6. BRACES—$:B2(\&1), where \&1 is to be enclosed in curly braces.

7. GENERATE—$:B9(\&1 \&2 \&3), where \&1 is to be enclosed in the bracketed pair given by \&2 and \&3.

8. BRACKET—$:B1(\&1), where \&1 is to be enclosed in square brackets.

9. CIRCLE—$:C(\&1), where \&1 is to be encircled.

10. ORDINARY DERIVATIVE—$:D(\&1 \&2 \&3), where the \&3th derivative of \&1 with respect to \&2 is to be plotted.

11. DIAMOND—$:D1(\&1), where \&1 is to be enclosed within a diamond.

12. SIMPLE FRACTION—$:F1(\&1 \&2), where the ratio of \&1 to \&2 is to be plotted. No measurement of arguments is performed: they should have equal widths, no undershoots, and full height.

13. GENERAL FRACTION—$:F(\&1 \&2), where the ratio of \&1 to \&2 is to be plotted. The arguments can have different dimensions.

14. GRID—$:G(\&1 \&2 \&3 \&4), where a rectangle is to be drawn containing \&1 by \&2 boxes each with length \&3 and height \&4. Plotting starts and terminates at the upper left-hand corner of the grid.

15. HEXAGON—$:H(\&1), where \&1 is to be enclosed within a hexagon.

16. INTEGRAL—$:I(\&1 \&2 \&3), where \&1 gives the lower limit of integration, \&2 the upper limit and \&3 the variable of integration.

17. MIDDLE—$:M(\&1 \&2), where \&2 is to be centered in the box \&1 characters long. One starts in the middle of the left side of the box and ends in the middle of the right side.

18. PAREN—$:P1(\&1), where \&1 is to be enclosed in parentheses.

19. PARTIAL DERIVATIVE—$:P(\&1 \&2 \&3), where the \&3rd partial derivative of \&1 with respect to \&2 is to be plotted.

20. SQUARE ROOT—$:R(\&1), where the square root of \&1 is to be plotted.

21. SUMMATION—$:S(\&1 \&2), where the summation of \&1 to \&2 is to be plotted. Note that \&1 must contain the equal sign if desired, e.g., if one wants \&1 = \&2 below the sigma, \&1 should equal 'A = \&2.'
22. TIME DERIVATIVE—$T(\&1)$, where a dot is to be plotted above the argument.

\[ \left( \frac{A - E(\&1.1)}{2} \right) \]

\[ Y.E = B + 2\.1., - E.2. \]

23. PRODUCT—$X(\&1 \&2)$, where the product (II) is to be plotted from \&1 to \&2. \&1 must contain the equal sign if desired as for summation.

\[ A9(\&1.2. \&9.8.) \]
INTRODUCTION

The AMTRAN system (Automatic Mathematical TRANslator) is a multiterminal conversational mode computing system which enables the mathematically oriented user to interact directly with the computer in a natural mathematical language. The first version of AMTRAN was developed at the George C. Marshall Space Flight Center in Huntsville, and was implemented on IBM 1620 and 1130 computers and, as a time-sharing version, on a Burroughs 5500 computer. A modified 1620 console version is currently in use at the University of Georgia.

In connection with the project of implementing a multiconsole version on an IBM 1130 computer, the AMTRAN language has been revised and formally defined at the University of Georgia Computer Center. The following objectives have been of primary importance in the development of the AMTRAN systems:

First, the initial use of a computer by the mathematically oriented nonprogrammer and scientist should be made easy by using a simple language as similar to mathematical notation as possible. The language should be designed for incremental learning so that a user may successfully use the system without knowing all of its details.

Second, the system should provide powerful programming capabilities for the solution of medium and large scale problems and complicated algorithms by the more experienced user or professional programmer.

Third, since AMTRAN was conceived as a special purpose language for mathematical and scientific use, the system should provide more flexibility in programming, debugging, and turnaround time compared to conventional computing systems.

Fourth, the new design and definition of the AMTRAN language should have as few restrictions, exceptional rules to remember, and departures from the well-known semantics of algebra as possible without reducing the power of the system. The system should be fully recursive and there should be no practical limitation to the length of variable and program names, the number of defined variables, the dimensions of arrays, etc.

DEFINITION OF THE PROGRAMMING LANGUAGE AMTRAN

The BNF notation is the only widely used formal method for the definition of a computer language. It has been derived from strictly structural considerations. The use of bracketed English words as metasymbols has made the notation look less formidable but, at the same time, has introduced semantic aspects which originally were not present. As long as these semantic aspects served only to characterize necessary structural categories, the method was as originally intended. As soon as strict semantic categories were established, with no structural characteristics, difficulties arose. This can be explained by a simple example. The ALGOL 60 report states:

\[
\text{variable identifier} ::\:= \text{identifier} \\
\text{simple variable} ::\:= \text{variable identifier}
\]

