NASA Technical Memorandum 58258

NASA-TM-58258 19840018260

A Representational Basis for the Development of a Distributed Expert System for Space Shuttle Flight Control

May 1984

National Aeronautics and
Space Administration

THIS DOCUMENT SUPERSEDES NASA-TM-58258 WHEREIN JSC FORM 1424 WAS OMITTED.

A Representational Basis for the Development of a Distributed Expert System for Space Shuttle Flight Control

John J. Helly, Jr.
Department of Computer Science, University of California, Los Angeles
and
The Aerospace Corporation

William V. Bates
NASA Johnson Space Center

Mel Cutler, Steve Kelem
The Aerospace Corporation

Langley Trust

LIBRARY MATERIAL REQUEST

INSTRUCTIONS:

- *List only one Library Item on a form – PLEASE.*
- *For purchasing books or theses use Langley Form 125, "Purchase Request/Purchase Order."*
- *Obtain approval of Section Head or higher official for classified material.*

NOTICE – WARNING CONCERNING COPYRIGHT RESTRICTIONS

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use," that user may be liable for copyright infringement.

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

TO BE FILLED IN BY REQUESTER *(As appropriate)*

Document or Book Call Number *(If available)*

Title an

Computer and Electrical Engineering.

AN EVALUATION OF THE DIRECTED FLOW GRAPH METHODOLOGY Final Report

Wesley E. Snyder and Sarah A. Rajala May 1984 61 p Original contains color illustrations

(Grant NAG1-20)

Report (NASA-CR-173593; NAS 1.26:173593) Avail: NTIS HC A04/MF A01 CSCL 09B

Source The applicability of the Directed Graph Methodology (DGM) to the design and analysis of special purpose image and signal processing hardware was evaluated. A special purpose image processing system was designed and described using DGM. The

DDC N design, suitable for very large scale integration (VLSI) implements a region labeling technique. Two computer chips were designed,

Journa both using metal-nitride-oxide-silicon (MNOS) technology, as well as a functional system utilizing those chips to perform real time

Article region labeling. The system is described in terms of DGM primitives. As it is currently implemented, DGM is inappropriate for describing synchronous, tightly coupled, special purpose systems. The nature of the DGM formalism lends itself more readily to modeling networks of general purpose processors. R.S.F.

Volum N84-26325*# Notre Dame Univ., Ind. Dept. of Electrical

If Library does not have material should it be: *(Check box)*

Borrowed from another Library	Ordered
☐ Yes ☐ No	☐ Yes ☐ No ☐ Indefinite Loan ☐ Library Copy

ROUTING

FROM

1. *(x proper box)* ☐ NASA ☐ NON-NASA ☐ Foreign National U.S. Citizen Yes ☐ No ☐

Requester	Mail Stop No.	Date

Division–Branch–Section or Affiliation for Non-NASA Library Patron	Telephone Extension

2. Section Head or Higher Official Signature *(Approval for classified material only)*

TO

3. 185/LIBRARY

4.

LIBRARY INTERNAL USE ONLY

	Order
	On Order
	L has hardcopy
	No Folder
	No Film
	Borrow
	Call In
	Reference
	Other:

LIBRARY ACTION

	Being obtained from Interlibrary Loan.
	Material is charged to you.
	Material is on indefinite loan to:
	Item is being called in from another patron and will be sent to you.
	Material is given to you.
	Other *(Specify)*

Table of Contents

List of Figures

List of Tables

Abstract

A new representation of malfunction procedure logic which permits the automation of these procedures using Boolean normal forms is presented. This representation is discussed in the context of the development of an expert system for Space Shuttle flight control including software and hardware implementation modes, and a distributed architecture. The roles and responsibility of the flight control team as well as previous work toward the development of expert systems for flight control support at Johnson Space Center are discussed. The notion of malfunction procedures as graphs is introduced as well as the concept of hardware-equivalence.

1 Introduction

The purpose of this paper is to present the preliminary results of work currently underway to automate part of the Space Shuttle Vehicle (SSV) flight control team function through the application of results from automata and sequential machine theory, and expert system research. Preliminary applications have been developed based on these results using software tools developed for the *computer-aided design (CAD)* of Very-Large Scale Integrated (VLSI) circuits and a high-level language for the implementation of expert systems. Although developed specifically for the SSV, the approach presented here appears to have application to the larger class of problems addressed by control systems in general. The text describes the development of a data representation which facilitates the automatic detection and resolution of anomalies occurring during SSV flight operations. by introducing

1. a method for the translation to, and representation of malfunction procedures in Boolean form,

2. the notion that malfunction procedures can be treated as graphs, and

3. the use of *normal forms* for the standardization of data structures to be used in computer processing of the procedural information.

In addition to providing a data representation which will simplify software development and reduce processor load with respect to previous approaches, we expect these improvements to

1. enhance the ability of the flight control team to detect and analyze anomalies faster and to an extent greater than humanly possible. This is especially true for *multiple dependent failures*; anomalies which are caused by one or more prior failures.

2. permit gradual transitioning from the current ground processing methods to an increasingly automated and more distributed flight control system while preserving the continuity and integrity of existing operational capabilities.

3. provide an environment designed to capture expert knowledge in machine-processable form thereby preserving individual and collective expertise of the flight control team members over time.

4. permit the arbitrary distribution of processing between the ground and the spacecraft as well as between hardware and software.

The discussion will begin with a description of the organizational structure of the *flight control team* and its responsibilities in SSV mission operations. We shall then briefly discuss expert systems in general terms before introducing a new representation for malfunction procedures using Boolean logic. Following this will be a discussion of the application of VLSI design techniques used to derive, from the Boolean representation, a *disjunctive normal form matrix*. The matrix representation

enables the processing of malfunction procedures by either software (using an *expert systems* approach) or conversion to hardware in the form of programmable logic arrays (PLAs). A short discussion of the impact this methodology can have on overall system design for space operations follows including the selective distribution of processing to support increased vehicle autonomy as well as reduced life-cycle costs for ground operations.

1.1 Flight Control Team

As described in [22], the *modus operandi* of the flight control team can be summarized, with respect to responsibilities and activities during the operations phase of a space shuttle flight, as follows:

1. The fundamental role of the flight control team is to monitor and analyze Space Shuttle systems for anomalous behavior, to analyze anomalies when they occur, to determine corrective action, and to coordinate this action with the flight crew.

2. Most of the data from which these determinations and analyses are made are based on computer processed, telemetry data originating from sensors on-board the SSV and transmitted to the ground processing system.

Table 1 provides an abbreviated list of activities for which the flight control team is responsible for during an SSV mission.

1.1.1 Organizational Structure

The SSV flight control team is organized as a **hierarchy** as depicted in figure 1. This organization reflects both the diversity and the discrete compartmentalization of SSV system disciplines. In general, SSV flight operations and procedures reflect a high degree of structure and definition in that they are:

1. **precise** : operational procedures are developed, tested, and reviewed under simulated mission conditions to achieve high precision.

2. **deterministic** : the SSV is a well understood, although complex, finite system. On-board computers and sensors provide telemetry data. These data in combination with SSV system design information, make possible the determination of the global state of the vehicle.

3. **documented** : Operational procedures are catalogued in printed form. SSV system performance is analyzed in detail both during and after a mission. Problems and corrective action (successful and unsuccessful) are reviewed and documented.

1.1.2 Task Complexity and System Reliability

The SSV is significantly more complex than prior United States manned space vehicles, primarily due to high redundancy of vehicle subsystems and the implementation of designs which reduce single point failures within any single subsystem. While, in terms of system design, subsystem

Table 1: Flight Controller Functions

Flight Dynamics	Prelaunch Analysis Trajectory Monitoring Spacecraft Tracking Aerodynamics and Structures Monitor Onboard Navigation State
SSV Systems	Prelaunch Analysis Manage Spacecraft Systems Consumables Analysis Fault Detection and Isolation Fault Recovery
SSV Data Acquisition	Manage Communication and Data Systems Manage Flight Data
Payloads	Manage Payload Activities Manage Payloads and Support Equipment
Operations Management	Policy Making Ground Network Management Coordinate Crew Activity Plan Landing Operations Medical Support Supplemental Technical Support

redundancy can greatly improve overall system reliability it also means that a system can have a larger number of states. The number of states is directly proportional to the degree of redundancy. The number and characteristics of these states must be known and considered in the analysis of known or possible system or subsystem failures. In addition to the increase in complexity introduced through redundancy, complexity is increased through the use of interdependent subsystems such that failure of a single component can affect the performance of several different subsystems both instantaneously, through total functional loss, and over time, through degraded performance. The detection, isolation and correction of any fault in the SSV systems is extremely important in terms of both crew safety and mission success. Not only is the detection and correction of real failures necessary but, as pointed out in [25], to maximize system availability, unjustified system shutdown or *pulldowns* must also be minimized.

