GOMI, T.
--A proof-of-concept experiment system for the spacecraft autonomy management system (SAMS).

DOC CONTRACTOR REPORT DOC-CR-SP-85-045

DEPARTMENT OF COMMUNICATIONS - OTTAWA - CANADA

SPACE PROGRAM

TITLE: A PROOF-OF-CONCEPT EXPERIMENTAL SYSTEM FOR THE SPACECRAFT AUTONOMY MANAGEMENT SYSTEM (SAMS)

AUTHOR(S): T. Gomi
N. Nakamura

ISSUED BY CONTRACTOR AS REPORT NO: AAIS-84-004

PREPARED BY: Applied AI Systems, Inc.
P.O. Box 13550
Kanata, Ontario
K2K 1X6

DEPARTMENT OF SUPPLY AND SERVICES CONTRACT NO: 06ST.36001-3-4454

DOC SCIENTIFIC AUTHORITY: R.A. Millar

CLASSIFICATION: UNCLASSIFIED

DATE: 6/5/85

(2)

A Proof-of-Concept Experiment System for the Spacecraft Autonomy Management System (SAMS)

Technical Report No. AAIS-84-004

By
(1)
T. Gomi
N. Nakamura

Applied AI Systems, Inc.
P.O. Box 13550
Kanata, Ontario
K2K 1X6

Under DSS/DOC contract 06ST.36001-3-4454

Version 20APR85

C O N T E N T S

G L O S S A R Y

AASC Advanced Autonomous Spacecraft Computer, a spacecraft computer system concept developed at CRC (CRC/AASC)

ACC Autonomy Control Cluster (AASC/SAMS/ACC)

AI Artificial Intelligence (Computer Science/AI)

AOCS Attitude and Orbiting Control Subsystem (Spacecraft/Subsystems/AOCS)

COMDB COMmon Data Base (AASC/SAMS/POC/COMDB). A data base accessed by both the LLKS and the HLKS as a short term memory

COMKB COMmon Knowledge Base (AASC/SAMS/POC/COMKB). A Knowledge base accessed by both the LLKS and the HLKS as a long term memory of knowledge

CRC Department of Communications, Communications Research Centre (DOC/CRC)

DEVISOR
JPL's domain independent purpose automated planner - schedular (JPL/[PEER]/DEVISOR)

DOC Department of Communications, Government of Canada

EEM External Environment Manager (AASC/SAMS/AAC/EEM) A SAMS function that manages the spacecraft's response to physical environmental parameters from external sources

EIM External Interface Manager (AASC/SAMS/ACC/EIM) A functional component of the SAMS' autonomy management cluster. Manages the autonomy management aspects of dealing with systems external to the spacecraft.

EIU Environment Interface Unit (AASC/SAMS/POC/EIU) An element of the SAMS POC system that generates simulated environmental conditions.

EMES Energy Management Expert System. An expert system designed for managing on-board energy consumption by spacecraft subsystems (martin Marietta)

FAITH Forming And Intelligently Testing Hypotheses, a JPL expert system to diagnose spacecraft malfunctions (JPL/PEER/FAITH)

FIES	Fault Isolation Expert System, an onboard fault isolation expert system for automating on-board power subsystem (Martin Marietta)
FTM	Fault-Tolerance Management, a generic name given to the lower layers of the AASC hierarchy (AASC/FTM)
HLKB	High Level Knowledge Base (AASC/SAMS/POC/HLKB). A knowledge base for the LLKS
HLKS	High Level Knowledge-based System (AASC/SAMS/HLKS)
JPL	Jet Propulsion Laboratory, California Institute of Technology (JPL)
KBS	Knowledge-Based System (AI/KBS). Synonym for Expert System, except in the KBS the knowledge source is not necessarily attributed to an expert.
LLKB	Low Level Knowledge Base (AASC/SAMS/POC/LLKB). A knowledge base for the LLKS.
LLKS	Low-Level Knowledge-based System (AASC/SAMS/LLKS)
OIU	Operator Interface Unit (AASC/SAMS/OIU). An element of the SAMS POC system which interfaces the system with the operator.
POC	Proof of Concept
SCC	Subsystem Control Cluster (AASC/SAMS/SCC). An adaptation of conventional on-board logistical subsystems for the SAMS architecture.
PEER	Planning and Execution with Error Recovery, a blanket AI system with the objective of automating spacecraft operation (JPL/PEER)
SA	Subsystem Administrator (AASC/SAMS/SACC/SA)
SAMS	Spacecraft Autonomy Management System, a substructure of the hierarchical design of the AASC (AASC/SAMS)
SGM	Spacecraft General Manager (AASC/SAMS/ACC/SGM)

Acknowledgements

The Spacecraft Autonomy Management System (SAMS) was developed by the authors for the Communications Research Centre (CRC) of the Federal Department of Communications (DOC) under contract to the Department of Supply and Services (Contract Number 06ST.36001-3-4454). Authors are thankful for the support given by Dr. S.P. Altman and Mr. R.A. Millar of the Communications Research Centre. They would also like to express their thanks to Mr. Dave Andean, also of the CRC, who provided them with knowledge of spacecraft operations management, and uponwhose expertise the experiments described herein depended.

Summary

The SAMS is conceived as the top layer of the Advanced Autonomous Space Computer (AASC) hierarchy developed at the CRC during the past three years. The AASC has the capacity for further upward expansion. The SAMS layers are characterized by their use of Artificial Intelligence (AI) techniques. The SAMS is described in the report "Functional Design of a Knowledge-based Spacecraft Autonomy Management System (SAMS)" (Technical Report No. AAIS-84-001, Applied AI Systems Inc.).

This report describes a set of expert systems developed as a Proof of Concept (POC) experimental system, and a series of experiments conducted using them. The two expert systems are called the Low-Level Knowledge-based System (LLKS) and the High-Level Knowledge-based System (HLKS). They are designed to prove the capability of autonomously managing on-board anomalies, the premise of the SAMS concept. The experiments involved testing the expert systems separately and testing operations run on the combined expert system complex.

1. Introduction

The SAMS concept was developed as a method for automating the management of a spacecraft. It can be applied to spacecraft autonomy management tasks, the like of which, conventional technology has been unsuccessful in automating. To compensate for the shortcomings of existing automation approaches which are based on classical control system theory theory, a new set of system control methodologies was introduced: a collection of knowledge-based systems or expert systems. Substantial structural and other renovations to the existing expert system architecture was necessary to make the knowledge-based expert systems acceptable as the POC experimental system. The changes were necessary because the existing systems are typically based on a fixed, narrowly defined mode of operation which differs substantially from the domain of autonomous spacecraft management.

As the need for autonomous spacecraft management increases, the search for new approaches to manage spacecraft operations intensifies. Many working in the field of spacecraft autonomy have discovered the need to investigate AI as a tool for autonomy management. There are several similar but mostly unrelated system developement efforts currently underway, mostly in the United States. Some have reached the stage of constructing an experimental system and actually conducting experiments [Wagner 83, 84], while others are still in the planning stage [Mitchel and Lemmer 84] [Dickey 84]. Very few have reached the stage of prototyping as of this writing, except for a few military systems in the U.S., details of which are not available.

Most of these development groups are running their experiments on a simulator which typically is a software-oriented computer simulation [Sauers 84] [Bein 84]. This was the approach chosen for the testing of the SAMS POC system.

In addition to autonomy management systems for spacecraft, there are similar autonomy management systems under development for avionics applications [Cross 84] [Milne 84] [Schundy 84] [Girad 84]. There are more similarities than differences between these systems and spacecraft autonomy management systems. The developers are concerned with building a system that operates in a dynamic, remote environment in order to achieve objectives similar in their attributes to those of the spacecraft autonomy systems. For that reason, their developments are worth monitoring.

Experiments built and conducted by Pisano and Jones [Pisano and Jones 84] on a dedicated computer system are significant in their successful demonstration of the capability of AI to control the plan guided behavior of an autonomous system. The project is also demonstrative of a proof of concept model which has had substantial engineering efforts already expended towards its eventual full scale implementation. Anderson and his group at Texas Instruments have also constructed and run an experiment in a similar domain, but using a different approach, and with less concrete results [Anderson, et al 84].

There are also a number of projects which involve the development of autonomous ground or underwater vehicles. The mode of operation and the functional architecture of autonomy management systems for these vehicles are again very similar to those of spacecraft or aircraft autonomy management systems. A project at the Naval Ocean Systems Center by Harmon and his group [Harmon 83] [Harmon, et al 84] is probably the most advanced among systems in this application domain. A convincing architecture for such a system has been defined and presently the implementation of two systems which realize the design is underway. Generally speaking, the architecture and the operating principles of the NOSC system are strikingly similar to those of the SAMS defined in the functional design. Other autonomous vehicle projects of significance which intend to build experimental systems for testing are those by the University of New Hampshire [Blidberg, et al 83], and by the Hughes Research Laboratory [Bullock, et al 83].

The SAMS adopted a combined layered and distributed architecture, as detailed in the report. In the proof of concept system, two expert systems are developed representing two of the key layers. They are placed in two Knowledge Engineering layers and named accordingly: the High-Level Knowledge-based System (HLKS) and the Lower-level Knowledge-based System (LLKS). A two-tiered expert system architecture was adopted to accomodate two conflicting requirements: the need to report to human operators and the requirement to interface with lower level system elements which in turn interface with the environment.

The mode of operation of a human operator, while highly flexible, is typically asynchronous, relatively slow, macroscopic, often irregular, and limited in judgemental and dexterous precision. An enhanced goal-driven reasoning scheme was adopted to interface the operator and overcome these drawbacks. The lower level machinery on the other hand, typically functions synchronously to inputs, fast and

regularly, with a high degree of precision, but greatly lacks in flexibility. An approach called data fusion as a form of a forward reasoning mechanism was employed at the heart of the lower level expert system to interface with the lower level system elements.

One important aspect of an autonomy system is that it must cope with very dynamic environmental phenomena which change rapidly. It thus must be constructed as a real-time system. A successful and definitive notion of a real-time AI system has yet to be developed and proven while such system formalism has been well developed in conventional computing systems during the past two decades. One obstacle is the coordination of the distinctive operational characteristics of the human and that of the lower level machines mentioned above which must be provided. Many of the projects noted earlier try to cope with this problem in various but often drastically different ways. Again, the approach taken by the NOSC group, which has its root in Carl Hewitt's distributed control system model [Hewitt and Baker 77] and the hierarchical architecture proposed by the HEARSAY projects [Erman et al 80] [Lesser & Corkill 81] seems superior.

The lack of appropriate hardware to carry out the real-time execution of an autonomy management system is another obstacle to be overcome. While the approach for selecting hardware and subservient system software differs greatly among projects, the need for more computing resource in a form appropriate to the operation is recognized by all concerned. Various efforts to develop such hardware and its accompanying system software seem to belong to so-called Fifth Generation Computer System (FGCS) projects. There are a number of different approaches proposed in this area. However, the most promising ones for the next several years seem to be those based on either dataflow machine or reduction machine architectures. These massive parallel computers are to be constructed using emerging Ultra Large Scale Integrated circuit (ULSI) technology, which include supporting developments in gallium-arsenide (GaAs) junctions and submicrometer line width microcircuits to realize roughly a 10,000 fold throughput improvement in the next several years. This advanced hardware will be built mostly for and used in non-numerical computations, the basic mode of operation of AI computers.

The availability of such powerful hardware is said to be at least five years away. In the mean time, it is expected a great deal of research will have to be conducted on the other issues described earlier, which, in many respects, are harder to solve. For this reason, the current use of non-realtime hardware and a slow software simulator for the

purpose of developing fundamental real-time applications is justified as an acceptable method of study. All existing real-time expert systems are running slower than real-time, except for IBM's YES/MVS [Hong et al 84], which performs the functions of an operator of a large main-frame computer, and Pisano's navigation expert system mentioned above. The former deals with a problem which poses relatively non-critical time constraints while the latter runs a simplified version of the simulation on a powerful AI computer. However it cannot be implemented for the intended target environment for at least two years.

2. The SAMS Proof-of-Concept Experimental System

2.1 Objectives of the Experiments

A series of experiments have been planned and conducted using the POC experimental system to test its proper functioning and to demonstrate the effectiveness of the SAMS concept. In order to realize these goals, the following specific requirements have been established:

(1) To test the appropriateness of a two-tiered expert system architecture as an effective method of asynchronously coordinating the real-time physical operational environment with the operator's environment using an Artificial Intelligence approach,

(2) To define a real-time expert system architecture as an effective structure for the LLKS,

(3) To define and test the HLKS as a management expert system which oversees the functioning of the autonomy system, and using it to identify attributes of an efficient high level expert system formalism which interface with human operators,

(4) To test the performance of the LLKS as a simulated real-time expert system,

(5) To demonstrate that a knowledge-based control loop can actually detect, report, analyze, and correct an on-board anomaly.

2.2 Structure of the POC Experimental System

Figure 2.1 shows the over-all functional structure of the POC experimental system. It contains the HLKS, the LLKS, the COMmon Knowledge-Base (COMKB) for storing the fault tree and object level diagnosis and recovery knowledge, a knowledge base for the HLKS (HLKB), a knowledge base for the LLKS (LLKB), the COMmon Data Base (COMDB) to contain the results of the LLKS' activities and to provide a search space for the HLKS, the Environment Interface Unit (EIU), and the Operator Interface Unit (OIU). The main module of the SAMS POC coordinates the operation of these modules. The main module also controls an initialization module, a module for restoring the internal state of the POC at the beginning of each simulation cycle, and a set of input generation modules employed to create simulated fault inputs.

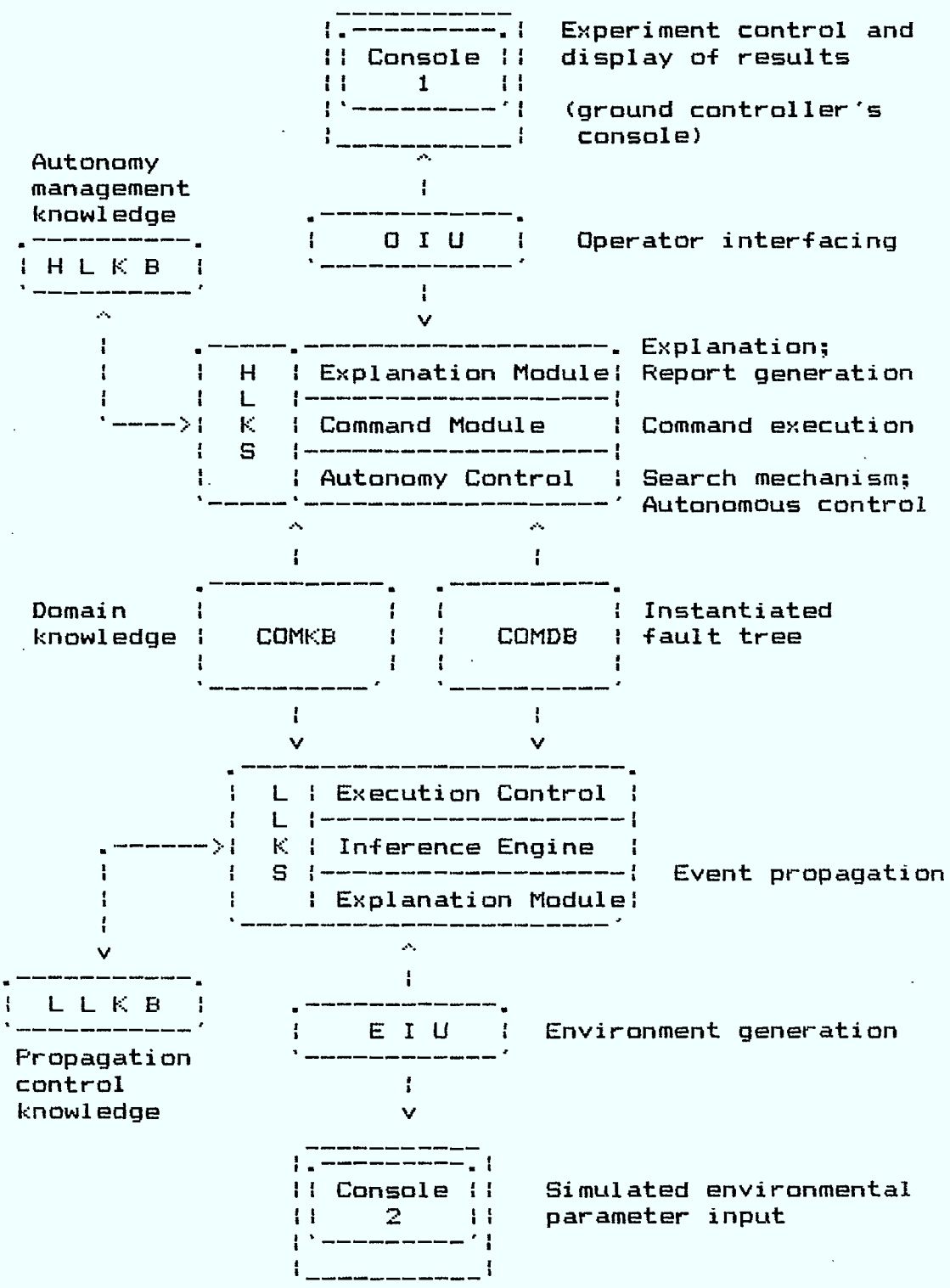

Figure 2.1 Functional Structure of the SAMS POC experimental system

The HLKS is a goal-driven expert system for high level fault processing and for managing autonomy procedures. It supervises fault detection, fault containment, fault analysis, and fault recovery processes. The generation of various reports is also controlled by this expert system. The HLKS is also involved in altering the contents of the knowledge bases, though this feature is not implemented in the present POC system.

Being a goal-driven expert system, the HLKS searches through an instantiated fault tree, which was developed in the COMKB by the LLKS. The controller may specify the search mode of the HLKS and give other instructions to the HLKS through the OIU. Rules in the HLKB can also determine a search mode. The knowledge-based system currently supports the depth-first, a width-first, a beam, or a mixture of these search modes. The HLKS interfaces the operator through the Operator Interface Unit (OIU). Section 3 describes the HLKS in further detail.

The LLKS is a data/event driven expert system operating in the domain of low level fault handling. It detects and analyzes faults caused by on-board and external environmental changes and on-board or ground system malfunctions. When authorized by the HLKS, and ultimately by the controller, it may perform selected low-level fault recovery functions. The LLKS obtains its inputs from the Environment Interface Unit. The EIU generates simulated environmental conditions using either internal generators or inputs entered by the experimenter. Whenever a new event of significance is detected by the LLKS, and its effect propagated in the COMDB, the LLKS notifies the HLKS so that the high level expert system acting as a manager can conduct its own investigation into the COMDB. Details of the LLKS are described in Section 4.

2.3 Method of the Experiments

The main control flow of the POC experimental system is shown in Figure 2.2. Following the system wide initialization, which includes setting up input modes for each input terminal, the main simulation loop begins. During one simulation cycle, the EIU generates assertions for all input terminals. These input events are propagated through the inference network by the LLKS. If the LLKS notices a significant event that may threaten the normal operation of the system, it notifies the HLKS and the HLKS begins its investigation. Each time it discovers a serious fault during the investigation, it issues a warning to the operator (a ground controller) and issues a system prompt. The operator may enter any of the system commands.

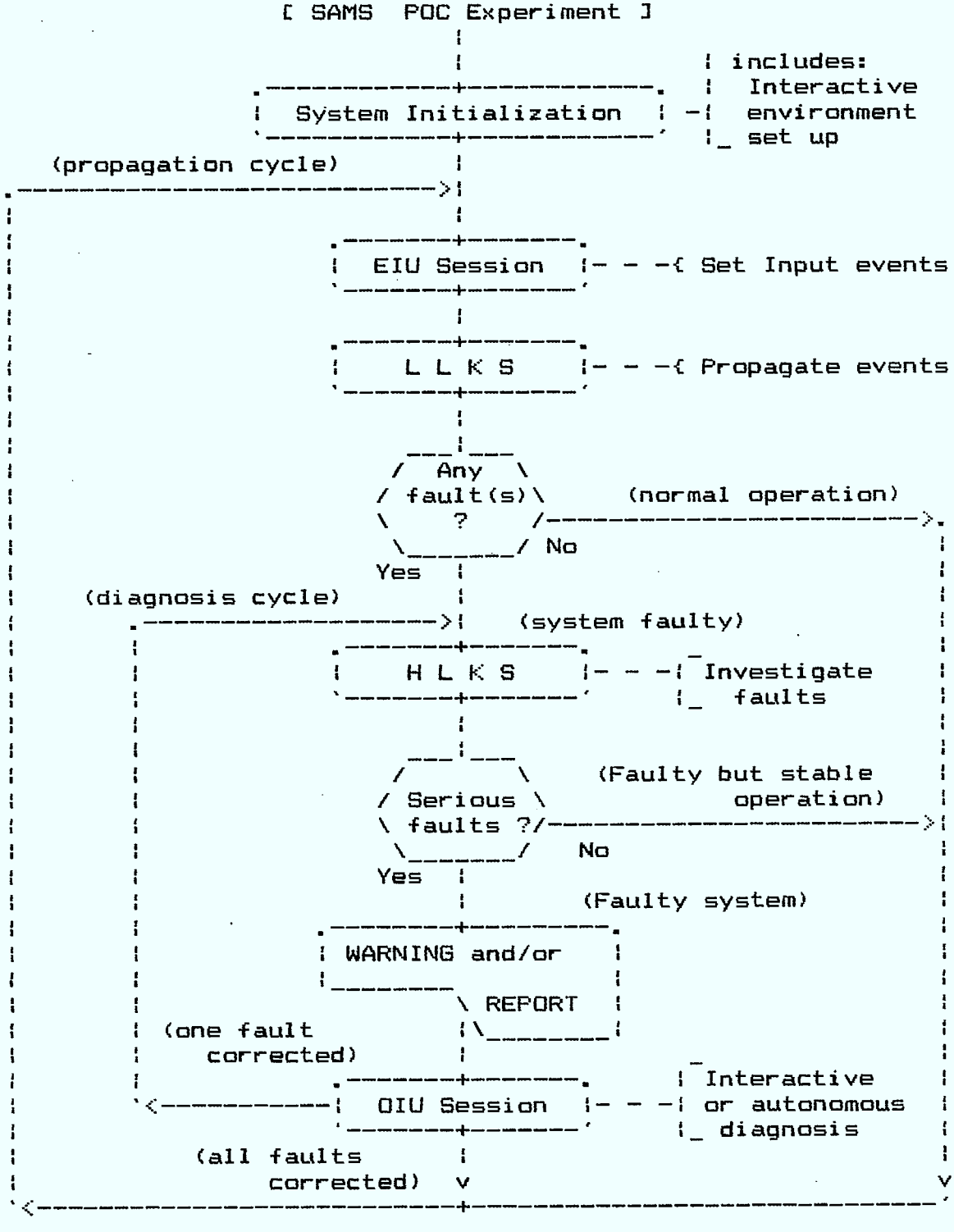

Figure 2.2 The POC experiment system main control

The HLKS consults the meta-level knowledge base (the HLKB) and the object level knowledge base (the COMKB) for rules applicable to the situation. Such autonomous diagnosis may include corrective actions. In the absence of these autonomous activities, the ground controller issues a sequence of commands interactively to the HLKS analyses the fault and attempts a fault recovery through the controller's console.

The OIU acts as the interface between the controller and the POC SAMS. Operator commands are entered through this console and the results from the POC system are displayed. Through the OIU the operator can access certain aspects of the LLKS' operation as well. These commands are described in detail in Sections 3 and 4.

The initialization module asks the user to select one of three possible input modes for each of the terminal nodes. When invoked by the main control at the beginning of a run, it prompts the experimenter with the node identifier of the terminal nodes. The experimenter may enter an 'r.' for the random number generation, an 'f.' for a 'fixed' input of 'high' or 'low', or an 'm.' for a 'manual input. Only if 'm.' is specified does the node ask for an input each time a simulation cycle needs new values. The inputs for the other two modes will be looked after automatically for the rest of the experiment. If an 'm.' was entered, the POC system prompts the operator at every simulation cycle to obtain the strength of assertion normalized between 0 and 100. The EIU is used for this exchange.

If the ranadom number option is selected for a terminal node, a random number is generated against the probability of the event that terminal node represents at each propagation cycle. Table 2.1 summarizes the current probability values for the terminal nodes of the inference network that represents the CTS/Hermes satellite's AOCS domain. The entire knowledge base for this domain is in Appendix A.3.

Table 2.1 Probabilities assigned to terminal events

telemetry_lost	.5
o4_previously_fired	.975
nitrogen_used_to_pressure_tank	1.0
impurities_in_tank	.035
fuel_in_tank_low	.3
heat_dissipation_uneven	.745
harmful_sun_reflections	.0296
shf_radiation	.000425
unstable_NESA_A_pivot	.025
motor_mechanism_contaminated	.00252
motor_fails	.173
motor_overheats	.00295
control_electronics_fails	.465
emi_to_electronics	.0015
power_needs_to_be_cut_ to_shut_NESA_A	.92
sun_position_always_changes	.914
anomalies_relates_to_sun_position	.15
nesa_A_output_must_be_cut_out	.8

3. The High-Level Knowledge-based Systems (HLKS)

3.1 Objective of the HLKS

The HLKS is a goal-driven expert system which conducts the following functions in the POC experiment:

- Interpretation of the ground controller's requests,
- Delivery of the controller's command to portions of the POC including itself,
- Monitoring of the execution of commands given by the controller,
- Compilation of reports and messages to be given to the controller,
- Survey and analysis of faults reported by the LLKS,
- Take actions necessary to contain the faults reported by the LLKS,
- Takes action necessary to recover from the selected faults reported by the LLKS.

3.2 Functional Structure of the High-Level Knowledge-based System

Figure 3.1 shows a functional structure of the HLKS.

The HLKS Interface exchanges messages with the OIU and with the LLKS. The OIU passes commands and requests to the HLKS entered by the controller through the interface. The HLKS informs the OIU of LLKS events worth investigating. The HLKS conveys those commands which affect or are destined for the LLKS.

Using various search techniques, the HLKS controls the execution of reasoning, explanation, and requested commands.

The HLKS Autonomy Control is the heart of the HLKS. It conducts reasoning using goal-driven, backward-chaining inference. Various search techniques are selectable depending on the type of problem to be solved. Warning messages are generated as it searches through the inference network. The search process will be enhanced by a meta-rule processor in the future.

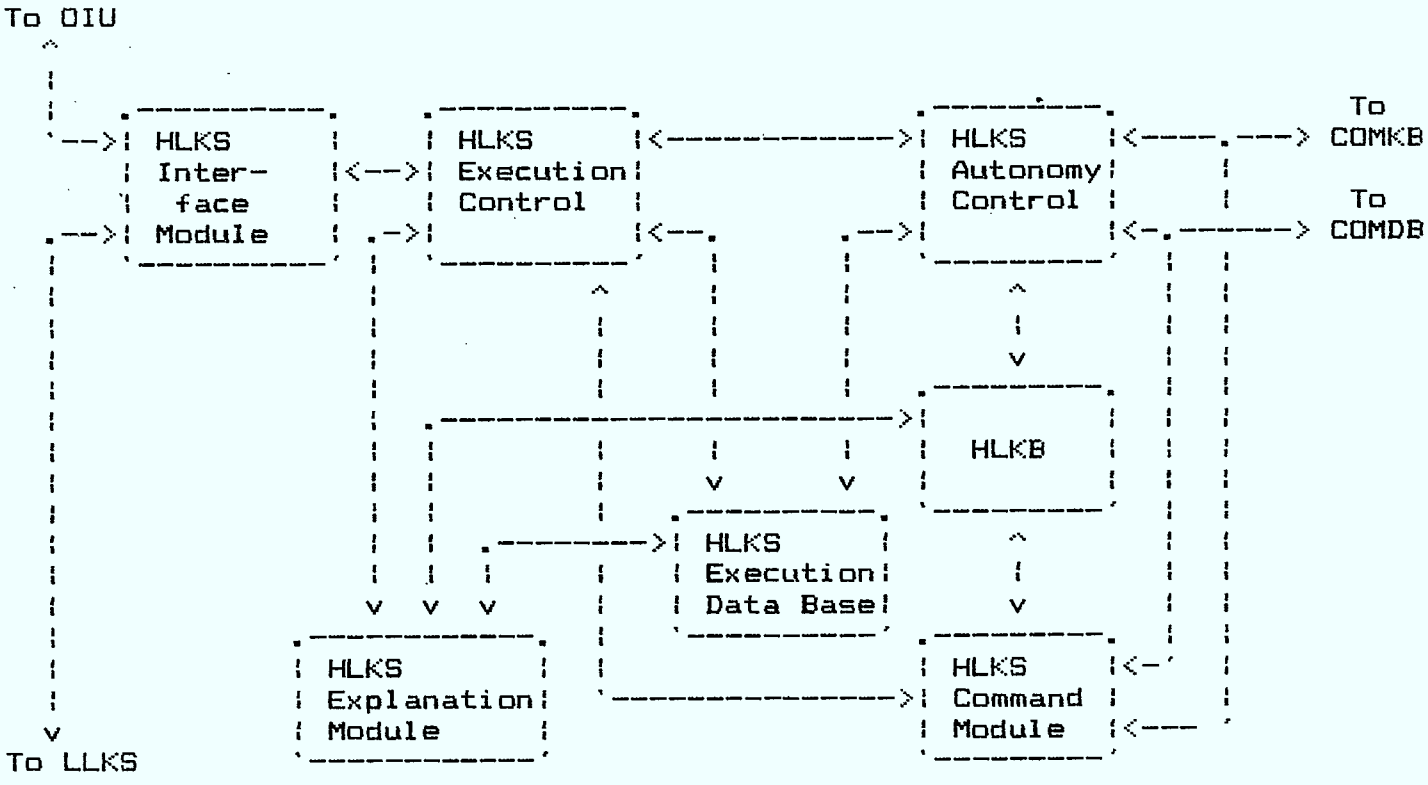

Figure 3.1 Functional Structure of the HLKS

The HLKS Command Module processes commands and requests, from ground control which are directed towards the HLKS. It receives commands from the HLKS Interface and sends back the results of processing. Report, Recommend, Assess, and Probe are current commands supported. The module compiles outputs from the COMKB, COMDB, and the HLKB.

