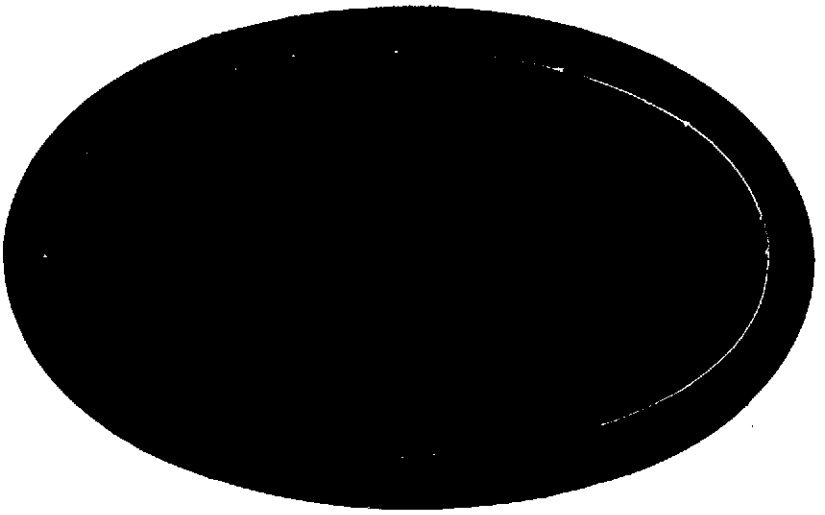
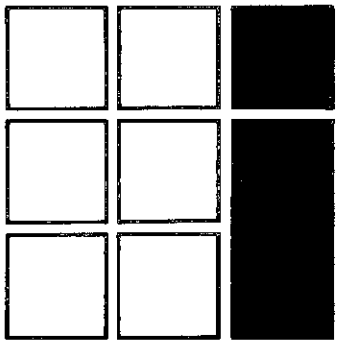
CR-134033



(NASA-CR-134033) GOAL-TO-HAL TRANSLATION STUDY Final Report (Intermetrics, Inc.) ~~85~~ 91 p HC $6.25 CSCL 09B

N73-31141

Unclas
G3/08 14032

INTERMETRICS

## STANDARD TITLE PAGE

| 1. Report No. | 2. Government Accession No. | 3. Recipient's Catalog No. |
|---|---|---|
| | | |

| 4. Title and Subtitle | 5. Report Date |
|---|---|
| FINAL REPORT ON GOAL-TO-HAL TRANSLATION STUDY | June 1973 |
| | 6. Performing Organization Code |
| 7. Author(s) Flanders, J.H., Helmers, C.T., Stanten, S.F. | 8. Performing Organization Report No. |
| 9. Performing Organization Name and Address<br>INTERMETRICS, INC.<br>701 Concord Avenue<br>Cambridge, Mass. 02138 | 10. Work Unit No. |
| | 11. Contract or Grant No.<br>NAS-9-12291, 6c |
| 12. Sponsoring Agency Name and Address<br>National Aeronautics and Space Administration<br>Johnson Space Center<br>Houston, Texas 77058 | 13. Type of Report and Period Covered<br>FINAL REPORT ON TASK 6c |
| | 14. Sponsoring Agency Code |

15. Supplementary Notes

16. Abstract

This report deals with the feasibility, problems, solutions, and "mapping" of a GOAL language to HAL language translator. Ground Operations Aerospace Language, or GOAL, is a test-oriented higher order language developed by NASA's John F. Kennedy Space Center to be used in checkout and launch of the Space Shuttle. HAL is a structured higher order language developed by NASA's Johnson Space Center to be used in writing the flight software for the onboard Shuttle computers. Since the onboard computers will extensively support ground checkout of the Space Shuttle, and since these computers and the software development facilities on the ground use the HAL language as baseline, the translation of GOAL to HAL becomes significant. The report examines the issue of feasibility and finds that a GOAL to HAL translator is feasible. Special problems are identified and solutions proposed. Finally, examples of translation are provided for each category of complete GOAL statement. A companion report is entitled: Final Report on Shuttle Avionics and the GOAL Language including the Impact of Error Detection and Redundancy Management.

| 17. KeyWords | 18. Distribution Statement |
|---|---|
| Test Oriented Language<br>Space Shuttle<br>GOAL<br>HAL | |

| 19. Security Classif.(of this report) | 20. Security Classif.(of this page) | 21. No. of Pages | 22. Price |
|---|---|---|---|
| Unclassified | Unclassified | 84 | |

KSC FORM 16-272 (5/72)

# FINAL REPORT
# ON
# GOAL-TO-HAL
# TRANSLATION STUDY

Change 6c
Contract NAS 9-12291

NOTICE

This report was prepared as an account of Government-sponsored work. Neither the United States, nor the National Aeronautics and Space Administration (NASA), nor any person acting on behalf of NASA:

(1) Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately-owned rights; or

(2) Assumes any liabilities with respect to the use of, or for damages resulting from the use of, any information, apparatus, method or process disclosed in this report.

As used above, "person acting on behalf of NASA" includes any employee or contractor of NASA, or employee of such contractor, to the extent that such employee or contractor of NASA or employee of such contractor prepares, disseminates, or provides access to any information pursuant to his employment or contract with NASA, or his employment with such contractor.

# FOREWORD

This Report has been generated in partial fulfillment of Change 6c to NASA Contract NAS 9-12291. The basic contract has had as its objective the development of a high-order programming language known as HAL to be used as a tool for developing onboard computer software for manned space flight. With the advent of the Shuttle Program, HAL will be used by the teams that will write the flight software for the onboard computer.

At the same time, a high-order language has been developed by NASA's John F. Kennedy Space Center. This language is Ground Operations Aerospace Language or GOAL. GOAL is specifically designed and specified for ground checkout applications.

The onboard computers will be used extensively to support the checkout of the Space Shuttle on the ground. The question of reliably and efficiently implementing GOAL statements in the onboard computers becomes important. Change 6c to the referenced contract is a study task to determine the feasibility of a translator which would take GOAL statements and, in one or more stages, generate verified HAL code which would then be fully compatible with the onboard computer system during ground checkout.

Related topics deal with executive support, particularly achieving compatibility with the Flight Computer Operating System, and the impact of error detection and redundancy management on GOAL. This Report deals with general compile-time and run-time features that must be present in the operating system. A parallel report published concurrently and entitled, Final Report on Shuttle Avionics and the GOAL Language, Including the Impact of Error Detection and Redundancy Management, takes up various options to consider for integrating GOAL-derived code into the onboard computer system and also addresses error detection and redundancy issues.

Note: Chapter 4.0 of this Report satisfies Task 1.3 of the Work Plan, dated 4/10/73. The whole Report satisfies Task 1.4 of the Work Plan.

TABLE OF CONTENTS

# 1.0 INTRODUCTION

## 1.1 Genesis of GOAL

The development of GOAL was brought about by the need for a standard test language to be used for maintenance, refurbishment, checkout, and launch of the Space Shuttle. Apollo experience had already proven the value of computer-automated checkout programs, while at the same time highlighting the importance of early source language capabilities. ATOLL was such a language and was applied to Saturn V checkout and launch.

As the requirements of the Shuttle program unfolded, it was evident that a high degree of checkout computer automation would be required to meet schedule and cost objectives. Furthermore, the opportunity existed to develop a high order language early in the program so that, from the beginning, it was an integral part of the system. Requirements contracts were let by KSC in July of 1970. In May 1971, a language requirements document was published (KSC-TR-111). Currently, three documents have been published which define the language. These are the GOAL Overview Document, the Syntax Diagrams Handbook (KSC-TR-1213), and the GOAL Textbook (KSC-TR-1228). Also, a GOAL compiler is currently being developed.

## 1.2 Genesis of HAL

The development of HAL was stimulated by the same combination of Apollo experience and anticipated Shuttle requirements that stimulated GOAL, except that HAL is oriented towards the onboard mission software for manned spaceflight with its great emphasis on 1) the mathematical requirements of navigation, guidance, and control and 2) the need for highly reliable real time control programs. Apollo experience had shown that the resources needed to program mission software in assembly language in a multi-program environment were excessive.

Development of HAL began with a contract let by JSC early in 1970. This contract supported the generation of requirements, a survey of other languages, synthesis of a new language, and the building of a HAL compiler to run on the IBM 360/75 at the JSC Real Time Control Center. This effort was augmented a year and one-half later by a JSC contract to advance HAL to an operational status. As a result of this last contract, the HAL language was ready when the decision to specify the onboard software for Shuttle came up, and HAL was chosen as the language in which the software will be written.

## 1.3 The Translation Problem

The translation of GOAL statements into HAL/S* is the subject of this section. The general problem of such a translation involves two aspects: 1) identifying simple parallel forms in the two languages and performing a mathematical syntax conversion; and 2) identifying more fundamental semantic differences which will require the synthesis of functional groups of statements in the target language, HAL, corresponding to forms of the source language, GOAL. This latter problem will prove the more complex part of the process.

As an example of each, consider two classic types of translators: assemblers and Fortran compilers. An assembler is fundamentally a simple program to make a one-to-one mapping from its source, symbolic machine instructions, to its target, machine code. In contrast, the Fortran compiler must translate a much more abstract source by synthesizing multiple primitives of the target language to accomplish the functions of its source language.

Of necessity, this Report is primarily concerned with the semantics, the meaning of actions performed by the GOAL statements, and the proper translation of GOAL semantics into corresponding HAL/S forms. Syntax, by itself, cannot be translated without reference to semantics, unless of course the languages are completely identical in their functional descriptions. GOAL and HAL/S are alike only to the extent that they are both computer languages; thus they share a common general framework of actions. However, their detailed functional characteristics are sufficiently different to make a mere mechanical syntax conversion out of the question except in special cases, which will be noted in the text which follows.

Also, many features of GOAL, particularly among those designated System Statements, are not actually language features. Instead, they represent requirements which will be applied to any operating system upon which GOAL is to be implemented. This topic is brought up in several appropriate places in this Report.

---

* HAL/S = HAL Shuttle, or HAL for the Shuttle Onboard Computer(s).

# 2.0 FEASIBILITY OF TRANSLATION

The GOAL-to-HAL feasibility discussion is divided into two main technical areas. The first deals with SYNTAX and semantic questions. The second area addresses other feasibility questions relating to compile and run time.

## 2.1 Syntax and Semantic Level

This section demonstrates that a syntax mapping from GOAL to HAL is feasible. The content reflects the results of the first effort at demonstrating feasibility and as such does not correspond exactly to the indepth mapping presented in Section 4. The differences arise due to the fact that there are a number of ways in which the translation may be implemented.

### 2.1.1 Global Block Structure

There are two major block forms in GOAL which may be defined:

a. The DATA Bank. This particular block identifies a rough equivalent of the HAL/S COMSUB/COMPOOL mechanism. Its purpose (as in the HAL/S forms) is to provide system-wide global information. However, unlike the HAL/S forms of COMPOOLs and COMSUBs, the GOAL Data Bank is in effect an extra-lingual data management system used to interface the GOAL translator or compiler with the world of the Operating System and the various hardware units to be controlled by test programs. It is available for symbolic reference by the compiler in the course of interpreting function designator names.

b. The PROGRAM. This particular GOAL block form identifies the equivalent of the HAL/S program block. It is a delimited block of code which stands alone for purposes of compilation.

A GOAL compilation, then consists of the processing of source code for a single program and its references to a data bank. This corresponds roughly to the HAL/S equivalent

of a set of COMPOOL/COMSUB templates (i.e., data bank) followed by a single outer block compilation.

Within the DATA BANK or PROGRAM blocks of a GOAL compilation, a single level of nesting of additional modules is allowed. (In no case is static nesting of GOAL blocks allowed beyond the single level to be described here.) These modules are potentially either subroutines for run-time execution, macros for compile-time substitution, or non-GOAL subroutines in object form for use at run-time.

a. The GOAL subroutine block is the equivalent of a HAL/S Procedure block without "input parameters". It is allowed any number of "ASSIGN" parameters. It is invoked by a PERFORM SUBROUTINE statement. The detailed correspondence of HAL/S and GOAL versions of this block form will depend upon whether or not subroutine invocation is limited to the "PERFORM" statement.

b. The GOAL macro block is a compile-time routine used to generate source code. The GOAL form of a MACRO is a simple replacement with no "conditional assembly". The macro is a skeleton with optional parameters. When it is invoked, the textual parameters are substituted and the completed macro is then compiled as if it were coded directly. The GOAL macro is thus effectively a HAL/S "REPLACE" with the addition of parameter substitution within text.

c. The non-GOAL subroutine is a data bank subroutine written in a language other than GOAL, presumably HAL or machine language. It may be invoked by the PERFORM SUBROUTINE phrase in the same manner as regular GOAL subroutines. A non-GOAL subroutine is assumed to be a separately compiled or assembled element and therefore enters the translation and compile process only by adhering to predetermined linkage conventions. A non-GOAL subroutine is incorporated into the compiled code by the link editor.

The global block structure of a GOAL program can thus be described in a schematic form as follows:

```
DATABANK
    SUBROUTINE
        MACRO
        STATEMENTS
    NON-GOAL SUBROUTINE
    MACRO
        STATEMENTS
    STATEMENTS
PROGRAM
    SUBROUTINE
        MACRO
        STATEMENTS
    MACRO
        STATEMENTS
    STATEMENTS
```

### 2.1.2 Data

2.1.2.1 Data Types. GOAL has four data types:

a) Number

This is a very simple data type with a fixed or floating point representation depending on the machine characteristics. It is analogous to a HAL/S scalar type. It contains no implicit dimensions and stands alone as a single element unless included in a table or a list.

b) Quantity

This is very similar to a HAL/S scalar type, but it has explicit units associated with it. Thus, a quantity is a scalar number of volts, a scalar number of ergs, a scalar number of pounds per square inch, etc.

c) State

This is a direct equivalent of a HAL/S Boolean. It is a single, unarrayed bit which can have a value of true or false, on or off, and open or closed.

d) Text

The text data type is the direct equivalent of a HAL/S character string data type. GOAL uses a qualification on a declaration of this data type to set a maximum length for input only, just as in HAL/S.

The problem of translating GOAL data types into a HAL/S equivalent is fairly simple. GOAL's number, state and text data types have direct HAL/S equivalents: scalar, boolean, and character respectively. Quantity data is, however, characterized by two aspects, a number in GOAL terminology and some units.

At present, the dimensions of a GOAL quantity are not used at all except for documenting printouts. The use of dimensional analysis is not contemplated in the GOAL language and therefore need not be dealt with in the HAL version. Scaling of units of a given dimension may be automatically performed in GOAL. (For instance, millivolts, volts, kilovolts). In translation to HAL this would turn into a scaling factor applied to a floating point scalar. It is suggested that quantity dimensions be placed in a special table, with one entry corresponding to each quantity type variable. Thus, when I/O is done with a quantity, a look-up in this dimension table will enable the dimension to be found and printed on output appropriately.

2.1.2.2 Data Organization. GOAL has two forms of organizing data: these are lists, and tables. The list form of data organization is roughly the equivalent of a HAL/S array or a structure with one dimension of arrayness. In either case a list must have only one dimension in GOAL. A GOAL table is somewhat more complicated and is indexed via name, and function designators.

The table organization in GOAL is a two-dimensional array of rows and columns. The column is addressed by a column name or index and the row is addressed by a function designator.