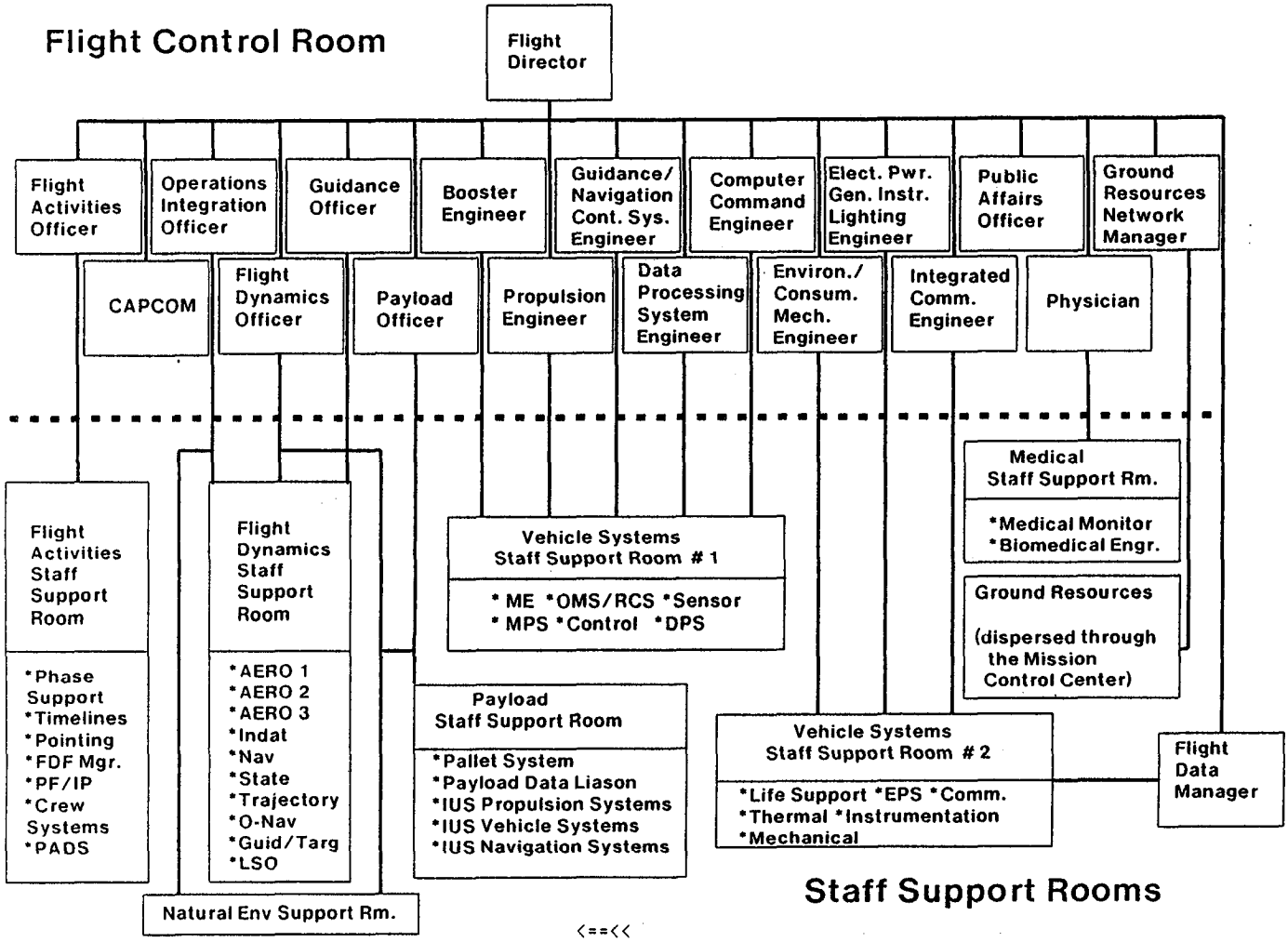

Figure 1: Flight Control Team Organization

1.1.3 Analytic Tools

The detection, analysis and recovery of faults is the responsibility of the flight control team and the flight crew. The principal tools available to flight controllers and crew to assist them in these tasks are either pocket checklists (small binders designed to hold all the responses to any problem that requires crew action within five minutes) or large books containing several hundred procedures, each comprising instructions to be executed when a fault arises which does not require immediate response. These more involved directions, known as **malfunction procedures** (or simply, malfs) are critical to accurate and speedy fault detection and isolation.

1.2 Malfunction Procedures

In the current method of fault isolation, the flight crew and flight control team consult malfunction procedures and reach a conclusion based on selected telemetry measurements and on-board observations of switch settings (i.e., the vehicle state). Similar procedures are used to reestablish full or partial system or subsystem function once an anomaly has been analyzed.

1.2.1 Types of Procedures

Four basic types of malfunction procedures are used by the flight teams [18]. Although they serve differing functions they are collectively referred to as malfunction procedures. While this work focuses primarily on *system or block malfunction procedures*, the methods presented in this paper are applicable to all of the types of procedures described below.

1. **System or Block Malfunction Procedures**: The system malfunction procedure is a deterministic representation of yes/no questions and crew procedures in the block and line format of a flow chart. This form ideally lends itself to expert systems-type programs due to its rule-based, "if-then" tree structure. Figure 2 is a representative block procedure.

2. **Special Subroutines (SSR)**: These are procedures that are accessed by many different malfunction procedures. Also, the SSR is in more of a checklist format in contrast to the block type used in the system malfunction procedures. Figure 3 is a representative SSR.

3. **Failure Recovery Procedures (FRP)**: The FRP is a non-deterministic procedure used when a General Purpose Computer (GPC) failure has already been determined and certain steps must be taken to reconfigure the orbiter to take advantage of any remaining capability in the failed GPC. Normally, FRP's are lengthier than the other malfunction procedures. Figure 4 is a representative FRP.

4. **Super Malfs**: The *super malf* acts as a transition aid from the checklist to the malfunction procedure itself. These procedures must be executed within five minutes of the annunciation of an alarm and are critical to crew and mission safety. The wording and format are identical to the ORB PCL (orbit pocket checklist) carried by the flight crew. Figure 5 is a representative pocket checklist procedure.

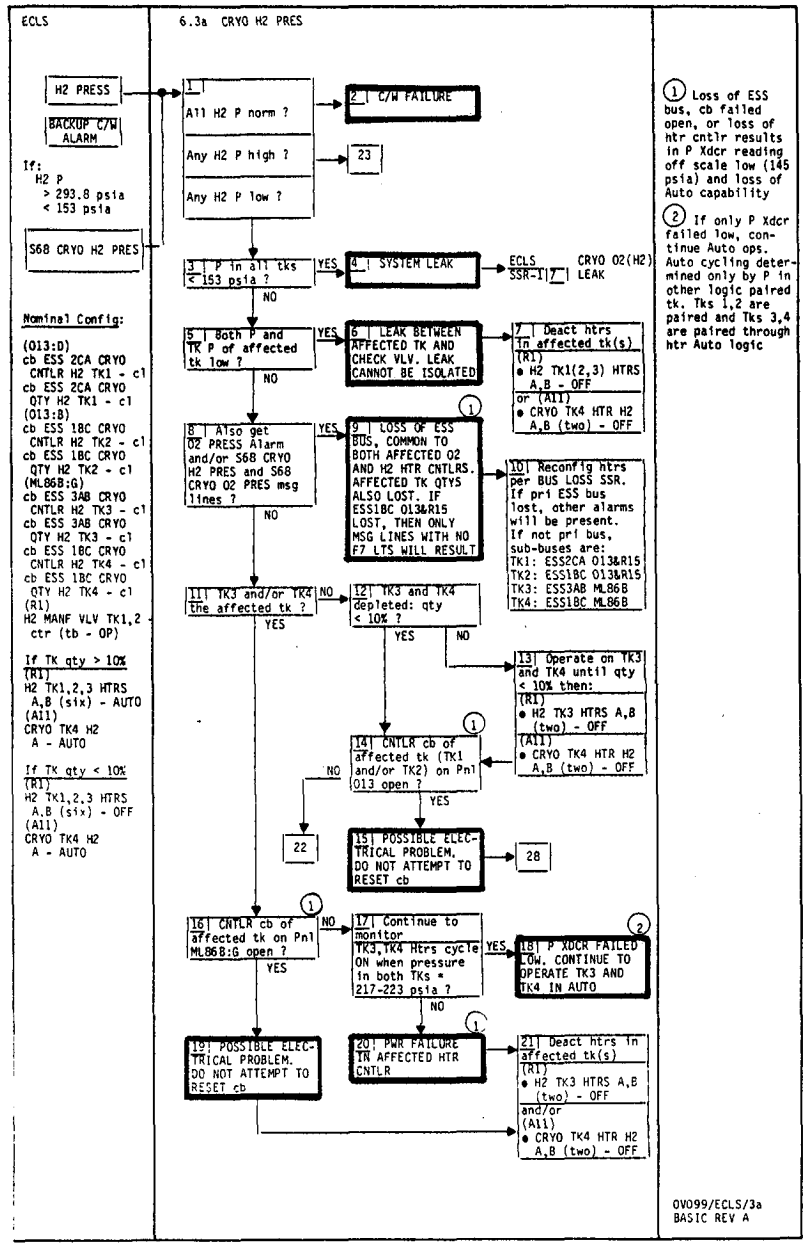

Figure 2: A Block Malfunction Procedure

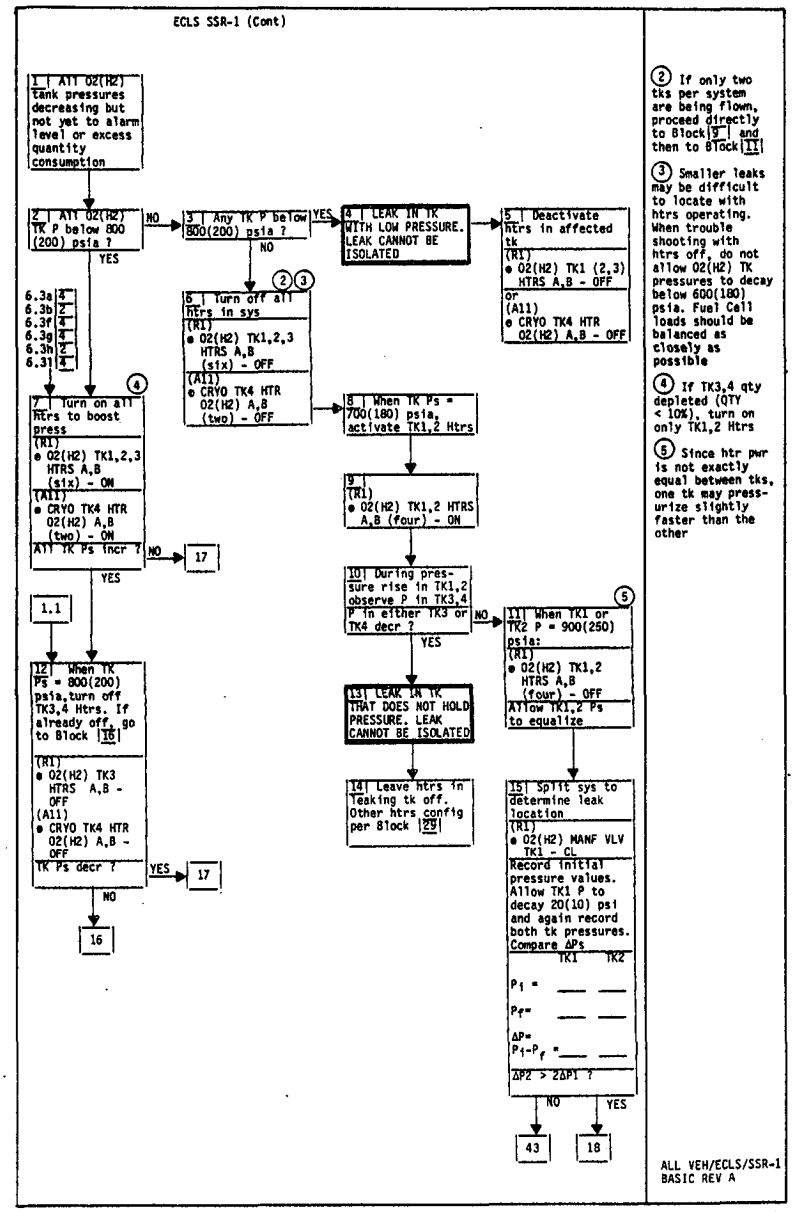

Figure 3: A Special Subroutine (SSR)

