

AD 658029

RADC-TR-67-454

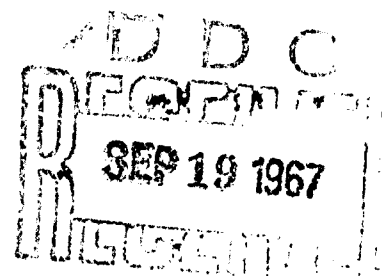


COMPILER GENERATION USING FORMAL SPECIFICATION
OF PROCEDURE-ORIENTED AND MACHINE LANGUAGES

Philip Gilbert
Measurement Analysis Corporation
William G. McLellan
Rome Air Development Center

TECHNICAL REPORT NO. RADC-TR-67-454
August 1967

This document has been approved
for public release and sale; its
distribution is unlimited.



707 000
Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, New York

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded, by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacturer, use, or sell any patented invention that may in any way be related thereto.

ACCESSION BY	
OPEN	WHITE SECTION <input checked="" type="checkbox"/>
DDP	PUFF SECTION <input type="checkbox"/>
UNRECORDED	<input type="checkbox"/>
JUSTIFICATION	
BY	
DISPOSITION/AVAILABILITY CODE	
INST.	AVAIL. and/or SPECIAL
1	

Do not return this copy. Retain or destroy.

COMPILER GENERATION USING FORMAL SPECIFICATION
OF PROCEDURE-ORIENTED AND MACHINE LANGUAGES

Philip Gilbert
Measurement Analysis Corporation

William G. McLellan
Rome Air Development Center

This document has been approved
for public release and sale; its
distribution is unlimited.

FOREWORD

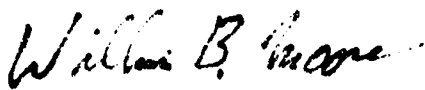
This technical report describes research accomplished under Project 4594 and is a revision of a paper presented by the authors at the 1967 Spring Joint Computer Conference held April 19th in Atlantic City.

The authors are indebted to Donald M. Gunn and Craig L. Schager, both for their significant contributions to this work and for their valuable suggestions regarding this paper.

This technical report has been reviewed by the Foreign Disclosure Policy Office (EMLI) and the Office of Information (EMLS) and is releasable to the Clearinghouse for Federal Scientific and Technical Information.

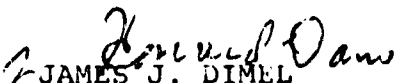
This technical report has been reviewed and is approved.

Approved:



WILLIAM B. MOORE
Chief, Recon Intel Data Handling Br

Approved:



JAMES J. DIMEL
Colonel, USAF
Chief, Intel & Info Processing Div.

FOR THE COMMANDER:



IRVING J. GABELMAN
Chief, Advanced Studies Group

ABSTRACT

A compiler generation system is described which is rigorously based and which allows formal specification both of the source (procedure oriented) languages and of the object (machine oriented) languages. An intermediate or "buffer" language, BASE, is interposed, reducing the required transformation techniques described. The system, so far, includes those elements in BASE necessary to produce ALGOL, FORTRAN, and JOVIAL compilers.

TABLE OF CONTENTS

<u>Section</u>		<u>Page</u>
1.	Introduction	1
2.	Theoretical Basis	2
3.	System Overview	3
4.	Compilation System Data Base	5
5.	POL Specification and Compilation System Operation	7
6.	Translation and Machine Specification	14
7.	Conclusions	17
	References	19

LIST OF ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1.	Overview of System	4
2.	Compiler Model Organization and System Data Entities	6
3.	Specification of the LEMMA2 Language	7
4.	Compilation of a LEMMA2 Program	9
5.	Preliminary Symbol Conversion	10
6.	Syntactic Analysis	11
7.	Performance of Internal Functions	12
8.	Output of a Code Sequence	13
9.	Some Typical Macro Definitions	16

1. INTRODUCTION

This paper reports on a recently developed compiler generation system which is rigorously based, and which allows formal specification both of source (procedure-oriented) languages (POLs) and of machine languages (MLs). Concepts underlying the system are discussed, an example correlating source language specification with system operation is given, and the status and potentialities of the system are discussed.

The crucial problem of compiler generation is the characterization of procedure-oriented languages; the process is of limited use unless such characterization allows machine-independent processing of programs in these languages (and hence allows invariance of the language itself from machine to machine). Our solution interposes between POL and ML a "buffer" or "intermediate" language, called BASE, thus reducing the required $\text{POL} \rightarrow \text{ML}$ transformation to two logically independent subtransformations:

(1) $\text{POL} \rightarrow \text{BASE}$ (called compilation)

(2) $\text{BASE} \rightarrow \text{ML}$ (called translation).

This arrangement isolates questions of POL characterization within the first transformation, and questions of ML characterization within the second transformation. BASE itself is an expandable set of non-machine-specific

operators¹, declarators, etc., expressed in a uniform "functional" or "macro" notation; the meaning or intent of such operators is arbitrary insofar as the compilation transformation is concerned. The POL \rightarrow BASE transformation may then be regarded as a machine-independent conversion, from a grammatically rich format to a simple linear format.