In the GOAL manual, lists are classified according to the GOAL data type in question. Thus, one may have a numeric list, a quantity list, a state list, or a text list. Mixtures of types are forbidden within a given list, as in a HAL/S Array.

a) DECLARE NUMERIC LIST

The GOAL declare numeric list statment would map directly into a scalar array in HAL/S. This form effectively declares an array with some length and some optional initialization just as it would be done in HAL, but with the syntax of GOAL.

b) DECLARE QUANTITY LIST

The declare quantity list statement in GOAL declares a list of quantities or an array of quantities. The equivalent would be 2 arrays, one for the numeric and one for the dimension information. For example, the statement:

```
DECLARE QUANTITY LIST(LIST A) WITH 3 ENTIRES;
```

would translate in HAL/S into the following statement:

```
DECLARE LIST_A ARRAY(3) SCALAR,
   DIM_LIST_A ARRAY(3) CHARACTER(6);
```

This assumes all dimensions can be specified in six characters or less.

c) DECLARE STATE LIST

A state in GOAL is a single, unarrayed, unstrung boolean quantity. This maps into the HAL/S boolean form. The declare state list statement sets up an array of these GOAL states. Consequently, the declaration of a state list will map into the declaration of an array of booleans. In view of the discrete nature of these states, it might be more appropriate to map them into an array of booleans in HAL/S. As an example of the translation which is possible, consider the following statement in GOAL:

```
DECLARE STATE LIST(LIST 3) WITH 3 ENTRIES OFF, OFF, OFF;
```

which would translate into the following HAL/S statement:

```
DECLARE LIST_3 BOOLEAN ARRAY(3) INITIAL (OFF,OFF,OFF);
```

d) DECLARE TEXT LIST

The declaration of a text list in GOAL will be directly translatable into declaring an array of character values or character string data type in HAL/S. Thus, for example, consider the following GOAL statement:

```
DECLARE TEXT LIST (INPUT OUTPUT MESS) WITH 2 ENTRIES
(PLACE ABOVE SWITCHES AS INDICATED),
    (SWITCH SCAN IN PROGRESS)
WITH A MAXIMUM OF 36 CHARACTERS;
```

This would translate in HAL/S to the following statement:

```
DECLARE INPUT_OUTPUT_MESS_CHARACTER(36)ARRAY(2)INITIAL
('PLACE ABOVE SWITCHES AS INDICATED','SWITCH SCAN IN
PROGRESS');
```

The second major form of organization of data in GOAL is the Table. This is an internal organization effectively in a row/column format combined with an implicit action keyed to some "function designators" defined elsewhere in the "Data Bank" of the GOAL program. The "function designators" identify rows in this row/column format and a user-supplied name identifies columns within this format. The column name is optional and may be left out uniformly throughout the whole table when it is declared. If column name is left out, then columns may be referenced via subscripting. A table can be declared with no data columns which leads directly to the interpretation as if it were "an array of functions". Each function designator is, in effect, a hook to some piece of code or data type which may be internal or external to the computer. It is assumed to exist within some GOAL compilation called the data bank and its name must be available to the compiler as a piece of global information.

Another characteristic of these tables is that they have a specific data type which must hold throughout the whole table. There are quantity tables, numeric tables, text tables, and state tables. A quantity table has quantities for every entry into it, or a numeric table has numeric GOAL data types associated with every entry. The table name itself and all the column names are internal names in that they are programmer-defined. It is only the row names (i.e., the function designators), which are externally defined and global. The actions which can occur

when you execute statements referencing these tables are a READ, a WRITE, ASSIGNMENT, etc. These statements define and use the values of function designators. The data portion of the table is the place to keep copies of the data after having defined them with function designators. The important characteristics of these tables then is that function designators implicitly fit into an I/O scheme and are interpreted whenever they are used in a GOAL statement that is to be converted to HAL. An example of the equivalent form is in Chapter 4.0.

### 2.1.3 Procedural Statements

2.1.3.1 Prefixes. The GOAL language procedural statements execute in the usual sequential fashion of a programming language unless modified by explicit means. The key to modifying the order of execution in GOAL is the procedure statement prefix form. Any or all of three prefixes may be applied.

1. A "step number" is used to reference the statement as a label.

2. A "time prefix" causes a delay or a wait for some absolute time.

3. A "verify prefix" is used to turn a statement into the equivalent of an "IF" statement.

Both the time prefix and the verify prefix are inherently simple to translate into HAL/S due to the fairly local effect. The step number, because of its global significance in the GOAL program, will ultimately require a more complicated treatment.

#### a) The Step Number Prefix

The step number prefix of a GOAL statement may translate into a HAL/S label or set of labels depending upon the type of statement being translated. The translation is direct and generates but a single label. Of course, the translation algorithms will, in certain instances, generate other labels for implementation purposes.

b) The Time Prefix

The translation of a GOAL time prefix into HAL/S reduces to the insertion of an appropriate WAIT statement.

c) The Verify Prefix

The translation of a verify prefix into HAL/S will depend upon its form. An IF...THEN verify prefix will translate into a simple HAL/S IF statement, possibly with a simple DO...END group to surround the translated statement. The verify prefix will turn into an IF statement employing the results of a comparison, an optional time dependent true-part enclosed in a DO-END group, and an optional ELSE clause used for an "output exception". For instance,

```
VERIFY <TEST POINT 3> IS BETWEEN 3 AND 5
  ELSE....
```

Becomes

```
IF TEST_POINT_3<3|TEST_POINT_3>5 THEN......
```

where in the HAL/S version the function designator <TEST_POINT_3> has been replaced by a HAL/S function block which returns an appropriate value of the current status of the test point.

2.1.3.2 External Test Action (Command/Response). "Command/Response" is the GOAL term which means "I/O". The GOAL language employs APPLY, ISSUE, SET and RECORD statements to send data to the system under test. It reads responses with the AVERAGE, READ and REQUEST KEYBOARD statements. The general address term used with all these statements is the "function designator", which can be interpreted as an external data address, an I/O unit channel, a discrete bit line, etc. depending upon the GOAL Data Bank definition. In all cases, when the function designator is referenced by one of these statements an appropriate linkage will be generated to read data from or output data to the I/O device in question.

Another way in which I/O through function designators is performed implicitly is by reference in a statement.

Command/Response statements should provide no great difficulties in translating to HAL/S assuming that the full meaning of a GOAL Function Designator is made explicit in each context of the GOAL language.

The most complicated (in terms of function) of the command/response statements is the Average statement, used for input of analog readings with a built-in averaging function. This statement can be generated in-line in HAL/S employing an iterative DO FOR loop surrounding appropriate calls to input routines for successive values of the reading being averaged. All the other Command/Response routines involve single operations upon data elements or lists of elements.

2.1.3.3 Internal Sequence Control. The internal sequence control statements of GOAL are discussed in this section.

The DELAY statement of GOAL takes two forms which have the verbal interpretations of: a) "Wait x time intervals" or b) "Wait until some condition is satisfied". The GOAL syntax allows both forms to be in the same statement. Also possible in GOAL is a specification of the units of time in every time referencing statement. The DELAY statement will become a WAIT statement in HAL/S with appropriate conversion of time to the HAL/S clock units, and use of an event expression for the function designator.

The GO TO statement of GOAL maps directly into the GO TO of HAL/S with no change. The GOAL language restricts GO TO targets to the local block so there is no semantic difference between the two languages.

The STOP statement presents a problem in translation: there is no semantic form of HAL/S which allows any statement to be executed subject to operator command. The translation of this form will have to be implemented by some form of hook into the run-time facilities to symbolically look up restart points within the program. The restricted case of an "indicate restart" will not require this treatment since a series of labels is explicitly listed, and can therefore be generated with the program.

The GOAL TERMINATE statement will have effects which depend upon its exact position in the procedure and its form. The simple TERMINATE statement will always translate

into the HAL/S RETURN statement, since it unconditionally implies this action. The TERMINATE SYSTEM statement of GOAL is in contrast a complete system stop and no such real form exists in HAL/S. One mechanization of this form would be to require that all processes in the equivalent HAL/S system be scheduled from a common HAL/S program which, following scheduling, reaches its CLOSE statement and an implicit WAIT until all dependent processes are finished. If all the programs are scheduled dependently, then if this central program dies then all its derivative processes die as well. Any dependent process can signal a request to cancel the whole test system by a HAL statement:

```
TERMINATE_GOAL_MASTER_PROGRAM;
```

With this approach, however, all tasks and programs scheduled would have to be scheduled in a dependent manner. This topic is discussed further in Section 3.2.

2.1.3.4 Discussion of REPEAT. The REPEAT statement in the general case is a problem. The GOAL language has a form of internal label called the "STEP NUMBER" which is used to identify individual executable statements within the GOAL test procedure or subroutine in question. In the translation of GOAL into HAL, these labels will have to be translated in a manner which is to a large extent dependent upon the global context of the procedure. In certain cases, a step number will translate directly into a simple statement label in the corresponding HAL program, provided that the original GOAL program references the step only from a GO TO context. In the more general case of a label referenced from GOAL's reiterative REPEAT statement, the use of global synthesis and reorganization of code will be required in order to handle all the possible cases. This latter case is intimately bound up with the problem of translating the GOAL repeat statement, and forms a major topic of Section 3.0.

2.1.3.5 Arithmetic/Logical Operations. The arithmetic/logical operations of GOAL are performed by the ASSIGN and LET EQUAL statements for state and arithmetic data respectively. There is no such thing as character assignment in HAL.

Both forms of statements are equivalent to the assignment statement of HAL/S but with the additional restrictions. First, the ASSIGN statement of GOAL applies only to the STATE data type, and involves no possibility for Boolean expressions involving state data. It is thus the equivalent of a HAL/S assignment in which a BOOLEAN on the left is set equal to a BOOLEAN on the right, or a BOOLEAN function, or a single bit literal. No mention is made of whether or not this form of an assignment may be arrayed in GOAL.

The LET EQUAL statement is the equivalent of a HAL/S arithmetic assignment statement, in which a suitable arithmetic term on the left is set equal to an expression on the right. The forms of expressions legal in GOAL are a subset of the possible forms of arithmetic expressions in HAL/S.

2.1.3.6 Execution Control. The execution control statements of GOAL include the CONCURRENT, RELEASE, PERFORM SUBROUTINE and PERFORM PROGRAM statements. The HAL/S equivalents of these statements are described in this section.

The simplest translations in this set of GOAL forms are PERFORM statement variations. The PERFORM SUBROUTINE statement of GOAL is available in two forms: critical and non-critical. The semantic interpretation GOAL gives to CRITICAL is "of highest priority". In either case, the basic mechanization of the PERFORM subroutine is via the CALL statement; with the subroutine itself translated into a HAL/S procedure.

In the case of a critical subroutine, the CALL must be preceded by what is effectively an "inhibit all software interruptions" action. This can be accomplished in HAL/S by updating the priority of the current process to a special "critical" priority which is higher than the priority of all interrupt routines which must be masked. Thus, the "PERFORM CRITICAL SUBROUTINE" statement of GOAL would be equivalent to the following HAL/S sequence:

```
SAVED_PRIORITY = PRIO;
UPDATE PRIORITY TO CRITICAL_VALUE;
CALL xxxxx;
UPDATE PRIORITY TO SAVED_PRIORITY;
```

The temporary SAVED_PRIORITY is used to record the current priority (the PRIO function reference) prior to resetting priority to the CRITICAL_VALUE during execution of the critical subroutine (xxxxx).

The PERFORM PROGRAM statement of GOAL is equivalent to the following sequence of HAL/S code which schedules the program in question and then waits for its completion:

```
SCHEDULE PROGRAM_X;
WAIT FOR ¬PROGRAM_X;
```

This sequence assumes that the process event declaration "DECLARE PROGRAM_X EVENT;" had been given previously.

The CONCURRENTLY PERFORM PROGRAM statement of GOAL is effectively equivalent to a HAL/S SCHEDULE statement for the program in question, subject to the variation that in GOAL the step number of this statement serves as a "process identification" when referenced in the RELEASE statement. GOAL has no provisions for priority, distinguishing independent or dependent processes, or making conditions upon the scheduling other than those needed for cyclic activation. The only question to resolve in connection with the translation of this statement is whether or not multiple CONCURRENT statements using the same program name are possible, in which case HAL/S is incapable of doing the appropriate scheduling because HAL/S allows only one process to be scheduled off one program or task name at any given time. This issue is discussed further in Section 3, where an indirect approach using dummy task blocks is introduced.

In the case of the CONCURRENTLY VERIFY and/or the CONCURRENTLY DISPLAY statements, the HAL/S equivalent becomes much more complex due to the fact that these statements define small concurrent tasks which are implicitly scheduled at the same point as the definition. The HAL/S handling of this statement would consist of generating a TASK block in line containing the actions specified (verify or display), then SCHEDULING the task so defined with the same priority as the original task. In this case, the RELEASE statement would translate into a HAL/S TERMINATE directed at the process so generated and scheduled.

2.1.3.7 Interrupt Control. The GOAL interrupt control statements consist of WHEN INTERRUPT and DISABLE INTERRUPT statements. The translation of these forms into the HAL/S language is the subject of this section.

The interrupt forms of GOAL involve either or both of a GO TO action and a "PERFORM" action. Of these two, the only available HAL/S form is the "PERFORM" equivalent. This is realized in HAL/S by issuing a SCHEDULE statement with an EVENT expression corresponding to the interrupt in question as the activation criterion and with a priority higher than any non-interrupt routine. Once the SCHEDULE has been given, the interrupt action in question is effectively a real time queue member set to activate whenever the event expression becomes true. The complementary action of the GOAL "INHIBIT" interrupt statement would be achieved by issuing a "CANCEL" statement in the corresponding HAL/S program.

The GO TO form of an interrupt action is not directly possible in HAL/S. With some difficulty (assuming this form of interrupt is desirable at all) it will be possible to implement the GO TO action by creating appropriate systems routines to raise a "pending interrupt" error condition which can cause an ON ERROR statement with the GO TO action to be activated. In such a case, the interrupt event should be checked for at the target of the GO TO, and the event must be latched; the INHIBIT action in such a case could consist of a HAL/S ON ERROR... IGNORE statement.

To conclude, a WHEN INTERRUPT...PERFORM statement of GOAL maps into a SCHEDULE statement of HAL/S, with the corresponding INHIBIT performed by the CANCEL statement; and the WHEN INTERRUPT... GO TO of GOAL maps (in conjunction with systems routines) into a HAL/S "pending interrupt" error code amenable to the ON ERROR...GO TO statement, with the ON ERROR...IGNORE performing the inhibit function.

2.1.3.8 Table Control. The ACTIVATE and INHIBIT table statements concern control of function designator use when referenced from a GOAL table. They essentially control whether or not the particular function designator and its row will be included in a reference to the whole table. Given the translation of a GOAL table into HAL/S equivalent forms as outlined in Section 4, the ACTIVATE and INHIBIT statements present no difficulties and have direct equivalents in the HAL/S code (see 4.2.7).

### 2.1.4 System Statements

The GOAL Language System statements are primarily concerned with the control of a GOAL compilation on translation. As such, their effects are generally confined to the translation process and are not evident in direct form in the translator's output. Note, however, that analogous HAL/S forms may be used in order to best generate HAL/S equivalents of various GOAL constructs as a matter of convenience. The problem of interpreting GOAL system statements is left to Section 3, in which a more detailed discussion of the system aspects of translation is presented.

### 2.1.5 Feedback Loop Limitations

There are no inherent HAL syntax feedback loop limitations, which would be binding on GOAL. HAL syntax feedback loop limitations are implementation dependent.

## 2.2 Other Feasibility Issues

Section 2.1 has presented various observations concerning the translation of GOAL programs into a form acceptable to the HAL compiler. This section will discuss the broader problem of running HAL-written and GOAL-written programs in the same computer. In order to give broad scope to consideration of feasibility, the discussion includes mention of inflight checkout situations. The purpose of the discussion is to introduce and discuss potential problem areas within the context of an overall feasibility determination.

### 2.2.1 Combining GOAL and HAL Programs

When considering combining GOAL and HAL programs into a single software system one may approach the problem from several points of view. All of the following possibilities assume that HAL is the major language and GOAL programs are used only for checkout.

a. GOAL to HAL Translator. This method transforms all GOAL data and program statements into corresponding HAL data and program statements. The results of the translation process are HAL program and compool statements which are used as input to the HAL compiler. Neither the HAL language nor compiler are modified. The overall process of creating, running and debugging a GOAL process in this environment is depicted in Figure 2-1.

b. GOAL to HALMAT Translation. The HAL compiler is segmented into two distinct portions as shown in Figure 2-2. The syntax analyzer is essentially machine-independent and accounts for the major portion of the compiler. It parses the source code, checks for legal HAL constructs, provides formatted HAL listings, and performs all the programmer interfaces required to produce reliable static code. The output of the syntax analysis is in the form of a standard intermediate language, called HALMAT.