GPC FRP-7

DPS RECONFIG FOR LOSS OF AV BAY COOLING (ASCENT/ORBIT)

NOTE

Procedure is designed to protect GPCs from overheating by moving G2, SM, BFS functions into cooled Av Bay GPCs (if not already there). G3FD and G2FD (or redundant G2) functions are placed into uncooled Av Bay GPCs and powered off. If BFS is moved, new BFS GPC will be dumped to provide good BFS for Entry

For Ascent, this FRP assumes PWRDN, LOSS OF AV BAY FANS procedure (ASC PKT) has been accomplished

If ORBIT config is G3 (OMS/RCS burns) or G8 (FCS C/O), substitute for G2 where appropriate. Major mode 301 assumed if no OPS 3

	Perform steps
ASCENT (G1)	a-h
ORBIT (G2,G8,G3/S2)	1-q

ASCENT

Initial ASCENT Config - Av Bay 1					
GPC	1	2	3	4	5
OPS	-	G1	G1	-	BFS1
POWER	OFF	ON	ON	OFF	ON
OUTPUT	NORM	NORM	NORM	NORM	BKUP
MODE	HALT	RUN	RUN	HALT	RUN
CONFIG	1				
GPC	12340				
STR 1	2				
2	3				
3	3				
4	2				
PL 1/2	2				
CRT 1	2				
2	2				
3	3				
4	0				
L 1	1				
2	2				
MM 1	2				
2	3				

Initial ASCENT Config - Av Bay 2					
GPC	1	2	3	4	5
OPS	G1	-	G1	G1	BFS
POWER	ON	OFF	ON	ON	OFF
OUTPUT	NORM	NORM	NORM	NORM	BKUP
MODE	RUN	HALT	RUN	RUN	HALT
CONFIG	1				
GPC	12340				
STR 1	1				
2	4				
3	3				
4	4				
PL 1/2	1				
CRT 1	1				
2	3				
3	4				
4	0				
L 1	1				
2	2				
MM 1	1				
2	3				

Initial ASCENT Config - Av Bay 3					
GPC	1	2	3	4	5
OPS	G1	G1	-	G1	BFS1
POWER	ON	ON	OFF	ON	ON
OUTPUT	NORM	NORM	NORM	NORM	BKUP
MODE	RUN	RUN	HALT	RUN	RUN
CONFIG	1				
GPC	12340				
STR 1	1				
2	2				
3	4				
4	4				
PL 1/2	1				
CRT 1	1				
2	2				
3	4				
4	0				
L 1	1				
2	2				
MM 1	1				
2	2				

a. FREEZE-DRY REASSIGNMENT..... SSR-5 ----->

- Freeze-dry GPC MODE - STBY (tb-bp), HALT
- PWR - OFF

b. Secure BFS (Av Bay 1,3 only)

- BFS CRT,GNC,OPS 000 PRO
- GPC 5 MODE - HALT (tb-bp)
- OUTPUT - NORM
- PWR - OFF
- BFC CRT DISP - OFF

	Av Bay 1	2	3
CONFIG	3	3	3
GPC	10000	02000	00300
STR 1	1	2	3
2	1	2	3
3	1	2	3
4	1	2	3
PL 1/2	0	0	0
CRT 1	1	2	3
2	1	2	3
3	1	2	3
4	0	0	0
L 1	0	0	0
2	0	0	0
MM 1	1	2	3
2	1	2	3

ALL VEH/OPS
FRP-7
BASIC REV A

Figure 4: A Failure Recovery Procedure (FRP)

MN BUS UNDERVOLTS/ FC VOLTS-AMPS

SM SYS SUMM 1

Failure confirmed by current and either voltage out of limits:

MN VOLTS	FC VOLTS	FC AMPS	ACTION
			SHORT or DEGRADED FC
LOW <26.4	LOW <26.6	HIGH Δ>50 or LOW Δ>50	If no MN BUS tied to affected FC MN BUS: If affected FC/MN BUS connected to P/L BUS: 1. Go to step 14 If affected FC/MN BUS not connected to P/L BUS: 2. Perform step 3 of affected FC SHUTDN, 5-8 (Cue Card), then: If FC VOLTS < 32.5 (FC Short): 3. Affected FC REAC VLV - CL If first FC failure: 4. Perform BUS TIE, 5-8 (Cue Card) 5. If all MN BUSES tied, MNC TIE BUS - OFF or 6. Go to PWRDN, LOSS of 1 FC/1 FREON LOOP, 10-18 >> If second FC failure: 7. Perform affected MN BUS LOSS ACTION, 5-15, then: 8. Go to PWRDN, LOSS of 2ND FC 10-20 >> If FC VOLTS > 32.5 (BUS Short): 9. Go to affected MN BUS LOSS ACTION, 5-15 >> If bus tied to affected FC/MN BUS: 10. Untie buses If short eliminated and MN BUS unpowered due to bus untie: or 11. Go to affected MN BUS LOSS ACTION, 5-15 >> If short not eliminated and MN BUS unpowered due to bus untie: 12. P/L PRI (three) - OFF 13. Perform BUS TIE, 5-8 (Cue Card) - good FC/MN BUS to unpowered bus, then: If affected FC/MN BUS connected to P/L BUS: 14. Disconnect P/L BUS from affected FC/MN BUS If short eliminated: or 15. Go to P/L BUS LOSS ACTION >> If short not eliminated: 16. Go to step 2 If affected FC/MN BUS not connected to P/L BUS: 17. Go to step 2

Figure 5: A Pocket Checklist Procedure

1.2.2 Development of Malfunction Procedures

Each malfunction procedure is the product of hundreds of man-hours of design, validation, and verification. Typically, a malfunction procedure is initially designed by one or more engineers. After the basic design has been determined, the procedure is then critiqued at a 'desktop' review where the malfunction procedures book manager, the author of the procedure and at least one crew member review the procedure and check its feasibility for use on the SSV. The next step is for the procedure to be tested by using both the Single System Trainer (SST) and the Shuttle Mission Simulator (SMS). Both the SST and the SMS provide a simulated SSV environment in which the procedure can be tested. In this way, simulated failures can be input into the system and crew and flight team response to the procedure can be evaluated through *integrated simulations*.

2 Automation of Malfunction Procedures

[1]The factors which influence a decision to automate a task performed by a human are straightforward and implicitly determined by human limitations and characteristics. Not surprisingly these factors include speed, accuracy, precision, complexity, and cost. While not all human activities lend themselves to automation, where an activity which does exists, as is the case with malfunction procedures, two fundamental questions must be addressed early in the process of automation. First, what types of automata are available for use, and second, how shall the information and data currently used by the human be represented for processing by those automata? Since we are only considering *programmable digital computers* or their equivalent in this work, the ensuing discussion will be limited to the selection and implementation of a methodology suitable for the representation, on this class of automata, of the procedural logic embodied in malfunction procedures.

2.1 Expert Systems and the Representation Problem

Although no precise definition of an expert system exists, a great deal has been written about them. A comprehensive review of the subject as well as the entire field of Artificial Intelligence can be found in [2]. Table 2 lists some of the more well-known expert systems with their particular application domain and references. According to [2],

> *[expert] systems are most strongly characterized by their use of large bodies of domain knowledge - facts and procedures, gleaned from human experts, that have proved useful for solving typical problems in their domain.*

Such systems differ from conventional programming in that they are intended as a general-purpose aid to problem-solving within a particular domain as opposed to being dedicated to a particular application within a domain. For example, an expert system might be used to determine which source

[1]Sections 2 - 2.5.1 are adapted from [15]

of tracking data is the most appropriate for use in updating a spacecraft state vector given the current ground and vehicle status. A conventional computer program could then be used to perform the computation from the source selected. A key problem in developing an effective expert system, as pointed out by [24],

> *is how to represent and use the knowledge that human experts in these subjects obviously possess and use. This problem is made more difficult by the fact that the expert knowledge in many important fields is often imprecise, uncertain, or anecdotal (though human experts use such knowledge to arrive at useful conclusions).*

Function	Domain	System	Reference
Diagnosis	Medicine	CASNET	[32]
	Medicine	INTERNIST	[26]
	Medicine	MYCIN	[28]
	Medicine	PUFF	[19]
	Engineering	SACON	[4]
	Geology	PROSPECTOR	[10]
Search	Chemistry	DENDRAL	[12]
	Chemistry	SYNCHEM	[14]
Problem Solving and Planning	Mechanics	MECHO	[6]
	Programming	PECOS	[3]
	Configuring Computers	R1	[21]
Measurement Interpretation	Medicine	VM	[11]
Computer-aided Instruction	Electronics	SOPHIE	[5]
	Medicine	GUIDON	[7]
Knowledge Acquisition	Diagnosis	TEIRESIAS	[9]
	Diagnosis	EMYCIN	[29]
	Diagnosis	EXPERT	[33]
System Building		ROSIE	[27]
		AGE	[23]
		HEARSAY III	[1]

Table 2: Some Existing Expert Systems

2.2 Rule-based Expert Systems

Rule-based systems are attractive because of their modularity, uniformity and ability to express human expert knowledge in a natural manner [2]. Expert knowledge can frequently be expressed in the form of IF-THEN relations; rule-based systems are designed to take advantage of this characteristic. In particular, IF-THEN information can be satisfactorily represented as *production rules*. Production rules are a computational formalism which can be used to define relationships among variables. The relationships are structured such that the satisfaction of preconditions, called *antecedents*, produce results, called *consequents*. Antecedents and consequents are generally expressed using symbolic variables and logical operators.

2.2.1 Production Rules and the Predicate Calculus

The use of production rules permits us to express some kinds of expert knowledge in a formal way and aids in the articulation of that knowledge. This formalism is also well-suited to expression in programming languages and in the *first-order predicate calculus* [24], [15]. The first-order predicate calculus is based on first-order logic (FOL) which, as the name implies, is a formal system of logic. Use of this system permits us to manipulate expert knowledge according to logical principles and to *obtain inferences* from known rules. A powerful result of this approach is the ability to form hypotheses (or theorems) and test (or prove) them as well as to ask questions about the validity and satisfiability of subsets of rules [24]. The representation of malfunction procedures, or for that matter any procedures, in the form of production rules is useful to us here for the following primary reasons:

1. It provides a standard, formal structure which is easily translated into programming languages. A familiar example of this is the IF-THEN syntax of FORTRAN although more sophisticated programming languages are generally used for implementation of rule-based systems.