2. THEORETICAL BASIS

Within our system, a POL is characterized principally by a grammar (i. e., set of syntactic productions), and the consequent processing of programs in the POL is syntax-driven. To assure adequacy with respect to completeness ambiguity [6], and finiteness of analysis, our syntactic method is rigorously based. A grammatical model (the analytic grammar) was developed [8], which provides a rigorous description of syntactic analysis via formalization of the notion of a scan. Within this model, the selection process of a scanning procedure can be precisely stated, and thus made amenable to theoretical investigation. Some characteristics of this model are:

- all analytic languages are recursive
- all recursive sets are analytic languages

1. Each BASE operation, declarator, etc., consists of a three-letter operation code followed by $n \geq 1$ operand type specifier/operand pairs S_i/X_i ; e. g., FFF ($S_1/X_1, \dots, S_n/X_n$).

- all phrase structure grammars are analytic grammars
- there is a simple sufficient condition under which an analytic grammar provides unique analyses for all strings.

The grammar in a POL specification permits certain abbreviations and orderings of productions (for convenience, brevity, and efficiency), but is nevertheless equivalent to a grammar using the simple scan \mathcal{S}_4 of [8]. (An equivalent grammar using \mathcal{S}_4 is obtainable via a simple construction.) Context-sensitive productions may be used. Our method guarantees uniqueness of analysis - it is impossible to embed syntactic ambiguity in a language specification. A simple test ensures finite analyses for all strings. Such a grammar is at least as inclusive as the context-sensitive phrase structure grammar, and there does not appear to be any grammatical structure which cannot be accommodated (grammars of ALGOL, JOVIAL, and FORTRAN were obtained without difficulty).

In fact, such grammars are sufficiently powerful to accommodate the notions of "definition" and "counting" (cf. [7] and the examples of [8]), but to actually do so is neither efficient nor expedient. Therefore, a POL characterization includes description of pertinent "internal operations" (see the example in this paper).

3. SYSTEM OVERVIEW

An overview of the generation system is shown in Figure 1. Using

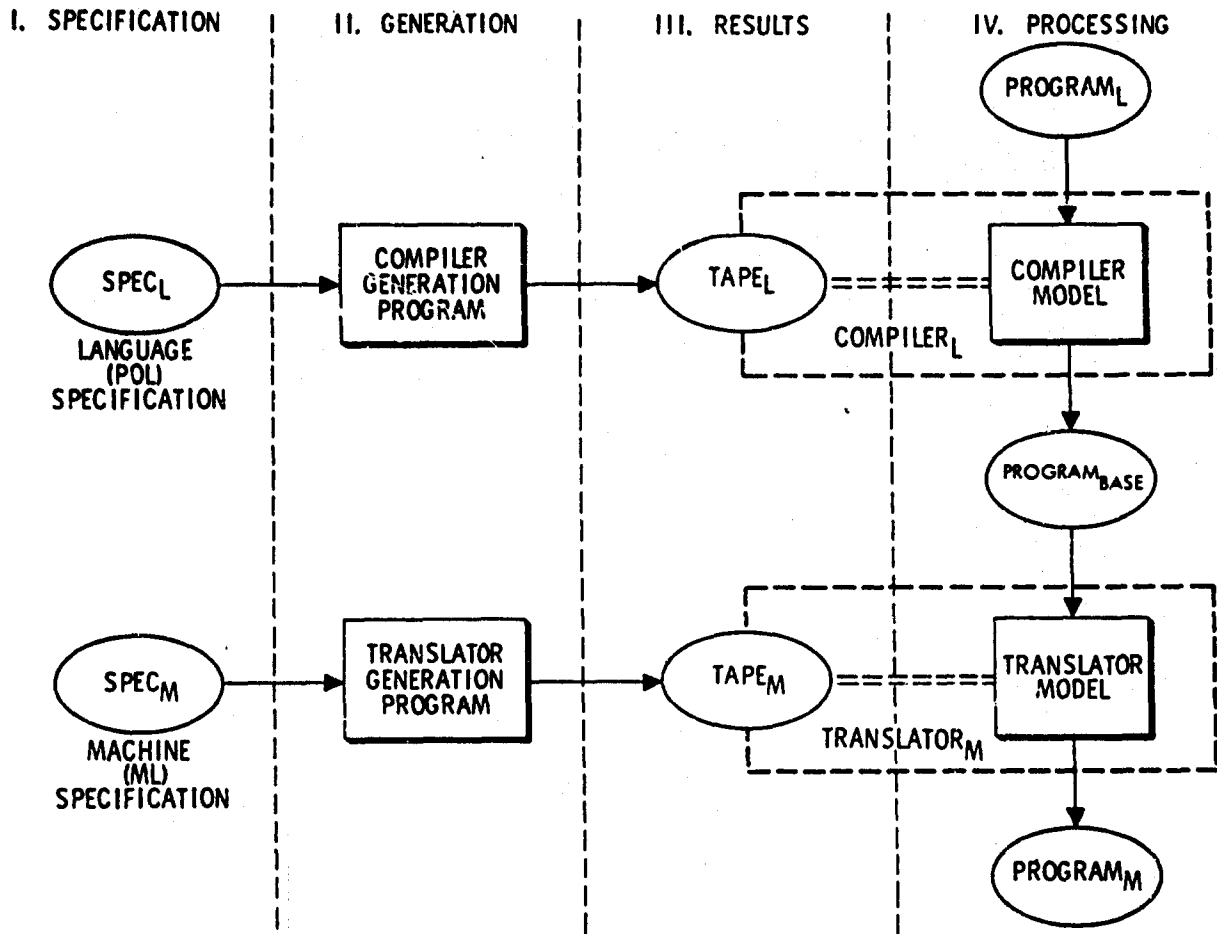


Figure 1. Overview of System

this system, the transformation from a source language L to a machine language M is achieved as follows:

A specification of L - an abstract description of the syntactic structure, "internal processing rules," and "output code" for L - is written. This specification is processed by the compiler generation system to produce a tape of L - a set of data tables corresponding to the specification. The compiler for L is then formed by conjunction of the tape of L with a compiler model program, a table-directed processor which acts simply as a machine for interpreting the tape of L.

Similarly, a specification of M is written designating macro-expansions appropriate to M. This specification is processed by a translator generation system to produce a tape of M - data tables containing the specified macro-expansions. The translator for M then formed by conjunction of this tape with a translator model program, which expands BASE operations to sequences of instructions in M as directed by the tape of M.

4. COMPILATION SYSTEM DATA BASE

Processing of input strings (POL programs) by a generated compiler is intended to occur in two parts:

(a) preliminary conversion of "raw" input symbols to yield a "syntactic" or "construct" string, which represents the raw input for all further processing, and then

(b) step-by-step syntactic analysis, and (at each analysis step) performance of prescribed sets of internal operations, prescribed output of "code blocks," output of diagnostic messages, and (if desired) performance of additional auxiliary processes.

The internal operations in a POL specification assume a set of data entities (the "data base"), which are later manipulated as prescribed by a generated compiler. Each entry of the construct string (which represents the raw input during processing) contains a construct (or syntactic type or token) and an associated datum, which is originally derived from the raw input, but may be internally altered. The use of appropriate string handling routines allows effectively a construct string of unbounded length. Other data entities are:

- (a) a set of function registers F_i , for storage and manipulation of "temporary" numeric data
- (b) a set of symbol registers S_i , for manipulation of symbol strings.
- (c) a property table of integer properties $P_i(J)$, for storage and manipulation of numeric data (e. g., number of dimensions) associated with "variables" in the input string. "Names" (i. e., contents of symbol registers) can be "defined" to the table to reserve table entries for associated data, and the table can be "searched." Defined names are placed in a property table index. The J^{th} table entry consists of four properties $P_0(J)$, $P_1(J)$, $P_2(J)$, $P_3(J)$. By convention, $P_0(J)$ is the syntactic class of the corresponding defined name.

See Figure 2 for further details.