   The code generator part of the compiler transforms HALMAT into the machine code. As such, it embodies

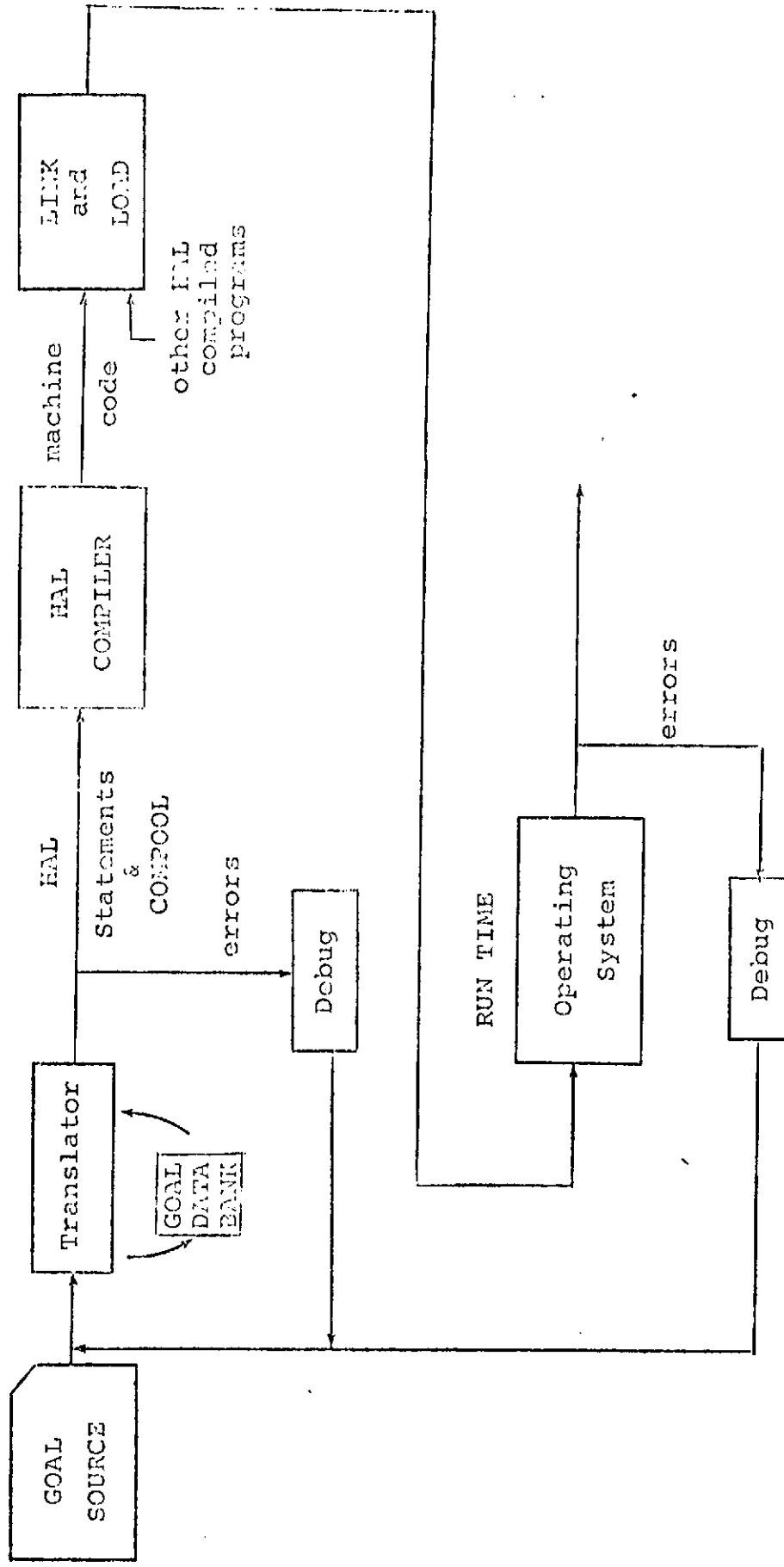
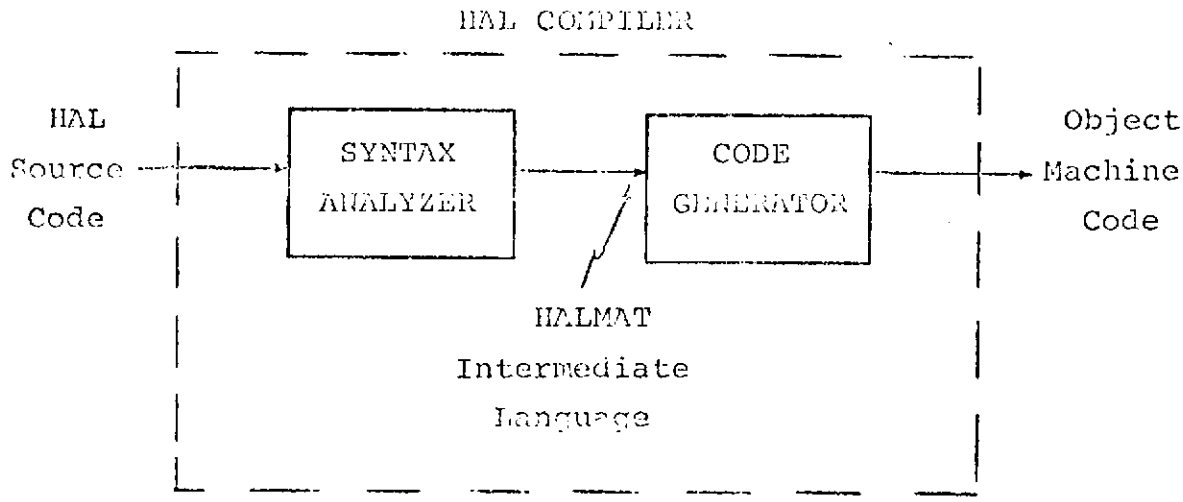



Figure 2-1
GOAL-HAL TRANSLATION

Figure 2-2

HAL COMPILER



all the machine-dependent features of the compiler. Instead of translating GOAL directly into HAL, it is possible to translate GOAL into HALMAT. An argument may be made for this case by realizing that the translator is a piece of software, which is assumed to be error free. It cannot produce incorrect HAL syntax. Therefore, many of the features of the HAL syntax analyzer would never be exercised for mechanically translated GOAL. Any of the source listings and software error reporting features of the HAL compiler would only produce indications relating to HAL source code. This would be very difficult for a GOAL programmer to understand.

All the GOAL error detection and output listing features would have to be implemented in the translator anyway. For these reasons one may seriously consider producing HALMAT directly from the GOAL translator. The GOAL translator would become the syntax analyzer of the GOAL compiler and would product HALMAT as an intermediate language.

c. GOAL as a Subset of HAL. A possibility exists that GOAL could be made a subset of HAL. One may eliminate this approach almost immediately by realizing that the syntactic constructs of GOAL and HAL are quite different. A major modification to the HAL language would have to be made. Unless GOAL programs constituted a large portion of the programming, this approach would be costly and could delay the present plans for HAL implementation.

d. GOAL Interpreter Written in HAL. Another possibility exists, whereby GOAL programs are executed interpretively. The interpreter would be written in HAL. In a sense, this interpreter can be viewed as a run-time GOAL-HAL translator. However, implicit in this approach is some sort of off-line syntax analysis for GOAL software error detection and output listing.

One might decide to use this interpretive approach for two reasons. First, if a large amount of GOAL code is anticipated then a potential memory savings is possible. Second, the interpretive approach would make the dynamic execution of GOAL

statements more controllable by the operator. The penalties are, of course, speed of execution and the cost of developing the interpreter.

The Intermetrics work statement specifies the GOAL-HAL translator approach. The use of HALMAT can still be considered in this context. For the most part, the translator approach is assumed in the remainder of this section. However, if GOAL were to be combined with HAL into a single operating environment then the question discussed above should be given more detailed thought, and HAL interpretive execution of GOAL is also a candidate approach.

### 2.2.2 Definition of a Translator

There are a number of requirements which a translator must meet. This is independent of whether the translation is made into HAL or HALMAT. Three basic language elements must be translated; data types, procedural statements, and system constructs.

a. Data Types. Both GOAL and HAL possess their own data types. A definition of how to represent the unique GOAL data types in terms of HAL data types must be made before the translation of procedural statements is performed. For example, GOAL numeric, quantity state and text elements must find their counterpart in HAL. Lists and Tables must be constructed out of HAL arrays and/or structures.

b. Procedural Statements. After the data types translations have been made then a correspondence between the various procedural statements must be determined. This correspondence involves GOAL command, response, sequence control, arithmetic, execution control, table control statements.

c. System Constructs. This is an area, on a higher logical plane than data and procedural statements, which must be considered in order to produce satisfactory GOAL execution. System constructs involve the translation of the fundamental GOAL concepts such as the Data Bank, GOAL Programs, and some of the GOAL interactive control sequences. These concepts form the environment in which GOAL statements compile and execute.

Many of the system constructs found in GOAL are normally not present in other languages since they are services provided by an operating system. For example, the idea of REVISION numbers is a GOAL semantic features. Conventionally, this is handled by the operating system's data management function. By introducing revision control into the language directly some sort of data management system is implicitly assumed. This must be provided by the operating system and forms a major interface with the GOAL-HAL translator. Discussion of these system construct issues is deferred to Section 3.0 and its discussion of Special Problems.

### 2.2.3 Run Time Issues

One of the main reasons for considering a GOAL-HAL translation process is to force HAL-generated code to be the only software interface with the operating system. However, in writing GOAL programs one cannot in reality, ignore the operating system. The operating system provides a large number of services to applications level software (a GOAL program being a particular application level module). For example, the operating system provides a timer control, process control, I/O services, data management control, memory control, etc. The GOAL programmer must be aware of which GOAL features are supported by the operating system and which features are not. This could potentially be a limiting factor.

### 2.2.4 Inflight Operation

The major emphasis of the present study is to demonstrate the feasibility of a GOAL-HAL translation process to be used on the ground for pre-flight checkout. That is, while the vehicle is on the ground the onboard computers would participate in the checkout process.

If, indeed, the idea proves feasible and cost-effective, then one must naturally inquire about the possibility of using GOAL for onboard checkout during flight and orbital operations as well as on the ground.

There are a number of fundamental differences between ground and flight operations. These differences make the use of GOAL for ground-based checkout much simpler than during flight. Some of these issues are:

a. Efficiency. One might inquire into how much efficiency is lost going through a translation process from GOAL-to-HAL. Efficiency is defined in terms of space (amount of memory used) and time (execution time of program). Efficiency becomes a big issue for in-flight programs especially memory size. On the ground memory overlap may be used, thus compensating for an inefficient translation process. Ground operations can employ a simple straightforward translation with minimal regard for efficiency. In-flight GOAL programs would require some sort of optimization process to utilize fully the limited resources of the flight computer.

b. Flight Qualified Software. A program executed during flight would have to be flight-qualified. This includes GOAL programs as well. Flight verification can be a very expensive process. However, ground checkout software need not be so rigorously verified.

c. GOAL-HAL Partitioning. If both GOAL and HAL were used in flight then questions arise as to which processes are written in GOAL and which in HAL. Where are the interfaces between the two languages? The utilization of two languages in flight can only complicate the control exercised by the operating system. Whether the readability and ease of programming which GOAL provides to the test engineer (and presumably which HAL doesn't provide) justifies the complication introduced by a two language system, is not clear.

### 2.2.5 Debugging GOAL-Translated Programs

The introduction of GOAL programs into a HAL-compiled environment produces some interesting problems during the software development process. Software errors are discovered both statically (by a compiler) and dynamically (by the Operating System or the software designer).

The software error detection normally provided by a compiler must be provided by the translator. See Figure 2-1. These error types include any syntax-related errors and inconsistencies. By definition, the HAL compiler would not detect any GOAL syntax errors since it is assumed that the output of the translator is perfect.

An error discovered during run time by the operating system must, of necessity, be related back to either HAL statements or variables. It would be extremely difficult to relate them to GOAL statements and variables. Any variable trace would have to be specified in terms of HAL variables and not GOAL variables. GOAL programs are two languages removed from the machine. One can appreciate the debugging problem by realizing that the GOAL-HAL translation process can be performed off-line. The results are HAL statements. The logic expressed in GOAL, must obviously, be maintained in the translated HAL form. However, the readability of the HAL produced by the translator will not be as clear to the GOAL programmer as the original GOAL statements. Debugging is made more difficult by the translation process.

### 2.2.6 Software Reliability Issues

One of the major attributes of the HAL language are those features which tend to produce code more reliable than that produced at the machine level by hand. Some of the HAL reliability features include:

a. Block Structures and Name Scope

b. COMPOOL and Access Control

c. Various Debug and Trace Features

d. Explicit Declaration of all Data Elements

In developing a combined GOAL-HAL environment one must make certain that none of the reliability-enhancing features of the HAL language are compromised. A combined language system is possible only if it is implemented with care and with full appreciation for all the features of both languages as well as careful resolution of the conflicts between the languages.

# 3.0 SPECIAL PROBLEMS AND SOLUTIONS

## 3.1 Individual GOAL Features

### 3.1.1 Revision Control

A revision number may be applied in specifying a program or a data bank name. BEGIN DATA BANK and BEGIN PROGRAM must use a REVISION suffix following the name. The invocation of a program or data bank by means of a PERFORM, USE, or FREE statement may optionally utilize the revision number in conjunction with the name.

If the revision number is always used then it becomes a fixed part of the name in the HAL/S equivalent form. However, if the revision number is not employed then either the run time operating system, the linking loader, or some special pre-run program must resolve the addressing problem. The GOAL manual indicates that when the revision number is not employed then the first program or data bank that the language processor encounters will be used regardless of the revision label.

The following suggestions are offered concerning revision control:

a) No run time decisions should be made by the flight computer operating system based upon revision number. Such run time decisions inherently create unreliable software situations. The program name including revision number must be determined statically prior to run time.

b) If revision labels are used, they must be used all the time. A program or data bank should always make reference to the revision label. There should be no implicit reference. This essentially makes the revision label part of the name.

### 3.1.2 STOP and Restart

There are two forms of STOP in GOAL, restricted and unrestricted. The restricted form is given a specific list of start-up points. The unrestricted form allows arbitrary statement numbers to be selected for start-up points. Both forms are equivalent to an appropriate computed GO TO or similar form following a simple halt. This could be implemented in HAL, with a look up table and the DO CASE statement. Essentially, every program would have to have its own little stop-reentering point so that it could have its DO CASE implemented when the stop occurs. This would be awkward.

The rationale for desiring to STOP and restart at pre-selected points during a checkout procedure is clear. However, the ability to restart at any statement seems to be unnecessary. It would be extremely difficult to verify that a program will perform satisfactorily if it is started at an arbitrary point. To implement the stop and restart anywhere in HAL is equivalent to a FORTRAN assigned GO TO using a control variable. The FORTRAN assigned GO TO has a label form in which you assign a name to an integer and then you go to this particular label or name. HAL has eliminated this particular form intentionally, as being unreliable due to the fact that it is calculated at run time. The address is not fixed in the compiler. It is recommended that this unrestricted form of the STOP statement be eliminated from GOAL for the same reason.

### 3.1.3 Repeat

The following discussion exposes the complexity of translating the Repeat statement, with all its implications, into HAL.

The GOAL language has a form of internal label called the "STEP NUMBER" which is used to identify individual executable statements within the GOAL test procedure or subroutine in question. In the translation of GOAL into HAL, these labels will have to be translated in a manner which is to a large extent dependent upon the global context of the procedure. In certain cases, a step number will translate directly into a simple statement label in the corresponding HAL program, provided that the original GOAL program references the step only from a GO TO context. In the more general case of a label referenced from GOAL's

iterative REPEAT statement, the use of global synthesis and reorganization of code will be required in order to handle all the possible cases. This latter case is intimately bound up with the problem of translating the GOAL statement, and forms the major topic of this section.