The HLKS Explanation Module answers questions and queries made by ground control on the reasoning of the HLKS Autonomy Control, ie., the explanation of how meta-rules are used in the reasoning and search process. The explanation subsystem obtains its source for the explanations from the HLKS Execution Data Base and the HLKB. This subsystem is not implemented in the present version of the POC.

The HLKS Execution Data Base is a scratch pad storage facility used during the reasoning, determination of search strategy, and during the composition of an explanation output.

The HLKB is a knowledge base containing knowledge used only by the HLKS for autonomy control and for processing commands. The knowledge for autonomy control is meta-knowledge for use by the HLKS Autonomy Control when conducting heuristic searches. Other knowledge is for selecting recommendations and compiling appropriate reports.

3.3 Operation of the HLKS

3.3.1 The HLKS

The HLKS performs various high level functions for the ground controller. It informs the operator of anomalies, advises him of the risk he is facing, notifies him of corrective actions to be taken or evasive actions already taken by the HLKS jointly with the LLKS. Upon request, the controller is informed of the steps to be followed to recover from a fault.

When invoked, the HLKS works on an instantiated fault tree generated by the LLKS in the COMDB. This is mainly a task performed by the HLKS Autonomy Control. It searches through the fault tree and attempts to clarify the fault already marked by the LLKS. The HLKS may use meta-rules (rules concerning how to better conduct a search for solutions, or reasoning) stored in the HLKB to aid the operations of the Autonomy Control. The HLKS then conducts an analysis based on knowledge nodes being searched stored in the COMKB. A recovery action may be generated by invoking a control sequence for a node defined in the COMKB. Upon authorization from the HLKS, the LLKS may take direct corrective action on selected local faults which require a quick response. This feature is explained in Section 4.

The controller issues queries to the HLKS through the OIU to obtain the following information:

- A status report which describes the logical status of a specific element of the spacecraft and its operation control system. An element can be the entire system, a collection of subsystems, a subsystem, or any portion of a subsystem represented in the COMDB and the knowledge bases. The HLKS compiles a report by collecting information from the COMDB and the COMKB.

- A system failure report on a fault or faults whose existence was reported by the LLKS. The HLKS conducts its own search into the COMDB to clarify the faults from the viewpoint of the manager of the system. It also uses information in the knowledge bases as reference.

- A recommendation for fault containment or recovery. The HLKS compiles such a recommendation using information in the COMDB and the knowledge bases. The recommendation is in the form of recommended action steps to be taken by the ground controller.

- An explanation of the reasoning steps taken by the LLKS.

Figure 3.2 summarizes these functions in terms of the input/output relationship between the HLKS and a controller.

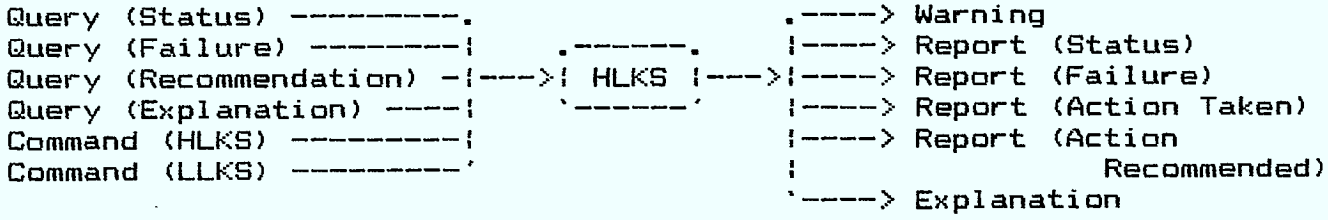

Figure 3.2 Input/Output relationship of the HLKS

3.3.2 The HLKS Interface and the OIU

The HLKS Interface performs message exchange functions both with the OIU and the LLKS.

3.3.2.1 Interface with the OIU

A ground controller enters commands to the SAMS POC system through the control console. This is the main access to the POC experimental system by the experimenter. These commands are received by the OIU and relayed to the HLKS. All commands, which are described in the balance of this and the following subsections, are implemented in the form of a PROLOG predicate. As such, it must follow the predicate syntax. In general, it has the following syntax:

Predicate (argument-1, argument-2, ..., argument-n).

The predicates are defined either in the HLKS or in the LLKS. It must begin with a lower case letter if a constant. If a variable is to be used for the arguments, it must begin with an Upper case letter. The command line **must be terminated by a period.** For example, a command to request a status report from a node called 'voltage_balance_lost' would look like this:

report (voltage_balance_lost).

Being a predicate, a command may be combined with other predicates to form macro commands. A macro thus created may even include rules.

The commands received are analyzed and processed by the HLKS Interface. Some comands are meant for the LLKS. Those addressed to the LLKS are immediately shipped to it by the HLKS Interface. Others are handed over to the HLKS Execution Control. The HLKS Execution Control dispatches each of the HLKS commands to its subsystems. There are some commands which have to be jointly processed by both the HLKS and the LLKS. The notification to the LLKS of their receipt is done through slots in the COMDB for specific nodes to which the commands are issued.

Commands sent by the OIU and addressed to and executed by the HLKS are the following:

- **search:** Initiates a search through the instantiated fault tree (inference network) built in the COMDB by the LLKS. Uses a search strategy set through its argument. The length of a search is also set each time by an argument. When used from outside, this command activates the HLKS,

- **continue:** Prompts the search mechanism to resume an interrupted search from a current node, using same search parameters (strategy and length),

- **terminate:** Terminates a search currently underway. The control returns to the top of the instantiated fault tree,

- **suspend:** Temporarily suspends the data fusion capability of a node in the inference network. The LLKS will no longer perform reasoning activities on that node until it is reactivated. Execution of this command is carried out in cooperation with the LLKS,

- **activate:** activates the data fusion capability of a node which was previously suspended, thus allowing the node to participate in reasoning activities. This command is carried out with the help of the LLKS,

- **entrust:** Jointly with the LLKS, designates a node as an autonomous action node. An autonomous action node will take a predefined action when a set of predetermined conditions are met. The condition for taking such action is defined individually for each of the autonomous action nodes.

- **relieve:** Relieves a node from being an autonomous action node. This command is also carried out in cooperation with the LLKS,

- **initialize:** Initializes the POC experimental system. Initializes the HLKS and then issues an initialization command to the LLKS for its initialization,

- **check:** Checks and verifies the structure of a designated knowledge base,

- **find_top:** Identifies a root node of the inference network. Note there can be more than one root node in an inference network.

- **report:** Reports the status of selected node(s) in the inference network, serching the inference network for supporting evidences,

- **probe:** Same as **report** but reports on one node at a time,

- **assess:** Assesses and reports on implication(s) of an anomaly,

- **recommend:** Makes recommendation(s) on steps to recover from a failure,

- **explain:** Explains the reason for a recommendation obtained using the **recommend** comand.

It must be noted that in the future all of the above commands may also be issued from within the HLKS as a result of reasoning.

3.3.2.2 Interface with the LLKS

The HLKS Interface issues to the LLKS a number of commands. Some of the commands are issued by the OIU and redirected by the HLKS Interface. Others are generated by the HLKS as a result of its operation. These commands and how they are executed in the LLKS are discussed in Section 4.3.

If, as the result of event propagation by the LLKS, there is an event worth investigating, the LLKS issues a message to the HLKS. The HLKS accesses the COMKB directly thereafter and investigates. In the investigation, the HLKS applies instructions given by the ground or its own knowledge to analyse the situation.

3.3.3 The HLKS Execution Control

The Execution Control supervises the over-all operation of the HLKS. The following processes are scheduled and their execution monitored in the HLKS:

- the HLKS Autonomy Control controlled by a goal-driven search mechanism. Selection of a search strategy and the shipment of outputs is controlled by the Execution Control,

- the HLKS Interface. Its dealings with the OIU and the LLKS are regulated,

- the HLKS Command Module. Selected commands are dispatched to the module and a reply is relayed to the HLKS Interface by the Execution Control,

- the HLKS Explanation Module. The HLKS Execution Control dispatches the subsystem with selected commands to explain system status. The results are sent out via the Execution Control.(not implemented)

3.3.4 The HLKS Autonomy Control

3.3.4.1 The operation of the HLKS Autonomy Control

Upon instruction from the Execution Control, the HLKS Inference Engine scans through the instantiated fault-tree in the COMDB using depth-first search, breadth-first search, beam search, or a combination of these search methods. The scan does not necessarily terminate when a goal (a faulty node) is detected, rather it awaits further instructions from the Execution Control and typically continues the search.

At each node the Autonomy Control performs one or more of the following four things:

- acknowledges and executes any command(s) handed down by the HLKS Execution Control, including a command for further search,
- examines the situation at the node and issues a warning message to the OIU if one is warranted,
- reasons about what corrective actions are to be taken for a troubled node for which a warning has been issued. (This function is not implemented in the present POC experimental system.)
- reasons about which search strategy to take next using meta-knowledge stored in the HLKB, (This function is not implemented in the present POC experimental system.)

For every node in which the strength of assertion exceeds a threshold, a warning message stored at the warning slot of a node is retrieved by the HLKS Autonomy Control and sent out to the OIU. Such warning messages make the controller aware of an anomaly in the system. By adjusting the threshold the message can be issued well before the situation becomes critical. Currently one threshold is set for the entire system. In the future, the threshold should be set for each node in the form of a logical expression. Such a logical expression may include procedural or functional elements as its terms, thus combining computations with deductions.

The data fusion model is effective in this regard as it is capable of predicting with a probability figure, a very slight possibility of something going wrong.

The reasoning for corrective actions will be carried out by using both the meta-knowledge stored in the HLKB and

the local domain knowledge stored for eack node, in the COMKB. An example of domain knowledge might be a set of conditions for disconnecting a suspiscious battery.

The reasoning process, which is executed by the inference engine in the HLKS Autonomy Control, also references various data in the data bases (eg., the strength of the assertion - how faulty it is - of the faulty node, the status of its neighbouring nodes - found in the COMDB, or certain parameters such as temperature readings or amount of fuel left).

A corrective action sequence itself is a part of the action knowledge stored in the slots of a node. The HLKS, on deciding upon an action, would request the LLKS to open these knowledge stores and execute them as stated. Further reasoning may take place locally as the sequence may include rules. Each step of the execution will be recited as messages to the HLKS and to the ground. The reciting of the reasoning steps taken by the HLKS Autonomy Control to a human controller is mandatory.

The reasoning for deciding on a search strategy is carried out purely as meta-level reasoning. This is reasoning for finding better ways to manage the autonomy process. The meta-reasoning is executed apart from reasoning in the fault handling domain. An example of the heuristics may be, "If nothing, maintain breadth-first search", or "If 'warning' is issued for a node, switch to beam search".

Alternative search methods are discussed below using examples:

(1) Depth-first search

Figure 3.3 shows an example of a depth-first search. Alternative nodes are chosen and tested in a strict left-to-right order, from top to bottom. Backtracking is repeatedly applied to the lowest possible untested alternative node, until the entire fault-tree is searched, or until the search is terminated by a command. This search method is suitable when a certain branch of the tree is believed to contain key facts for the solution of an anomaly and its branches are not very long but are similar in length. This approach, unfortunately, will lead the HLKS to an extremely time-consuming and arduous search if improperly applied. A fault that exists in a right-hand side branch (eg., node 9 in Figure 3.3) may be picked up much faster if a more suitable search strategy is used. Rarely does a ground

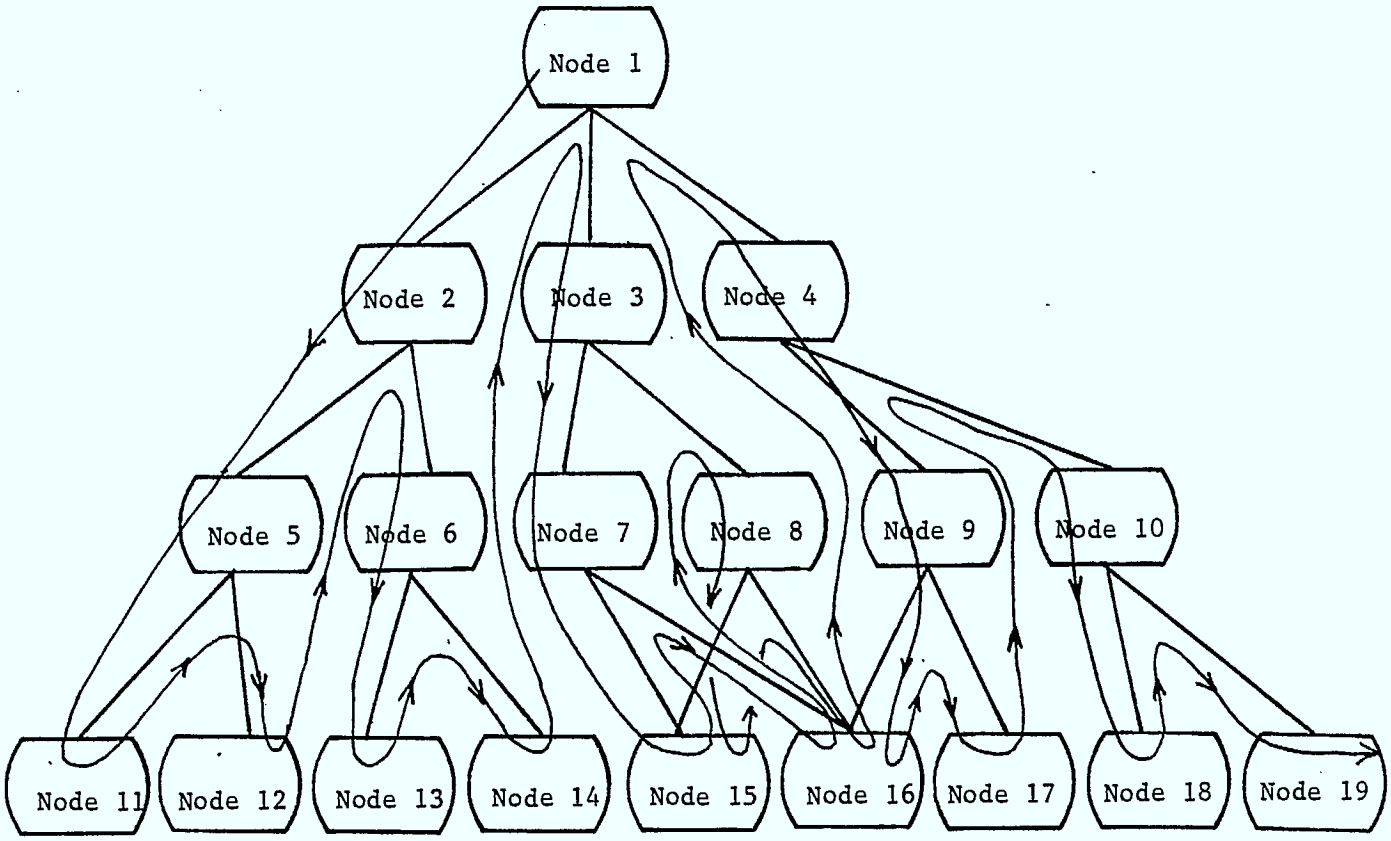

Figure 3.3 The depth-first search

controller or the POC knowledge bases have knowledge about the likely location of a fault in the topology of a fault tree. To avoid such a pitfal, depth-first search is used rarely in the HLKS for this reason.

(2) Breadth-first search

All nodes at a given depth are examined before turning to their siblings in breadth-first search, as demonstrated in Figure 3.4. This approach is particularly suited for performing diagnosis from a supervisor's point of view. It allows the conductor of the search to examine events or assertions impartially. For this reason, breadth-first search is the default search scheme of the HLKS.

(3) Beam search

Beam search examines a selected group of nodes belonging to a limited number of branches. Breadth-first

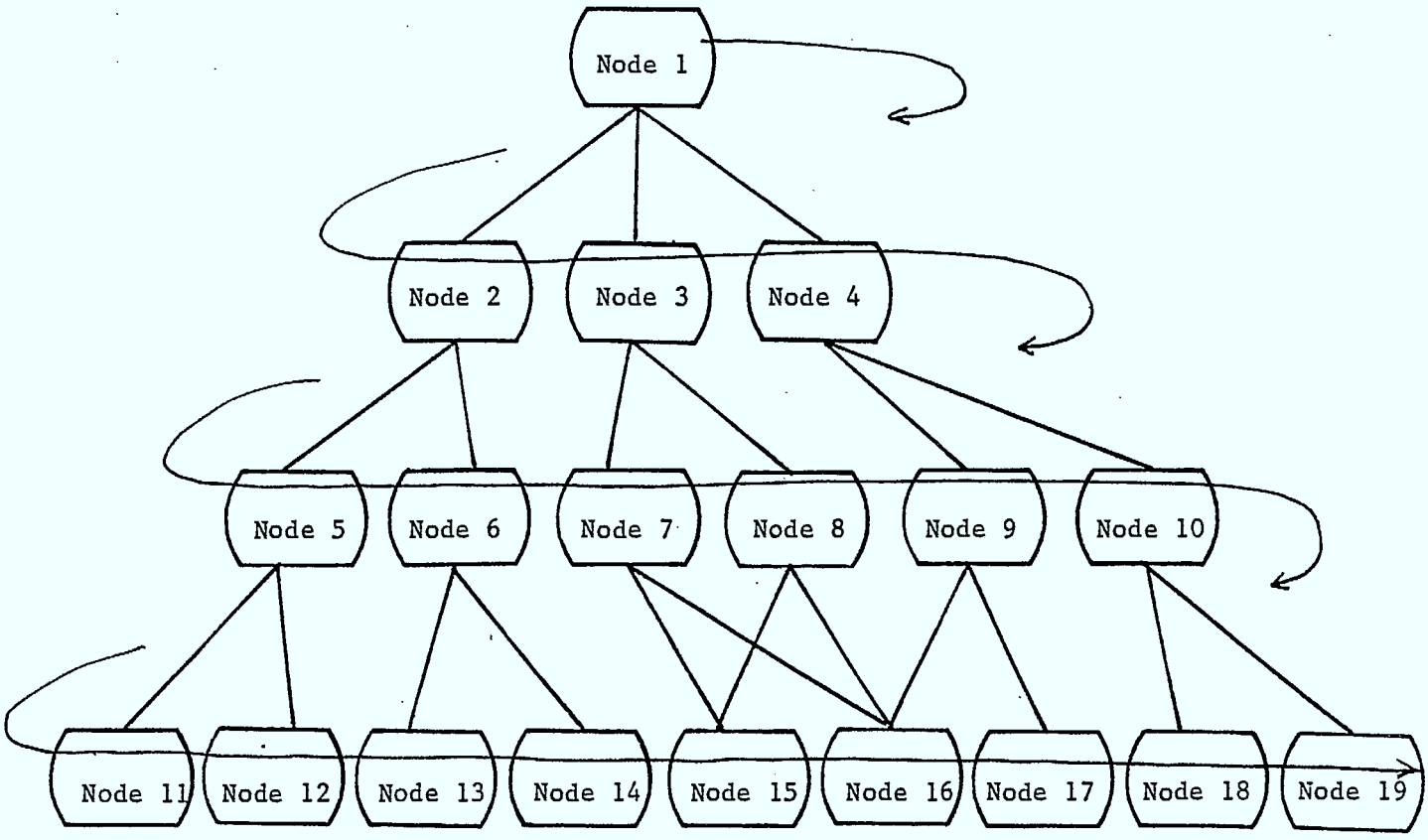

Figure 3.4 The breadth-first search

search is conducted within the chosen branch. All other branches will be ignored. The HLKS selects only one node at a given level when switching to beam search. All nodes in the branch of the tree which are headed by the chosen node will be searched breadth-first. In ordinary beam search, the search terminates when all nodes in the chosen branch are examined. As a twist to ordinary beam search, the control after termination recommences the search at the node next to the node which was chosen before. In subsequent beam searches in the same tree, those nodes which are searched during earlier searches may be re-visited, if they are also a part of newly chosen branch. This variation is so that a number of system problems may be examined in turn, and from different points of view. The HLKS resumes breadth-first search each time it completes a designated beam search. Figure 3.5 shows the operation of the repeatable beam search adapted for the HLKS.

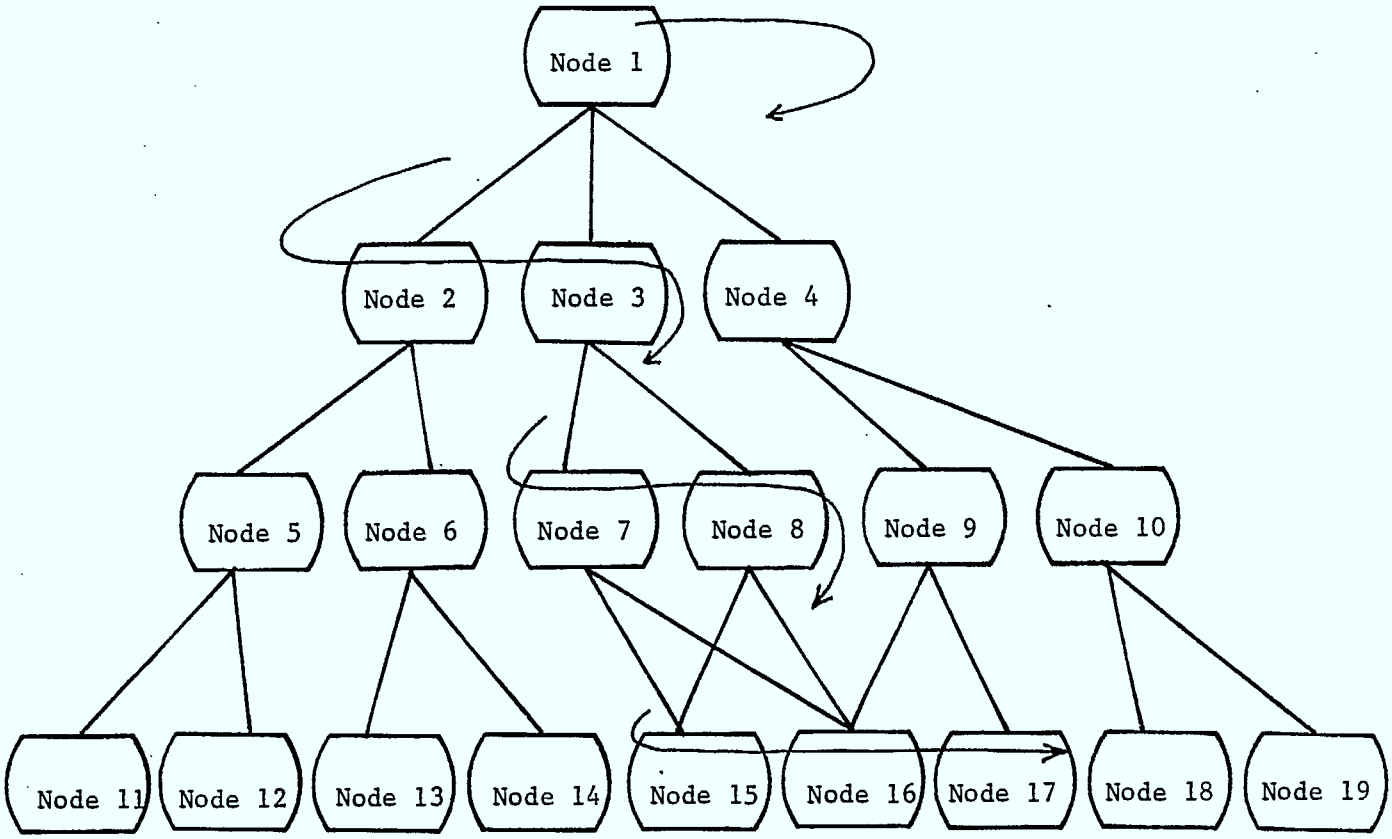

Figure 3.5 The beam search

The approach described above may be explored to accomodate various diagnostic situations. For example, suppose a condition that is strongly linked to the eventual loss of a spacecraft is detected during a routine breadth-first search. The HLKS Autonomy Control also detected at the same search level a condition which will likely result in the loss of a payload function. The HLKS Execution Control, under instructions from a ground controller or from rules in the HLKB, may decide to investigate the first node in further detail. Such an investigation will result in examining all sibling nodes belonging to the first branch. This may or may not remove the cause of the faulty situation which threatens the life of the spacecraft. The second branch headed by the fault which asserts that a loss of a payload function is likely will be examined then regardless of the result of the first investigation. The present implementation does not include the automatic rule-controlled switching of beam search in the middle of a search. It is being considered for future implementation.

In Figure 3.5, the search began using the default search (breadth-first search) strategy. As nodes 3 and 4 are examined and determined to be representative of a faulty condition, the Execution Control awaits at node 3 instructions from the controller as to what should be done. If a beam search is specified, nodes 7, 8, 15, 16, and 17 are then examined. After that control returns to node 4 and awaits further instructions from the ground.

3.3.4.2 The **search** command

The **search** command initiates a search. It takes three arguments, as shown below:

search (Node, Strategy, Length).

The node parameter Node specifies the node from which the search begins.

The strategy selector Strategy sets a search mode for the HLKS Autonomy Control. One of the {**depth**, **breadth**, or **beam**} must be selected. Since any command may be entered at any prompt, the search strategy can be altered after every search step.

The length of a search may be determined by the Length parameter. It can have one of the following values:

- node: The search is interrupted after each node is visited,
- level: The search occurs for one level and the Autonomy Control pauses after all nodes in the current level of the tree are visited. If the strategy is set to depth-first, the search stops after visiting the first node in the current search level,
- branch: The entire branch of which the root is the selected node will be searched without interruption,
- tree: The rest of the tree from the specified node down will be searched without interruption,
- <id>: If a valid node identifier is given, the search continues until that node is encountered. Else, the search exhausts the entire tree without further interruption.

3.3.4.3 The **continue** command

The **continue** command is used to resume the search from where it was interrupted, using the identical search strategy and search length as before. If either of the two is to be altered before resuming the search, the **search** command must be used.

3.3.4.4 The **terminate** command

A search may be terminated anytime by entering a **terminate** command to a prompt. Similarly, a beam search may be terminated and control returned to the default search method at any point during the local search into a branch by entering the **terminate** command to a prompt.

3.3.5 The HLKS Command Module

Some of the commands given to the HLKS are processed by the HLKS Command Module. They are the: **suspend**, **resume**, **entrust**, and **relieve** commands. The last two belong to a group of commands for creating autonomous action nodes for the LLKS. They are implemented in the HLKS Command Module, but parts belonging to the LLKS are not. This feature is beyond the scope of the current POC experimental system. Also, in the future all commands will not only be executed by a human operator but will also be made executable by the HLKS Autonomy Control as a result of a reasoning process. Such a reasoning process will operate on knowledge stored in the COMKB and the HLKB. Currently, a limited number of commands are made executable in this fashion.

In addition, there is a set of utility commands handled by the HLKS Command Module to deal with the maintenance of the system and the knowledge bases. They are: **initialize**, **check**, and **find_top**.

3.3.5.1 The **suspend** command

The **suspend** command suspends the reasoning capability of a node in the inference network during event propagation by the LLKS. This results in the detachment of the node from future event propagation. A faulty system element represented by the node may be 'suspended' from the POC system so that the effects of the removal may be studied. This facility allows the ground controller to examine such effects in simulation. This will aid him in making a decision for or against the actual removal of the element.

The command is executed by the HLKS Command Module by asserting a 'suspend' status in the COMDB in the control slot for the node. The LLKS, during the next propagation cycle recognizes the suspension and in turn asserts a 'suspended' status in the status slot for the node, also in the COMDB.

3.3.5.2 The **activate** command

The **activate** command performs the reversal of the **suspend** command. It removes the 'suspend' assertion in the control slot for the node in the inference network. The LLKS then cancels the 'suspended' status, thus restoring the reasoning capacity of the node again for the event propagation process. This will allow a controller to re-engage a previously suspended system element.

The facility can also be used to simulate the effect of introducing an element of a system which has been kept dormant since the beginning of the operation. System redundancy components such as a 'hot' or a blank spare are of this type. System elements, including these spares, can be either hardware or software, as the node can represent both entities equally well. The effect of introduction may be observed in subsequent propagation cycles by the LLKS and by the higher level functions of the HLKS.

3.3.5.3 The **entrust** command

The HLKS may delegate to the LLKS its authority to autonomously recover from an anomaly, if such a decision will benefit the over-all operation of a spacecraft. The **entrust** command is used for this purpose. If there are situations in which the LLKS might identify an anomaly with potentially very negative implications, and if an examination by the HLKS cannot be conducted quickly enough, a ground controller may decide to allow the lower level expert system to take corrective action without waiting for instructions from him or from the HLKS. The node is said to become an **autonomous action node**, and the **entrust** command is used to designate a node to be one.

