A DISTRIBUTED EXPERT SYSTEM FOR SPACE SHUTTLE FLIGHT CONTROL

John Joseph Helly, Jr.

1984
Report No. CSD-840038

UNIVERSITY OF CALIFORNIA

Los Angeles

A Distributed Expert System for Space Shuttle Flight Control

A dissertation submitted in partial satisfaction of the requirements for the

degree of Doctor of Philosophy in Computer Science

by

John Joseph Helly, Jr.

1984

DEDICATION

This dissertation is dedicated to the memory of my parents, Mae and Jack, and my grandmother, Mrs. Margaret McDermott.

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

I wish to acknowledge some of the support and encouragement which made it possible to complete this work. In particular, I wish to thank Dr. John S. Stephens, Jr., who taught me to be a biologist, and who has provided example and inspiration for hundreds of students at the Moore Lab; Dr. Ed Deland, who helped me mature in my thinking about science and it applications and without whom this degree would never have been completed. I would like to thank Dr. Steve Crocker and the Information Science Research Organization at the Aerospace Corporation for technical assistance and computer services; Mr. William V. Bates at the NASA Johnson Space Center, and Capt. L. Dianne Langmade for help in understanding the systems, and operations of the Space Shuttle Vehicle and for their critical evaluation of the utility of the methods presented here. Finally, I wish to thank the members of my committee, especially my Chairman and Graduate Advisor, Dr. Jacques Vidal, for guidance and for their comments and review of this dissertation.

VITA

August 5, 1952 -- Born, Elizabeth, New Jersey

1976-78 -- M.S. University of California, Los Angeles.

1972-75 -- B.A., M.A., Occidental College, Los Angeles, California

PUBLICATIONS

[Helly 76] Helly, J. J., Jr.
Effects of temperature and thermal distribution on glycolysis in two rockfish species.
Marine Biology 37:89-95, 1976.

[Helly 80] Deland, E., J. J. Helly, Jr.
Compartmental physiological models with chemical reactions.
In *Proceedings of IMACS*, pages 333-338.
IMACS, Sorrento, Italy, 1980.

[Helly 81a] Follette, D., K. Fey, G. Buckberg, J. J. Helly, Jr.
Reducing postischemic damage by temporary mmodification of reperfusate calcium, potassium, pH and osmolarity.
J. Thoracic and Cardiovascular Surgery 82(2):221-238, 1981.

[Helly 81b] Helly, J. J., Jr., E. Deland
Complex system modeling with statistical methods.
Winter Simulation Conference Proceedings 82(2):611-615, 1981

ABSTRACT OF THE DISSERTATION

A Distributed Expert System for Space Shuttle Flight Control

by

John Joseph Helly, Jr.

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1984

Professor Jacques Vidal, Chair

A new representation of malfunction procedure logic, based on the programmable-logic array, which facilitates the automation of these procedures for real-time detection and analysis of spacecraft anomalies using Boolean normal forms is presented. Processing of anomalies makes explicit use of time to permit non-monotonic reasoning in the form of partial system resets corresponding to recovery of system functions through operator intervention. A methodology for the synthesis of new rules is developed which permits the modification of the rule-base both dynamically, in real-time as part of the systems monitoring function, and statically, as part of a systems analysis. A distributed architecture for expert systems based on an extension of the programmable-logic array is introduced . This representation is discussed in the context of Space Shuttle systems and has general applicability to the control and monitoring of complex systems.

Chapter 1
Introduction

This work presents the results of the design and preliminary development of a *rule-based expert* system, which introduces novel results in the field of expert systems in terms of its unique *rule-base structure*, which represents knowledge in the form of *compiled implications*; the *explicit* incorporation of time in its set of active rules; the utilization of *temporal, non-monotonic reasoning*; and the provision of a means of *generating new knowledge* through the identification of increasingly complex *failure classes* from empirical observations and its existing rule-base. This research was originally motivated by the need to automate some of Space Shuttle Vehicle (SSV) flight control team functions to aid in reducing the labor-intensive nature of space operations. It was quickly realized that it also afforded a research opportunity to explore expert systems problems in a particularly well-structured problem domain. The need for *good problem domains* has been recently identified as an important feature for basic AI research [McCarthy83 83]. Although developed specifically with the SSV flight control problem in mind, this approach is applicable to the larger class of problems addressed by control and feedback systems for the purpose of *failure diagnosis and performance monitoring* [Pau 81].

1.1. Organization

This work is presented in four chapters. Chapter 1 is principally introductory material describing the problem domain, the goals of the work reported here and providing an overview of other, relevant work in the field of expert systems. Chapter 2 describes the structure of the rule-base and particularly addresses the development of the data representation used as a basis rule-based, automatic detection and resolution of anomalies occurring during SSV flight operations. Chapter 3 presents the techniques for implementing an operational system, capable of functioning in *real-time* especially addressing the introduction of linear time into the rule-base as well as the problems associated with that capability. Chapter 4 discusses the adaptation of the *time-dependent system* introduced in Chapter to the problem of generating *meta-rules* as models of *complex failure modes*. The identification of such new modes is considered to be an example of automatic learning. Chapter 5 discusses the issues surrounding the use of this methodology in a distributed environment which includes not only the conventional notion of distribution among processors but also the distribution of function between space and ground.

1.2. Rule-based Expert Systems

While there is little agreement as to what Artificial Intelligence is [Bundy83 83], it would be generally conceded that research in *expert systems* has been one of the most exciting of its subspecialties. It is, however, also true that no precise definition of an expert system exists, although 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 [Barr 81]. According to [Barr 81],

> *[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 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. Table 1-1 lists some well-known expert systems as well as some obscure but more recent systems with their particular application domain. These references reflect the diversity of application domains found in expert system research topics.

Function	Domain	System	Reference
Diagnosis	Medicine	CASNET	[Weiss 77]
	Medicine	INTERNIST	[Pople 77]
	Medicine	MYCIN	[Shortliffe 76a]
	Medicine	PUFF	[Kunz 78]
	Engineering	SACON	[Bennett 79]
	Geology	PROSPECTOR	[Duda 79]
Search	Chemistry	DENDRAL	[Feigenbaum 71]
	Chemistry	SYNCHEM	[Gelernter 77]
Problem Solving and Planning	Mechanics	MECHO	[Bundy 79]
	Programming	PECOS	[Barstow 79]
	Configuring Computers	R1	[McDermott 80]
		DEVISER	[Vere 83]
	Procedures	REF-ARF	[Fikes70 70]
Measurement Interpretation	Medicine	VM	[Fagan 79]
Computer-aided Instruction	Electronics	SOPHIE	[Brown 74]
	Medicine	GUIDON	[Clancey 79]
	Medicine	FLUIDMOD	[Deland 74]
Knowledge Acquisition	Diagnosis	TEIRESIAS	[Davis 79]
	Diagnosis	EMYCIN	[Shortliffe 76b]
	Diagnosis	EXPERT	[Weiss 79]
	Diagnosis	SEEK	[Politakis 84]
System Building		ROSIE	[ROSIE 81]
		AGE	[Nii 79]
		HEARSAY III	[Balzer 80]
		XPLAIN	[Swartout 83]
Temporal References	General Robots	CHRONOS	[Bruce 72] [Hendrix 73]

Table 1-1: Some Existing Expert Systems

1.2.1. General Features

Research in this area has resulted in the generalization of some of the major features or properties of expert systems. Rule-based systems are a type of the larger class of *Pattern-Directed Inference Systems* (PDIS's) characterized by the separation of *data examination*, corresponding to antecedent terms of a rule, and *data modification*, corresponding to the consequent terms of a rule [Waterman 78]. One of the unique features of rule-based systems is that they are primarily *event or data-driven* as opposed to procedurally-driven. The description of the behavior of rule-based systems in terms of a *recognize/act cycle* [McDermott78a 78] emphasizes both the data examination and data modification aspects of such systems. This characteristic is the direct result of the representation of knowledge in *declarative* form. This is uniquely the domain of AI research as pointed out by [Nilsson83 83] in the remark that

> There are component pieces, however, for which AI has no competition. These pieces are at the core of what might be called *high-level* reasoning and perception. this core is mainly concerned with the collection, representation, and use of *propositional* or *declarative* knowledge. (Such knowledge is of the type that can be stated in sentences of some form, as contrasted, say, with knowledge that is implicit only in procedures or in ad hoc data structures.)

The rules, which declare knowledge, are compared to data to determine via the pattern of the antecedent terms, which rules apply to the observed data. The identification of such a *conflict set* [McDermott78a 78] is then resolved by some *decision or control process* to determine which of the consequents holds for the set of data. Rule-based systems are attractive because of their modularity, uniformity and ability to express human expert knowledge in a natural manner [Barr 81]. 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.

1.2.2. Production Systems

Production system is an ambiguous term in the literature used to indicate that a *system*, whatever it may be, is based on *production rules*. In general, a production system makes use of a *production memory* and a *working memory* [McDermott78a 78]. The working memory has also been referred to as *short-term memory* [Zisman 78]. Essentially the production memory is what we refer to as the *rule-base*. In the particular implementation presented in this work the rule-base consists of *implication tables* or *implication relations* (cf. section 2.5.4). Through some form of pattern-matching, rules stored in production memory are copied into working memory as part of the *active set of rules* which, presumably, reflect the current state of the system from which the data are derived.

1.2.3. Advantages and Disadvantages of Production Systems

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* [Nilsson 80]. The first-order predicate calculus is based on first-order propositional 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 [Nilsson 80]. This is not to say that production systems are without their critics. These limitations are well known, however, and have been previously noted by some of the principle proponents of *logic programming*. For example, [Kowalski79 79] points out such limitations, though not in great detail, in the context of Horn clauses versus non-Horn clauses. [Nilsson83 83] raises some of these issues in the context of current challenges to AI research posed by Minsky's *dead duck* challenge which also overlaps the problems of *non-monotonic reasoning* [Waldrop 84]. In spite of its particular limitations, the use of propositional logic has proved to be a powerful and general method for the purpose of implementing human knowledge in machine-processable form.

1.2.4. Suitability to the Flight Control Problem

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.

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.

1.3. Flight Control Team

As described in [NASA 78], 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 1-2 provides an abbreviated list of activities for which the flight control team is responsible for during an SSV mission.

Table 1-2: 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

1.3.1. Organizational Structure

The SSV flight control team is organized as a **hierarchy** as depicted in figure 1-1. 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.

1.3.2. 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 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 [Pau 81], to maximize system availability, unjustified system shutdown or *pulldowns* must also be minimized.

1.3.3. Analytical Tools for Flight Control

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.

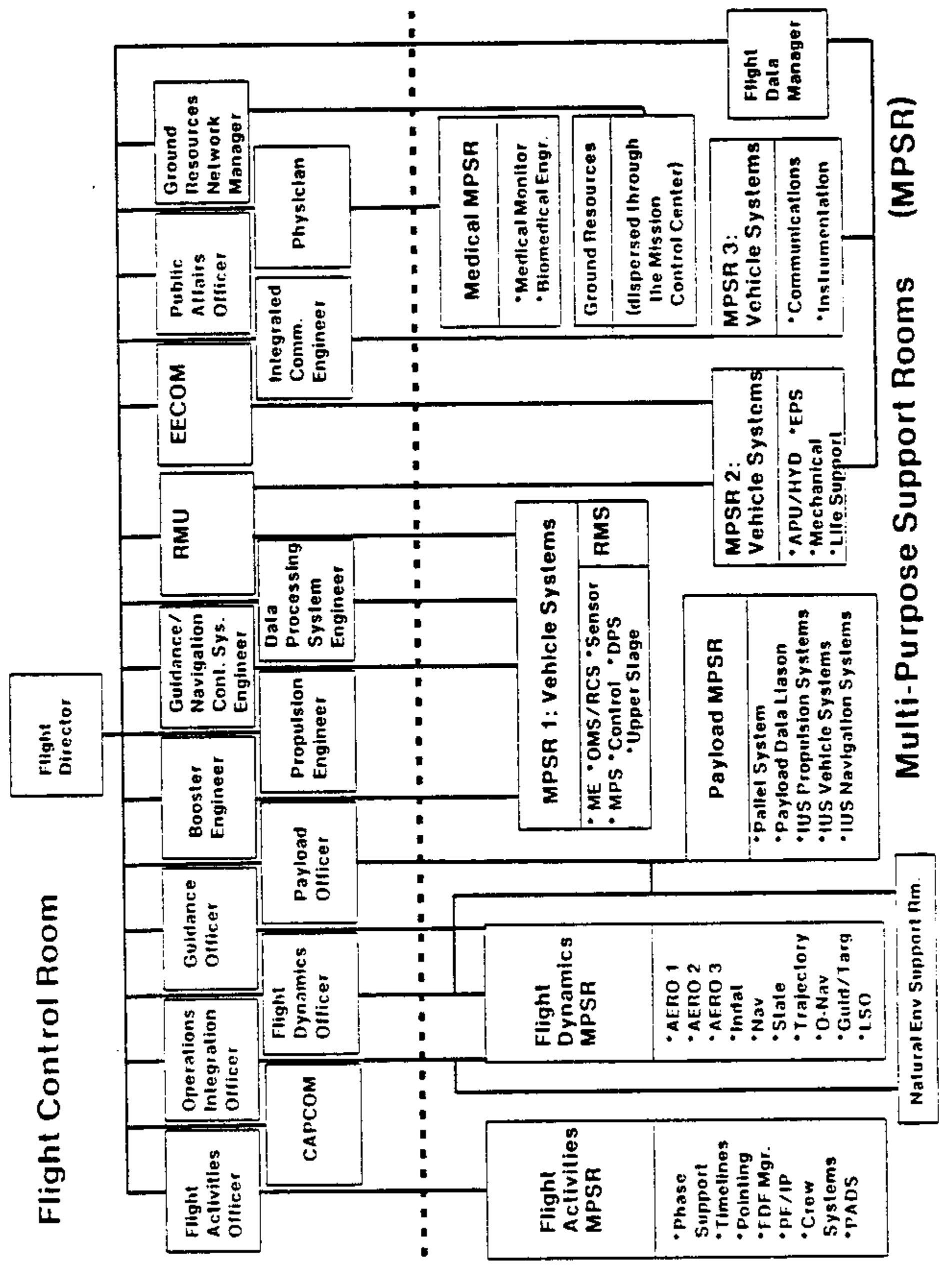

Flight Control Room
Flight Director
Flight Activities Officer
Operations Integration Officer
Guidance Officer
Booster Engineer
Guidance/Navigation Cont. Sys. Engineer
RMU
EECOM
Public Affairs Officer
Ground Resources Network Manager
CAPCOM
Flight Dynamics Officer
Payload Officer
Propulsion Engineer
Data Processing System Engineer
Integrated Comm. Engineer
Physician
Flight Activities MPSR
•Phase Support •Timelines •Pointing •FDF Mgr. •PF/IP •Crew Systems •PADS
Flight Dynamics MPSR
•AERO 1 •AERO 2 •AERO 3 •Inrtal •Nav •State •Trajectory •O-Nav •Guid/Targ •LSO
Natural Env Support Rm.
Payload MPSR
•Pallet System •Payload Data Liason •IUS Propulsion Systems •IUS Vehicle Systems •IUS Navigation Systems
MPSR 1: Vehicle Systems
RMS
•ME •OMS/RCS •Sensor •MPS •Control •DPS •Upper Stage
MPSR 2: Vehicle Systems
•APU/HYD •EPS •Mechanical •Life Support
MPSR 3: Vehicle Systems
•Communications •Instrumentation
Medical MPSR
•Medical Monitor •Biomedical Engr.
Ground Resources
(dispersed through the Mission Control Center)
Flight Data Manager
Multi-Purpose Support Rooms (MPSR)

Figure 1-1: Flight Control Team Organization

1.3.4. 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.

1.3.4.1. Types

Four basic types of malfunction procedures are used by the flight teams [JSCMALFS 82]. 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 1-2 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 1-3 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 1-4 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 1-5 is a representative pocket checklist procedure.