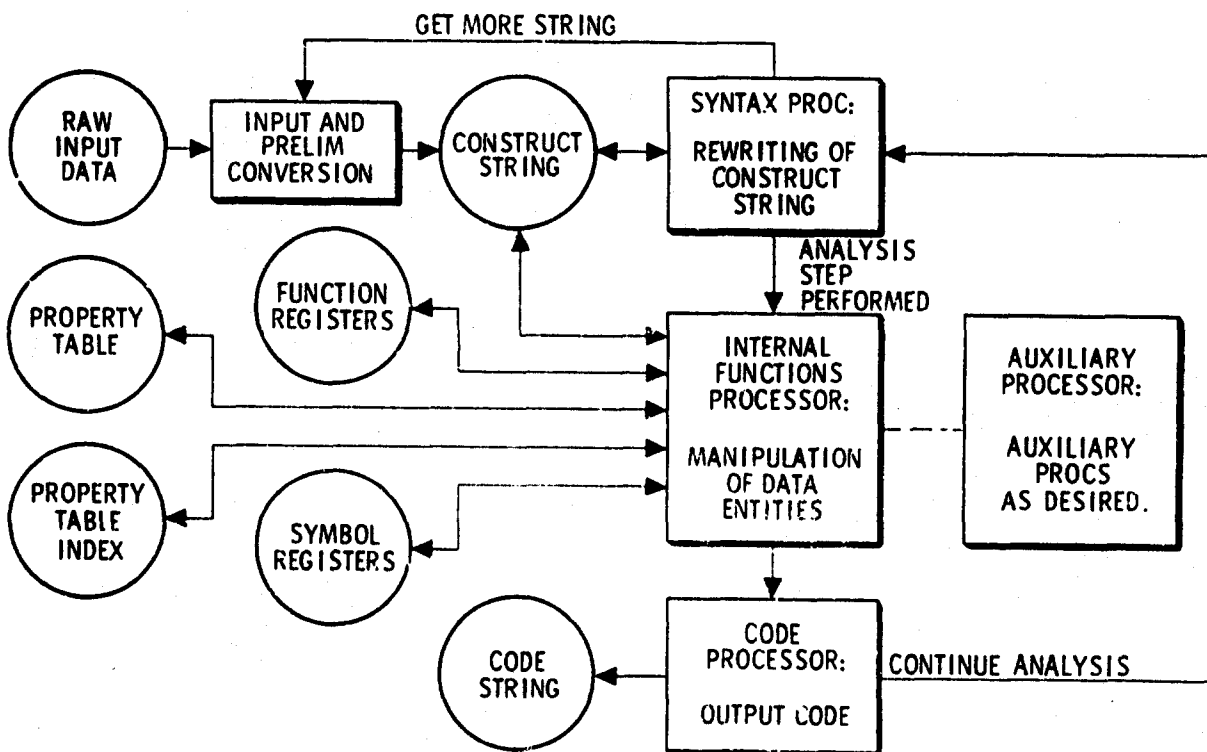


Figure 2. Compiler Model Organization and System Data Entities

5. POL SPECIFICATION AND COMPILATION SYSTEM OPERATION

The relation between a POL specification and the consequent compilation system processing is best shown via an example. Figure 3 shows a specification² of the language LEMMA2 (first exhibited in Lemma 2 of [2]).

```
* TITLE(LEMMA2)
* SYMBOLS
(1)A      (A)      (1)
(1)B      (B)      (2)
(1)C      (C)      (3)
(1)'      (END)    (0)
(1)       (NULL)   (0)
((EOC))   (NULL)   (0)
* END SYMBOLS
* SYNTAX
001      (A)(B)(K)  == (B)
002      (B)(A)(A)  == (B)(K)(A)
003      (B)(A)(B)  == (Q)
004      (B)(B)(K)  == (B)(X)(K)
005      (B)(Q)(B)  == (Q)
006      (END)(A)(Q)(C)(C)(C)(END) == (Z)
007      (X)(B)     == (K)(B)
008      (X)(K)     == (X)(B)
* END SYNTAX
* INTERNAL FUNCTIONS
001      RTV F3 -2
          INC F3 1
          ASD -3 F3
003      SET F5 1
          SET F6 2
          PUT S1 VO(-1)
          SUF S1 VO(0)
          DEF S1 ((A))
          ASD 0 F0
          SET P1(F0) 1
005      INC F5 1
          MPY F6 2
006      PRN 1 S1
* END INTERNAL FUNCTIONS
* CODE
003      BEG(V/R(0))
          PWR(C/F6)
005      PWR(C/F6)
006      AAA(C/R(-5))
          BBB(C/F5)
* END CODE
* DIAGNOSTICS
0001 ***** END OF SAMPLE ANALYSIS *****
* END DIAGNOSTICS
* END DATA
```

Figure 3. Specification of the LEMMA2 Language

2. The specification is shown in "reference" format, which differs trivially from the format used in machine processing of specifications.

which consists of sentences having the form

'A^mBⁿA^mBⁿCCC'

where X^k signifies a sequence of k X's. Some sentences of LEMMA2 are

'AABBBAABBCC'

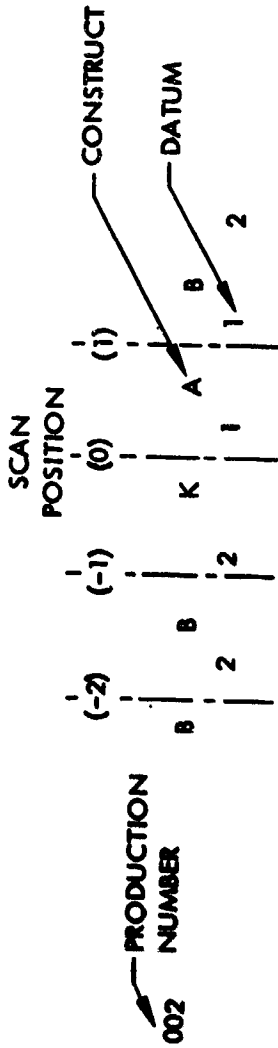
'A A A B B A A A B B C C C'

'A A A B B B B A A A B B B B C C C'

The specification contains five sections:

- (1) Symbols - specifies the preliminary conversion of input symbols and "reserved words" to construct string entries
- (2) Syntax - a set of syntactic productions for use in syntactic analysis
- (3) Internal Functions - the internal processing to be carried out at each analysis step
- (4) Code - the sequences of codes to be output at each analysis step
- (5) Diagnostic Messages - a set of messages for output

The sections containing internal functions, code and diagnostic messages are unnecessary in defining the language structure, but have been added to illustrate these mechanisms. The codes BEG, PWR, AAA and BBB appearing in the code section were invented expressly for this example; arbitrary BASE operation codes may be designated at will, since these codes are merely transmitted during compilation. The following discussion can be correlated with Figure 4, which shows the compilation analysis trace for a LEMMA2 program, together with resulting values of function registers and code output at each analysis step.



Production Number	Scan Position (-2)	Scan Position (-1)	Scan Position (0)	Scan Position (1)	Construct	Datum
002	A	B	K	A	B	2
004	A	B	X	K	A	2
008	A	B	X	B	A	2
007	A	B	X	B	B	2
004	END					
008	A	B	X	B	A	2
007	END					
001	NULL	END				
003	END					
005	NULL	END				
006	NULL	END				
007	NULL	END				
008	NULL	END				
009	NULL	END				
010	NULL	END				
011	NULL	END				
012	NULL	END				
013	NULL	END				
014	NULL	END				
015	NULL	END				
016	NULL	END				
017	NULL	END				
018	NULL	END				
019	NULL	END				
020	NULL	END				
***** END OF SAMPLE ANALYSIS *****						

- Each step of the analysis trace shows the string in the vicinity of the scan position, after application of the production, performance of internal functions, and code output.
- The code output for production p precedes the trace line for production p.
- Values of the first 20 function registers are shown at each analysis step: on line with construct F0 through F9, on line with datum F10 through F19.