The introduction of the categories \text{simple variable} and \text{variable identifier} is necessary to distinguish between a strictly formal \(\alpha\)-numeric string (\text{identifier}) and a special type of a variable. This distinction is based only on the meaning assigned to a particular string; the structure remains the same. Therefore, the difference is not really expressed by the above method since the distinction between the categories is left to the arbitrary interpretation of the metasymbols (\text{identifier}), and (\text{simple variable}) and is by no means formally defined. Omitting all these questionable 'semantic categories' would diminish the content of such a language definition considerably.
Realizing these difficulties, a new general method of formal definition of a mathematically oriented computer language was developed at the University of Georgia Computer Center. By introducing different levels of the language, one structural and several semantic levels, it is possible to distinguish between the structure of a language and the meaning attached to the structural elements. A large part of the semantics is systematized in notions like type, range, sign, dimension of numerical quantities, and binding power of mathematical operators. It is found to be sufficient to introduce a few well-defined semantic values.

Each structural element is assigned a semantic and dimensional characteristic which carries the information associated with the structural unit. A structural unit and its semantic and dimensional characteristic are thus combined to form a constituent of the language. The idea of the new method of definition is to describe the language by setting up production rules for constituents rather than for structural units by themselves. This systematization and formalization covers a much larger part of the language than mere 'syntax' definitions or the definition by BNF notation. Now it is possible to resolve the previous ALGOL example. The structure of \{simple variable\}, \{variable identifier\}, and \{identifier\} are the same, an alphanumeric string. But, since the semantic characteristics of a strictly formal alphanumeric string and of a simple variable are different, they form different constituents of the language. Therefore, the distinction between these categories is not left to the interpretation of the reader but is a part of the language definition.

**Absence of dimension statements**

Arrays are created and changed with complete freedom at run time.

Examples:

\[ X = \text{ARRAY} (0, 5, 5) \]

creates an array \( X \) with the values

\[ 0, 1, 2, 3, 4, 5 \]

\[ Y = X + 1 \]

creates an array \( Y \) with the values

\[ 1, 2, 3, 4, 5, 6 \]

even if \( Y \) had been a scalar or an array with another dimension before. Another operator to construct arrays is the concatenate operator \&.

\[ X = 0 \& 1 \& 2 \& 3 \& 4 \& 5 \]

creates the same array as array \( X \) in the previous example.

**Absence of declaration statements**

The type of a variable is automatically defined through the assignment statement until it is changed by another assignment statement. At execution time, the control routine for each operator checks the type and range, sign and dimension of the operands.

A new concept is used for the handling of integers. They are stored and treated internally as real numbers, but a special rounding routine preserves their integer status through any arithmetical manipulations. Every time an operator requires an integer argument, the system examines the real representation of the value of the operand to determine whether it represents an integer number; an error message is typed if it does not. Thus, AMTRAN will give the right results for \((-2.5)^3\) as well as for \((-2.5)^{2\sin(\pi/6)+0.5}\).

**Automatic array arithmetic**

The basic operators and functions mentioned in 3.1.1. and the relational operators can be used not only for scalars, but also on arrays. Thus, the user may compute directly with the numerical representations of functions without writing loops.

For example, the function

\[ y = \frac{2}{\sqrt{\pi}} e^{-x} \sin x \]
is represented in AMTRAN as

\[ Y = \frac{2}{\sqrt{\pi}} \exp - X \sin X \]

where \( X \) can represent an array of 100 equally spaced intervals generated by \( X = \text{ARRAY} (0, \pi, 100) \). The resulting function \( Y \) is represented by an array of 101 numbers, where each \( Y \)-value is the value of the above function for the corresponding \( X \)-value.

**Conditional operators:** IF, THEN, ELSE

**Relational operators:** GT, GE, EQ, LE, LT

**Boolean operators:** NOT, AND, OR

They are basically the same as in ALGOL 60 except that each IF has a corresponding FI (ALGOL 68 style) at the end of the conditional expression to avoid the dangling ELSE problem.

**Unconditional branch and loop**

The GO TO operator can be used for transfer of control to any numbered statement in a program. The argument of GO TO may be any expression which returns a scalar value. Non-integer values cause a warning message.

The REPEAT-operator is used to repeat a group of statements a specified number of times. These repeat-loops can be nested arbitrarily.

**Generality of operands**

As a general rule, every operand or parameter in AMTRAN can be an expression, but the result of this expression has to fulfill the semantic requirements its operator asks for. Example: The third parameter of the ARRAY operator (number of intervals) may be an expression, but the result has to be an integer with the dimension one.

**Fully recursive programming capabilities**

An example of an inherently recursive function is Ackermann's function \( A(M, N) \), defined over the positive integers and zero:

1. \( M = \text{IN} 1, N = \text{IN} 2 \)
2. \( A = \text{IF} M \text{EQ} 0 \text{ THEN } N + 1 \text{ ELSE IF} N \text{ EQ} 0 \text{ THEN } A(M - 1, 1) \text{ ELSE IF} A(M - 1, A(M, N - 1)) \text{ FI FI} \)
3. \( \text{NAME} A \)

Statement 1 picks up two arguments which have to be provided by the program call. For example, \( A(2, 4) \) will give \( M = 2 \) and \( N = 4 \) upon execution of statement 1.