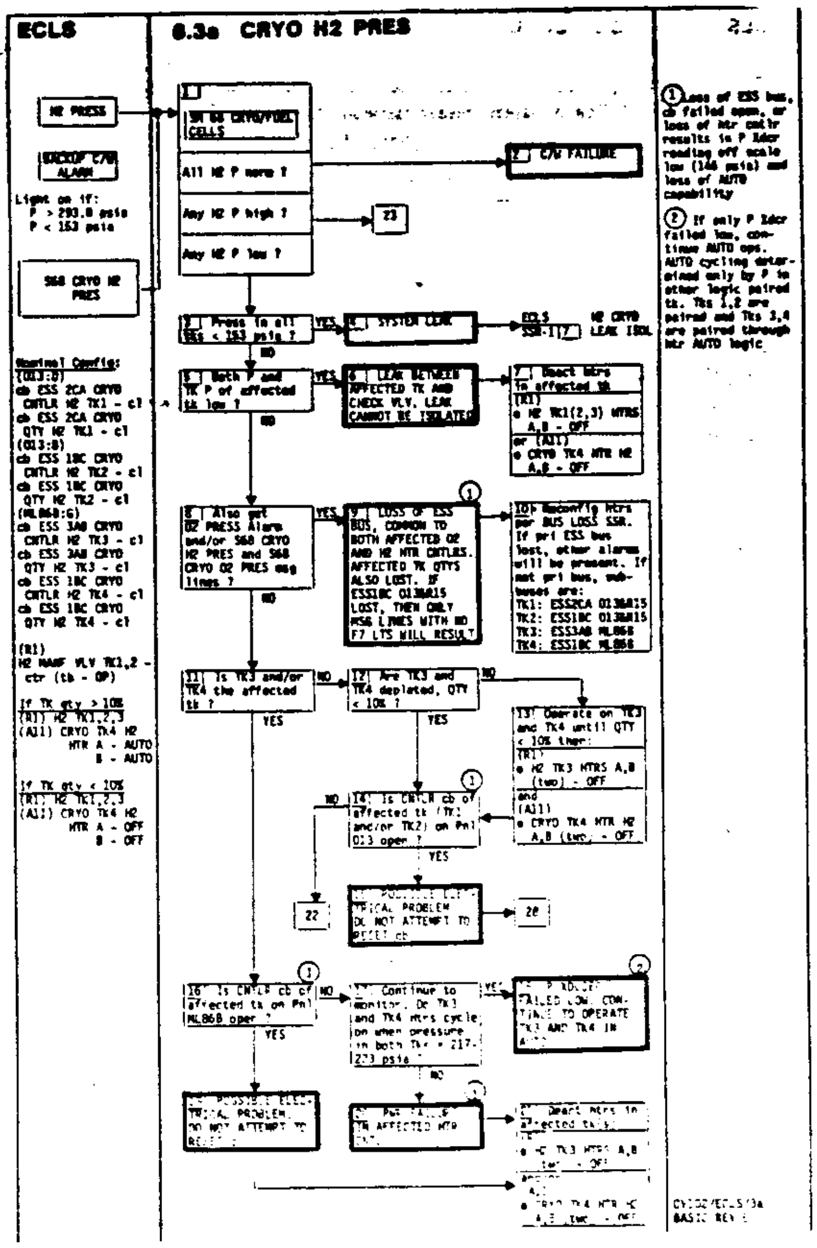

ECLS
6.3a CRYO H2 PRES
H2 PRESS
BACKUP C/W ALARM
Light on if: P > 293.8 psia P < 153 psia
S68 CRYO H2 PRES
Nominal Config: (O13:B) cb ESS 2CA CRYO CNTLR H2 TK1 - cl cb ESS 2CA CRYO QTY H2 TK1 - cl (O13:B) cb ESS 1BC CRYO CNTLR H2 TK2 - cl cb ESS 1BC CRYO QTY H2 TK2 - cl (ML86B:G) cb ESS 3AB CRYO CNTLR H2 TK3 - cl cb ESS 3AB CRYO QTY H2 TK3 - cl cb ESS 1BC CRYO CNTLR H2 TK4 - cl cb ESS 1BC CRYO QTY H2 TK4 - cl
(R1) H2 MANF VLV TK1,2 - ctr (tb - OP)
If TK qty > 10% (R1) H2 TK1,2,3 (A11) CRYO TK4 H2 HTR A - AUTO B - AUTO
If TK qty < 10% (R1) H2 TK1,2,3 (A11) CRYO TK4 H2 HTR A - OFF B - OFF
1 SM 68 CRYO/FUEL CELLS
All H2 P same ?
Any H2 P high ?
Any H2 P low ?
2 C/W FAILURE
23
3 Press in all TKs < 153 psia ?
YES
NO
4 SYSTEM LEAK
ECLS SSR-1 7 H2 CRYO LEAK ISOL
5 Both P and TK P of affected tk low ?
6 LEAK BETWEEN AFFECTED TK AND CHECK VLV. LEAK CANNOT BE ISOLATED
7 Deact htrs in affected tk (R1) • H2 TK1(2,3) HTRS A,B - OFF or (A11) • CRYO TK4 HTR H2 A,B - OFF
8 Also get O2 PRESS Alarm and/or S68 CRYO H2 PRES and S68 CRYO O2 PRES msg lines ?
9 LOSS OF ESS BUS, COMMON TO BOTH AFFECTED O2 AND H2 HTR CNTLRS. AFFECTED TK QTYS ALSO LOST. IF ESS1BC O13A15 LOST, THEN ONLY MSG LINES WITH NO F7 LTS WILL RESULT
10 Reconfig htrs per BUS LOSS SSR. If pri ESS bus lost, other alarms will be present. If not pri bus, sub-buses are: TK1: ESS2CA O13A15 TK2: ESS1BC O13A15 TK3: ESS3AB ML86B TK4: ESS1BC ML86B
11 Is TK3 and/or TK4 the affected tk ?
12 Are TK3 and TK4 depleted, QTY < 10% ?
13 Operate on TK3 and TK4 until QTY < 10% then: (R1) • H2 TK3 HTRS A,B (two) - OFF and (A11) • CRYO TK4 HTR H2 A,B (two) - OFF
14 Is CNTLR cb of affected tk (TK1 and/or TK2) on Pnl O13 open ?
22
POSSIBLE ELECTRICAL PROBLEM DO NOT ATTEMPT TO RESET cb
20
16 Is CNTLR cb of affected tk on Pnl ML86B open ?
17 Continue to monitor. Do TK3 and TK4 htrs cycle on when pressure in both Tks = 217-223 psia ?
P XDUCER FAILED LOW. CONTINUE TO OPERATE TK3 AND TK4 IN AUTO
POSSIBLE ELECTRICAL PROBLEM. DO NOT ATTEMPT TO RESET
PWR FAILURE IN AFFECTED HTR CNTLR
Deact htrs in affected tk(s)
• H2 TK3 HTRS A,B (two) - OFF
and/or (A11)
• CRYO TK4 HTR H2 A,B (two) - OFF
1 Loss of ESS bus, cb failed open, or loss of htr cntlr results in P Xdcr reading off scale low (144 psia) and loss of AUTO capability
2 If only P Xdcr failed low, continue AUTO ops. AUTO cycling determined only by P in other logic paired tk. Tks 1,2 are paired and Tks 3,4 are paired through htr AUTO logic
CY102/ECLS/3a BASIC REV

Figure 1-2: A Block Malfunction Procedure

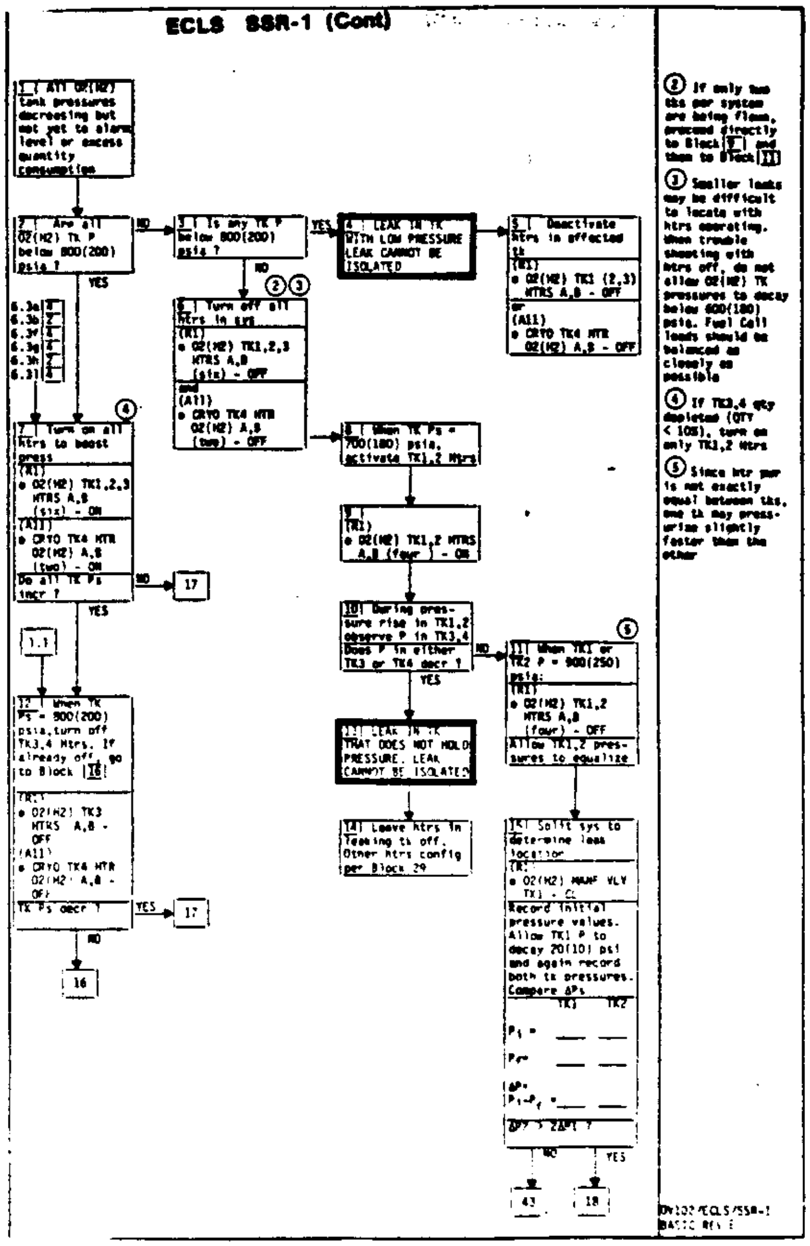
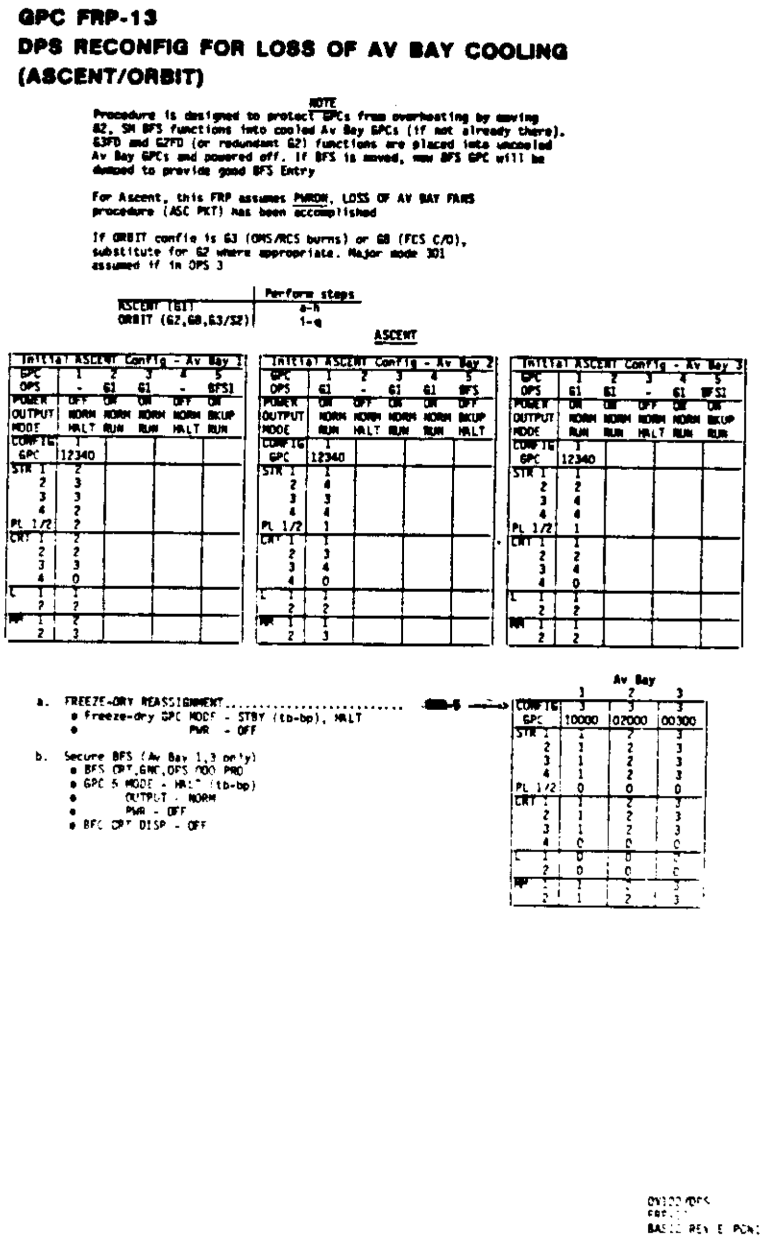

ECLS SSR-1 (Cont)
1 All O2(H2) tank pressures decreasing but not yet to alarm level or excess quantity consumption
2 Are all O2(H2) TK P below 800(200) psia ?
NO
YES
3 Is any TK P below 800(200) psia ?
YES
NO
4 LEAK IN TK WITH LOW PRESSURE LEAK CANNOT BE ISOLATED
5 Deactivate htrs in affected tk
(R1)
● O2(H2) TK1 (2,3) HTRS A,B - OFF
or
(A11)
● CRYO TK4 HTR O2(H2) A,B - OFF
6.3a 4
6.3b 2
6.3f 4
6.3g 4
6.3h 2
6.3i 4
6 Turn off all htrs in sys
(R1)
● O2(H2) TK1,2,3 HTRS A,B (six) - OFF
and
(A11)
● CRYO TK4 HTR O2(H2) A,B (two) - OFF
7 Turn on all htrs to boost press
(R1)
● O2(H2) TK1,2,3 HTRS A,B (six) - ON
(A11)
● CRYO TK4 HTR O2(H2) A,B (two) - ON
Do all TK Ps incr ?
NO
17
YES
8 When TK Ps = 700(180) psia, activate TK1,2 Htrs
9
(R1)
● O2(H2) TK1,2 HTRS A,B (four) - ON
10 During pressure rise in TK1,2 observe P in TK3,4
Does P in either TK3 or TK4 decr ?
NO
YES
11 When TK1 or TK2 P = 900(250) psia:
(R1)
● O2(H2) TK1,2 HTRS A,B (four) - OFF
Allow TK1,2 pressures to equalize
1.1
12 When TK Ps = 900(200) psia, turn off TK3,4 Htrs. If already off, go to Block 16
(R1)
● O2(H2) TK3 HTRS A,B - OFF
(A11)
● CRYO TK4 HTR O2(H2) A,B - OFF
TK Ps decr ?
YES
17
NO
16
13 LEAK IN TK THAT DOES NOT HOLD PRESSURE. LEAK CANNOT BE ISOLATED
14 Leave htrs in leaking tk off. Other htrs config per Block 29
15 Split sys to determine leak location
(R1)
● O2(H2) MANF VLV TK1 - CL
Record initial pressure values. Allow TK1 P to decay 20(10) psi and again record both tk pressures. Compare ΔPs
TK1 TK2
Pi = ___ ___
Pf = ___ ___
ΔP = Pi-Pf = ___ ___
ΔP2 > 2ΔP1 ?
NO
YES
43
18
② If only two tks per system are being flown, proceed directly to Block 7 and then to Block 11
③ Smaller leaks may be difficult to locate with htrs operating. When trouble shooting with htrs off, do not allow O2(H2) TK pressures to decay below 600(180) psia. Fuel Cell loads should be balanced as closely as possible
④ If TK3,4 qty depleted (QTY < 10%), turn on only TK1,2 Htrs
⑤ Since htr pwr is not exactly equal between tks, one tk may pressurize slightly faster than the other
OV102/ECLS/SSR-1
BASIC REV E

Figure 1-3: A Special Subroutine (SSR)



GPC FRP-13

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 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 in 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				

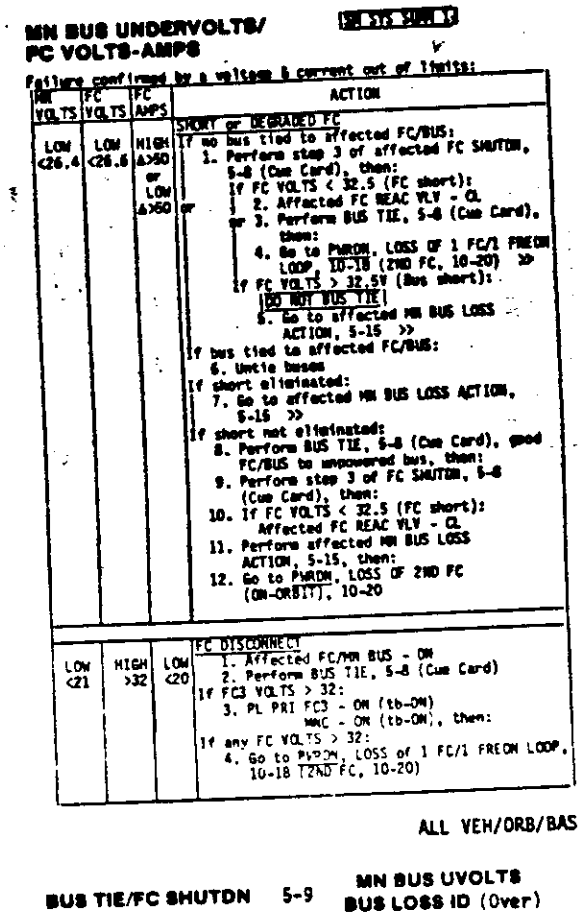
a. FREEZE-DRY REASSIGNMENT........................... [illegible]-5 →
- Freeze-dry GPC MODE - STBY (tb-bp), HALT
- PWR - OFF

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	[illegible]
2	0	0	0
MM 1	1	[illegible]	3
2	1	2	3

OV102/DPS
FRP-[illegible]
BASIC REV E PCN[illegible]

Figure 1-4: A Failure Recovery Procedure (FRP)



MN BUS UNDERVOLTS/
FC VOLTS-AMPS SM SYS SUMM 1

Failure confirmed by a voltage & current out of limits:

MN VOLTS	FC VOLTS	FC AMPS	ACTION
LOW <26.4	LOW <26.6	HIGH Δ>50 or LOW Δ>50	SHORT or DEGRADED FC If no bus tied to affected FC/BUS: 1. Perform step 3 of affected FC SHUTDN, 5-8 (Cue Card), then: If FC VOLTS < 32.5 (FC short): 2. Affected FC REAC VLV - CL or 3. Perform BUS TIE, 5-8 (Cue Card), then: 4. Go to PWRDN, LOSS OF 1 FC/1 FREON LOOP, 10-18 (2ND FC, 10-20) >> If FC VOLTS > 32.5V (Bus short): DO NOT BUS TIE 5. Go to affected MN BUS LOSS ACTION, 5-15 >> If bus tied to affected FC/BUS: 6. Untie buses If short eliminated: 7. Go to affected MN BUS LOSS ACTION, 5-15 >> If short not eliminated: 8. Perform BUS TIE, 5-8 (Cue Card), good FC/BUS to unpowered bus, then: 9. Perform step 3 of FC SHUTDN, 5-8 (Cue Card), then: 10. If FC VOLTS < 32.5 (FC short): Affected FC REAC VLV - CL 11. Perform affected MN BUS LOSS ACTION, 5-15, then: 12. Go to PWRDN, LOSS OF 2ND FC (ON-ORBIT), 10-20
LOW <21	HIGH >32	LOW <20	FC DISCONNECT 1. Affected FC/MN BUS - ON 2. Perform BUS TIE, 5-8 (Cue Card) If FC3 VOLTS > 32: 3. PL PRI FC3 - ON (tb-ON) MNC - ON (tb-ON), then: If any FC VOLTS > 32: 4. Go to PWRDN, LOSS of 1 FC/1 FREON LOOP, 10-18 (2ND FC, 10-20)

ALL VEH/ORB/BAS

BUS TIE/FC SHUTDN 5-9 **MN BUS UVOLTS**
BUS LOSS ID (Over)

Figure 1-5: A Pocket Checklist Procedure

1.3.4.2. Development

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.*

Chapter 2
Rule-Base Structure

This chapter presents results describing the construction of a *rule-base* suitable for the representation of *procedural logic* in a *declarative, representational form*. The results presented here provide a method of literally *compiling expert knowledge* into a rule-base for use by a variety of possible software implementations which might need to use this rule-base.

2.1. Expert Systems and the Representation Problem

A key problem in developing an effective expert system, as pointed out by [Nilsson 80],

> *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).*

Not only is the characterization of human expertise a problem in developing a machine-processable form but the manner in which it is implemented is a limiting factor in the *efficiency* of an implementation. [McDermott78a 78]found, experimentally, that the cost of running a production system is directly proportional to the product of the size of the rule-base and the number of the currently active rules. Clearly, the size of the rule-base is affected by the manner in which *human rules* are converted to machine-representation. A representation which keeps the total number of rules small will, all other things equal, reduce the cost of operating the rule-based system.

2.1.1. Rule-Base Representations

The rule-bases of expert systems are typically composed of *conditional* relations in the form of *implications* although their logical and physical forms vary. PROLOG, for example, constructs relations from data entered into the database and *computes* relations [Dahl82 82]. While not a rule-base in the usual sense [Dahl83 83], PROLOG treats conditional relations as *Horn clauses* such that the antecedent terms are considered to be *procedure calls* and the consequent is the *name of the procedure* composed of the procedure calls [Dahl82 82]. A PROLOG query is a request to the database to find a *constructive proof* that satisfies the antecedent terms which may, themselves, be constructed as procedure calls, in the form of database queries. In this sense, PROLOG has *deductive, backward-chaining* capabilities typical of query-oriented database systems. As a result, it has features very much like what is usually meant by an expert system. DEVISER [Vere 83] and MYCIN [Shortliffe 76a], on the other hand, incorporate rules directly as LISP expressions. MYCIN [Shortliffe 76a] also uses backward-chaining to perform its diagnoses but represents its rules as LISP functions according to a BNF grammar. This is more typical of current rule-base implementations.

2.1.2. Knowledge Compilation

The *knowledge compilation* concept was apparently first introduced in the ARPA SUR project [Barr 81]. The basic idea is that *valid* information about recognizing sentences is contained in a *single, pre-compiled decision table*. The resulting table or *knowledge source* then was used to analyze voice signals. This concept is not very different from the use of syntax tables in a programming language compiler. A major advantage of this approach is that it is fast and efficient since all possible legal patterns exist for, essentially, pattern matching to the observed voice signal. This is in contrast to other systems such as HEARSAY, pointed out by [Barr 81], which determines valid sequences *on the fly*. Another example, more recent, is PROLOG which also *computes* the valid set of *instances* satisfying a query. One could argue that such difference between, for example, SUR and HEARSAY and PROLOG are really a matter of degree inasmuch as the extent to which rules are pre-defined they are, in a sense, pre-compiled. The distinction is clearer if one considers the disadvantages of compiled knowledge. Again, as pointed out in [Barr 81], compiled knowledge, as implemented in SUR, is relatively inflexible and changes to the rule-base require complete, and time-consuming, recompilation. This is due to the use of a singe data structure for the rule-base in SUR and, in HARPY, to the explicit connection between the decomposed network used by that system. We will show that this is not a fundamental limitation to the use of *compiled knowledge*. A second disadvantage, although perhaps more appropriately described as a characteristic, of compiled knowledge rule-bases is that they are constrained to a particular syntax. This characteristic is truly a fundamental limitation since the compiled knowledge must be stored and the data structure selected to represent that knowledge limits the way in which the data it contains is interpreted. It is the *interpretation* of the data contained in the data structure which constitutes the knowledge within the system. This interpretation must be based on some assumptions about how the data is stored in order to *understand* it syntactically. We will demonstrate this *syntactic limitation* in section 2.5.4 in the context of our use of *Disjunctive Normal Form*.