A step number is used in the following senses:

a) as a reference point for branching

b) as a delimiter of a block (internal) of code referenced from a REPEAT statement

In translating a GOAL program into the HAL/S language, the first step may be to analyze the GOAL program's step numbers for their ultimate usage as follows:

a) identify all labels which are the targets of GO TO statements within the procedure

b) identify the objects of all REPEAT statements, in terms of the starting and ending statement step numbers, static overlap of ranges, etc.

The first result of this analysis will be a list of step numbers with various possible combinations of references:

1. Unreferenced
2. Referenced in one or more GO TO's
3. Referenced as the start or end of REPEAT group
4. Referenced as a GO TO target and as the start or end of a REPEAT group.

The second result will be a list of REPEAT groups broken down into several classes depending upon their physical relation to all the other REPEAT groups in the GOAL test procedure or subroutine:

1. Independent (not nested within another REPEAT and not overlapping another REPEAT group).
2. Nested. This form of REPEAT group is entirely contained between the starting and ending labels of another repeat group.

3. Overlapping. This form of REPEAT group has either its opening or closing statement (but not both) contained within the scope of another REPEAT group. The common statements can be a single statement or any number of statements.

It should be noted that the physical nesting of GOAL's REPEAT groups has no effect upon the dynamic execution which occurs; it is merely a static blocking of statements known at compile time. In contrast, the dynamic relations which occur enter when a given REPEAT group contains a REPEAT statement referencing another REPEAT group. Thus, in GOAL the local structure of iterative forms has no relation to the physical layout of program text as it does in the HAL/S nested DO forms.

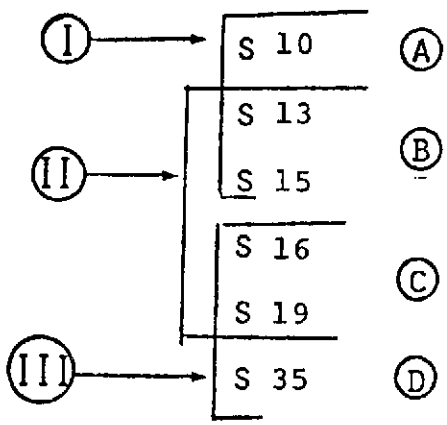
Generating the equivalent of a GOAL REPEAT group in the HAL/S language will prove to be fairly complicated. For instance, consider the case of overlapping ranges:

```
I    REPEAT S_10 TO S 15 FOR 3 TIMES;

II   REPEAT S 13 TO S_19;

III  REPEAT S 16 TO S 35 FOR 5 TIMES;
```

The three distinct overlapping REPEAT groups can be depicted as follows:




These three groups can be divided into 4 zones, with each zone being made a procedure with the name indicated below:

| ZONE | START | END | NAME |
|---|---|---|---|
| A | S 10 | S 12* | A_PROC |
| B | S 13 | S 15 | B_PROC |
| C | S 16 | S 19 | C_PROC |
| D | S 20 | S 35 | D_PROC |

The translation to HAL/S of the three GOAL statements then becomes:

```
I:    DO FOR J = 1 TO 3;
         CALL A_PROC;
         CALL B_PROC;
      END;
II.   CALL B_PROC;
      CALL C_PROC;
III.  DO FOR J = 1 TO  5 ;
         CALL C_PROC;
         CALL D_PROC;
      END;
```

This particular translation does not involve any GO TO actions from within REPEAT groups to points outside their range. This may be handled by further partitioning the code and using control variables as arguments of the translator-generated procedures.

The above discussion clearly demonstrates the complexity involved in treating overlapping REPEAT groups. REPEAT is obviously a necessary construct in GOAL especially when one realizes that there is no DO form. However, it is difficult to understand why the general overlap of REPEAT groups is required. Other modern programming higher order languages such as PL/1, HAL, FORTRAN, etc. do not allow the overlapping of program blocks. Programming limitations have not been discovered due to the lack of this feature.

It is therefore suggested that:

a) <u>REPEAT groups be limited to non-overlapping groups with the grouping performed by the programmer</u>.

b) Overlapping REPEAT groups should be flagged as an error condition by the translator compiler.

c) The generality of arbitrary REPEAT groups tends to complicate the software verification process.

### 3.1.4 Software Interrupts

The software interrupt capability, with the conditional GO TO and RETURN options, presents some unique problems for a GOAL-to-HAL translator. There are two basic GOAL constructs to consider. The GOAL statement,

```
WHEN INTERRUPT X OCCURS PERFORM SUBROUTINE(SUB);
```

can be handled by a HAL SCHEDULE statement

```
SIGNAL ACTIVE_X ON;
SCHEDULE SUB ON INTERRUPT_X & ACTIVE_X & INTERRUPTABLE
PRIORITY (PRIO+1);
```

INTERRUPT_X is a translator defined event corresponding to the GOAL defined interrupt function designator labeled X. ACTIVE_X is a translator defined event which is SIGNALLED ON by the WHEN INTERRUPT statement and SIGNALLED OFF by the DISABLE statement. INTERRUPTABLE is a global pulsed event which is signalled at the end of every GOAL translated statement. This guarantees that GOAL programs are only interrupted at the end of GOAL statements. It should be noted that the interpretive execution of GOAL statements would require essentially the same mechanism. PRIO is a HAL/S built-in function whose numeric value is the priority of the process which is currently running.

The modifier "AND RETURN TO STEP 900" can be applied to a WHEN INTERRUPT statement in GOAL. This form has no corresponding HAL construct. It can only be implemented by an additional element in the HAL/S language. This element would take the form

```
ON event-expression statement;
event expression::= INTERRUPT_X & ACTIVE_X & INTERRUPTABLE
statement::= DO; GO TO; SIGNAL INTERRUPT_X OFF; END;
             DO; CALL SUB; SIGNAL INTERRUPT_X OFF; END;
             DO; CALL SUB; GO TO; SIGNAL INTERRUPT_X OFF; El
```

It should be noted that the event variables INTERRUPT_X, ACTIVE_X, INTERRUPTABLE must be declared (by the translator) for the HAL/S compiler, at the beginning of the program.

For example, the GOAL statement

```
WHEN INTERRUPT X OCCURS PERFORM SUBROUTINE (SUB) AND
 RETURN TO S900;
```

translates into:

```
SIGNAL ACTIVE_X ON;
ON INTERRUPT_X & ACTIVE_X & INTERRUPTABLE
  DO;
    CALL SUB;
    SIGNAL INTERRUPT_X OFF;
    GO TO S_900;
  END;
```

The GOAL Statement

```
WHEN INTERRUPT X OCCURS GO TO STEP 900;
```

will translate into:

```
SIGNAL ACTIVE_X ON;
ON INTERRUPT_X & ACTIVE_X & INTERRUPTABLE
  DO;
  SIGNAL INTERRUPT_X OFF;
  GO TO S900;
  END;
```

If the RETURN TO and GO TO options of the WHEN INTERRUPT statement are disallowed then the new HAL/S construct would be unnecessary. The form was considered in the specification of the original HAL language, but was rejected because of its potential misuse and its inherent contribution to software unreliability.

### 3.1.5 Non-GOAL

Non-GOAL subroutines may be incorporated into a Data Bank by means of the LEAVE and RESUME statements. These statements are system directives and as such do not produce any executable HAL code. The Non-GOAL subroutine which is bracketed by LEAVE and RESUME are combined with other GOAL subroutines and programs during the link edit process.

### 3.1.6 Data Bank

The "Data Bank" is a concept introduced into the GOAL language for a number of reasons.

a) The concept will isolate the test programmer from the details of manipulating the test equipment.

b) It allows the test programmer to access test points in a concise English format which is more meaningful to the logic of the test sequence.

In a sense, the Data Bank provides a cross reference between the test programmer's English-like statements and the parameters required by the I/O routines to access a particular test point.

The "Data Bank" concept involves more than a simple language construct which is ultimately translated into machine code. The creation, control and use of the Data Bank imposes requirements upon the run time operating system as well as upon the translator (or compiler). In addition, there is an implicit data management system involved in the generation of the Data Bank.

Figure 3-1 shows the basic elements involved in the Data Bank system. A discussion of these elements is important in order to ascertain the impact upon the translator.

3.1.6.1 Data Bank Creation. Fundamentally, management and engineering must indicate to the test programmer all the system test points and sequences, what they measure, and how to invoke them by means of a GOAL function designator. A formal documentation system for the test engineers must be developed and maintained. In addition a computerized data management system must be implemented to control the creation of the Data Bank and provide revision control. The Data Bank is compiled independent of a GOAL program. It provides a common pool of information for use by programs. It can be modified, via revision control and specified statements.

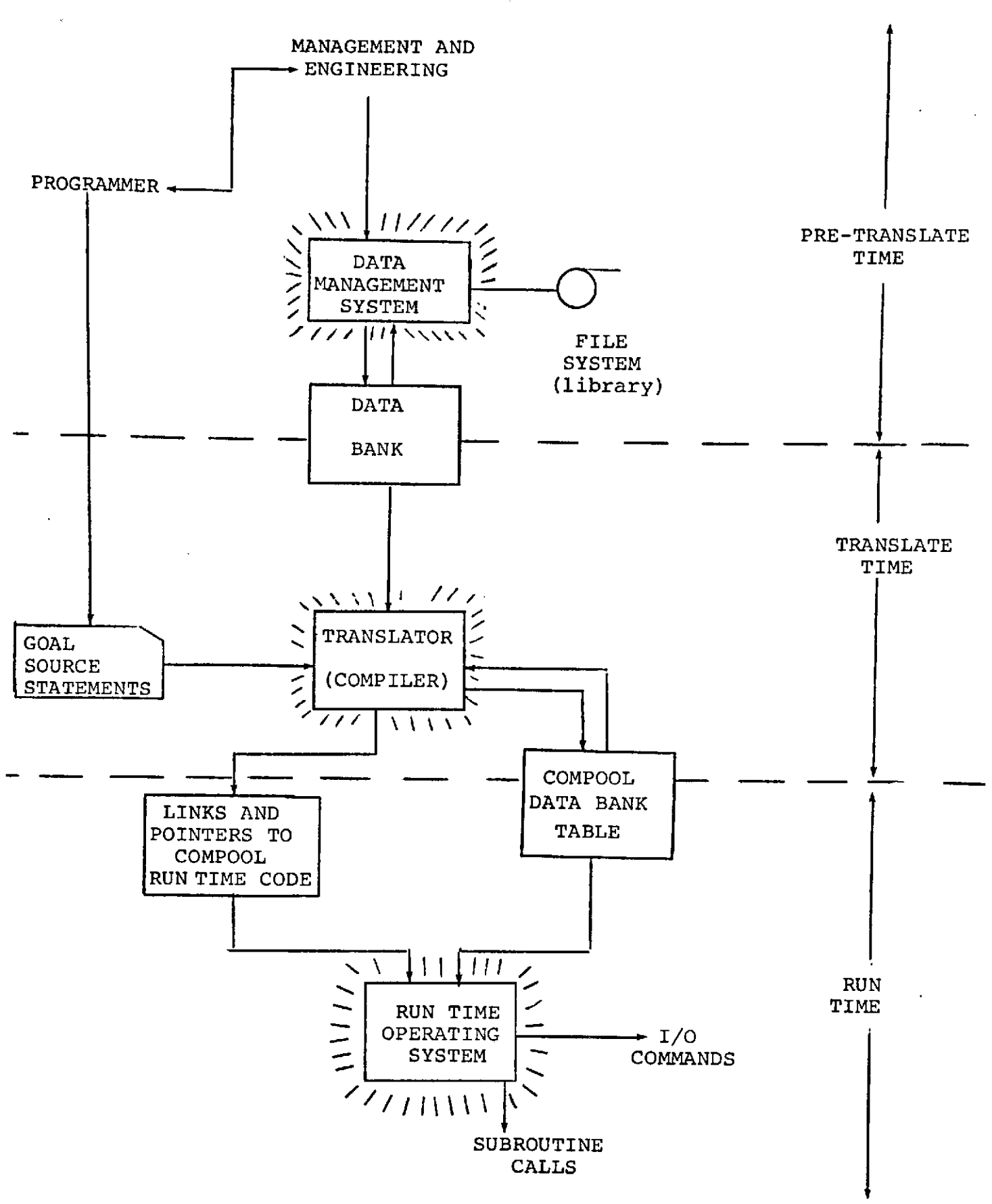



Figure 3-1

Elements of Data Bank System

3.1.6.2 Data Bank Control. Even though Data Banks have been created, each Data Bank must be specifically controlled by use of the GOAL USE and FREE statements. These statements are directives to the translator. They do not produce any code. They are provided to allow controlled access rights to the various Data Banks and enable a translate time error detection capability if unauthorized access to a Data Bank is attempted. The interface between the translator and Data Bank provides a control table for this purpose.

3.1.6.3 Translator Inputs and Outputs. The translator's role in terms of Data Bank creation, control, and use can be determined by investigating the various inputs and outputs to the translator.

There are two input sources to the translator. These are GOAL source statements and the Data Bank. The GOAL statements and primitives involved with the Data Bank include:

a) USE, FREE and SPECIFY Data Bank

b) BEGIN and END Data Banks

c) Function Designators

d) Revision Labels

The Data Bank is a creation of the data management system. The data management system contains all the possible test point access information. This information is placed into the data bank by means of the SPECIFY statement. A particular program may eliminate entries in the Data Bank by means of the FREE statement and may direct the compiler to include a given Data Bank by means of a USE statement. Basically, all the GOAL system constructs involve communication between GOAL source code and the Data Bank. These statements would not be involved in the GOAL-to-HAL translator. The syntax analyzer (the front end of the GOAL compiler) would be the same, independent of whether a GOAL-to-HAL translator was used or not. The impact of the Data Bank concept upon the GOAL system is very large. However, the impact upon a GOAL-to-HAL translator can be made small (depending of course, on what is included in a translator).

The GOAL SPECIFY statement causes the translator to place in a HAL COMPOOL block all the information required by the I/O routines in the operating system to access the

given test point. In addition, any specialized routines needed to access a given test point must be specified. The GOAL function designator causes the appropriate run time code along with the necessary linkages and COMPOOL pointers to be generated. The details of this process are very dependent upon the run time operating system.

If one considers function designators to be external test points which are accessed by means of a channel number then each specific function designator might be replaced by a HAL FILE statement containing two literals:

$C_i$, which indicates the $i^{th}$ channel

$A_i$, which indicates the $i^{th}$ address

For example, the function designator <TEST POINT 5> might be accessed by channel $C_i$ and address $A_i$. The equivalent HAL form would then be:

FILE($C_i$, $A_i$)

The Data Bank actually supplies the value of $C_i$ and $A_i$. These were previously generated by the Specify statement*.

A possibility exists that certain test points are not directly addressable and require a complex set of actions to access them. These actions can be very device-dependent. For these situations, specialized I/O routines must be written to service these devices. If the Data Bank indicates that a specialized subroutine is required then the HAL translation becomes a CALL statement to that routine.

Consider the following example of the specification and utilization of a Data Bank element.

---

* This method is an illustration of an equivalent HAL/S I/O form. Final decisions will be based on the I/O available in HAL/S under the FCOS.

```
SPECIFY <TEST POINT 5> AS SENSOR USING SUBROUTINE
(CONVERT) *CH3, ADDRESS12;
```

The data management system extracts from the SPECIFY statement all the pertinent information and places it in the Data Bank. When function designator <TEST POINT 5> is encountered by the translator, Data Bank information is accessed through a control table as illustrated in Figure 3-2. The following HAL code is produced.

```
X = FILE (CH3, ADDRESS12)

CALL CONVERT(X) ASSIGN TEST_POINT 5;
```

The HAL compiler and the operating system will use CH3, ADDRESS12 to provide the details of the I/O.