2. It is compatible with the constructs of first-order logic and therefore first-order predicate calculus. These formalisms have desirable properties for the development of automatic, rule-based deduction systems.

This paper does not address the application of FOL to malfunction procedures other than to point out the suitability of the representation presented here to the techniques of FOL as well as some results from abstract algebra. Our primary purpose here is to establish the equivalence of a logical representation of procedural information to the existing representation of the same information on paper and to point out the advantages of the logical representation. The subsequent discussion will focus on some early work done at Johnson Space Center, and then discuss how procedural information can be expressed using Boolean functions and describe two methods of implementing this representation.

2.3 Early Work on Systems for Flight Control

A few prototype systems were developed to examine the feasibility of automating malfunction procedures. These systems provided a good basis for experimentation and gave those involved in the project good demonstration tools. The initial thrust was to pick different malfunction procedures from various SSV disciplines and, by coding them in Fortran 77, determine commonalities in format, syntax, logical structure, and so forth. The goal was to define a small family of logical operators with some type of logical array as operands. A minimum of one type of operator was envisioned for each of the four types of malfunction procedures. Initial work was begun in June, 1982 by analyzing the systems drawings of the SSV Pressure Control System (PCS) and constructing *functional loss* logic diagrams. By combining the drawing, logic diagram, and malfunction procedure for the PCS, EXPRES was born in October, 1982. The malfunction procedure provided the rule-based logic and interpretation of the drawing and diagram supplied the basis for an extensive query structure.

2.3.1 EXPRES

EXPRES served its purpose as a demonstration tool quite well, but had two major shortcomings. First, EXPRES acted as if the only inputs it had were provided by human operators, whereas, eventually a system was hoped for which could be linked to a data bus and thereby access vehicle telemetry directly. In other words, the human had to do a large amount of data checking in order to answer the questions posed by the procedure. A computer should be able to do the monitoring itself. Secondly, since it was a demonstration tool, EXPRES had the PCS malfunction procedure logic *procedurally* programmed or *hard-coded*. This meant that if another malfunction procedure were to be implemente the EXPRES program would require modification. This type of maintenance and modification is costly and error-prone as well as difficult to control.

2.3.2 CRYEX

In February of 1983, CRYEX, another demonstration expert system which used the cryogenic hydrogen pressure malfunction procedure. was designed to remove one of the shortcomings of EXPRES. CRYEX differed from EXPRES in that CRYEX allowed the user to define the symptoms of the problem prior to the program execution by changing the values of certain parameters. This feature made the program act as if the computer were interfaced with a data bus thereby accessing data directly from the vehicle. This showed that the computer could make many decisions without the assistance of a human. It still had the disadvantage of being *hard-coded*, however.

2.3.3 GENEX

As a response to the second criticism, GENEX was conceived. GENEX was built as a generic expert system operator. In other words, it was built to process any systems malfunction procedure. Benefitting from the experience gained from EXPRES and CRYEX, GENEX proved more efficient and shorter than the previous demonstration products but only partially realized its goal. The major problem remained that each procedure had to be uniquely coded; therefore generalizing the representation of the four types of procedures to reduce the complexity of the software was difficult.

2.3.4 Derived Requirements

Experience with EXPRES, CRYEX and GENEX resulted in the identification of a number of requirements for an expert system to be used to support the flight control team. The system should

1. be based on existing system diagrams, procedures and functional loss diagrams and, to the extent possible, be traceable to these source documents.

2. make maximal use of telemetry data to reduce operator interaction with the system.

3. not require the integral *hard-coding* of SSV subsystem logic.

2.4 Development of a Boolean Representation

To satisfy these requirements, the approach was developed that represented the procedures as sets of *Boolean functions*. Figure 6 depicts the relationship of the source document, the original malfunction procedure, to its Boolean equivalent. Because the procedures are generalized as Boolean functions, one can

1. apply the techniques of automata theory, switching theory and abstract algebra [30].

2. take advantage of two methods of implementation software and hardware, as depicted in figure 6.

Our discussion of the first point will be relatively limited because much of this work is still in progress [15]. With regard to the second point, not only does this approach provide a standard data structure it also provides the system architect the ability to selectively migrate processing between software and hardware *using a single common representation*, thereby offering tremendous flexibility in the design, development. testing and implementation of automated malfunction procedures. One of the malfunction procedures. CRYO 6.3a, has been implemented in both modes as a proof-of-concept demonstration.

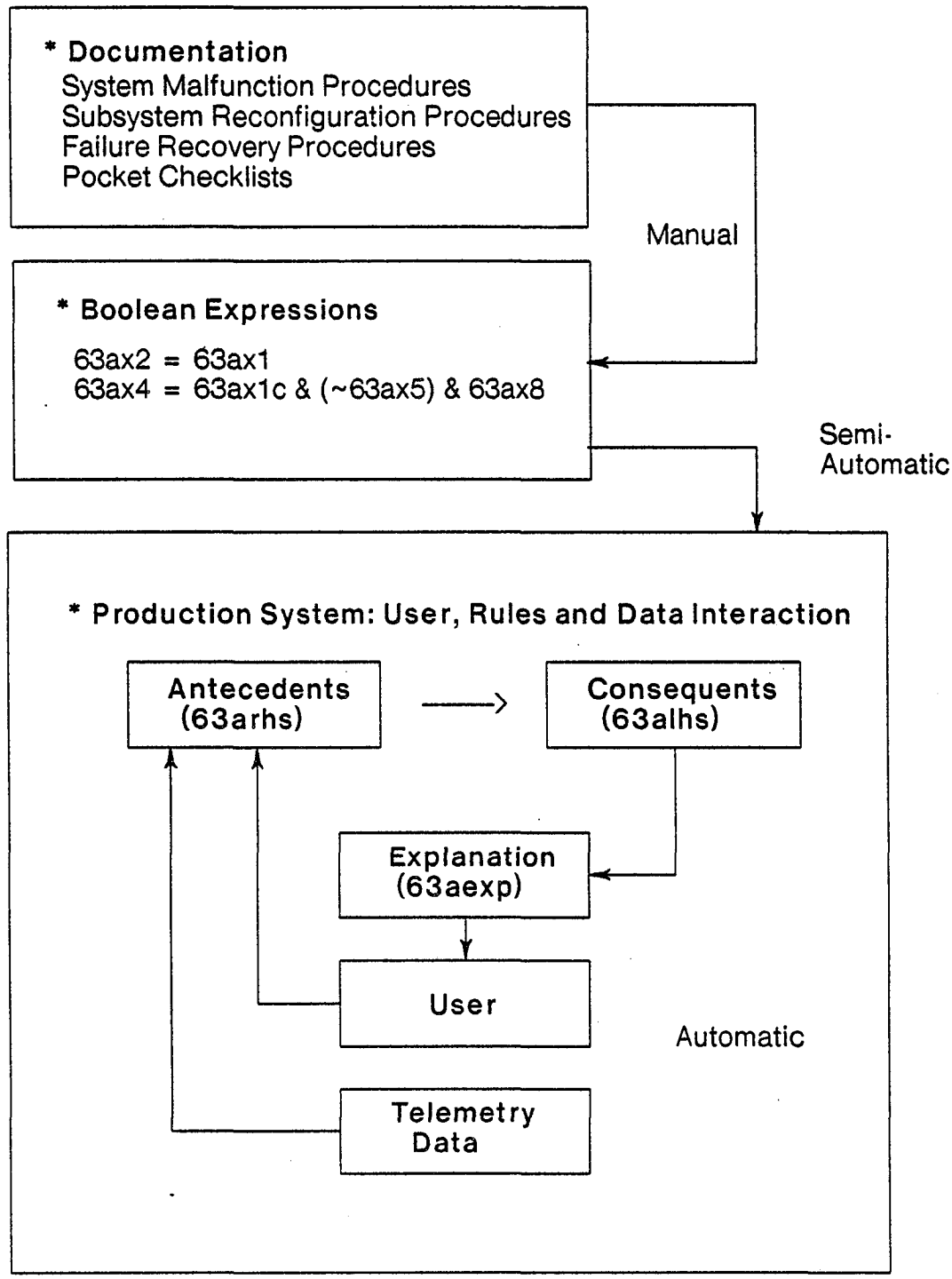

Figure 6: Equivalent Representations of Procedural Information

2.4.1 Malfunction Procedures as Graphs

The malfunction procedure itself can be thought of as a *directed graph* or *digraph*, $G=(V,E)$, where $V=\{v_1,v_2,....,v_k\}$, a set of vertices and E is a set of *arcs* such that each element of E is an ordered pair of vertices, (v_i,v_{i+1}). An arc from v_i to v_{i+1} can be denoted as $v_i \rightarrow v_{i+1}$ [16]. In this terminology the boxes in the malfunction procedure are *vertices*, and the connecting lines represent *arcs*. A *path* in the graph is a sequence of vertices $v_1,v_2,....,v_k$. $k>1$, such that an arc, $(v_i \rightarrow v_{i+1})$, exists for each i, $1 \leq i < k$. The path is said to be from v_i to v_k. For any vertex, v_i in a path, vertices v_j. $j<i$ are referred to as *predecessors* of v_i while vertices v_k. $k>i$ are the *successors* of v_i. The numbering of the vertices used here should not be confused with the numbers used to *name* the boxes in the malfunction procedure. The numbering of the vertices is used to indicate order; the numbering of the boxes is used as identification and not necessarily order. An interesting note is that *malfunction procedures are not trees*. As defined in [16], a *tree* is a digraph with the properties that:

1. There exists a vertex, the *root*, without predecessors, from which there is a path to every vertex.

2. Each vertex other than the root *has exactly one predecessor.*

3. The successors of each vertex are *ordered from the left.*

Malfunction procedures fail to satisfy properties two and three, as seen in figure 2. We introduce the concept of graphs here to permit us to be more precise in our subsequent discussion as well as to establish the groundwork for the application of other analytical methods.

2.4.2 Generation of Boolean Functions

The translation of block malfunction procedures to Boolean functions is straightforward. Failure Recovery Procedures (FRP) present a somewhat less direct translation but are nonetheless convertible to the representation described below for block malfunction procedures. As an example, consider the following expression:

$$63a9 = 63a1c \wedge \sim 63a3 \wedge \sim 63a5 \wedge 63a8 \qquad (1)$$

Referring to figure 2, note the heavily bordered box labelled with the number, 9, in the upper left-hand corner of the box. This box, as do others with the heavy black border, represents a *termination* or *diagnostic* state within the procedure. When a user of the procedure reaches one of these boxes by following the logic of the procedure, a conclusion about the state of the subsystem of interest has been reached.