2.1.3. Validity of the Rule-Base

Since the correctness and validity of the analyses performed by the system is based on the correctness and validity of the rule-base there must be mechanisms for verifying that the rule-base is *logically consistent*. This is similar to the problem of *integrity constraints* [Ullman 80] well-know in database research. This area has not received a great deal of attention in AI research, however. Most systems rely on the human operator, the source of the rules, to maintain the consistency of the rule-base. Methods like *evaluative tests* [Shortliffe 76a], which can never be exhaustive, Part of the reason for this is that the method of verifying consistency or detecting inconsistency is dependent of the representation of the rules themselves. Rule-base validity is really two problems under one name. One of the problems addresses the question of *logical consistency* in the expression of, for example, production rules based on propositional logic. The second problem is essentially a *configuration management* problem. This could be stated as

- **Given:** all production rules in the rule base are *logically consistent*.

- **Question:** do all production rules in the rule base belong there?

An example which illustrates the difference between these two problems can be taken from the SSV problem domain. If we have, by a means to be described, assured that the Boolean expressions used to described a malfunction procedure are consistent (ignoring for the minute the problem of completeness), how do we know that only those for the right SSV are contained in the rule-base. Right now there are two SSV's, *Columbia* and *Challenger.* The second problem is a general database problem which we will not specifically address here. We will show, however, how the rule-base representation presented here permits the verification of logical consistency. It is asserted that this is a significant advance in the available methods for verifying the integrity of expert system rule-bases.

2.1.4. Overview of the Current Approach

The method of representing rules presented here offers significant advantage in contrast to the storage of rules as expressions in a programming language such as LISP, or constructing them in the form propositional queries as in PROLOG. Since *malfunction procedure logic* is expressible in Boolean form and in the form of relational tables, as will be shown, *redundancy* and *logical inconsistency* in the rules may be eliminated automatically through the use of *relational normalization* and *Boolean minimization and variable substitution*. Using the current approach, it is also possible to define logical relationships among the production rules in the form of *equivalence classes* such that each production rule is a member of an equivalence class. We will show that these relationships can be described in terms of *functional dependencies* [Ullman 80] and logical implications thereby permitting both forward and backward chaining using the same data structure.

2.2. Representation of Procedural Logic

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.

2.2.1. 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 implemented the EXPRES program would require modification. This type of maintenance and modification is costly and error-prone as well as difficult to control.

2.2.2. 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.

2.2.3. 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. Benefiting 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.

2.2.4. 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.

2.3. Development of a Boolean Representation

To satisfy these requirements, the approach was developed that represented the procedures as sets of *Boolean functions*. Table 2-1 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 [T. L. Booth 67].

2. take advantage of two methods of implementation software and hardware, as depicted in table 2-1.

The first point will be discussed in greater depth in the next chapter. 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.

2.3.1. Malfunction Procedures as Graphs

The malfunction procedure itself can be thought of as a *directed graph* or *digraph*, $G = (V,E)$, where $V=\{v_1,v_2,\dots,v_k\}$, 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 $v_i \rightarrow v_{i+1}$ [Hopcroft 79]. 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 $v_1,v_2,\dots,v_k$, $k>1$, 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 [Hopcroft 79], 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 1-2. 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.

2.3.2. 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:

$$63a9 = 63a1c \wedge \sim 63a3 \wedge \sim 63a5 \wedge 63a8 \qquad (2.1)$$

Referring to figure 1-2, note the heavily bordered box labeled 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.

2.3.3. Assignment of Variable Names

To represent the logic leading to the conclusion named *9* in figure 1-2 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 1.2.1 this variable is the *consequent* of the *antecedent* conditions on the right-hand side (RHS) of equation (2.1) in section 2.3.2. 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 1-2. 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.

2.3.4. 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 ← ∧ ← 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 2-1. 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 2.3.3, 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.

$63a1E = \sim(63a1a \wedge \sim 63a1b \wedge \sim 63a1c \vee \sim 63a1a \wedge 63a1b \wedge$
$\sim 63a1c$
$\quad \vee \sim 63a1a \wedge \sim 63a1b \wedge 63a1c)$
$63a2 = 63a1a \wedge \sim 63a1E$
$63a4 = (63a1c) \wedge (63a3) \wedge \sim 63a1E$
$63a6 = (63a1c) \wedge (\sim 63a3) \wedge (63a5) \wedge \sim 63a1E$
$63a7 = (63a6)$
$63a9 = (63a1c) \wedge (\sim 63a3) \wedge (\sim 63a5) \wedge (63a8) \wedge \sim 63a1E$
$63a10 = (63a9)$
$63a14 = (63a12) \vee ((\sim 63a12) \wedge (63a13))$
$63a15 = (63a1c \wedge \sim 63a1E) \wedge (\sim 63a3) \wedge (\sim 63a5) \wedge (\sim 63a8) \wedge (\wedge$

$\quad \sim 63a11)(63a12) \wedge (63a14)$
$63a18 = (63a1c \wedge \sim 63a1E) \wedge (\sim 63a3) \wedge (\sim 63a5) \wedge (\sim 63a8) \wedge (63a11)$

$\quad \wedge (63a16) \wedge (63a17)$
$63a19 = (63a1c \wedge \sim 63a1E) \wedge (\sim 63a3) \wedge (\sim 63a5) \wedge (\sim 63a8)$
$\quad \wedge (\sim 63a11) \wedge (63a16)$
$63a20 = (63a1c \wedge \sim 63a1E) \wedge (\sim 63a3) \wedge (\sim 63a5) \wedge (\sim 63a8) \wedge (63a11)$
$\quad \wedge (63a16) \wedge (\sim 63a17)$
$63a21 = (63a19) \vee (63a20)$
$63a24 = 63a22 \wedge (\sim 63a14) \wedge 63a12 \wedge (\sim 63a11) \wedge (\sim 63a8) \wedge$

$\quad (\sim 63a5) \wedge (\sim 63a3) \wedge 63a1c \wedge \sim 63a1E$
$63a25 = (\sim 63a22) \wedge (\sim 63a14) \wedge 63a12 \wedge (\sim 63a11) \wedge (\sim 63a8) \wedge (\sim 63a$
$\quad \wedge (\sim 63a3) \wedge 63a1c \wedge \sim 63a1E$
$63a27 = (63a26) \wedge 63a23 \wedge 63a1b \wedge \sim 63a1E$
$63a28 = (63a25) \vee 63a15$
$63a30 = 63a29 \wedge (\sim 63a26) \wedge 63a23 \wedge 63a1b \wedge \sim 63a1E$
$63a33 = (\sim 63a32) \wedge 63a31 \wedge (\sim 63a29) \wedge (\sim 63a26)$

$\quad \wedge 63a23 \wedge 63a1b \wedge \sim 63a1E$
$63a35 = 63a32 \wedge 63a31 \wedge (\sim 63a29) \wedge (\sim 63a26)$

$\quad \wedge 63a23 \wedge 63a1b \wedge \sim 63a1E$
$63a36 = 63a34b \wedge (\sim 63a31) \wedge (\sim 63a29) \wedge (\sim 63a26)$

$\quad \wedge 63a23 \wedge 63a1b \wedge \sim 63a1E$
$63a37 = 63a30 \vee 63a33 \vee 63a35 \vee 63a38$
$63a38 = 63a34a \wedge (\sim 63a31) \wedge (\sim 63a29) \wedge (\sim 63a26)$

$\quad \wedge 63a23 \wedge 63a1b \wedge \sim 63a1E$
$63a39 = 63a36$
$63a42 = 63a41 \wedge 63a40 \wedge (\sim 63a23) \wedge (63a1b) \wedge \sim 63a1E$
$63a43 = (\sim 63a41) \wedge 63a40 \wedge (\sim 63a23) \wedge (63a1b) \wedge \sim 63a1E$

Table 2-1: CRYO 6.3a (figure 1-2) 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* [Garey 79] 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 2.3.4 is small.

2.4. Prototype Implementation in a High-Level Language

The approach described above was validated through the development of a prototype including the entire set of CRYO malfunction procedures as well as several Failure-Recovery Procedures (FRPs). Using the approach to variable naming and logic representation described above, these procedures were implemented in a convenient high-level language (HLL), ROSIE (Rule-oriented System for Implementing Expertise) [ROSIE 81]. The primary goal of this implementation was to verify that all of the implicit logical relationships contained in the malfunction procedures could be mapped, one-to-one, into the Boolean representation and that this representation could accurately the analysis normally performed by a human, flight controller. 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.

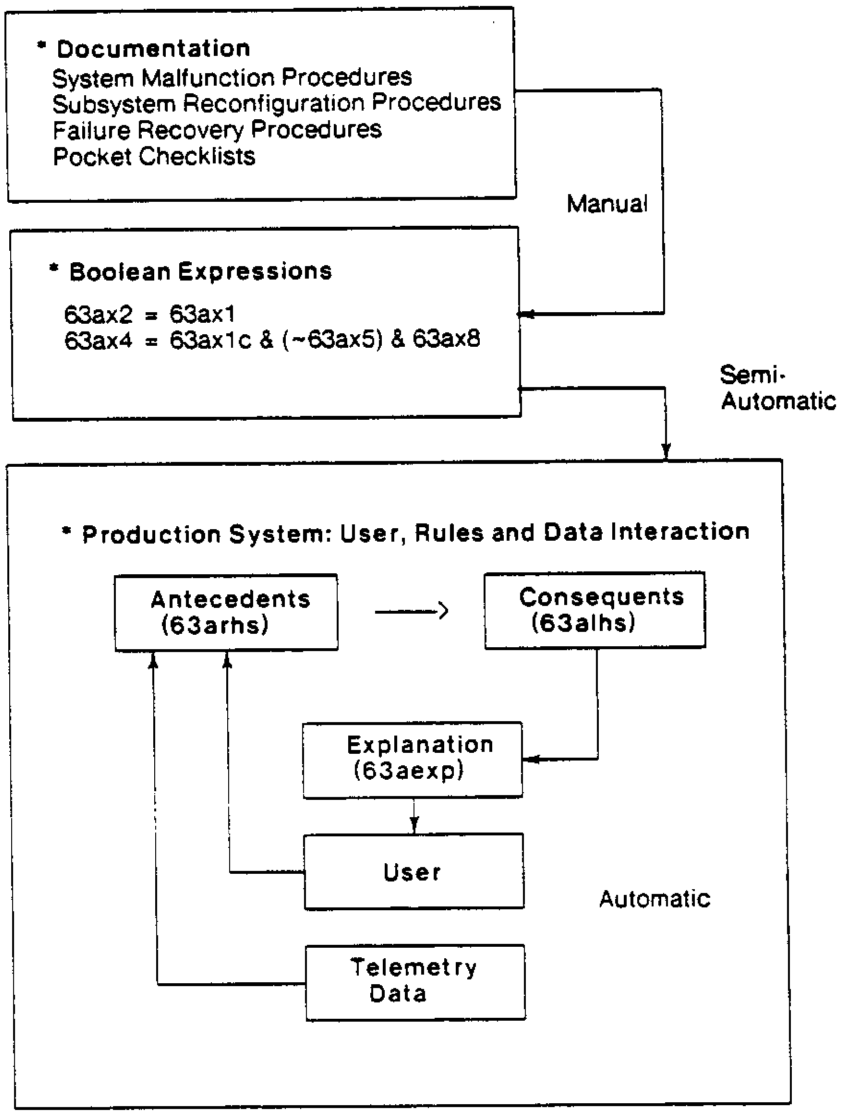

* Documentation
System Malfunction Procedures
Subsystem Reconfiguration Procedures
Failure Recovery Procedures
Pocket Checklists
Manual
* Boolean Expressions
63ax2 = 63ax1
63ax4 = 63ax1c & (~63ax5) & 63ax8
Semi-Automatic
* Production System: User, Rules and Data Interaction
Antecedents (63arhs)
Consequents (63alhs)
Explanation (63aexp)
User
Automatic
Telemetry Data

Figure 2-1: Equivalent Representations of Procedural Information

2.4.1. Structure of the Rule-Base: Software Implementation

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 2-1. 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 2-1.

   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 2-2, 2-3, 2-4 show the form each of these categories takes when constructed as ROSIE rule-sets.

Table 2-2: 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 2-3: 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 2-4: 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",retu
```

2.5. Programmable Logic Array Model of the Rule-Base

Although the HLL implementation was successful, it was obvious that the rule-base would grow quite large very quickly. It was apparent that a more efficient representation of the same logical relations could be achieved by representing the Boolean expressions as bit vectors.

2.5.1. Hardware Equivalence

Although it is unlikely that malfunction procedures would ever be implemented in hardware, one of the principal results of this work is the concept of *hardware-equivalence*. By hardware equivalence, we mean the ability to use the Boolean formalism 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 literally a form of *compiled knowledge*, independent of any software implementation, programming language or host machine. 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, consider the problems of EXPRES and CRYEX, 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 describe the steps necessary to convert Boolean expressions to their PLA form and present the results of an actual PLA synthesis and the resultant *truth-table* which will be the prime example used throughout this work.

2.5.2. Generation of the PLA Description

Once a 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, and almost entirely automatic. A set of Boolean equations is *transformed* into PLA descriptions through an automated series of translations involving different intermediate representations. Each of the representations is a description of a different aspect of what would, if a physical circuit were to be manufactured, be the implementation process. The Boolean equations are converted into a *truth-table*; 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 2-2 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 [Mayo 83]. The names of the major programs involved are listed in figure 2-2 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 conceivably be identified for hardware implementation.

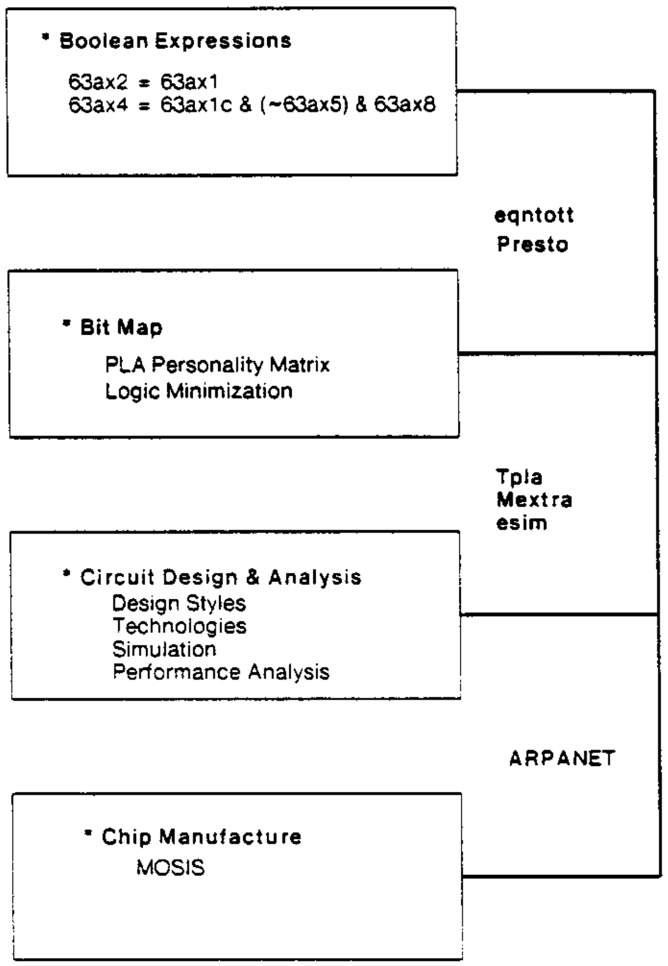


Figure 2-2: Design and Manufacture Sequence for VLSI Devices

2.5.3. 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 2-1 displays the input to *eqntott* while figure 2-3 shows the resulting *personality matrix* and the PLA resulting from that matrix as described in section 2.5.5. 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.

2.5.4. Minterms, Truth-tables and PLAs

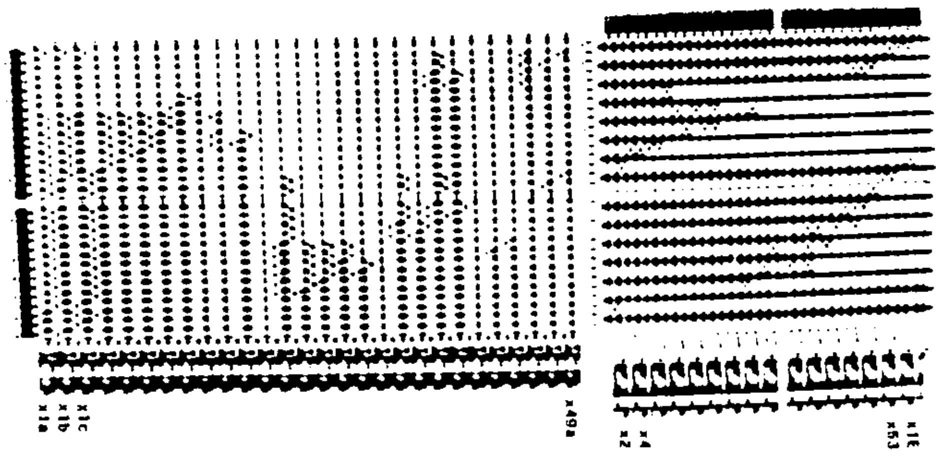
Disjunctive normal form (DNF) expresses Boolean functions as *sums of products* or *minterms* [Thomas 79]. One method of organizing information contained in Boolean functions in general and functions in DNF in particular is through the use of a truth-table. Each in the truth table is a *minterm*. A truth table serves to enumerate 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 use of *don't cares* is a form of data compression and is entirely equivalent to a fully enumerated truth-table. There is no loss of information due to the introduction of *don't care* terms.