The execution of the command by the HLKS Command Module results in the 'entrust' assertion in a control slot for the node. This in turn results in the assertion of 'entrusted' in the status slot for the node, which is acknowledged by the propagation mechanism of the LLKS. Further activation of the node during an event propagation may result in an autonomous invocation of a sequence of corrective or preventive actions defined for the anomaly. Such actions are recorded in the COMKB in the form of rules, procedures, and predicates.

3.3.5.4 The **relieve** command

The **relieve** command removes the ability to autonomously act on an anomaly from an **autonomous action node**. A change in the control slot is translated by the LLKS into a change in the status slot. The node then becomes an ordinary node which passively participates in the event propagation.

3.3.5.5 The **initialize** commands

The **initialize** command initializes the entire POC experimental system. In particular, it resets three

databases: the COMDB, execution databases for the HLKS and the LLKS. This results in the resetting of the status slots of all the nodes of the inference network and the renewal of short-term scratch pad memory for both the HLKS and the LLKS. Some areas of the HLKS-EDB will be reset each time a new search or reasoning takes place. Similarly, portions of the LLKS-EDB are erased at the beginning of each event propagation.

3.3.5.6 The **check** command

The **check** command takes an identifier of a node slot as its sole argument. It scans through the slots of all nodes and checks the completeness of the knowledge base and the connections among the nodes. In the case of the slot that defines the structure of the inference network, the connectivity and the completeness is checked by trying to account for all the evidences of causal relations among the nodes defined in the form of rules. If an evidence is not used in any of the rules or if a causal relationship depends on an evidence that does not exist, an error condition is asserted. The completeness and connectivity checking of all other slots, which store elements of knowledge in various forms, is conducted by using the taxonomy of the network stored in the COMKB as a reference point.

If the **check** command is used without an argument, it checks the structural consistency of all knowledge bases known to the POC experimental system. This operation can take a substantial amount of time.

3.3.5.7 The **find_top** command

The **find_top** command identifies all independent peaks of the inference network. Such an operation becomes necessary when the HLKS or a ground controller wishes to start a search from the highest local node of the network. There can be a number of peaks in a inference network.

3.3.6 The HLKS Explanation Module (not implemented)

The HLKS Explanation Module answers questions or request for additional information issued by a controller: for further clarification of any of the inferences made by the LLKS; on how or why the HLKS Autonomy Control took or failed to take specific action(s). The answers to the questions of the first type often takes the form of a report. The HLKS Explanation Module investigates the inference network, or the instantiated fault tree, retrieves information from it, and compiles output.

The questions of the second type cannot be handled presently as their processing is beyond the scope of the current POC system and is left to future development. For these questions, the explanation subsystem will re-trace and explain the reasoning the HLKS Automy Control made using meta-rules in the HLKB, the object-level (domain specific) knowledge in the COMKB, the instructions given by ground control, or their combination. The output will be composed and formatted for easier interpretation.

The HLKS Explanation Module expects questions or requests for the clarification of the following types of reasoning made either by the HLKS or the LLKS:

- reasoning which resulted in a specific search path,
- reasoning which resulted in a message to ground control,
- reasoning which resulted in autonomous control action(s) initiated by the HLKS.

3.3.6.1 The **report** command

The **report** command compiles a report on a specified node of the fault tree. The fault tree represents the current status of a spacecraft being monitored. The node corresponds to the status of a portion of the spacecraft or an aspect of the spacecraft operation.

It describes the status of the node in question, and how that status was obtained. The command processor re-enacts the data fusion process that took place around the node

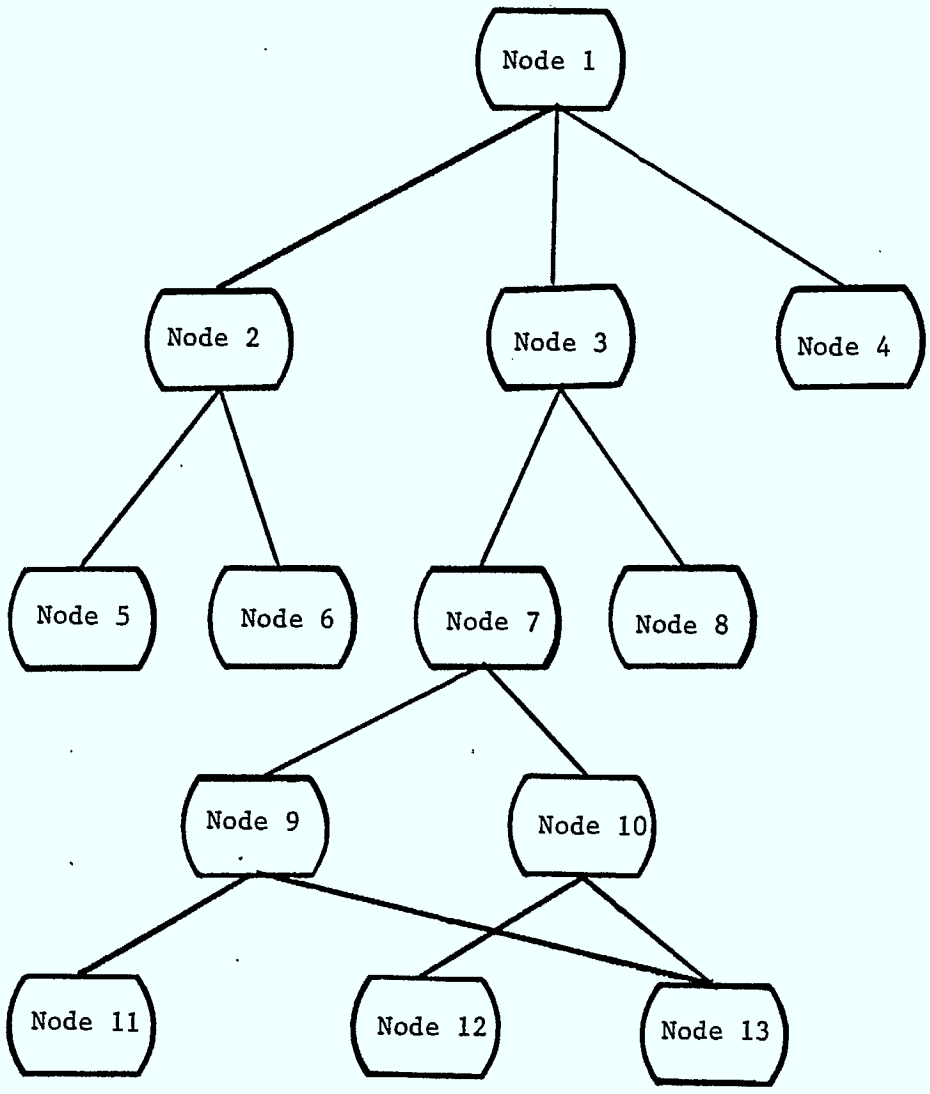

Figure 3.6 Issuing a **report** command

during event propagation conducted by the LLKS. Contributing assertions and their strengths are shown, as well as dependency among supporting items of evidence and the type of

the fusion of assertions that took place. The report may be generated for entire assertions and their supporting assertions that altogether contributed to the status of the node in question. Since this often results in a lengthy output, the report generation may be focused to limit output.

In the partial network depicted in Figure 3.6, suppose Node 3 has issued a warning, which was picked up by the HLKS Autonomy Control and sent down to ground control. The ground control then issued a **report** command, to which the HLKS Command Module responded, as shown below:

(system) WARNING ** (Node 3) **

(ground) **report** (Node 3).

(system) Node 3 (description of Node 3) is true with probability .55.
The state is determined by the rule: Node 3 (**or** Node 7 Node 8).
Node 7 (explanation of Node 7) is true with probability .72.
Node 8 (explanation of Node 8) is true with probability .14.

Enter {report., report([Id1, Id2,...,Idn])., or no.}

The HLKS has described the meaning of the fault about which a warning had been issued, and further described the rule (hypothesis) that fired the warning, and gave supporting items of evidence and their strength. In actual cases 'Node 3' would be 'sensor_malfunction_2', and the its description would be "Star sensor has been generating intermittent outputs". Note that rules are shown in the form:

[consequent (fusion-type antecedent-1, antecedent-2, ..., antecedent-n)]

where, fusion-type may be **and**, **or**, or **not**, and consequent and antecedents are all identifiers of a node in the inference network.

(ground) **report**([Node 7]).

(system) Node 7 (description of Node 7) is true with probability .72.
The state is determined by the rule: Node 7 (**and** Node 9 Node 10).
Node 9 (description of Node 9) is true with probability .76.
Node 10 (description of Node 10) is true with probability .78.

Enter {report., report([Id1, Id2,...,Idn])., or no.}

The **report** command is designed to report on all

relevant hypotheses and their items of evidence which have contributed to the original assertion of a fault. Therefore, it prompts the user each time a hypothesis and a set of evidences are presented. When specifying a selected set of nodes to be reported on, the nodes are put into a list format using [...]. If no nodes are selected, the reporting starts from the items of evidence which support the last hypothesis reported.

(ground) **report.**

(system) Node 9 (description of Node 9) is true with probability .76.
The state is determined by the rule: Node 9 (**or** Node 11 Node 13).
Node 11 (description of Node 11) is true with probability .05.
Node 13 (description of Node 13) is true with probability .98.

(system) Node 10 (description of Node 10) is true with probability .78.
The state is determined by the rule: Node 10 (**or** Node 12 Node 13)
Node 12 (description of Node 12) is true with probability .11.
Node 13 (description of Node 13) is true with probability .98.

By issuing a **report** command without specifying nodes, the entire hypothesis-evidence relationship is reported. Note the **report** does it by beam-searching the nodes involved.

By adding more knowledge based processing to the report generation, the report command may be able to generate output that is better focused. For example, using heuristics such as:

> "If there is a sudden detection of a fault at a highly placed node, look for contributing terminal nodes. If there is one with an abnormally strong assertion, say more than .15, assume it is a cause of the reported fault and report it next to the original faulty node.",

the report process can generate a more to-the-point report. Such improvement is considered for future versions.

3.3.6.2 The **probe** command

The **probe** command works similar to the **report** command, except that it only reports on one hypothesis and a single set of evidences supporting it. It is most useful for spot-checking the inference network by traversing it and probing suspected nodes free from any fixed search strategy.

The example below simulates an instance in which a controller attempts to find out if an on-board thruster maintained its firing after accidental ignition due to an improperly controlled fuel valve, the malfunction of control electronics, and the existence of multiface flow in a fuel line. "O4" is the thruster in question.

(ground) **probe** (o4_firing_continues).

(system) o4_firing_continues (The firing of the offset thruster O4 is maintained) is true with probability .21.

(system) The state is determined by the rule:
[O4_firing_continues (and high_rate_command_continues pressure_in_fuel_line_maintaines o4_fires)]

(system) high_rate_command_continues (The thruster O4 fire command continues at a high rate) is true with probability .96.

(system) pressure_in_fuel_line_maintains (The pressure in the fuel line is maintained) is true with probability .28.

(system) o4_fires (The negative pitch offset thruster O4 fires) is true with probability .51.

(system) Enter command:

The conroller would thus understand that there was a relatively small possibility of the thruster having continued firing, and the reason for that conclusion.

3.3.6.3 The **assess** command

The **assess** command displays the consequence(s) of an existing fault, alone or when combined with other existing faults. The command is used to assess how much impact the fault might have had on further events. Nodes for which the fault is a contributing assertion input are sought after and their present status retrieved. Guided by the structure and type of knowledge stored in the COMKB, the command processor extracts values from the COMDB and composes a report.

An example output of the assessment report is shown below:

(ground) **assess** (large_cone_develops)

(system) (A large nutation cone develops around the pitch axis). This,
(On-board momentum control wheel has stopped) and
(The spacecraft's pitch changes greatly from nominal negative pitch to a large positive pitch)

will jointly cause:

(Spacecraft is tumbling) with probability .13

(system) (A large nutation cone develops around the pitch axis). This,
(The spacecraft is not receiving command sequences), and
(The spacecraft's pitch changes greatly from nominal negative pitch to a large positive pitch)

will jointly cause:

(Attitude control is no longer effective) with probability .15.

(system) (A large nutation cone develops around the pitch axis). This, and
(The spacecraft's pitch changes greatly from nominal negative pitch to a large positive pitch)

will jointly cause:

(The solar arrays are not facing the sun) with probability .14.

(system) (A large nutation cone develops around the pitch axis). This, or
(The spacecraft's pitch changes greatly from nominal negative pitch to a large positive pitch)

will jointly cause:

(The command receive antenna is not properly aligned to the ground) with probability .38.

(system) (A large nutation cone develops around the pitch axis). This, or
(The spacecraft's pitch changes greatly from nominal negative pitch to a large positive pitch)

will jointly cause:

(The SHF communication channel is lost) with probability .31.

The present implementation only seeks implications one level above the node in question. In future improvements, the

assess process can be designed to chase further implications recursively to an arbitrary height in the hierarchy. This will give a reverse-**report** facility. Similar protocol used for the **report** command would be introduced to control the extent of reporting.

3.3.6.4 The **recommend** command

A recommendation on how to contain a detected anomaly, or how to restore the affected system elements from it, may be obtained for a ground controller when he issues a **recommend** command. Such information is stored also in the COMKB and would be sent out to the OIU. The SAMS works as a database in such instances, providing the ground controller with advice. Although the controller may be aware of all the implications and actions to be taken in given situation, he may not be able to retrieve them in time, in a pressing atmosphere of a control room which is coping with a serious operational difficulty.

The **recommend** command is typically issued when a ground controller notices a warning message from the SAMS and requests a report on it. It may go as follows:

(system) WARNING: ** (uvs_trips) **

(ground) **report** (uvs_trips).

(system) uvs_trips (Under voltage protection system is activated)
is true with probability .17.

The state is determined by the rule:

uvs_trips (and batteries_exhausted charging_limited)

batteries_exhausted (On-board batteries are exhausted)
is true with probability .93.

charging_limited (Solar array's ability to charge on-board batteries is now limited) is true with probability .16.

(system) Enter {report., report([Id1, ..., Idn])., or end.}

(ground) end.

(system) Enter command:
(ground) **recommend** (uvs_trips).

(system) RECOMMENDATION (Under voltage protection system is activated):

** Disable UVS **

(system) Enter command:
(ground) **explain** (uvs_trips).

(system) In the state of (Under voltage protection system is activated) with probability .17, the UVS is known to malfunction.

The current POC implementation only retrieves such information stored in the form of steps to be taken for each anomalous situation. In the future, the recommendation would be edited to create a more tailored output sensitive to minute but essential situational differences that may exist each time it is requested.

3.3.6.5 The **explain** command

The **explain** command is used when a ground controller wishes to know the justification for a recommendation obtained by the **recommend** command. See its use in the above example. The present implementation of the **explain** command in the POC experimental system retrieves a message composed of stored texts, node variables, and some system variables. In the future, the explanation will take a form of a description of results from a simulated propagation-assessment session in which the implication of implementing a recommendation is explained by actually propagating events on a subnetwork consisting of the recommended changes and nodes around it.

4. The Low-Level Knowledge-based System (LLKS)

4.1 Objectives of the LLKS

Through inferences the LLKS acts as an intelligent agent overseeing the monitoring functions of a spacecraft management system. The LLKS also executes commands sent from the High Level Knowledge-based System (HLKS). In this capacity, the LLKS accepts orders given in the form of a fixed number of commands, executes them, and then reports the results to the HLKS.

More specifically, the LLKS tries to accomplish the following goals:

- Execute the data/event driven inference on data or events collected from the environment, both external and on-board, and report the results to the HLKS,
- Carry out a set of operations autonomously when designated to do so by the HLKS,
- Process system control commands sent down by the HLKS for execution,
- Provide explanations to the HLKS on the reasoning it made.

4.2 Functional Structure of the LLKS

The functional structure of the LLKS is shown in Figure 4.1.

The LLKS Interface links the LLKS to the HLKS and to the Environment Interface Unit (EIU) of the POC experimental system (See Figure 2.1). The LLKS Interface is responsible for dispatching any arriving data/events, messages or commands to an appropriate subsystem of the LLKS, and for collecting and sending out messages and reports generated by the LLKS to other components of the POC experimental system.

The LLKS Execution Control coordinates the over-all operation of the LLKS. It schedules the operation of other components of the LLKS by dispatching both incoming commands and commands generated by itself.

The LLKS Inference Engine performs propagation of events through the inference network using probablistic reasoning. Since the knowledge about the spacecraft system, whose operation is to be monitored is structured into a fault tree, the inference network after the propagation becomes an instantiated fault tree. It is stored in the COMDB.

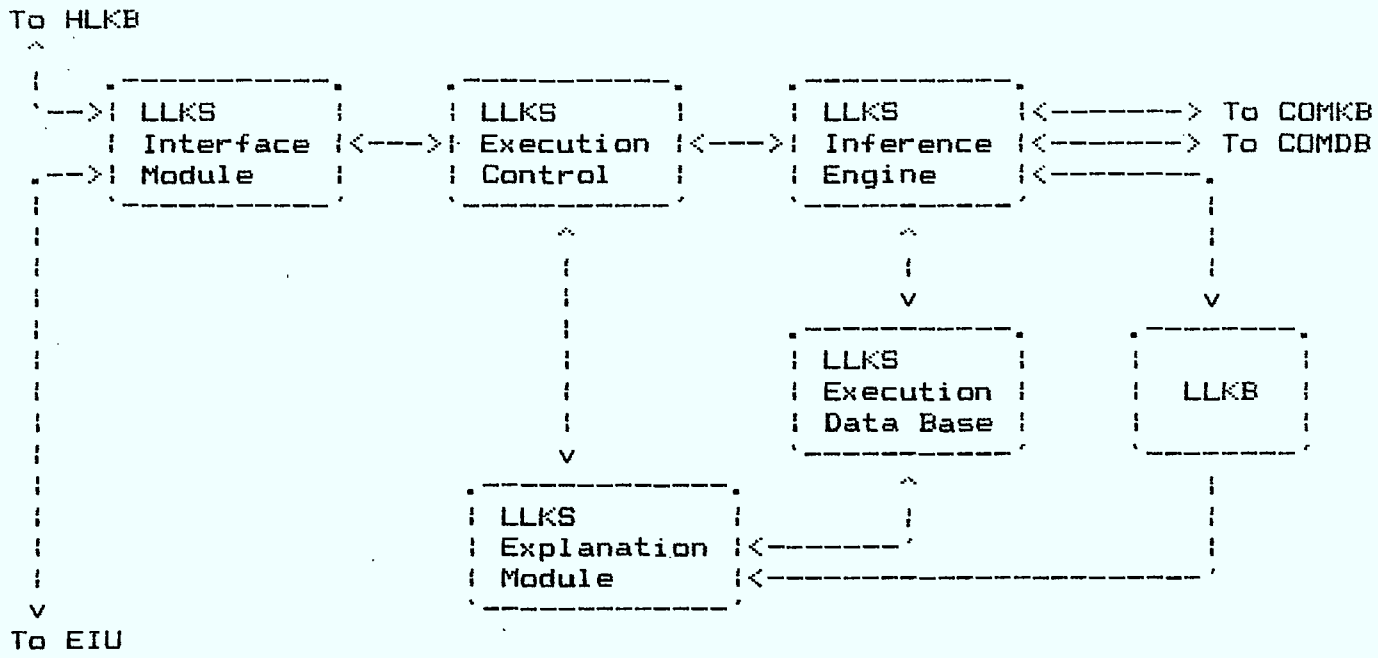

Figure 4.1 Functional Structure of the LLKS

The LLKS Execution Data Base is used by the inference engine as a temporary storage for intermediate results of the reasoning process.

The LLKS Explanation Module processes requests for clarification of reasoning undertaken by the LLKS. The requests are either issued by a ground controller and relayed by the HLKS or generated from within the HLKS.

4.3 Operation of the LLKS

4.3.1 The LLKS

The LLKS performs data/event-driven inference on the inference network using knowledge stored at the node level. It propagates the effect of detected events or incoming data using a so-called data fusion model, a form of information processing commonly used in the Signal Processing community. Data fusion is, in short, a method of finding out how changes in the operational environment will affect other aspects and levels of the spacecraft management.

After evaluation of several available models of data fusion, some sophisticated but not practical, others too simplistic, Rauch's dependence sensitive model [Rauch 84] was chosen for the LLKS.

The LLKS responds to assertions of events such as faults detected by sensors in on-board and external environments. These changing assertions are supplied in the POC experimental system in the form of simulated input signals from the Environment Interface Unit (EIU). Selected input may be entered manually through a console at the begining of each simulation cycle through a console attached to the EIU. This console differs from the operator interface described in Section 3.3.1, in that, the former is for controlling the process of the experiment, while the latter is a simulated operator console dedicated to ground control functions.

The LLKS then performs fusion of assertions through probalistic gates, or nodes. A fusion is conducted by attempting to prove a hypothesis (the invocation of a rule defined at the node) by applying a forward inference on the rule. A node typically corresponds to a fault or faulty situation in a spacecraft. For example, it can be "Thruster No.2 fires intermittently", "The sun sensor lost the sun from its view", "The cylistor in the voltage regulator is stuck open", or "The frame of the satellite is vibrating vigorously".

These nodes are formed into what is called an inference network, a network over which assertions are propagated via inference. Each node contains rules which dictate how the fusion should take place. These rules are collectively stored in the Common Knowledge Base (COMKB). The output of a data fusion process is the strength of assertion that the node represents as supported by the input hypotheses

of the node. It ranges from zero (false) to one (true), and can take a value between the two. These logical outputs (assertions qualified by their strength) are propagated upward in the fault tree (toward the root of the tree) to a set of nodes at the next higher level in the hierarchy. The process repeats until all possible assertions are propagated and the system reaches an equilibrium.

The objective of the fusion process is to determine the implications of a set of changes that occurs in the environment and within the system itself. To do so, symbolic reasoning is used, not calculation or computation. This use of symbolic reasoning on stored knowledge distinguishes the method from other conventional signal processing techniques. By applying inferences, rather than numeric calculations and comparisons, therefore using Artificial Intelligence techniques, one can hope to create a system that, after several refinement cycles, would eventually match some of the capabilities of human thought processes. Intelligent cabilities that humans display are well above what existing machinery so far has reproduced.

Operation of each of the component modules of the LLKS is described in Sections 4.3.2 through 4.3.5.

4.3.2 The LLKS Interface Module and the EIU

The LLKS Interface Module exchanges data, events, and messages with the POC system modules outside the LLKS.

4.3.2.1 The interface with the HLKS

The HLKS hands down commands that must be executed by the LLKS. Some of them are from the OIU representing the controller's requests, while others are generated from within the HLKS by its reasoning process. Results of executing these commands will be passed to the HLKS as they become available.

Commands sent down by the HLKS and executed by the LLKS are as follows:

- **initialize_llks:** Initializes the LLKS,
- **disp_tree:** Displays the inference network in a simulated tree format, before or after event propagation,
- **disp_ref:** Displays references from which knowledge was obtained.

- **disp_exp:** Displays a rule expression,

- **check_loop:** Finds a loop in the structure of a knowledge base,

The following commands are issued or passed down by the HLKS. However, their invokation has already been made and the LLKS only adjusts its internal data structures in accordance with a specific command.

- **suspend:** Suspends the data fusion capability of a node This command is executed implicitly through slots. Full description of the command is given in Section 3.3,

- **activate:** Engages a node to data fusion process. Executed implicitly. Full description of the command is given in sSection 3.3,

- **entrust:** Designates a node as an autonomous action node. An autonomous action node will initiate a predefined action when a set of predetermined conditions is met,

- **relieve:** Relieves a node from being an autonomous action node.

4.3.2.2 Interface with the EIU

Inputs from the environment come to the SAMS in the form of data (eg., an input voltage) or logical assertions or events (eg., contact sensor output, confirmation of object by a vision system). The numerical values are converted into strengths of assertions which have a nominal value between 0 and 1. The values are given to corresponding terminal nodes.

In the POC experimental system, the EIU generates these values using a random number generator, or by prompting the conductor of the experiment for values. In the case of internal generation, an output from the random number generator is modulated to reflect relative occurances of the events to be generated. In favor of observing simulation results faster than would be in real time, the absolute probability of generating faults is amplified substantially (say, 100 times) to create deliberately unstable spacecraft operating conditions.

During the initialization of the POC experiments, the system initialization module requests the experimenter to specify the mode of input generation. For each terminal node the experimenter may choose from:

- **random:** input for the terminal is generated by using a random number and the probability of the occurance of the event assigned to the terminal,

- **fixed:** input for the terminal is fixed for the entire duration of the experiment either to a **high** or to a **low**. A **high** corresponds to the strength of assertion for the event assigned to the terminal being one, while a **low** corresponds to that being zero,

- **manual:** input for the terminal will be entered manually through the experiment control console at each simulation cycle in the form of the strength of assertion for the event the terminal is assigned to.

4.3.3 The LLKS Execution Control

The LLKS Execution Control coordinates the operation by dispatching commands to other portions of the LLKS. Most commands come from HLKS. However, the most important command of all for the LLKS, the **propagate** comand, is generated by the LLKS Execution Control itself. The **propagate** command maintains the regular propagation cycle. For each issuance of the command, one full propagation cycle follows. It ends when an equilibrium is reached in the inference network and no further inference can be made for the given set of input assertions. The process is detailed in Section 4.3.4 below.

Some of the commands the LLKS Execution Control dispatches are executed by itself. For example, the **initialize_llks** command gets executed by the LLKS Execution Control initializing specific sections of the COMKB. The **suspend, activate, entrust,** and **relieve** commands are all initiated by the HLKS. The LLKS detects the assertions made in the control slots of the node and makes appropriate assertions in the status slot. If, for example, the **entrust** command is issued to a node by a ground controller, an 'entrust' assertion will be made by the HLKS in the control slot for the node. This then is translated into 'entrusted' assertion in the status slot for the node by the LLKS Execution Control. The LLKS Inference Engine honors the new status each time it propagates events thereafter.

4.3.4 The LLKS Inference Engine

The LLKS Inference Engine has the following functional characteristics:

- Performs inference based on data fusion,
- Performs inference on an inference network of arbitrary topology connected by AND, OR, NOT, and terminal gates,
- Performs inference probabilistically,
- Performs inference on an inference network with a loop - a chain of reasoning which returns to an earlier premise - with some limitations.

To describe the operation of the inference engine, a simple inference network shown in Figure 4.2 is used.

In the inference network the following assumptions are made. Note a terminal node is an input point for external data and events:

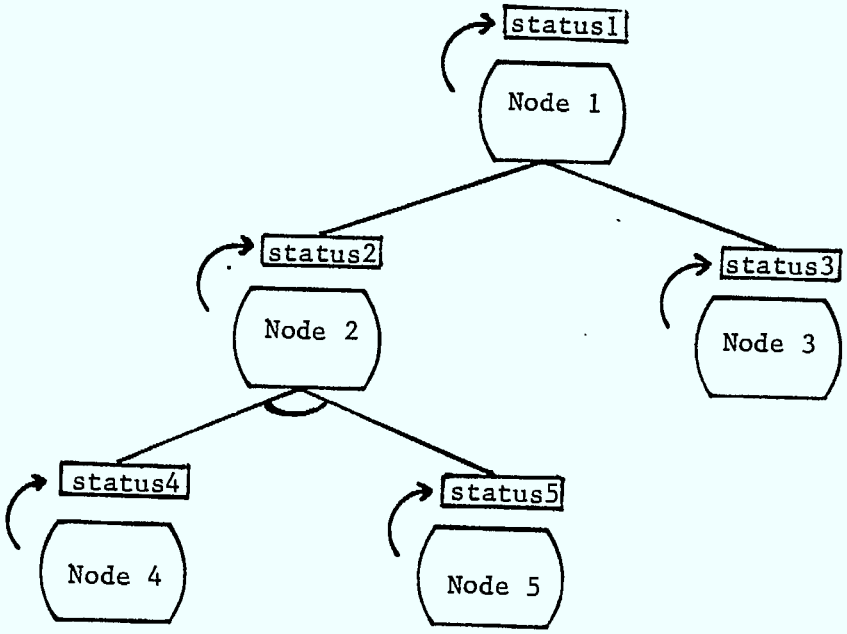

Figure 4.2 An example inference network

- Node_1 is an OR gate with two items of evidence, Node_2 and Node_3,

- Node_2 is an AND gate with two items of evidence, Node_4 and Node_5,

- Node_3, Node_4, and Node_5 are terminal nodes with only one input.

In propagation, each node is treated independently of others. Inference is made solely based on the strength of a node's supporting items of evidence, and stores the result (an assertion) in its own status slot. It is stored structurally in the Common Knowledge Base (COMKB).

In one inference, the LLKS INference Engine retrieves knowledge associated with a node from the COMKB. The order of processing is arbitrary. Assuming in the example, knowledge about the nodes is stored in the order of appearance in the diagram, one cycle of propagation looks as follows:

(1) The inference engine finds out that items of evidence needed to support Node_1 are Node_2 and Node_3. Since Node_3 is a terminal node, its value is obtained without delay from the EIU. However, no inference has happened for Node_2 yet. Its strength is still undefined (the status slot of the node is empty). No fusion can take place for Node_1.

(2) At Node_2, items of evidence needed are found, from the COMKB, to be Node_4 and Node_5, and the type of inference AND, as well as the degree of dependence. Since both Node_4 and Node_5 are terminal nodes, the status slots for these nodes are already filled. An AND fusion takes place using Rauch's AND fusion model. The result is stored in the status slot for Node_2 in the COMDB.

