General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

REPORT SAMSO-TR-76-137

(2)

FC.

AD A029312

Evaluation of HAL/S Language Compilability Using SAMSO's Compiler Writing System (CWS)

Systems Software Department
Information Processing Division
Engineering Science Operations
The Aerospace Corporation
El Segundo, Calif. 90245

20 August 1976

Final Report

Prepared for
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
LANGLEY RESEARCH CENTER
Langley Station, Hampton, Virginia 23365

and

SPACE AND MISSILE SYSTEMS ORGANIZATION
AIR FORCE SYSTEMS COMMAND
Los Angeles Air Force Station
P.O. Box 92960, Worldway Postal Center
Los Angeles, Calif. 90009

This final report was submitted by The Aerospace Corporation, El Segundo, California 90245, under Contract F04701-75-C-0076 with the Space and Missile Systems Organization, Deputy for Advanced Space Programs, P.O. Box 92960, Worldway Postal Center, Los Angeles, California 90009. It was reviewed and approved for The Aerospace Corporation by S. C. McCarty and K. F. Steffan, Engineering Science Operations and G. W. Anderson, Advanced Programs Division. The Air Force project engineer was Captain Craig E. Miller, SAMSO/YAD.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication. Publication of this report does not constitute Air Force approval of the report's findings or conclusions. It is published only for the exchange and stimulation of ideas.

Craig E. Miller

Craig E. Miller, Capt., USAF
Project Officer
Computer Technology Function
Development Directorate

Frank P. Dyke

Frank P. Dyke, Lt. Col., USAF
Chief, Computer Technology Function
Development Directorate

FOR THE COMMANDER

Roger W. Johnson

Roger W. Johnson, Col., USAF
Deputy for Advanced Space Programs

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: SAMSO-TR-76-137	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Evaluation of HAL/S Language Compilability Using SAMSO's Compiler Writing System (CWS).	5. TYPE OF REPORT & PERIOD COVERED: Final Report. Mar [redacted] 75 – Feb [redacted] 76
6. PERFORMING ORG. REPORT NUMBER: TR-0076(6803)-1	
7. AUTHOR(s): M. Feliciano, H. D. Anderson, J. W. Bond, III	8. CONTRACT OR GRANT NUMBER(s): F04701-75-C-0076
9. PERFORMING ORGANIZATION NAME AND ADDRESS: The Aerospace Corporation, El Segundo, California 90245	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: National Aeronautics and Space Administration, Langley Research Center, Langley Station, Hampton, Va 23365	12. REPORT DATE: 20 Aug [redacted] 1976 13. NUMBER OF PAGES: 70
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Space and Missile Systems Organization, Air Force Systems Command, Los Angeles Air Force Station, P.O. Box 92960, Worldway Postal Center, Los Angeles, Ca 90009	15. SECURITY CLASS. (of this report): Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited.

12) 74p.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer language
Language constructs
Compiler
Compiler writing system
Language comparisons
Programming languages

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

NASA/Langley is engaged in a program to develop an adaptable guidance and control software concept for spacecraft such as shuttle-launched payloads. It is envisioned that this flight software be written in a higher-order language, such as HAL/S, to facilitate changes or additions. To make this adaptable software transferable to various onboard computers, a compiler writing system capability is necessary. A joint program with the Air Force Space and Missile Systems Organization was initiated to determine if the

OVER

DD FORM (FACSIMILE) 1473

UNCLASSIFIED 409 525

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AB

19 KEY WORDS (Continued)

20 ABSTRACT (Continued)

Compiler Writing System (CWS) owned by the Air Force could be utilized for this purpose. The present study explores the feasibility of including the HAL/S language constructs in CWS and the effort required to implement these constructs. This will determine the compilability of HAL/S using CWS and permit NASA/Langley to identify the HAL/S constructs desired for their applications. The study consisted of comparing the implementation of the Space Programming Language using CWS with the requirements for the implementation of HAL/S. It is the conclusion of the study that CWS already contains many of the language features of HAL/S and that it can be expanded for compiling part or all of HAL/S.

It is assumed that persons reading and evaluating this report have a basic familiarity with (1) the principles of compiler construction and operation, and (2) the logical structure and applications characteristics of HAL/S and SPL.

PREFACE

This report was prepared for the Advanced Programs Division, Guidance Program Office of The Aerospace Corporation and for the Stability and Controls Branch of the Flight Dynamics and Controls Division, National Aeronautics and Space Administration, Langley Research Center.

The report authors are M. Feliciano, H. D. Anderson, and J. W. Bond, III, of the Information Processing Division. Substantial contributions and helpful suggestions to this report by Allan Gott and Leila Jennings of the Information Processing Division and Linda Lisak of the Advanced Programs Division are gratefully acknowledged.

CONTENTS

CONTENTS (Continued)

FIGURES

TABLES

1. INTRODUCTION

1.1 BACKGROUND

NASA/Langley is engaged in a program to develop an adaptable guidance and control software concept for spacecraft such as shuttle-launched payloads. Such a software package would contain generalized control algorithms, hardware interfaces, and digital filters which could be adapted to different spacecraft configurations or missions by parameter changes rather than by recoding. It is envisioned that the flight software be written in a higher order language, such as HAL/S, to facilitate changes and/or additions. To make this adaptable software transferable to various onboard computers, a compiler writing system capability is necessary. A joint program with the Air Force Space and Missile Systems Organization (SAMSO) was thus initiated to determine if the Space Programming Language (SPL) Compiler Writing System (CWS) could be utilized for this purpose. This approach would take advantage of the compiler writing capability already owned by the government.

1.2 PURPOSE OF THE STUDY

The present study is intended to establish the feasibility of representing the HAL/S language constructs in the CWS intermediate language and to determine the effort required to implement these constructs in CDC 6600 object code. This will define the compilability of HAL/S using the SAMSO CWS and permit NASA/Langley to identify the HAL/S constructs or subset desired for the onboard guidance and control software applications.

1.3 SUMMARY OF RESULTS AND RECOMMENDATIONS

The resources available to the above mentioned compiler-writing system are sufficient to produce a compiler for HAL/S with a reasonable amount of effort. This is the result of having used CWS for writing compilers for SPL, a higher order aerospace oriented programming language sharing common goals with HAL/S. The following features of HAL/S require attention:

structures, which do not exist in SPL; real-time features that are dependent on a real-time executive or operating system, which are more elaborate than those found in SPL; error-recovery features that depend on an error-recovery executive, which are more elaborate than in SPL; and arrays of matrices and vectors.

It is estimated that implementation of the above mentioned features will take up most of the effort involved in incorporating HAL/S features into the CWS intermediate language. A complete account of each HAL/S language feature is given in the table in the Appendix. Section 4.2 of this report contains detailed comments on the CWS compatibilities or compiler problems associated with each feature described in the Appendix.

It is the conclusion of the study that the CWS intermediate language already contains many of the HAL/S language features and can be readily expanded to serve very well for compiling part or all of HAL/S.

2. GENERAL DISCUSSION

Section 2 is given as general technical background to the discussions in subsequent sections.