```
-0-0-0-----------------------------------------------------   0000000000000000000000000000000001
--------------------------------------------0--------1---1-   0000000000000000000000000000000110
--------------------------------------------0--------1-1---   0000000000000000000000000000011000
-------------------------------------------1--0------1-----   0000000000000000000000000000100000
-------------------------------------------1-1-------1-----   0000000000000000000000000011000000
----------------01-----------------------------------------   0000010000000000000000000000000000
---------------1-------------------------------------------   0000010000000000000000000000000000
-0-01--0-0-0-0-----1---------------------------------------   0000000001010000000000000000000000
-0-01--0-0-0-01--------------------------------------------   0000000100000000100000000000000000
-0-01--0-0-01------1--0------------------------------------   0000000000011000000000000000000000
-0-01--0-0-01------1-1-------------------------------------   0000000010000000000000000000000000
-0-01--0-01------------------------------------------------   0000110000000000000000000000000000
-0-01--01--------------------------------------------------   0011000000000000000000000000000000
-0-01-1----------------------------------------------------   0100000000000000000000000000000000
-01--0--------------------0------------0---0-------------1-   0000000000000000000000000000000110
-01--0--------------------0------------0---0-----------1---   0000000000000000000000000000011000
-01--0--------------------0------------0--1--0-------------   0000000000000000000000000000100000
-01--0--------------------0------------0--1-1--------------   0000000000000000000000000011000000
-01--0--------------------0-----------1--0-----------------   0000000000000000000000001000000000
-01--0--------------------0-----------1-1------------------   0000000000000000000000010100000000
-01--0-------------------1--0-0-0------------------1-------   0000000000000000000001100000000000
-01--0-------------------1--0-0-0----------------1---------   0000000000000000000010010000000000
-01--0-------------------1--0-01--0------------------------   0000000000000000001001000000000000
-01--0-------------------1--0-01-1-------------------------   0000000000000000000101000000000000
-01--0-------------------1--01-----------------------------   0000000000000000010001000000000000
-01--0-------------------1-1-------------------------------   0000000000000001000000000000000000
--1-1------------------------------------------------------   0000000000000000000000000000000001
1--0-0-----------------------------------------------------   1000000000000000000000000000000000
1---1------------------------------------------------------   0000000000000000000000000000000001
1-1--------------------------------------------------------   0000000000000000000000000000000001
x1a x1b x1c                                           x49a     x2 x4                        x53 x11
```

Figure 2-3: Disjunctive Normal Form Matrix and Corresponding Circuit Topology

2.5.5. 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 described in section 2.5.4. 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 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.

2.5.6. 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 2-3 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. Equivalence Classes

Equivalence classes are introduced to provide a common denominator for the discussion of *functional dependencies* and *equivalent fault states*. Their importance is associated with the fact that *fault states* may be entered in a variety of ways and yet result in the same functional loss. The relationship between causes of faults and faults themselves is, therefore, *many-to-one*. Some interesting work has been done on the definition and identification of *fault equivalence* in sequential machines [Boute 71] and we will discuss this later in the context of *finite state automata*. In the meantime we will develop a description of equivalence classes relevant to the malfunction procedure problem. To aid in this development, the following theorem is reproduced from [Dean 66], without proof, for convenience since we will refer to it later.

> **Theorem 1:** Let A be a nonempty set and let σ denote an equivalence relation on A. For all $x \in A$ let $S_x = \{y \mid y \sigma x\}$. A is the set sum of the set of mutually disjoint, nonempty subsets S_x; $A = \{x \mid x \in A\}$. S_x is called an *equivalence class* of A under σ.

To complete the description of equivalence class, we require that *equivalence relations* exhibit the properties of *reflexivity, symmetry, and transitivity* [Kemeny 66].

2.6.1. Construction of Fault Equivalence Classes

Because malfunction procedures describe anomaly conditions and these *anomalous states* may be sometimes entered in a variety of ways, it is useful to consider the characteristics of the states. Equivalence classes provide a construct to do that. To construct these classes based on the *data representation* developed in this work, we define equivalence classes over the set of *implicants, I*, corresponding to the *tuples* or *rows* of the relations corresponding to malfunction procedures. From our previous discussion it is clear that each *tuple* can be considered to be a bit-vector with each bit-position corresponding to an attribute of the relation. Each attribute corresponds to a single Boolean variable in an antecedent term of a production rule. As first described in section 2.5.4, implicants compose the AND-plane of the PLA personality matrix and, consequently, also the left-hand part of the relational table, as seen in figure 2.7. In order to construct equivalence classes over the set of implicants for a relation we must show that the relation composed of the implicants satisfies the properties of reflexivity, symmetry and transitivity.

2.6.1.1. Reflexivity

The reflexive property is simple to satisfy in the PLA by introducing a *feedback* term equivalent to the production, $A \rightarrow A$. These are not necessary nor normally carried in the relation nor in the PLA since it is wasteful of storage in the case of the relational table and chip space on a PLA. It is reassuring, however, to see that its implementation is straightforward and consistent with the development presented so far.

2.6.1.2. Symmetry

Symmetry requires that, for the production $A \rightarrow B$, the relation, $B \rightarrow A$ also holds. This is easy to prove using our definition of *conditional* relation. First, transform $A \rightarrow B$ and $B \rightarrow A$ to clause form, $\sim A \vee B$ and $\sim B \vee A$. Symmetry is satisfied if and only if $(A \rightarrow B) \wedge (B \rightarrow A)$. Manipulation of this expression using the axioms of Boolean algebra leads to

$$[(\sim A \wedge \sim B) \vee (B \wedge \sim B)] \vee [(\sim A \wedge A) \vee (A \wedge B)]$$

Inspection of the relational table in figure 2.7 shows that each column or *attribute* in the right-hand side of the table can be represented in the form $A \Rightarrow B$ where, in general, B is a set of *implicants* whose composition is indicated by the position of 1 in the appropriate row of the column corresponding to B. Conversely, since the table is produced from expressions of the form $B \Rightarrow A$ we know that those relations also hold in the table. This demonstrates that the relations realized as rows and columns of the PLA personality matrix and the resultant relational table are symmetric and that the tables preserve symmetry. Since the columns of the right-hand side of the table represent the disjunction of the implicants on the left-hand side of the table, the table is symmetric over the set of implicants.

2.6.1.3. Transitivity

Transitivity requires that relations of the form $A \rightarrow B$ and $B \rightarrow C$ also result in $A \rightarrow C$. As under reflexivity, this condition can be assured by the use of *feedback*, which results in the inclusion of tuples which explicitly record the $A \rightarrow C$ relation or through variable substitution. Variable substitution has the effect of substituting for B the Boolean variables (attributes) of which A is composed. This is realized as a new tuple in the relation table which is equivalent to $A \wedge B$.

2.6.2. Example of Equivalence Classes in CRYO6.3a

Equivalence Class: *63a43*
$63a43 = \sim 63a41 \wedge 63a40 \wedge \sim 63a23 \wedge 63a1b$

Equivalence Class: *63a44*
$63a44 = 63a42$
$63a44 = 63a47$

Equivalence Class: *63a47*
$63a47 = 63a46 \wedge 63a45 \wedge \sim 63a40 \wedge \sim 63a23 \wedge 63a1b$
$63a47 = 63a46 \wedge 63a45 \wedge 63b8$

Equivalence Class: *63a48*
$63a48 = \sim 63a46 \wedge 63a45 \wedge \sim 63a40 \wedge \sim 63a23 \wedge 63a1b$
$63a48 = \sim 63a46 \wedge 63a45 \wedge 63b8$

Equivalence Class: *63a50*
$63a50 = 63a49b \wedge \sim 63a45 \wedge \sim 63a40 \wedge \sim 63a23 \wedge 63a1b$
$63a50 = 63a49b \wedge \sim 63a45 \wedge 63b8$

Equivalence Class: *63a51*
$63a51 = 63a50$

Equivalence Class: *63a52*
$63a52 = 63a49a \wedge \sim 63a45 \wedge \sim 63a40 \wedge \sim 63a23 \wedge 63a1b$
$63a52 = 63a49a \wedge \sim 63a45 \wedge 63bx8$

Equivalence Class: *63a52*
$63a53 = 63a52$

Table 2-5: Equivalence Classes Within **CRYO 6.3a**

Equivalence classes, as defined above, are common occurrences in malfunction procedures. Table 2.6 displays some of the more interesting, non-trivial, equivalence classes within CRYO6.3a. Trivial classes are defined as those classes which are subsets of Λ with cardinality of one.

2.6.3. Generation of Disjoint Subsets

The requirement of *disjointness* is not a restriction for our purposes since can decompose non-disjoint elements of Λ, corresponding to paths, as necessary in the following manner. For arbitrary elements of Λ taken pair-wise, $(B \rightarrow A, C \rightarrow A)$, such that $(B \rightarrow A) \cap (C \rightarrow A) \neq \emptyset$ perform the following decomposition. Create the new production, $D \rightarrow A$, such that $B = \{D \cup \beta\}$ and $C = \{D \cup \gamma\}$ and $\beta \cap \gamma = \emptyset$. This decomposition can always be performed unless $B = C$ in which case *B* and *C* are redundant and one of them can be eliminated. An example of this decomposition using one the equivalence classes of Table 2.6 is shown below.

Equivalence Class: *63a48*

$63a48 = \sim 63a46 \wedge 63a45 \wedge \sim 63a40 \wedge \sim 63a23 \wedge 63a1b$

$63a48 = \sim 63a46 \wedge 63a45 \wedge 63b8$

We re-write these expressions in the more common form of productions and according to the previous definition of *production rules*:

$(\sim 63a46 \wedge 63a45 \wedge \sim 63a40 \wedge \sim 63a23 \wedge 63a1b) \rightarrow 63a48$

$(\sim 63a46 \wedge 63a45 \wedge 63b8) \rightarrow 63a48$

Then we define:

$B = \{\sim 63a46, 63a45, \sim 63a40, \sim 63a23, 63a1b\}$

$C = \{\sim 63a46, 63a45, 63b8\}$

and then:

$D = \{\sim 63a46, 63a45\}$

$\beta = \{\sim 63a40, \sim 63a23, 63a1b\}$

$\gamma = \{63b8\}$.

This results in the generation of a new equivalence class, D, which may be considered to be a subproposition of the parent, $63a48$, for the purposes of variable naming as discussed in section 2.3.3.

2.7. Relational Model: Malfunction Procedures as Relations

The relational model [Codd 70], [Ullman 80], [Date 77], provides a convenient and powerful method of manipulating data which is constructed in the form of tables. If we view the *personality matrix* of section 2.5 as a *relational table*, as it is represented in Figure 2.7, then we, *de facto*, have constructed the corresponding malfunction procedure as a *relation*. In Figure 2.7, the names of the antecedents shown in Table 2-1 are the *attributes* of the left portion of the relation and the consequents are the *attributes* on the right part of the relation. In the discussion of *functional dependencies* we will show that the antecedents are the *attributes composing the relational key*. Each row of the table is a *tuple* in relational terminology.

```
6666666666666666666666666666666666666666666  666666666666666666666666666666666
3333333333333333333333333333333333333333333  333333333333333333333333333333333
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1111356891111111122222233333333344444444558  124679111112222223333333444445555
         123456790235690123445680125679902         045890145780356789234780123
Eabc                       ab         ab     E
______________________________________________________________________________
-000---------------------------------------  100000000000000000000000000000000
-----------------------------------------1-  000000000000000000000000000000001
----------------------------------------1--  000000000000000000000000000000100
-----------------------------------0---1--1  000000000000000000000000000001000
-----------------------------------0--1---1  000000000000000000000000000000010
-------------------------------------1-----  000000000000000000000000001000000
-----------------------------------10-----1  000000000000000000000000000010000
-----------------------------------11-----1  000000000000000000000000000100000
----------------------------------1--------  000000000000000000000000001000000
-------------------------------1-----------  000000000000000000000100000000000
------------------------------1------------  000000000000000000000001000000000
-----------------------------1-------------  000000000000000000000100000000000
--------------------------1----------------  000000000000000000000100000000000
-----------------------1-------------------  000000000000000000000100000000000
--------------------1----------------------  000000000000000010000000000000000
-----------------1-------------------------  000000000000100000000000000000000
----------------1--------------------------  000000000000100000000000000000000
-------------1-----------------------------  000000000000000010000000000000000
----------01-------------------------------  000000010000000000000000000000000
----------1--------------------------------  000000010000000000000000000000000
--------1----------------------------------  000000100000000000000000000000000
------1------------------------------------  000010000000000000000000000000000
______________________________________________________________________________
6666666666666666666666666666666666666666666  666666666666666666666666666666666
3333333333333333333333333333333333333333333  333333333333333333333333333333333
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1111356891111111122222233333333344444444558  124679111112222223333333444445555
         123456790235690123445680125679902         045890145780356789234780123
Eabc                       ab         ab     E
```

Figure 2-4: Interpretation of CRYO6.3a as A Relational Table

2.7.1. Relational Description of the Rule-Base Derived from the PLA Model

The rule-base may be defined as the set, S, of malfunction procedures, M_k, such that $S = \{M_1, \ldots, M_t\}$. Each malfunction procedure, M_k, is a *relation* with its *tuples* corresponding to a path through the malfunction procedure *digraph*, as described in section 2.5.4, and is one of the *implicants* in the *personality matrix* of the malfunction procedure. Each tuple, therefore, is composed of *antecedent* terms, corresponding to the AND-plane of the PLA, and *consequent* terms, corresponding to the OR-plane terms of the PLA. We write each tuple in the form $t_i = (A_1, \ldots, A_l, B_{l+1}, \ldots, B_n)$. The A_i are the terms of the antecedents and the B_i are the terms of the consequents.

2.7.1.1. Integrity Constraints Implicit in Disjunctive Normal Form

Because the implicants are in DNF, the relation M_k implies the following logical relations:

$$\prod_{j=1}^{l} A_{ij} \rightleftarrows \sum_{j=l+1}^{m} B_{ij} \tag{2.2}$$

$$\sum_{i=1}^{n} B_{ij} \rightleftarrows \sum_{j=l+1}^{m} (\prod_{j=1}^{l} A_j)_i \tag{2.3}$$

The first equation states that a functional output is true if and only if *all* of the antecedent conditions for that tuple are satisfied as well as the converse. This equation is said to be *row-oriented* or *tuple-oriented*. The second equation states that an output term is true if and only if the antecedent terms of *some* tuple are satisfied. as well as the converse. These may be considered integrity constraints in the sense that every relational table corresponding to the PLA format is constrained to satisfy these expressions.