Figure 4. Compilation of a LEPMA2 Program

The conversion specified in the Symbols section, of raw input symbols to construct string format, is performed specifically to eliminate dependency of processing on particular machine character sets and hollerith codes. A construct string entry containing a construct and an associated datum replaces each input symbol (or symbol sequence constituting a reserved word); Figure 5 illustrates this process. An arbitrary numeric or hollerith datum may be specified. Data from the construct string may be used to construct symbol strings (names), but this usage is not dependent on the specific hollerith codes which are used.

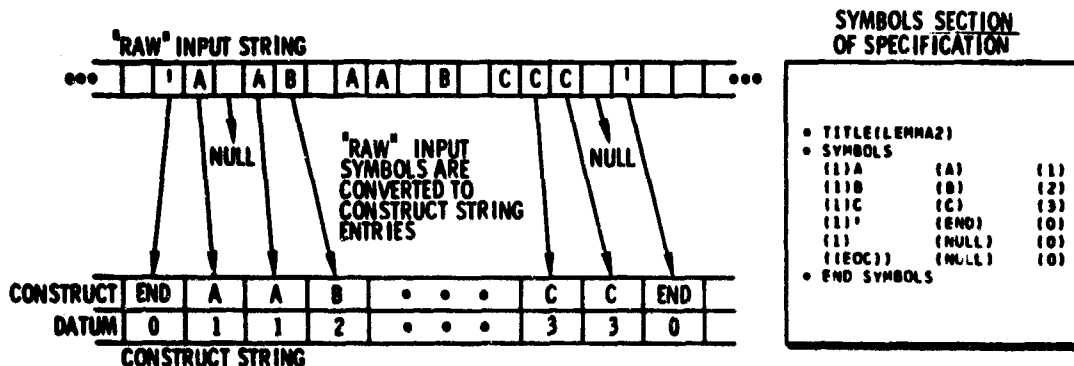


Figure 5. Preliminary Symbol Conversion

- The number in parentheses on the left indicates the number of characters comprising the reserved word. The symbols of the reserved word follow.
- A construct (e. g. , (END)) is specified for each symbol or reserved word. Use of the construct (NULL) specifies that no construct string entry is to be made; thus "blanks" are ignored above.
- A datum is specified for each symbol or reserved word. Either a numeric datum (e. g. , (3)) or a hollerith datum (e. g. , ((h), where h is the desired hollerith datum) may be specified.
- The special notation ((EOC)) denotes the "end of card symbol", which in many languages is regarded as a punctuation mark. A representation of ((EOC)) must be given in every Symbols section.

The syntactic productions in a specification's Syntax section are applied (as determined by the compiler model's scan) to "rewrite" the construct string, in a step-by-step fashion (see Figure 6). The succession of these rewritings constitutes the syntactic analysis of the construct string. In selective productions from the set of Figure 6, the compiler model uses the "leftmost" scan \mathcal{S}_1 of [8], i. e., at each step the production chosen is the one whose "left side" occurs first (leftmost) in the construct string. Thus at the first analysis step, the substring chosen is BAA; at the second, ABK; and so on. To allow explicit reference to the data which accompany the

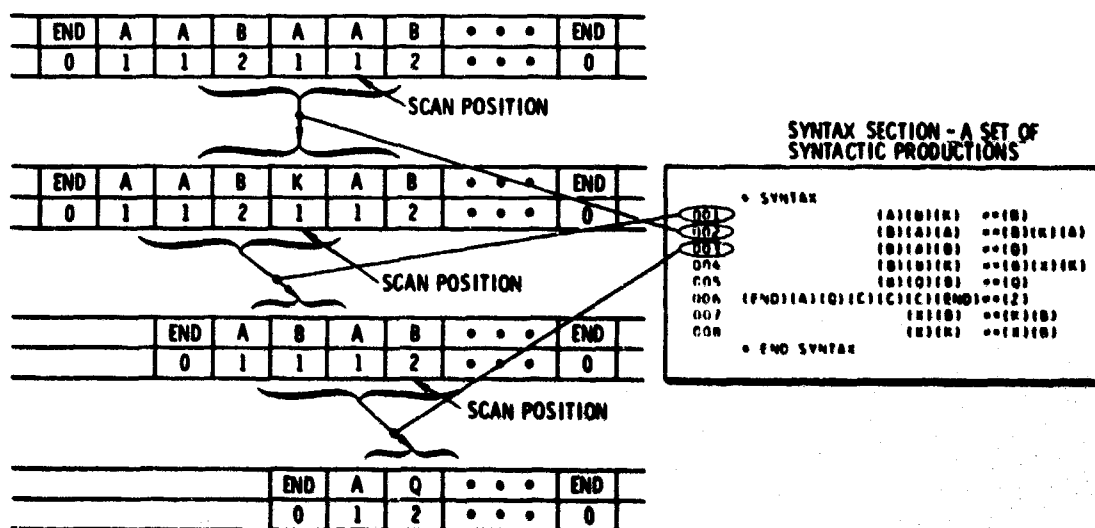


Figure 6. Syntactic Analysis

constructs of the substring chosen, a scan position is defined (at each step) to occur at the last (rightmost) construct of the selected substring (see Figure 6).