(3) Since Node_1 now has the needed items of evidence, an inference in the form of fusion takes place in the way described in the Rauch's OR fusion model. The resulting strength of assertion is stored in the status slot of the node in the COMDB.

The terminal nodes have no evidences to fuse by themselves but assertions are obtained through an input terminal of the inference network. A node connected to an output terminal of a sensor is expected to convert its signal

into a logical assertion. In the POC experimental system, terminal assertions are generated by simulation or through manual input from the experimenter, and are provided through the EIU. In the example, the strength of assertions obtained through the terminals is made available and stored in the status slots for Node_3, Node_4, and Node_5, in the COMDB, whenever necessary.

To improve the performance of the inference engine, a mechanism which sequences inference to economize data collection is devised. In the example of Figure 4.3, Node_1, an AND node, can be asserted if all supporting items of evidence, Node_2 through Node_5, have an assertion. Node_6 and Node_7 (terminal nodes) are examined first and the strength of assertion collected. If the fusion at Node 5 does not yield a positive assertion, no collection of values is attempted for other terminal nodes.

The inference engine performs data fusion in an 'arbitrary' order, picking a node as it appears in the knowledge base. The network, in general, is highly irregular in its topology. There will be some terminal nodes much higher or lower in the hierarchy than other terminal nodes. This eliminates the possibility of picking the nodes with

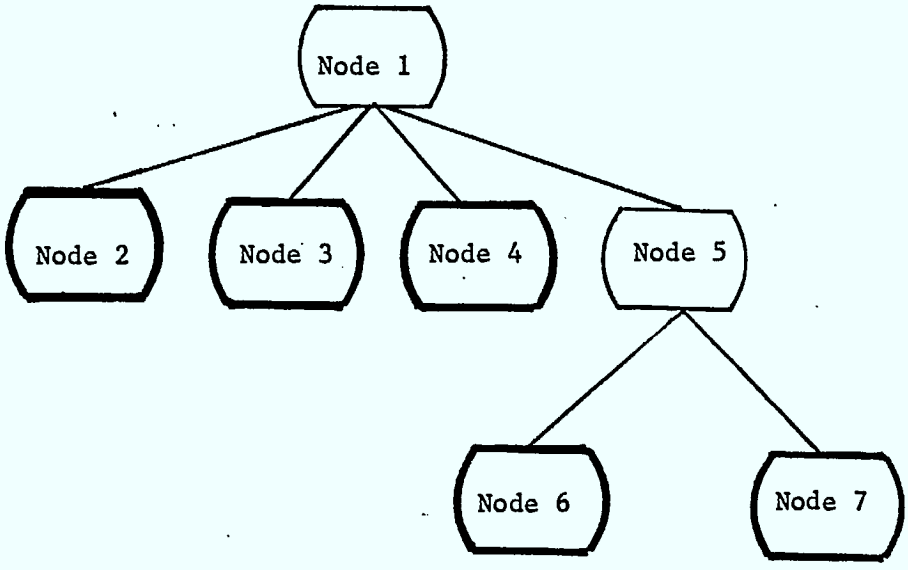

Figure 4.3 Sequencing of data fusion process

readily availaible items of evidence first. Following the topology precisely and performing fusion strictly according to the structure of a tree or a network creates an inhibitive amount of house-keeping for an improvement which is not guaranteed. Testing and repeating the fusion process until all nodes are treated seems to be the only practical and effective method.

If a node is in the state of being 'suspended', its existence is completely ignored in the fusion process. It simply does not exist.

Handling of uncertainty is one significant aspect of human reasoning. It is generally accepted that the human operator's ability to cope with incomplete information and uncertain rules is a source of the flexibility, and the strength of the human reasoning process. It is also acknowledged that researchers are still very far away from understanding precisely why and how a human's way of handling uncertainty is effective [McCarthy 84]. There are only a very limited number of methods and models of how it should be done. These methods are grossly limited in handling various aspects of uncertainty in reasoning. The recent rise of knowledge based systems seems to have only highlighted the lack of adequate approaches.

Three approaches are investigated for the LLKS. They are:

- Fuzzy logic [Zadeh 76] [Winston 84],
- Bayesian approach [Duda, et al 81],
- Tactical data fusion method [Rauch 84],
- The MERIT model by James Slagle [Slagle 84].

Fuzzy set theory appears to be a special case of Rauch's tactical fusion model. Bayesian approach, widely used in expert systems, including MYCIN and PROSPECTOR, ignores the treatment of dependence among supporting items of evidence, a critical shortcoming in an application where the use of redundancy is essential for creating fault-tolerance in a system. For example, use of multiple sensors, such as multiple earth sensors on-board a spacecraft, is common practice for increasing the reliability of measurement. However, it cannot be accurately modeled using Bayesian approach.

The MERIT model used by Slagle in his BATTLE expert system requires realtime computation of a system of partial differential equations, an approach we could not adopt due to resource limitations. Rauch's method of handling probability and dependence at the same time, which is a part of the data fusion model already described, appears to be a reasonable compromise, in comparison, for a prototype realtime expert system.

Dempster-Shafer Theory is at the root of Rauch's method. The authors did not get original material for the theory [Shafer 76] until too late to study in detail for the POC system. In all, it seems the study of uncertainty must be continued in AI for much longer. Rauch's method, for example has an obvious limitation, in that it can take only one dependence value among inputs to a node. In practical cases, dependence may differ among pairs of items of evidence that support a node (hypothesis). Furthermore, dependence in general can be directional between two items of evidence. For example, evidence A may depend heavily on evidence B, but not vice versa. The following is a summary of Rauch's tactical fusion method.

Rauch's method calculates standard deviation, as well as the probability of the strength of assertion after fusion, and propagates both of them through data fusion. It can handle cases where items of evidence are not necessarily independent but where there is a statistical dependence among supporting items of evidence.

Table 4.1 shows the probability of a hypothesis (or the strength of assertion that a hypothesis is true) calculated for probablistic AND and OR gates with two pieces of evidence.

Table 4.1 Probability of hypothesis with two items of evidence

Statistical Dependence	AND operation: PROB (A and B)	OR operation: PROB (A or B)
Independent	Pa * Pb	Pa+Pb-Pa*Pb
Maximum Dependence	MIN (Pa, Pb)	MAX (Pa, Pb)
Negative Max Dependence	MAX (Pa, Pb)	MIN (Pa+Pb, 1)

Pa and Pb are the probability that evidences A and B are true, respectively. MIN and MAX are a selection function. Consequence of logical AND and logical OR operations can be obtained for various degrees of independence between supporting items of evidence: independence, maximum dependence, and negative maximum dependence. The last case implies a situation when 'A is most unlikely if B is asserted'. Interpolaration is used to obtain values between extremes.

For a hypothesis with more than two items of evidence, Rauch's model does not provide ways for calculation. Table 4.2 is an extension made by the authors for multiple items of evidence.

For example, if there are two items of evidence,

PROB (A or B) = Pa + Pb - Pa * Pb

for three items of evidence,

PROB (A or B or C)
= Pa + Pb + Pc - Pa * Pb - Pb * Pc - Pc * Pa
+ Pa * Pb * Pc.

Table 4.2 Probability of a hypothesis with multiple items of evidence

	AND operation: PROB (1,2, ..., and N	OR operation: PROB (1,2, ..., or N)
Independence	P1 * P2 *, ..., * Pn	Note 1
Maximum Dependence	MIN (P1, P2, ..., Pn)	MAX (P1, P2, ..., Pn)
Negative Max Dependence	MAX (P1 + P2 ... + Pn - (N-1), 0)	MIN (P1 + P2 ... + Pn, 1)

Note 1: Probability for independent OR may be obtained from the Euler's chart.

The interploration is performed by first calculating the probability under the assumption that the items of evidence are independent (probability from these calculations will be designated C1). When the dependence D is positive, the second calculation is of maximum dependence (designated C2). When D is negative, the two calculations are probabilities under the assumption of independence (C1) and

under the assumption of minimum dependence (C3). The resulting probability is a linear combination of the two appropriate calculations:

$$P = \{D * C2 + (1-D) * C1, \text{ for } 0 <= D <= 1\}$$
$$= \{ABS(D) * C3 + (1 - ABS(D)) * C1, \text{ for } -1 <= D <= 0\}$$

where, ABS is the absolute function.

One of the limitations of the Rauch's method, the lack of universal treatment of dependency among items of evidence, was discussed earlier. Another difficulty with the method is that as the number of items of evidence increases, it becomes cumbersome to deliver an equation for calculating probability for OR cases.

4.3.5 The LLKS Explanation Module

The LLKS Explanation Module provides low level explanation to ground control. There are five commands belonging to this module. They are the **disp_tree**, the **disp_exp**, the **disp_ref**, the **check**, and the **check_loop** commands. In the future the HLKS Autonomy Control may invoke these commands on behalf of a controller from within a reasoning process.

4.3.5.1 The **disp_tree** command

The **disp_tree** command displays the inference network as a tree-like data structure. This conversion of formalism is so that the hierarchical nature of an inference network becomes visible. The display depicts both the structure and the status of nodes in a network.

The command may be applied either to a non-instantiated tree (before event propagation) or to an instantiated (after event propagation) tree. An example of a simple inference network is shown in Figure 4.4. An example ouput which corresponds to the example network before propagation is shown in Figure 4.5.

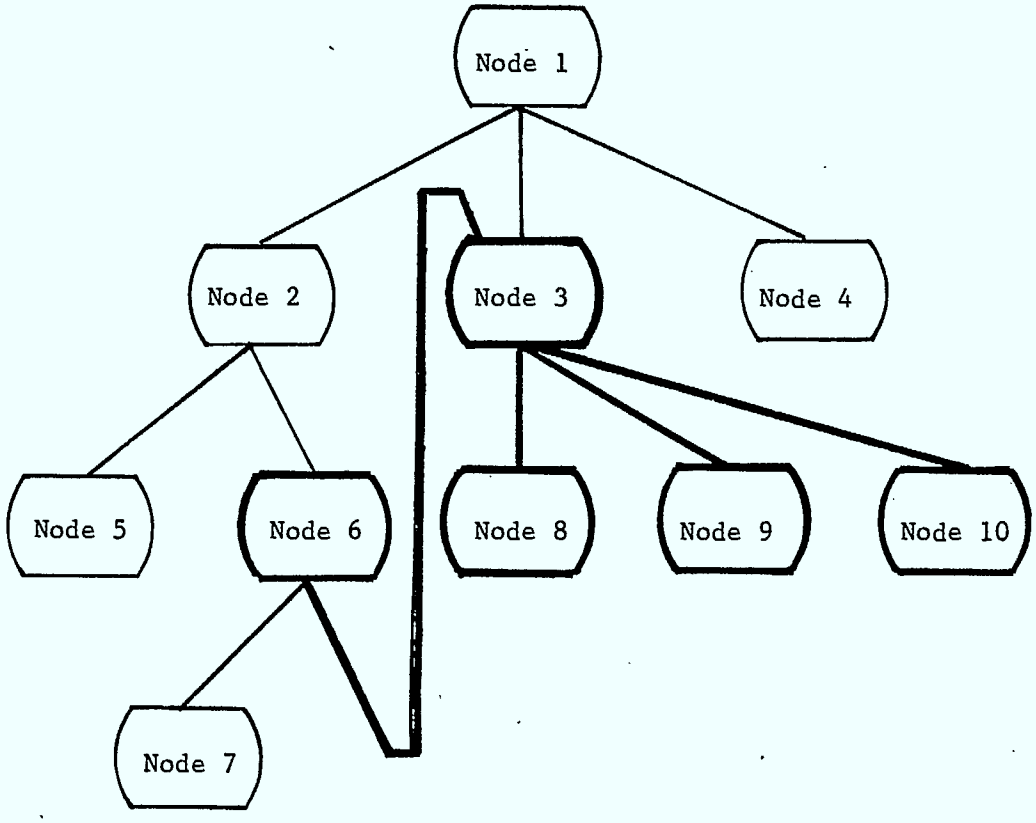

Figure 4.4 An example inference network

Note in Figure 4.5 that nodes 3, 8, 9, and 10 appear twice in the display. This is because there is a link between the output (assertion) of node 3 and input (evidence) of node 6, constituting a logical loop.

```
  (user) disp_tree (node_1).
(system) node_1 (and, 3)  D=.8
             node_2 (and, 2)  D=.5
                 node_5 (terminal)
                 node_6 (or, 2)  D=.4
                    node_7 (terminal)
                    node_3 (and, 3)  D=.7
                       node_8 (terminal)
                       node_9 (terminal)
                       node_10 (terminal)
             node_3 (and, 3)  D=.7
                node_8 (terminal)
                node_9 (terminal)
                node_10 (terminal)
             node_4 (terminal)
```

Figure 4.5 The **disp_tree** command applied before propagation

The result of propagation is seen in Figure 4.6, in which the **disp_tree** command was applied after the propagation of events by the LLKS Inference Engine took place. The outputs of the **disp_tree** command follow a format described in Figure 4.7. Asterisks(*) placed in front of some of the lines imply these node have fired as the result of propagation. Probability P will not appear for nodes if propagation did not affect them. Dependency D will not appear in terminal nodes.

```
(ground) disp_tree (node_1).
(system) node_1 (and, 3)  D=.8
             node_2 (and, 2)  D=.5
                node_5 (terminal)
              * node_6 (or, 2)  D=.4 P=.68
                    node_7 (terminal)
                  * node_3 (and, 3)  D=.7 P=.86
                     * node_8 (terminal)  P=.75
                     * node_9 (terminal)  P=.34
                     * node_10 (terminal)  P=.91
             node_3 (and, 3)  D=.7
              * node_8 (terminal)  P=.75
              * node_9 (terminal)  P=.34
              * node_10 (terminal)  P=.91
            node_4 (terminal)
```

Figure 4.6 The **disp_tree** command applied after propagation

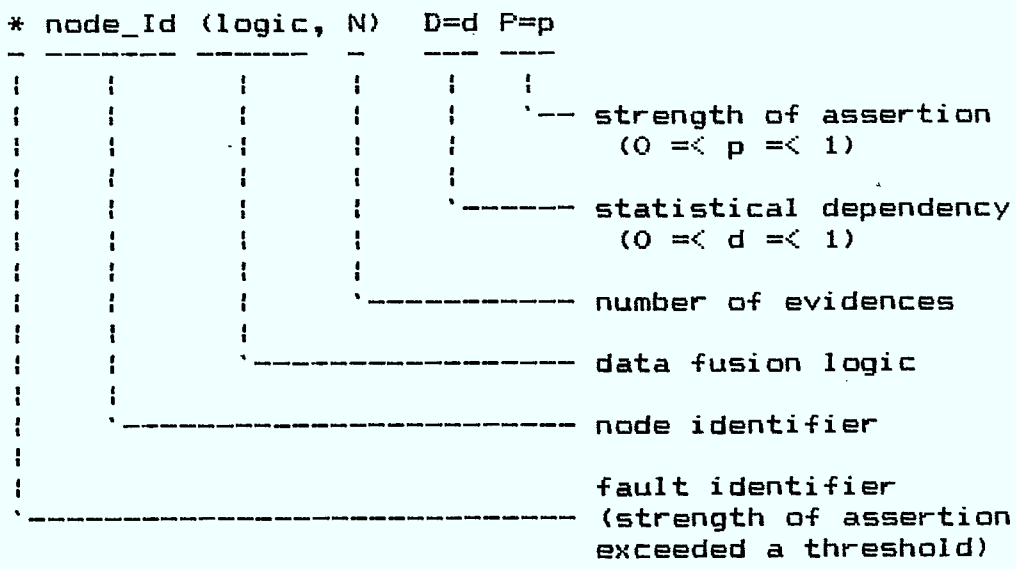

Figure 4.7 **disp_tree** display format

In figure 4.6, node 6 is marked by an asterisk, meaning that a faulty condition exists at that node. To find out what has caused it, contributing nodes, nodes 3 and 7 must be looked at. Thus node 3 is found to be faulty. Since node 6 is an **or** node, this is a sufficient condition to cause a faulty status in that node. Since node 3 is an **and** node, all three contributing evidences, nodes 8, 9, and 10 must be faulty. Nodes 1 and 2 are not affected as they are both **and** nodes and only one of the contributing evidences is faulty in each case. As shown here, the **disp_tree** may be used to trace where a fault originates and how it has spread in the system.

Parameters may be given to the disp_tree command to focus the area of display on the tree. There are three command formats:

- **disp_tree** (NO)

  All nodes of the tree directly or indirectly subordinate to NO will be displayed,

- **disp_tree** (NO, N1)

  Nodes that are directly or indirectly subordinate to NO, but no deeper than N1 branchings away from NO are shown, as seen in Figure 4.8,

```
disp_tree (node_1, 2).

node_1 (and, 3)  D=.8
   node_2 (and, 2)  D=.5
      node_5 (terminal)
    * node_6 (or, 2)  D=.4 P=.68
   node_3 (and, 3)  D=.7
    * node_8 (terminal)  P=.75
    * node_9 (terminal)  P=.34
    * node_10 (terminal)  P=.91
   node_4 (terminal)
```

Figure 4.8 Nodes within 2 branchings from node_1

- **disp_tree** (N0, N1, N2)

  The display begins at the depth N1 from N0 and all nodes no deeper than N2 from N0 are displayed, as shown in Figure 4.9.

```
disp_tree (node_1, 3, 4).

node_7 (termianl)

* node_3 (and, 3)  D=.7 P=.86
   * node_8 (terminal)  P=.75
   * node_9 (terminal)  P=.34
   * node_10 (terminal  P=.91
```

Figure 4.9 Nodes 3 branchings away from node_1 but no farther than 4 branchings away

4.3.5.2 The **disp_ref** command

The **disp_ref** command retrieves information regarding the source of knowledge from the COMKB. This information is stored in the reference slot of each node. The example below shows the usage of the command:

```
(user) disp_ref (large_cone_develops).

(system) large_cone_develops (and, 2): D. Andean, Interview, 29 AUG 84.
            o4_firing_continues (and, 3): CTS Operations Report 3.4.
            negative_pitch_develops (or, 1): Earlier report, Sep. '78.
```

The report provides accountability to knowledge being used in the SAMS and aids the knowledge update or revision process. The command has the same node-focusing mechanism as in the **display_tree** command described in Section 4.3.5.1 above, and the following command formats, described there, are acceptable:

display_ref (NO).

display_ref (NO, N1).

display_ref (NO, N1, N2).

4.3.5.3 The **disp_exp** command

The **disp_exp** command retrieves a rule which supports a hypothesis at a node. The rules are retrieved from the COMKB and have the format show in the following example: The **disp_exp** command displays only one node at a time.

```
  (user) disp_exp (spacecraft_lost).
(system) [ spacecraft_lost (and electronics_inert telemetry_lost
         antenna_ineffective spacecraft_mechanically_frozen) ]
```

The rule shown is equivalent in English to:

IF on-board electronics are inert,
and the antenna for the command link is ineffective,
and the telemetry from the spacecraft is lost,
and the spacecraft is mechanically frozen,
THEN the spacecraft is assumed to be lost.

The general format of a rule is as shown in Section 3.3.6.1, ie.,

[node (logic Evidence-1, Evidence-2, ..., Evidence-n)]

where,

node: Node (hypothesis) identifier, or the name of the rule,

logic: Data fusion logic, **and**, **or**, or **not**,

Evidence-i: identifier of a node which contributes to the hypothesis.

4.3.5.4 The **check_loop** command

The **check_loop** command is for finding a loop in the knowledge structure. A node is specified as the sole argument and the command processor notifies whether the node is a part of a loop or not. An example of a loop is shown in Figure 4.9. THe following sequence is an example of applying the command on node_3 and node_4 of the example network.

```
(user) check_loop (node_13).

(system) Node node_13 is in a loop.

(user) check_loop (node_14).

(system) Node node_14 is not in a loop.
```

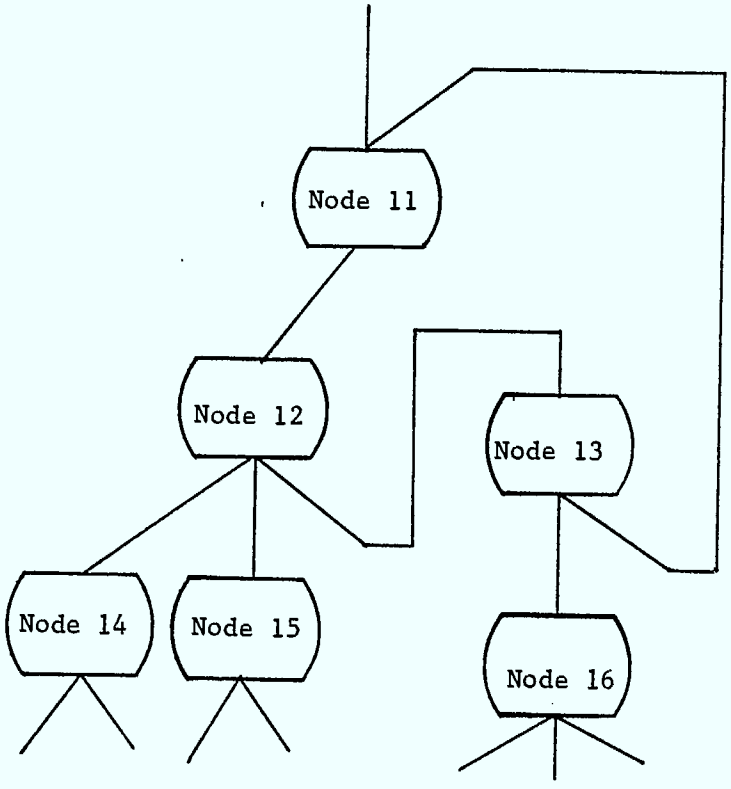

Figure 4.9 An example of a loop in an inference network

5. The Experiments

5.1 The LLKS test and experiment

5.1.1 The objective of the experiment

To test and confirm proper functioning of the LLKS' event propagation mechanism as the basis of the fault inference methodology used in the POC experimental system.

5.1.2 The method of experiment

Using a configuration of Figure 5.1, conduct the experiment in the folowing order:

(1) Initialize the COMKB so that any previous knowledge is eliminated,

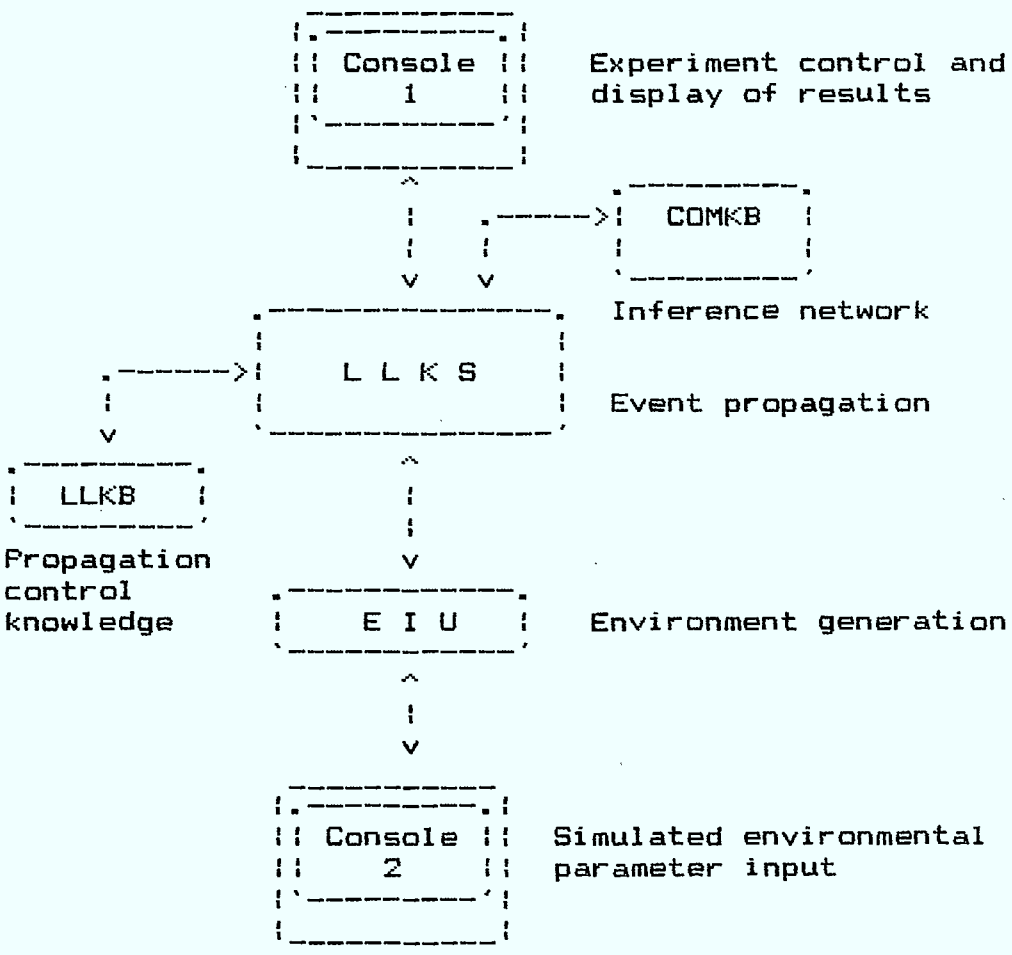

Figure 5.1 The event propagation test facility

(2) Establish in the COMKB, a set of knowledge each piece of knowledge representing an inference node. Select inference network (and domain) appropriate for testing propagation of events. Use the **disp_tree** and other commands to confirm the network in the COMKB,

(3) Using Console 1, invoke the LLKS and start its propagation mechanism,

(4) Enter simulated environmental inputs through Console 2 and let the values propagate,

(5) Wait till the propagation reach its equilibrium,

(6) Display on Console 1, the results of propagation using the **disp_tree** command. Verify the results by hand calculation.

The portions of the LLKS concerned with the propagation process is described below:

```
propagate :-
        find_executable_nodes (Node_list).
        find_propagatable_nodes (Node_list,P_node_list).
        mark_no_propagation (Node_list, P_node_list).
        propagate_one_step (P_node_list).
        change_terminal_status.
        change_suspended_nodes.
        find_executable_nodes (Node_list).
        propagate_loop (P_node_list).

propagate_loop (P_node_list) :-
        find_propagatable_nodes (Node_list, P_node_list).
        mark_no_propagation (Node_list, P_node_list).
        change_terminal_status.
        find_executable_nodes (Node_list).
        propagate_loop (P_node_list).
```

where,

find_executable_nodes:

Creates a list of those nodes that on which a propagation may be conducted,

find_propagatable_node:

Creates P_node_list, which is a list of nodes found in Node_list and whose value has changed since last propagation,

mark_no_propagation:

Mark those nodes that were not selected in Node_list as 'unchanged',

propagate_one_step:
Apply a single step data fusion on nodes in P_node_list,

mark_terminal_status:
Mark status of terminal nodes 'unchanged',

mark_suspended_nodes:
Mark suspended nodes 'unchaged'.

5.1.3 Data used in the experiment

The following knowledge (reproduced in English form) was used in the experiment:

If (the relative position of the sun to the spacecraft always changes)
And (heat dissipation around fuel tank is uneven)

Then (temperature within fuel tank cycles)

If (the sun reflection causes the spacecraft electrically charged)
Or (radiation from the on-board Super High Frequency equipment causes the charge)

Then (the spacecraft structure may be electrically charged).

These two pieces of knowledge represents two inference nodes, as shown in Figures 5.2a and 5.2b. The first knowledge constitutes a probablistic logical AND gate, while the latter a probablistic OR gate.