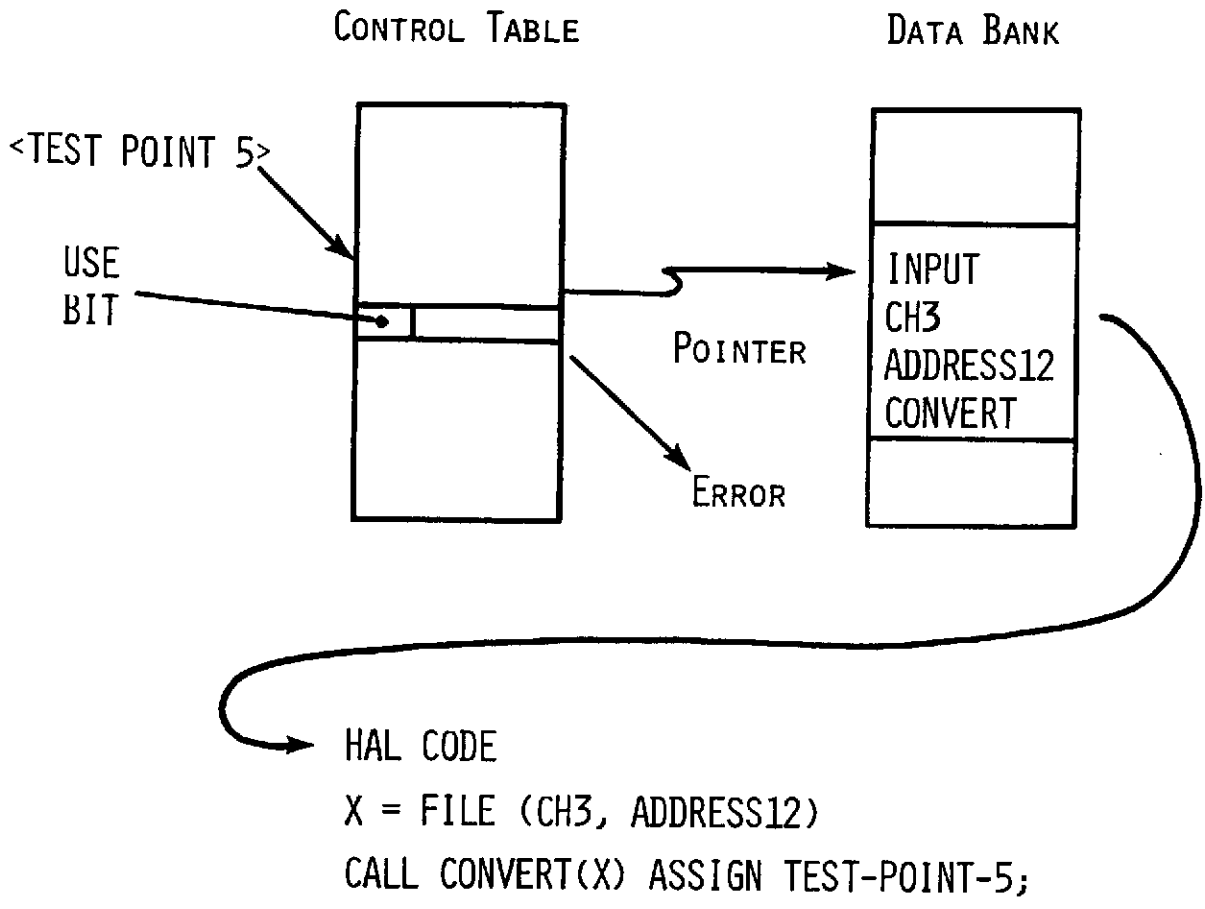



Figure 3-2

Function Designator Access of Data Bank

## 3.2 The GOAL Master Program Concept

### 3.2.1 Overall Organization

There are several translation and mapping problems which can be handled together by a single unified strategy - the concept of a "GOAL Master Program" written in HAL, produced by the translator program, and responsible for coordinating all the HAL/S blocks produced by the translation process. The problems which lead to use of this strategy are several:

a) GOAL has a "TERMINATE SYSTEM" statement which is supposed to cause "complete GOAL application program system shutdown". Thus, a means must be provided to make its effects global to all translated GOAL modules at execution time.

b) GOAL allows a "CONCURRENTLY PERFORM PROGRAM" statement with no restrictions on the number of such statements referencing a single PROGRAM and how many such concurrent uses exist simultaneously off a single program module.

c) The set of translated GOAL programs have their own internal system of software interrupts and other signals which, for reliability, should be kept self-contained to prevent unwanted interaction with the other HAL/S applications software.

In order to treat these problems properly, a GOAL Master Program should be provided, with the following characteristics:

a) When the GOAL system is to be initiated in the flight computer, it is the GOAL Master Program which is scheduled and executed.

b) When the GOAL system is to be shutdown by a TERMINATE SYSTEM" action (as translated from the GOAL source into HAL/S), the effect is achieved by a HAL/S TERMINATE with the name "GOAL_MASTER_PROGRAM" as the object of the action (i.e. the name of the highest level process in the GOAL section of the software).

c) Each GOAL program which is to be part of the running GOAL system will become a single HAL/S

Procedure within the GOAL Master Program, nested at the program level. If the original GOAL Program is subject to multiple concurrent perform statements, then real time attributes of REENTRANT or EXCLUSIVE will have to be applied in order to assure no conflicts.

d) Each PERFORM Program statement, whether with a CONCURRENT statement or not, will generate a uniquely named HAL/S task block nested at the program level. The executable action "PERFORM PROGRAM" will consist of a SCHEDULE for the task so generated, optionally followed by a WAIT for the end of the task if it is not a concurrent PERFORM. This task block is a dummy whose sole purpose is to execute a single statement: A CALL to the appropriate GOAL PROGRAM as translated into a HAL/S procedure.

e) At the GOAL_MASTER_PROGRAM level, event variables can be maintained as software interrupts available to all the GOAL system's blocks, for use as described in Section 3.1.4.

Within this strategy, the GOAL-to-HAL/S translator performs a range of functions:

a) It must act as a source level link editor, combining all the GOAL programs required for a given system into a common framework for submission to the HAL/S compiler.

b) It must resolve all references to various GOAL source modules and incorporate them in the output as a self-consistent and complete HAL/S package.

c) It must translate each individual GOAL component as a self-contained entity within the framework of the GOAL_MASTER_PROGRAM block to be created.

In order to support such a GOAL_MASTER_PROGRAM yet still allow modular compilation of the GOAL programs, two variations of compiler operation are required:

a) A single-program translation in which emphasis is placed on analyzing a single component of the GOAL system for syntax errors. This type of compilation is primarily for preliminary testing.

b) A complete system compilation in which all programs referenced in the GOAL test system are gathered into appropriate libraries translated as Procedures and generated Tasks, and merged into a single HAL/S Program - The GOAL_MASTER_PROGRAM and all its subsidiary tasks and procedures as described above.

Figure 3-3 shows the general conception of translator operation required to produce the GOAL_MONITOR_PROGRAM. The figure illustrates that a whole system of GOAL programs must ultimately be combined via the translator into a single integrated HAL/S program.

Figure 3-4 shows the "output" of the translator (the GOAL_MASTER_PROGRAM) and identifies several of its components and their derivation from the GOAL inputs. In the example which follows, the GOAL Master Program is explored in some more detail.

### 3.2.2 GOAL Master Program Example

The preceding has described the general outline of a HAL/S organization which will adequately handle the GOAL-HAL translation problems of concurrency and of certain systems statements. In the discussion which follows, the mapping is set forth in a specific example.

Suppose that the GOAL test is to consist of six GOAL programs called TEST_A, TEST_B, TEST_C, TEST_D, TEST_E, and TEST_F, and that the following relations hold:

1. TEST_A and TEST_B are to be initiated by system start-up procedures.

2. TEST_A PERFORMS TEST_C AND TEST_D, and concurrently performs TEST_E from two different locations.

3. TEST_B CONCURRENTLY PERFORMS TEST_C, TEST_E, and TEST_F.

4. TEST_D CONCURRENTLY PERFORMS TEST_F and calls TEST_E.

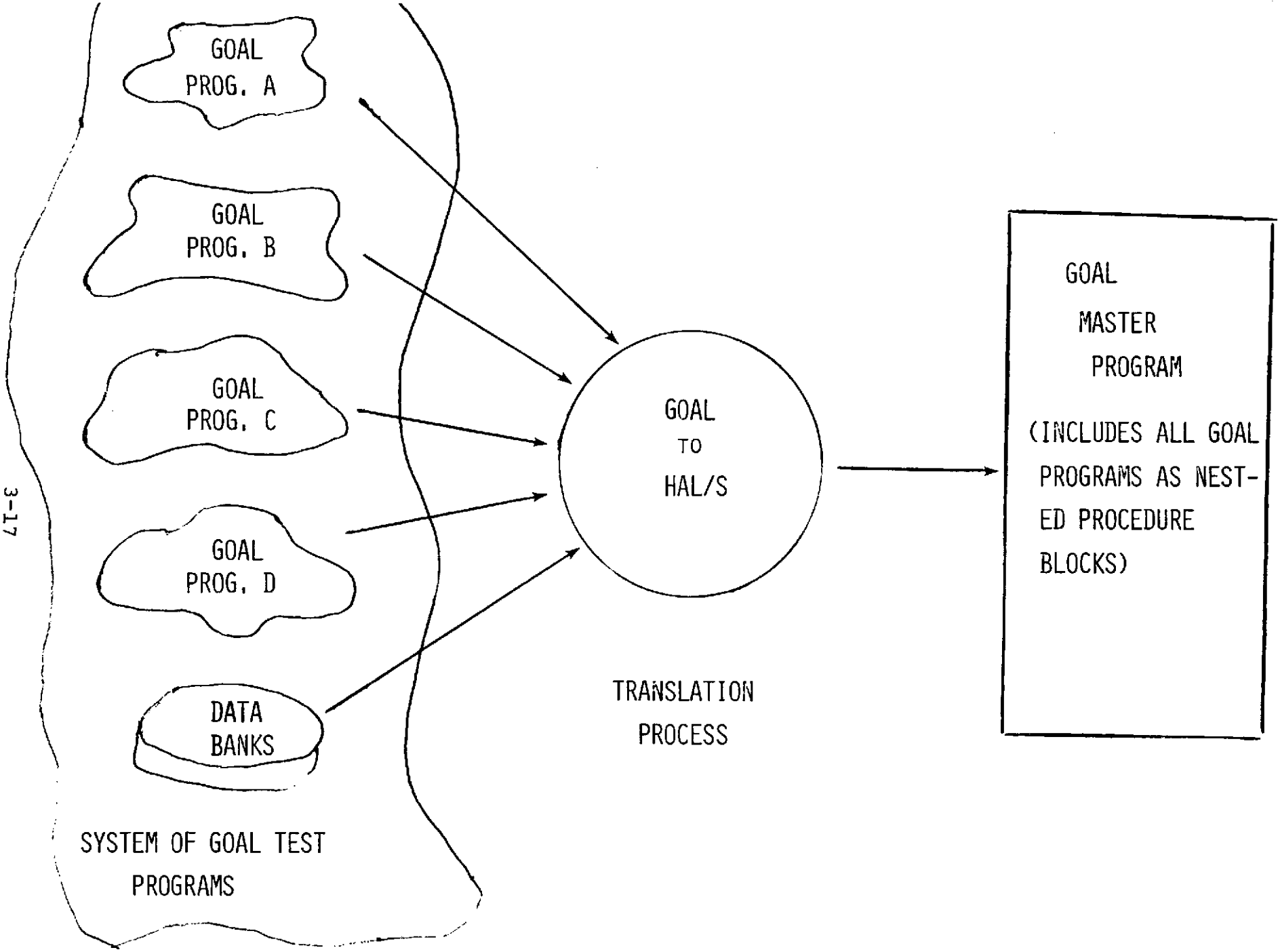
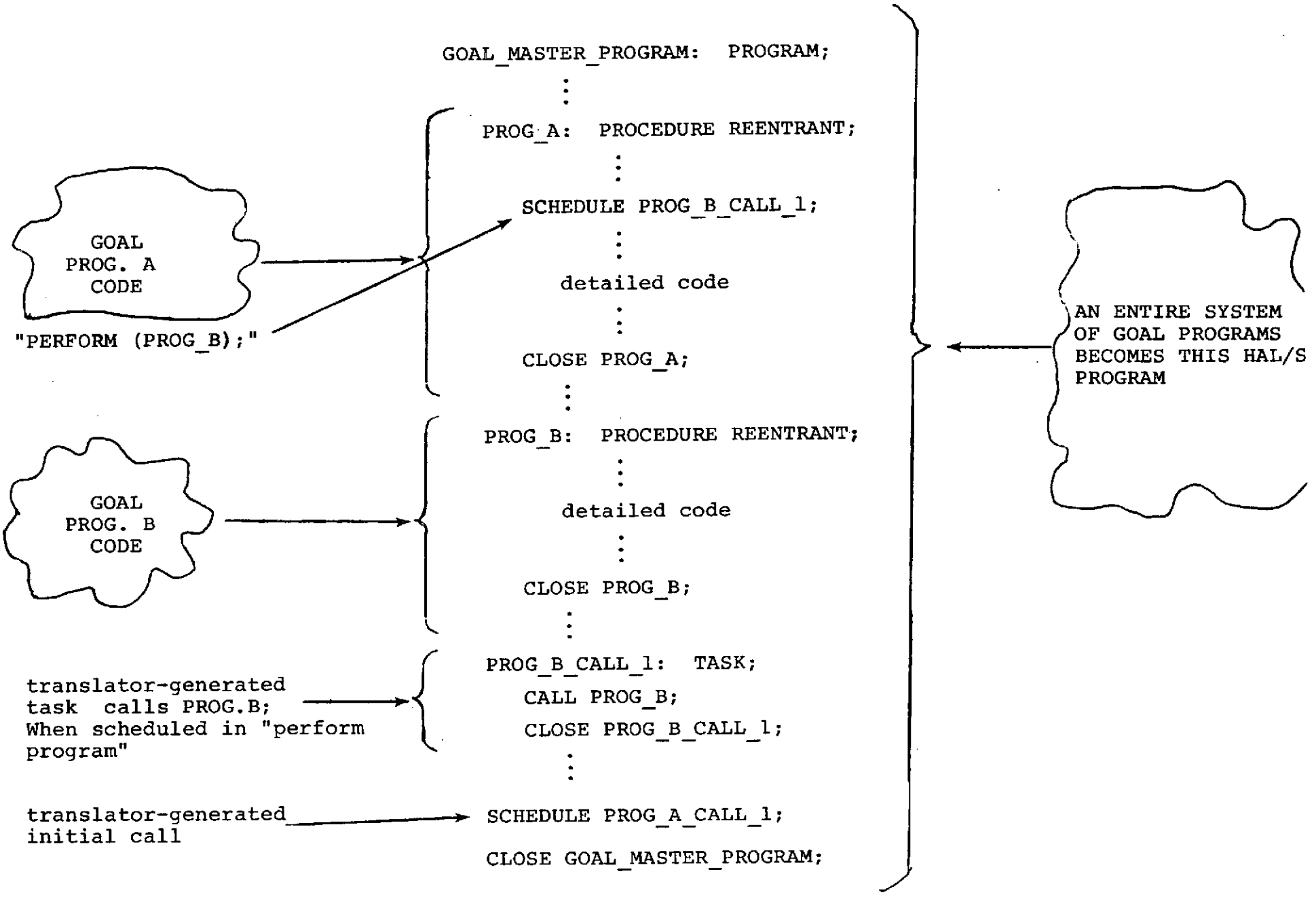

GOAL
PROG. A
GOAL
PROG. B
GOAL
PROG. C
GOAL
PROG. D
DATA
BANKS
SYSTEM OF GOAL TEST
PROGRAMS
INPUT
GOAL
TO
HAL/S
TRANSLATION
PROCESS
GOAL
MASTER
PROGRAM
(INCLUDES ALL GOAL
PROGRAMS AS NEST-
ED PROCEDURE
BLOCKS)
HAL/S OUTPUT

Figure 3-3

Figure 3-4


GOAL_MASTER_PROGRAM: PROGRAM;
PROG_A: PROCEDURE REENTRANT;
SCHEDULE PROG_B_CALL_1;
detailed code
CLOSE PROG_A;
PROG_B: PROCEDURE REENTRANT;
detailed code
CLOSE PROG_B;
PROG_B_CALL_1: TASK;
CALL PROG_B;
CLOSE PROG_B_CALL_1;
SCHEDULE PROG_A_CALL_1;
CLOSE GOAL_MASTER_PROGRAM;
GOAL PROG. A CODE
"PERFORM (PROG_B);"
GOAL PROG. B CODE
translator-generated task calls PROG.B; When scheduled in "perform program"
translator-generated initial call
AN ENTIRE SYSTEM OF GOAL PROGRAMS BECOMES THIS HAL/S PROGRAM