At each analysis step, internal operations associated with the selected production are performed: function registers or properties within the

property table may be set, used, or arithmetically manipulated; character strings may be placed in, prefixed to, or suffixed to symbol registers, and so on. The Internal Functions section (see Figure 7) consists of sequences of internal functions operations. The first operation of each sequence has the label of the production for which action is taken. Thus the sequence RTV F3 -2, etc., is performed each time production 001 is selected.

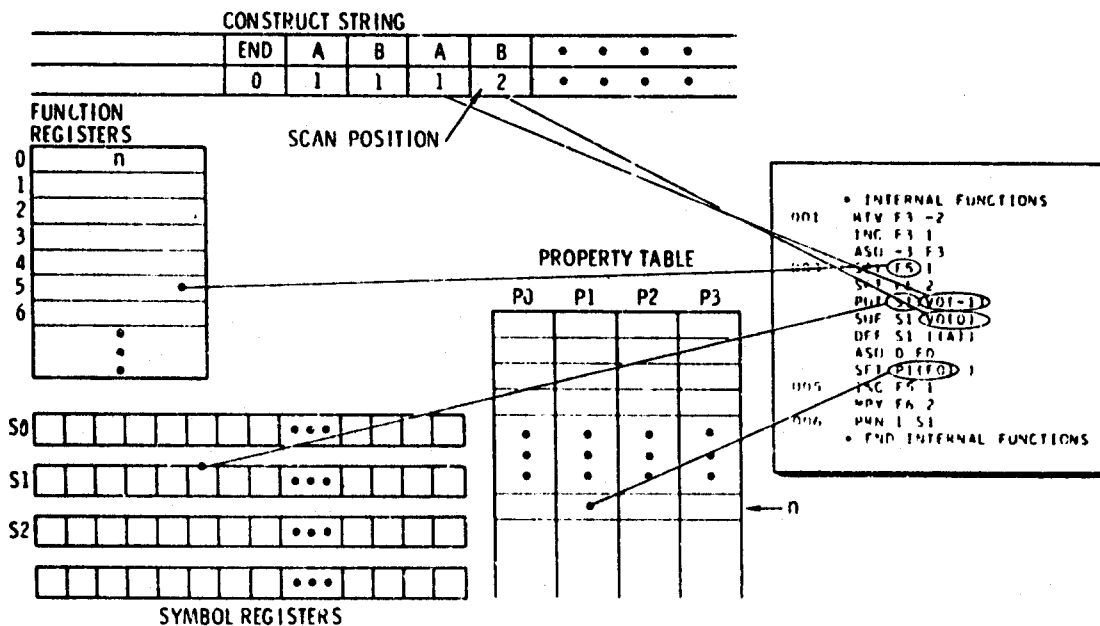


Figure 7. Performance of Internal Functions

- SET F5 1 places the value 1 in the function register F5
- PUT S1 V0(-1) places the datum (regarded as hollerith) from construct string position (-1) - relative to the scan position - into the symbol register S1. All previous contents of S1 are deleted.
- SUF S1 V0(0) suffixes to the string in S1 the datum from construct string position 0.
- DEF S1 ((A)) "defines" the string in S1 to the property table; a property table entry (say the n^{th}) is reserved, the string in S1 is entered into the property table index, together with the entry number n. The number representing the construct (A) is placed in P0(n), and n is placed in F0.
- ASO 0 F0 "associates" the value in F0 with the construct in string position 0: the value F0 is placed in the datum of position 0.
- SET P1 (F0) 1 places the value 1 in P1(F0), i. e., in P1(n).

Care has been taken in formulating the internal operations to achieve economy of means - simple operations, a minimum of system data entities, and a minimum of compiler model machinery. Such a formulation allows a simple compiler model program, while language complexities must be expressed within the language specification. Some anomalies of notation still remain from our earlier efforts, but it is planned to revise and clarify notation.

Operation sequences pertaining to different productions are independent of each other, since there is no "GOTO" operation (a "skip forward" is sometimes permitted). Thus a finite sequence of operations is performed at any analysis step.

Code may be output at any analysis step. Operation codes and operand type specifiers given in the Code section (see Figure 8) are merely