**Powerful instruction set which can easily be expanded by the user**

The philosophy of AMTRAN is that the user should be given a powerful basic set of instructions which are intrinsic to the system, together with a disc file library of instructions which are actually routines written in AMTRAN. In addition, the user can define his own high level operators by writing special AMTRAN routines. Operators for automatic numerical analysis (integration, derivation), satisfactory for routine situations, are also included.

**ASK- and TEACH-operators**

The ASK-operator can be used to program a dialogue between the computer and the user. If the user is not satisfied with a system message, he can react with 'ASK,' and the system will respond with a more detailed message. This feature makes AMTRAN to a truly conversational system. The experienced user does not have to spend time running through the questions and answers, which are of great importance for the average and beginning AMTRAN-user.

The TEACH-program allows the user to learn how to use AMTRAN directly on the console. The new AMTRAN-user does not have to take a course in programming; he need not study a programming language or learn how to read computer outputs. He can get started with a simple teach program on the console in a few minutes without having to learn complicated rules. If there occurs a problem in using AMTRAN, the user can use the TEACH-operator and run the part of the teach program which refers to his problem.

**Call by symbol concept**

The call by symbol concept allows the passing of executable strings as parameters to subprograms. This symbolic expression can contain variables local to the calling program and variables local to the subprogram. Everytime the parameter is invoked within the subprogram, the symbolic expression is evaluated using the actual internal and external variables.
Three modes of operation

1. Execute mode: An interactive system must have an execution mode (or desk calculator mode) where each statement is executed immediately and control is returned to the keyboard. This is the default mode in AMTRAN.

2. Suppressed mode: The suppressed mode (delayed mode) allows the user to construct programs which are syntax checked and stored for execution at a later time.

3. Checking mode: AMTRAN has a third mode, the checking mode, which allows the user to execute parts of suppressed programs while they are being constructed. This is an important aid for online program construction.

Implementation on the IBM 1130 computer

This AMTRAN version was implemented on an IBM 1130 computer with 8K of core and a disc. A typewriter version is currently being tested, and a multiconsole version is under development.

Some of the goals for the implementation were high speed, fully dynamic storage allocation, powerful editing and checking capabilities, and a completely re-entrant structure for multiconsole use with short response time. The length of programs, the dimension of arrays, the ratio of program area to variable area, the number of defined variables and programs can be chosen with complete freedom as long as the available core storage, dependent upon implementation, is not exceeded.

A special monitor system, independent of IBM software, has been developed for a more efficient use of the disc to obtain short response times.

AMTRAN as a tool for pedagogic purposes

The interactive system AMTRAN is highly useful not only for research purposes but also as an educational tool. Lowering the level of difficulty in programming makes the computing facility available for students who are not basically interested in computer science but want to expand their understanding of mathematics or physics. Graphic display capabilities are very well suited for studying and demonstrating the behavior of functions. By interacting directly with the computer, the student also gets a better feeling for the kind of problems involved in programming a computer.

A multiconsole system eliminates keypunch problems, and there are none of the time delay, debugging, or control language problems usually found in batch mode.

COMPARISON WITH OTHER HIGH LEVEL LANGUAGES

A comparative study between AMTRAN and other high level languages has to be divided into two parts. Only language features can be compared with batch mode languages, whereas the whole AMTRAN-system can be taken into account for a comparison with other interactive systems.

Batch mode languages

Most likely, PL/1, ALGOL, or FORTRAN would be used to solve mathematical, technical or scientific problems in batch mode. A comparison with AMTRAN is not really feasible as the basic philosophy and design principles of batch mode languages are completely different from AMTRAN.

Since language development goes more and more in the direction of powerful general purpose languages, it becomes more and more difficult, time consuming, and cumbersome for the nonprogrammer to make the first step towards use of a computer. But even for the experienced user, the three languages mentioned above do not provide the convenience and facilities in programming that AMTRAN does. They need type and dimension declarations; the flexibility in changing types and dimensions at run time is lacking; and they do not have AMTRAN's array handling capabilities.

PL/1 with its default philosophy, its various types of storage allocation, and certain automatic array arithmetic features is close to AMTRAN's facilities and philosophy of programming convenience. On the other hand, it is inconvenient for the user to keep track of storage allocation problems in writing recursive or re-entrant programs or in using arrays with computed origin.

PL/1 is truly a general purpose programming language. It is designed for programming needs in science, engineering, data management, and business. AMTRAN, on the other hand, is a special purpose programming language for mathematical, scientific, and technical applications and has not been designed to compete in general with a language like PL/1. It is not intended to handle extensive data; therefore, it does not need powerful I/O-capabilities and sophisticated formatting facilities. But it can compete or even perform better within the limits of its special purpose.

Interactive systems

An interactive console system fills the gap between a desk top calculator and conventional batch mode