```
666666666666666666666666666666666666666666 666666666666666666666666666666666
333333333333333333333333333333333333333333 333333333333333333333333333333333
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1111356891111111222222333333333444444444558 124679111112222223333333444445555
          123456790235690123445680125679902       045890145780356789234780123
Eabc                             ab        ab   E
_____________________________________________________________________________
0--100-0-0----1---------------------------- 000000000010000000000000000000000
0--100-0-01-0-----0------------------------ 000000000000001000000000000000000
0--100-0-01-0-----1------------------------ 000000000000010000000000000000000
0--100-0-01-1------------------------------ 000000001000000000000000000000000
0--100-0-1----10--------------------------- 000000000001000000000000000000000
0--100-0-1----11--------------------------- 000000000100000000000000000000000
0--100-1----------------------------------- 000001000000000000000000000000000
0--101------------------------------------- 000100000000000000000000000000000
0--11-------------------------------------- 001000000000000000000000000000000
0-1---------------0------------0--0---1---- 000000000000000000000000000001000
0-1---------------0------------0--0--1----- 000000000000000000000000000000010
0-1---------------0------------0--10------- 000000000000000000000000000010000
0-1---------------0------------0--11------- 000000000000000000000000000100000
0-1---------------0------------10---------- 000000000000000000000000010000000
0-1---------------0------------11---------- 000000000000000000000000100000000
0-1---------------1-00-0---1--------------- 000000000000000000001000000000000
0-1---------------1-00-0--1---------------- 000000000000000000000100000000000
0-1---------------1-00-10------------------ 000000000000000000100000000000000
0-1---------------1-00-11------------------ 000000000000000000010000000000000
0-1---------------1-01--------------------- 000000000000000001000000000000000
0-1---------------1-1---------------------- 000000000000000100000000000000000
-011--------------------------------------- 100000000000000000000000000000000
--11--------------------------------------- 100000000000000000000000000000000
01----------------------------------------- 010000000000000000000000000000000
-1-1--------------------------------------- 100000000000000000000000000000000
-101--------------------------------------- 100000000000000000000000000000000
-11---------------------------------------- 100000000000000000000000000000000
-110--------------------------------------- 100000000000000000000000000000000
-111--------------------------------------- 100000000000000000000000000000000
_____________________________________________________________________________
666666666666666666666666666666666666666666 666666666666666666666666666666666
333333333333333333333333333333333333333333 333333333333333333333333333333333
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1111356891111111222222333333333444444444558 124679111112222223333333444445555
          123456790235690123445680125679902       045890145780356789234780123
Eabc                             ab        ab   E
```

Figure 2-4, continued

2.7.1.2. Buried Functional Dependencies

These equations are very interesting because they imply *functional dependencies* which are a direct result of the representation of the PLA model in *disjunctive normal form.* From the definition of the *biconditional*, $\rightleftarrows$, the *Equivalence Theorem* of [Fagin 77] for functional dependencies and logical implications and equations (2.2), (2.3), we can write, for values of $i, 1 \leq i \leq n$ and $k, l < k \leq m$

$$\prod_{j=1}^{l} A_{ij} \rightarrow B_{ik}$$

This expression is an immediate consequence of equation (2.2) since it is the special case where the sum of the B_{ij} is such that $m = j$. This can be stated by saying that each functional dependency corresponds to a single term in the sum on the right-hand side of equation (2.2).

2.7.2. Functional Dependencies as Implications

Functional dependencies (FD's) are logical implications of the form $X \Rightarrow Y$.[1] This expression is read as X *functionally determines* Y. This notation is different from that used for productions as a result of the fact that FD's are statements about the relationships of subsets of attributes within a relation. These relationships are independent of the truth-value of the attributes. As specified in definition 2.5.4, *FD's are implication relations.* Remember that *attributes* correspond to the *Boolean variables* in figure 2.7.

[1]Although this definition is identical to that of [Ullman 80] the notation differs.

Chapter 3
Real-Time Analysis of Malfunctions

Results from section 2.2 provide the basis for the generalization of the problem of representing and executing automated malfunction procedures in *real-time* to that of procedures of a more generic nature. This chapter presents the methodology and results of an approach which adapts the *knowledge representation* of section 2.2 to a *control strategy* suited to the problems faced by the flight control team. There are minimum, fundamental requirements that an automatic system must meet. It must

1. Monitor and validate nominal SSV system functions. That is, the automated system must *be aware* of the state of the SSV by the time-sampling of telemetry data.

2. Identify the subset of malfunction procedures relevant to the current vehicle state. This process is sometimes referred to as *conflict resolution* [Georgeff 82].

3. Detect and annunciate the violation of nominal conditions; referred to as *limit-sensing*.

4. Invoke and control the execution of malfunctions procedures, with or without man-in-the-loop, for anomaly resolution.

5. Provide an explanation for the actions of the control process and the resulting analysis.

3.1. Introducing Time into Rule Processing

One of the most difficult problems in the field of expert system research has been the incorporation of *a sense of time* into the analyses provided by expert systems. The difficulty is related to the many ways in which time is used by humans as well as the diverse vocabulary for describing different aspects of time. Work by [Kahn 77] resulted in the description of three constructs for the organization of temporal knowledge. These constructs included a simple, temporal sorting using *date-lines*;

special reference events; and *before/after chains*. This work concluded that the date-line construct was simplest and most efficient unless temporal information was sparse or a wide, and unspecified, range of time-related queries were expected. In either of the latter situations, date-lines were less desirable than the other constructs. Most recent work which makes use of time is related to the development of *planning systems* such as that of [Vere 83]. Earlier, and related, work on planning systems for robots resulted in PERT-like scenarios for robot time-lines [Hendrix 73]. In particular, [Hendrix 73] used time both as a measure of location with respect to a robot time-line as well as an interrupt to initiate and terminate robot activities. The most systematic, and probably most general, approach to the description of time-concepts has been exhibited by the work of [Bruce 72]. This work attempts to define a formal description of time-concepts (e.g., before, after, during) using first-order logic. This approach has the advantage of providing a *mapping*, through the use of propositional definitions, between natural-language syntax and the syntax of propositional logic.

3.1.1. Relationship of Time to Causality

The distinction between implication and conditional relations, introduced in section 2.6, is important both in describing the *domain features* as well as in the development of the notion of *causality* which is fundamental to the analysis of anomalies. The purpose of analyzing malfunctions, the largest part of the flight controllers' job, is to determine the *cause* of failures. It is fortunate that this problem domain is *strictly monotonic in time*. This eliminates much of the ambiguity associated with time-concepts and permits fairly simple assumptions in the determination of causal sequences. It is important to note that although the physical system represented by the set of implication relations is time-monotonic, the analysis of failures is *not*. As described later, *non-monotonic reasoning* is required both in real-time analysis and rule-synthesis.

3.1.1.1. Domain Features

To facilitate our discussion of causality, we introduce a classification of domain features according to the following definitions.

> **Definition 1:** A *first-order*, domain feature is any production-rule contained in the set of implication tables. These *features* are defined *apriori* during the *knowledge compilation* process (cf. section 2.1.2). These features correspond to the set of rules sometimes referred to as *production memory* [McDermott78a 78]. The major distinction of these features is that they are static.
>
> **Definition 2:** A *second-order*, domain feature is the result of the dynamic, (i.e., time-dependent) satisfaction of production rules. This is correspondent to the notion of *working memory* also used by [McDermott78a 78].
>
> **Definition 3:** *Zeroth-order*, domain features are the lowest level (closest to physical systems) and consist of variables and logical connectives used to construct first-order features and which are directly determined by external data (e.g., telemetry parameters) or directly computed in the process of determining second-order features.

The definitions used to describe the domain features can be summarized by thinking of implications as *first-order* domain features as the abstractions determined by the logical structure of the domain. They are *static, structural features* since their validity is not affected by the truth-value assumed by any subset of the *conditional relations* at any point in time. The realization of the *dynamic, time-dependent* character of the domain constructs the *second-order* features. The logical structure of the domain is the set of logical *statements*, composing the *conditional relations*, which are constructed constructed from variables and connectives. As such, these statements are *zeroth-order* or *atomic* domain features.

3.1.1.2. Causality

Causality is a slippery subject and has been much discussed in philosophy [Armstrong 71] as well as computer science [Barr 81], [Hofstadter 79]. Fortunately, our situation is relatively simple. Given that we are dealing with a time-monotonic system, the SSV, the criteria for establishing causality are:

1. the cause of an anomaly must be a member of the equivalence class of that failure

2. the antecedents (first-order features) of the cause must have been

realized as second-order features *before* the realization of the equivalence class.

While these seem like obvious, and innocuous requirements for the establishment of causality, they are both *necessary* and *sufficient* conditions analyzing causes. If either of these conditions is not examined or is unsatisfied in the isolation of causes, the result is merely a correlation rather than a cause. This will be shown in the example presented later.

3.2. A Description of R/T Processing Based on Levels of Abstraction

Viewing the problem of monitoring the SSV through the artifact of *levels of abstraction*, figure 3-1, is useful for a variety of reasons. The decomposition of the overall problem of monitoring the system into various levels

1. provides a view of the system architecture as it currently operates

2. identifies the interconnections in the communications paths required for the identification and resolution of anomalies

3. establishes a baseline against which progress in automating the entire process can be measured.

The salient features of each level is described below.

3.2.1. Level 0: SSV Systems

The SSV systems are monitored through instrumentation at the level of switch and valve, temperature sensors, pressure sensors and, in general, chemo, and electro-mechanical devices. There are also a large number of crew-controlled, manual switches which are important to higher-level processing and interpretation of telemetry.

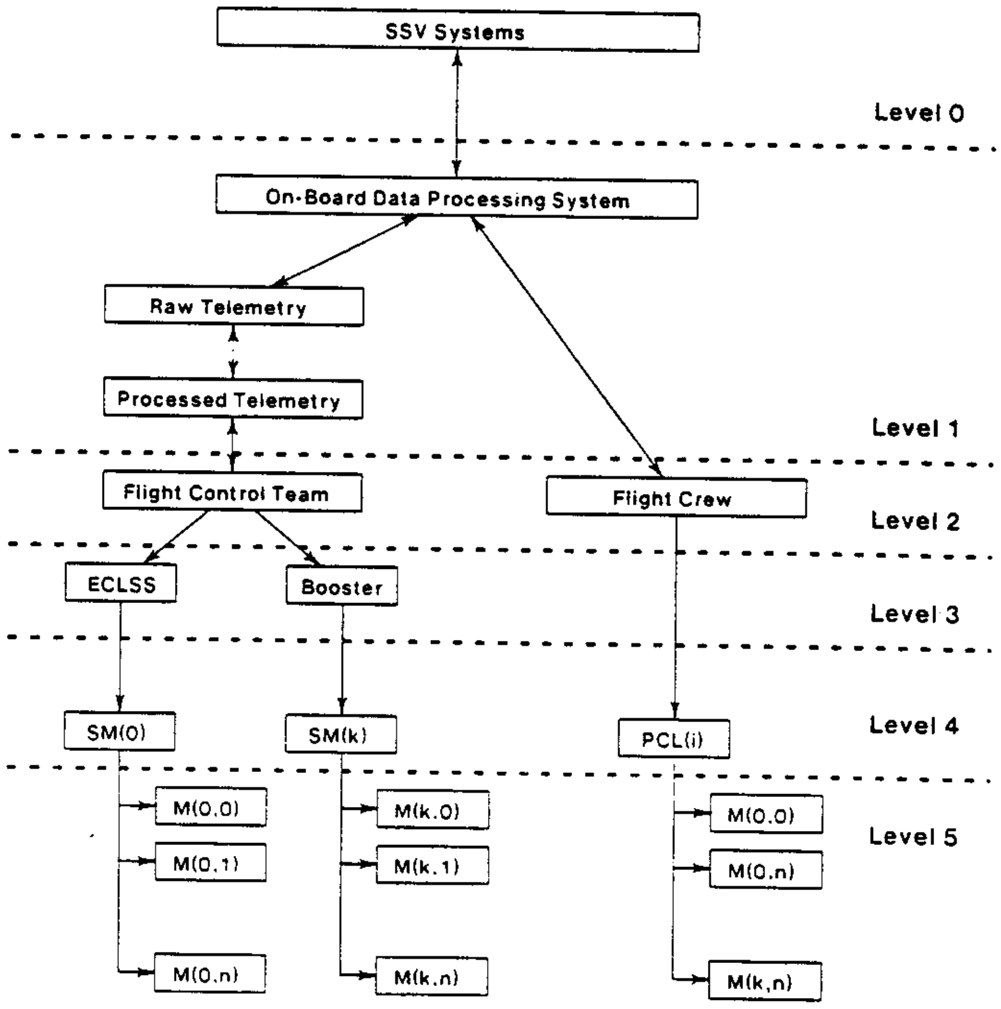


Figure 3-1: Levels of Abstraction in SSV Monitoring

3.2.2. Level 1: On-Board Data Processing and Telemetry

Level 0 data is interfaced to the on-board data processing systems and this is made available both to the flight crew and the ground controllers. Limited data processing is performed on-board although much of the flight critical processing is available directly to the crew if necessary to support emergencies.

3.2.3. Level 2: Flight Control Team and Flight Crew

This is the level at which operational decisions are made. Data passed to the flight control team is interpreted by the controllers both in the Flight Control Room and the Multi-Purpose Support Rooms, figure 1-1, and resultant actions are coordinated with the flight crew through the use of voice circuits. The flight control team has the primary anomaly processing responsibility.

3.2.4. Level 3: Decomposition into SSV Systems

Although level 2 is the primary interface to the crew and the SSV, this level is the first functional decomposition of responsibility into SSV systems and subsystems; sometimes referred to as disciplines. At this level, malfunction processing first becomes compartmented into distinct *first-order domains*.

3.2.5. Level 4: Super-Malfs and Pocket Checklists

Level 4 is included because it corresponds to a higher level of logical processing than that of the malfunction procedures themselves. For example, super-malfs provide the interface between the human controllers and any single malfunction procedure. By analogy with programming. the super-malfs are the main procedure from which the malfunction procedure subroutines are called. Based on currently available documentation, this is the lowest-numbered level which is mapped into first-order features.

3.2.6. Level 5: Malfunction Procedures

Level 5 is the level of the documented, malfunction procdedures discussed extensively already in section 1.3.4. This level represents the lowest level of documented information used to compose the rule base directly. The bulk of first-order features *reside* at, but are not limited to, this level.

3.3. Constructing Real-Time, Second-Order, Domain Features

To perform the construction we must further describe the operating environment. As depicted in figure 3-1, telemetry data is passed to the ground system, and eventually, to the flight control team in the form of time-sampled, parametric observations of on-board sensors. Via data processing, limit checking can be performed for the purpose of setting a bit in the *status vector* which is shown in figure 3-2. When this status vector is altered, a process is initiated which performs a controlled search of the library of first-order features which we also refer to as the *set of implication relations*.

3.3.1. Forward Chaining: Data Driven Analysis

By performing a pattern match of the *status vector* with the first-order tables, the process, called FSCAN in figure 3-2, finds the *relevant* rules and copies them into the second-order set which is a time-ordered, PLA-table structured in the same manner as the first-order table. That is, all antecedent zero-order terms in the first-order table are present in the left part of the second-order table and all consequent zero-order terms are included in the right hand part of the second-order table. The resultant second-order table can be thought of as a state-table which defines the state of the SSV system or subsystem described by the first-order tables which are considered in the pattern-match performed by FSCAN. For example, CRYO63a is a malfunction procedure within the Cryogenics system of the SSV. Therefore, the corresponding state-table generated by FSCAN, when it has been applied to the entire CRYO63 library, describes the state and chronology of the cryogenics subsystem of the SSV. Since the state-table preserves the DNF of the first-order tables it is possible to summarize this table operations, by mapping the time-dependent tuples contained in

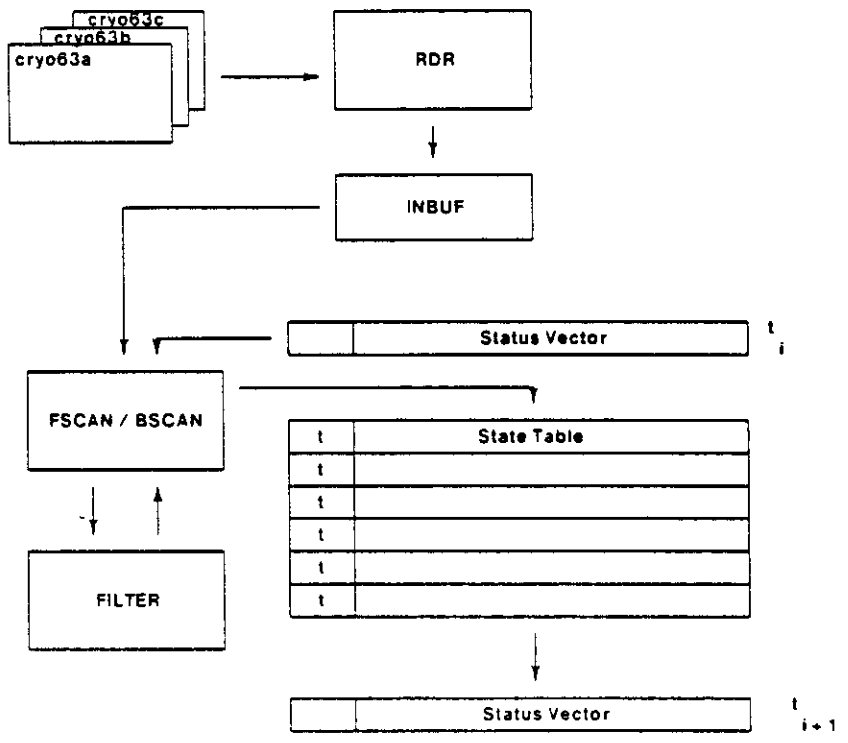


Figure 3-2: Real-Time Malfunction Processing

the state-table into a single, *new status vector.* This process is described in section 3.3.4.

3.3.2. Backward Chaining: Hypothesis-Driven Analysis

While FSCAN performs the data-driven analysis. BSCAN performs the *what-if* type of analysis necessary to explore possible causes of anomalies in the face of anticipated failures or incomplete data. BSCAN functions by constructing the equivalence classes corresponding to selected failures. A good example of the utility of this method of analysis is the identification of the set of *next-worst failures*. This type of analysis is critical to contingency planning as well as the elimination of irrelevant information during the analysis of anomaly processing. BSCAN operates in exactly the inverse manner of FSCAN. This is accomplished by performing the pattern-match on the right hand part (OR-plane) of the first-order tables rather than the left-hand (AND-plane) of the tables. By performing a similar copy operation into a *virtual state-table*, since BSCAN performs hypothetical or *virtual* analyses, it is possible to collect the set of potential causes of various system anomalies. As in the case of FSCAN, a single status vector can be generated as the basis for reporting the possible causes back to the operator, however unlike FSCAN, this method of hypothesis exploration has the potential of generating *logical inconsistencies* in the resultant virtual state-table. In other words, the virtual. second-order set may contain *mutually exclusive* rules which cannot be collapsed into a single virtual status vector without loss of information and. therefore, introducing an error. This will be discussed further in the example below.

3.3.3. Semantic Attachment

Since the processing of the first-order tables is entirely *bit-vector* oriented. it is necessary to perform *semantic attachment* to the resultant second-order tables and status vectors for human interpretation when that is needed. Note that the ultimate goal of automation of malfunction processing is to minimize human involvement and that the burden of processing associated with *semantic attachment*. while small. can be expected to decrease as the system becomes *smarter*, as describe later.

3.3.3.1. Storage of Semantic Values for Zero-Order Features

Zero-order features are the variables and logical connectives composing the rules (first-order features). Variables correspond to the boxes in the source malfunction procedure document as shown in figure 1-2. The text associated with these boxes is the *semantic value* of the zero-order variables. These can be stored as simple sequential lists, as shown in table 2-4. Note that this association of text to the zero-order variable was used in the high-level language implementation described in section 2.4.

3.3.3.2. Retrieval of Zero-Order Semantic Values

A major advantage of the use of the PLA model is the fact that there is a fixed *meaning* to the rule structure. That is, all first-order features (rules) have antecedents which are conjunctions. The equivalence classes, which are composed of first-order features, have a disjunctive relationship between the members of any given equivalence class. Therefore the construction of the explanation of a malfunction analysis merely requires the insertion of *and* and *or* in the proper place during the retrieval of the semantic values for reporting to the human operator. This is shown in table 3-3 for a hypothetical state vector resulting from the analysis of CRYO6.3a.

63a1a	= Hydrogen Pressure Normal
63a1b	= Hydrogen Pressure High
63a1c	= Hydrogen Pressure Low
63a1E	= Hydrogen Pressure Sensor Error (more than one indicator set)
63a2	= C/W Failure
63a3	= Pressure in all tanks < 153 psi
63a4	= System Leak
63a5	= Both P and TK P of affected tk low?
63a6	= leak between affected tk and check vlv. leak cannot be isolated.
63a7	= Deact htrs in affected tk. (R1) H2 TK1(2,3) HTRS A,B - OFF or (A11) TK4 HTR H2 A,B - OFF
63a8	= Also get O2 Press Alarm and/or S68 H2 Press and S68 O2 Press msg lines?
63a9	= Loss of ESS bus, common to both affected O2 and H2 HTR cntlrs. Affected TK qtys also lost. If ESS1BC O13 & R15 lost, then only msg lines with no F7 lts will result.
63a10	= Reconfig htrs per BUS LOSS SSR. If pri ESS bus lost, other alarms will be present. If not pri bus, subbuses are: TK1: ESS2CA O13 & R15; TK2: ESS1BC O13 & R15; TK3: ESS31B ML86B; TK4: ESS1BC ML86B
63a11	= is TK3 and/or TK4 the affected tk?
63a12	= are TK3 and TK4 depleted, QTY < 10 ?
63a13	= operate on TK3 and TK4 until QTY < 10 then: (R1) H2 TK3 HTRS A,B (two) - OFF and (A11) TK4 HTR H2 A,B (two) - OFF
63a14	= Is cntlr cb of affectk (TK1 and/or TK2) on Pnl O13 open?
63a15	= Possible electrical Problem. Do not attempt to reset cb.
63a16	= Is CNTLR cb of affect tk on Pnl ML86B open?
63a17	= Continue to monitor. Do TK3 and TK4 Htrs cycle on when pressure in both TKs= 217-223 psia?
63a18	= P transducer failed low. Continue to operate TK3 and TK4 in auto.
63a19	= Possible electrical problem. Do not attempt to reset cb.
63a20	= PWR failure in affected htr cntlr.
63a21	= Deact htrs in affected tk(s). (R1) H2 TK3 Htrs A,B (two) - OFF and/or (A11) TK4 HTR H2 A,B (two) - OFF

Table 3-1: Some Semantic Values of Zero-Order Features for CRYO6.3a

```
6666666666666666666666666666666666666666666  666666666666666666666666666666666
3333333333333333333333333333333333333333333  333333333333333333333333333333333
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1111356891111111122222233333333344444444558  124679111112222223333333444445555
         123456790235690123445680125679902         045890145780356789234780123
Eabc                       ab          ab    E
-------------------------------------------  ---------------------------------
1-1----------------1-00-11-----------------  000000000000000000010000000000000
```

Table 3-2: A Single Rule from the CRYO 6.3a First-Order Table

```
63a1E = Hydrogen Pressure Sensor Error
              (more than one indicator set)          and
63a1b = Hydrogen Pressure High                        and
63a23 = Deactivate htrs in affected tks.
               TK3 and/or TK4/5 affected.             and
63a26 = TK3,TK4/TK5 htrs deactivated.
           All htr switches off when problem occurred.and
63a29 = Pressure in both TK3, TK4/5 > 293.8 psia.     and
63a31 = Both P and TK P of affected tk high.          and
63a32 = MANF Ps agree with P and TK P of affected TK  and