2.4.3 Assignment of Variable Names

To represent the logic leading to the conclusion named *9* in figure 2 in machine-processable form we have assigned a Boolean (binary-valued) variable named *63a9* to the vertex labelled *9*. In the context of discussion in section 2.2 this variable is the *consequent* of the *antecedent* conditions on the right-hand side (RHS) of equation (1) in section 2.4.2. When the consequent has the value *1*, the proposition within the box is true, when it is *0*, the proposition is false. Since antecedent and consequent terms are assigned names in an identical manner we will present an example of naming only a single variable. Each vertex within a procedure is assigned a variable name composed of the procedure name, for example *63a*, and a numeric suffix which is the label of the vertex. This process can be summarized by the following production or *rewriting* rules.

[0] $S \rightarrow AABAC$

[1] $A \rightarrow 1|2|3|4|5|6|7|8|9$

[2] $B \rightarrow a|b|c|...|x|y|z$

[3] $C \rightarrow B|\lambda$

These rules specify a string of literals of length four or five. The first literal, represented by the first *A* in rule [0], indicates the chapter in the malfunction procedure handbook where the procedure can be found. The second *A* indicates the page in the chapter. The third literal,*B*, indicates the procedure within the SSV engineering discipline. The fourth literal,*A*, specifies the vertex containing the text of the proposition. Finally, the fifth literal, *C*, specifies subpropositions which may be posed in association with any single proposition as in the case of *63a1a, 63a1b, 63a1c* of figure 2. When no subpropositions are represented, this literal evaluates to *null* represented by λ. The application of these rules is illustrated below in the generation of the variable name *63a9*. The selective application of these rules at each step beginning with the start symbol, *S*, *rewrites* the literal string, *AABAC*, to the variable name *63a9*.

$S \rightarrow AABAC$ [0]

$AABAC \rightarrow 6ABAC$ [1]

$6ABAC \rightarrow 63BAC$ [1]

$63BAC \rightarrow 63aAC$ [2]

$63aAC \rightarrow 63a9C$ [1]

$63a9C \rightarrow 63a9$ [3]

The bracketed number after each step is the number of the rule applied at that step. The use of

rewriting rules provides a systematic method of assigning names to variables the number of which would rapidly become unmanageable if variable names were assigned in an *ad hoc* fashion. These conventions assure that each variable will have a unique name and that every reference to any given proposition (associated with a vertex) within a malfunction procedure will be made by the same name. This is especially important since there are cross-references to the same vertex between the malfunction procedures whereby one malfunction procedure branches into another.

2.4.4 Procedural Logic in Boolean Form

With a method of assigning variable names, capturing the logic of the procedure as a Boolean function is trivial. Realizing that *a Boolean function is equivalent to a path through the graph of the malfunction procedure* makes this immediately apparent. The number of antecedent terms in the resulting Boolean function is $k-1$ where k is the number of vertices in the path; the *path length* is $k-1$. Note that the consequent term of the function is an element of the path and contributes to the size of k. The *enumeration* of a set of functions which *covers* or *spans* the malfunction procedure can be accomplished by the simple recursive procedure described below.

```
GENERATE(FUNCTION);
  v ← NAME.OF(vertex);
  N ← GET.NUMBER.OF.OUTPUT.ARCS;
  if FUNCTION ~= nil then LOGIC-AND ← ' ∧ ';
  else LOGIC-AND ← nil;
  case 1:  if N = 0 then
     do;
       FUNCTION ← v || = || FUNCTION;
       output FUNCTION;
     end;
  case 2: else do until (N = 0);
    N ← N-1;
    if VALUE(arc_N) = 'yes' or nil then FUNCTION ←FUNCTION ||LOGIC-AND||v;
    if VALUE(arc_N) = 'no'  then FUNCTION←FUNCTION ||LOGIC-AND~||v;
    call NEXT.VERTEX;
    call GENERATE(FUNCTION);
   end;
return;
end GENERATE;
```

If this procedure is applied at the root of each malfunction procedure the result is a set of Boolean functions like that in Table 3. We assume the existence of some other functions to service the GET.NUMBER.OF.OUTPUT.ARCS, NAME.OF, VALUE. and NEXT.VERTEX calls. In particular, GET.NUMBER.OF.OUTPUT.ARCS returns *0*, *1*, or *2* for the various alternatives of *nil*, *yes*, or *yes* and *no*. NAME.OF assigns a variable name to the vertex consistent with the rules of section 2.4.3, NEXT.VERTEX follows the arc $v_i \rightarrow v_{i+1}$ to the next vertex. This procedure is executed manually at present although it could be implemented on a computer to support the automated design of malfunction procedures.

$63a2 = 63a1a$
$63a4 = 63a1c \wedge 63a3$
$63a6 = 63a1c \wedge \sim 63a3 \wedge 63a5$
$63a7 = 63a6$
$63a9 = 63a1c \wedge \sim 63a3 \wedge \sim 63a5 \wedge 63a8$
$63a10 = 63a9$
$63a14 = 63a13$
$63a15 = 63a1c \wedge \sim 63a3 \wedge \sim 63a5 \wedge \sim 63a8 \wedge \sim 63a11 \wedge 63a12 \wedge 63a14$
$63a18 = 63a1c \wedge \sim 63a3 \wedge \sim 63a5 \wedge \sim 63a8 \wedge 63a11 \wedge 63a16 \wedge 63a17$
$63a19 = 63a1c \wedge \sim 63a3 \wedge \sim 63a5 \wedge \sim 63a8 \wedge \sim 63a11 \wedge 63a16$
$63a20 = 63a1c \wedge \sim 63a3 \wedge \sim 63a5 \wedge \sim 63a8 \wedge 63a11 \wedge 63a16 \wedge \sim 63a17$
$63a21 = 63a19$
$63a21 = 63a20$
$63a24 = 63a22 \wedge \sim 63a14 \wedge 63a12 \wedge \sim 63a11 \wedge \sim 63a8 \wedge \sim 63a5 \wedge \sim 63a3 \wedge 63a1c$
$63a25 = \sim 63a22 \wedge \sim 63a14 \wedge 63a12 \wedge \sim 63a11 \wedge \sim 63a8 \wedge \sim 63a5 \wedge$
$\sim 63a3 \wedge 63a1c$
$63a27 = 63a26 \wedge 63a23 \wedge 63a1b$
$63a28 = 63a25$
$63a28 = 63a15$
$63a30 = 63a29 \wedge \sim 63a26 \wedge 63a23 \wedge 63a1b$
$63a33 = \sim 63a32 \wedge 63a31 \wedge \sim 63a29 \wedge \sim 63a26 \wedge 63a23 \wedge 63a1b$
$63a35 = 63a32 \wedge 63a31 \wedge \sim 63a29 \wedge \sim 63a26 \wedge 63a23 \wedge 63a1b$
$63a36 = 63a34b \wedge \sim 63a31 \wedge \sim 63a29 \wedge \sim 63a26 \wedge 63a23 \wedge 63a1b$
$63a37 = 63a30$
$63a37 = 63a33$
$63a37 = 63a35$
$63a37 = 63a38$
$63a38 = 63a34a \wedge \sim 63a31 \wedge \sim 63a29 \wedge \sim 63a26 \wedge 63a23 \wedge 63a1b$
$63a39 = 63a36$
$63a42 = 63a41 \wedge 63a40 \wedge \sim 63a23 \wedge 63a1b$
$63a43 = \sim 63a41 \wedge 63a40 \wedge \sim 63a23 \wedge 63a1b$
$63a44 = 63a42$
$63a44 = 63a47$
$63a47 = 63a46 \wedge 63a45 \wedge \sim 63a40 \wedge \sim 63a23 \wedge 63a1b$
$63a47 = 63a46 \wedge 63a45 \wedge 63b8$
$63a48 = \sim 63a46 \wedge 63a45 \wedge \sim 63a40 \wedge \sim 63a23 \wedge 63a1b$
$63a48 = \sim 63a46 \wedge 63a45 \wedge 63b8$
$63a50 = 63a49b \wedge \sim 63a45 \wedge \sim 63a40 \wedge \sim 63a23 \wedge 63a1b$
$63a50 = 63a49b \wedge \sim 63a45 \wedge 63b8$
$63a51 = 63a50$
$63a52 = 63a49a \wedge \sim 63a45 \wedge \sim 63a40 \wedge \sim 63a23 \wedge 63a1b$
$63a52 = 63a49a \wedge \sim 63a45 \wedge 63bx8$
$63a53 = 63a52$

Table 3: CRYO 6.3a (figure 2) as Boolean Expressions

The conversion of the malfunction procedure to a set of Boolean expressions is equivalent to the *enumeration* of the possible paths through the graph. While enumeration problems are associated with classes of problems considered *intractable* [13] we are not faced. here, with the full enumeration

problem since the set of paths for conversion to Boolean form is small and well-defined by virtue of having been written down. This is the same as saying that the *search space* of the algorithm of 2.4.4 is small.

2.5 Software Implementation

Using the approach to variable naming and logic representation described above, a selected subset of malfunction procedures was implemented in a convenient high-level language (HLL), ROSIE (Rule-oriented System for Implementing Expertise) [27]. As used in this work, ROSIE is implemented on a VAX 11/780 running the UNIX operating system. It should be noted that ROSIE was selected primarily as a matter of convenience and that this work does not require its use. Any programming language could be used to implement the methods presented here. The principal advantages of ROSIE are the interactive nature of the system, the existing system utilities for string manipulation, and the relational structure of the underlying database system. These features present a powerful development environment. The principal disadvantage is the large system overhead associated with the interpretive ROSIE language which is itself based on a dialect of LISP, INTERLISP-D. The dependency on INTERLISP-D also limits the variety of systems on which ROSIE may be hosted at present.

2.5.1 Organizing Principle

To retain the terminology found in existing malfunction procedures and, at the same time, retain the Boolean representation common to both the software and hardware implementations, the information contained in malfunction procedures was conveniently organized into three components. This decomposition was suggested by the form of the Boolean functions as shown in figure 6. As an example, for the malfunction procedure, CRYO 6.3a, the rule-sets contain the following three components:

1. **63arhs** - Antecedents. Contains the logic necessary to determine the binary value of the consequents of the Boolean expression. In general, this represents information which must be requested from the flight controller or the flight crew. The value of these Boolean variables are potentially ascertainable from telemetry data. Grouping the variables into this category essentially constructs the set of *data which must be obtained from some source external* to the malfunction procdure itself.

2. **63alhs** - Consequents. Contains the inferential logic which determines the binary value of the *consequents* as a function of the antecedents. Consequents are associated with either or both

    a. **diagnosis** - the detection of a condition associated with the heavy, black bordered boxes displayed in the original malfunction procedure document reproduced in figure 6.

b. **action** - the detection of a condition associated with a box in the original procedure which requires human intervention or a change of vehicle state necessary to provide further information or ensure vehicle safety before executing the rest of the procedure.