2.1 THE COMPILATION PROCESS

A compiler is a computer program which accepts as input a source program written in a higher-order language and produces as output an equivalent program called the object program in a form acceptable to a computer. The compiler resides in the host computer and produces object programs for the target computer. The relationship between host and target computers is illustrated in Figure 1a.

Although a variety of techniques can be used in a compiler implementation, a number of functions are common to most compilers. These include lexical analysis, syntactic analysis, semantic analysis, code generation, bookkeeping, code optimization, and error recovery and error analysis. These functions are discussed below. A compiler is not necessarily structured according to the various functions it performs. In practice the above mentioned functions are conceptual entities which are difficult to separate from each other in an actual compiler. Nevertheless, they are useful concepts for descriptive purposes.

A compiler may examine the source program or equivalent representations of it one or more times. Each such examination is called a pass. The result of a pass is a transformation of its input into an equivalent form. A compiler may perform several functions in a pass; in fact, there exist one-pass compilers although many, if not most, compilers contain at least two passes.

2.1.1 Lexical Analysis

A higher-order language uses a set of characters from which are formed the primitives of the language. In HAL/S examples of primitives are the comma separator, the reserved word DECLARE, and an identifier.

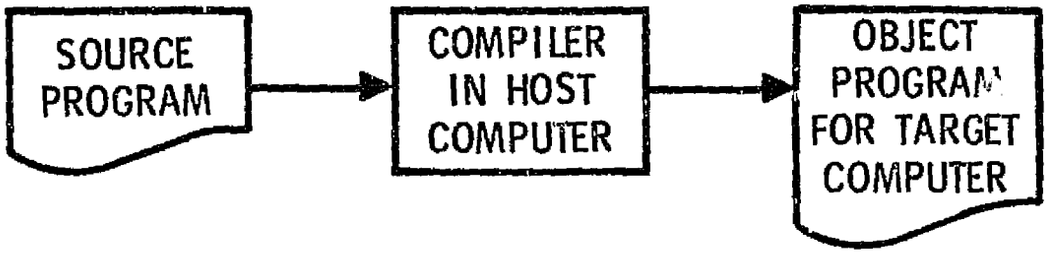

Figure 1a. Relationship Between Host and Target Computers

Primitives, then, are substrings of one or more characters that are considered to be an entity of the language very much like words and punctuation marks are considered entities in a natural language.

Lexical analysis is the process of identifying the various primitives used in a source program. Usually the lexical analyzer delivers a token which contains two components: the identification of the primitive, and the information about the primitive. Thus, for example, the token of a variable may consist of the designation of the primitive as an identifier and the name of the identifier. In actual practice more information is given. Lexical analysis is common to all compilers.

2.1.2 Syntactic Analysis

A sentence of a natural language is analogous to a declaration or a statement of a higher-order programming language. The purpose of syntax analysis is to parse the "sentences" of the source program and to form parse trees. It was observed that lexical analysis groups characters into tokens. Roughly speaking, syntactic analysis groups tokens into trees. The parse trees formed by a syntactic analyzer arise from the specified constructs of the language. The parse trees give a complete syntactic analysis of the statements and declarations of the program. Therefore, the result of syntactic analysis is a transformed input source program in which relationships among the primitives (tokens) are given in exacting detail. In many compilers the output of syntactic analysis is explicit and consists of a set of parse trees. Each declaration (nonexecutable statement) results in one or more dictionary entries, and each executable statement has its own tree. Thus, there are assignment trees, IF... THEN trees, and so on.

The trees are then processed to produce object code. It will be seen in the comments on specific features of HAL/S that reference is frequently made to the availability of various kinds of trees in CWS.

2.1.3 Semantic Analysis

The function of semantic analysis is to establish if valid relationships exist among various elements of the program. For example, does a GOTO statement have an admissible label? Do the number of parameters in a procedure CALL agree with the number of parameters in the procedure declaration? Semantic analysis is also required to resolve forward references. How much semantic analysis is performed depends on the objectives of a compiler as well as on technical grounds.

2.1.4 Code Generation

Code generation is the ultimate purpose of the compiler. Some compilers are designed to produce machine language code. Most, however, produce assembler language code. An advantage gained by producing assembler code is that some "tightening" or optimization of the object code can best be achieved at this level.

2.1.5 Bookkeeping

The identifier token produced during lexical analysis must deliver information about the identifier such as its name and its type. All information about identifiers is listed in a symbol table or dictionary. In this way, the identifier token need only point to the place in the symbol table which contains the information about it. The management of the symbol table is termed bookkeeping. Bookkeeping also includes the management of any other tables used during compilation.

The symbol table is constructed to facilitate obtaining information from it and storing information into it. The symbol table is used during code generation to provide the information needed for allocation of storage to the various identifiers and constants used in the source program.

2.1.6 Code Optimization

Many compilers use algorithms to reduce the run time or the storage requirements of the object program. Although no compiler can guarantee that an object code is optimized in time or in space, or in both, the

term "code optimization" persists. The speed of the compiler is generally reduced as the amount of optimization increases.

2.1.7 Error Recovery and Error Analysis

This function refers to the ability of the compiler to continue processing a source program that contains errors and to the ability to determine the kinds of errors in the source program. This function is common to all compilers.

2.2 COMPILER WRITING SYSTEMS

The purpose of a compiler writing system is to facilitate writing compilers. Several resources are available in compiler writing systems. A compiler writing system may make available one or more languages specifically designed for writing compilers. The tasks of bookkeeping and of program and data management may be simplified. Some functions--lexical analysis and syntactic analysis--can be mechanized or they can be written quickly by using established patterns. Also, the system may allow the use of previously developed compilation algorithms.

The success of compiler writing systems is based on various functions of the compilation process being well understood and on the proven ability of compiler writers to develop efficient, general purpose algorithms that require only minor modifications for a specific compiler implementation.

It is not true that compiler writing systems have been developed which make writing compilers a trivial task. At present, compiler writing systems require a thorough knowledge of certain aspects of the compilation process. Also, present systems are large, complex programs which are not easily manageable. Despite this, compiler writing systems facilitate to a great extent the task of writing compilers.

2.3 CWS

The compiler writing system CWS is a significant upgrade of the SPLIT system originally developed by Systems Development Corporation for the Air Force Space and Missile Systems Organization.

CWS allows the use of two languages--Syntax Analysis Language (SAL) and Generator Language (GEN). These languages were designed to simplify the writing of compilers.

CWS has been used primarily to produce three-pass compilers. The first pass consists of lexical analysis and syntactic analysis. From it are obtained the symbol table and the parse trees. These serve as input to the second pass, which consists of semantic analysis, data allocation, and global optimization. Modifications to the symbol table and to the parse trees are the result of the second pass. The third pass uses this information to generate code and to do some local code optimization. A compiler configuration built using CWS is sketched in Figure 1b.

Usually, the first pass is called the front end of the compiler, and the other two passes are called the back end. The library of subroutines used by the three passes are termed the support package of the CWS. Note that the first pass depends on the language being compiled. The information used by CWS to produce the first pass constitutes the specifications of the language. Once the specifications of a language are given in an acceptable form, the front end of the compiler can be built without regard to the target computer; i.e., it is machine independent.