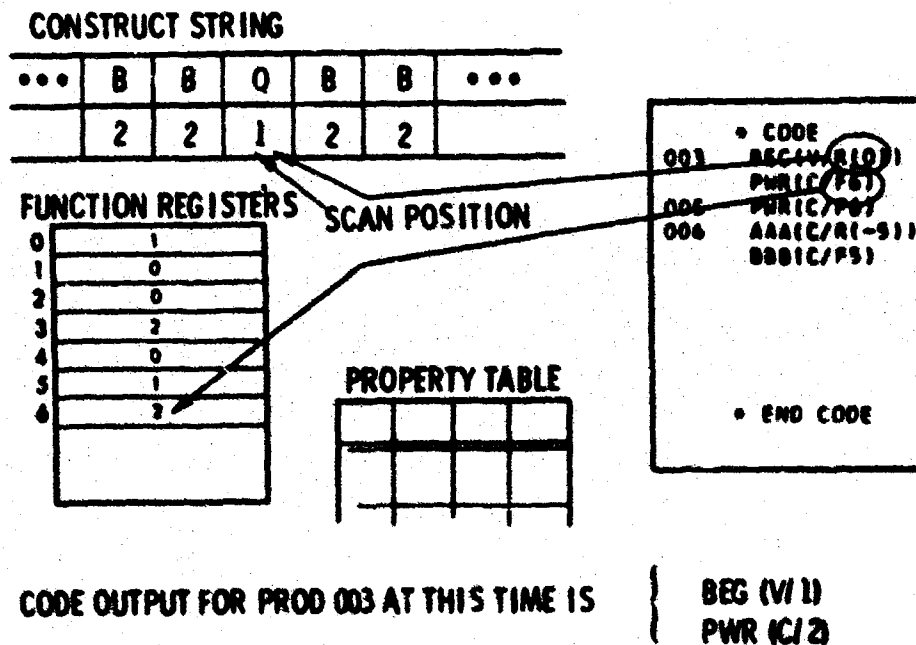


Figure 8. Output of a Code Sequence

transferred to the output, while operands are inserted as specified.

The Diagnostic Message section contains a set of messages, which are output by PRN internal operations. The operation PRN1 S1, which is executed for production 006, prints message 001 and the contents of S1.

6. TRANSLATION AND MACHINE SPECIFICATION

A translator for a given target machine (ML) produces, from an input program of BASE operations, an equivalent program in the target assembly language, in a format acceptable to the target assembler. The production of assembly language guarantees compatibility of the object program with the machine's monitor system, and allows the assumption in translation of system subroutines and macros.

A BASE program contains generalized item declarators, array declarators, etc., and generalized computation operators (e. g. , ADD, SUB). Since data definition is explicit, the BASE computation operators do not take account of the data types involved in the operations. Thus for each computation operation, there is an equivalent set of standard suboperations: e. g. , corresponding to ADD are the standard suboperations

"add a floating item to a fixed item"

"add a fixed item to a floating item"

and so on. Determination of the specific suboperation required for a given BASE operation, taking into account the data types involved, is performed within the translator.

Translation thus occurs in two parts:

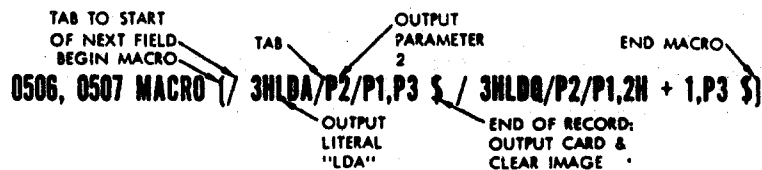
- (a) analysis of BASE operations by an analysis section, to derive equivalent sequences of standard suboperations, followed by
- (b) expansion of the standard suboperations by a macro-processor section, to produce assembly code.

A machine specification defines expansions of the standard suboperations. In other words, it defines for each standard suboperation an equivalent sequence of assembly language instructions. Embedded in these expansions are format specifiers, which cause the appropriate format to be generated. A machine specification is processed by the translator generation system to produce corresponding data tables, which are combined with the translator model program to form the desired translator. These data tables direct the expansions performed by the translator's macro-processor.

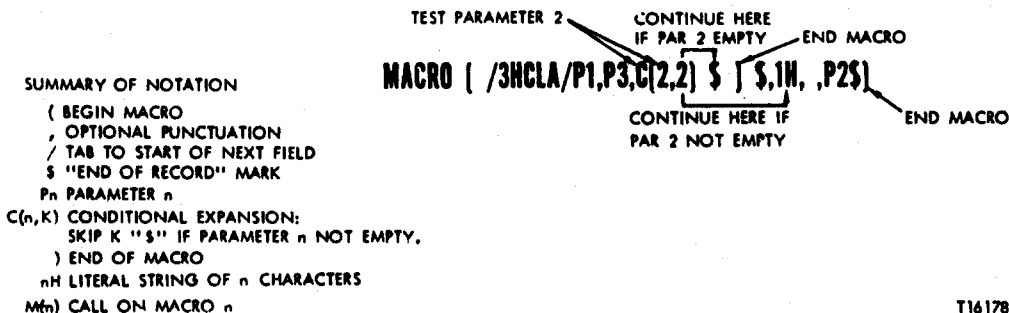
Parameters required by the expansions are furnished by the translator's analysis section via a communication table, from which they are retrieved as necessary by the macro-processor section. Within a machine specification, parameters are specified via position in this table.

Our present machine specification notation is processor-oriented, and not easily readable; however, it is planned to formalize this notation. Some typical macro definitions are shown in Figure 9, in a contemplated notation, as an illustration of the features provided in a machine specification.

LOADING ACC AND MQ WITH DOUBLE PRECISION OR COMPLEX NUMBER (FOR CDC 1604):



LOAD ACCUMULATOR (FOR IBM 7094 FAP):



T16178

Figure 9. Some Typical Macro Definitions

The translator model program, except possibly for one output procedure, is machine-independent. The analysis of BASE operations is dependent only on the operator, accumulator data type, and operand data type involved, while macro expansion is table-driven. All dependency on the target machine is isolated within the data tables used to direct expansions. Assembly code is output in the form of 80 column card images, which are almost universally acceptable by target assemblers. Unusual cases might require simple modification of the output procedure.