3. **63aexp** - Explanation. Contains the logic to interpret the values of the Boolean variables into English text for presentation to the flight controllers or crew. This category uses exactly the wording of the original malfunction procedure itself. This ability is a **major** advantage in eliminating problems associated with introducing new terminology to a highly specialized application.

This decomposition provides a strong organizational structure or paradigm for converting these rule-sets into compiled, high-level languages such as FORTRAN or PASCAL as well as interpreted and compilable languages like LISP. Tables 4, 5, 6 show the form each of these categories takes when constructed as ROSIE rule-sets.

Table 4: Partial Listing of *63aRHS*

```
[4]  if H2 Press is true obtain 63a1a of "H2 P Normal".
[5]  if 63a1a is true go 63a1hs.
[6]  if 63a1a is false obtain 63a1b of "H2 P High".
[7]  if 63a1b is true obtain 63a23 of "TK3 and/or TK4 the affected tk".
[8]  if 63a1b is false obtain 63a1c of "H2 P Low".
```

Table 5: Partial Listing of *63aLHS*

```
[1] if (63a1a is true) assert 63a2 is true.
[2] if (63a1c is true and 63a3 is true) assert 63a4 is true.
[3] if (63a1c is true and 63a3 is false and 63a5 is true)
       assert 63a6 is true.
[4] if (63a6 is true) assert 63a7 is true.
```

Table 6: Partial Listing of *63aEXP*

```
[8] if 63a1a is true  send {"* All H2 P norm",return}.
[9] if 63a1b is true  send {"* H2 P High",return,
    "* ACTION: Deact htrs in affected tk(s).",return,
    "  (R1)  H2   TK1(2,3) HTRS   A,B - OFF and/or",return,
    "  (A11) CRYO TK4      HTR H2 A,B - OFF",RETURN}.
[10] if 63a1c is true  send {"* H2 P low",return}.
[11] if 63a2  is true  send {"* DIAGNOSIS: C/W Failure",return}.
```

2.6 Hardware Implementation

The reason that we discuss the topic of hardware implementation at all is to point out the concept of *hardware-equivalence*. By virtue of the use of Boolean formalism we are able to represent the complete logic of the malfunction procedure as if it were a *programmable logic array* (PLA). This means that procedural logic contained in the malfunction procedures can be transformed to a *bit-map* which is independent of the method or language of implementation. Since the bit-map is essentially a truth-table with a standard structure, described below, a single processor or program can be constructed which processes these tables. By adopting this representation we eliminate the need for a large software system which would be necessary to capture and process the logic of a large number of malfunction procedures and permit the processing of the same procedural logic at the level of register-register operations. This implies not only significantly reduced storage requirements but also very high processing speeds. The next few sections summarize the steps necessary to convert Boolean expressions to their PLA form and present the results of an actual PLA synthesis.

2.6.1 Generation of Hardware Descriptions

Once the malfunction procedure has been specified in Boolean form, the process of creating descriptions of hardware which are functionally equivalent to the malfunction procedure is straightforward. A set of Boolean equations can be translated into integrated circuits through an automated series of translations between different intermediate representations. Each of the representations is a description of a different aspect of the implementation process. In this case, the Boolean equations will be converted into a truth table which is sometimes referred to as a *personality matrix*. The truth table will be translated into an architecture known as an AND-OR programmable logic array (AND-OR PLA). (In the remainder of this discussion, the term PLA will be used to mean AND-OR PLA.) The PLA can be implemented as a custom integrated circuit. Figure 7 summarizes the sequence of steps leading from the Boolean functions to the manufacture of the hardware device. Most of these steps are implemented using programs developed in the University of California, Berkeley, Computer Science Division [20]. The names of the major programs involved are listed in figure 7 according to their role in the sequential design and manufacture of a VSLI device. Although one can actually generate a semiconductor device which executes the logic of the malfunction procedure, as shown below, to do so probably is impractical for most of the kinds of procedures currently existing, due to the volatile nature of these procedures and the consequent frequent changes made to them. While the redesign of these devices would be trivial, the manufacture and integration of the resultant devices would not be generally cost-effective. A small subset of highly stable procedures could possibly be identified for hardware implementation.

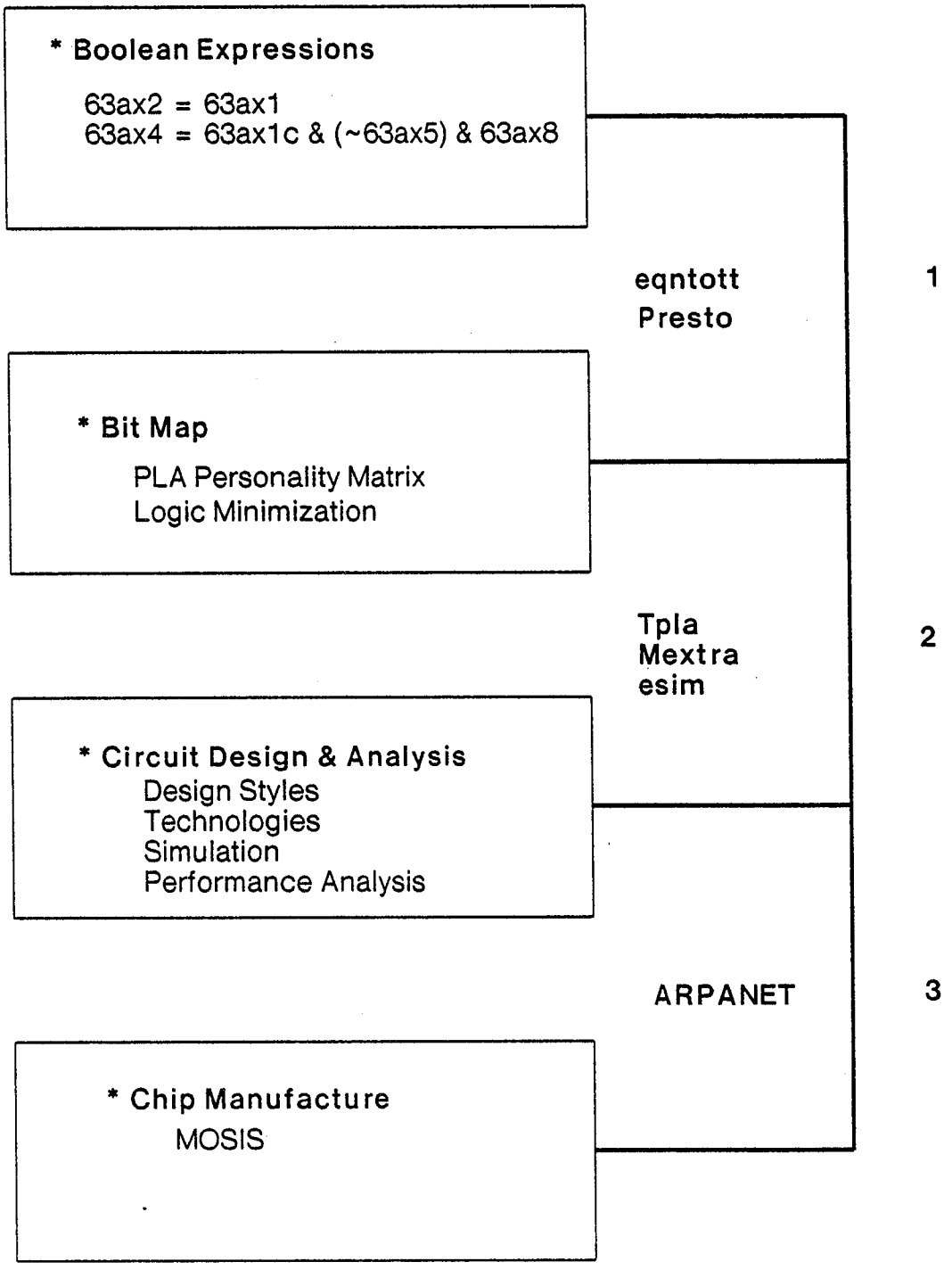

Figure 7: Design and Manufacture Sequence for VLSI Devices

2.6.2 Reduction of Boolean Functions to Normal Form

The design automation program *eqntott* generates a personality matrix suitable for PLA programming from a set of Boolean functions that define the PLA outputs in terms of its inputs. Table 3 displays the input to *eqntott* while figure 8 shows the resulting *personality matrix* and the PLA resulting from that matrix as described in section 2.6.4. A personality matrix is a representation of a set of Boolean functions in *disjunctive normal form* (DNF) that defines a template for a circuit implementation of those functions.

2.6.3 Minterms, Truth-tables and PLAs

Disjunctive normal form expresses Boolean functions as *sums of products* or *minterms*. The interested reader is directed to [31] for further discussion of normal forms for Boolean functions. One method of organizing information contained in Boolean functions in general and functions in DNF in particular is through the use of a truth-table. The mathematical term for a row in a truth table is a *minterm*. A truth table is an enumeration of the output values of a set of Boolean equations for a given set of input values. For example, the truth table for the Boolean function $a \wedge (b \vee c)$ is

```
a b c | out
0 0 0 |  0
0 0 1 |  0
0 1 0 |  0
0 1 1 |  0
1 0 0 |  0
1 0 1 |  1
1 1 0 |  1
1 1 1 |  1
```

Each minterm is the conjunction (logical AND) of one or more terms. The second row of this truth table is a minterm whose value is 0 if a is 0 and b is 0 and c is 1. A truth table can be reduced in size by the use of *don't care* terms. A *don't care* term is represented by '-' in the truth table. One possible truth table for $a \wedge (b \vee c)$ using *don't care* terms is

```
a b c | out
0 - - |  0
1 0 0 |  0
1 1 - |  1
1 - 1 |  1
```

The truth table representation and the PLA architecture were chosen for their close similarities.