The second pass requires some knowledge of the target computer, but not to the extent required by the third pass. In fact, some of the tasks in the second pass are parametrized. To the extent that this has been done, the second pass can be reused with only minor modifications. The third pass, in contrast, requires a thorough knowledge of the target computer and must usually be rewritten when code is to be generated for a new target computer.

2.4 SPL

The higher order programming language SPL was developed by Systems Development Corporation for the Air Force Space and Missile Systems Organization. SPL is intended for the programming of space and airborne applications.

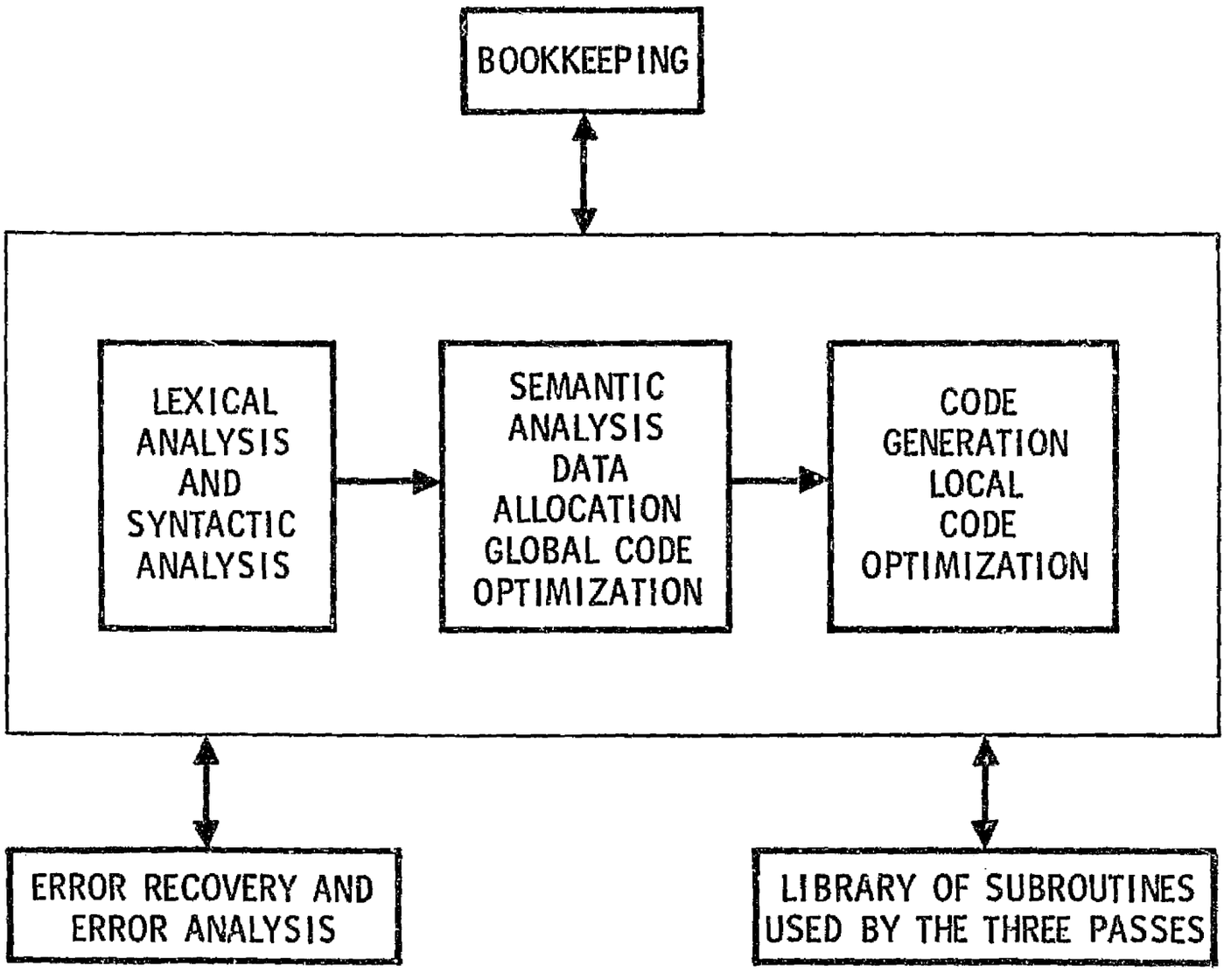

Figure 1b. A Compiler Configuration Built Using CWS

The CWS has been used to develop compilers for SPL for several target computers. To the extent that other languages are similar to SPL, the resources developed within CWS for the production of SPL compilers can be used in the production of compilers for other languages. Since most higher order programming languages share many common features, many of the facilities of CWS can assist the writing of compilers for languages other than SPL. Specifically, because they are designed for the same types of applications, SPL and HAL/S share many common features. Therefore, the CWS has many features which facilitate writing a compiler for HAL/S.

In brief, SPL does not handle the HAL/S structures. The real-time features and the error processing features of HAL/S are handled in SPL by the CHRONIC statement. However, in HAL/S these features are extended and elaborated beyond the corresponding features in SPL. The above are the major portions of concern. Another concern is the elaborate scheme of arrays in HAL/S; however, this turns out to be mostly a problem of syntax. In other areas differences are accounted for by SPL having more operators than does HAL/S and in turn by HAL/S having more data configurations than does SPL.

3. METHODOLOGY

This report establishes the degree of difficulty required to write a compiler for HAL/S using CWS. This was accomplished by listing all features of HAL/S and establishing for each feature the degree of effort which would be required to process it. In turn, this was achieved by establishing the degree of correspondence between each HAL/S feature and the available resources in CWS.

3.1 OBJECTS TO BE COMPARED

Each HAL/S feature was ranked as to degree of difficulty involved in its compilation. Examples of HAL/S features are assignment statements, scalar declarations, procedure calls, and structure declarations. The rank of difficulty was established by first establishing semantic correspondence with an SPL feature. Next, the syntactic organization of the two features was compared. Finally, the process used to compile the SPL feature was compared with the HAL/S feature to establish a correspondence in the process.

For example, the HAL/S assignment statement is similar to the SPL assignment statement. Their syntactic structures were compared and were found to be the same. The process involved in compiling the SPL assignment statement was matched with the requirements in compiling the HAL/S assignment statement. As an aside, it was found that the SPL assignment statement is more elaborate than the one in HAL/S and, therefore, a HAL/S compilation process would require that the existing SPL process be slightly restricted.

Briefly, a HAL/S feature was matched with an SPL feature. Requirements for compiling the HAL/S feature were compared with the corresponding resources available for compiling SPL. Discrepancies between HAL/S requirements and available SPL resources were noted.

3.2 CRITERIA FOR EVALUATION

Whenever a HAL/S feature matched with an SPL feature in semantic content, their syntactic structures were compared. If the syntactic structures matched, it was expected and verified that the available SPL compilation process could be used with only minor modifications to process the HAL/S feature.