This system was imagined in an ad-hoc fashion to illustrate the process of organizing the GOAL_MONITOR_PROGRAM; it does not necessarily correlate to any real system. The process of analyzing begins by tabulating the relationships:

| Program | Scheduled By |
|---|---|
| TEST_A | System |
| TEST_B | System |
| TEST_C | TEST_A, TEST_B |
| TEST_D | TEST_A, |
| TEST_E | TEST_B, TEST_D, (2) TEST_A |
| TEST_F | TEST_B, TEST_D |

Since each reference to a GOAL program with or without arguments causes a separate task block to be generated, there will be 11 task blocks generated in this example, labeled:

TEST_A_SYSTEM,

TEST_B_SYSTEM,

TEST_C_A, TEST_C_B

TEST_D_A

TEST_E_A, TEST_E_A2,

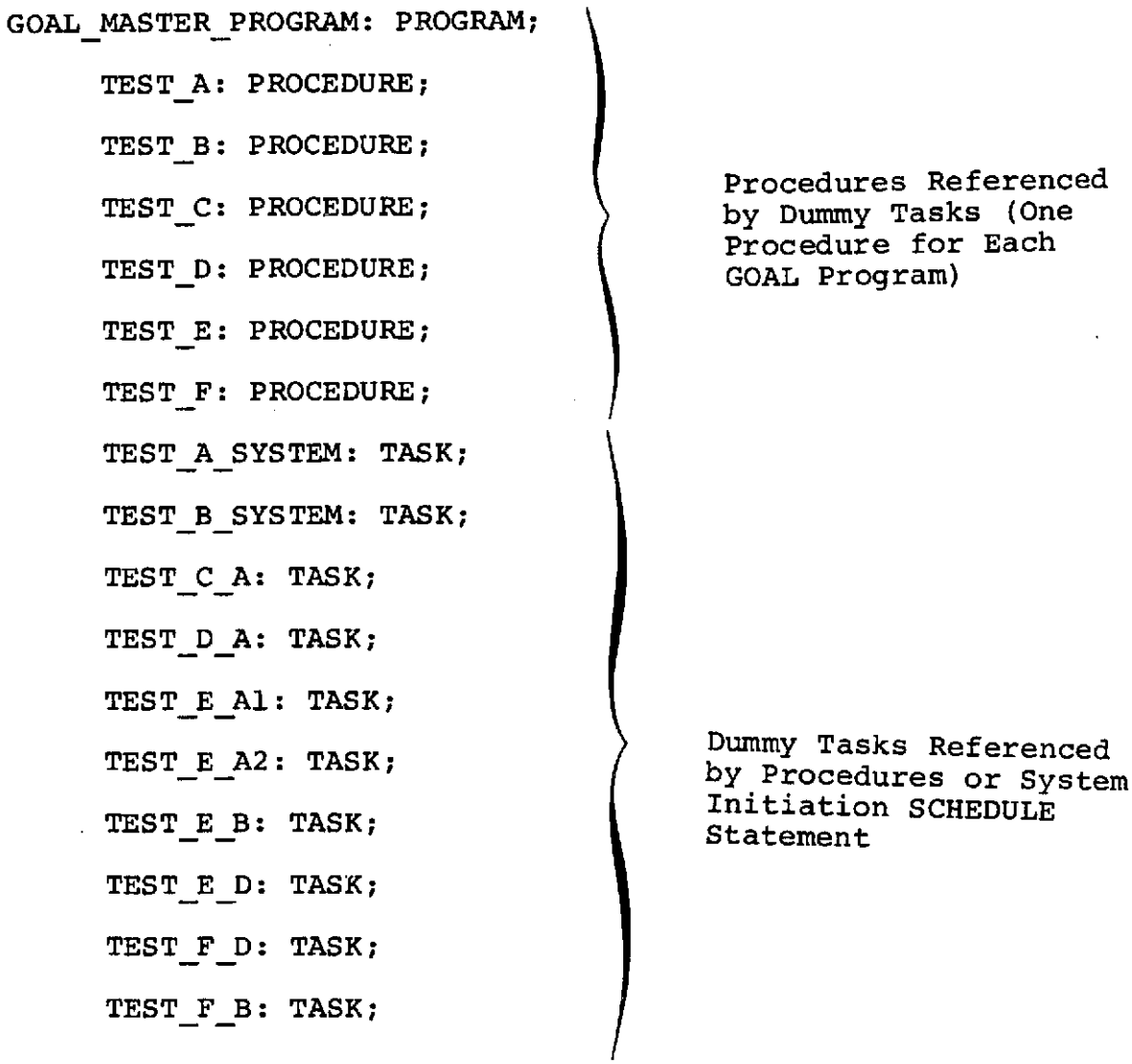
TEST_E_N

TEST_E_D,

TEST_F_D, TEST_F_B

The suffixes here are mechanically derived from the label of the referencing block; thus when Block D references Block E, the task name is TEST_E_D.

The HAL/S program layout in this example can be diagrammed as follows:


GOAL_MASTER_PROGRAM: PROGRAM;
TEST_A: PROCEDURE;
TEST_B: PROCEDURE;
TEST_C: PROCEDURE;
TEST_D: PROCEDURE;
TEST_E: PROCEDURE;
TEST_F: PROCEDURE;
TEST_A_SYSTEM: TASK;
TEST_B_SYSTEM: TASK;
TEST_C_A: TASK;
TEST_D_A: TASK;
TEST_E_A1: TASK;
TEST_E_A2: TASK;
TEST_E_B: TASK;
TEST_E_D: TASK;
TEST_F_D: TASK;
TEST_F_B: TASK;
Procedures Referenced by Dummy Tasks (One Procedure for Each GOAL Program)
Dummy Tasks Referenced by Procedures or System Initiation SCHEDULE Statement

The program level code in this example would consist of statements needed to concurrently initiate the two blocks TEST_A and TEST_B when GOAL_MASTER_PROGRAM is initiated. This code is simply:

```
SCHEDULE TEST_A_SYSTEM;

SCHEDULE TEST_B_SYSTEM;
```

When executed, these two statements cause the two tasks mentioned to be immediately entered into the FCOS queues as active tasks, (i.e. subject only to priority restrictions on running). The task block for TEST_A_SYSTEM would have the following form, which is completely typical of all 11 Task blocks in the system:

```
TEST_A_SYSTEM: TASK;

   CALL TEST_A;

   CLOSE TEST_A_SYSTEM;
```

When TEST_A (as initialized) enters the HAL/S equivalent of the GOAL statement:

```
"PERFORM TEST_C"
```

the following HAL/S code will be used:

```
SCHEDULE TEST_C_A;

WAIT FOR ¬TEST_C_A;
```

Because the "non-concurrent" PERFORM is equivalent to the CALL operation, sequential execution is assumed; thus a WAIT for completion of the task is required, using the "task event" TEST_C_A which is true so long as TEST_C_A is in the FCOS real time queues. This example of a simple PERFORM translation assumes that the HAL/S statement

```
DECLARE TEST_C_A TASK EVENT;
```

appears in the declare group of TEST_A, the "calling program".

As execution continues, the HAL/S equivalent of "CONCURRENTLY PERFORM PROGRAM TEST_E" is encountered from within TEST_B. Here, the HAL/S equivalent is simply a

schedule statement without the wait required to simulate a CALL linkage:

```
SCHEDULE TEST_E_B;
```

The effect of the HAL/S statement here is to enter TEST_E_B as a new process contending for resources on the machine.

Later on, during execution of TEST_F, it may become necessary to terminate the whole GOAL system with the "TERMINATE SYSTEM" action. With the organization outlined, this can be achieved by the HAL/S statement:

```
TERMINATE GOAL_MASTER_PROGRAM;
```

When this statement executes, since all the nested code of the various tasks is dependent upon the program level, the entire system is ended by ending the global program.

This demonstrates an overall strategy of translation which will enable the translator to implement HAL/S equivalents of GOAL TERMINATE and concurrently PERFORM PROGRAM statements, as applied to the specific case illustrated.

# 4.0 GOAL-TO-HAL MAPPING

In this section, the proposed relationships between the GOAL and HAL statements are described in some detail. This description has been designated "mapping" in order to distinguish it from the ultimate objective which is a translator specification. Mapping will give an explanation or example of each complete GOAL statement. A Specification would rigorously define all variations, permutations, and combinations of each GOAL statement and is beyond the scope of this study.

## 4.1 Declaration Statements

### 4.1.1 Single Data Type

GOAL Statement:

```
DECLARE NUMBER(RESULTS);
```

Purpose:

This statement declares a numeric data item with the symbolic name (RESULTS) for purposes of general computation within the GOAL program.

Equivalent HAL/S form:

DECLARE RESULTS; This statement will declare an unarrayed single precision scalar variable which can be used in the same context as the corresponding GOAL form.

---

GOAL Statement:

```
DECLARE QUANTITY (OFFSET) = .5 PSI, (PUMP PRESS);
```

Purpose:

This statement declares a GOAL "quantity" (a real number and an associated dimension) as a variable for use in a program. Since the units associated with these entities exist for annotation purposes only, the equivalent HAL/S declarations include a scalar declaration and a character string dimension.

HAL/S Equivalent Form:

```
DECLARE OFFSET INITIAL(0.5), DIM_OFFSET CHARACTER(6)
INITIAL('PSI'), PUMP_PRESS, DIM_PUMP_PRESS CHARACTER(6);
```

Note that the initialization value of 0.5 was provided in the GOAL form as a compile time assignment (the equal-sign) which becomes the initial value 0.5 in the HAL/S form. The GOAL declaration becomes an equivalent multiple HAL/S form. The 6-character "DIM" fields of each quantity serves to hold its current unity value during execution.

---

GOAL Statement:

```
DECLARE STATE (FLAG A) = ON, (FLAG B);
```

Purpose:

This statement is used in the context of a GOAL program to declare the existence of the single bit booleans FLAG A and FLAG B. The initial value of the variable FLAG A is set to be ON. FLAG B is not initialized.

Equivelant HAL/S form:

```
DECLARE BOOLEAN, FLAG_A INITIAL(ON), FLAG_B;
```

---

GOAL Statement:

```
DECLARE TEXT(ERROR MESSAGE) = (6D10 BATTERY VOLTAGE LOW);
```

Purpose:

This statement is used in a GOAL program to declare a fixed character message which will be used in some I/O operation. Since there is no character manipulation or assignment in GOAL, this message must always be either fixed or defined in some input operation.

Equivalent HAL/S statement:

```
DECLARE ERROR_MESSAGE CHARACTER(24) INITIAL('6D10 BATTERY
                                    VOLTAGE LOW');
```

### 4.1.2 List Type

GOAL Statement:

```
DECLARE NUMERIC LIST(LIST NUM) WITH 4 ENTRIES;
```

Purpose:

This statement declares to the GOAL compiler that the programmer wishes to create a linear array of 4 numeric elements without any initialization.

Equivalent HAL/S form:

```
DECLARE LIST_NUM ARRAY(4);
```

---

GOAL Statement:

```
DECLARE NUMERIC LIST (ROOT 3) WITH 10 ENTRIES 1.000,
 1.260,1.442,1.587,1,710,1.817,1,903,2.000,2.080,2.154;
```

Purpose:

This statement declares to the GOAL compiler that the programmer wishes to create a linear array of 10 numeric elements with the indicated initialization.

Equivalent HAL/S form:

```
DECLARE ROOT_3 ARRAY(10) INITIAL(1.00,1.260,1.442,
1.587,1.710,1.817,1.903,2.000,2.080,2.154);
```

---

GOAL Statement:

```
DECLARE QUANTITY LIST (LIST A) WITH 3 ENTRIES;
```

Purpose:

This statement creates a list of 3 GOAL quantities in a linear array form. Each quantity has a scalar value and a physical units dimension.

HAL/S Equivalent Form:

```
DECLARE ARRAY(3) LIST_A, DIM_LIST_A CHARACTER(6);
```

---

GOAL Statement:

```
DECLARE QUANTITY LIST (VOLTAGE LIST) WITH 6 ENTRIES
    28V,+0.5V,-0.5V,0V,50V,10 SECS;
```

Purpose:

This statement sets up a list of 6 GOAL quantities, with initialization to the values indicated. A more consistent practice might be to keep the identification of units out of the language - or introduce syntax and semantic checking of the units of scalars to flag such combinations as are illegal by dimensional analysis.

HAL/S Equivalent Form:

```
DECLARE ARRAY(6) VOLTAGE_LIST,INITIAL(28,.5,-.5,0,
   50,10) DIM VOLTAGE_LIST CHARACTER(6)
   INITIAL('V','V','V','V','V','SECS');
```

Note, that again the units are present in the HAL/S character string form.

---

GOAL Statement:

```
DECLARE STATE LIST (FLAG LIST) WITH 10 ENTRIES;
```

Purpose:

This statement sets up an array of 10 booleans for reference within the GOAL program. No initialization is performed.

HAL/S Equivalents:

In HAL/S there are two ways such an array might be handled; each with its own relative merits.

a) DECLARE FLAG_LIST ARRAY(10) BOOLEAN; This version corresponds directly with the GOAL form, since it is a simple array of one bit values.

b) DECLARE FLAG_LIST BIT(10); In this form, the use of a single bit string is designed to do the same thing. This form has the advantage of efficient storage but it is much more costly in terms of time overhead, and it cannot handle arrays longer than the maximum length of a HAL/S bit string. It will be assumed that the array form is to be used.

---

GOAL Statement:

```
DECLARE STATE LIST (LIST STATE) WITH 6 ENTRIES
   ON,ON,ON,OFF,OFF,ON;
```

Purpose:

This statement declares an array of 6 binary values (Booleans) for use within the GOAL program to store the states of discretes.

Equivalent HAL/S form:

```
DECLARE LIST_STATE ARRAY(6)BOOLEAN INITIAL(ON,ON,ON,OFF,OFF,ON);
```

---

GOAL Statement:

```
DECLARE TEXT LIST(INPUT) WITH 2 ENTRIES WITH A
   MAXIMUM OF 25 CHARACTERS;
```

Purpose:

This statement sets up an array of two text strings for use as the receiver of some input followed by later use as the source of some output (no internal manipulations of text are provided by GOAL).

Equivalent HAL/S form:

```
DECLARE INPUTZ ARRAY(2) CHARACTER(25);
```

Note that in this example the original name duplicated a HAL/S keyword and thus had to be modified in some way following the translation. In the example, the letter Z was appended to the original name thereby resolving the keyword conflict. The maximum length of 25 carries over directly.

---

GOAL Statement:

```
DECLARE TEXT LIST (OPERATOR INSTRUCTION) WITH 2 ENTRIES
   (PLACE SWITCHES INDICATED),(*PREFLIGHT TM CAL IN
   PROGRESS*);
```

Purpose:

This statement declares an array of two character strings which are initialized to the values indicated, with a maximum length determined by implication from the length of the initial values.

Equivalent HAL/S form:

```
DECLARE OPERATOR_INSTRUCTION ARRAY(2) CHARACTER(30)
  INITIAL ('PLACE SWITCHES INDICATED','*PREFLIGHT
  TM CAL IN PROGRESS*');
```

This is a direct translation, with the only subtlety being the determination of the length maximum of all the initial values so that the character string maximum length may be found used in the declaration.

### 4.1.3 Table Types

The translation of GOAL table data types requires a strategy employing several HAL/S arrays to accomplish the same ends within the framework of a HAL/S program. A set of arrays which will accomplish this is the following:

1. A main data array with row and column dimensions identical to the row and column dimensions of the original GOAL table. The HAL/S data type of this table will be SCALAR, BOOLEANS, or CHARACTER depending upon the original GOAL table's data type (QUANTITY & NUMERIC, STATE, or TEXT respectively). This array will store data in the HAL/S version. For GOAL QUANTITY tables, an auxiliary character array of dimensions is required.