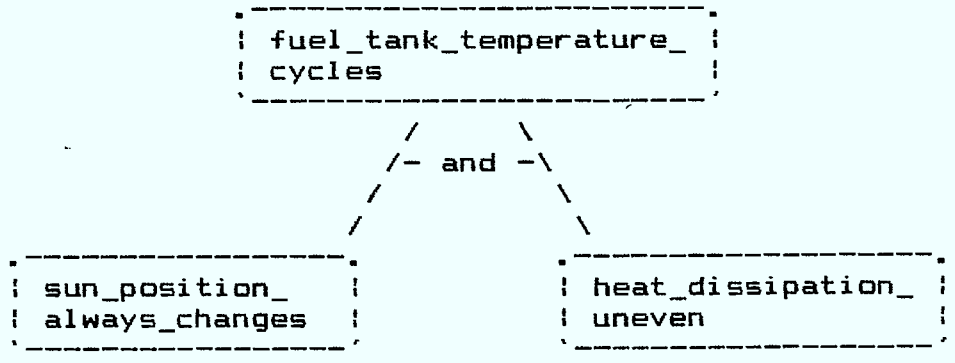

Figure 5.2a A probablistic logical AND gate

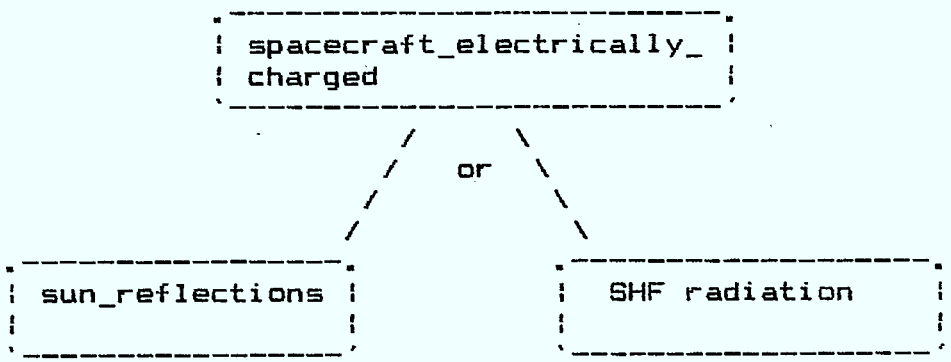

Figure 5.2b A probablistic logical OR gate

5.1.4 The results of the experiment

The following are the results of the data fusion experiments obtaqined by applying the **disp_tree** command to the COMKB after the propagation. There are four cases each for the AND and OR gates:

```
?disp_tree(charged_energy).
* charged_energy (or,2) D=.8 P=.97
   * sun_reflections (terminal) P=.86
     shf_radiation (terminal) P=.16
Yes
```

```
?disp_tree(charged_energy).
* charged_energy (or,2) D=.8 P=.68
     sun_reflections (terminal) P=.18
   * shf_radiation (terminal) P=.52
Yes
```

```
?disp_tree(charged_energy).
* charged_energy (or,2) D=.8 P=.98
   * sun_reflections (terminal) P=.62
   * shf_radiation (terminal) P=.73
Yes
```

```
?disp_tree(charged_energy).
* charged_energy (or,2) D=.8 P=.25
     sun_reflections (terminal) P=.07
     shf_radiation (terminal) P=.19
Yes
```

```
?disp_tree(fuel_tank_temp_cycles).
* fuel_tank_temp_cycles (and,2) D=1 P=.57
   * sun_position_always_changes (terminal) P=.86
   * heat_dissipation_uneven (terminal) P=.57
Yes
```

```
?disp_tree(fuel_tank_temp_cycles).
  fuel_tank_temp_cycles (and,2) D=1 P=.15
    * sun_position_always_changes (terminal) P=.64
      heat_dissipation_uneven (terminal) P=.15
Yes
```

```
?disp_tree(fuel_tank_temp_cycles).
  fuel_tank_temp_cycles (and,2) D=1 P=.17
     sun_position_always_changes (terminal) P=.17
   * heat_dissipation_uneven (terminal) P=.52
Yes
```

```
?disp_tree(fuel_tank_temp_cycles).
  fuel_tank_temp_cycles (and,2) D=1 P=.08
     sun_position_always_changes (terminal) P=.16
     heat_dissipation_uneven (terminal) P=.08
Yes
```

5.1.5 Discussion

The following are the justification of the results shown above obtained by comparing them with the results of hand calculation:

(1) The verification of the AND data fusion process

Hand calculations were performed using methods discussed in Section 4.3.4 on the AND fusion cases shown below, and the results were successfully compared with the outputs from the LLKS' fusion mechanism presented in 5.1.4 above:

Result:

```
* fuel_tank_temperature_cycles (and 2) D=1 P=.57
   * sun_position_always_changes  P=.86
   * heat_dissipation_uneven P=.57
```

Verification:

Since the maximum dependence (D=1) is assumed between the two supporting items of evidence,

$$Pe = MIN (Pa, Pb) = .57$$

where MIN is the minimum selection function.

Since P0 = 0, P1 = 1,

$$Ph = (P1-P0) * Pe + P0 = (1-0) * .57 + 0 = .57.$$

This value of Ph justifies the firing of the top assertion of the output identified by an asterisk attached in front of it, as the system-wide threshold for firing a node is set at .20 and .57 is greater than this value.

Result:

```
fuel_tank_temperature_cycles (and 2) D=1 P=.15
 * sun_position_always_changes  P=.64
   heat_dissipation_uneven  P=.15
```

Verification:

Using similar calculation as above, Pe = .15 is obtained. Since

$$Pe = (1-0) * .15 + 0 = .15,$$

the top assertion of this case does not fire, as shown by the absence of an asterisk (*) in front of it.

Result:

```
fuel_tank_temperature_cycles (and 2) D=1 P=.17
   sun_position_always_changes  P=.17
   heat_dissipation_uneven  P=.52
```

Verification:

Using the same set of equations, Pe = .17, Ph = .17 are obtained. Since the threshold is .20, the top assertion does not register itself (no asterisk).

Result:

```
fuel_tank_temperature_cycles (and 2) D=1 P=.08
   sun_position_always_changes  P=.16
   heat_dissipation_uneven  P=.08
```

Verification:

Calculated results for Pe and Ph both equals only to .08. Hence the the node fails to fire and the absence of an asterisk on the top assertion is justified.

(2) The verification of the OR data fusion process

The results of the four runs made on the OR propagation, shown in Section 5.1.4, are compared with the results of hand calculation performed on each of the cases.

Result:

```
* charged_energy (or 2) D=.-8 P=.97
   * sun_reflections P=.85
     shf_radiation P=.16
```

Verification:

The strength of assertion for a node of which supporting evidences are fully dependent on each other, C1 is:

$$C1 = Pa + Pb - Pa * Pb = .85 + .16 - (.85 * .16) = .88.$$

Similarly, for the minimum dependency case C3 is calculated as follows:

$$C3 = MIN\ (Pa + Pb,\ 1) = 1.$$

Therefore, applying interpolation on D, C1 and C3, the strength Pe of assertion for the top assertion of the example is,

$$Pe = |D| * C3 + (1 - |D|) * C1 = .8 * 1 + (1 - .8) * .88 = .8 + .176 = .976$$

This matches with the result and the firing of the node is justified as the threshold is still set at .2.

Result:

```
* charged_energy (or 2) D=-.8 P=.68
     sun_reflections P=.18
   * shf_radiation P=.52
```

Verification:

$$C1 = Pa + Pb - Pa*Pb = .18 + .52 - (.18 * .52) = .606$$
$$C3 = MIN\ (Pa + Pb, 1) = .7$$

$$Pe = |D| * C3 + (1 - |D|) * C1 = |-.8| * .7 + (1-.8) * .6 = .56 + (.2) * .6 = .68$$

$$Ph = (P1 - P0) * Pe + P0 = .68$$

Therefore, the node fires at strength = .68.

Result:

```
* charged_energy (or 2) D=-.8 P=.98
   * sun_reflections P=.62
   * shf_radiation P=.73
```

Verification:

$$C1 = Pa + Pb - Pa*Pb = .62 + .73 - (.62 * .73) = .9$$
$$C3 = MIN\ (Pa + Pb, 1) = 1$$

$$Pe = |-.8| * 1 + (1 - |-.8|) * .9 = .8 + .18 = .98$$

The node has fired as its strength of assertion is greater than the threshold.

Result:

```
charged_energy (or 2) D=-.8 P=.25
   sun_reflections P=.07
   shf_radiation P=.19
```

Verification:

$$C1 = Pa + Pb - Pa*Pb = .07 + .19 - (.07 * .19)$$
$$= .25$$
$$C3 = MIN\ (Pa + Pb,\ 1) = .26$$

$$Pe = |D| * C3 + (1 - |D|) * C1$$
$$= |-.8| * .26 + (1 - |-.8|) * .25$$
$$= .2 + .05$$
$$= .25$$

Hence the node has fired.

5.2 Testing of the HLKS Autonomy Control search mechanism

5.2.1 The objective of the test

The three control modes of the search mechanism of the HLKS Autonomy Control are tested. They are:

(1) depth-first search,
(2) breadth-first search,
(3) beam search.

5.2.2 The method of testing

Figure 5.4 shows the facility used for the tests. Steps in the test are:

(1) Initialize the COMDB,

(2) Enter knowledge for the experiment in the COMDB,

(3) Activate Autonomy Control's search mechanism by issuing an appropriate **search** command from Console 1. Search takes place on the test tree in the COMDB,

(4) Observe the results of search displayed on Console 1 by the HLKS as its search mechanism picks a new node.

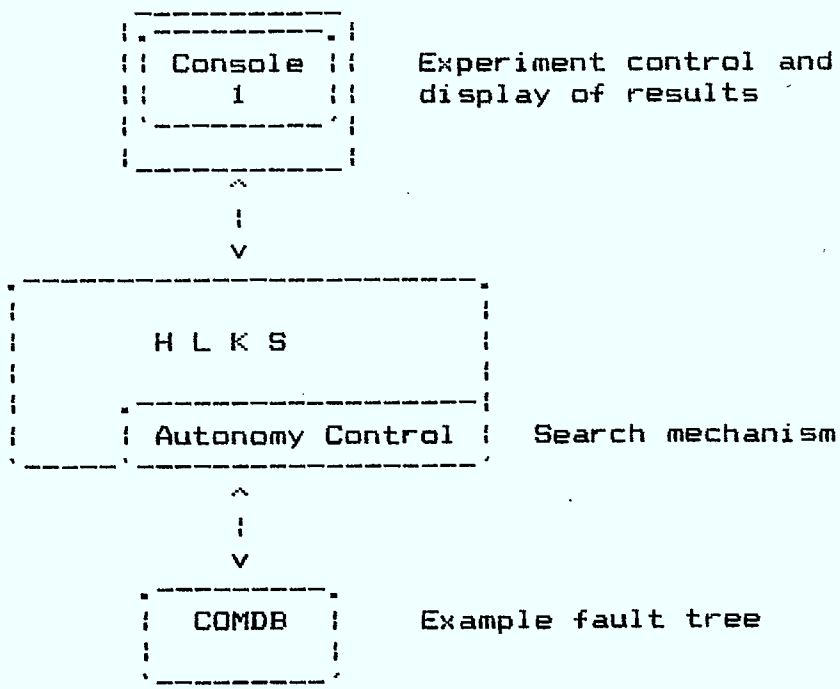

Figure 5.4 Facility for search mechanism test

The above procedure has been repeated for all three search strategies.

The HLKS Autonomy Control searches the inference network in the COMDB and visits nodes according to the search strategy in effect and issues messages identifying which node is visited. The order the messages are issued is studied to confirm the correctness of the search.

For the depth-first search, PROLOG's backtracking mechanism was used. The heart of the depth-first search algorithm used in this experiment is shown below:

```
hlks_search1 (Node) :-
        clock(N), write ("depth-first_search begins  time =  "),
        write (N), nl, hlks_depth_first (Node).
```

This predicate starts a clock (for measuring the speed of search) and initiates a depth-first search.

```
hlks_search1 (Node) :-
        write ("depth-first search finished  time = "), clock (N),
        write (N), nl.
```

Upon completion of the search, this predicate reports the current elapsed time.

```
hlks_depth_first (Node) :-
        node_structure (Node, _, evidence (E_list), _),
        hlks_test11_action (Node), hlks_decide_depth (Node, E_list).
```

This predicate collects evidences for current node and initiates the action to decide which node to search next.

```
hlks_decide_depth (Node, []) :-  fail.
hlks_decide_depth (Node, [E_head|E_tail]) :-
        hlks_depth (Node, E_head);
        hlks_decide_depth (Node, E_tail).
```

Continues search until an end of a branch is reached. The process backtracks then and tries an alternative at a level above.

```
hlks_test11_action (Node) :-
        write ("Searched:  ("), write (Node), write (")"),
        write_time, nl.
```

This predicate identifies the node being searched and prints out the elapsed time.

The algorithm for the breadth-first search is documented in the HLKS listings in Appendix A.1, predicate 'flexible_breadth_first' being its entry point.

The beam search begins, as shown in Figure 3.5, first as a breadth-first search and turns into a beam search half way down (otherwise, it will be a simple breadth-first search, if it were to begin from the top). The algorithm is contained in the one for the breadth-first search.

5.2.3 Data used in the experiment

The identical node structure (inference network) was used for all three search modes. It is the 102-node fault tree structure developed for the Attitude and Orbiting Control System of the CTS/Hermes satellite (See Appendix A.3).

5.2.4 The results of the experiment

The result of the depth-first search is presented in Figure 5.5 below. The output consists of 'Seached:' followed by the identifier of the node searched, followed by the elapsed system time from an arbitrary origin. According to the knowledge base shown in Appendix A.3, the order in which the nodes are searched is appropriate. Search time is measured in milliseconds. Average search time for a new node is 40 ms approx.

Figure 5.6a is a part of the output from the breadth-first search experiment. Again, the order of the search was found to be correct after comparing it with the knowledge base. The average per node search time was considerably longer than the depth-first search (140ms approx.). This was because the method could not take advantage of the built in backtracking mechanism of the PROLOG language system.

```
?search(spacecraft_lost,depth,tree).
depth_first_search begin   time = 8160
Searched: (spacecraft_lost)  time = 8180
Searched: (electronics_innert)  time = 8210
Searched: (heaters_ineffective)  time = 8240
Searched: (electrical_shutdown)  time = 8270
Searched: (uvs_trips)  time = 8300
Searched: (batteries_exhausted)  time = 8320
Searched: (power_loss_1)  time = 8350
Searched: (catalyst_bed_heater_on)  time = 8380
Searched: (recovery_procedure_begins)  time = 8420
Searched: (nesa_a_output_saturates)  time = 8450
Searched: (nesa_a_saturation_1)  time = 8490
Searched: (charged_energy)  time = 8530
Searched: (sun_reflections)  time = 8570
Searched: (shf_radiation)  time = 8610
Searched: (mirror_stuck)  time = 8660
Searched: (scan_mechanism_fails)  time = 8700
Searched: (thermal_distortion)  time = 8730
Searched: (sun_position_always_changes)  time = 8780
Searched: (anomalies_relate_to_sun_pos)  time = 8820
Searched: (unstable_pivot)  time = 8870
Searched: (mechanism_contamination)  time = 8910
Searched: (scan_motor_fails)  time = 8950
Searched: (motor fails)  time = 8990
```

Figure 5.5 Sample output from the depth-first search test

```
?search(spacecraft_lost,breadth,tree).
breadth_first_search begin  time = 141490
Searched: (spacecraft_lost)  time = 141790
Searched: (antenna_ineffective)  time = 142140
Searched: (electronics_innert)  time = 142160
Searched: (spacecraft_mechanically_frozen)  time = 14218
Searched: (telemetry_lost)  time = 142200
Searched: (electrical_shutdown)  time = 142560
Searched: (heaters_ineffective)  time = 142580
Searched: (spacecraft_tumbles)  time = 142600
Searched: (electrical_shutdown)  time = 143060
Searched: (large_cone_develops)  time = 143090
Searched: (pitch_changes_greatly)  time = 143100
Searched: (uvs_trips)  time = 143120
Searched: (wheel_stops)  time = 143150
Searched: (batteries_exhausted)  time = 143770
Searched: (charging_limited)  time = 143780
Searched: (electrical_shutdown)  time = 143800
Searched: (negative_pitch_develops)  time = 143830
Searched: (o4_firing_continues)  time = 143850
Searched: (uvs_trips)  time = 143870
Searched: (batteries_exhausted)  time = 145070
Searched: (charging_limited)  time = 145090
Searched: (high_rate_command_continues)  time = 145120
Searched: (o4 fires)  time = 145140
```

Figure 5.6a A portion of output from the breadth-first search test

Figure 5.6b is a portion of the output from the beam search experiment. In the diagram, the search strategy was switched from the breadth-first to the beam search on the seventh node (heaters_ineffective). The order of the search was found to be correct before and after the switch.

The average per node search time was the same as the breadth-first search time since it uses beam search.

```
?search(spacecraft_lost,beam,tree).
beam search begin  time = 61350
Searched:  (spacecraft_lost)  time = 61660
Searched:  (antenna_ineffective)  time = 62010
Searched:  (electronics_innert)  time = 62030
Searched:  (spacecraft_mechanically_frozen)  time = 62040
Searched:  (telemetry_lost)  time = 62070
Searched:  (electrical_shutdown)  time = 62410        breadth
Searched:  (heaters_ineffective)  time = 62430        beam
Searched:  (electrical_shutdown)  time = 62580
Searched:  (uvs_trips)  time = 62780
Searched:  (batteries_exhausted)  time = 63030
Searched:  (charging_limited)  time = 63050
Searched:  (power_loss_1)  time = 63440
Searched:  (power_loss_2)  time = 63460
Searched:  (tracking_partially_successful)  time = 63480
Searched:  (attitude_control_lost)  time = 63940
Searched:  (catalyst_bed_heater_on)  time = 63960
Searched:  (heavy_tracking_power)  time = 63980
Searched:  (solar_array_off_angle)  time = 64010
Searched:  (command_not_receivable)  time = 64490
Searched:  (continuous_tracking)  time = 64520
Searched:  (large_cone_develops)  time = 64550
Searched:  (pitch_changes_greatly)  time = 64580
Searched:  (recovery_procedure_begins)  time = 64610
Searched:  (attitude_control_lost)  time = 65500
Searched:  (negative_pitch_develops)  time = 65530
Searched:  (nesa_a_output_saturates)  time = 65560
Searched:  (o4_firing_continues)  time = 65570
Searched:  (receive_antenna_off_angle)  time = 65590
Searched:  (solar_array_off_angle)  time = 65620
Searched:  (command_not_receivable)  time = 66760
Searched:  (excessive_nesa_a_power_cycling)  time = 66790
Searched:  (high_rate_command_continues)  time = 66810
Searched:  (large_cone_develops)  time = 66840
Searched:  (nesa_a_saturation_1)  time = 66870
Searched:  (nesa_a_saturation_2)  time = 66890
```

Figure 5.6b Sample output from the beam search test

5.3 Automatic generation of a warning message

5.3.1 The objective of the experiment

The HLKS Autonomy Control subsystem has, as a part of its autonomous control function, an ability to detect a situation for which a warning must be issued. This experiment is to test and demonstrate that capability.

5.3.2 The method of the experiment

The experiment uses the test facility depicted in Figure 2.1. The flowchart shown in Figure 2.2 is also descriptive of the steps taken in this experiment. They are summarized below:

(1) Initialize the COMKB and the COMKB. Load the COMKB with the CTS/Hermes (AOCS) knowledge base,

(2) Initialize the HLKB and load it with the heuristic knowledge to search, detect, and report a node whose status warrants a warning,

(3) From Console 1, invoke the LLKS and start its event/event propagation process,

(4) From Console 1, invoke the POC main control. After the system level initialization, it will activate the EIU,

(5) Through Console 2 enter terminal events (sensor data) as required by the EIU. Select parameters and values so that a desired number of warnings are likely to arise. When the EIU is satisfied, the LLKS begins its event propagation process. The results of the propagation will be stored in the COMDB,

(6) Let the POC main module invoke the HLKS. The HLKS Autonomy Control will scan through the instantiated fault tree in the COMDB, applying knowledge in the HLKB to determine if a warning is warranted,

(7) Obtain a warning message issued by the HLKS Autonomy Control. The HLKS will then automatically execute the **probe** command under rule control (If the situation is bad enough to warrant a warning, then issue a probe on it to clarify the causal relationships between the anomaly and its supporting evidences) so that more information is generated on the node,

(8) Observe warning messages output by the Autonomy Control and check if all messages are justifiable, and all that have to be issued are there.

(9) Run the **disp_tree** command on the instantiated fault tree in the COMKB to observe nodes with a positive assertion greater than the threshold,

(10) Complete the experiment by issuing the **terminate** command to a system prompt.

```
?main2(spacecraft_lost).
Enter simulation magnifier (1 ... 1000000):
1.
telemetry_lost
f.
Enter { h., or l., }.
l.
o4_previously_fired
r.
diaphragm_leaks
r.
nitrogen_to_pressure
r.
impurities_in_tank
r.
fuel_in_tank_low
r.
heat_dissipation_uneven
r.
sun_reflections
r.
shf_radiation
r.
unstable_pivot
r.
mechanism_contamination
r.
motor_fails
r.
motor_overheats
r.
control_electronics_fails
r.
emi_to_electronics
r.
power_needs_to_be_cut_to_eliminate_output
r.
sun_position_always_changes
r.
anomalies_relate_to_sun_pos
r.
power_cut_to_eliminate_output
r.
nesa_a_output_must_be_cut_out
r.
```

Figure 5.7 The EIU input for the experiment

5.3.3 Data used in the experiment

The knowledge structure used for the previous experiment is used for this experiment. In addition, knowledge for the HLKS is added as shown in Appendix A.4.

Figure 5.7 is a record of the EIU interaction in which environmental parameters are entered for 20 terminal nodes in the experiment. It shows that all but one ('telemetry_lost' is fixed to 'low') parameters are generated under the control of a random number generator, as marked by an 'r.'.

5.3.4 Result of the experiment

Figure 5.8 is the output from the experiment. It shows that, after the EIU interaction, in which random number generation was specified for all terminal nodes, the POC main module invoked the LLKS. It completed the event propagation and handed over the control to the HLKS.

The HLKS picked up the first anomaly. A warning message was generated by the HLKS Autonomy Control, using the control knowledge in the HLKB. The ground controller then issued the **report** command on the node on which an anomaly was discovered. The result of the command is shown in the seven lines that follow. Normally, an operator in this situation would continue conversation with the system and further study the anomaly. In the experiment, the session was terminated by the **terminate** command. Notice that the sequence after the warning was under the control of the human operator. The POC system acted only in the capacity of an autonomous advisory system.

```
< LLKS starts >

< LLKS completes >

< HLKS starts >

WARNING:  ** (antenna_ineffective) **

Enter command:
report(antenna_ineffective).

antenna_ineffective (Command receive antenna is not functioning at all)
 is true with probability 1.

The state is determined by the rule:
[ antenna_ineffective (and spacecraft_tumbles electrical_shutdown) ]

spacecraft_tumbles (Spacecraft is tumbling) is true with probability 1.

electrical_shutdown (On-board electrical system is shut down)
is true with probability 1.

Enter {report., report([Id1, ..., Idn])., or end.}
end.

Enter command:
terminate.
< HLKS completes >
```

Figure 5.8 The result of the autonomous WARNING generation experiment

5.4 An autonomy control loop

5.4.1 The objective of the experiment

Operation of the HLKS can either be under the explicit control of the operator, or controlled by the meta-level knowledge stored in the HLKB. In addition, it can be controlled by domain level knowledge stored in the COMKB. This experiment is to test the cooperation between the knowledge-based control facilities of the HLKS and reasoning mechanism of the LLKS. By designing knowledge structures in these knowledge bases properly, one can construct a fault management control loop which will autonomously identify, analyse, report on, and correct an anomaly in the system.

5.4.2 The method of the experiment

The experiment is conducted using the entire POC experimental system described in Section 2.2. As shown in Figure 2.1, the two expert systems are linked with their knowledge bases, the COMKB and the COMDB acting as communication channels between them. The experiment roughly follows the flowchart of Figure 2.2. Below is a scenario in which pieces of knowledge are used to complete a control loop in order to solve an on-board anomaly:

An event propagation is conducted using the LLKS. It discovers an anomaly and reports it to the HLKS. An investigation by the HLKS follows, its search being controlled by the meta-level knowledge in the HLKB. The HLKS, also uses domain specific (object level) knowledge stored in the COMKB for each of the nodes it visits and recognizes that the anomaly reported by the LLKS is a serious one. It uses the general (meta-level) control knowledge in the HLKB and decides to take autonomous control of the node.

It first issues a warning message to ground control, identifying the fault. All actions taken and commands issued by the HLKS autonomously will be reported through the OIU, to the operator with a distinctive message identification. The HLKS then isues the **probe** command on the faulty node and reports the result to ground control. The Autonomy Control of the HLKS now consults the control knowledge in the COMKB, reasons on the control options, and decides on appropriate action. The node chosen for the experiment here autonomously recommends that the node itself be disconnected from the rest of the systems so as to contain the fault. The HLKS executes the recommended action and reports the fact to the ground.

The lower level expert system (LLKS) propagates input events through the inference network again, and this clarifies that the fault was eliminated from the system for the time being. This fact is reported to the ground.

The steps in the experiment are summarized below:

(1) Initialize the COMKB and load it with the CTS/Hermes (AOCS) knowledge base,

(2) Initialize the HLKB and load it with the heuristic knowledge to search, detect, and report, and take corrective action on a node whose status warrants these actions,

(3) From Console 1, invoke the LLKS and start its event/event propagation process,

(4) Through Console 2 enter terminal events (sensor data) as required by the propagation process. Select parameters and values so that a desired anomaly will arise. The result of the propagation will be stored in the COMDB,

(5) Run the **disp_tree** and other commands on the instantiated fault tree in the COMKB to study nodes with a positive assertion greater than the threshold (fault),

(6) Activate the HLKS. The HLKS Autonomy Control will scan through the instantiated fault tree in the COMDB, applying knowledge in the HLKB to determine if a corrective action is warranted. If so, the HLKS then proceeds to access knowledge for the troubled node in the COMKB so as to decide on the corrective action,

(5) Observe messages output by the Autonomy Control and study the sequence of actions which the HLKS Autonomy Control chose to execute,

(6) The LLKS automatically runs itself after the HLKS completes its operations. The new cycle of propagation must not report the same fault that was reported in a previous cycle.

5.4.5 Data used in the experiment

The same inference network as in the previous two experiments is used. This is enhanced by additional control knowledge in the COMKB and meta-knowledge in the HLKB, which is shown below. Some of the rules refer to knowledge stored at node level in the COMKB:

```
hlks_action (warning, Node) :-
        node_status (Node,_,_,t,P,_,_,_,_,w), number (P),
        decide_true (Node,_,P), get_action_list (Node, Action_list),
        check_warning (Action_list).
```

This rule determines if a warning message is warranted. It checks if the condition at the node is serious enough.

```
hlks_action (suspend, Node) :-
        node_control (Node, C,_,Entrust,_,_,_), ne(C, suspend),
        node_status (Node,_,_,t,P,_,_,_,_,_), number (P),
        decide_true (Node,_,P), get_action_list (Node, Action_list),
        check_warning (Action_list).
```

This rule decides if a suspension of a node is appropriate. Among other conditions it checks if the node in question is 'entrusted' to the HLKS for autonomous action. The following two rules are action rules and are used by a backward chaining inference engine local to the Autonomy Control.

```
take_hlks_action (Node, warning) :-
        write ("WARNING:  ** ("), write (Node), write(") **"), nl,
        write ("HLKS Autonomy Control: probe("), write (Node),
        write (")."), nl, probe (Node),
        change_node_status_for (Node,_,_,_,_,_,_,_,_,done).
```

This rule is used to issue to the operator a warning message on an anomaly.

```
take_hlks_action (Node, suspend) :-
        suspend (Node), write ("HLKS Autonomy Control: suspend("),
        write (Node), write (")."), nl, write("("), write (Node),
        write (")"), write(" is autonomously suspended by HLKS."), nl.
```

This rule is invoked to actually suspend a node.

```
take_hlks_action(Node, breadth).
```

The default search scheme in the HLKS is a breadth first search. This rule sets the default.

```
take_hlks_action (Node, beam) :-
        (ask_continue_beam (Node,R),/,equal (R,y),
           node_structure (Node, _, evidence (E_list),_)),
        flexible_breadth_first (E_list),/;/.
```

This meta-rule determines when to switch to a beam search, while executing other search strategy.

Other rules of the HLKB are shown in Appendix A.4 The knowledge stored in the COMKB is listed in Appendix A.3. Parameters are generated under the control of a random number generator, as marked by an 'r.'.

5.3.4 Result of the experiment

Figure 5.9 is the output from the experiment. It shows that, after the EIU interaction of Figure 5.7, in which random number generation was specified for all terminal nodes, the POC main module invoked the LLKS. Because the probability for some of the terminal events is very high (See Table 2.1), the LLKS must have found several faults in the system.

The HLKS operation that followed picked up the first and the most serious anomaly (Note that the HLKS searches basically top-down). A warning message was generated by the HLKS Autonomy Control. Using the control knowledge in the HLKB, it then issued the **probe** command. The result of the command is shown in the succeeding eight lines of the diagram.

The HLKS then used another set of knowledge in the HLKB and the COMKB and decided to suspend the node's operation. This corresponds to the situation, in which a faulty UVS (Under Voltage protection System) is removed from the system. The HLKS again reports its action.

The HLKS completes an autonomy management session with the system, and the LLKS starts a new propagation cycle. This time, the removal of the faulty unit resulted in the elimination of the key anomaly in the system, and the LLKS does not report a fault. The propagation cycles that follow proceed eventlessly.

```
< LLKS starts >

< LLKS completes >

< HLKS starts >

WARNING:  ** (uvs_trips) **

HLKS Autonomy Control:  probe(uvs_trips).


uvs_trips (Under voltage protection system is activated)
  is true with probability 1.

The state is determined by the rule:
[ uvs_trips (and batteries_exhausted charging_limited) ]

batteries_exhausted (On-board batteries are exhausted) is true with probat
1.

charging_limited (Solar array's ability to charge on-board batteries
  is now limited) is true with probability 1.

HLKS Autonomy Control:  suspend(uvs_trips).
(uvs_trips) is autonomously suspended by HLKS.
< HLKS completes >

< LLKS starts >

< LLKS completes >

< HLKS starts >

< HLKS completes >

< LLKS starts >

< LLKS completes >

< HLKS starts >


< HLKS completes >

< LLKS starts >

< LLKS completes >
```

Figure 5.9 The result of the autonomous control loop experiment

6. Conclusions and discussion

Through the development of and experiments using the SAMS POC experimental system, the following conclusions can be stated:

(1) The data fusion model proposed by Rauch is an important contribution to a real-time knowledge-based system paradigm. The model was taken and expanded to include n-input AND gates and three-input OR gates, and was used as the basis for the LLKS Inference Engine. This choice was justified because no other methods exist which take into account belief dependency among input signals, while offering a high degree of implementability. Bayesian theory, which is commonly used in expert systems (such as MYCIN and PROSPECTOR) as a source for their uncertainty handling mechanism, ignores the input dependency and thus cannot be adopted. There appears to be other similar approaches for handling dependent inputs, but none of them are as amenable for reasonable implementation.