If a HAL/S feature matched an SPL feature in semantic content but not in syntax, then two processes were investigated; syntactic analysis and the second pass, particularly the latter. If the second pass matched the requirements of the HAL/S feature, then the requirements for syntactic analysis were examined. An evaluation was made of the difficulty of doing the syntactic pass. It was expected and found that wherever a parse tree existed for the feature, only the syntactic analyzer would be affected.

If a HAL/S feature did not match any SPL feature in semantic content, then a ranking was established according to three categories: (1) an SPL feature exists which contains more than the HAL/S feature, (2) an SPL feature exists which is contained in the HAL/S feature, and (3) no SPL feature exists which approximates the HAL/S feature.

In category (1), the problem reduces to restricting the compilation process to the HAL/S semantic subset. It was examined and verified that in syntactic analysis the parse tree could be restricted through default options which would impose satisfactory restrictions on further processing.

In category (2), the choice was made to determine the necessary extensions which would be required in the second pass. Then a corresponding modification to the parse tree would be expected. Then, the difficulties in modifying the second pass and in producing the parse tree were determined.

In category (3), emphasis was placed on the second pass. The reason for this is that most of the difficulties in introducing a new parse tree would be found in the second pass. Then the syntactic analysis for the feature was examined to determine whether any additional problems could be found.

In general, the ranks and the compatibility used in the table in the Appendix were established using the above criteria.

3.3 EXPLANATION OF THE TABLES

The tables in the Appendix present the study findings. A partial table example is shown below:

11. Regular Expressions

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	Arithmetic expressions of scalars and integers	Arithmetic expressions of simple variables	1	1

At the upper left is found "11. Regular Expressions," which is a numbered category of features. The first column is labeled "Item ID" and contains the number 1. This number, when coupled with the category number (11), gives 11.1, which is the specific item number. The next column is "HAL/S Feature," which contains "Arithmetic expressions of scalar and integers." Thus, item 11.1 refers to the HAL/S feature "Arithmetic expressions of scalars and integers." The associated SPL feature is "Arithmetic expressions of simple variables." This establishes the correspondence between the HAL/S feature and the SPL feature. If the entry under SPL feature were blank, then there is no corresponding SPL feature.

Three other columns need explanation: "Priority," "Compatibility," and "Rank." The entries in these columns are integers and they are explained below.

"Priority" refers to the need of the HAL/S feature for various purposes. The entries here are integers whose values and meanings are listed below.

1. The HAL/S feature is a basic feature of most higher order programming languages. This implies that all programming languages of interest have the feature and that no reasonable subset of the language could do without the feature.

2. The HAL/S feature is useful or necessary for in-flight computer applications. However, features marked 2* are not strictly necessary. Most of the features marked 2* depend upon a real-time executive which supports the feature. In-flight computers have been programmed so that any real-time control is performed by the real-time executive and not by the applications programs. Therefore, the selection of features of priority 2 and 2* can only be made after the mode of operation of the in-flight computer has been designed.

3. A FORTRAN IV-like language can be built from features of priority 1, some features of priority 2, and features of priority 3. This then would form a HAL/S subset for use in ground computers.

4. A HAL/S feature of priority 4 is desirable for some applications and not for others. Therefore, the selection of features of priority 4 depends on the applications envisioned for the particular environment.

"Compatibility" refers to the degree of match which exists between an SPL feature and the HAL/S feature. The value shown establishes the degree to which the correspondence occurs. The values and the meanings of the entries are listed below.

1. The HAL/S feature and the listed SPL feature match in syntax and in semantics (with the possible exception of minor modifications in syntax).

2. The HAL/S feature is semantically like the SPL feature.

3. The syntax of the HAL/S feature and of the SPL feature match.

4. The HAL/S feature and the SPL feature are similar but with modifications.

5. There is no corresponding SPL feature.

"Rank" refers to the estimated degree of difficulty required to modify the existing intermediate language forms required for implementation of the HAL/S feature. Listed below are the values and their meanings.

1. A straightforward process which for the most part uses resources already available to CWS.

2. This feature can be processed using minor modifications of existing elements in CWS, or it is a new feature which can be implemented with little effort.

3. This feature requires substantial modification to the available resources, or it is a new feature which can be implemented with a moderate amount of effort.

4. This feature requires a major effort to implement under the existing resources, or it is an advanced new feature.

4. RESULTS

In the following paragraphs are given the general findings of the study, detailed comments on each feature of HAL/S, and a comment on defining subsets of HAL/S.

4.1 SUMMARY OF RESULTS

The study shows that HAL/S and SPL have many features in common. The study also shows that most features of HAL/S can be compiled with reasonable effort using the resources available to CWS. Salient exceptions to the above are listed below.

STRUCTURES do not exist in SPL and they require a method of data referencing not currently implemented.

TASK BLOCKS AND UPDATE BLOCKS require some modification to the existing intermediate language for their implementation.

ARRAYS in HAL/S are more elaborate than they are in SPL. Again, modifications are needed here.

REPLACE WITH ARGUMENTS may lead to serious difficulties because of the nesting of replace statements. This feature is implementation-dependent and nesting can be disallowed.

PARTIAL INITIALIZATION may result in a slower compilation.

REAL-TIME FEATURES are more elaborate than exist in SPL. They depend on a real-time executive. Their implementation depends upon the support of the real-time executive.

ERROR RECOVERY AND CONTROL is much more elaborate in HAL/S than it is in SPL. Implementation depends on the support of an error recovery executive.

% MACROS involve the use of the operating system because of necessary linkages.

4.2 COMMENTS ON SPECIFIC FEATURES

Below are given comments on the HAL/S features. The features are listed along with a numbering scheme for cross-reference to the tables in the Appendix of this report. Thus, T11.1 refers to Table 11, line 1, "Arithmetic expressions of scalars and integers."

4.2.1 Primitives (T1.)

The set of primitives of HAL/S and the rules for forming them are similar to those used by many other languages. A straightforward lexical analyzer should be sufficient to convert the primitives of HAL/S to tokens. A prepass may be required if two-dimensional formats are admissible in an implementation of the language (T1.4). It should be observed that the REPLACE statement (T3.1 and T3.2) may affect the lexical analyzer because REPLACE allows for substitution of a string of text by another string of text.

4.2.1.1 Character Set (T1.1)

The HAL/S character set contains 86 characters and seven additional extended-set symbols. An implementation of HAL/S may be affected by the peculiarities of the character set for any given computer system. For example, the alphabet may be restricted to uppercase letters.

4.2.1.2 Reserved Words (T1.2)

Reserved words tend to simplify the lexical analyzer.

4.2.1.3 Identifiers (T1.3)

The rule for constructing an identifier is used in many languages. The maximum number of characters in an identifier is 32.

4.2.1.4 Literals (T1.4)

The type of an arithmetic literal (integer or scalar) is contextually defined, and no distinction is made between single and double precision. Because of this, arithmetic literals must be processed exactly as would be arithmetic variables. This is particularly true in computations occurring

during compilation; for example, the computation of the size of each dimension in an array declaration. Also noted about arithmetic literals is the admissibility of multiple exponents. The maximum number of exponents allowed in an arithmetic literal is implementation-defined and could be restricted to one.