2. An auxiliary "activation array" of BOOLEAN elements controlling whether or not the given row is to be active at some time during execution.

These two arrays cover all the variable information about a GOAL table as translated into HAL/S. During translation only, two other sets of data are needed for each table:

1. A list of pointers to various function designator processing routines.

2. A list of the names of the various data array columns.

In both cases, there is not necessarily any need to code a HAL/S array, although such an array might prove convenient in the translated code. In the examples that follow, only the required arrays are shown in the HAL/S equivalent form.

---

GOAL Statement:

```
DECLARE NUMERIC TABLE(HIGH LOW RUN) WITH 3 ROWS
     AND 4 COLUMNS TITLED

          (HIGH), (LOW), (RUN), (CUR) WITH ENTRIES

<E1 GG CHAMBER P> ,  1000.1,   1.0,   500.0,    ,
<E2 GG CHAMBER P> ,  1001.2,    .9,   500.0,    ,
<E3 GG CHAMBER P> ,   999.8,   1.2,   500.0,    ;
```

Purpose:

This statement sets up a GOAL table with initial values in 3 columns, and 3 rows of function-designators. A fourth column is left uninitialized.

Equivalent HAL/S form:

Main Data Array:

```
DECLARE A_HIGH_LOW_RUN ARRAY(3,4) INITIAL(
   1001.1,   1.0,   500.0,   0,
   1001.2,   0.9,   500.0,   0,
    999.8,   1.2,   500.0,   0);
```

Note here that the HAL/S initialization cannot embed uninitialized values within its list so a "0" has been used in the fourth-column entries.

Activation Array:

```
DECLARE A_HIGH_LOW_RUN ARRAY(3) BOOLEAN
   INITIAL (TRUE, TRUE, TRUE);
```

Here the value "true" has been assumed to indicate that the row in question has an active function designator - initialized active as in the GOAL specification.

In addition to these two explicit tables, the GOAL-to-HAL/S translator program keeps track in this case of:

1. The function designator routine addresses for <E1 GG CHAMBER P>, <E2 GG CHAMBER P>, and <E3 GG CHAMBER P>.

2. The texts "HIGH", "LOW", "RUN" and "CUR" associated with columns 1 to 4 of the table, respectively.

---

GOAL Statement:

```
DECLARE QUANTITY TABLE (MAIN FUEL FLOW) WITH 5 ROWS
   AND 3 COLUMNS WITH ENTRIES

<E1 MAIN FUEL>, 0.1 PPS, 300.1 PPS,   ,
<E2 MAIN FUEL>, 0.3 PPS, 300.2 PPS,   ,
<E3 MAIN FUEL>, 0.4 PPS, 300.1 PPS,   ,
<E4 MAIN FUEL>, 0.2 PPS, 300.1 PPS,   ,
<E5 MAIN FUEL>, 0.1 PPS, 299.8 PPS,   ;
```

Purpose:

This statement sets up a GOAL quantity table with 5 function designators and 3 columns. Since this is a quantity table, and since quantity units can change as data, the HAL/S equivalent will have two main data arrays.

Equivalent HAL/S forms:

Main Data Arrays:

```
DECLARE MAN_FUEL_FLOW  ARRAY(5,3) INITIAL (
   0.1,  300.1,  0,
   0.3,  300.2,  0,
   0.4,  300.1,  0,
   0.2,  300.1,  0,
   0.1,  299.8,  0);

DECLARE MAIN_FUEL_FLOW_UNITS ARRAY(5,3) CHARACTER(6)
   INITIAL (15#'PPS');
```

Activation Array:

```
DECLARE A_MAIN_FUEL_FLOW ARRAY(5) BOOLEAN INITIAL
   (5#TRUE);
```

The main data array is again initialized in HAL/S with zeros replacing uninitialized embedded values. The units array for this quantity table is initialized with 'PPS' in all 15 positions; since the uninitialized positions of the original GOAL table can have arbitrary values, this in particular may be used as a value. The activation array is initialized "true" in all 5 positions.

In addition to these explicit arrays, the translator must (of course) keep track of the function designator routine addresses so that proper code will be generated.

---

GOAL Statements:

```
DECLARE STATE TABLE (THRUST OK) WITH 5 ROWS AND
  3 COLUMNS TITLED (THRUST OK), (THRUST NOT OK),
  (STATE) WITH ENTRIES
  <THRUST OK 1E1> , ON, OFF,  ,
  <THRUST OK 1E2> , ON, OFF,  ,
```

```
<THRUST OK 1E3> , ON, OFF,  ,
<THRUST OK 1E4> , ON, OFF,  ,
<THRUST OK 1E5> , ON, OFF,  ,
```

Purpose:

This statement sets up a GOAL "state table" with 3 columns and 5 rows, initialized as shown.

Equivalent HAL/S form:

Main Data Array:

```
DECLARE THRUST_OK ARRAY(5,3) BOOLEAN INITIAL (
   5#(TRUE, FALSE, FALSE));
```

Here, the initialization of the third column has been specified as "false" in each case in order to pick an arbitrary value for a field which was left uninitialized in GOAL. The values of each row in the table are identical, so tha repetition factor ("5#") is used rather than writing out the values.

Activation Array:

```
DECLARE A_THRUST_OK ARRAY(5)
   BOOLEAN INITIAL (5# TRUE);
```

As in previous example, this activation array masks each function designator as "on" (true) at the start of processing.

The translator in this case will keep track of the column names "THRUST_OK", "THRUST_NOT_OK", and "STATE"; it will also supply the code required to reference the five function designators.

---

GOAL Statement:

```
DECLARE TEXT TABLE (MESSAGE TABLE) WITH 2 ROWS
   AND 1 COLUMN TITLED
                  (MESSAGE A) WITH ENTRIES
<224 DISPLAY B35> , (SWITCH SCAN IN PROGRESS),
<224 DISPLAY B42> , (PLACE ABOVE SWITCHES AS INDICATED);
```

Purpose:

This GOAL statement prepares a table of character string data associated with two function designators. The table has but a single column.

Equivalent HAL/S form:

Main Data Array:

```
DECLARE MESSAGE_TABLE ARRAY(2) CHARACTER(33) INITIAL(
   'SWITCH SCAN IN PROGRESS',
   'PLACE ABOVE SWITCHES AS INDICATED');
```

Activation Array:

```
DECLARE A_MESSAGE_TABLE ARRAY(2) BOOLEAN
   INITIAL (2# TRUE);
```

Note that in this case, a maximum of 50 characters is allowed in the main data array elements.

## 4.2 Procedural Statements

### 4.2.1 Prefixes

GOAL Step Number Prefix:

```
STEP 163 ... rest of GOAL statement ...
```

HAL/S Equivalent Form:

```
STEP_163: ... rest of HAL statement ...
```

---

GOAL Time Prefix:

```
WHEN <COUNT_DOWN_CLOCK> IS
   -80 HRS 27 MIN 00 SEC THEN
     ... rest of GOAL statement
```

Purpose:

Cause the GOAL program to wait until the <COUNT_DOWN_CLOCK> value is greater than or equal to -80:27:00.

Equivalent HAL/S form:

```
WAIT UNTIL XXXX;
```

In this example, "XXXX" is -80 hrs., 27 minutes, as converted to absolute time in "machine units".

The effect of the WAIT statement of HAL/S is independent of any following statements (it is not a "prefix" as in GOAL). However, by delaying processing of statements which follow it, the WAIT works as if it were the prefix. The effect is also identical to the GOAL "WHEN" prefix since any clock time greater than or equal to the specified time will cause the halt to be ended and/or ignored.

Note also that all HAL/S Real Time statements assume a single real time clock. In order to allow the possibility of multiple clocks (not ruled out by GOAL specification), the translator will have to incorporate a scaling and offset algorithm so that all clock function designators can be driven by the single HAL/S Real Time Operating System clock.

---

GOAL Time Prefix:

```
AFTER <COUNT DOWN CLOCK> IS -80 HRS 27 MINUTES
   00 SECS THEN ...
```

Purpose:

Delay execution of the particular statement until after the time named. In any time-dependent digital system, "after" may only mean "one system clock tick" later than the specified time. Thus, this statement is the same as a WHEN statement with a time value increased by the unit of the basic clock period.

Equivalent HAL/S form:

Thus, the HAL/S Equivalent becomes:

```
WAIT UNTIL XXXX + ΔX;
```

where XXXX is the machine-unit (absolute) equivalent of the specified time and ΔX is the granularity of time in the clock, expressed in "machine units".

---

GOAL Statement:

```
IF (MIDDLE GIMBAL ANGLE) IS GREATER THAN (MIDDLE
  GIMBAL LIMIT) THEN
```

HAL/S Equivalent Form:

```
IF MIDDLE_GIMBAL_ANGLE >
   MIDDLE_GIMBAL_LIMIT THEN DO;
```

In this equivalent form, "MIDDLE_GIMBAL_ANGLE" is assumed to be a HAL/S function reference to a block which performs the function designator I/O.

---

GOAL Verify Prefix:

```
VERIFY <MIDDLE GIMBAL ANGLE> IS LESS THAN
 (MIDDLE GIMBAL LIMIT) ELSE ...
```

HAL/S Equivalent Form:

```
READ (CHANNEL) MIDDLE_GIMBAL_ANGLE ;
IF MIDDLE_GIMBAL_ANGLE >=
   MIDDLE_GIMBAL_LIMIT THEN
 DO;*
   :
   :
   HAL/S Statements equivalent to "ELSE" part
     of GOAL statement
   :
   :
 END;*
```

* Note that if the GOAL form following this prefix becomes a single HAL/S statement, then the DO...END group is not needed.

---

### 4.2.2 External Test Actions

GOAL Statement:

```
SEND 10V TO <POWER SELECTOR 1>, <POWER SELECTOR 2>;
```

Purpose:

This statement is used to implement an I/O operation to some specific external device. Assuming a mechanism which employs a channel address and a subaddress for location within the channel, then a source language construct such as the HAL/S FILE statement might be used to specify such I/O.

HAL/S Equivalent Form:

```
FILE(Channel, address of power selector 1) = 10;
FILE(Channel, address of power selector 2) = 10;
```

The lower case texts in these statements will be filled-in with the appropriate references to hardware in the translated form of arithmetic literals. In the event that a conversion routine is specified in the data bank for the function designator in question, then an appropriate function reference with "10" as its argument could be used in place of the simple literal shown.*

---

GOAL Statement:

```
APPLY PRESENT VALUE OF <POWER BUS 1> TO
   <POWER BUS 2>;
```

Purpose:

This statement is used to implement an input operation from power bus 1 and a corresponding output operation to power bus 2 with no intermediate storage in program variables.

HAL/S Equivalent Form:

```
FILE(Channel, address of POWER BUS 1) = FILE (channel,
    address of POWER BUS 2);
```

In this example, as in the previous, the file statement of HAL/S has been assumed to be the I/O mechanism used. The translator must generate the channel addresses and channel numbers based upon data bank information.

---

GOAL Statement:

```
ISSUE (OCTAL SEVENS), (OCTAL ONES) TO
  <PANEL LIGHTS 32>, <PANEL LIGHTS 31>;
```

---

* Throughout the report we have used the "file statement" of HAL/S in the function designator context. Other possibilities include various forms of READ or WRITE statements, depending on particulars of Flight Computer System.

Purpose:

This statement is supposed to send a "digital pattern" in the form of a numeric internal variable to a selected I/O word identified by the selected function designators. In this case, the internal variable (OCTAL SEVENS) is sent to "PANEL LIGHTS 32" and (OCTAL ONES) is sent to "PANEL LIGHTS 31";

HAL/S Equivalent Form:

Assuming a channel designation "a" and channel address designations "b" and "c" derived by the translator from the data bank at compile time, then the following HAL/S File statements could achieve the same effect:

```
FILE(a,b) = OCTAL_SEVENS;
FILE(a,c) = OCTAL_ONES;
```

---

GOAL Statement:

```
ISSUE PRESENT VALUE OF <CH 63> TO <CH 11>;
```

Purpose:

As in the above example, this is simply some I/O of data to a particular set of channel addresses in some channel or channels of the implementations' I/O hardware. Assume that the I/O designations of channel "a" and addresses "b" or "c" correspond (via the Data Bank) to "CH 63" or "CH 11" respectively.

HAL/S Equivalent Form:

```
FILE(a,b) = FILE(a,c);
```

Note that this I/O entirely bypasses setting any programmer-defined variables during the course of the I/O operation.

---

GOAL Statement:

S 104 AFTER <CLOCK> IS -1 HRS, OPEN <HELIUM SUPPLY>;

Purpose:

This GOAL statement has a time prefix, and a label prefix, used to control when and under what conditions the action of sending a bit valve corresponding to an OPEN value to an external register symbolically identified by the function designator "HELIUM SUPPLY".

HAL/S Equivalent Form:

Assuming that the compiler translates the units of hours into units of seconds (or any others depending on implementation), and that the value of a bit controlling a valve is "0" if "OPEN", and that the symbolic address of the HELIUM SUPPLY valve control bit is given by channel a, subchannel address b, then the following HAL/S statement accomplishes the same function - with the time prefix implemented by a WAIT statement:

```
S_104
  WAIT UNTIL -3600.0001
  FILE(a,b) = BIN'0';
```

The label has been translated directly; the time value for the clock in the executive is treated as seconds in this example, with a granularity of .1 millisecond.

---

GOAL Statement:

STEP 5: TURN ON (THRUST OK IND) FUNCTIONS;

Purpose:

This GOAL statement is supposed to set all the bits in all the active function designators of a STATE table to the "ON" value.

HAL/S Equivalent Form:

Assuming that each I/O operation could translate into a single FILE statement, the result of this one would be a series of FILE statements.

---

GOAL Statement:

```
RECORD (INTERNAL TIME) TO <MAG 2-5>;
```

Purpose:

This statement is supposed to send data from a GOAL internal variable named "Internal time" to the function designator identified. This means in effect that an I/O operation of writing the internal value to the implicit I/O address of the function designator is required.

HAL/S Equivalent Form:

Assuming that channel a, address b is the external location of the function designator "MAG 2-5" and that no conversion could be used to do this I/O operation:

```
FILE(a,b) = INTERNAL_TIME;
```

---

GOAL Statement:

```
DISPLAY TEXT (ALL SYSTEMS READY FOR POWER TRANSFER)
    TO <CRT 9>;
```

Purpose:

This statement is supposed to write the given text out onto a character-oriented I/O device, namely a display. Since this output is character-oriented, the equivalent HAL/S form will have a character output statement form.

HAL/S Equivalent Form:

```
WRITE(display) 'ALL SYSTEMS READY FOR POWER TRANSFER';
```

The channel number of the display unit will have to be filled in by the translator, by means of Data Bank information.

---

GOAL Statement:

```
AVERAGE 10 READINGS OF <IU COOLANT  TEMPERATURE> AND
    SAVE AS (COOLANT TEMP);
```

Purpose:

The purpose of this statement is to read a hardware input channel 10 times, averaging the readings. No time delay other than the response time of the software is to be employed explicitly between successive readings.

HAL/S Equivalent Form:

The HAL/S equivalent could be coded with a file statement enclosed in a DO FOR loop, followed by the final division operation needed to produce an average from a sum of the components. Thus:

```
SIGMA = 0
DO FOR I = 1 TO 10;
TEMP = FILE (a,b);
SIGMA = SIGMA + TEMP;
END;
COOLANT_TEMP = SIGMA/10.0;
```

---

GOAL Statement:

```
READ <PC STAGE INLET PRESSURE> AND SAVE AS
   (INLET PRESSURE);
```

Purpose:

This statement is supposed to evaluate the present value of the function designator addressed, then assign the value into the internal variable INLET PRESSURE.

HAL/S Equivalent Form:

This is another example in which the file statement may be used. However, in this case it will be assumed that a conversion is required as indicated in the corresponding Specify statement for the function designator:

```
INLET_PRESSURE = FILE(a,b);

INLET_PRESSURE = CONV_FUNC(INLET_PRESSURE);
```

In this case, (a,b) is the address of the input channel/ location and that CONV_FUNC is the conversion function required.