63a35 = Diagnosis: AUTO PRESSURE CONTROL OR RPC FAILURE.
```

Table 3-3: Retrieval of Semantic Values of Zero-Order Features Table 3.3.4

3.3.4. Propagation of Status Vectors Between Levels

As previously stated, the tables of first-order features are in DNF and, this structure is preserved throughout the second- and third-order features. A very useful result of this form is that the *state table* can be summarized into a single, time-independent *status vector* which can be propagated across the *levels of abstraction* both as a means of communication as well as input data to higher level processing.

3.3.4.1. Boolean Summation

This process of summarization is called *Boolean Summation* is accomplished by simply OR-ing the rows of the status table into a single status vector. This results in a single status vector for each system discipline as depicted for three particular disciplines in figure 3-3. Figure 3-3 demonstrates the receipt, at level 5, of an input status vector. After processing by FSCAN/BSCAN, an *updated status vector* is generated which is the result of *Boolean Summation*.

3.3.4.2. Constructing State Tables from Summed Status Vectors

The summed status vectors represent all known information about system and subsystem states to the extent that telemetry parameters have been mapped into first-order features. In general, these vectors are time-tagged to permit temporal considerations to be introduced both in real-time analysis and rule-synthesis. Summed status vectors cannot preserve time-information since the are the result of summation over some time-window, represented in figure 3-3, as the state tables. These status vectors can, however, be assembled into higher-level, time-independent *state tables*. This process is discussed in chapter 5. An overview of the use of these time-independent, state tables is shown in figure 5-1.

3.4. Non-Monotonic Reasoning

If we consider monotonic reasoning to be the methods of reasoning by which the cardinality of the set of theorems derivable from a given set of axioms is a monotonic, increasing function of the cardinality of the set of axioms [Barr 81], much of the confusion about the nature of non-monotonic reasoning can be eliminated. For example, if we consider the entries in the state table of figure 3-2 to be axioms, then

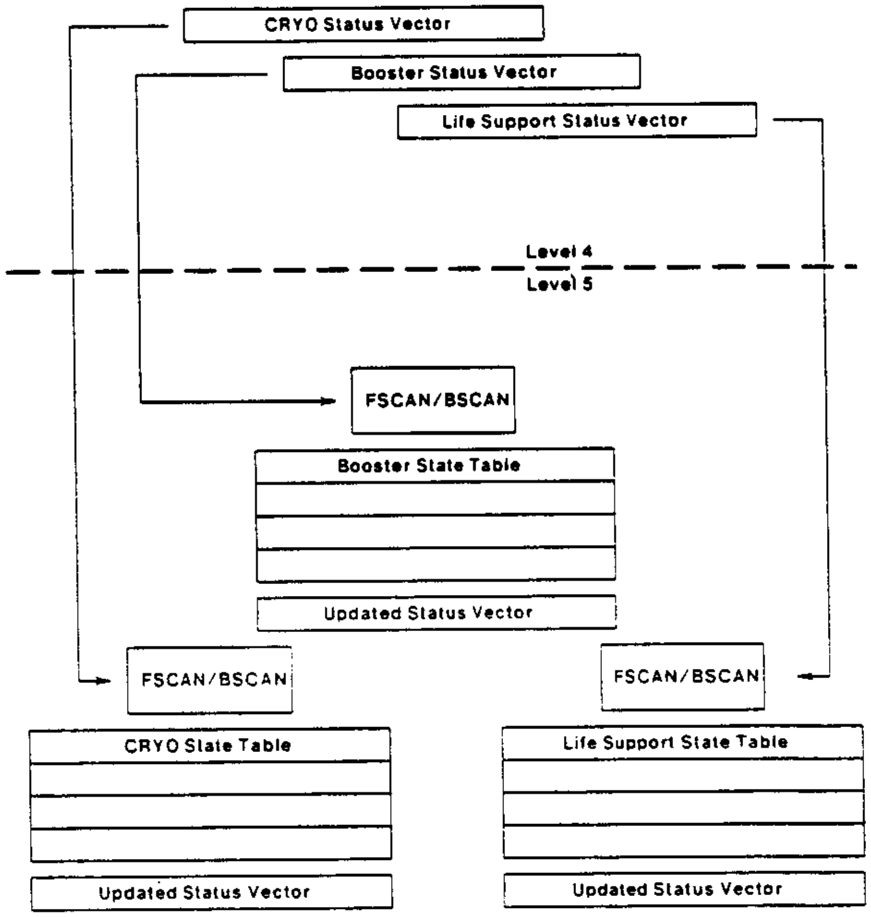


Figure 3-3: Propagation of Status Vectors Between Levels

the theorems, derivable from these axioms, correspond to the *diagnoses* implicit in the set of table entries. This is the same thing as saying that the set of axioms *implies* the set of theorems.

3.4.1. Logical Inconsistency and Imperfect Data

Brief reflection on the possibility of randomly occurring, non-mutually exclusive failures, suggests that the addition of entries to the state table may either increase *or* decrease the possible *diagnoses* with time as the table grows. This is due to the fact that, barring logical inconsistencies in the first-order tables, the second-order table may contain logical inconsistencies as the result of imperfect data or incomplete system knowledge reflected in the first-order tables. One consequence could be the introduction of an additional second-order feature, or axiom for the purpose of this argument, would increase the number of possible *diagnoses* while in another situation, the addition of a second-order feature would decrease the number.

3.4.2. System Reset as Non-Monotonic Reasoning

Another, perhaps more common, conception of non-monotonic reasoning is that of *backtracking* during a search for a solution or *proof*. In fact, this situation is essential to the real-time processing of malfunctions since it is necessary to be able to restore *partial* or *full* function to a previously failed or degraded system or subsystem and that the restoration be reflected in subsequent analyses. This corresponds to backtracking, or *non-monotonic reasoning*, in the sense that during the entire processing which leads up to the *state* at time *t*, the expert reasoning process has been *watching* a certain logical path through the entire space described by the first-order feature set. When this process identifies a correctable failure, this corresponds to a *fault state* which is equivalent to the state table structure at that time. When the identified fault is full or partially corrected it is as if the finite-state machine, implemented by the state table, were *reset* to a non-fault state. In this sense, *non-monotonic* reasoning is equivalent to a *full* or *partial* reset.

3.4.3. Implementing N-M Reasoning with the PLA Model

Using a pattern-matching mechanism similar to that of BSCAN and FSCAN, the state-table entry which triggered the diagnosis, or fault state transition, is found in the state-table. There are two methods of handling full and partial resets which do not require altering the form of the table by using, for example, extra bits to indicate deletion.

3.4.3.1. Entry Deletion

One method is to delete the offending entry from the table. Note that, since the table is time-ordered, this has no effect on the chronology and, therefore, the implicit causality of events but alters the record of the time-sequential events. This can, of course, be overcome by the maintenance of a separate log as a record. This is an important issue for analyzing system performance after a mission, for example, and/or the appropriateness of human or machine corrective action to a fault but it is more advantageous to keep the state-table small so that the deletion of corrected failures is a desirable method.

3.4.3.2. Set to Null

Setting the corrected failures to *don't care* conditions in the state-table entry, using the same pattern matching facility to locate the proper entry used above, is another approach although there is little to recommend it. Once reset has been accomplished, the table entry contains no information since all its positions are set to *don't cares*. The table grows monotonically larger. A variation on this theme would be to keep the table entry around and selectively reset first-order features as they are corrected until the entry is entirely nulled. This is inconsistent with the simple notion that each state-table entry is identical to that of a first-order table. In addtion to violating that ground-rule it adds a substantial amount of housekeeping to pattern-match for partial resets which require deletion and is significantly more difficult to program and maintain.

Chapter 4
Rule-Synthesis

The goal of rule-synthesis is to articulate the heuristics used at levels 2 and 3 depicted in figure 3-1 through the rule-base structure of chapter 2. This can be thought of as a methodology for the reduction of human participation in the malfunction detection and analysis process as well as *a method to extend the scope of the degree of determinism, realized at levels 4 and 5, to levels 2 and 3*. There are two basic types of *rule-synthesis* which are important to the processing of malfunctions. The first type we will call *time-correlation* and the second type, *structural integration*.

4.1. Static and Dynamic Rule-Synthesis

This division of types of learning reflects a recurring theme in this work. Time-correlation addresses the problem of the identification of *causal sequences* of predefined events. Structural integration, on the other hand, addresses the problem of identifying *hidden*, or *buried* failure modes. Since learning can take many forms, we limit the scope of our discussion to the *generation of new, first-order features*, and we will call this *rule-synthesis*. All subsequent discussion of learning will be made in this context. Rule-synthesis via time-correlation and structural integration are both based on the methodology of *fractional-domain features* which is introduced below.

4.2. Fractional-Order Domain Features

Fractional-order features are based on a relaxation method for the generation of new equivalence classes from the set of first-order features. Our use of equivalence classes was described in section 2.6. We extend this concept to permit us to make use of the imperfect information contained in our rule-base to construct new rules and to *permit the monitoring of incipient malfunctions* in a manner which is analogous to the concept of *pipelining* used in some computer architectures [Tannenbaum 76].

4.2.1. Relationship to First- and Second-Order Domain Features

The construction of the second-order features requires *complete-matching* of the first-order features to the input, status-vector as depicted in figure 3-2. If we define second-order features to be the set resulting, in real-time, from a *relaxation factor* of zero, then we are able to define new equivalence classes. These new classes are *intermediate* between the set of static, first-order features and the dynamic, second-order features and correspond to *relaxation factors* greater than zero. This relationship is depicted in figure 4-1.

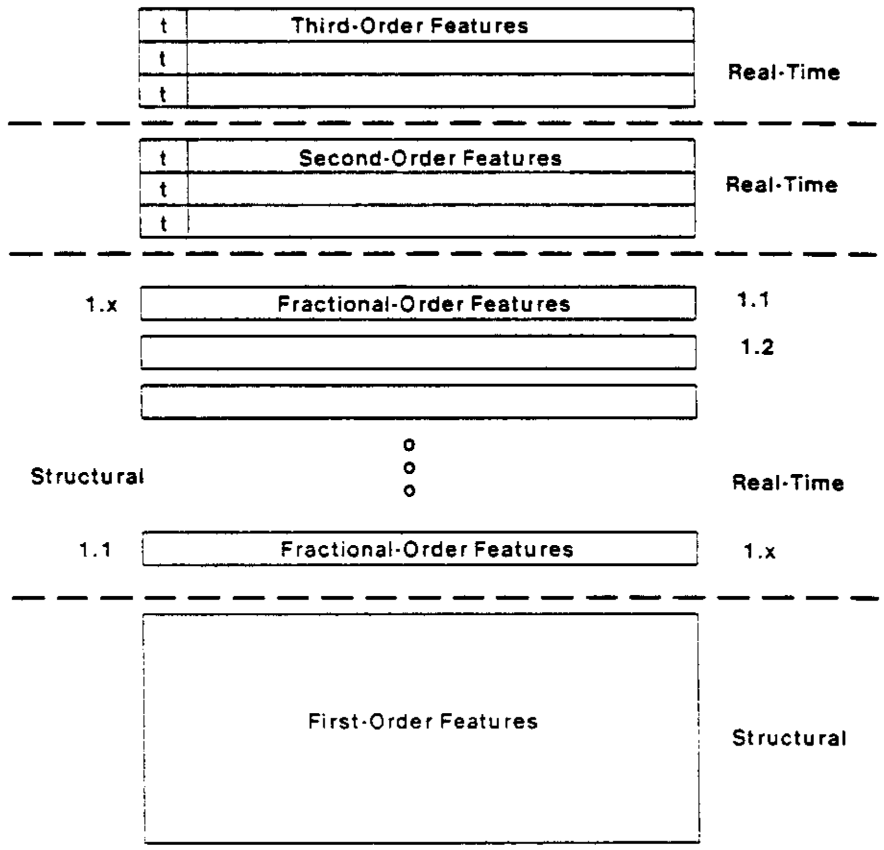


Figure 4-1: Ordering of Domain Features

4.2.2. Relaxation by Partial-Matching

Partial matching is a *relaxation* technique which is implemented using a *relaxed version* of **FSCAN**. The *degree of relaxation* is treated as a global search parameter which sets the *minimum* number of antecedent terms which must be matched in any given first-order feature for that that feature be copied into the resultant, new, equivalence class composed of the set of *partial-matches*. This process also suggests a metric for assessing the *goodness-of-fit*, a concept from statistics, of the first-order feature to the system status.

4.2.3. Relaxation Factor Determines Fractional-Order of Partial-Match

Figure 4-1 shows how *partial-match equivalence classes* relate to first and second-order features. The equivalence class corresponding to a relaxation factor of *one* contains all first-order features whose antecedent terms match the status vector on all but one term (zero-order feature). Likewise, the equivalence class corresponding to a relaxation factor of two is composed of all first-order features whose antecedent terms match the status vector on all but two zero-order features. Note that, in general, these new equivalence classes have members which compose first-order equivalence classes. Since these equivalence classes are intermediate between first and second-order feature sets we refer to them as *fractional-order features* using a notation similar in appearance to real numbers but with a different interpretation.

4.2.4. Notation of Fractional-Orders and Problems of Scale Normalization

For the fractional-order set of feature, or equivalence class, with relaxation factor of one, we write 1.1; for relaxation factor of two, we write 1.2 and so on. In general, we write *1.<relaxation factor>*. Note that this is merely a naming procedure and not a scale. Since the number of antecedent terms contained in first-order features is theoretically unbounded there is no way to normalize the relaxation factors into a scale without other constraints on the number of fractional-order, equivalence classes possible. Also note, that the fractional-order of a class does not convey any information about the *severity* or *criticality* of that class or members of that class in the sense of *next-worst failure*.

4.3. Rule-Synthesis by Time-Correlation

Time-correlated learning is probably the conceptually simplest to understand. Since, in general, we are dealing with systems which are continuous in time but which have been sampled or *discretized*, for convenience we will first describe the rationale behind the method using the notation of continuous function. Later, we will show how this is mapped into the domain of the first-order features described previously.

4.3.1. Relationship to Next-Worst Failure Analysis

Time-correlated, fractional-order generation is similar, but not identical to, *next-worse failure analysis*. The major difference is that there are no weights provided to permit the quantitative assessment of the qualitative concept of *next-worst*. However, structural integration facilitates the construction of an ordered list of *incipient failures* based on the *completeness* of the partial-match of the antecedent conditions. This is more powerful than the next-worst type of analysis in the sense that the controller is provided with a more complete picture of where any given system stands, in *functional adjacency*, to another failure.

4.3.2. Multiple Dependent Failures

A basic premise to the construction of the Boolean expressions, which define the first-order features of the rule-base, is that each expression is a stand-alone, *path through the digraph* of the malfunction procedure. A consequence of this fact is that *any first-order feature can be removed without disturbing the rest of the rule-base* since there are no *implicit* interdependencies between the expressions. More precisely, from any set of Boolean expressions it is possible to generate another set, equivalent to the first, such that no *zero-order feature*, which appears in the set as a consequent, appears in the set as an antecedent. If this were not the case then it would be possible that the deletion of a first-order feature would make another first-order feature non-executable since the deleted consequent would be contained in the antecedents which are, by definition of DNF, in conjunctive form. However, it is important to remember that the set of first-order features is only an incomplete model of the physical system. It commonly occurs that the set of second-order features, generated in real-time, are in fact related; sometimes in ways that were not known or realized until they were observed to occur together. Since the we are fundamentally concerned, in malfunction processing, with failures we refer to these *related, second-order features* as *multiple, dependent failures*.

4.3.3. Third-Order Features Are Time-Dependent

The mere inclusion of two, or more, first-order features in the second-order set does not mean that those features are interdependent. Furthermore, we have insured, by means of conversion DNF as well as via the integrity constraints described in section 2.1.3, first-order features which are mutually independent. The result of this is that the only possible relationship between members of the set of second-order features, which is *not* merely the consequence of a structural, first-order relationship discussed in section 4.4, is that of a *time-correlation*. Stated another way we can say that it is probable that there are *third-order features* which are functions of *second-order features* and *time*. Such third-order features are a special case of multiple, dependent failures since the dependency is via time in the sense of ...*if A and, then B, then C*. We would say that *(A and then B)* is a third-order feature rather than A is a third-order feature and B is a third-order feature.

4.3.4. A Third-Order Temporo-logical Operator

Third-order logical operators can be constructed very simply using the explicit time-tag attached to second-order features and simple logical operators. An example of this is the *THAND*, *then-and*, which can be used to evaluate the sequential nature of on otherwise simple conjunctive expression like *if A and B then C*. If A and B have an order-dependency (*if A and, then B, then C*), this can be restated *if A thand B then C*. Parentheses can be used to control the precedence order for the purposes of evaluation making it possible to subset conjunctive expressions based on requirements for time-sequential occurence.

4.3.5. Identification of New Rules through Empirical Analysis

Time-correlation is fundamentally an empirical method for the synthesis of new rules. It is, in a sense, rule-synthesis by example if one allows that empirical observation may be described *as example*. Construction techniques range from simple pairing of events; as in A *always* is observed before B, to sophisticated methods of cluster analysis. This is the most promising approach since it is a multivariate, statistical technique suited to the analysis of large datasets. The set of second-order features over the course of a single mission may be analyzed for recurrent failures whose time-relationship remains obscure under the current methods of mission management as well as the analysis of system behavior between missions. This approach offers the potential for detection of previously unrecognized *failure modes* which can then be represented as *new equivalence classes*.

4.4. Rule-Synthesis by Structural Integration

Structural integration corresponds to the construction of new, equivalence classes through the process of *partial matching* of the antecedent conditions of first-order features *during a a systems analysis*. This analysis corresponds to processing the entire first-order rule-base for the purpose of detecting implied relationships which have been previously unrecognized, or otherwise obscured, by the compartmentalization of system disciplines. Structural integration is based on the method of fractional-orders but does not incorporate, nor depend on, information contained in the real-time generated *status vector*. It is a static form of learning as opposed to *time-correlated rule-synthesis* which is based on fractional-orders determined by the *status vector* and is, therefore, a dynamic method of rule synthesis.

4.4.1. Fractional-Orders in Structural Integration

The philosophy of structural integration is to *grow the tree* up from Level 5 in the direction of Level 0. This is accomplished using equivalence classes generated by the method of fractional-orders in a manner *different than that used for time-correlation*. The difference is simple but quite important.

4.4.1.1. Difference Between Time-Correlation and Structural Integration

The fractional-order generated by *time-correlation* constructs equivalence classes which are similar in the sense of requiring satisfaction of the number of antecedent terms equal to the fractional-order for the first-order feature to be mapped into the set of second-order features. On the other hand, the equivalence classes generated by *structural integration* have their fractional-orders assigned as a function of the *commonality* of antecedent terms between first-order features. For example, consider the first-order features below taken from table 2-1,

$$63a52 = 63a49a \wedge \sim 63a45 \wedge \sim 63a40 \wedge \sim 63a23 \wedge 63a1b$$
$$63a52 = 63a49a \wedge \sim 63a45 \wedge 63bx8$$

A fractional-order of 1.2 is assigned to the equivalence class $63a49a \wedge \sim 63a45$ since this class has a commonality of two. Note that this interpretation of fractional-order does not require that the two zero-order features be the same for all equivalence classes with equal fractional orders. This means that there can be many equivalence classes with the same fractional-order but with commonality on different zero-order features within an equivalence class.

Equivalence Class: $63a49a \wedge \sim 63a45$

$63bx8$

$\sim 63a40 \wedge \sim 63a23 \wedge 63a1b$

4.4.1.2. Structural Integration Produces Meta-States Rules

The creation of new equivalence classes changes the meaning of our earlier use of equivalence classes with respect to the interpretation their logical meaning. The equivalence class, $63a49a \wedge \sim 63a45$, does not correspond to the implication $63a52 = 63a49a \wedge \sim 63a45$ since the antecedents are not complete. These new equivalence classes really are *meta-states* without a convenient name but with, obviously, similar dependencies within the physical system they correspond to. Their usefulness is derived from the information they provide about commonality within the system they correspond to. The meta-states require the introduction of new zero-order features to maintain consistency with the implicational interpretation of first-order features.

Chapter 5
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. In particular, this approach makes it practical to distribute processing function in ways previously prohibited by technology.

5.1. Characteristics of Distributed Architectures

Distributed processing architectures have general characteristics 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 [Davies 83], 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.

5.2. 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 (transmitted/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.

5.3. 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 [JSCDPS 84]:

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.

5.4. 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 microcomputers*. 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*.

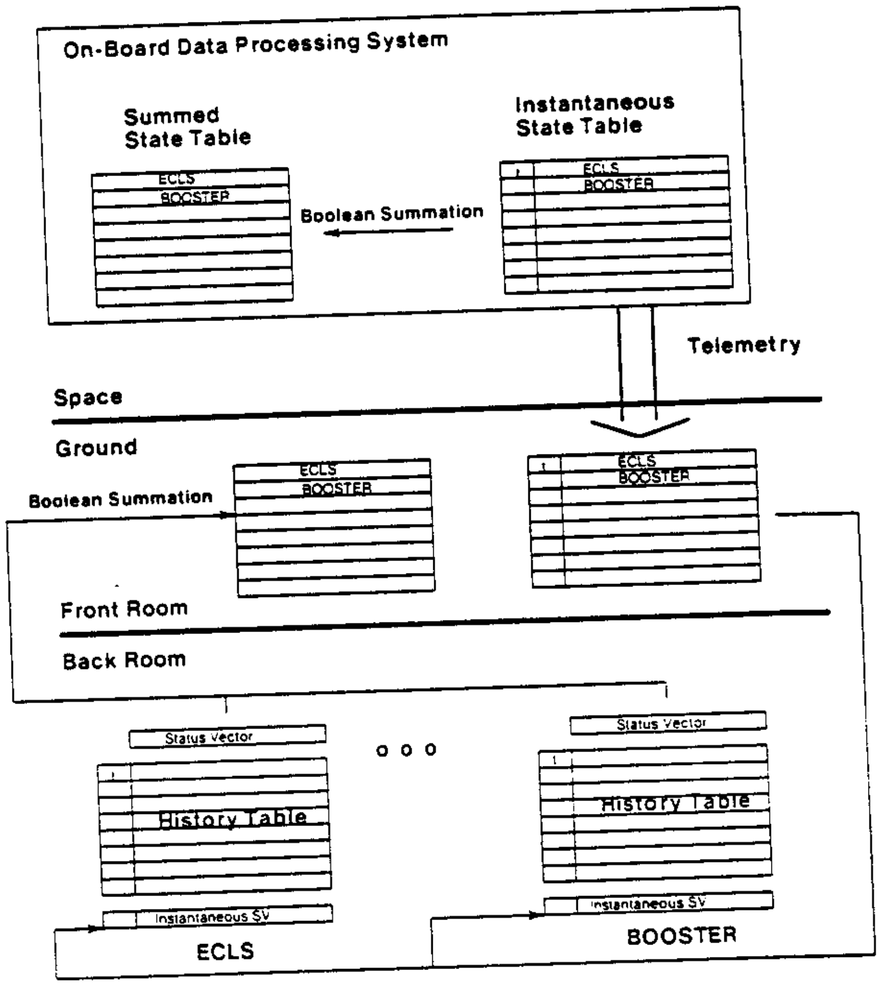


Figure 5-1: Prototype Distributed Processing Architecture

5.5. Architecture and Interconnections

The prototype architecture depicted in figure 5-1 makes use of the basic first-order feature construct as its fundamental, or *atomic* piece of information. It is, however, manipulated in a number of ways some of which, for example the *status vector*, were introduced in section 3-2. The architecture itself introduces some new arrangements of the first-order features which are discussed below. The organization of these arrangements are driven by the boundary thresholds represented by the heavy, horizontal lines separating *space and ground* as well as *front room and back room* in figure 5-1.

5.5.1. Major Distributed Components

Before discussing the rationale for these boundaries we introduce major constructs and processing used in the architecture; the *summed state table*, the *instantaneous state table*, *Boolean summation*, and the *history table*. These are the major components used to organize and distribute information within the system.

5.5.1.1. Instantaneous State Table

This table represents the collection of *time-tagged, status vectors*. Each row in the table is the *status vector* of the first-order features for a single, SSV system or subsystem. This is represented in figure by the labels for two of the systems, ECLS and BOOSTER. In the complete SSV table there would also be rows for all of the other flight control disciplines as depicted in figure 1-1. This table is instantaneous because it represents a *time-tagged sample* for a specific time, *t* and is *memory-less* in that it is re-written every time SSV systems are polled. The polling process is equivalent to a digitization process and results in the setting of the first-order features in the status vector for each table row.

5.5.1.2. Summed State Table

This table is the result of the logical-OR of all occurrences of the instantaneous status vectors, by discipline, over time. Because the rows of this table are Boolean sums over time, time-tags are dropped. The result is a state table which summarizes the current condition of each system and subsystem regardless of when any given zero-order feature was set. It is important that the values of this table be explicitly reset when a condition corresponding to an antecedent, and its consequent, is reset since the process for generating the row entries is a simple OR-ing with the instantaneous state table. This requires a separate process which is not depicted in figure 5-1.

5.5.1.3. Instantaneous Status Vectors

These are the row entries of the instantaneous status tables. These vectors are communicated to the ground via telemetry. *They are the primary method of communicating system status information.* These vectors are collected in a common table, a copy of the instantaneous state table on-board, for distribution to the responsible back-room positions.

5.5.1.4. Summed Status Vectors

These status vectors, like the summed state tables, are independent of time and, consequently, do not retain time-tags. They are constructed by the logical-ORing of the history tables, which are themselves composed of instantaneous status vectors. The summed status vectors are used to maintain, through Boolean summation, an independent, *ground-generated, state table.*

5.5.1.5. History Tables

These tables are the complete record of the history of the SSV states and are critical to the process of *time-correlated rule synthesis* as well as to the determination of all time-dependent processes. These are essentially a log corresponding to a complete time-history of system and subsystem states. They are available for both real-time and non-real-time processing and analysis.

5.5.2. Space-Based Processing

Space-based processing is focused on the generation and parameterization of sensor and other instrumentation data. Because the size of the data processing system on-board a spacecraft is limited by both weight and volume it is best if the amount of storage on-board is kept to a minimum. On the other hand, time-delays. obscuration, loss-of-contact events all make it desirable for the system status to be *determined*, through the computation of *instantaneous status vectors* on-board for transmission to the ground. It is also desirable to keep a *local copy* of the spacecraft *summed state table* on-board for two reasons. First, it makes possible the implementation of self-monitoring, and self-correcting functions without dependence on the ground. These functions may be limited, at least initially, to the processes described as *real-time malfunction processing* in section 3-2 and not extend to rule-synthesis due to the additional processing and validation required to develop *reliable* new rules. This may be ultimately desirable, however, for long-lived spacecraft such as those transitting deep-space.

5.5.3. Ground-Based Processing

The development of nearly continuous communications makes it possible to view the ground-based data processing system, the system used by flight controllers, as the *host computer system* and that of the spacecraft, in a sense, *as a user*. Although most of the monitoring and analysis is performed on the ground, if the spacecraft is capable of maintaining the *summed state-table* and generating *instantaneous status vectors* then the ground system can evolve into a *development environment* for the synthesis of new rules which can be incorporated into the on-board rule base using the methods described in chapter 4. Given this kind of an architecture, what is then in place is a kind of *knowledge-funnel* for the organization and articulation of flight controller expertise developed with the aid of the extensive computing resources available on the ground, translation into *first-order form* and incorporation into the rule-base of the spacecraft. The system can evolve during the course of many missions in the case of the SSV, or during the lifetime of the spacecraft for other types of vehicles.

Chapter 6
Summary and Conclusions

This work has presented an integrated approach to the development of an expert system for *real-time malfunction processing*. The fundamental new concept is the use of the programmable logic array (PLA) as a data structure incorporating a *disjunctive normal form* (DNF) of logical information. This technique represents a new type of knowledge compilation which takes advantage of advances in VLSI research and suggests the future ability to actually build a custom, hardware, expert-system using techniques similar to the ones represented here. There are, at least at the present, few *expert systems* which have attempted real-time processing of data although some of the work which is currently underway in the field seismic analysis for oil-exploration is focused in this direction.

6.1. Utilization of Existing Knowledge

This application is apparently one of the first in the field of expert systems to take advantage of a major source of existing, documented human expertise. This puts this system in position to make major experimental advances through development and operation. Not only is there a wealth of existing information for the development of the first-order rule-base, but there is also a wealth of observational data with which to test and operate the system as a result of the well-developed data processing environment for Space Shuttle operations. The difficulty of extracting human expertise. as well as observational data is significantly reduced given the opportunity to establish an initial rule-base of malfunction procedures.

6.2. Future Extensions

The mechanism introduced here for processing time as an explicit zero-order feature points out, indirectly, the power of *semantic attachment* which was also the mechanism for attaching humanly interpretable meanings to the zero-order features. The attachment of other zero-order features to consequents and antecedents of first-order features provides a mechanism for incorporating the *particular semantics* necessary for many types of analysis. An example of this is the *next-worse failure* problem. The process of *semantic attachment* permits the association of a zero-order feature containing the *criticality value* of that first order feature through a completely natural extension to the use of semantic attachment for the association of English text. This is a very powerful result since these zero-order features can be stored, for example, as attributes in a relational database.

6.3. Uses for System Design

This methodology was developed largely as a means of supplanting the limited human resource represented by the flight control team. It has become apparent in the course of the work, however, that this approach, has great promise in the area of system design. Consider that the use of *fault equivalence classes* provides a mechanism for describing existing failure modes through the implementation of malfunction procedures. If a schematic diagram, which can be frequently represented using logical operators, could be implemented using the PLA representation developed here, then the opportunity exists for the analysis and monitoring of failure modes (equivalence classes) which are introduced into the system design during development. This appears to be a powerful method of controlling and documenting the failure modes of systems during the development cycle which simultaneously establishes the rule-base for use in operational monitoring of the completed system.

Appendix A
ROSIE Implementation of CRYO63a

A.1. 63aRHS

[: 63arhs parsed Tue Oct 11 17:50:38 1983 by Helly :]

To 63arhs:

[1] send {return,"* **
*",return}.

[2] send {"* ECLS 6.3a CRYO H2 PRES OV102/BAS E: Malfunction Proc (pg. 6-30)", return}.

[3] send {"* ** *",return}.

[4] if H2 Press is true obtain 63ax1a of "H2 P Normal".

[5] if 63ax1a is true go 63alhs.

[6] if 63ax1a is false obtain 63ax1b of "H2 P High".

[7] if 63ax1b is true obtain 63ax23 of "TK3 and/or TK4 the affected tk".

[8] if 63ax1b is false obtain 63ax1c of "H2 P Low".

[9] if 63ax1c is true obtain 63ax3 of "Press in all tks < 153 psia".

[10] if 63ax3 is true go 63alhs.

[11] if 63ax3 is false obtain 63ax5 of "Both P and TK P of affected tk low".

[12] if 63ax5 is true go 63alhs.

[13] if 63ax5 is false obtain 63ax8 of "Received O2 PRESS Alarm and/or S68 CRYO H2 PRES and S68 CRYO 2 PRES msg lines".

[14] if 63ax8 is true go 63alhs.

[15] if 63ax8 is false obtain 63ax11 of "TK3 and/or TK4 the affected tk".

[16] if 63ax11 is true obtain 63ax16 of "CNTLR cb of affected tk on Pnl ML868 open".

[17] if 63ax11 is false obtain 63ax12 of "TK3 and TK4 depleted, QTY < 10%".

[18] if 63ax12 is true obtain 63ax14 of "CNTLR cp of affect tk (TK1 and/or TK2) on Pnl 013 open".

[19] if 63ax14 is true go 63alhs.

[20] if 63ax16 is false obtain 63ax17 of "TK3 and TK4 Htrs cycle on when press in both tks = 217-223 psia".

[21] if 63ax17 is true or 63ax17 is false go 63alhs.

[22] if (63ax23 is true or 63bx7 is true) obtain 63ax26 of "TK3 and TK4 htrs were deactivated (all htrs switches in OFF when problem occurred)".

[23] if 63ax23 is false obtain 63ax40 of "Pressure in both TK1 & TK2 > 293.8 psia".

[24] if 63ax26 is true go 63alhs.

[25] if 63ax26 is false obtain 63ax29 of "Press in both TK3 and TK4 > 293.8 psia".

[26] if 63ax29 is true go 63alhs.

[27] if 63ax29 is false obtain 63ax31 of "Both P and TK P of affected tk high".

[28] if 63ax31 is true obtain 63ax32 of "MANF Ps agree with P and TK P of affected TK".