Bit literals and character literals can be processed straightforwardly. Notice that "/*" is the opening delimiter of a comment even in a character literal (T1.6).

4.2.1.5 Two-Dimensional Source Formats (T1.5)

The specifications of HAL/S regard the two-dimensional form as standard. However, an implementation of HAL/S is not required to follow the norm. Two-dimensional formats may require a prepass; that is, a pass through the source program before lexical analysis.

4.2.1.6 Comments and Blanks (T1.6)

Rules governing blanks are precisely defined. Comments may occur wherever blanks are legal (T1.4).

4.2.2 Block Structure and Organization (T2.)

The HAL/S concept of a program complex allows for the execution of several programs within the framework of an executive operating system. Of interest here is that a program may be activated by another program within a program complex. Hence, a program module may be thought of as a procedure that is activated by the executive operating system and upon termination returns control to the operating system.

To some extent a HAL/S program is block structured. In fact, the block structure of HAL/S is the same as that of SPL. However, neither has the block structuring capabilities of a block-structured language such as ALGOL 60.

4.2.2.1 Unit of Compilation (T2.1)

The unit of compilation is defined as one of four blocks (PROGRAM, FUNCTION, PROCEDURE, or COMPOOL block) optionally preceded by one or more templates. This is similar to an SPL compilation. A HAL/S program may reference more than one COMPOOL block. The maximum number of COMPOOL blocks allowed in a program complex is implementation defined.

4.2.2.2 Templates (T2.2)

Block templates provide information to the outermost code block about external blocks. This kind of information is needed for implementing independent compilations.

4.2.2.3 Program Block (T2.3)

A HAL/S PROGRAM block is similar to the SPL program. The program template of HAL/S and the START declaration of SPL have similar objectives. A HAL/S PROGRAM block does not admit arguments, whereas an SPL program does.

4.2.2.4 PROCEDURE and FUNCTION Blocks (T2.4)

The HAL/S PROCEDURE and FUNCTION blocks are similar in form and purpose to SPL procedures and functions.

4.2.2.5 TASK Blocks (T2.5)

SPL does not have a construct exactly like the HAL/S TASK block. A TASK block is a named block, identified as a TASK block and having no parameters. It can only be activated by scheduling it as a process under control of a real-time executive. Therefore, a TASK block is similar to an SPL procedure without parameters which can only be invoked by a CHRONIC statement. This implies the need of a TASK tree or of an extension of attributes in the procedure tree. The TASK block depends on the real-time executive.

4.2.2.6 UPDATE Blocks (T2.6)

The UPDATE block does not occur in SPL. The UPDATE block is an optionally named block identified as an UPDATE block and having no parameters. The UPDATE block is executed in line. The SPL inline procedure or the "statement" can be restricted to have the same effect as the HAL/S UPDATE block. The form of the UPDATE block is very restrictive. It does not contain input or output statements or other UPDATE blocks, and it cannot invoke FUNCTION or PROCEDURE blocks defined outside of the UPDATE block. Only certain real-time programming statements are admissible to the UPDATE block. The UPDATE block is intended for use in data sharing in a real-time environment.

Considerable effort may be required to ascertain that restrictions on the UPDATE block are observed. It will be necessary to use an update tree or to extend the attributes of a procedure.

4.2.2.7 COMPOOL Blocks (T2.7)

The HAL/S COMPOOL block is similar to the SPL COMPOOL block; however, HAL/S allows the use of many COMPOOLs, whereas SPL allows for only one.

4.2.2.8 Statements (T2.8)

HAL/S statements play the same role that SPL statements do.

4.2.2.9 Header Statements (T2.9)

Header statements identify the kind of block being declared. Except for the cases of TASK blocks (T2.5) and UPDATE blocks (T2.6), the header statement is similar to the SPL START, PROCEDURE, and FUNCTION declarations. The forms may differ slightly but not the meaning.

4.2.2.10 CLOSE Statement (T2.10)

The CLOSE statement is the closing delimiter of a block. In SPL a program is terminated with the TERM statement, and procedures and functions with the EXIT statement.

4.2.2.11 Name Scope (T2.11)

The name scope rules in HAL/S are the same as those in other block-structured languages.

4.2.3 Declare Group (T3.)

4.2.3.1 REPLACE Without Arguments (T3.1)

The action required by this construct is to replace a character string by another character string possibly of greater semantic complexity. This can be accomplished by text manipulation or by token manipulation. This feature is not available in SPL, but no substantial difficulties are expected.

4.2.3.2 REPLACE With Arguments (T3.2)

This form of the REPLACE statement allows for the selective replacement of textual strings within the source text. Notice that HAL/S specifications allow for nested REPLACE statements with arguments. The nesting of REPLACE statements with arguments leads to serious complications; particularly, since recursive nesting is allowed. In turn, this leads to obscure programs and also to possible interminable loops during the compilation process. However, the amount of nesting allowed is implementation-dependent. It is suggested that nesting of REPLACE statements not be allowed. Even without nesting this feature is difficult to implement.

4.2.3.3 Structure Templates (T3.3)

The structure concept is not available in SPL. It requires a considerable amount of effort to implement because it implies a different method of referencing data than is currently available. For example, it may be necessary to include information about structure templates in the object program.

4.2.3.4 DENSE and ALIGNED (T3.4)

These instructions indicate how data are to be stored. These attributes are of use only in some computers.

4.2.3.5 DECLARE (T3.5)

This is a factored declaration feature that exists in SPL.

4.2.4 Label Attributes (T4.)

4.2.4.1 FUNCTION (T4.1)

The use of FUNCTION in this context is to indicate to the compiler that the identifier associated with the keyword FUNCTION is referenced before the FUNCTION is defined. This device is used to simplify linkages at a later time.

4.2.4.2 PROCEDURE (T4.2)

This feature is similar to FUNCTION (T4.1).

4.2.4.3 NONHAL (T4.3)

This is a keyword indicating an external routine in some other language. This is implementation-dependent in the sense that the implementation must support the other language along with all linkage functions.

4.2.4.4 TASK (T4.4)

This feature is similar to FUNCTION (T4.1). When a task is referenced before definition, this indicates to the compiler that error conditions are to be suppressed.

4.2.5 Type Specification (T5.)

4.2.5.1 MATRIX (T5.1)

This is a specialized array. The compiler can do a certain amount of semantic analysis of occurrences of variables of this type. Checking of operational consistency will prevent certain runtime errors.

4.2.5.2 VECTOR (T5.2)

See comments in T5.1.

4.2.5.3 SCALAR (T5.3)

This is the basic floating point type.

4.2.5.4 INTEGER (T5.4)

Integer type. Implemented as in SPL.

4.2.5.5 BIT (T5.5)

This is similar to the SPL Logical.

4.2.5.6 CHARACTER (T5.6)

This is the same as SPL Textual. Notice that SPL often dictated textual to be used as a secondary type for I/O purposes. This does not appear to be the case in HAL/S.