7. CONCLUSIONS

Using the syntactic model of [8], we have developed a system to formally characterize languages which are rich in grammatical structure.

and to subsequently process strings in such languages. Such processing can produce linear code (BASE language). The BASE language contains computation and data declaration operations sufficient to accommodate the functions of ALGOL, FORTRAN, and JOVIAL. BASE is expandable, so that more convenient or efficient operations may be introduced when these are desirable. We have shown the feasibility of formally characterizing machine (assembly) language, and of machine-independent translation (BASE \rightarrow ML). In sum, we have presented a rigorously based, machine-independent compiler generation system.

A consequence of these results is that language invariance can be maintained from machine to machine. It is possible to have a standard version of each procedure-oriented language, rather than machine-dependent variants.

The system is presently running on the CDC 1604 computer. Specifications of ALGOL, FORTRAN, and JOVIAL have been written, as has machine specification for the CDC 1604. The ALGOL and FORTRAN specifications have undergone tentative checkout and modification, as has the CDC 1604 specification. Preliminary comparisons of operating characteristics have been made. For a small number of short programs, our system produces object programs about the same size as do the manufacturer-supplied compilers, and requires between twice and three times the computer time. Since our system is a prototype, these results indicate that it may be possible to generate compiler/translator systems which have competitive efficiencies. We contemplate major operational

changes, without the sacrifice of theoretical rigour, which should increase system speed by a factor of between 3 and 5.

The compiler (POL \rightarrow BASE) portion of this system has other uses. The ability to formally characterize grammatically rich languages and to subsequently process strings in such languages is of importance wherever string-structure-dependent processing is required.

REFERENCES

1. Chomsky, N., Syntactic Structures, Mouton and Company, The Hague, 1957.
2. Chomsky, N., "On Certain Formal Properties of Grammars," Inform. Contr. 2 (1959), 137-167.
3. Ginsburg, S. and Rice, H.G., "Two Families of Languages Related to ALGOL," J.ACM 9 (July 1962), 350-371.
4. Naur, P. (Ed.), "Report on the Algorithmic Language ALGOL 60," Comm. ACM 4 (May 1960), 299-314.
5. Irons, E.T., "A Syntax-Directed Compiler for ALGOL 60," Comm. ACM 5 (Jan. 1961), 51-55.
6. Cantor, D.G., "On the Ambiguity Problem of Backus Systems," J. ACM 9 (Oct. 1962), 477-479.
7. DiForino, A.C., "Some Remarks on the Syntax of Symbolic Programming Languages," Comm. ACM 6 (Aug. 1963), 456-460.
8. Gilbert, P., "On the Syntax of Algorithmic Languages," J. ACM 13 (Jan 1966), 90-107.
9. Gilbert, Hosler, and Schager, "Automatic Programming Techniques," RADC-TDR-62-632, AD 400 325L, American Systems, AF 30(602)-2400.
10. Gilbert, Gunn, and Schager, "Automatic Programming Techniques," RADC-TR-66-54, AD 488 851L, Teledyne Systems, AF 30(602)-3330.
11. Gilbert, Gunn, Schager, and Testerman, "Automatic Programming Techniques," RADC-TR-66-665, AD 811 144L (Vol. I), AD 811 145L (Vol. II), Teledyne Systems, AF 30(602)-3330.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D		
<i>(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)</i>		
1. ORIGINATING ACTIVITY (Corporate author) Rome Air Development Center (EMIRD) Griffiss Air Force Base, New York 13440		2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
		2b. GROUP N/A
3. REPORT TITLE COMPILER GENERATION USING FORMAL SPECIFICATION OF PROCEDURE-ORIENTED AND MACHINE LANGUAGES		
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) IN-HOUSE		
5. AUTHOR(S) (First name, middle initial, last name) Philip Gilbert William G. McLellan		
6. REPORT DATE August 1967	7a. TOTAL NO. OF PAGES 20	7b. NO. OF REFS 11
8a. CONTRACT OR GRANT NO. N/A	9a. ORIGINATOR'S REPORT NUMBER(S) RADC-TR-67-454	
b. PROJECT NO. 4594	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) None	
c.		
d.		
10. DISTRIBUTION STATEMENT This document has been approved for public release and sale; its distribution is unlimited.		
11. SUPPLEMENTARY NOTES None	12. SPONSORING MILITARY ACTIVITY Rome Air Development Center (EMIRD) Griffiss Air Force Base, New York 13440	
13. ABSTRACT <p>A compiler generation system is described which is rigorously based and which allows formal specification both of the source (procedure oriented) languages and of the object (machine oriented) languages. An intermediate or "buffer" language, BASE, is interposed, reducing the required transformation techniques described. The system, so far, includes those elements in BASE necessary to produce ALGOL, FORTRAN, and JOVIAL compilers.</p> <p>This paper was presented at the 1967 Spring Joint Computer Conference.</p>		

DD FORM 1 NOV 65 1473

UNCLASSIFIED

Security Classification

UNCLASSIFIED

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Programming Languages Compilers FORTRAN JOVIAL ALGOL						

UNCLASSIFIED

Security Classification