```
[29] if 63ax31 is false obtain 63ax34a of "P High".

[30] if 63ax34a is false obtain 63ax34b of "TK P High".

[31] if 63ax34a is true or 63ax34b is true go 63alhs.

[32] if 63ax32 is true or 63ax32 is false go 63alhs.

[33] if (63ax40 is true or 63bx8 is true) obtain 63ax41 of "Alarm on next htr cycle".

[34] if 63ax41 is true or 63ax41 is false go 63alhs.

[35] if (63ax40 is false or 63bx8 is false) obtain 63ax45 of "Both P and TK P of affected tk high".

[36] if 63ax45 is true obtain 63ax46 of "MANF Ps agree with P and TK P of affected tk".

[37] if 63ax46 is true or 63ax46 is false go 63alhs.

end.
```

A.2. 63aLHS

[: 63alhs parsed Thu Oct 6 01:19:31 1983 by Helly :]

To 63alhs:

[* ** *] [* ECLS 6.3a CRYO H2 PRES *] [* computes truth/falsehood of left-hand side (lhs) *] [* and displays the nominal assumptions of the malf. *] [* ** *]

[1] if (63ax1a is true) assert 63ax2 is true.

[2] if (63ax1c is true and 63ax3 is true) assert 63ax4 is true.

[3] if (63ax1c is true and 63ax3 is false and 63ax5 is true) assert 63ax6 is true.

[4] if (63ax6 is true) assert 63ax7 is true.

[5] if (63ax1c is true and 63ax3 is false and 63ax5 is false and 63ax8 is true) assert 63ax9 is true.

[6] if (63ax9 is true) assert 63ax10 is true.

[7] if (63ax12 is true or (63ax12 is false and 63ax13 is true)) assert 63ax14 is true.

[8] if (63ax1c is true and [Possible electrical problem] 63ax3 is false and 63ax5 is false and 63ax8 is false and 63ax11 is false and 63ax12 is true and 63ax14 is true) assert 63ax15 is true.

[9] if (63ax1c is true and [P 63axducer failed low] 63ax3 is false and 63ax5 is false and 63ax8 is false and 63ax11 is true and 63ax16 is true and 63ax17 is true) assert 63ax18 is true.

[10] if (63ax1c is true and 63ax3 is false and 63ax5 is false and 63ax8 is false and 63ax11 is true and 63ax16 is true) assert 63ax19 is true.

[11] if (63ax1c is true and 63ax3 is false and 63ax5 is false and 63ax8 is false and 63ax11 is true and 63ax16 is true and 63ax17 is false) assert 63ax20 is true.

[12] if (63ax19 is true or 63ax20 is true) assert 63ax21 is true.

[13] if (63ax22 is true and 63ax14 is false and 63ax12 is true and 63ax11 is false and 63ax8 is false and 63ax5 is false and 63ax3 is false and 63ax1c is true) assert 63ax24 is true.

[14] if (63ax22 is false and 63ax14 is false and 63ax12 is true and 63ax11 is false and 63ax8 is false and 63ax5 is false and 63ax3 is false and 63ax1c is true) assert 63ax25 is true.

[15] if (63ax26 is true and ((63ax23 is true and 63ax1b is true) or (63bx7 is true))) assert 63ax27 is true.

[16] if (63ax25 is true or 63ax15 is true) assert 63ax28 is true.

[17] if ((63ax1b is true and 63ax23 is true) or (63bx7 is true)) and (63ax26 is false and 63ax29 is true) assert 63ax30 is true.

[18] if (63ax32 is false and 63ax31 is true and 63ax29 is false and 63ax26 is false and 63ax23 is true and 63ax1b is true) assert 63ax33 is true.

[19] if (63ax32 is true and 63ax31 is true and 63ax29 is false and 63ax26 is false and 63ax23 is true and 63ax1b is true) assert 63ax35 is true.

[20] if (63ax34b is true and 63ax31 is false and 63ax29 is false and 63ax26 is false and 63ax23 is true and 63ax1b is true) assert 63ax36 is true.

[21] if (63ax30 is true or 63ax33 is true or 63ax35 is true or 63ax38 is true) assert 63ax37 is true.

[22] if (63ax34a is true and 63ax31 is false and 63ax29 is false and 63ax26 is false and 63ax23 is true and 63ax1b is true) assert 63ax38 is true.

[23] if (63ax36 is true) assert 63ax39 is true.

[24] if (63ax41 is true and 63ax40 is true and 63ax23 is false and 63ax1b is true) assert 63ax42 is true.

[25] if (63ax41 is false and 63ax40 is true and 63ax23 is false and 63ax1b is true) assert 63ax43 is true.

[26] if (63ax42 is true or 63ax47 is true) assert 63ax44 is true.

[27] if (63ax46 is true and 63ax45 is true and 63ax40 is false and 63ax23 is false and 63ax1b is true) assert 63ax47 is true.

[28] if (63ax46 is true and 63ax45 is true and 63bx8 is true) assert 63ax47 is true.

[29] if (63ax46 is false and 63ax45 is true and 63ax40 is false and 63ax23 is false and 63ax1b is true) assert 63ax48 is true.

[30] if (63ax46 is false and 63ax45 is true and 63bx8 is true) assert 63ax48 is true.

[31] if (63ax49b is true and 63ax45 is false and 63ax40 is false and 63ax23 is false and 63ax1b is true) assert 63ax50 is true.

[32] if (63ax49b is true and 63ax45 is false and 63bx8 is true) assert 63ax50 is true.

[33] if (63ax50 is true) assert 63ax51 is true.

[34] if (63ax49a is true and 63ax45 is false and 63ax40 is false and 63ax23 is false and 63ax1b is true) assert 63ax52 is true.

[35] if (63ax49a is true and 63ax45 is false and 63bx8 is true) assert 63ax52 is true.

[36] if (63ax52 is true) assert 63ax53 is true.

[37] go 63aexp. [provide explanation and diagnostics]

end.

A.3. 63aEXP

[: 63aexp parsed Sun Oct 30 00:53:10 1983 by Helly :]

To 63aexp:

[* ** *] [* explains current global database state *] [* ** *] [1] send{ ">> ECLS 63.a : Summary and Diagnostics >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>",return}.

[2] send {return,"* NOTE: The following nominal conditions are assumed:",return}.

[3] send {"* Backup C/W Alarm light on if:",return, " P > 293.8 psia",return, " P < 153 psia",return}.

[4] send {"* NOMINAL CONFIGURATION:",return, " (013:D) cb ESS 2CA CRYO CNTLR H2 TK1 - c1",return, " cb ESS 2CA CRYO QTY H2 TK1 - c1",return, " (013:B) cb ESS 1BC CRYO CNTLR H2 TK2 - c1",return, " cb ESS 1BC CRYO QTY H2 TK2 - c1",return, " (ML868:G) cb ESS 3AB CRYO CNTLR H2 TK3 - c1",return, " cb ESS 3AB CRYO QTY H2 TK3 - c1",return, " cb ESS 1BC CRYO CNTLR H2 TK4 - c1",return, " cb ESS 1BC CRYO QTY H2 TK4 - c1",return}.

[5] send {"* (R1) H2 MANF VLV TK1,2 - ctr (tb - OP)",return}.

[6] send {"* If TK qty > 10% then:",return, " (R1) H2 TK 1,2,3",return, " (A11) CRYO TK4 H2 HTR A - OFF",return, " B - OFF",return}.

[7] send {"* If TK qty < 10% then:",return, " (R1) H2 TK 1,2,3",return, " (A11) CRYO TK4 H2 HTR A - OFF",return, " B - OFF",return}.

[8] if 63ax1a is true send {"* All H2 P norm",return}.

[9] if 63ax1b 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 63ax1c is true send {"* H2 P low",return}.

[11] if 63ax2 is true send {"* DIAGNOSIS: C/W Failure",return}.

[12] if 63ax3 is true send { "* Press in all tks is less than 153 psia",return}.

[13] if 63ax3 is false send { "* Press in all tks is greater than or equal to 153 psia",return}.

[14] if 63ax4 is true send { "* DIAGNOSIS: System leak. Execute ECLS SSR-1 (7)",return}.

[15] if 63ax5 is true send { "* Both P and TK P of affected tk are low",return}.

[16] if 63ax5 is false send { "* P and TK P of affected tk are not both low",return}.

[17] if 63ax6 is true send {"* DIAGNOSIS: Leak between affected TK and check valve. Leak cannot be isolated",return}.

[18] if 63ax7 is true send {"* ACTION: Deact htrs in affected tk.",return, " (R1) H2 TK1(2,3) HTRS A,B - OFF",return, " (A11) CRYO TK4 HTR H2 A,B - OFF ",return}.

[19] if 63ax8 is true send {"* Also got O2 PRESS alarm and/or",return, " S68 CRYO H2 PRES and",return, " S68 CRYO O2 PRES msg lines",return}.

[20] if 63ax8 is false send {"* Did not get O2 PRESS alarm and/or",return, " S68 CRYO H2 PRES and",return, " S68 CRYO O2 PRES msg lines",return}.

[21] if 63ax8 is true send { "* DIAGNOSIS: Loss of ESS bus common to both affected",return, " O2 and H2 htr cntlrs.",return, " Affected tank quantities are also lost.",return, " If ESS1BC 013 & R15 are lost then only msg lines with",return, " no F7 lights will result",return}.

[22] if 63ax9 is true send {"* RECOVERY: Reconfig htrs per BUS LOSS SSR. ",return, " If pri ESS bus lost, other alarms will be present.",return, " If not pri bus, subbuses are:",return, " TK1: ESS2CA 013 & R15",return, " TK2: ESS1BC 013 & R15",return, " TK3: ESS3AB ML868",return, " TK4: ESS1BC ML868",return}.

[23] if 63ax11 is true send {"* Either or both TK3 and TK4 are affected",return}.

[24] if 63ax11 is false send {"* Neither TK3 nor TK4 are affected",return}.

[25] if 63ax12 is true send{"* TK3 and TK4 are depleted. QTY < 10%",return}.

[26] if 63ax12 is false send{"* ACTION: TK3 and TK4 are not depleted.",return, " Operate on TK3 and TK4 until QTY < 10% then:",return, " (R1) H2 TK3 HTRS A,B (two) - OFF and",return, " (A11) CRYO TK4 HTR H2 A,B (two) - OFF",return}.

[27] if 63ax14 is true send {"* CNTLR cb of affected tk (TK1 and/or TK2) on Pnl 013 is open",return}.

[28] if 63ax14 is false send {"* CNTLR cb of affected tk (TK1 and/or TK2) on Pnl 013 is closed",return}.

[29] if 63ax15 is true send {"* DIAGNOSIS: Possible electrical problem. Do not attempt to reset circuit breaker",return}.

[30] if 63ax16 is true send {"* CNTLR cb of affected tk on Pnl ML686 is open",return}.

[31] if 63ax16 is false send {"* CNTLR cb of affected tk on Pnl ML686 is closed",return, return}.

[32] if 63ax17 is true send {"* Continue to monitor. TK3 and TK4 Htrs cycle on when pressure in both TKs = 217 - 223 psia",return}.

[33] if 63ax17 is false send {"* Continue to monitor. TK3 and TK4 Htrs do not cycle on when pressure in both TKs = 217 - 223 psia",return}.

[34] if 63ax18 is true send {"* DIAGNOSIS: P 63axduce failed low. Continue to operate TK3 and TK4 in AUTO",return}.

[35] if 63ax19 is true send {"* DIAGNOSIS: Possible electrical problem. Do not attempt to reset circuit breaker",return}.

[36] if 63ax20 is true send {"* DIAGNOSIS: PWR failure in affected HTR CNTLR",return, return}.

[37] if 63ax21 is true send {"* ACTION: Deact htrs in affected tk(s).",return, " (R1) H2 TK3 HTRS A,B (two) - OFF and/or",return, " (A11) CRYO TK4 HTR H2 A,B (two) - OFF",return, return}.

[38] if 63ax22 is true send {"* TK1 & TK2 htrs cycle on when press in both TKs = 200-206 psia",return}.

[39] if 63ax22 is false send {"* TK1 & TK2 htrs do not cycle on when press in both TKs = 200-206 psia",return}.

[40] if 63ax23 is true send {"* TK3 and/or TK4 are affected.",return}.

[41] if 63ax23 is false send {"* TK3 and/or TK4 are not affected.",return}.

[42] if 63ax24 is true send {"* DIAGNOSIS: P Xducer failed low. Continue to operate TK1 & TK2 in AUTO",return}.

[43] if 63ax25 is true send {"* DIAGNOSIS: PWR failure in affected htr cntlr",return, return}.

[44] if 63ax26 is true send {"* TK3 and TK4 htrs were deactivated (all htr switches in OFF when problem occurred",return}.

[45] if 63ax26 is false send {"* TK3 and TK4 htrs not deactivated when problem occurred",return}.

[46] if 63ax27 is true send {"* DIAGNOSIS: Instrumentation failure. No action required",return}.

[47] if 63ax28 is true send {"* ACTION: Operate TK1 & TK2 htrs in manual mode. (R1) H2 TK1,2 HTRS A,B - ON/OFF (as reqd)",return}.

[48] if 63ax29 is false send {"* Pressure in both TK3 & TK4 <= 293.8 psia",return}.

[49] if 63ax29 is true send {"* Pressure in both TK3 & TK4 > 293.8 psia",return}.

[50] if 63ax30 is true send {"* DIAGNOSIS: Auto pressure control failure",return}.

[51] if 63ax31 is true send {"* Both P and TK P of affected tk is high",return}.

[52] if 63ax31 is false send {"* Both P and TK P of affected tk not high",return}.

[53] if 63ax32 is true send {"* MANF Ps agree with P and TK P of affected tk",return, return}.

[54] if 63ax32 is false send {"* MANF Ps do not agree with P and TK P of affected tk",return, return}.

[55] if 63ax33 is true send {"* DIAGNOSIS: Line blockage in tk reading high",return}.

[56] if 63ax34a is true send {"* P high",return}.

[57] if 63ax34b is true send {"* TK P high",return}.

[58] if 63ax35 is true send {"* DIAGNOSIS: Auto pressure control or RPC failure",return, return}.

[59] if 63ax36 is true send {"* DIAGNOSIS: Instrumentation failure",return}.

[60] if 63ax37 is true send {"* ACTION: Leave affected htrs deactivated until MCC",return, " develops consumables management plan",return}.

[61] if 63ax38 is true send {"* DIAGNOSIS: Instrumentation failure",return}.

[62] if 63ax39 is true send {"* ACTION: Activate htrs.",return, " (R1) H2 TK3 HTRS A,B - AUTO",return, " (A11) CRYO TK4 HTR H2 A,B - AUTO",return}.

[63] if 63ax40 is true send {"* Pressure in both TK1 & TK2 > 293.8 psia",return}.

[64] if 63ax40 is false send {"* Pressure in both TK1 & TK2 ~> 293.8 psia",return}.

[65] if 63ax41a is true send {"* ACTION: When P < 220 psia, activate htrs.",return, " (R1) H2 TK1,2 HTRS A,B - AUTO",return}.

[66] if 63ax41b is true send {"* Alarm obtained on next htr cycle",return}.

[67] if 63ax41b is false send {"* Alarm not obtained on next htr cycle",return}.

[68] if 63ax42 is true send {"* DIAGNOSIS: Auto pressure control failure",return}.

[69] if 63ax43 is true send {"* DIAGNOSIS: Pressure overshoot caused original alarm",return, return}.

[70] if 63ax44 is true send {"* ACTION: Operate htrs in manual mode.",return, " (R1) H2 TK1,2 HTRS A,B - ON/OFF (as reqd)",return}.

[71] if 63ax45 is true send {"* Both P and TK P of affected tk is high",return}.

[72] if 63ax45 is false send {"* Both P and TK P of affected tk not high",return}.

[73] if 63ax46 is true send {"* MANF Ps agree with P and TK P of affected tk",return}.

[74] if 63ax46 is false send {"* MANF Ps don't agree with P and TK P of affected tk",return}.

[75] if 63ax47 is true send {"* DIAGNOSIS: Auto pressure control or RPC failure",return, return}.

[76] if 63ax48 is true send {"* DIAGNOSIS: Line blockage in tk reading high",return}.

[77] if 63ax49a is true send {"* P high",return}.

[78] if 63ax49b is false send {"* TK P high",return}.

[79] if 63ax50 is true send {"* DIAGNOSIS: Instrumentation failure",return}.

[80] if 63ax51 is true send {"* ACTION: Activate htrs.",return, " (R1) H2 TK1,2 HTRS A,B - AUTO",return}.

[81] if 63ax52 is true send {"* DIAGNOSIS: Instrumentation failure",return}.

[82] if 63ax53 is true send {"* ACTION: Operate htrs in manual mode when TK3,4 depleted (QTY < 10%):",return,

" (R1) H2 TK1,2 HTRS A,B - ON/OFF (as reqd)",return}.

[* ** *] [* footnotes *] [* ** *] [83] if 63ax28 is true or 63ax44 is true or 63ax53 is true send { "* NOTE: When operating in manual mode, reset pressure annun limits",return, " limits to alert when htr cycle reqd.",return, " P transducer (C&W) should be used in preference to ",return, " TK P (SM Alert) when possible. Htrs in both tks",return, " should be turned ON when P or TK P in either tk decreases",return, " to 180 psia, and OFF when P or TK P in either tk increases",return, " to 250 psia.",return}.

[84] if 63ax48 is true send {"* NOTE: Even with htrs OFF, environmental heat leak will eventually cause tk relief vlv to crack.",return}.