4.2.5.7 EVENT (T5.7)

The Event type is similar to Boolean, but differs in that identifiers of this type are to be used for real-time applications. HAL/S Event identifiers are similar to the Boolean formulas which must be evaluated during the execution of an SPL Chronic statement. It n ay be necessary to add an attribute value for this data type.

4.2.5.8 BOOLEAN (T5.8)

SPL BOOLEAN and HAL/S BOOLEAN are similar.

4.2.5.9 STRUCTURE (T5.9)

Having a structure type in addition to a structure declaration allows numerous identifiers to be declared to have the same form with terminal and substructure names. Proper referencing will have to be checked during compile time. If checking is to be performed during execution at least two ramifications appear--execution time will be increased and elaborate linkages will have to be developed for data allocation.

4.2.6 Initialization (T6.)

4.2.6.1 CONSTANT (T6.1)

This is analogous to a read only memory. The compiler will ensure that stores are not made to identifiers with this attribute.

4.2.6.2 INITIAL (T6.2)

This is the same as the SPL "Preset" concept. Partial initialization can be achieved at the expense of a more complicated and slower compiler.

4.2.7 Data Referencing (T7.)

4.2.7.1-3 Referencing Nonstructure Variables (T7.1-3)

Simple variables are referenced by name in the same manner in which they are referenced in many other programming languages including SPL. The attributes of the simple variable are known from the data declaration. Variables may also be subscripted as in many other languages.

4.2.7.4 Referencing Structures (T7.4)

References to a structure as a whole are made by the declared name. References to parts of a structure are either qualified or unqualified. The reference is unqualified when the structure name is the same as the structure template name. In this case, reference may be made using the names of the structure parts. A reference is qualified if the structure name differs from the template name. In this case, first the structure name is taken followed by a period, and then the names of the intermediate nodes on the path to the terminal or substructure of interest. All intermediate node names are separated by periods. These two types of referencing permit several different structures with different names to share the same structure template. A reference to a structure terminal is considered the same as a variable reference. Structures are not found in SPL.

4.2.8 Natural Sequence (T8.)

The natural sequence is important whenever a data structure is converted to a linear string of data elements or vice versa. This occurs primarily during I/O and data conversions. The natural sequence defines the order in which the data structure is unraveled or raveled.

4.2.9.10 Subscripting (T9,10)

Three classes of subscripting can occur: structure, array, and component. Therefore, it is possible for a reference to involve three subscripts. The syntax of all three subscripts is identical. The HAL/S subscripting is somewhat more general than SPL in that SPL does not include arrays of vectors and matrices, and SPL does not, of course, include structures. SPL does permit all simple data types except vectors and matrices to be arrayed. HAL/S permits subscripts themselves to be arrays. It also permits subscripting to select subarrays in addition to individual components. In both cases, the permitted subscripting is more general than that found in SPL.

The use of subscripting involves the concept of arrayness. A variable possesses arrayness either from a declaration specification or from being a member of a structure with multiple copies. The number of dimensions of arrayness is the sum of the number of dimensions from both sources. The arrayness of expressions is determined by the arrayness of the variables contained in them. The general rule governing arrayness is that all variables in an expression must possess the same arrayness. For a compiler, semantic checking is implied to determine that arrayness is correct.

The inclusion of the extended HAL/S subscripting capabilities would impact each of the three passes of the compiler. The major impact in the syntactic pass is the extension of the dictionary to contain the new data forms. There are several impacts in the second pass. First, the data allocation routines must be extended. Also, the checks for consistency of data forms must be more extensive than at present. Finally, the checks for proper arrayness must be included. In the code generation pass, looping mechanisms must be established for the processing of arrayed expressions. Provision must be made for the allocation of arrayed temporaries.

The subscripting of structures has a major impact on the compiler because structures are a completely new data form. The impact is primarily in semantic processing since the syntax of structures is similar to existing syntactic forms.

4.2.11 Regular Expressions (T11.)

Regular expressions are arithmetic expressions, bit expressions, character expressions, and structure expressions. Except for structure expressions, which are not in SPL, regular expressions can be processed straightforwardly.

4.2.11.1 Arithmetic Expressions (T11.1)

In HAL/S an arithmetic expression may have matrix or vector operands as well as scalars or integers. This is also true in SPL. In HAL/S the notation A**T (or its two-dimensional equivalent) is used for the transpose of the matrix A. The method of inverting the matrix A is not specified; it is assumed that a standard technique such as the Gauss-Jordan method is acceptable.

The precedence rules for arithmetic operators establish a left-to-right evaluation for operators having the same order of precedence, but exponentiation and scalar division follow a right-to-left order.

4.2.11.2 Bit Expressions (T11.2)

Bit expressions in HAL/S correspond to logical formulas in SPL. The concatenation operator does not exist in SPL. Bit expressions should not present serious difficulties. See also T11.3.

4.2.11.3 Character Expressions (T11.3)

Character operands in HAL/S correspond to textual operands in SPL. There are no textual operators in SPL. The only character operator in HAL/S is concatenation. A concatenation tree could be defined which would be used in bit expressions (T11.2) as well as in character expressions.

4.2.11.4 Structure Expressions (T11.4)

A structure expression is either a structure or a normal function of type structure. If these elements are defined, then the implementation of structure expressions should present no major difficulties.

4.2.12 Conditional Expressions (T12.)

The HAL/S conditional expression corresponds to the SPL Boolean formula.

4.2.12.1 Conditional Operators (T12.1)

Conditional operators in HAL/S are the logical AND and the logical OR; both of which are operators in SPL.

4.2.12.2 Conditional Operands (T12.2)

Conditional operands in HAL/S are comparisons of parenthesized conditional expressions. In SPL they are called relational formulas and Boolean formulas.

4.2.12.3 Arithmetic Comparisons (T12.3)

Except comparisons of vectors and matrices there should be no problems. Comparisons of vectors and of matrices could be processed in a number of ways, none of which requires much effort.

4.2.12.4 Bit Comparisons (T12.4)

Minor difficulties can occur here in comparing bit operands of different length. Arrayed comparisons may also cause problems.

4.2.12.5 Character Comparisons (T12.5)

Arrayed comparisons may cause problems; otherwise, this construct does not present difficulties.

4.2.12.6 Structure Comparisons (T12.6)

If structures are defined, then this construct can be processed in a number of ways.

4.2.12.7 Comparisons Between Arrayed Operands (T12.7)

This construct requires a looping mechanism.

4.2.13 Event Expressions (T13.)

Event expressions are handled very much like conditional expressions. Restrictions and implications need to be considered carefully.

4.2.14 Normal Functions (T14.)

No difficulties are envisioned with this construct. It is not clear if input arguments are call-by-reference or call-by-value.

4.2.15 Explicit Type Conversions (T15.)

4.2.15.1 Integer-Scalar Conversions (T15.1)

These are the most useful conversions and also the simplest to process.

4.2.15.2 Other Unsubscripted Conversions (T15.2)

These conversions require at most minor modifications to the existing system.

4.2.15.3 Matrices and Vectors (T15.3)

These require some, but not substantial, modifications to the existing system.

