TRANSLATION OF ARTIFICIAL LANGUAGES BY

COMPILER PROGRAMS:

Research Report and Design for Future Languages

Robert F. Rosin

RAND Corporation
University of Michigan

SUMMARY

I. Purpose of the work.

This work was carried out primarily to investigate artificial languages from the standpoint of translation by machine. It was desired to learn what representations are necessary to make this task more easily accomplished. It was felt that this work would apply not only to translation of artificial languages but also to natural language translation by machine and other complex symbol manipulation programs as well. It must be pointed out that the problem under consideration was not the translation algorithm, but the code necessary to transmit this algorithm to the machine. For this reason two relatively uncomplicated languages were used.

As a by-product of this work it was hoped to show that the more flexibility with which a language is endowed, the more powerful programs it may produce. GAT was not intended to be a symbol-manipulator-compiler, but a compiler with enough power to be useful in both the educational and applied areas of a university computer installation. To ensure these ends a very simple, yet powerful tool was added; the ability to operate with (read, write, compare, etc.) alphabetic variables. This was facilitated by the internal structure of the IBM 650 with alphabetic and special character devices.

To make this work as pure a demonstration as possible, the translator was to be written in such a way as to use a minimum of externally produced subroutines. This meant that some operations which might be carried on very efficiently in some other languages (e.g. SOAP) would be time and space consuming in the translator. Due to the character of the GAT language these subroutines may be used as operators to some extent and allow for easy coding of conversions from one mode of internal storage to another.

It should be noted here that there was never any intent to produce a production level translator. Compilers today use storage less efficiently than human coders and also are not equipped to take advantage of all of the devices available to the human programer. Nevertheless, it was happily noted that the translation of arithmetic FORTRANSIT statements to the corresponding GAT statements was relatively fast. A representative time might be about ten statements per minute. This varies according to the length and complexity of the statement.

II. The translation process.

GAT is essentially an IT-type language with certain added features which increase its power. The ability to manipulate alphabetic variables and a more flexible subroutine format have already been mentioned. FORTRANSIT is the 650 version of FORTRAN which is essentially limited in several non-critical ways to enable it to be used on the 650.

Certain limitations of the 650 have caused the translator to be a limited version of the ideal program. Mainly, it has been found that 2000 words of storage are not enough to store the program, the variables which are used and the constants and linkages and subroutines which the compiled program needs to function. For that reason there is a large class of FORTRANSIT statements which are not treated though they are detected by the program and this is indicated in the output.

Most important among these is the DIMENSION statement which serves to indicate parameter storage for FORTRANSIT. Since this is not included in the translated output, subscripted variables and matrix variables are not handled properly. However, the treatment is coded and could be accomplished in a machine with greater than 2000 words of storage.

Arithmetic statements are translated as follows. Subroutine names, which end in "F" in FORTRANSIT, have their final character changed to "." to correspond with GAT notation. This also allows a programer skilled in the GAT language to code in FORTRANSIT but to use GAT library subroutines or to write his own GAT subroutines, which is a very simple process. Fixed point FORTRANSIT variable names, which must begin with the letters I through N, are translated into subscripted X variables in GAT. The remaining, which are floating point, are translated into subscripted C variables. The "* *" (exponentiation) in FORTRANSIT becomes P in accordance with GAT notation.

III. Conclusions.

A primary conclusion which may be drawn from this work is that it is quite feasible to write translators in compiler-type languages. The coding time for this program was quite minimal, and the entire project was carried out in spare time in a period of two and one half months.

The conclusions concerning the symbol manipulator language may be summarized as follows:

1. It would be convenient to be able to operate with variables of greater than 5(650) or 6 (704) characters in length. This is especially true when one approaches the problem of natural language translation.

2. It is necessary to be able to extract a group of characters from any part of a variable and operate on it and make decisions based on its content. This should be defined as a unary operator (as the sine function) and, thus, should be allowed as a part of any program statement.

3. Decisions should be based on a broad set of functions as is defined in the I.A.L.

4. Some sort of recursive list structure, such as is part of the Newell-Shaw-Simon I.P.L.'s should be available.

5. One should be able to define special characters within his program to facilitate translation.

6. The operation of concatenation (serial conjoining) should be easily expressible within the language.

There are two approaches to providing these features. The most desirable is the design of machines capable of carrying out these tasks internally. But, until such machines are available, languages utilizing these features can be developed.

A final result of this work lies in a better understanding on the part of the author of the complexities of compilers and the translation process. These realizations are applicable to writing natural language translators, artificial language translators (compilers), pattern recognizers (for what are patterns but multi-dimensional strings of characters to be deciphered) and other such programs. The ultimate aim is to produce a program which will allow specification of both source and object languages alone to produce a compatible translator between the two.