2.6.4 Interpretation of the Personality Matrix

The personality matrix consists of a line for each sum of products term, *implicant*, which begins with that implicant followed by the values of the various outputs. The implicant is composed of a single character (0, 1, or -) for each input variable in the conventional fashion descibed in section 2.6.3. The output values are represented by one of three characters (0, 1, or x). The PLA architecture is physically similar to the truth table, in addition to implementing it functionally. The AND-OR PLA has

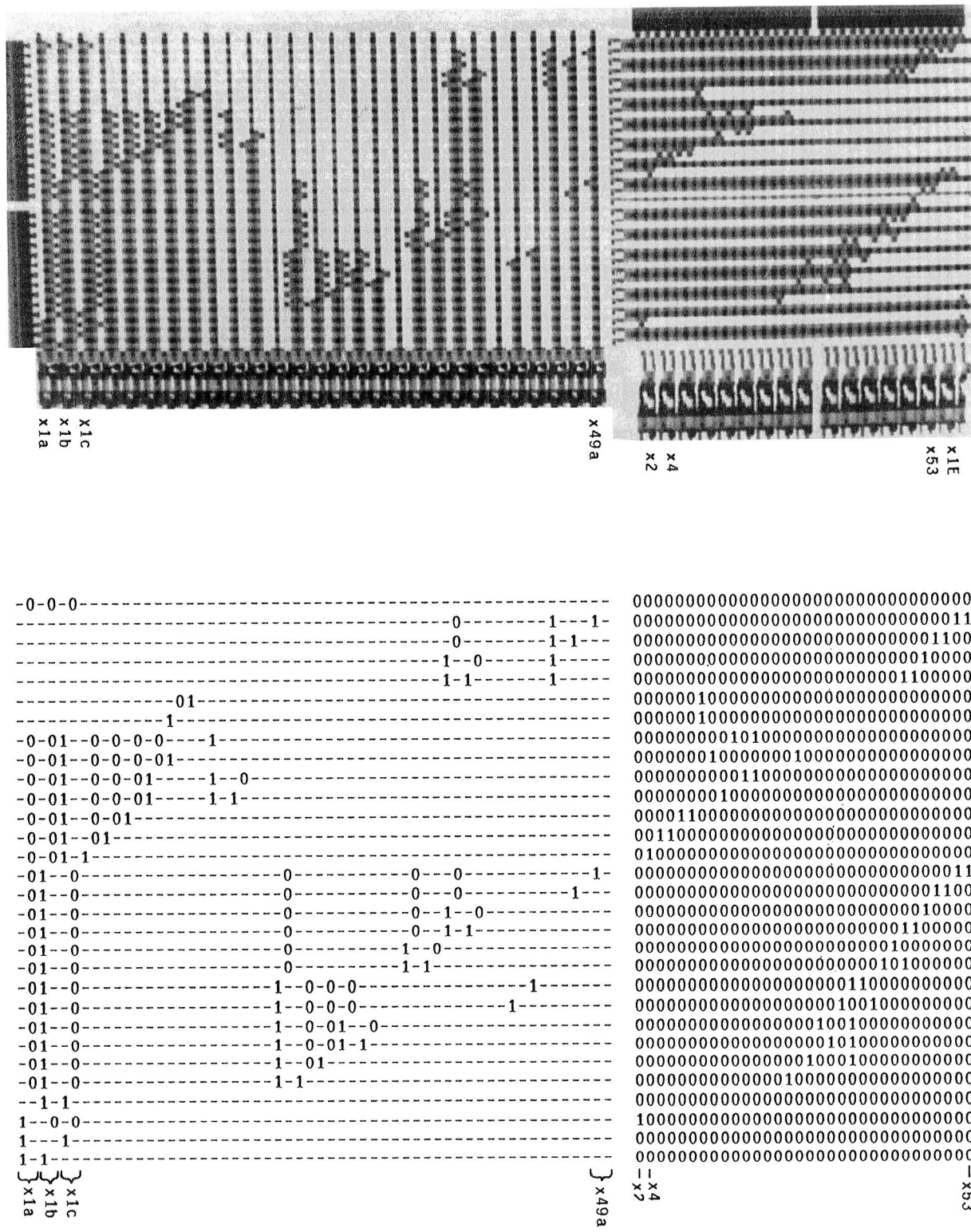

Figure 8: Disjunctive Normal Form Matrix and Corresponding Circuit Topology

two parts that are each implemented as planar portions of the (planar) integrated circuit. These portions are called the and-plane and the or-plane. The and-plane lies adjacent to the or-plane in the same way that the input portion of a truth table lies adjacent to the output portion. The and-plane and or-plane have the same number of rows. Each row contains the circuitry to calculate one of the minterms from the truth table.

2.6.5 True and Complement Format

Each pair of columns in the AND-plane contributes the true and complement of an input variable to each of the appropriate minterms. A 1 contributes the true form of the input variable, a 0 contributes the complement of the input variable. A minterm is calculated by anding the appropriate form of each input variable that is selected by a 1 or a 0 in that row. An - in the truth table indicates that the input variable does not contribute anything to the minterm. For example, the third row in the reduced truth table above indicates that the third minterm is calculated by anding the true form of *a* with the true form of *b* and that neither the true nor complement of *c* is used. Each column in the OR-plane specifies an output variable. The output from a column is the logical OR of 0 with each minterm that is selected by the presence of a one in that column.

2.6.6 Comparing Logical and Physical Domain Formats

The reduction, by *eqntott*, of the set of Boolean functions into disjunctive normal form creates a matrix (bit-map) describing the normal form which is directly comparable to the physical realization of the PLA in terms of integrated circuit mask layers. To illustrate this, figure 8 displays both the truth-table for malfunction procedure CRYO 6.3a and the resulting PLA. They have been lined-up to show the data format of the disjunctive normal form of the truth-table juxtaposed with the corresponding circuit topology of the PLA. The ones in the left-hand part of the matrix are reflected in a connection to the true input column in the circuit, the zeros in the left-hand part of the matrix are reflected in a connection to the complemented input column in the circuit, and the ones in the right-hand part of the matrix are reflected in a connection to the output column in the circuit. The zeroes in the right-hand part of the matrix specifies no connection to the physical output column in the circuit.

2.6.7 PLA Performance Analysis

The circuit description can be analyzed for area, timing, and power dissipation characteristics, and can be simulated, with *esim*; see figure 7, as a further validation of the conversion from procedural to Boolean form.

cryo6.eqn			
Design Style	Area (square microns)	Power Dissipation (W)	Worst Delay (ns)
nMOS PLA (cis)	1176480	.115 (.065)	29.8
nMOS PLA (trans)	1247688	.115 (.065)	29.8
CMOS PLA (cis)	5031180	N/A	N/A
CMOS PLA (trans)	4374832	N/A	N/A

Figure 9: Automated Analysis of Implementation Characteristics of *CRYO 6.3a* Equations.

3 Distributed Architecture

Previous sections have concentrated on the representation and implementation of procedural logic using new approaches and new technology. Greater benefits may be realized from these techniques than from simple machine processing. We have demonstrated the equivalence of hardware representations to the logical representation of the original malfunction procedure. Based on these discussions the benefits such an approach may have for the increasing distribution of mission processing between space and ground as well as between hardware and software, are apparent.

3.0.1 Characteristics of Distributed Architectures

Distributed processing architectures have general characterstics which can be seen to be lacking from the current SSV ADP architecture, although most of these characteristics can be found in the operations network as a whole. Discussion here is limited to the characteristics of the ADP architecture for mission support. As discussed in [8], these include:

- processing supported by a *network* which provides high-level control over inter-process communication in a standard, network-wide protocol.
- communication strictly between asynchronous processes as opposed to mere remote data access.
- data access accomplished via inter-process communication.
- resource sharing not limited to data-sharing and especially including sharing of processing in support of a single task.

3.1 Limitations of the Current Communications Architecture

In the past, unless a space vehicle was in *view* of a ground-based tracking station, which was only 15-20% of the time during an orbit for manned space flights, space/ground communication was not possible. During the time when spacecraft were *blacked-out* of communication with the ground, all of the data which the craft may have been collecting regarding its on-board systems, experiments or external sensors had to be stored on-board. Only when communication was re-established could data be communicated (transmited/received) with the ground. This situation is changing inasmuch as the Tracking and Data Relay Satellite System (TDRSS) is anticipated, when fully operational, to permit communication with the ground for 80-85% of a given flight.

3.2 Limitations of Current On-board Processing Architecture

Historically, the tasks performed in support of manned space flight have been distributed between the on-board *General Purpose Computers (GPCs)* and the ground computers. This physical separation was also a logical separation of processing responsibility, largely the result of the limited space-ground communication. Tighter logical sharing of processing was restricted not merely by the physical separation of the processors but fundamentally by the isolation resulting from low transmission speeds and irregular opportunity for communication due to limited ground coverage. This approach has been perpetuated in the SSV data processing systems. Within the current system architecture, the on-board General Purpose Computers (GPCs) perform these major functions [17]:

1. guidance, navigation and control of the SSV through ascent, on-orbit, and landing,

2. systems management which includes software to acquire, process and route data for systems evaluation and management,

3. payload software which permits the modification of the contents of mass memory units (MMUs), and loading of the software to support display electronics units (DEUs).

The software to support these functions must be completely prepared and loaded on the MMU prior to launch. There is extremely limited variation of system processing once a launch has been executed. Alteration of the MMU processing sequence can be accomplished only through the use of single commands to the processor in the form of GPC instructions or data. These commands must be built manually by either the flight crew or the controllers on the ground. For the purposes of malfunction procedure processing, the GPCs act only as the interface between the flight team and the vehicle systems and subsystems. The GPCs themselves do not have the capacity to support extensive anomaly processing.

3.3 Effect of Limitations on SSV Autonomy

These two factors, limited communications and limited on-board processing, apply opposing forces in attempts to achieve increased SSV autonomy. On the one hand limited communications makes it desirable to provide as much stand-alone capability on-board as possible. On the other hand, the desire for high reliability of the SSV systems, in the face of high SSV system complexity, requires the expertise of more than just the flight crew for *both* nominal and off-nominal operations. The addition of computers and software to support increased vehicle autonomy is not simple. Since the existing computer systems are *flight-critical*, the alteration of their operational capabilities is an expensive undertaking, and also subject to high schedule risk due to the nature of software development. This is not, however, an impossible or even impractical goal. One method for additional processing on-board, without requiring the alteration of existing on-board systems, is through the use of *carry-on micrcomputers*. If we assume that the problem of additional processing is solvable, and it is, still to be faced is the more difficult problem of capturing the collective expertise of the flight control team in a computing system. The improvement in space-ground communications represented by TDRSS makes this expertise more accessible to the flight crew but does change the very labor intensive character of mission monitoring *nor does it alone increase vehicle autonomy.*

3.4 Options for System Architecture and Interconnections

The ability to select the mode of implementation, software or hardware, as well as the location, space or ground, offers four options for any given system design attempting to the flight control function. These are indicated in figure 10. *The choices are not mutually exclusive*. This fact implies the slightly more subtle point, however, that the methods presented in this paper make it possible and relatively simple task incorporate significant increases in redundancy and autonomy for procedural processing, with all the attendant benefits and without great cost. This can be accomplished with a single logical representation and in an automated fashion. The importance of these results can be seen by envisioning a spacecraft which, when normally in contact with the ground, is monitored by an automatic malfunction processing system staffed by humans but which is capable of *going autonomous* for the same function when, for any reason, it is out of communication with the ground.