4.2.16 Explicit Precision Conversions (T16.)

This feature refers to SINGLE and DOUBLE precision. No difficulties are found here.

4.2.17 IF Statement (T17.)

4.2.17.1 IF Statement with Arithmetic Comparisons (T17.1)

Whether the IF statement is of the form IF... THEN, or of the form IF... THEN... ELSE, the use of arithmetic comparisons is taken to be basic.

4.2.17.2 IF Statement with Other Than Arithmetic Comparisons (T17.2)

IF statements with other than arithmetic comparisons can be easily implemented provided the items being compared are already in the language.

4.2.18 Assignment Statement (T18.)

For the most part, assignments can be processed straightforwardly. Some difficulty can be expected in structure assignments.

4.2.19 CALL Statement (T19.)

The CALL statement is basic to a programming language. Minor difficulties may occur with parameters which are structures or arrays.

4.2.20 RETURN Statement (T20.)

This is a basic feature which is easily processed.

4.2.21 DO...END Statement Group (T21.)

The looping statement is a programming convenience which is considered almost basic; particularly, by those who insist on visibility of program organization.

4.2.21.1 Simple DO Statement (T21.1)

The simple DO statement is used to make a basic statement out of one or more statements. This is similar to the ALGOL 60 compound statement. No difficulties are expected here.

4.2.21.2 DO CASE Statement (T21.2)

In SPL, decision tables are used which can be used with or without the ELSE mechanism. Some adjustments may be required.

4.2.21.3 DO WHILE Statement (T21.3)

Test of loop control variable is performed prior to execution of the group of statements to be repeated. In SPL this is the LOOP WHILE statement.

4.2.21.4 DO UNTIL Statement (T21.4)

The DO UNTIL statement requires that the group of statements to be repeated be executed at least once. Minor modifications are required to use the SPL LOOP UNTIL statement.

4.2.21.5 Discrete DO Statement (T21.5)

SPL does not have this statement type. A discrete-DO tree may be required. This is not a substantial effort.

4.2.21.6 Iterative DO Statement (T21.6)

This is the most common form of the looping statement.

4.2.21.7 END Statement (T21.7)

This statement is used to terminate loops.

4.2.21.8 REPEAT Statement (T21.8)

This statement is similar to the SPL TEST statement.

4.2.21.9 EXIT Statement (T21.9)

This is used to escape from a loop; it should be easy to process.

4.2.22 The SCHEDULE Statement (T22.)

The SCHEDULE statement consists of various keywords: SCHEDULE, AT, IN, ON, PRIORITY, DEPENDENT, REPEAT, AFTER, EVERY, WHILE, and UNTIL along with the arguments for the keywords. The compiler will generate code consisting of placement of arguments in registers or a predetermined area of core and a transfer to the system executive routine. Implementation of the schedule statement is not possible without a real-time executive.

4.2.23 Other Real-Time Executive Statements (T23.)

The statements in this category require a real-time executive (RTE) for implementation. If the RTE is available, the compiler will issue code consisting of a transfer to the RTE with the correct parameters stored appropriately.

4.2.24 Error Recovery and Control (T24.)

For routine error control, HAL/S specifies an error recovery executive which handles system-defined errors or user-defined errors. For system-defined errors, the error recovery executive performs a standard procedure unless otherwise directed by the user. A specific implementation may allow the user to define additional error conditions. The HAL/S language allows the programmer to communicate with the error recovery executive. The programmer can direct that a procedure other than the standard procedure be performed for certain system-defined error conditions. It is also possible to send an error condition to the error recovery executive in user-defined errors or to simulate system-defined errors. The HAL/S facilities in this area go beyond those found in SPL.

4.2.25,26 Input and Output Statements (T25,26)

All input and output is machine and operating system dependent. I/O is directed to software channels which are associated with hardware devices in an actual implementation. In addition to sequential I/O found also in SPL, HAL/S provides for random access record oriented I/O. External sequential files are viewed as lines of characters with conversion taking place during I/O. When complex data structures are involved in I/O, the order of conversion to a string of characters or vice versa is determined by the "natural sequence" of the data.

4.2.27 Systems Language Features (T27.)

4.2.27.1 Inline Function Blocks (T27.1)

The inline function is a method of defining and invoking a function at the same time. The syntax of the inline function is similar to that of the

"regular" HAL/S function but is slightly restricted. The inline function can be used to increase the usefulness of the parametric REPLACE statement.

4.2.27.2 % Macros (T27.2)

The % macro allows the addition of special purpose features to the HAL/S language. The details of the implementation of a % macro depend on the particular application. A typical use would be to set up the linkage to external routines. The inclusion of the % macro feature in a HAL/S compiler involves the operating system because of the necessary linkages. % macros may be typed or typeless.

4.2.27.3 TEMPORARY Variables (T27,3)

The TEMPORARY variable form is a directive to the compiler which can be used to aid code optimization. It pertains only to variables used in DO groups and permits assignment of these variables to such items as registers, scratch pad and memory, depending on the hardware of the target computer. Usually, reference to variables stored in this type of location can be made more quickly using a more compact instruction than can references to variables stored in main memory. An equivalent construct is found in SPL. The usefulness of the concept depends on the structure of the target machine.

4.2.27.4 NAME Facility (T27.4)

The NAME facility provides a pointer capability within HAL/S. A NAME identifier always points to the location of an identifier of the same type. Although SPL contains a similar construct, HAL/S contains data types not found in SPL; therefore, to implement the full NAME facility in a HAL/S compiler, new identifier types are required in addition to those found in SPL. Several statements are provided for manipulating NAME variables.

4.3 SUBSETS

The design of a language requires a great deal of effort and an enormous amount of attention to minute details. To some extent, the same can be said of establishing subsets of a given language. Less than an exacting analysis may lead to the production of dangling constructs or mismatched elements.

The above observation is made because the richness of HAL/S data types and data structures magnifies the difficulties of subset definition. It may be desirable to establish a subset by converging from a set of desirable features to a complete set of necessary features during a detailed analysis and design of the compiler for the subset.