---

GOAL Statement:

```
READ (TABLE A) FUNCTIONS AND SAVE AS (CURRENT VALUE);
```

Purpose:

The purpose of this statement is to evaluate the current value of all the function designators (which are active) in the TABLE_A and assign the results into corresponding positions in the table column indicated.

HAL/S Equivalent Form:

Assume that the mapping of each function designator into an I/O operation can be done in terms of an I/O address pair of channel and address: (a,b) for a file statement. Further, assume for simplicity, that the pointer table associated with this table contains the addresses for each function designator on a common channel a. Also assume that an array of booleans is associated with the table to distinguish active functions from inactive functions, and that "z" represents the column index of "CURRENT VALUE). Thus:

| | |
|---|---|
| TABLE_A dimensioned x by y. | Scalar |
| TABLE_A_BOOLEANS, dimensioned x. | Boolean |
| TABLE_A_POINTERS, dimensioned x. | Integer |

The HAL/S statements are:

```
DO FOR I = 1 TO x;
   IF TABLE_A_BOOLEANS I  THEN
   TABLE_A I,z  = FILE(a,TABLE_A_POINTERS I);
END;
```

The IF statement checks for whether or not the row is active, in which case the input file statement completes the operation.

---

GOAL Statement:

```
REQUEST TEXT (DEGREES PITCH) FROM <CRT 7> AND SAVE
  AS (DEG PTCH);
```

Purpose:

Display a request on the console and then reads in the result. Since a character-oriented operation with conversion to scalar is involved, a conversion function in the HAL/S version will be used.

HAL/S Equivalent Form:

```
WRITE(CRT 7) 'DEGREES PITCH';
READ(CRT_7) DEG_PTCH; (automatic conversion is done)
```

---

### 4.2.3 Internal Sequence Control

GOAL Statement:

```
DELAY 5 SECS;
```

Purpose:

Cause the program to "go to sleep" for a time interval of 5 seconds.

HAL/S Equivalent Form:

```
WAIT 5;
```

Where the units of time involved in the wait statement are assumed to be the same as the GOAL units prior to translation, or the translator will scale the internal value accordingly.

---

GOAL Statement:

```
WAIT UNTIL <SIVB 3200 PSIA SUP VENT> IS OPEN;
```

Purpose:

Cause the program to "go to sleep" for a time interval of unspecified length until the boolean (state) function designator is recognized as being "OPEN".

HAL/S Equivalent Form:

```
WAIT UNTIL¬SIVB_3200_PSIA_SUP_VENT
```

This is equivalent, provided that in the case of function designators used as events, the translation process turns them into event variables, which may be tested in the context shown. The definition of "OPEN" is defined to be a binary value of 0.

---

GOAL Statement:

```
GO TO S 20;
```

HAL/S Equivalent Form:

```
GO TO S_20;
```

where the label S 20 of GOAL has been translated into a HAL/S identifier.

---

GOAL Statement:

```
STOP AND INDICATE RESTART LABELS S100, S200;
```

Purpose:

To give the GOAL program an ability to cease active execution and wait for operator intervention, followed by a jump to one of the indicated labels.

HAL/S Equivalent Forms:

Assume that the restart routine is in line, then:

```
BACK:  DO WHILE TRUE;
   WRITE (CONSOLE) 'ENTER RESTART LABEL S100 OR S200';
   READ (CONSOLE) RESTART;
   IF RESTART = 'S100' THEN GO TO S_100;
   IF RESTART = 'S200' THEN GO TO S_200;
   WRITE (CONSOLE) 'ERROR';
   END BACK;
```

It is assumed that if the keyboard message is other than 'S100' or 'S200' then an 'ERROR; is sent to the CONSOLE and the entire sequence is repeated.

---

GOAL Statement:

```
TERMINATE;
```

Purpose:

The TERMINATE statement of GOAL is used to:

a) Return control to the calling program if it is found in a GOAL subroutine.

b) Stop execution of a program if found at the program level, returning control to the caller.

HAL/S Equivalent Form:

In either case, the HAL/S statement:

```
RETURN;
```

will be equivalent. This HAL/S statement will return control to the calling routine from a HAL/S procedure. Assuming the PERFORM PROGRAM maps into a:

```
"SCHEDULE X"          /*SCHEDULE*/
"WAIT FOR ¯X"         /*AND WAIT FOR COMPLETION*/
```

sequence in HAL/S, the return to the operating system of this statement at the PROGRAM level in HAL/S will have the same effect - the caller resumes execution after the "WAIT FOR COMPLETION OF X" part of the HAL/S form.

---

GOAL Statement:

```
TERMINATE SYSTEM;
```

Purpose:

This statement is supposed to shut down an entire system of programs. As such, it has special executive effects. Its operation was discussed in Section 3.0 above.

---

GOAL Statement:

```
REPEAT STEP 5 THRU STEP 7;
```

HAL/S Equivalent Form:

For a discussion of this equivalent and the purpose of this GOAL form, see Section 3.0.

---

### 4.2.4 Arithmetic/Logical Operations

GOAL Statement:

```
ASSIGN (FLAG B) = ON;
```

Purpose:

Set a new value of "ON" into the (FLAG B) internal name.

HAL/S Equivalent Form:

```
FLAG_B = TRUE;
```

This equivalence assumes the convention that "ON" has a binary value of "1" or "true". FLAG_B is assumed to be a HAL/S BOOLEAN.

---

GOAL Statement:

```
LET (A) = (A) + 1;
```

Purpose:

Assign a new value to GOAL internal variable (A) calculated as shown.

HAL/S Equivalent Form:

```
A = A + 1;
```

---

### 4.2.5 Execution Control

GOAL Statement:

```
CONCURRENTLY PERFORM PROGRAM (BE01);
```

HAL/S Equivalent Form:

```
SCHEDULE BE01;
```

---

GOAL Statement:

```
CONCURRENTLY VERIFY <PRESS ENG 102_GIMBAL>
IS BETWEEN 1665PSIA and 1465PSIA and
DISPLAY EXCEPTION TO <CRT12>;
```

HAL/S Equivalent Form:

```
EX_DES16 = PRESS_ENG_102 GIMBAL;
HIGH = 1665;
LOW = 1465;
SCHEDULE VERIFY;
DISP = CRT12;
SCHEDULE DISPLAY_EXCEPTION ON EXCEP;
```

In the above example it is assumed that EX_DES16, HIGH, LOW and DISP have been previously declared as compool variables. The verify procedure uses EX_DES16 to access the test point PRESS_ENG_102_GIMBAL and test between the limits of 1665 and 1465. If an out of tolerance condition occurs then the event EXCEP will be signalled. The procedure DISPLAY_EXCEPTION will use an appropriate compool variable to be displayed on DISP. If a GOAL table were involved above, the translated form would involve an iterative DO group to scan the table.

---

GOAL Statement:

```
RELEASE STEP 10;
```

Purpose:

Step 10 had "concurrently' set up some cyclic process at a previous time. This statement "releases" the concurrent process initiated, by removing it from the implicit executive queues involved (see Section 3.0).

---

GOAL Statement:

RELEASE ALL

Purpose:

Terminate all concurrently scheduled operations within the current GOAL component.

HAL/S Equivalent Forms:

Assuming the translator had generated process names A, B, and C in translating the current component, then the release would become:

```
TERMINATE A,B,C;
```

Terminate causes immediate cessation of execution; if cessation prior to the next execution cycle is required then use: "CANCEL A, B, C; (see Section 3.0).

---

GOAL Statement:

PERFORM PROGRAM (LVDC POWER ON);

Purpose:

Branch to the program selected, execute it and return.

HAL/S Equivalent Form:

```
DECLARE LVDC_POWER_ON PROGRAM EVENT
   :
   :
SCHEDULE LVDC_POWER_ON;
WAIT FOR  ¬LVDC_POWER_ON
   :
   :
```

The HAL/S PROGRAM form requires that a SCHEDULE invoke it, not a call. The call action can be duplicated with SCHEDULE and WAIT; SCHEDULE invokes the program and turns "on" its "PROGRAM EVENT;" The WAIT is for the PROGRAM EVENT to turn off, indicating that the program is done and out of the queue.

---

GOAL Statement:

```
PERFORM CRITICAL SUBROUTINE
   (CALCULATE DELAY TIME);
```

Purpose:

Inhibit all software-level interruptions by other system components (this does not refer to physical interrupts which the OS handles) during the execution of the subroutine.

HAL/S Equivalent Form:

```
X = PRIO;
UPDATE PRIORITY TO HIGHEST;
CALL CALCULATE_DELAY_TIME;
UPDATE PRIORITY TO X;
```

This HAL/S form achieves the same function, since any routine at the highest priority may not be interrupted. Since the new, highest priority is only to last while the subroutine is being performed, the second update and a temporary variable are required to return to the original state following the UPDATE.

---

### 4.2.6 Interrupt Control

GOAL Statement:

```
WHEN INTERRUPT <POWER FAILURE> OCCURS
  GO TO STEP 9000;
```

**Purpose:**

Send control to step 9000 when the indicated interrupt occurs.

HAL/S Equivalent Form:

Refer to Section 3.0 for a discussion of the implicit operating system functions of this statement and how it would be handled in HAL/S.

---

GOAL Statement:

```
WHEN INTERRUPT <CLOCK T-22 MINS> OCCURS PERFORM
 SUBROUTINE (START_TANK CHILLDOWN) AND RETURN TO
 STEP 9999;
```

Purpose:

When the indicated interrupt occurs, perform a subroutine then unconditionally branch to the indicated step number.

HAL/S Equivalent Form:

Refer to Section 3.0.

---

GOAL Statement:

```
DISABLE STEP 20;
```

Purpose:

Inhibit a software interrupt set up by a "WHEN INTERRUPT" at the step indicated (STEP 20);

HAL/S Equivalent Form:

Refer to Section 3.1.4.

---

### 4.2.7 Table Control

GOAL Statement:

```
ACTIVATE (TABLE A) ROW 1, ROW 3;
```

Purpose:

Activate the function designators associated with the indicated rows so that future I/O to this table will include the rows in question.

HAL/S Equivalent Form:

The HAL/S translation of a GOAL table involves a data array and an activation array of BOOLEANS. Assuming that "A_TABLE_A" is TABLE A's activation array, and that "TRUE" means the corresponding row is active, then the HAL/S equivalent is simply:

$$A\_TABLE\_A_1 \ , \ A\_TABLE\_A_3 \ = \ TRUE;$$

---

GOAL Statement:

```
INHIBIT (TABLE A) ROW 2, ROW 3;
```

Purpose:

Disable the function designators associated with the indicated rows, so that future I/O to this table will exclude the rows in question.

HAL/S Equivalent Form:

Analogous to the ACTIVATE case:

$$A\_TABLE\_A_2 \ , A\_TABLE\_A_3 \ = \ FALSE;$$

## 4.3 System Statements

The GOAL system statements serve primarily as inputs governing the course of the GOAL-HAL/S translator's operation. Some of these statements have implications which are reflected in the HAL/S code produced.

### 4.3.1 Boundary Statements

GOAL Statement:

```
BEGIN DATA BANK (S2 DATA BANK) REVISION 0;
```

Purpose:

Mark the beginning of the Data Bank named for input to an appropriate data bank compilation. This statement has no equivalent in the generated HAL/S code since the generation process resolves all references in detail.

---

GOAL Statement:

```
BEGIN PROGRAM (LV TM CAL) REVISION 0;
```

Purpose:

Mark the beginning of a GOAL program.

HAL/S Equivalent Form:

```
LV_TM_CAL_0:  PROGRAM;
```

Note that a series of revision numbers is an operating system and translator system feature (see Section 3.0), not a language feature. These two are kept separate in HAL/S.

---

GOAL Statement:

```
BEGIN SUBROUTINE (FORCE_TERM)(PARAMETER_1);
```

Purpose:

This statement begins a GOAL subroutine block, passing one formal parameter.

HAL/S Equivalent Form:

A direct HAL/S equivalent exists if the SUBROUTINE of GOAL maps into the HAL/S PROCEDURE block form. Then the BEGIN SUBROUTINE becomes:

```
FORCE_TERM: PROCEDURE ASSIGN(PARAMETER 1);
```

---

GOAL Statement:

```
BEGIN MACRO AZ (PARAMETER 1);
```

Purpose:

This statement specifies the start of a GOAL source language macro. Since macros only refer to the source code and are expanded within the translator, there is no HAL/S equivalent.

This does not exclude use of a HAL/S macro form as part of the generated HAL/S source code if a GOAL to HAL/S translation is done at the source language level. But such use is completely separate from the GOAL macro and its use.

---

GOAL Statement:

```
END DATA BANK;
END PROGRAM;
END SUBROUTINES:
END MACRO;
```

Purpose:

Mark the end of the particular GOAL block (or component).

HAL/S Equivalent Form:

Since DATA BANK and MACRO forms exist only in the source inputs to the GOAL-HAL translation, the END statement for these blocks are similarly devoid of HAL/S equivalents.

The HAL/S equivalents of END PROGRAM and END SUBROUTINE are provided by the HAL/S CLOSE statement; i.e.

```
CLOSE;
  or
CLOSE X;
```

---

GOAL Statement:

LEAVE:

Purpose:

Link to some other language subroutine (in object form). This is an operating system function which has no equivalent in HAL/S (see Section 3.0).

---

GOAL Statement:

RESUME;

Purpose:

Return to GOAL compiling after LEAVE. See comments in Section 3.0 and in the discussion of "LEAVE" above.

### 4.3.2 System Directive Statements

The GOAL system directive statements are USE, FREE, and SPECIFY - all of which refer to the DATA BANK.

These statements have no HAL/S equivalent (see Section 3.0) and are system-oriented inputs to the translation process. All data bank references are resolved by the translator - the data bank (as such) does not appear in the output of the translator.

### 4.3.3 Special Aid Statements

GOAL Comment Statement:

```
$  POWER TRANSFER SWITCH VERIFICATION;
```

Purpose:

Annotate listing.

HAL/S Equivalent Forms:

a. Embedded Comment

```
/* POWER TRANSFER SWITCH VERIFICATION */
```

b. Comment Line

```
pos 1
  C     POWER TRANSFER SWITCH VERIFICATION
```

Note: Comments map directly into comments with identical text and minor syntax changes.

---

GOAL Statement:

```
EXPAND MACRO ADJUST, <AC SIGNAL>,
   (0.5V), 5340,;
```

Purpose:

Expand a GOAL substitution macro prio to further compilation.

HAL/S Equivalent Form:

None. All macros involve expanding GOAL source statements, so the macro itself and its expansion disappear in the translation process.

---

GOAL Statement:

```
REPLACE <POWER SUPPLY NO 1>
    WITH <POWER SUPPLY NO 2>;
```

Purpose:

Replace is a source level substitution of characters prior to compilation, a sort of "mini-macro" facility. As with macros, it disappears following translation because it must be expanded in order to translate.

# 5.0 CONCLUSIONS

A GOAL-to-HAL translation with the design and operational objective of running GOAL-derived HAL-compiled code in the Shuttle onboard computers has been evaluated in this Report and pronounced feasible. The individual categories of GOAL statements as set forth in the NASA/KSC Syntax Diagram Handbook and Textbook have been studied and divided into 1) syntax and semantic issues and 2) other issues.

The overall GOAL statement roster has been mapped. Those statements which are syntax and semantic issues only have been mapped into HAL/S equivalents. In the case of GOAL statements which are basically operating system issues, two courses of action have been proposed. First, a GOAL_MASTER_PROGRAM has been identified and described as the means of fitting the GOAL-derived checkout programs into the Flight Computer Operating System (FCOS). Secondly, in some cases, limitations have been proposed to some HAL features, such as the overlapping of REPEAT groups. The primary effect of these proposed limitations is to enhance reliability. There are no inherent HAL/S syntax feedback loop limitations, such limitations being implementation-dependent.

GOAL-to-HAL/S translation has matured as a concept to where it is ready for a Specification. Such a Specification must include a careful statement about operating system assumptions, and it should spell out requirements in the areas of implementation, verification, and documentation.