These other models will have to be studied further in the future and a more elaborate data fusion model which better represents the real phenomena may have to be created by fully understanding the limitations of the current model. For example, shortcomings such as the lack of ways in the model to describe directional dependency among inputs (Input A as an event may be dependent on input B, and vice versa, but with a different degree of dependency) can be studied more carefully.

(2) A data/event driven expert system paradigm is more suited as a method for applying Knowledge Engineering to real-time systems than its goal-driven counterpart and its variations. There are attempts to interface a goal-driven expert system to real-time events [Anderson et al 84], but awkwardness is undeniable. In highly time critical systems, it will become impossible to complete any reasonable amount of heuristic search to prove goals and subgoals, let alone to conduct question/answer sessions with a human operator. However, these are the basic premises of the goal-driven systems.

(3) On the other hand, the existing methods for creating a goal-driven expert system appear to be adequate for the HLKS. It was felt that more user-friendly interfacing approaches, such as adoption of an elaborated icon-oriented graphic input/output facility, would be highly desirable, considering the peculiarities of the environment in which the autonomy management system will be used.

(4) An interesting analogy may be drawn between the characteristics of the two types of expert system architecture and knowledge processing conducted by conscious and subconscious minds of human beings. The data/event driven approach shares many aspects of subconscious input acceptance and response giving that the subconscious mind does. On the other hand, the conscious mind often and continuously, if not always and constantly, tries to resolve goals, one after another. A human being in his/her mode of operation as a knowledge processor is a real time system. It appears certain that most realtime intelligent systems will require a multi-tiered architecture for efficient processing.

(5) Therefore, the choice of combining these two different paradigms to construct a system which has to interface with a real-time environment, at the same time answering the needs of the human operator in enquiring the status, asking for control steps, and issuing instructions seems to be justified.

(6) Faulty situations in a system which must be identified and responded to faster than an operator can, should be left to autonomous systems. This policy has been adopted in most spacecraft subsystems already using conventional approaches. But the concept should be expanded into areas which require more elaborate judgements, which involve ambiguities of judgement criteria and of incoming information. An example of such judgement would be the careful handling necessary in managing the on-board charging system when a spacecraft emerges from an eclipse. An autonomous system which knows the causal relationships , as well as structural and functional knowledge about the spacecraft may be instructed to take precautionary measures.

(7) There are a number of potentially dangerous events which a spacecraft operator identifies and applies his learned techniques to avoid a catastrophe. Much of this knowledge may be coded in a knowledge based system and made autonomously available to execution vehicles and their control software. Compared to a system in which high level decisions are left solely to human operators, this would reduce accidents by omission, and thus contribute to an improved operational reliability. This benefit will be in addition to the ability of a system so-equipped to cope with high speed intelligent decision making needs well beyond the ability of human operator. Emergency situations in nuclear reactors and avionics systems are prime examples of this type of potential application. Increasingly more

delicate decisions should be left to autonomous systems as the performance of the decision-making mechanism improves with the advancing technology.

(8) Contrary to widespread myths in North America about its limitations, programming using PROLOG, a logic programming language, is an effective way, though may not be an ideal way, to construct expert systems. The fact that logic programming has been adopted as a foundation for several fifth generation computer projects throughout the world underwrites the satisfying experience the authors had.

REFERENCES

[Anderson, B.M. 84]
Anderson, B.M., et. al., "Intelligent Automation of Emergency Procedures in Advanced Fighter Aircraft". Proc. First Conference on Artificial Intelligence Applications, Denver, Colorado, December, 1984 (CAIA-84), sponsored by IEEE and AAAI.

[Bein 84]
Bein, Jonathan, "FIES: An Expert System for Isolating Faults of Spacecraft Hardware". Proc. 'Conference on Intelligent Systems and Machines', April 1984, Oakland University, Rochester, MI. (to be published).

[Blidberg et al. 83]
Blidberg, D.R., Westneat, A.S., Corell, R.W., "Expert Systems, A Tool for Autonomous Underwater Vehicles". In Proc. Trends & Applications 1983, IEEE Computer Society, May 1983, Washington D.C.

[Bullock, et al 83]
Bullock, B.D., et al, "Autonomous Vehicle Control: An Overview of the Hughes Project". Proc. IEEE Trends & Applications Conference, Washington D.C., May 1983, pp. 12-17.

[Cross 84]
Cross, Steve, "Expert Systems Architecture for Flight Domain Applications". Proc. Conference on Intelligent Systems and Machines, April, 1984, Rochester Michigan.

[Dicky & Toussaint 84]
Dicky, F. J., and Toussaint, Amy L., "ECESIS: An Application of Expert Systems to Manned Space Stations". Proc. The First Conference on Artificial Intelligence Applications, December, 1984, Denver, Colorado, December, 1984 (CAIA-84).

[Duda et al 81]
Duda, Richard, Hart, P.E., Nilsson, Nils j., "Subjective Bayesian method for rule-based inference systems". In Readings in Artificial Intelligence, Bonnie Webber, Nils Nilsson ed., Tioga Publishing Company, Palo Alto, California, pp. 192-199.

[Erman et al 80]
Erman, L.D., Heyeth-Roth, F., Lesser, V.R., and Reddy, D.R., "HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty". In Computing Survey, Vol. 12, No. 2, 1980.

[Hong et al 84]
Hong, S.J., et. al., "YES/MVS: A Continuous Real Time Expert System". Proc. National Conference on Artificial Intelligence, Austin Texas, August, 1984 (AAAI-84), pp. 130-136.

[Girad 84]
Girad, Jerry, L., "Fighter Pilot Aid by Expert Systems (Phase 1)". Proc. Conference on Intelligent Systems and Machines, April, 1984, Rochester Michigan.

[Gomi 84]
Gomi, T., "Functional Design of a Knowledge-based Spacecraft Autonomy Management System (SAMS)". Technical Report, Communications Research Centre, Department of Communications, Government of Canada, December, 1984.

[Harmon 83]
Harmon, S.Y., "Coordination between Control and Knowledge Based Systems for Autonomous Vehicle Guidance". Proc. IEEE Trends & Application Conference, Washington D.C., May 1983, pp.8-11.

[Harmon et al 84]
Harmon, S.Y., Gage, W.D., Aviles, W.A., Biancini,G.L., "Coordination of Intelligent Subsystems in Complex Robots". Proc. The First Conference on Artificial Intelligence Applications, Denver Colorado, December, 1984 (CAIA-84).

[Hewitt and Baker 77]
Hewitt, Carl and Baker, H., "Laws for Communicating Parallel Processes". Proc. 1977 IFIP Congress, IFIP 1977.

[Lesser and Corkill 81]
Lesser, Victor R., and Corkill, Daniel, "Functionally-accurate, Cooperative Distributed Systems". IEEE Trans. on Systems, Man and Cybernetics, SMC-11(1), January, 1981, pp. 81-96.

[McCarthy '84]
McCarthy, John., "What is Common Sense" In Proc. AAAI Presidential Address, August 1984, University of Texas at Austin.

[Milne 84]
Milne, R., "Maintenance Expert Systems for Analog Circuits"., Proc. Conference on Intelligent Systems and Machines, April, 1984, Rochester Michigan.

[Mitchell and Lemmar,84]
Mitchell, Brian T., and Lemmer, John F., "RADES: A Demonstration Expert System for Scientific, Space Station Experiments". Proc. 'Conference on Intelligent Systems and Machines 1984' Conference, April 1984, Oakland University, Rochester, MI. (to be published).

[Pisano and Jones 84]
Pisano, A.D., Jones, H.L., "An Expert System Approach to Adaptive Tactical Navigation". Proc. The First Conference on Artificial Intelligence Application, Denver Colorado, December, 1984 (CAIA-84).

[Rauch 84]
Rauch, Herbert E., "Probability Concepts For An Expert System Used For Data Fusion". The AI Magazine, Fall 1984, pp. 55-60.

[Sauers 84]
Sauers, Ron, "EMES: An Expert System for Spacecraft Energy Management". Proc. 'Conference on Intelligent Systems and Machines 1984', April 1984, Oakland University, Rochester, MI. To be published.

[Schundy 84]
Schundy, Robert, "Expert Systems in Tactical Aircraft". Proc. 'Conference on Intelligent Systems and Machines 1984', April 1984, Oakland University, Rochester MI. To be published.

[Shafer 76]
Shafer, Glenn, "A Mathematical Theory of Evidence". Princeton University Press, Princeton and London.

[Slagle 84]
Slagle, James, "BATTLE Expert System", Private note presented at Smart Systems Technology's AI course in July, 1984.

[Wagner 1983]
Wagner, Robert E.,"Expert Systems for Spacecraft Command and Control". Proc. 'Computers in Aerospace IV' Conference (AIAA-2372), Hartford, Conn., October 1983, pp. 216-223.

[Wagner 84]
Wagner, R., Private discussion on the progress of his system development effort, December, 1984.

[Winston 84]

Winston, Partick H., "Artificial Intelligence", Second Edition. Addison-Wesley Series in Computer Science, Addison-Wesley Publishing Comapny, Inc., pp. 191-197.