APPENDIX

TABLES OF HAL/S FEATURES

Table A-1. Listing of Appendix Tables of HAL/S Features

1. Primitives

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	Character set	Character set	1	1
2	1	Reserved words	Keywords	1	1
3	1	Identifiers	Identifiers	1	1
4	1	Literals	Constants	1	1
5	4	Two-dimensional Source format	----	5	2
6	1	Comments and blanks	Comments and blanks	1	1

2. Block Structure

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	Unit of compilation	Program, procedure, function, and compool	1	1
2	1	Templates, general concept	START, PROCEDURE, COMPOOL, and EXTERNAL declarations	2	2
3	1	PROGRAM block	PROGRAM	1	1
4	1	PROCEDURE and FUNCTION blocks	PROCEDURE and FUNCTION	1	1
5	2	TASK blocks	PROCEDURE or CLOSE	5	2
6	2	UPDATE blocks	INLINE PROCEDURE, CLOSE	4	2
7	1	COMPOOL blocks	COMPOOL block	1	2
8	1	Statement	Statement	1	1
9	1	Header statement	Keywords, attributes	1	1
10	1	CLOSE	TERM and EXIT	1	1
11	1	Name scope	Name scope	1	1

3. Declare Group

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	4	REPLACE without arguments	DEFINE	1	2
2	4	REPLACE with arguments	----	5	3
3	4	Structure templates	----	5	4
4	4	DENSE and ALIGNED	----	5	3
5	1	DECLARE	Factored declarations and declarations	4	1

4. Label Attributes

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	FUNCTION	FUNCTION	1	1
2	1	PROCEDURE	PROCEDURE	1	1
3	4	NONHAL	----	5	3
4	2*	TASK	PROCEDURE	4	3

5. Type Specification

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	2	MATRIX	ARRAY	1	1
2	2	VECTOR	ARRAY	1	1
3	2	SCALAR	Floating-point	1	1
4	1	INTEGER	INTEGER	1	1
5	2*	BIT	LOGICAL	1	1
6	4	CHARACTER	TEXTUAL	1	1
7	2*	EVENT	BOOLEAN formula	3	2
8	2*	BOOLEAN	BOOLEAN	1	1
9	4	STRUCTURE	----	5	4

b. Initialization

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	2*	CONSTANT	CONSTANT	1	1
2	2*	INITIAL	PRESET or "="	1	2

7. Data Referencing

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	Simple variable, unsubscripted	Unsubscripted variable	1	1
2	1	Simple variables, subscripted, except arrays of vectors and matrices	Subscripted variables	1	1
3	4	Simple variables, subscripted arrays of vectors and matrices	----	4	3
4	4	Structure variable	----	5	4

8. Natural Sequence

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	Nonstructures	Array order	1	1
2	4	Structures	----	5	4

9. Subscript Forms

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	Simple index	Simple index	1	1
2	4	AT and TO partitions	FOR loop	4	3
3	4	Asterisk	Blank subscript	1	1
4	4	Subscripts that are arrays	Nested subscripts with FOR loops	4	4

10. Subscript Classes

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	Component subscript	Subscript	1	1
	2*		BIT modifier	1	1
	4		BYTE modifier	1	1
2	4	Array subscript, except arrayed vector and matrix	Array subscript	1	1
3	4	Array subscript for arrayed vector and matrix	----	4	3
4	4	Structure subscript	----	5	4
5	4	Combinations	TEXTUAL and BIT components of arrays	3	3

11. Regular Expressions

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	Arithmetic expressions of scalars and integers	Arithmetic expressions of simple variables	1	1
2	2	Arithmetic expressions of scalars and integers, vectors, and matrices	Arithmetic expressions of simple variables, vectors, and matrices	1	1
3	1	Arithmetic operands	Arithmetic operands	1	1
4	2*	Bit expressions	Bit expressions	1	1
5	2*	Bit operands	Bit operands	1	1
6	3	Character expressions	Byte expressions	1	1
7	3	Character operands	Byte operands	1	1
8	4	Structure expressions	----	5	2

12. Conditional Expressions

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	2	Conditional operators	Boolean formulas	1	1
2	2	Conditional operands	Boolean operands	1	1
3	1	Arithmetic comparisons	Relational formulas	1	1
4	1	Bit comparisons	Logical expressions	2	1
5	3	Character comparisons		2	1
6	4	Structure comparison	----	5	4
7	4	Comparison between arrayed operands	Array comparison	4	4

13. Event Expressions

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	2	Event expressions	Boolean expressions	4	1

14. Normal functions

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	Built-in functions	Built-in functions	1	1
2	1	User-defined functions	User-defined functions	1	1

15. Explicit Type Conversions

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	Integer-scalar	Integer-floating point	1	1
2	4	All other unsubscripted	----	4	2
3	4	Matrices and vectors	----	4	2

16. Explicit Precision Conversions

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	3	SINGLE, DOUBLE	Precision	1	1

17. IF Statement

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	IF... THEN... and IF... THEN... ELSE... with arithmetic comparisons	IF... THEN... and IF... THEN... ELSE... with arithmetic comparisons	1	1
2	2	IF... THEN... and IF... THEN... ELSE... with other than arithmetic comparisons	IF... THEN... and IF... THEN... ELSE... with other than arithmetic comparisons	1	1

18. Assignment Statement

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	Simple and multiple assignment for scalars, integers, vectors, matrices, and other types (bit, etc.)	Simple and multiple assignment	1	1
2	4	Structure assignments	----	5	3

19. CALL Statement

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	General form and semantics	General form and semantics	1	1
2	4	Structures as parameters	----	5	2
3	4	Arrays of vectors and matrices as parameters	----	5	2

20. RETURN Statement

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	RETURN for functions and for procedures	RETURN for functions for procedures	1	1

21. DO...END Statement Group

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	3	Simple DO		5	2
2	4	DO CASE with or without ELSE	Switches or decision tables	3	2
3	2	DO WHILE	FOR and LOOP WHILE	1	1
4	2	DO UNTIL	FOR and LOOP UNTIL	1	1
5	4	Discrete DO	For with no assigned value otherwise no correspondence	3	2
6	4	Iterative DO	LOOP	1	1
7	1	END	END	1	1
8	4	REPEAT	TEST	1	1
9	4	EXIT	----	5	2

22. SCHEDULE Statement

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	2*	SCHEDULE	Set of SPL elements that includes CHRONIC	4	3
2	2*	AT and IN	HARDWARE, CHRONIC	4	3
3	2*	ON	ON...CHRONIC	4	3
4	2*	PRIORITY	System dependent	5	3
5	4	DEPENDENT	System dependent	5	3
6	2*	REPEAT, AFTER, EVERY	ON, CLOCK, CHRONIC	5	3
7	2*	WHILE, UNTIL	ON...CHRONIC	5	3

23. Other RTE Statements

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	4	CANCEL	----	5	3
2	4	TERMINATE	----	5	3
3	2*	WAIT	WAIT	5	2
4	4	UPDATE PRIORITY	----	5	3
5	2*	EVENT control: SET, RESET, and SIGNAL	HARDWARE, BOOLEAN, STATUS, used with CHRONIC	4	3
6	2*	Data sharing and UPDATE block	LOCK, UNLOCK	4	3

24. Error Recovery and Control

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	4	ON ERROR	ON	1	3
2	4	OFF ERROR	LOCK, UNLOCK	4	3
3	4	SEND ERROR	----	5	3

25. Sequential I/O Statements

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	1	READ	READ	4	3
2	4	READALL	READ	4	3
3	1	WRITE	WRITE	4	3
4	3	Control functions TAB, COLUMN, SKIP, LINE, PAGE	Control functions and declarations	4	3

26. Random Access I/O

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	4	FILE	----	5	4

27. Systems Language Features

Item ID	Priority	HAL/S Feature	Associated SPL Feature	Compatibility	Rank
1	4	Inline function blocks	CLOSE, FUNCTION, INLINE	5	3
2	4	% macros	----	5	3
3	4	TEMPORARY variable	INDEX declaration	1	2
4	4	NAME	LOCATION	1	3