=3*B+C!
   *OK*

←B=-5!
   *OK*

←A?
   ?PLEASE DEFINE WHAT THE VARIABLE "C"=

←C=Z+25!
   *OK*
   ? PLEASE DEFINE WHAT THE VARIABLE "Z"=

←A=?
   !A=3*B+C
   ? PLEASE DEFINE WHAT THE VARIABLE "Z"=

←Z=-4!
   *OK*
   !A=6

←B?
   !B=-5

←B=+5!
   *OK*

←A?
   !A=36

←Z=106!
   *OK*

←A?
   !A=146
```

In the current implementation on the PDP-6 time-sharing system, the only conditional semantics is the conditional expression of the form:

⟨expression 1⟩ [⟨logical expression⟩] ⟨expression 2⟩

The left bracket is read as "IF," and the right bracket is "ELSE." The conditional expression is treated as a single subexpression, and may be nested in many ways to depths of interest only to programmers.

This symmetric form is more readable in complex statements than the common IF ... THEN ... ELSE ... form; it seems almost as natural to use, and common algebraic notation is easily translated into it. For example,

$$f = \begin{cases} 12a, & \text{if } a < 6 \\ 17 - b, & \text{if } a \geq 6 \text{ and } b \leq 2 \\ 12\, a/b \text{ otherwise} \end{cases}$$

is written in VENUS as

```
F = 12 * A[A < 6]17 - B[B < = 2]12 * A/B
```

An extension for user convenience is the recognition of sets. A user may define a symbol as being a set of other symbols, which may be variables or other sets, e.g.,

```
SET2: A2, BETA, X
```

The values of the entire set may then be requested by

```
  (user)  SET2 ?
(VENUS)  SET2 :

          ! A2    = -32E6
          ! BETA  = 184.700
          ! X     = 1.98100
```

If in the course of calculation, some variable is undefined, VENUS simply asks the user to define it and continues when possible. This assures the user that nonexistent values are not used, as they might be in most procedural languages, and thus takes part of the burden of bookkeeping off the user. VENUS also checks for circular definitions such as

```
A = A + 1
A = 15 * B
B = A - 17,  etc.
```

Currently these are treated as errors, but in future versions it may be possible to detect and solve certain forms of simultaneous equations in this manner. To change any definition, the user merely enters the new definition. Since each variable may have only one definition, and since the user is communicating at all times with the VENUS processor rather than a compiled code as produced by most language processors, it is a trivial operation to identify and replace any definition. A user may retrieve the current definition of any symbol by typing

⟨symbol⟩ ⟨definition operator⟩ ?

The following is an example:

```
←A=?
    !A=2*B
←SET:?
    ! SET: A, BETA, SET2, GAMMA
```

Other features of the current implementation are:

(1) library functions and user defined recursive functions,

(2) expression substitution,

(3) the ability to switch particular values or variables from base 10 to any base from 2 to 36 for input or output form, and

(4) system keywords (preceded by a "$") which may be used for built-in constants (e.g., $TRUE, $FALSE, $PI, and $E) or for system commands.

Planned extensions are:

(1) the use of units such as FEET, LBS, etc. with automatic conversion,

(2) the addition of subscripts for array and matrix work, and

(3) the addition of iteration clauses and functions similar to the FOR, WHERE, SUM, and PRODUCT functions of the JOSS language.

The last two features would add considerable power to VENUS in a form very similar to common algebraic notation. For instance, the product of two matrices

$$G_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}, \qquad \text{for} \quad i = 1,2,\ldots,n, \quad \text{and} \quad j = 1,2,\ldots,p.$$

would be written

```
G[I,J] = SUM(K = 1 TO M,  A[I,K] * B[K,J]),
  FOR I = 1 TO N,     FOR J = 1 TO P
```

Conclusions

VENUS, by using the well known semantics of algebra, is easily learned, lends itself to quick problem solution, and fits very naturally into the conversational mode of problem solution. Although it does not exhibit the power of a procedural language like BASIC, it does fit many engineering computational needs in a manner requiring no programming sophistication of the user, and a minimum of typing for the nontypist. Yet it keeps the user in the problem solution "loop" so that trial and error methods are straight forward, parameter variation is natural, and redefinition requires no "editing" from the user. Subproblems or side-problems may be worked at any point. Often a set of equations may be transcribed almost directly from a handbook or engineer's notes to the computer with no "programming" involved. Chances for misunderstanding between the man and the machine are greatly reduced; and, this reduction is the main purpose of computer languages.