[Zadeh 76]
Zadeh, Lotfi, "Fuzzy Systems Theory: A Framework for the Analysis of Humanistic Systems". In Systems Methodology in Social Science Research: Recent Developments, Kluwer-Nijhoff Publishing, Boston, The Hague, London.

```
$ ty hlks.log
/*-------------------------------------------------------------------
/* High Level Knowledge based System
/*
report_to_hlks(fault) :-
     node_status(X,_,_,t,P,_,_,_,_,_), number(P),
       decide_true(X,ST,P) .
report_to_hlks(no_fault) .

propagate_t(STATE) :-
     check_status, propagation, report_to_hlks(STATE) .

llks(STATE) :-
     nl, write("( LLKS starts )"), nl, nl, propagate_t(STATE),
       write("( LLKS completes )"), nl .

entrust_llks :-
     node_control(X,_,entrust,_,_,_,_),
       change_node_status_for(X,_,_,_,_,_,_,_,entrusted,_), fail .
entrust_llks .

relieve_llks :-
     node_control(X,_,relieve,_,_,_,_),
       change_node_status_for(X,_,_,_,_,_,_,_,relieved,_), fail .
relieve_llks .

del_each_evidence :-
     each_evidence(X), fdelclause(each_evidence(X)), fail .
del_each_evidence .

del_ro :-
     route(X), fdelclause(route(X)), fail .
del_ro .
del_ro :-
     route(X), fdelclause(route(X)), fail .
del_ro .

del_c_product :-
     c_product(X), fdelclause(c_product(X)), fail .
del_c_product .

del_sum :-
     sum(X), fdelclause(sum(X)), fail .
del_sum .

set_status_resume([]) .
set_status_resume(NODE_LIST) :-
     get_level_evidence(NODE_LIST,EVI_LIST),
       (resume_breadth(NODE_LIST);
        set_status_resume(EVI_LIST)) .

resume_llks :-
     get_resume_node(NODE_LIST), set_status_resume(NODE_LIST) .

get_resume_node(NODE_LIST) :-
     bagof(N,get_resume(N),NODE_LIST) .
get_resume_node([]) .

get_resume(NODE) :-
     node_control(NODE,resume,_,_,_,_,_) .

resume_breadth([]) :-
     fail .
resume_breadth([N_H|N_T]) :-
     change_node_status_for(N_H,connected,active, . . . . . ), /.
```

```
        resume_breadth(N_T) .

adjust_suspend :-
      get_top_node(TOP_NODE_LIST),
        remove_extra_suspend(TOP_NODE_LIST) .

remove_extra_suspend([]) .
remove_extra_suspend(NODE_LIST) :-
      get_active_evidence(NODE_LIST,EVI_LIST),
        (remove_suspend(NODE_LIST);
         remove_extra_suspend(EVI_LIST)) .

remove_the_suspend(N_H) :-
      change_node_status_for(N_H,connected,active,_,_,_,_,_,_,_) .

get_active_evidence([],EVI_LIST) :-
      addclause(each_evidence(##)),
        setof(E,each_evidence(E),E_LIST), del_each_evidence,
        erase_first(E_LIST,EVI_LIST) .
get_active_evidence([N_HIN_T],EVI_LIST) :-
      node_structure(N_H,_,evidence(EVIDENCE_LIST),_),
        check_store_evi(EVIDENCE_LIST),
        get_active_evidence(N_T,EVI_LIST) .

check_store_evi([]) .
check_store_evi([E_HIE_T]) :-
      check_active(E_H,YES),
        (equal(YES,yes),addclause(each_evidence(E_H)),/;/),
        check_store_evi(E_T) .

check_active(E_H,no) :-
      node_control(E_H,suspend,_,_,_,_,_) .
check_active(E_H,no) :-
      node_status(E_H,suspended,_,_,_,_,_,_,_,_),
        check_plural_input(E_H,plural), check_loop_of(E_H,loop) .
check_active(E_H,yes) .

remove_suspend([]) :-
      fail .
remove_suspend([N_HIN_T]) :-
      remove_the_suspend(N_H), /, remove_suspend(N_T) .

check_plural_input(NODE,RESULT) :-
      bagof(N,plural(NODE,N),N_LIST), check_plural(N_LIST,RESULT) .

plural(NODE,N) :-
      node_structure(N,_,evidence(EVIDENCE_LIST),_),
        member(NODE,EVIDENCE_LIST) .

check_plural(N_LIST,plural) :-
      length(N_LIST,L), L)1 .
check_plural(N_LIST,not_plural) .

check_loop_of(NODE,LOOP) :-
      loop_bf_search([NODE],NODE,LOOP) .

check_loop(NODE) :-
      check_loop_of(NODE,LOOP), write("Node "), write(NODE),
        write(" is "), write_not(LOOP), write(" in a loop."), nl .

write_not(loop) .
write_not(not_loop) :-
      write("not") .

loop_bf_search([],NODE,LOOP) .
```

```
      get_level_evidence(NODE_LIST,EVI_LIST),
        detect_loop(NODE,EVI_LIST,LOOP),
          (equal(LOOP,not_loop),
             loop_bf_search(EVI_LIST,NODE,LOOP), /;
           /) .

detect_loop(NODE,EVI_LIST,loop) :-
      member(NODE,EVI_LIST) .
detect_loop(NODE,EVI_LIST,not_loop) .

generate_llks_command(COMMAND_LIST) :-
      bagof(X,llks_command(X),COMMAND_LIST) .
generate_llks_command([]) .

take_llks_action([]) .
take_llks_action([HIT]) :-
      llks_action(H), take_llks_action(T) .

suspend_llks :-
      set_llks_suspend .

set_status_suspend([]) .
set_status_suspend(NODE_LIST) :-
      get_level_evidence(NODE_LIST,EVI_LIST),
        (suspend_breadth(NODE_LIST);
         set_status_suspend(EVI_LIST)) .

suspend_breadth([]) :-
      fail .
suspend_breadth([N_HIN_T]) :-
      change_node_status_for(N_H,suspended,idele,_,_,_,_,_,_,_), /,
        suspend_breadth(N_T) .

find_suspended_nodes(NODE_LIST) :-
      bagof(X,find_suspended_node(X),NODE_LIST) .
find_suspended_nodes([]) .

set_llks_suspend :-
      find_suspended_nodes(NODE_LIST),
        set_status_suspend(NODE_LIST) .

find_suspended_node(NODE) :-
      node_control(NODE,suspend,_,_,_,_,_) .

del_node_str :-
      node_structure(_,_,_,_), fdelclause(node_structure(_,_,_,_)),
        fail .
del_node_str .

check_str(NODE) :-
      depth_co(NODE) .
check_str(NODE) :-
      write(NODE), write(" check successuful."), nl .

check(structure) :-
      find_top(NODE_LIST), check_structure(NODE_LIST) .
check(type) :-
      check_typ .
check(explanation) :-
      check_exp .
check("action") :-
      check_act .
check(control) :-
      check_cnt .
check(status) :-
      check_status .
```

```
check(X) :-
     write("### Argument("), write(X), write(") is undefined."),
       nl .

check :-
     check(structure), check(type), check(explanation),
       check("action"), check(control), check(status) .

check_structure([]) .
check_structure([H|T]) :-
     check_str(H), check_structure(T) .

depth_co(NODE) :-
     node_structure(NODE,_,evidence(E_LIST),_),
       decide_co(NODE,E_LIST) .
depth_co(NODE) :-
     node_structure(NODE,_,_,_), /, fail;
     write("### "), write("("), write(NODE), write(")"),
       write(" does not exist !"), nl .

decide_co(NODE,[]) :-
     fail .
decide_co(NODE,[E_H|E_T]) :-
     depth_serch_co(NODE,E_H);
     decide_co(NODE,E_T) .

depth_serch_co(NODE,E_H) :-
     cut(depth_co(E_H)) .

check_typ :-
     node_structure(NODE,_,_,_), check_type(NODE), fail .
check_typ .

check_sta(NODE) :-
     node_status(NODE,_,_,_,_,_,_,_,_,_,_), / .
check_sta(NODE) :-
     write("### "), write("("), write(NODE), write(")"),
       write(" status was not generated "), nl, / .

check_type(NODE) :-
     node_type(NODE,M,dependency(N),_,_,_,_,_,_,_), /, number(N),
       N)= -10, N=(10, integer(N) .
check_type(NODE) :-
     node_type(NODE,M,dependency(undefined),_,_,_,_,_,_,_), / .
check_type(NODE) :-
     write("### "), write("Type("), write(NODE), write(")"),
       write(" does not exist or incorrect !"), nl, / .

check_exp :-
     node_structure(NODE,_,_,_), check_explanation(NODE), fail .
check_exp .

check_explanation(NODE) :-
     node_explanation(NODE,g_type(X),v_depth(Y),_,_,_,description(
         [Z_H|Z_T])), /, integer(X), integer(Y), / .
check_explanation(NODE) :-
     write("### "), write("("), write(NODE), write(")"),
       write(" explanation does not exit or incorrect !"), nl, / .

check_explanation([],NODE) :-
     fail .
check_explanation([(AID,T,P,_,_,MSG)|A_T],NODE) :-
     equal(T,explanation), write_explanation(NODE,MSG);
     check_explanation(A_T,NODE) .
```

```
      node_structure(NODE,_,_,_), check_action(NODE), fail .
check_act .

check_action(NODE) :-
      node_action(NODE,_,_,_,_,action([H|T])), /,
        check_content_of([H|T]), / .
check_action(NODE) :-
      write("### "), write("Action("), write(NODE), write(")"),
        write(" does not exist or incorrect !"), nl, / .

check_content_of([]) .
check_content_of([(AID,T,P,_,_,MP)|T]) :-
      check_one_element(AID,T,P,_,_,MP), check_content_of(T) .

check_one_element(AID,T,P,_,_,MP) :-
      (equal(T,warm);equal(T,advice);equal(T,recommend);equal(T,
          report)), (number(P),integer(P),P>=0,P=<9), /,
        check_mp_connection(MP) .
check_one_element(AID,T,P,_,_,MP) :-
      write("### "), write("Argument of action("), write(NODE),
        write(")"), write(" incorrect!"), nl .

check_mp_connection(MP) :-
      node_message(MP,[_]) .
check_mp_connection(MP) :-
      write("### "), write("Message("), write(MP), write(")"),
        write(" does not exist or incorrect!"), nl .

check_cnt :-
      node_structure(NODE,_,_,_), check_control(NODE), fail .
check_cnt .

check_control(NODE) :-
      node_control(NODE,_,_,_,_,_,_), / .
check_control(NODE) :-
      write("### "), write("Control("), write(NODE),
        write(") does not exist or incorrect!"), nl, / .

display_reference(NODE) :-
      depth_re_search(NODE) .

display_reference.

disp_ref(NODE) :-
      display_reference(NODE) .

depth_re_search(NODE) :-
      node_structure(NODE,logic(LOGIC),evidence(EVIDENCE),_),
        addclause(re_route(NODE)),
        bagof(N,re_route(N),INDENTY_LIST),
        write_indenty(INDENTY_LIST), write_re_tree(NODE),
        check_re_terminal(NODE), decide_re_way(NODE,EVIDENCE) .

write_re_tree(NODE) :-
      node_structure(NODE,logic(LOGIC),_,REFERENCE), write(NODE),
        write(" "), write(REFERENCE), nl .

check_re_terminal(NODE) :-
      node_structure(NODE,logic("terminal"),evidence([]),_),
        fdelclause(re_route(NODE)), fail .
check_re_terminal(NODE) :-
      node_structure(NODE,logic(LOGIC),evidence(EVIDENCE),_),
        (equal(LOGIC,"not");
         equal(LOGIC,and);
         equal(LOGIC,or)) .
```

```
decide_re_way(NODE,[]) :-
     fdelclause(re_route(NODE)), fail .
decide_re_way(NODE,[EVIDENCE_H|EVIDENCE_T]) :-
     depth_search_on(NODE,EVIDENCE_H);
     decide_re_way(NODE,EVIDENCE_T) .

depth_search_on(NODE,EVIDENCE_H) :-
     cut(depth_re_search(EVIDENCE_H)) .

read_ans(EVI_LIST,TRUE_OR_FAIL) :-
     read(READ_DATA), syntax_check(READ_DATA),
       (check_tf(READ_DATA,TRUE_OR_FAIL);
        check_why(READ_DATA,EVI_LIST,TRUE_OR_FAIL);
        check_how(READ_DATA,EVI_LIST,TRUE_OR_FAIL)) .

check_why(why,EVI_LIST,TRUE_OR_FAIL) :-
     why_explanation(EVI_LIST,NODE_LIST,TRUE_OR_FAIL),
       read_ans(NODE_LIST,TRUE_OR_FAIL) .
check_why(why(USER_EVI_LIST),EVI_LIST,TRUE_OR_FAIL) :-
     why_explanation(USER_EVI_LIST,NODE_LIST,TRUE_OR_FAIL),
       read_ans(NODE_LIST,TRUE_OR_FAIL) .

check_tf(t,t) .
check_tf(f,f) .

why_explanation(EVI_LIST,NODE_LIST) :-
     make_node_list(EVI_LIST,NODE_LIST) .

make_node_list([],NODE_LIST) :-
     set_of(NODE,why_node(NODE),NODE_LIST), del_why_node .
make_node_list([EVI_H|EVI_T],_) :-
     all_node(EVI_H);
     make_node_list(EVI_T,_) .

all_node(EVI_H) :-
     node_structure(N,_,evidence(EVIDENCE_LIST),_),
       member(EVI_H,EVIDENCE_LIST), write_why_messi(EVI_H,N),
       addclause(why_node(N)), fail .

del_why_node :-
     why_node(X), fdelclause(why_node(X)), fail .
del_why_node .

write_why_messi(EVIDENCE,NODE) :-
     node_type(EVIDENCE,E_STATE,_,_,_,_,_,_,_,_),
       node_type(NODE,N_STATE,_,_,_,_,_,_,_,_), write(EVIDENCE),
       write(" is "), write(E_STATE), write(" cause of "),
       write(NODE), write(" is "), write(N_STATE),
       write(" as showed bellow."), nl, disp_exp(EVIDENCE) .

breadth_first_serch(NODE_LIST) :-
     breadth_first(NODE_LIST) .

breadth_first([]) .
breadth_first(NODE_LIST) :-
     get_level_evidence(NODE_LIST,EVI_LIST),
       (breadth(NODE_LIST);
        breadth_first(EVI_LIST)) .

get_level_evidence([],EVI_LIST) :-
     addclause(each_evidence(##)),
       setof(E,each_evidence(E),E_LIST), del_each_evidence,
       erase_first(E_LIST,EVI_LIST) .
get_level_evidence([N_H|N_T],EVI_LIST) :-
     node_structure(N_H,_,evidence(EVIDENCE_LIST),_),
       select_store_evi(EVIDENCE_LIST),
```

```
        get_level_evidence(N_T,EVI_LIST) .

select_store_evi([]) .
select_store_evi([E_H|E_T]) :-
     addclause(each_evidence(E_H)), select_store_evi(E_T) .

erase_first([E_H|E_T],E_T) .

breadth([]) :-
     fail .
breadth([N_H|N_T]) :-
     get_node_result(N_H,T_OR_F), /, take_action(N_H,T_OR_F), /,
       breadth(N_T) .

get_node_result(N_H,T_OR_F) :-
     node_status(N_H,_,_,T_OR_F,_,_,_,_,_,_) .
get_node_result(N_H,nt) .

take_action(N_H,t) :-
     get_action_list(N_H,ACTION_LIST),
       message_pro(ACTION_LIST,T_OR_F) .
take_action(N_H,T_OR_F) .

get_action_list(N_H,ACTION_LIST) :-
     node_action(N_H,_,_,_,_,action(ACTION_LIST)) .

message_pro([],T_OR_F) .
message_pro([(AID,T,P,_,_,MP)|A_T],T_OR_F) :-
     decide_control(T,P,MP,T_OR_F), message_pro(A_T,T_OR_F) .

decide_control(T,P,MP,T_OR_F) :-
     node_message(MP,MESSAGE_LIST), write_message(MESSAGE_LIST) .
decide_control(T,P,MP,T_OR_F) .

write_message([]) :-
     nl .
write_message([M_H|M_T]) :-
     write(M_H), write_message(M_T) .

bf(NODE) :-
     breadth_first_serch([NODE]) .

read_node_status(ID,C,A,S,P,RT,CH,SR,_,_) :-
     node_status(ID,C,A,S,P,RT,CH,SR,_,_) .

change_node_status_for(ID,A1,A2,A3,A4,A5,A6,A7,A8,A9) :-
     node_status(ID,C,A,S,P,RT,CH,SR,LE,WD), change_a1(C,A1,X1),
       change_a2(A,A2,X2), change_a3(S,A3,X3), change_a4(P,A4,X4),
       change_a5(RT,A5,X5), change_a6(CH,A6,X6),
       change_a7(SR,A7,X7), change_a8(LE,A8,X8),
       change_a9(WD,A9,X9),
       fdelclause(node_status(ID,C,A,S,P,RT,CH,SR,LE,WD)),
       addclause(node_status(ID,X1,X2,X3,X4,X5,X6,X7,X8,X9)), / .

change_node_type_for(ID,A1,dependency(A2),p1(A3),p0(A4),threshold(
          A5),error_rate(A6),_,_,_) :-
     node_type(ID,M,dependency(D),p1(P1),p0(P0),threshold(T),
         error_rate(E),_,_,_), change_type_a1(M,A1,X1),
       change_type_a2(D,A2,X2), change_type_a3(P1,A3,X3),
       change_type_a4(P0,A4,X4), change_type_a5(T,A5,X5),
       change_type_a6(E,A6,X6),
       fdelclause(node_type(ID,M,dependency(D),p1(P1),p0(P0),
          threshold(T),error_rate(E),_,_,_)),
       addclause(node_type(ID,X1,cependency(X2),p1(X3),p0(X4),
          threshold(X5),error_rate(X6),_,_,_)), / .
```

```
change_type_a1(M,A1,A1) :-
     string(A1) .
change_type_a1(M,A1,M) .

change_type_a2(D,A2,A2) :-
     number(A2);
     string(A2) .
change_type_a2(D,A2,D) .

change_type_a3(P1,A3,A3) :-
     number(A3) .
change_type_a3(P1,A3,P1) .

change_type_a4(P0,A4,A4) :-
     number(A4) .
change_type_a4(P0,A4,P0) .

change_type_a5(T,A5,A5) :-
     number(A5) .
change_type_a5(T,A5,T) .

change_type_a6(E,A6,A6) :-
     number(A6) .
change_type_a6(E,A6,E) .

change_a2(P,A2,A2) :-
     string(A2) .
change_a2(P,A2,P) .

change_a3(A,A3,A3) :-
     string(A3) .
change_a3(A,A3,A) .

change_a4(P,A4,A4) :-
     number(A4) .
change_a4(CH,A4,CH) .

change_a5(RT,A5,A5) :-
     string(A5) .
change_a5(RT,A5,RT) .

change_a6(CH,A6,A6) :-
     string(A6) .
change_a6(CH,A6,CH) .

change_a1(C,A1,A1) :-
     string(A1) .
change_a1(C,A1,C) .

change_a7(SR,A7,A7) :-
     string(A7) .
change_a7(SR,A7,SR) .

change_a8(LE,A8,A8) :-
     string(A8) .
change_a8(LE,A8,LE) .

change_a9(WD,A9,A9) :-
     string(A9) .
change_a9(WD,A9,WD) .

read_node_control(ID,C,_,_,_,_,_) :-
     node_control(ID,C,_,_,_,_,_) .

change_node_control_for(ID,A1,A2,A3,A4,A5,A6) :-
```

```
        change_control_a2(LE,A2,X2), change_control_a3(HE,A3,X3),
        change_control_a4(IM,A4,X4), change_control_a5(HL,A5,X5),
        fdelclause(node_control(ID,C,LE,HE,IM,HL,_)),
        addclause(node_control(ID,X1,X2,X3,X4,X5,_)) .
change_node_control_for(ID,A1,A2,A3,A4,A5,A6) .

initialize_control(ID,A1,A2,A3,A4,A5,A6) :-
     node_control(ID,C,LE,HE,IM,HL,_), change_control_a1(C,A1,X1),
     change_control_a2(LE,A2,X2), change_control_a3(HE,A3,X3),
     change_control_a4(IM,A4,X4), change_control_a5(HL,A5,X5),
     initialize_node_control(ID,X1,X2,X3,X4,X5,_), fail .
initialize_control(ID,A1,A2,A3,A4,A5,A6) .

change_control_a1(C,A1,A1) :-
     string(A1) .
change_control_a1(C,A1,C) .

change_control_a2(LE,A2,A2) :-
     string(A2) .
change_control_a2(LE,A2,LE) .

change_control_a3(HE,A3,A3) :-
     string(A3) .
change_control_a3(HE,A3,HE) .

change_control_a4(IM,A4,A4) :-
     string(A4) .
change_control_a4(IM,A4,IM) .

change_control_a5(HL,A5,A5) :-
     string(A5) .
change_control_a5(HL,A5,HL) .

disp_each :-
     h_evi(X), write("hypo --)"), write(X), fail .
disp_each .

node_top(TOP_NODE_LIST) :-
     get_top_node(TOP_NODE_LIST) .

get_top_node(TOP_NODES) :-
     setof(X,look_for_top(X),TOP_NODES) .

look_for_top(X) :-
     node_structure(X,_,_,_), check_node_x(X,F_OR_S) .

find_top(NODE_LIST) :-
     get_top_node(NODE_LIST) .

check_node_x(X,F_OR_S) :-
     look_up_node(X,F_OR_S), /, check_success(F_OR_S) .

look_up_node(X,fail) :-
     node_structure(_,_,evidence(EVIDENCE_LIST),_),
       (look_up(X,EVIDENCE_LIST,STATE);STATE is 1), equal(STATE,0),
       /, /;
     fail, / .
look_up_node(X,success) :-
     / .

look_up(X,[],_) :-
     fail .
look_up(X,[E_HIE_T],Y) :-
     equal(X,E_H), Y is 0, /;
     look_up(X,E_T,_) .
```

```
check_success(fail) :-
     fail .
check_success(success) .

ask_user_continue(HYPO_LIST,NEXT_LIST) :-
     write("Enter {report., report([Id1, ..., Idn])., or end.}"),
       nl, read(USER_RESPONSE), check_how_syntax(USER_RESPONSE),
       nl,
         (equal(USER_RESPONSE,end), /, /;
          decide_next_list(USER_RESPONSE,HYPO_LIST,NEXT_LIST)) .

check_how_syntax(READ_DATA) .

decide_next_list(report(NEXT_LIST),_,NEXT_LIST) .
decide_next_list(report,NEXT_LIST,NEXT_LIST) .

explain_how([]) :-
     fail .
explain_how([N_H|N_T]) :-
     explain_node_evidence(N_H), /, explain_how(N_T) .

explain_node_evidence(NODE) :-
     node_status(NODE,_,_,f,_,_,_,_,_,_), / .
explain_node_evidence(NODE) :-
     explain_how_node(NODE), explain_how_evidence(NODE) .

explain_how_node(NODE) :-
     node_status(NODE,_,_,t,PRO,_,_,_,_,_),
       node_type(NODE,STATE,dependency(D),_,_,_,_,_,_,_),
       write_how_node(NODE,STATE,D,PRO) .

write_how_node(NODE,STATE,D,PRO) :-
     node_explanation(NODE,_,_,_,_,_,description(D_LIST)), nl, nl,
       write(NODE), write(" ("), write_description(D_LIST),
       write(")"), write(" is true with probability "),
       write_probability_only(PRO), write("."), nl, nl,
       write("The state is determined by the rule:"), nl,
       write("["), disp_exp(NODE), write(" ]"), nl .
write_how_node(NODE,STATE,D,PRO) :-
     nl, write(NODE), write(" is true with probability "),
       write_probability_only(PRO), write("."), nl, nl,
       write(" The state is determined by the rule:"), nl,
       write("["), disp_exp(NODE), write("]"), nl .

how_pro(NODE_LIST) :-
     how_breadth_first(NODE_LIST) .

report(NODES) :-
     how_pro([NODES]) .

how :-
     node_top(TOP_NODE), how_pro(TOP_NODE) .

explain_how_evidence(NODE) :-
     node_structure(NODE,logic(LOGIC),evidence(E_LIST),_),
       explain_how_logic(LOGIC,E_LIST) .

explain_how_logic(and,E_LIST) :-
     how_and(E_LIST) .
explain_how_logic(or,E_LIST) :-
     how_or(E_LIST) .
explain_how_logic("not",E_LIST) :-
     how_not(E_LIST) .

how_not([]) .
```

```
      node_status(E_H,_,_,TF,PRO,_,_,_,_,_),
        node_type(E_H,STATE,dependency(D),_,_,_,_,_,_,_),
        write_how_not(E_H,STATE,D,PRO), how_not(E_T) .

write_how_not(E_H,STATE,D,PRO) :-
      node_explanation(E_H,_,_,_,_,_,description(D_LIST)), nl,
        write(E_H), write(" ("), write_description(D_LIST),
        write(") "), write("is fale with probability "),
        write_probability_only(PRO), write("."), nl, nl .
write_how_not(E_H,STATE,D,PRO) :-
      nl, write(E_H), write(" is fale with probability "),
        write_probability_only(PRO), write("."), nl, nl .

how_or([]) :-
      nl .
how_or([E_H|E_T]) :-
      get_or_data(E_H);
      how_or(E_T) .

get_or_data(E_H) :-
      node_status(E_H,_,_,TF,PRO,_,_,_,_,_),
        node_type(E_H,STATE,dependency(D),_,_,_,_,_,_,_),
        write_how_or(TF,E_H,STATE,D,PRO), /, fail .

how_and([]) :-
      nl .
how_and([E_H|E_T]) :-
      node_status(E_H,_,_,TF,PRO,_,_,_,_,_),
        node_type(E_H,STATE,dependency(D),_,_,_,_,_,_,_),
        write_how_and(E_H,STATE,D,PRO), how_and(E_T) .

write_how_or(t,E_H,STATE,D,PRO) :-
      node_explanation(E_H,_,_,_,_,_,description(D_LIST)), nl,
        write(E_H), write(" ("), write_description(D_LIST),
        write(") "), write("is true with probability "),
        write_probability_only(PRO), write("."), nl .
write_how_or(t,E_H,STATE,D,PRO) :-
      nl, write(E_H), write(" is true with probability "),
        write_probability_only(PRO), write("."), nl .
write_how_or(_,E_H,STATE,D,PRO) .

write_how_and(E_H,STATE,D,PRO) :-
      node_explanation(E_H,_,_,_,_,_,description(D_LIST)), nl,
        write(E_H), write(" ("), write_description(D_LIST),
        write(") "), write("is true with probability "),
        get_prob_range(PRO,W_TYPE), write_prob(PRO,W_TYPE),
        write("."), nl .
write_how_and(E_H,STATE,D,PRO) :-
      nl, write(E_H), write(" is true with probability "),
        write_probability(PRO), write("."), nl .

get_prob_range(0,zero) .
get_prob_range(100,hundred) .
get_prob_range(PRO,one_9) :-
      PRO)=1, 10)PRO .
get_prob_range(PRO,ten_99) .

write_probability_only(PROB) :-
      get_prob_range(PROB,W_TYPE), write_prob(PROB,W_TYPE) .

write_prob(PRO,hundred) :-
      write("1") .
write_prob(PRO,zero) :-
      write("0") .
write_prob(PRO,one_9) :-
```

```
write_prob(PRO,ten_99) :-
     write("."), write(PRO) .

how_breadth_first([]) .
how_breadth_first(NODE_LIST) :-
     get_follow_hypo(NODE_LIST,HYPO_LIST),
       (explain_how(NODE_LIST);
        ask_user_continue(HYPO_LIST,NEXT_LIST),
          how_breadth_first(NEXT_LIST)) .

get_follow_hypo([],HYPO_LIST) :-
     addclause(h_evi(##)), setof(H,h_evi(H),H_LIST), del_h_evi,
       erase_first(H_LIST,HYPO_LIST) .
get_follow_hypo([N_H|N_T],HYPO_LIST) :-
     node_structure(N_H,_,evidence(EVIDENCE_LIST),_),
       select_store_hypo(EVIDENCE_LIST),
       get_follow_hypo(N_T,HYPO_LIST) .

select_store_hypo([]) .
select_store_hypo([E_H|E_T]) :-
     node_structure(E_H,logic(LOGIC),_,_),
       (equal(LOGIC,"terminal"),/,/;addclause(h_evi(E_H))),
       select_store_hypo(E_T) .

del_h_evi :-
     h_evi(X), fdelclause(h_evi(X)), fail .
del_h_evi .

probe(NODE) :-
     probe_pro([NODE]) .

probe_pro(NODE_LIST) :-
     probe_breadth_first(NODE_LIST) .
probe_pro(NODE_LIST) .

probe_breadth_first([]) .
probe_breadth_first([H|T]) :-
     explain_how([H|T]), probe_breadth_first(T) .

suspend(NODES) :-
     suspend_nodes_hlks([NODES]) .

suspend_nodes_hlks([]) .
suspend_nodes_hlks([H|T]) :-
     set_suspend(H), suspend_nodes_hlks(T) .

set_suspend(NODE) :-
     node_control(NODE,suspend,_,_,_,_,_), write("  The "),
       write(NODE), write(" is already suspended."), nl .
set_suspend(NODE) :-
     node_control(NODE,_,_,_,_,_,_),
       change_node_control_for(NODE,suspend,_,_,_,_,_) .
set_suspend(NODE) :-
     write("### "), write(NODE),
       write(" does not exist or misspelled!"), nl .

activate(NODES) :-
     resume_nodes_hlks([NODES]) .

check_status :-
     node_structure(NODE,_,_,_), check_sta(NODE), fail .
check_status .

resume_nodes_hlks([]) .
resume_nodes_hlks([H|T]) :-
```

```
set_resume(NODE) :-
     node_control(NODE,resume,_,_,_,_,_), write("  The "),
       write(NODE), write(" is already resumed."), nl .
set_resume(NODE) :-
     node_control(NODE,C,_,_,_,_,_),
       change_node_control_for(NODE,resume,_,_,_,_,_) .

initialize_control :-
     node_control(NODE,_,_,_,_,_,_),
       initialize_node_control(NODE,no_command,relieve,relieve,r,_
           ,_), fail .
initialize_control .

initialize_node_control(NODE,C,LE,HE,IM,HL,_) :-
     fdelclause(node_control(NODE,_,_,_,_,_,_)),
       asserta(node_control(NODE,C,LE,HE,IM,HL,_)) .

initialize_status :-
     node_status(NODE,_,_,_,_,_,_,_,_,_),
       initialize_node_status(NODE,connected,active,_,_,_,
           unchanged,breadth,relieved,w), fail .
initialize_status .

initialize_llks :-
     initialize_status .

initialize_node_status(NODE,C,A,A1,A2,A3,CH,CS,LE,WD) :-
     fdelclause(node_status(NODE,_,_,_,_,_,_,_,_,_)),
       asserta(node_status(NODE,C,A,A1,A2,A3,CH,CS,LE,WD)) .

hlks(fault,NODE) :-
     nl, write("( HLKS starts )"), nl, nl, search(NODE),
       write("( HLKS completes )"), nl .
hlks(no_fault,NODE) .

remove_the_evidences(ORI_EVI,NEW_EVI) :-
     remove_the_evi(ORI_EVI,NEW_EVI) .
remove_the_evidences(ORI_EVI,[]) .

remove_the_evi(ORI_EVI,NEW_EVI) :-
     bagof(X,check_for_remove(X,ORI_EVI),NEW_EVI) .

check_for_remove(X,ORI_EVI) :-
     node_status(X,C,A,STATE,_,_,_,_,_,_), member(X,ORI_EVI),
       string(STATE), equal(C,connected) .

flexible_breadth_search(ORI_LIST) :-
     remove_by_control(ORI_LIST,NODE_LIST),
       flexible_breadth_first(NODE_LIST) .

search(ORI_LIST) :-
     nl, kill(each_evidence), flexible_breadth_search([ORI_LIST]) .

flexible_breadth_first([]) .
flexible_breadth_first(NODE_LIST) :-
     get_level_evidence(NODE_LIST,ORI_LIST),
       remove_by_control(ORI_LIST,EVI_LIST),
         (flexible_breadth(NODE_LIST,_);
          (ask_condition(NODE_LIST),ask_user(R),decide_conti(
              EVI_LIST,E_LIST,R);decide_conti(EVI_LIST,E_LIST,
              continuous)), flexible_breadth_first(E_LIST)) .

decide_conti(EVI_LIST,EVI_LIST,continuous) .
decide_conti(EVI_LIST,EVI_LIST,continue) .
```

```
decide_conti(EVI_LIST,EVI_LIST,X) :-
     X, /, fail .

ask_user(R) :-
     nl, write("Enter command:"), nl, read(R),
       (equal(R,continue);
        equal(R,terminate);
        equal(R,X)) .

flexible_breadth([],terminate) .
flexible_breadth([],_) :-
     fail .
flexible_breadth([N_H|N_T],_) :-
     action_process(N_H,FOUND), stop_node(FOUND,N_T,NN_T,R), /,
       flexible_breadth(NN_T,R) .

stop_node(_,N_T,N_T,continue) :-
     db_pause(level) .
stop_node(FOUND,N_T,N_T,continue) :-
     db_pause(branch) .
stop_node(FOUND,N_T,N_T,continue) :-
     db_pause(tree) .
stop_node(not_found,N_T,N_T,continure) :-
     db_pause(node) .
stop_node(found,N_T,NN_T,R) :-
     pause(node), back, ask_user(R), decide_conti_node(R,N_T,NN_T) .

decide_conti_node(continue,N_T,N_T) .
decide_conti_node(terminate,N_T,[]) .
decide_conti_node(X,N_T,N_T) :-
     X, /, fail .

back :-
     dummy;
     back .

dummy .

del :-
     del_each_evidence .

ask_conti_beam(N_H,R) :-
     get_description(N_H,D_LIST),
       write("Do you wish to continue a search from "), write("("),
       write_description(D_LIST), write(") ?"), nl,
       write("Enter y. or n. ."), nl, read(R) .

remove_by_control(X,X) .

ask_condition([]) :-
     fail .
ask_condition([N_H|N_T]) :-
     db_pause(level), get_action_list(N_H,ACTION_LIST),
       check_ask_cond(N_H,ACTION_LIST);
     ask_condition(N_T) .

check_ask_cond(N_H,[]) :-
     fail .
check_ask_cond(N_H,[(AID,T,P,_,_,_)|A_T]) :-
     (node_status(N_H,_,_,t,PRO,_,_,_,_,_),decide_true(N_H,_,PRO)),
       /, check_stop(T), /;
     check_ask_cond(N_H,A_T) .

strategy(NODE,SEARCH) :-
     check_input_search(SEARCH),
```

```
check_input_search(SEARCH) :-
     equal(SEARCH,breadth);
     equal(SEARCH,beam) .

check_stop(warning) .
check_stop(X) :-
     fail .

collect_action(N_H,A_LIST) :-
     bagof(X,hlks_action(X,N_H),A_LIST) .

action_process(NODE,found) :-
     collect_action(NODE,A_LIST), /, execute_action(NODE,A_LIST) .
action_process(NODE,not_found) .

execute_action(N_H,[]) .
execute_action(N_H,[H|T]) :-
     take_hlks_action(N_H,H), execute_action(N_H,T) .

check_warning([]) :-
     fail .
check_warning([(AID,T,P,_,_,MSG)|A_T]) :-
     equal(T,warning), /, /;
     check_warning(A_T) .

threshold(0) .

db_pause(tree) .

magnifier(1) .

ne(X1,X2) :-
     not(equal(X1,X2)) .

assess(NODE) :-
     nl, get_parent_node(NODE,P_LIST), report_pro(NODE,P_LIST);
     report_top(NODE) .

report_top(NODE) :-
     write_the_node(NODE), write(" is top node.") .

get_parent_node(NODE,P_LIST) :-
     bagof(X,find_parent_node(X,NODE),P_LIST) .

find_parent_node(X,NODE) :-
     node_structure(X,_,evidence(E_LIST),_), member(NODE,E_LIST) .

report_pro(NODE,[]) .
report_pro(NODE,[P_H|P_T]) :-
     report_exp(NODE,P_H), report_pro(NODE,P_T) .

report_exp(NODE,P_H) :-
     node_structure(P_H,logic(LOGIC),evidence(E_LIST),_),
       remove_the(NODE,E_LIST,NE_LIST), write_the_node(NODE),
       write_evi_of(LOGIC,NE_LIST), write_parent_node(P_H), nl .

write_evi_of(LOGIC,[]) .
write_evi_of(LOGIC,[NE_H|NE_T]) :-
     length([NE_H|NE_T],LENGTH),
       (equal(LENGTH,1),write(", "),write(LOGIC),write(" "),/;write
           (", ")), write_each_node(NE_H),
       write_evi_of(LOGIC,NE_T) .

write_each_node(NE_H) :-
```

```
        write_description(D_LIST), write(")") .

remove_the(NODE,E_LIST,NE_LIST) :-
      bagof(X,check_same_nodes(X,NODE,E_LIST),NE_LIST) .

check_same_nodes(X,NODE,E_LIST) :-
      node_structure(X,_,_,_), member(X,E_LIST),
        check_same_node(X,NODE) .

check_same_node(X,NODE) :-
      X==NODE, fail .
check_same_node(X,NODE) :-
      X=/=NODE .

write_the_node(NODE) :-
      write_each_node(NODE), write(". This") .

report_demo :-
      assess(large_cone_develops) .

write_parent_node(P_H) :-
      node_status(P_H,_,_,STATE,P,_,_,_,_,_), nl, nl,
        write("will jointly cause:"), nl, write_each_node(P_H),
          (equal(STATE,t), write(" with "), write(" probability "),
             write_probability_only(P), write("."), nl, /;
           write("."), nl) .

pause(ARG) :-
      check_pause(ARG), kill(db_pause), addclause(db_pause(ARG)) .
pause(ARG) :-
      write("### Argument("), write(ARG), write(")"),
        write(" is undefined!"), nl .

disp_db :-
      db_pause(X), write("pause("), write(X), write(")"), nl, fail .
disp_db .

kill(db_pause) :-
      addclause(db_pause(X)), del_db_pause .
kill(magnifier) :-
      addclause(magnifier(X)), del_magnifier .
kill(each_evidence) :-
      addclause(each_evidence(X)), del_each_evidence .
kill(route) :-
      addclause(route(X)), del_ro .
kill(sum) :-
      addclause(sum(X)), del_sum .
kill(c_product) :-
      addclause(c_product(X)), del_c_product .

del_magnifier :-
      magnifier(X), fdelclause(magnifier(X)), fail .
del_magnifier .

del_db_pause :-
      db_pause(X), fdelclause(db_pause(X)), fail .
del_db_pause .

entrust(NODE) :-
      change_node_control_for(NODE,_,entrust,_,_,_,_) .

relieve(NODE) :-
      change_node_control_for(NODE,_,relieve,_,_,_,_) .

check_pause(node) .
check_pause(ave) .
```

```
check_pause(branch) .
check_pause(terminate) .
check_pause(tree) .

recommend(NODE) :-
     get_action_list(NODE,ACTION_LIST),
       check_recommendation(ACTION_LIST,NODE) .
recommend(NODE) :-
     write("Recommendation not found in knowledge base.") .

check_recommendation([],NODE) :-
     fail .
check_recommendation([(AID,T,P,_,_,MSG)|A_T],NODE) :-
     equal(T,recommendation), write_recommendation(NODE,MSG);
     check_recommendation(A_T,NODE) .

write_recommendation(NODE,MSG) :-
     get_description(NODE,D_LIST), nl, write("RECOMMENDATION ("),
       write_description(D_LIST), write("):"), nl, nl,
       outspaces(7), write("   **"), outspaces(3),
       get_recommendation(NODE,MSG,R_LIST),
       write_description(R_LIST), write("   **"), nl .

get_recommendation(NODE,MSG,R_LIST) :-
     node_message(MSG,R_LIST) .

explain(NODE) :-
     get_action_list(NODE,ACTION_LIST),
       check_explanation(ACTION_LIST,NODE) .
explain(NODE) :-
     write("### Explanation of recommendation not found!"), nl .

write_explanation(NODE,MSG) :-
     get_description(NODE,D_LIST),
       get_explanation(NODE,MSG,EX_LIST),
       node_status(NODE,_,_,T,P,_,_,_,_,_), nl,
       write("In the state of ("), write_description(D_LIST),
       write(") with probability "), write_probability_only(P),
       write(", "), write_description(EX_LIST), nl .

get_explanation(NODE,MSG,EX_LIST) :-
     node_message(MSG,EX_LIST) .

hlks_search1(NODE) :-
     clock(N), write("depth_first_search begin   time = "),
       write(N), nl, hlks_depth_first(NODE) .
hlks_search1(NODE) :-
     write("depth_first_search finished   time = "), clock(N),
       write(N), nl .

hlks_depth_first(NODE) :-
     node_structure(NODE,_,evidence(E_LIST),_),
       hlks_test1_action(NODE), hlks_decide_depth(NODE,E_LIST) .

hlks_decide_depth(NODE,[]) :-
     fail .
hlks_decide_depth(NODE,[E_H|E_T]) :-
     hlks_depth(NODE,E_H);
     hlks_decide_depth(NODE,E_T) .

hlks_test1_action(NODE) :-
     hlks_test_action(warning,NODE) .

hlks_test_action(warning,NODE) :-
     node_status(NODE,_,_,t,P,_,_,_,_,_), decide_true(NODE,_,P),
```

```
        decide_test_action(NODE,ACTION_LIST) .
hlks_test_action(_,NODE) .
```

```
test2 :-
     hlks_search2(spacecraft_lost) .

write_time :-
     clock(N), write("  time = "), write(N) .
```

```
action_test_hlks(N_H,warning,P,MSG) :-
     get_description(N_H,D_LIST), write("WARNING:  ** ("),
       write_description(D_LIST), write(") **"), write_time, nl .
action_test_hlks(N_H,_,P,MSG) .

decide_test_action(N_H,[]) .
decide_test_action(N_H,[(AID,T,P,_,_,MSG)|A_T]) :-
     action_test_hlks(N_H,T,P,MSG), decide_action(N_H,A_T) .

hlks_search2(NODE) :-
     clock(N), write("breadth_first_search begin  time = "),
       write(N), nl, hlks_breadth_first([NODE]),
       write("breadth_first_search finished  time = "), clock(T),
       write(T), nl .

hlks_breadth_first([]) .
hlks_breadth_first(NODE_LIST) :-
     get_level_evidence(NODE_LIST,EVI_LIST),
       (hlks_breadth(NODE_LIST);
        hlks_breadth_first(EVI_LIST)) .

hlks_breadth([]) :-
     fail .
hlks_breadth([N_H|N_T]) :-
     hlks_test2_action(N_H), /, hlks_breadth(N_T) .

hlks_test2_action(N_H) :-
     hlks_test_action(warning,N_H) .

hlks_depth(NODE,E_H) :-
     cut(hlks_depth_first(E_H)) .

test1 :-
     hlks_search1(spacecraft_lost) .

main2(NODE) :-
     initialize_suspend_system, execute_command_loop(NODE) .

execute_command_loop(NODE) :-
     initialize_status_part, generate_llks_command(COMMAND_LIST),
       take_llks_action(COMMAND_LIST), set_terminal_data,
       llks(STATE), hlks(STATE,NODE), execute_command_loop(NODE) .

initialize_status_part :-
     node_status(NODE,C,A,_,_,_,_,_,S,_),
       initialize_node_status(NODE,C,A,_,_,_,unchanged,breadth,S,w
           ), fail .
initialize_status_part .

initialise :-
     initialize .

initialize :-
     kill(sum), kill(c_product), initialize_status,
       initialize_control .
```

```
      set_random(5749317), set_magnifier, set_mode .

set_magnifier :-
      back, write("Enter simulation magnifier (1 ... 1000000):"),
        nl, read(R), check_magnifier_range(R), kill(magnifier),
        addclause(magnifier(R)) .

set_mode :-
      set_input_mode .

check_magnifier_range(R) :-
      number(R), R)=1, R(1000000 .
check_magnifier_range(R) :-
      write("### Magnifier ("), write(R), write(")"),
        write("is not number or not in the range!"), nl, nl, /,
        fail .

set_input_mode :-
      node_structure(X,logic("terminal"),_,_),
        node_status(X,connected,_,_,_,_,_,_,_,_), ask_input_mode(X),
        fail .
set_input_mode .

ask_input_mode(NODE) :-
      ask_check_mode(NODE,R,HL),
        change_node_control_for(NODE,_,_,_,R,HL,_), / .

check_input_mode(m) .
check_input_mode(f) .
check_input_mode(r) .
check_input_mode(X) :-
      write("### "), write(X), write(" undefined!"), nl,
        write("Enter { m., f., or r., }."), nl, /, fail .

ask_check_mode(NODE,R,HL) :-
      write(NODE), nl, back, read(R), check_input_mode(R), /,
        ask_hl(R,HL) .

initialize_system :-
      initialize, check_status, check_cnt, environment .

initialize_suspend_system :-
      kill(sum), kill(c_product), initialize_status,
        initialize_control(_,connect,_,_,r,_,_), check_status,
        check_cnt, environment .

ask_hl(r,_) .
ask_hl(m,_) .
ask_hl(f,HL) :-
      write("Enter { h., or l., }."), nl, back, read(HL),
        check_hl(HL) .

check_hl(h) .
check_hl(l) .
check_hl(X) :-
      write("### "), write(X), write(" undefined!"), nl,
        write("Enter { h., or l., }."), nl, /, fail .

set_terminal_data :-
      node_structure(X,logic("terminal"),_,_), set_status_data(X),
        fail .
set_terminal_data .

set_status_data(NODE) :-
      node_status(NODE,connected,_,_,_,_,_,_,_,_),
        node_control(NODE,_,_,_,M,H,_) .
```

```
        node_type(NODE,_,_,_,_,_,error_rate(ER),_,_,_),
        set_data(NODE,ER,IM,HL), / .

set_data(NODE,ER,m,HL) .
set_data(NODE,ER,f,HL) :-
      get_fixed_data(HL,D),
        change_node_status_for(NODE,_,_,t,D,t,changed,_,_,_) .
set_data(NODE,ER,r,HL) :-
      get_random(RN), check_if_set(RN,ER,D),
        change_node_status_for(NODE,_,_,t,D,t,changed,_,_,_) .

t0 :-
      set_random(5749317), t1 .

check_if_set(RN,ER,D) :-
      magnify_rate(RN,MRN), ER)=MRN, get_fixed_data(h,D) .
check_if_set(RN,ER,D) :-
      magnify_rate(RN,MRN), ER(MRN, get_fixed_data(1,D) .

magnify_rate(RN,MRN) :-
      magnifier(M), div(RN,M,MRN) .

get_error_rate(ER,ER1) :-
      magnifier(M), calculate_error_rate(M,ER,ER1) .

get_fixed_data(h,100) .
get_fixed_data(1,0) .

get_random(RN) :-
      random(R,0,999999), rem(R,10000,RR), times(RR,100,RN) .

t1 :-
      get_random(RN), write_random(RN), outspaces(2), t1 .

write_random(RN) :-
      RN(=9, write("00000"), write(RN) .
write_random(RN) :-
      RN)9, RN(=99, write("0000"), write(RN) .
write_random(RN) :-
      RN)99, RN(=999, write("000"), write(RN) .
write_random(RN) :-
      RN)999, RN(=9999, write("00"), write(RN) .
write_random(RN) :-
      RN)9999, RN(=99999, write("0"), write(RN) .
write_random(RN) :-
      RN)99999, RN(=999999, write(RN) .

execute(NODE) :-
      set_terminal_data, llks(STATE), hlks(STATE,NODE) .

main0(NODE) :-
      initialize_system, execute(NODE) .

main1(NODE) :-
      initialize_system, execute_loop(NODE) .

execute_loop(NODE) :-
      initialize_status, set_terminal_data, llks(STATE),
        hlks(STATE,NODE), execute_loop(NODE) .

error_rate :-
      node_structure(X,logic("terminal"),_,_), set_error_rate(X),
        fail .
error_rate .

--- error_rate(X) ---
```

```
      write(X), nl, back, read(R), check_error_rate(R),
        change_node_type_for(X,_,_,_,_,_,error_rate(R),_,_,_), / .

check_error_rate(R) :-
      number(R), R>=0, R<=999999 .
check_error_rate(R) :-
      number(R),
        write("### Error-rate must be in the range(0 ... 999999)!"),
        nl, write("Enter(0. ... 999999.):"), nl, /, fail .
check_error_rate(R) :-
      string(R), write("### Error-rate must be number!"), nl,
        write("Enter(0. ... 999999.):"), nl, /, fail .
check_error_rate(R) .
```

```
$ ty llks.log
module unnamed_module.
```