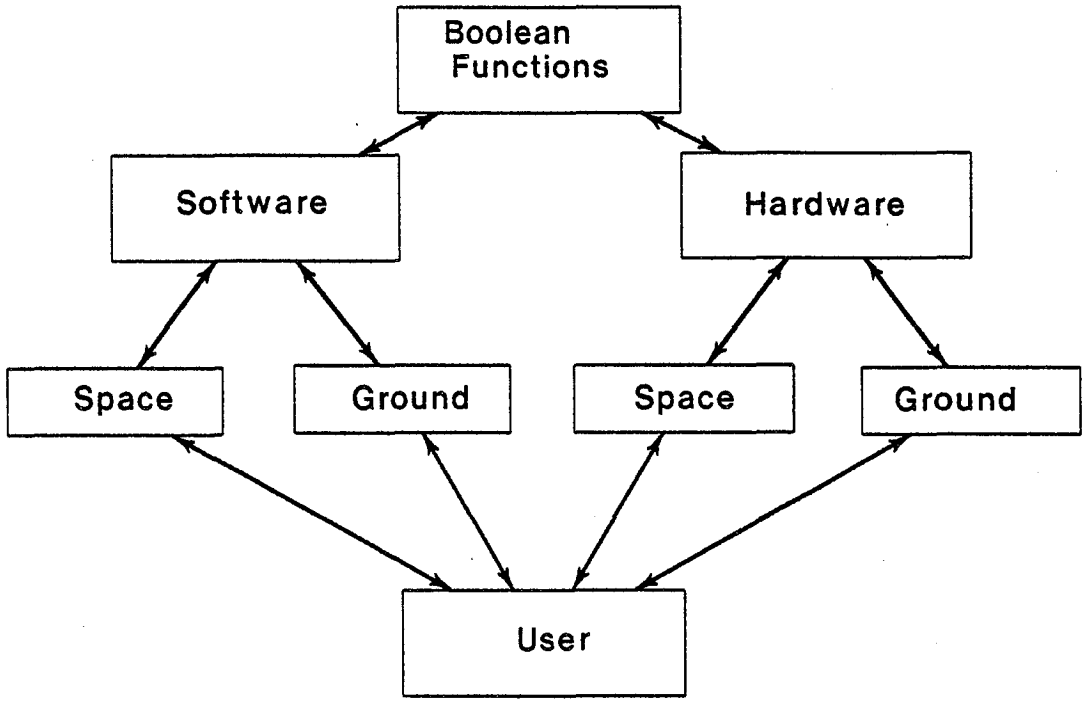

Figure 10: Distributed Processing Architecture: Software and Hardware

4 Summary

We have presented a new method for the representation of SSV malfunction procedures which permit their conversion to a form suitable for processing by computers. These results are derived from [15] which originated this representation for the development of a rule-based expert system capable of automatic inference and the implementation of higher-level logic than that contained in the malfunction procedures. Additionally we have presented the concept of *hardware equivalence* and discussed the implications of the distribution of procedural processing between hardware and software as well as space and ground. *The results of this work stand-alone in that this representation can be implemented based soley on the results presented in this paper and independent of any further work to develop an expert system based on them.*

References

[1] Balzer, R., Erman, L. D., London, P. and Williams, C.
HEARSAY-III: A Domain-Independent Framework for Expert Systems.
In *Proc. First Ann. Conf. on Artificial Intelligence*, pages 108-110. 1980.

[2] Barr, A. and Feigenbaum, E. eds.
The Handbook of Artificial Intelligence.
HeurisTech Press, 1981.

[3] Barstow, D. R.
An Experiment in Knowledge-based Automatic Programming.
Artificial Intelligence 12:7-119, August, 1979.

[4] Bennett, J. S., Engelmore, R. S.
SACON: A Knowledge-Based Consultant for Structural Analysis.
In *Proc. of Sixth Intl. Joint Conf. on Artificial Intell.*, pages 47-49. August, 1979.

[5] Brown, J. S., Burton, R. R., Bell, A. G.
SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An Example of AI in CAI).
Technical Report F41609-73-C-006, Bolt, Beranek, and Neuman Inc., 1974.

[6] Bundy, A. et al.
Solving Mechanics Problems Using Meta-Level Inference.
In D. Michie (editor), *Expert Systems in the Microelectronic Age*, . Edinburgh University Press, 1979.

[7] Clancey, W. J., Shortliffe, E. H., Buchanan, B. G.
Intelligent Computer-Aided Instruction for Medical Diagnosis.
In *Proc. of 3rd Symp. on Computer Application in Medical Care*. 1979.

[8] Davies, D. W.
Applying the RSA Digital Signature to Electronic Mail.
Computer :55-62, February, 1983.

[9] Davis, R.
Interactive Transfer of Expertise: Acquisition of New Inference Rules.
Artificial Intelligence 12:121-157, August, 1979.

[10] Duda, R., J. G. Gaschnig, and P. E. Hart.
Model Design in the Prospector Consultant System for Mineral Exploration.
In D. Michie (editor), *Expert Systems in the Microelectronic Age*, . Edinburgh University Press, 1979.

[11] Fagan, L., J. Kunz, E. A. Feigenbaum and J. Osborn.
Representation of Dynamic Clinical Knowledge: Measurement Interpretation in the Intensive Care Unit.
In *Proc. of the Sixth Conf. on Artificial Intelligence*. 1979.

[12] Feigenbaum, E. A., B. G. Buchanan, and J. Lederberg.
On Generality an Problem Solving: A Case Study Using The DENDRAL Program.
In B. Meltzer and D. Michie (editor), *Machine Intelligence 6*, . American Elsevier, 1971.

[13] Garey, M., Johnson, D.
Mathematical Sciences: Computers and Intractability. A Guide to the Theory of NP-Completeness.
Freeman, San Francisco, 1979.

[14] Gelernter, J. L. et al.
Empirical Explorations of SYNCHEM.
Science :1041-1049, September, 1977.

[15] Helly, J. J., Jr.
A Distributed Expert System for Space Shuttle Flight Control.
PhD thesis, University of California, Los Angeles, in preparation.

[16] Hopcroft, J., Ullman, J.
Computer Science: Introduction to Automata Theory, Languages, and Computation.
Addison-Wesley, Reading, Massachusetts, 1979.

[17] Suffredini, M.
Data Processing System Overview Workbook
DPS OV 2102 Advanced Training Series edition, NASA Flight Training Branch, NASA JSC, Houston, TX 77058, January 1984.

[18] NASA.
System Malfunction Procedures Requirements
Final, Rev. B edition, Flight Operations Division, Lyndon B. Johnson Space Center, Houston, TX 77058, June 15, 1982.

[19] Kunz, J. et al.
A physiological rule-based system for interpreting pulmonary function test results..
Technical Report HPP-78-19, Stanford University, 1978.

[20] Robert N. Mayo, John K. Ousterhout, and Walter S. Scott, editors.
1983 VLSI Tools.
Technical Report UCB/CSD 83/115, University of California, Berkeley, Computer Science Division (EECS), March, 1983.

[21] McDermott, J.
R1: An Expert in the Computer Systems Domain.
In *Proc. of the First National Conf. on Artificial Intelligence.* 1980.

[22] Flight Operations Directorate: Crew Training and Procedures Division.
JSC-12770 Shuttle Flight Operations Manual
NASA Johnson Space Center, 1978.
Preliminary.

[23] Nii, H. P., and N. Aiello.
AGE (Attempt to Generalize): A knowledge-based program for building knowledge-based programs.
IJCAI :645-655, 1979.

[24] Nilsson, N.
Principles of Artificial Intelligence.
Tioga, 1980.

[25] L. F. Pau.
Control and Systems Theory. Volume 11: *Failure Diagnosis and Performance Monitoring*.
Marcel Dekker, Inc., New York, 1981.

[26] Pople, H.
The formation of composite hypotheses in diagnostic problemsolving-an exercise in synthetic reasoning.
IJCAI :1030-1037, 1977.

[27] J. Fain, D. Gorlin, F. Hayes-Roth, S. Rosenschein, H. Sowiziral, D. Waterman.
The ROSIE Language Reference Manual
RAND Corporation, 1981.

[28] Shortliffe, E. H.
Computer-based medical consultations: MYCIN.
American-Elsevier, New York, 1976.

[29] Shortliffe, E. H.
Computer-Based Medical Consultations: Mycin.
Elsevier, 1976.

[30] T. L. Booth.
International Series in Applied Mathematics: Sequential Machines and Automata Theory.
John Wiley and Sons, Inc., New York, New York, 1967.

[31] R. Thomas.
Kinetic Logic. A Boolean Approach to the Analysis of Complex Regulatory Systems.
In *Lecture Notes in Biomathematics*. European Molecular Biology Organization, 1979.

[32] Weiss, S. M., C. A. Kulikowski, A. Safir.
A model-based consultation system for the long-term management o glaucoma.
IJCAI :826-832, 1977.

[33] Weiss, S. M., C. A. Kulikowski.
EXPERT: A system for developing consultation nodes.
IJCAI :642-947, 1979.

★ U.S. GOVERNMENT PRINTING OFFICE: 1984—769-013/6281

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
TM 58258		

4. Title and Subtitle	5. Report Date
A representational basis for the development of a distributed expert system for Space Shuttle flight control	May 1984
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
John J. Helly, Jr. UCLA, Aerospace Corp. William V. Bates JSC Mel Cutler, Steven Kelem Aerospace Corp.	S-533
	10. Work Unit No. 561-85-00-00-72

9. Performing Organization Name and Address	11. Contract or Grant No.
NASA Lyndon B. Johnson Space Center Houston, TX 77058	

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics & Space Administration Washington, D. C. 20546	Technical Memorandum
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

A new representation of malfunction procedure logic which permits the automation of these procedures using Boolean normal forms is presented. This representation is discussed in the context of the development of an expert system for Space Shuttle flight control including software and hardware implementation modes, and a distributed architecture. The roles and responsibility of the flight control team as well as previous work toward the development of expert systems for flight control support at Johnson Space Center are discussed. The notion of malfunction procedures as graphs is introduced as well as the concept of hardware-equivalence.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Malfunctions Flight control Space Shuttles Computer Systems Programs Hardware System failures Distribution (Property) Architecture (Computers) Failure	unclassified- unlimited subject category: 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
unclassified	unclassified	38	

*For sale by the National Technical Information Service, Springfield, Virginia 22161