```
[85] if 63ax52 is true send { "* NOTE: Failure of the P transducer high will prevent AUTO htr ops.",return}.

[86] send{ "<< ECLS 63.a : Finished <<<<<<<<<<<<<<<<<<<<<<<<<<<<<".return}.

end.
```

Appendix B
rosie2eqn

```
/* rosie grammar %W% %G%
  This grammar is used to syntactically recognise rosie rules.
  These rules are translated to the input format for the eqntott program.
  Eqntott is part of the cad tool package from Berkeley.
*/
%{
#include "utility.h"
SYMBOLREF yysymbol;
%}
%union { struct nodE * nodetype;
     int inttype;
     }

%left or←symbol
%left and←symbol
%nonassoc is←symbol
%right not←symbol     /* fake priority declaration for unary not */

%token <inttype> To←symbol and←symbol assert←symbol
%token <inttype> close←symbol colon←symbol comma←symbol
%token <inttype> end←symbol false←symbol go←on←symbol go←symbol if←symbol
%token <inttype> is←defined←as←symbol is←symbol not←symbol open←symbol
%token <inttype> or←symbol period←symbol true←symbol
%token <nodetype> tag←symbol
%type <nodetype> assertion boolean←denoter boolean←expression tag

%%

/* production rules */
rosie :
     initialization destination actions end;

action :
     error
    | goto
    | rule ;
```

```
actions :
      action
    | actions action ;

assertion :
      assert←symbol tag is←symbol boolean←denoter
       {  IF $4->sym->Symbol←Token←Type = = true←symbol
         THEN        /* assert tag is true */
           $$ = $2;   /* return the tag */
         ELIF $4->sym->Symbol←Token←Type = = false←symbol
         THEN        /* assert tag is false */
                  /* return 'not tag' */
           $$ = monad(not←symbol←entry, $2);
         ELSE yyerror("unknown Boolean value\n")
         FI
       };


boolean←denoter :
             /* return a node with a representation of TRUE or FALSE */
      false←symbol
       { $$ = leaf(yysymbol);
       }
     | true←symbol
       { $$ = leaf(yysymbol);
       } ;

boolean←expression :
       boolean←denoter
     | boolean←expression and←symbol boolean←expression
       { $$ = dyad(and←symbol←entry, $1, $3);
       }
     | boolean←expression is←symbol boolean←expression
       {  IF $3->num←nodes = = 0
          THEN IF $3->sym->Symbol←Token←Type = = true←symbol
            THEN $$ = $1;     /* reduce 'x is true' to 'x' */
            ELIF $3->sym->Symbol←Token←Type = = false←symbol
            THEN          /* reduce 'x is true to 'not x' */
              $$ = monad(not←symbol←entry, $1);
            ELSE yyerror("unknown Boolean type")
            FI
          ELSE $$ = dyad(is←symbol←entry, $1, $3);
          FI
       }
     | boolean←expression or←symbol boolean←expression
       { $$ = dyad(or←symbol←entry, $1, $3);
       }
    | not←symbol boolean←expression
      { $$ = monad(not←symbol←entry, $2);
      }
    | open←symbol boolean←expression close←symbol
      { $$ = monad(open←symbol←entry, $2);
      }
    | tag
    ;

destination :
    To←symbol tag colon←symbol;

end :
    end←symbol period←symbol;

goto :
    go←symbol tag period←symbol;


if←rule :
    if←symbol boolean←expression assertion period←symbol
      {
        IF $3->sym->Symbol←Token←Type = = not←symbol
        THEN printf("r%s = ~(", $2->node1->sym->Symbol);
          print←inorder($2);
          printf(");\n")
        ELSE printf("r%s = ", $3->sym->Symbol);
          print←inorder($2);
          printf(";\n")
        FI
      };

initialization :
      {
        initialize←symbol←table();
        yyterminit();
      }
    ;

rule :
      if←rule ;

tag :
      tag←symbol
      { $$ = leaf(yysymbol);   /* return the pointer to the symbol */
      }
    ;
```

References

[Armstrong 71]	Armstrong, D. M. *International Library of Philosophy and Scientific Method.* Volume 9: *A Materialist Theory of the Mind.* Routledge & Kegan Paul, New York, 1971.
[Balzer 80]	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.
[Barr 81]	Barr, A. and Feigenbaum, E. eds. *The Handbook of Artificial Intelligence.* HeurisTech Press, 1981.
[Barstow 79]	Barstow, D. R. An Experiment in Knowledge-based Automatic Programming. *Artificial Intelligence* 12:7-119, August, 1979.
[Bennett 79]	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.
[Boute 71]	Boute, R., E. J. McCluskey. *Fault Equivalence in Sequential Machines.* Technical Report 15, Stanford University, June, 1971.
[Brown 74]	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.
[Bruce 72]	Bruce, B. C. A Model for Temporal References and Its Applicatin in a Question Answering Program. *Artificial Intelligence* 3:1-25, 1972.
[Bundy 79]	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.
[Bundy83 83]	. Bundy, A. The Nature of AI: A Reply to Schank. *AI Magazine* IV(4):29-31, 1983.

[Clancey 79] 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.

[Codd 70] Codd, E. F.
A Relational Model of Data for Large Shared Data Banks.
CACM 13(6):377-387, June, 1970.

[Dahl82 82] Dahl, V.
On Database Systems Development Through Logic.
ACM Tran. on Database Systems 7(1):103-123, March, 1982.

[Dahl83 83] Dahl, V.
Logic Programming as a Representation of Knowledge.
Computer :106-112, October, 1983.

[Date 77] Date, C. J.
The Systems Programming Series. Volume 5: *An Introduction to Database Systems.*
Addison-Wesley Publishing Company, Reading, Massachusetts, 1977.

[Davies 83] Davies, D. W.
Applying the RSA Digital Signature to Electronic Mail.
Computer :55-62, February, 1983.

[Davis 79] Davis, R.
Interactive Transfer of Expertise: Acquisition of New Inference Rules.
Artificial Intelligence 12:121-157, August, 1979.

[Dean 66] Dean, R. A.
Elements of Abstract Algebra.
John Wiley and Sons, Inc., New York, London, Sydney, 1966.

[Deland 74] Deland, E.
Fluidmod.
Technical Report, Rand Corporation, 1974.

[Duda 79] 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.

[Fagan 79] 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.

[Fagin 77] Fagin, R.
Functional Dependencies in a Relational Database and Propositional Logic.
IBM J. Res. Develop. :534-544, November, 1977.

[Feigenbaum 71] 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.

[Fikes70 70] Fikes, R. E.
REF-ARF: A System for Solving Problems Stated as Procedures.
Artificial Intelligence 1(1/2):27-120, March, 1970.

[Garey 79] Garey, M., Johnson, D.
Mathematical Sciences: Computers and Intractability. A Guide to the Theory of NP-Completeness.
Freeman, San Francisco, 1979.

[Gelernter 77] Gelernter, J. L. et al.
Empirical Explorations of SYNCHEM.
Science :1041-1049, September, 1977.

[Georgeff 82] Georgeff, M. P.
Procedural Control in Production Systems.
Artificial Intelligence 18:175-201, 1982.

[Hendrix 73] Hendrix, G. G.
Modeling Simultaneous Actions and Continuous Processes.
Artificial Intelligence 4:145-180, 1973.

[Hofstadter 79] Hofstadter, D.
Godel, Escher, Bach.
Vintage Books, New York, 1979.

[Hopcroft 79] Hopcroft, J., Ullman, J.
Computer Science: Introduction to Automata Theory, Languages, and Computation.
Addison-Wesley, Reading, Massachusetts, 1979.

[JSCDPS 84] 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.

[JSCMALFS 82] NASA.
System Malfunction Procedures Requirements
Final, Rev. B edition, Flight Operations Division, Lyndon B. Johnson Space Center, Houston, TX 77058, June 15, 1982.

[Kahn 77] Kahn, K., G. A. Gorry.
Mechanizing Temporal Knowledge.
Artificial Intelligence 9:87-108, 1977.

[Kemeny 66] Kemeny, J. G., J.L. Snell, G.L. Thompson.
Prentice-Hall Mathematics Series: Introduction to Finite Mathematics.
Prentice-Hall, Inc., Englewood Cliffs, N. J., 1966.

[Kowalski79 79] Kowalski, R.
Artificial Intelligence Series. Volume 7: *Logic for Problem Solving.*
North-Holland, New York, 1979.

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

[Mayo 83] 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.

[McCarthy83 83] McCarthy, J.
AI Needs More Basic Research.
AI Magazine IV(4):5, 1983.

[McDermott 80] McDermott, J.
R1: An Expert in the Computer Systems Domain.
In *Proc. of the First National Conf. on Artificial Intelligence.* 1980.

[McDermott78a 78]
McDermott, J, Newell. A., Moore, J.
The Efficiency of Certain Production System Implementations.
In Waterman, D. A., Hayes-Roth, F. (editor), *Pattern-Directed Inference Systems,* . pages 155-176. Academic Press, New York, 1978.

[NASA 78] Flight Operations Directorate: Crew Training and Procedures Division.
JSC-12770 Shuttle Flight Operations Manual
NASA Johnson Space Center, 1978.
Preliminary.

[Nii 79] Nii, H. P., and N. Aiello.
AGE (Attempt to Generalize): A knowledge-based program for building knowledge-based programs.
IJCAI :645-655, 1979.

[Nilsson 80] Nilsson, N.
Principles of Artificial Intelligence.
Tioga, 1980.

[Nilsson83 83] Nilsson, N.
Artificial Intelligence Prepares for 2001 .
AI Magazine IV(4):7-14, 1983.

[Pau 81] L. F. Pau.
Control and Systems Theory. Volume 11: *Failure Diagnosis and Performance Monitoring.*
Marcel Dekker, Inc., New York, 1981.

[Politakis 84] Politakis, P., S. M. Weiss.
Using Empirical Analysis to Refine Expert System Knowledge Bases.
Artificial Intelligence 22:23-48, 1984.

[Pople 77] Pople, H.
The formation of composite hypotheses in diagnostic problemsolving-an exercise in synthetic reasoning.
IJCAI :1030-1037, 1977.

[ROSIE 81] J. Fain, D. Gorlin, F. Hayes-Roth, S. Rosenschein, H. Sowiziral, D. Waterman.
The ROSIE Language Reference Manual
RAND Corporation, 1981.

[Shortliffe 76a] Shortliffe, E. H.
Computer-based medical consultations: MYCIN.
American-Elsevier, New York, 1976.

[Shortliffe 76b] Shortliffe, E. H.
Computer-Based Medical Consultations: Mycin.
Elsevier, 1976.

[Swartout 83] Swartout, W. R.
XPLAIN: a System for Creating and Explaining Expert Consulting Programs.
Artificial Intelligence 21:285-325, 1983.

[T. L. Booth 67] T. L. Booth.
International Series in Applied Mathematics: *Sequential Machines and Automata Theory.*
John Wiley and Sons, Inc., New York, New York. 1967.

[Tannenbaum 76] Tannenbaum.
Series in Automatic Computation: *Structured Computer Organization.*
Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

[Thomas 79] R. Thomas.
Kinetic Logic. A Boolean Approach to the Analysis of Complex Regulatory Systems.
In *Lecture Notes in Biomathematics*. European Molecular Biology Organization, 1979.

[Ullman 80] Ullman, J. D.
Computer Software Engineering Series: Principles of Database Systems.
Computer Science Press, Potomac, Maryland, 1980.

[Vere 83] Vere, S. A.
Planning In Time: Windows and Durations for Activities and Goals.
PAMI 5(3):53-60, May, 1983.

[Waldrop 84] Waldrop, M. M.
The Necessity of Knowledge.
Science :1279-1282, 23 March, 1984.

[Waterman 78] Waterman, D. A., Hayes-Roth, F.
An Overview of Pattern-Directed Inference Systems.
In Waterman, D. A., Hayes-Roth, F. (editor), *Pattern-Directed Inference Systems*, , pages 3-22. Academic Press, New York, 1978.

[Weiss 77] Weiss, S. M., C. A. Kulikowski, A. Safir.
A model-based consultation system for the long-term management o glaucoma.
IJCAI :826-832, 1977.

[Weiss 79] Weiss, S. M., C. A. Kulikowski.
EXPERT: A system for developing consultation nodes.
IJCAI :642-947, 1979.

[Zisman 78] Zisman, M. D.
Use of Production Systems for Modeling Asynchronous, Concurrent Processes.
In Waterman, D. A., Hayes-Roth, F. (editor), *Pattern-Directed Inference Systems*, , pages 53-68. Academic Press, New York, 1978.