Appendix A.2 LLKS Source Listings

```
/*$eject*/
body.
/*                                        */
/* Low Level Knowledge based System */
/*                                        */

check_evidence(and,[]) .
check_evidence(and,[HEAD|TAIL]) :-
     node_structure(HEAD,logic(LOGIC),evidence(EVIDENCE_LIST),_),
       /,
       (equal("terminal",LOGIC);node_status(HEAD,C,A,_,_,_,_,_,_
           ),equal(C,suspended);node_status(HEAD,_,_,STATE,_,_,_,
           ,_,_),string(STATE)), check_evidence(and,TAIL) .
check_evidence(or,[]) :-
     fail .
check_evidence(or,[HEAD|TAIL]) :-
     node_structure(HEAD,logic(LOGIC),evidence(EVIDENCE_LIST),_),
       ((equal("terminal",LOGIC);node_status(HEAD,C,A,_,_,_,_,_,_
            ,_),equal(C,suspended);node_status(HEAD,_,_,STATE,_,_
            ,_,_,_,_),string(STATE));
        check_evidence(or,TAIL)) .
check_evidence("not",[HEAD|TAIL]) :-
     node_structure(HEAD,logic(LOGIC),evidence(EVIDENCE_LIST),_),
       equal("terminal",LOGIC);
     node_status(HEAD,_,_,STATE,_,_,_,_,_,_), string(STATE);
     node_status(HEAD,C,A,_,_,_,_,_,_,_), equal(C,suspended) .
check_evidence("terminal",[]) :-
     fail .

member(ITEM,[ITEM|TAIL]) .
member(ITEM,[HEAD|TAIL]) :-
     ITEM=/=HEAD, member(ITEM,TAIL) .

check_evidence_change([]) :-
     /, fail .
check_evidence_change([E_H|E_T]) :-
     (node_status(E_H,_,_,STATE,_,_,X,_,_,_),string(STATE)), /,
       (equal(X,changed),/,/;check_evidence_change(E_T)), /;
     node_structure(E_H,logic(LOGIC),_,_),
       (equal(LOGIC,"terminal"), /, /;
        check_evidence_change(E_T)) .

decide_propagatable_node(X,NODE_LIST) :-
     node_structure(X,_,evidence(E_LIST),_), member(X,NODE_LIST),
       check_evidence_change(E_LIST) .

find_propagatable_node(NODE_LIST,P_NODE_LIST) :-
     bagof(X,decide_propagative_node(X,NODE_LIST),P_NODE_LIST) .
find_propagatable_node(NODE_LIST,[]) .

decide_executable_node(NAME) :-
     node_structure(NAME,logic(LOGIC),evidence(EVIDENCE_LIST),_),
       node_status(NAME,C,A,_,_,_,_,_,_,_), equal(C,connected),
       check_evidence(LOGIC,EVIDENCE_LIST) .

remove_suspended_evidence(E_LIST,EVIDENCE_LIST) :-
     bagof(X,check_suspended_arg(X,E_LIST),EVIDENCE_LIST) .

check_suspended_arg(X,E_LIST) :-
     node_status(X,C,A,_,_,_,_,_,_,_), member(X,E_LIST),
       equal(C,connected) .
```

```
find_executable_node(NODE_LIST) :-
     setof(NAME,decide_executive_node(NAME),NODE_LIST) .

make_list([],LIST,LIST) .
make_list([H|T],X_LIST,R_LIST) :-
     node_structure(H,_,_,_), quick_bagof(H,X_LIST),
       shift_one(X_LIST,S_LIST), make_list(T,S_LIST,R_LIST),
       insert_list(H,R_LIST);
     make_list(T,X_LIST,R_LIST) .

insert_list(X,[X]) .
insert_list(X,[X|T]) .

shift_one([O_H|O_T],[Y,O_H|O_T]) .
shift_one(L1,L2) :-
     reverse_list(L1,L2) .

quick_bagof(X,[X]) :-
     decide_executable_node(X) .
quick_bagof(X,[X|T]) :-
     decide_executable_node(X) .

make_structure_list(LIST) :-
     setof(X,make_str_list(X),LIST), write(LIST), nl,
       make_list(LIST,L,R), write(R), nl, nl .

test :-
     make_structure_list(LIST) .

make_str_list(NODE) :-
     node_structure(NODE,_,_,_) .

propagate :-
     check_status, display_input, propagation, reporti .

propagation :-
     find_executable_node(NODE_LIST),
       find_propagatable_node(NODE_LIST,P_NODE_LIST),
       change_non_propagatable(NODE_LIST,P_NODE_LIST),
       propagate_once(P_NODE_LIST), change_terminal_status,
       change_all_suspended_state,
       find_executable_node(NEXT_NODE_LIST),
         (equal(NODE_LIST,NEXT_NODE_LIST), /, /;
          propagate_loop(NEXT_NODE_LIST)) .

propagate_loop(NODE_LIST) :-
     find_propagatable_node(NODE_LIST,P_NODE_LIST),
       change_non_propagatable(NODE_LIST,P_NODE_LIST),
       propagate_once(P_NODE_LIST), change_terminal_status,
       find_executable_node(NEXT_NODE_LIST),
         (equal(NODE_LIST,NEXT_NODE_LIST), /, /;
          propagate_loop(NEXT_NODE_LIST)) .

write_for_test([]) :-
     nl, nl .
write_for_test([H|T]) :-
     write("  "), write(H), write_for_test(T) .

change_non_propagatable([],P_NODE_LIST) .
change_non_propagatable([N_H|N_T],P_NODE_LIST) :-
     member(N_H,P_NODE_LIST),
       change_non_propagatable(N_T,P_NODE_LIST), /;
     fdelclause(node_status(N_H,A,B,C,D,E,F,G,H,I)),
       addclause(node_status(N_H,A,B,C,D,E,unchanged,G,H,I)),
       change_non_propagatable(N_T,P_NODE_LIST) .
```

```
change_terminal_status :-
     node_status(X,A,B,C,D,E,changed,_,_,_), change_status_c(X),
       fail .
change_terminal_status .

change_all_suspended_state :-
     node_structure(X,logic(LOGIC),evidence(E_LIST),_),
       find_suspended_by_evidence(X,LOGIC,E_LIST),
       change_node_status_for(X,_,_,_,_,_,unchanged,_,_,_), fail .
change_all_suspended_state .

find_suspended_by_evidence(X,LOGIC,E_LIST) :-
     ne(LOGIC,"terminal"), remove_the_evidences(E_LIST,NE_LIST), /,
       length(NE_LIST,0), / .

change_status_c(NODE) :-
     node_structure(NODE,logic("terminal"),_,_),
       fdelclause(node_status(NODE,A,B,C,D,E,F,G,H,I)),
       addclause(node_status(NODE,A,B,C,D,E,unchanged,G,H,I)) .
change_status_c(NODE) .

delete_db :-
     node_status(NODE,_,_,_,_,_,_,_,_,_),
       node_structure(NODE,_,_,_),
       fdelclause(node_status(NODE,_,_,_,_,_,_,_,_,_)), fail .
delete_db .

report1 :-
     node_structure(NODE,logic(LOGIC),_,_),
       check_and_write(NODE,LOGIC) .
report1 .

check_and_write(NODE,LOGIC) :-
     (equal(and,LOGIC);equal(or,LOGIC);equal("not",LOGIC)),
       node_status(NODE,_,_,t,POST,_,_,_,_,_),
       node_type(NODE,STATE,_,_,_,_,_,_,_,_),
       write_state(NODE,STATE,POST), /, fail .

display_input :-
     node_status(NODE,_,_,STATE,P,_,_,_,_,_),
       write_state(NODE,STATE,P), fail .
display_input .

delete_kb .

propagate_once([]) .
propagate_once([NODE_H|NODE_T]) :-
     node_structure(NODE_H,logic(LOGIC),evidence(EVIDENCE_LIST),_),
       propagate_one_level(LOGIC,EVIDENCE_LIST,NODE_H),
       propagate_once(NODE_T) .

get_state(and,[]) .
get_state(and,[EVIDENCE_H|EVIDENCE_T]) :-
     (node_status(EVIDENCE_H,_,_,TRUE_OR_FAIL,PRIOR,LS,LN,_,_,_),
         string(TRUE_OR_FAIL);node_structure(EVIDENCE_H,logic(
         "terminal"),evidence([]),_),node_status(EVIDENCE_H,
         suspended,_,_,_,_,_,_,_,_);node_structure(EVIDENCE_H,logic
         ("terminal"),evidence([]),_),query_the_user(EVIDENCE_H,
         USER_STATE,CERTAINTY),make_m_node_data(EVIDENCE_H,
         USER_STATE,CERTAINTY)), get_state(and,EVIDENCE_T) .
get_state(or,[]) .
get_state("not",[EVIDENCE]) :-
     node_status(EVIDENCE,_,_,TRUE_OR_FAIL,PRIOR,LS,LN,_,_,_),
       string(TRUE_OR_FAIL);
     node_structure(EVIDENCE,logic("terminal"),evidence([]),_),
       node_status(EVIDENCE,suspended
```

```
      node_structure(EVIDENCE,logic("terminal"),evidence([]),_),
        query_the_user(EVIDENCE,USER_STATE,CERTAINTY),
        make_m_node_data(EVIDENCE,USER_STATE,CERTAINTY);
      node_structure(EVIDENCE,logic(_),evidence(_),_) .
get_state(or,[EVIDENCE_H|EVIDENCE_T]) :-
      (node_status(EVIDENCE_H,_,_,TRUE_OR_FAIL,PRIOR,LS,LN,_,_,_),
          string(TRUE_OR_FAIL);node_structure(EVIDENCE_H,logic(
          "terminal"),evidence([]),_),node_status(EVIDENCE_H,
          suspended,_,_,_,_,_,_,_,_);node_structure(EVIDENCE_H,logic_
          ("terminal"),evidence([]),_),query_the_user(EVIDENCE_H,
          USER_STATE,CERTAINTY),make_m_node_data(EVIDENCE_H,
          USER_STATE,CERTAINTY);node_structure(EVIDENCE_H,logic(_),
          evidence(_),_)), get_state(or,EVIDENCE_T) .

propagate_one_level(and,EVIDENCE_LIST,NODE) :-
      get_state(and,EVIDENCE_LIST), propagate_and(NODE) .
propagate_one_level(or,EVIDENCE_LIST,NODE) :-
      get_state(or,EVIDENCE_LIST), propagate_or(NODE) .
propagate_one_level("not",EVIDENCE_LIST,NODE) :-
      get_state("not",EVIDENCE_LIST), propagate_not(NODE) .

query_the_user(NODE_NAME,t,CERTAINTY). :-
      get_description(NODE_NAME,DESCRI_LIST), write(NODE_NAME), nl,
        write("Enter probability for $""),
        write_description(DESCRI_LIST), write("$":"), nl,
        read(CERTAINTY), nl .

get_description(NODE,DESCRIPTION) :-
      node_explanation(NODE,_,_,_,_,description(DESCRIPTION)) .
get_description(NODE,[" ? "]) .

write_description([]) .
write_description([D_H|D_T]) :-
      write(D_H), write_description(D_T) .

propagate_not(NODE) :-
      node_structure(NODE,_,evidence(EVIDENCE_LIST),_),
        remove_the_evidences(EVIDENCE_LIST,E_LIST),
        length(E_LIST,L), L)0, reverse(EVIDENCE_LIST,TRUE_OR_FAIL),
        calculate_not_probability(NODE,TRUE_OR_FAIL,POST_P,NEW_LS,
            NEW_LN),
        change_node_status(NODE,TRUE_OR_FAIL,POST_P,NEW_LS,NEW_LN,_
            ,_) .
propagate_not(NODE) :-
      node_status(NODE,_,_,STATE,_,_,_,_,_,_), string(STATE);
      write_suspended_all(NODE),
        query_the_user(NODE,STATE,CERTAINTY),
        make_m_node_data(NODE,STATE,CERTAINTY) .

change_node_status(NODE,STATE,P,_,_,_,_,_) :-
      node_status(NODE,C,A,ST,PR,_,_,_,_,_), string(ST),
        equal(STATE,ST), number(PR), equal(P,PR),
        change_node_status_for(NODE,C,A,STATE,P,_,unchanged,_,_,_) .
change_node_status(NODE,STATE,P,TS,TP,_,_,_) :-
      node_status(NODE,C,A,_,_,_,_,_,_,_),
        change_node_status_for(NODE,C,A,STATE,P,TS,changed,_,_,_) .
change_node_status(NODE,STATE,P,TS,TP,_,_,_) :-
      write("### "), write("("), write(NODE), write(")"),
        write(" status is not exist."), nl .

calculate_and_probability(NODE,E_LIST,TRUE_OR_FAIL,POST_P,_,_) :-
      node_type(NODE,_,dependency(D),_,_,_,_,_,_,_),
        (less(0,D), calculate_and_positive(E_LIST,D,POST_P), /;
         calculate_and_negative(E_LIST,D,POST_P)) .
```

```
      node_type(NODE,_,dependency(D),_,_,_,_,_,_,_),
        (less(0,D), calculate_or_positive(E_LIST,D,P), /;
        calculate_or_negative(E_LIST,D,P)) .

calculate_or_positive(E_LIST,D,P) :-
      length(E_LIST,L), or_independence(E_LIST,L,C1),
        max_or_dependence(E_LIST,C2),
        resulting_probability(D,C1,C2,P,positive) .

calculate_or_negative(E_LIST,D,P) :-
      length(E_LIST,L), or_independence(E_LIST,L,C1),
        min_or_dependence(E_LIST,C3),
        resulting_probability(D,C1,C3,P,negative) .

calculate_and_positive(E_LIST,D,P) :-
      and_independence(E_LIST,C1), max_and_dependence(E_LIST,C2),
        resulting_probability(D,C1,C2,P,positive) .

calculate_and_negative(E_LIST,D,P) :-
      and_independence(E_LIST,C1), min_and_dependence(E_LIST,C3),
        resulting_probability(D,C1,C3,P,negative) .

reverse([NODE],t) :-
      node_status(NODE,_,_,f,_,_,_,_,_,_) .
reverse([NODE],f) :-
      node_status(NODE,_,_,t,_,_,_,_,_,_) .

calculate_not_probability(NODE,T_OR_F,POST_P,_,_) :-
      node_structure(NODE,_,evidence([EVIDENCE_H|_]),_),
        node_status(EVIDENCE_H,_,_,_,PROBABILITY,_,_,_,_,_),
        minus(100,PROBABILITY,POST_P) .

propagate_or(NODE) :-
      node_structure(NODE,_,evidence(EVIDENCE_LIST),_),
        remove_the_evidences(EVIDENCE_LIST,E_LIST),
        length(E_LIST,L), L)0, check_one_true(E_LIST,TRUE_OR_FAIL),
        calculate_or_probability(NODE,E_LIST,TRUE_OR_FAIL,P,NS,NN),
        change_node_status(NODE,TRUE_OR_FAIL,P,NS,NN,_,_,_) .
propagate_or(NODE) :-
      node_status(NODE,_,_,STATE,_,_,_,_,_,_), string(STATE);
      write_suspended_all(NODE),
        query_the_user(NODE,STATE,CERTAINTY),
        make_m_node_data(NODE,STATE,CERTAINTY) .

write_suspended_all(NODE) :-
      write("The node ("), write(NODE), write(") is not terminal."),
        nl, write("But all the evidences of it "),
        write(" have been suspending by the SUSPEND command."), nl,
        nl .

check_one_true([],t) :-
      fail .
check_one_true([EVIDENCE_H|EVIDENCE_T],t) :-
      node_status(EVIDENCE_H,_,_,STATE,_,_,_,_,_,_), equal(STATE,t);
      check_one_true(EVIDENCE_T,t) .
check_one_true(_,f) .

propagate_and(NODE) :-
      node_structure(NODE,_,evidence(EVIDENCE_LIST),_),
        remove_the_evidences(EVIDENCE_LIST,E_LIST),
        length(E_LIST,L), L)0, check_all_true(E_LIST,TRUE_OR_FAIL),
        calculate_and_probability(NODE,E_LIST,TRUE_OR_FAIL,P,NS,NN),
        change_node_status(NODE,TRUE_OR_FAIL,P,NS,NN,_,_,_) .
propagate_and(NODE) :-
      node_status(NODE,_,_,STATE,_,_,_,_,_,_), string(STATE);
      write_suspended_all(NODE).
```

```
        query_tne_user(NODE,STATE,CERTAINTY),
        make_m_node_data(NODE,STATE,CERTAINTY) .

check_all_true([],t) .
check_all_true([EVIDENCE_H|EVIDENCE_T],t) :-
      node_status(EVIDENCE_H,_,_,STATE,_,_,_,_,_,_), equal(STATE,t),
        check_all_true(EVIDENCE_T,t) .
check_all_true(_,f) .

make_m_node_data(NODE,USER_STATE,PROBABILITY) :-
      calculate_probability(NODE,USER_STATE,INT_STATE,PROBABILITY,
          INT_PRO),
        change_node_status(NODE,INT_STATE,INT_PRO,USER_STATE,
            PROBABILITY,_,_,_) .

calculate_probability(NODE,USER_STATE,INT_STATE,PROBABILITY,
            INT_PRO) :-
      equal(USER_STATE,t),
        (less(PROBABILITY,0),
           (change_state(USER_STATE,INT_STATE),minus(100,
               PROBABILITY,INT_PRO)), /;
         unchange_state(USER_STATE,INT_STATE),
           INT_PRO is PROBABILITY) .
calculate_probability(NODE,USER_STATE,INT_STATE,PROBABILITY,
            INT_PRO) :-
      equal(USER_STATE,f),
        (less(PROBABILITY,0),
           (change_state(USER_STATE,INT_STATE),minus(100,
               PROBABILITY,INT_PRO)), /;
         unchange_state(USER_STATE,INT_STATE),
           INT_PRO is PROBABILITY) .

change_state(t,f) .
change_state(f,t) .

unchange_state(t,t) .
unchange_state(f,f) .

disp_pro :-
      c_product(X), write(" c_product("), write(X), write(")"), nl,
        fail .
disp_pro .

disp_max :-
      maximum(X), write(" maximum("), write(X), write(")"), nl,
        fail .
disp_max .

write_state(NODE,STATE,PROBERBILITY) :-
      string(STATE), write("The "), write(NODE), write(" is "),
        write(STATE), write(" with proberbility ("),
        write_probability_only(PROBERBILITY), write(")"),
        write("."), nl, / .
write_state(NODE,STATE,PROBERBILITY) :-
      / .

display_inference_net(NODE,N1,N2) :-
      depth_first_serch(NODE,N1,N2) .
display_inference_net(NODE,N1,N2) .

depth_first_serch(NODE,N1,N2) :-
      node_structure(NODE,logic(LOGIC),evidence(EVIDENCE),_),
        addclause(route(NODE)), bagof(N,route(N),INDENTY_LIST),
        length(INDENTY_LIST,LENGTH),
        check_range(LENGTH,N1,N2,WRITE),
        (equal(WRITE,write), (shift_indentation(LENGTH,N1,
```

```
        NEW_INDENTY),write_new_indenty(NEW_INDENTY),write_tree(
        NODE),/),/;/), /,
    check_terminal(NODE,EVIDENCE,NEW_EVIDENCE,LENGTH,N2),
    decide_a_way(NODE,NEW_EVIDENCE,N1,N2) .

write_new_indenty(NEW_INDENTY) :-
    times(3,NEW_INDENTY,PRODUCT), outspaces(PRODUCT) .

check_range(LENGTH,N1,all,write) :-
    LENGTH)=N1 .
check_range(LENGTH,N1,N2,write) :-
    LENGTH)=N1, LENGTH(=N2 .
check_range(LENGTH,N1,N2,not_write) .

shift_indentation(LENGTH,N1,NEW_INDENTY) :-
    minus(LENGTH,N1,NEW_INDENTY) .

write_tree(NODE) :-
    node_structure(NODE,logic(LOGIC),evidence([EVIDENCE_H|
        EVIDENCE_T]),_), get_dependence(NODE,DEPENDENCE),
      write_mark(NODE), write(NODE), write(" "), write("("),
      write(LOGIC),
      write_evidence_number([EVIDENCE_H|EVIDENCE_T]), write(")"),
      write(" "), write_dependence(DEPENDENCE),
      get_true_fail(NODE,TF),
      (equal(TF,t),write_probability(NODE),/;/), nl .
write_tree(NODE) :-
    node_structure(NODE,logic("terminal"),evidence([]),_),
      write_mark(NODE), write(NODE), write(" "), write("("),
      write("terminal"), write(")"), write_probability(NODE), nl .

get_true_fail(NODE,TF) :-
    node_status(NODE,_,_,TF,_,_,_,_,_,_) .
get_true_fail(NODE,nothing) .

write_probability(NODE) :-
    node_status(NODE,_,_,_,PRO,_,_,_,_,_), /, write(" "),
      write("P="), get_prob_range(PRO,W_TYPE),
      write_prob(PRO,W_TYPE) .
write_probability(NODE) :-
    / .

write_dependence(DEPENDENCE) :-
    write("D="), get_depend_range(DEPENDENCE,D_TYPE),
      write_depend(DEPENDENCE,D_TYPE) .

write_depend(_,minus_one) :-
    write("-1") .
write_depend(_,zero) :-
    write("0") .
write_depend(_,ten) :-
    write("1") .
write_depend(D,minus_9_1) :-
    times(D,-1,N), write("."), write(N) .
write_depend(D,one_9) :-
    write("."), write(D) .

get_depend_range(-10,minus_one) .
get_depend_range(0,zero) .
get_depend_range(10,ten) .
get_depend_range(D,minus_9_1) :-
    D(0, D) -10 .
get_depend_range(D,one_9) :-
    D)0, D(10 .
get_depend_range(D,d_range_error) :-
    write("### "), write("dependence range error!"), nl
```

```
        write(X), write(D),
        write(" is must be in the range -10...+10.") .

write_evidence_number(E_LIST) :-
      length(E_LIST,N), write_number(N) .

write_number(0) .
write_number(N) :-
      write(","), write(N) .

get_dependence(NODE,D) :-
      node_type(NODE,_,dependency(D),_,_,_,_,_,_,_) .
get_dependence(NODE,?) .

del_node_typ :-
      node_type(_,_,_,_,_,_,_,_,_,_),
        fdelclause(node_type(_,_,_,_,_)), fail .
del_node_typ .

write_mark(NODE) :-
      node_status(NODE,_,_,STATE,P,_,_,_,_,_), string(STATE),
        equal(STATE,t), decide_true(NODE,STATE,P), /, write("* ") .
write_mark(NODE) :-
      node_status(NODE,C,_,_,_,_,_,_,_,_), string(C),
        equal(C,suspended), /, write("s ") .
write_mark(NODE) :-
      write("  "), / .

decide_true(NODE,STATE,P) :-
      node_type(NODE,_,_,_,_,_,_,_,_,_), threshold(TH),
        adjust(P,PN), PN)=TH, / .

adjust(P,P) .

disp_route :-
      route(X), write("route -) "), write(X), nl, fail .
disp_route .

check_terminal(NODE,EVIDENCE,EVIDENCE,LENGTH,N2) :-
      check_fail(LENGTH,N2),
        node_structure(NODE,logic("terminal"),evidence([]),_),
        fdelclause(route(NODE)), fail .
check_terminal(NODE,EVIDENCE,[],LENGTH,N2) :-
      number(N2), LENGTH=:=N2, fdelclause(route(NODE)), fail .
check_terminal(NODE,E,E,LENGTH,N2) :-
      check_fail(LENGTH,N2),
        node_structure(NODE,logic(LOGIC),evidence(EVIDENCE),_),
          (equal(LOGIC,"not");
           equal(LOGIC,and);
           equal(LOGIC,or)) .

check_fail(LENGTH,N2) :-
      number(N2), LENGTH=:=N2, fail .
check_fail(LENGTH,N2) :-
      number(N2), LENGTH)N2 .
check_fail(LENGTH,N2) :-
      number(N2), LENGTH(N2 .
check_fail(LENGTH,all) .

decide_a_way(NODE,[],N1,N2) :-
      fdelclause(route(NODE)), fail .
decide_a_way(NODE,[EVIDENCE_H|EVIDENCE_T],N1,N2) :-
      depth_serch(NODE,EVIDENCE_H,N1,N2);
      decide_a_way(NODE,EVIDENCE_T,N1,N2) .

depth_serch(NODE,EVIDENCE_H,N1,N2) :-
```

```
      call(depth_first_search(EVIDENCE_H,N1,N2)) .

cut(X) :-
      X, / .

disp_tree(NODE) :-
      kill(route), display_inference_net(NODE,1,all) .

disp_tree(NODE,N) :-
      kill(route), display_inference_net(NODE,1,N) .

disp_tree(NODE,N1,N2) :-
      kill(route), display_inference_net(NODE,N1,N2) .

and_independence(EVIDENCE_LIST,AND_C1) :-
      addclause(c_product(1)), multiply(EVIDENCE_LIST),
        divide(EVIDENCE_LIST,AND_C1), fdelclause(c_product(_)) .

multiply([]) .
multiply([EVIDENCE_H|EVIDENCE_T]) :-
      c_product(X), get_probability_or(EVIDENCE_H,PROBABILITY),
        times(PROBABILITY,X,PRODUCT), fdelclause(c_product(X)),
        addclause(c_product(PRODUCT)), multiply(EVIDENCE_T) .

or_independence(EVIDENCE_LIST,2,OR_C1) :-
      addclause(sum(0)), addition(EVIDENCE_LIST), sum(SUM),
        fdelclause(sum(SUM)), addclause(c_product(1)),
        multiply(EVIDENCE_LIST), divide(EVIDENCE_LIST,PRODUCT),
        fdelclause(c_product(_)), minus(SUM,PRODUCT,OR_C1) .
or_independence(EVIDENCE_LIST,3,OR_C1) :-
      addclause(sum(0)), addition(EVIDENCE_LIST), sum(P),
        fdelclause(sum(P)), addclause(c_product(1)),
        multiply(EVIDENCE_LIST), divide(EVIDENCE_LIST,PPP),
        fdelclause(c_product(_)),
        separate_arg(EVIDENCE_LIST,[E_A,E_B,E_C]),
        addclause(c_product(1)), multiply([E_A,E_B]),
        divide([E_A,E_B],PAB), fdelclause(c_product(_)),
        addclause(c_product(1)), multiply([E_A,E_C]),
        divide([E_A,E_C],PAC), fdelclause(c_product(_)),
        addclause(c_product(1)), multiply([E_B,E_C]),
        divide([E_B,E_C],PBC), fdelclause(c_product(_)),
        plus(P,PPP,X), minus(X,PAB,X1), minus(X1,PAC,X2),
        minus(X2,PBC,OR_C1) .
or_independence(EVIDENCE_LIST,1,OR_C1) :-
      one_evidence_pro(EVIDENCE_LIST,OR_C1) .

separate_arg([E_A,E_B,E_C],[E_A,E_B,E_C]) .

one_evidence_pro([EVIDENCE],OR_C1) :-
      node_status(EVIDENCE,_,_,_,OR_C1,_,_,_,_,_) .

divide(EVIDENCE_LIST,RESULT) :-
      length(EVIDENCE_LIST,LENGTH), minus(LENGTH,1,L),
        power(100,L,X), c_product(Y), div(Y,X,RESULT) .

max_or_dependence(EVIDENCE_LIST,OR_C2) :-
      addclause(maximum(0)), find_max(EVIDENCE_LIST),
        maximum(OR_C2), fdelclause(maximum(X)) .

find_max([]) .
find_max([EVIDENCE_H|EVIDENCE_T]) :-
      maximum(MAX_PRO), get_probability_or(EVIDENCE_H,PROBABILITY),
        (less(PROBABILITY,MAX_PRO),/,/;fdelclause(maximum(MAX_PRO))
            ,addclause(maximum(PROBABILITY))), find_max(EVIDENCE_T) .

get_probability_or(EVIDENCE_H,PROBABILITY) :-
```

```
     node_status(EVIDENCE_H,_,_,_,PROBABILITY,_,_,_,_,_) .
get_probability_or(EVIDENCE_H,0) .

max_and_dependence(EVIDENCE_LIST,AND_C2) :-
     addclause(minimum(100)), find_min(EVIDENCE_LIST),
       minimum(AND_C2), fdelclause(minimum(X)) .

min_or_dependence(EVIDENCE_LIST,OR_C3) :-
     addclause(sum(0)), addition(EVIDENCE_LIST), sum(SUM),
       fdelclause(sum(SUM)), less(SUM,100), OR_C3 is SUM, /;
     OR_C3 is 100 .

addition([]) .
addition([EVIDENCE_H|EVIDENCE_T]) :-
     sum(SUM), get_probability_or(EVIDENCE_H,PROBABILITY),
       plus(SUM,PROBABILITY,NEW_SUM), fdelclause(sum(SUM)),
       addclause(sum(NEW_SUM)), addition(EVIDENCE_T) .

disp_sum :-
     sum(X), write(" sum("), write(X), write(")"), nl, fail .
disp_sum .

find_min([]) .
find_min([EVIDENCE_H|EVIDENCE_T]) :-
     minimum(MIN_PRO),
       node_status(EVIDENCE_H,_,_,_,PROBABILITY,_,_,_,_,_),
       (less(MIN_PRO,PROBABILITY),/,/;fdelclause(minimum(MIN_PRO))
           ,addclause(minimum(PROBABILITY))), find_min(EVIDENCE_T) .

min_and_dependence(EVIDENCE_LIST,AND_C3) :-
     addclause(sum(0)), addition(EVIDENCE_LIST), sum(SUM),
       fdelclause(sum(X)), length(EVIDENCE_LIST,LENGTH),
       minus(LENGTH,1,L), times(100,L,Y), minus(SUM,Y,Z),
       less(0,Z), AND_C3 is Z, /;
     AND_C3 is 0 .

resulting_probability(D,C1,C2,P,positive) :-
     times(D,C2,X), minus(10,D,Y), times(C1,Y,Z), plus(X,Z,X1),
       div(X1,10,P) .
resulting_probability(D,C1,C3,P,negative) :-
     times(D,-1,X2), times(X2,C3,X), minus(10,X2,Y), times(C1,Y,Z),
       plus(X,Z,X1), div(X1,10,P) .

display_expression(NODE) :-
     addclause(ori_node(NODE)), addclause(ex_list([init])),
       depth_first_ex(NODE) .
display_expression(NODE) :-
     fdelclause(ori_node(X)), fdelclause(ex_list(X)) .
display_expression(NODE) :-
     del_ex_l .

check_explainable(NODE) :-
     node_structure(NODE,_,_,_), ori_node(O_NODE),
       (equal(NODE,O_NODE), /, /;
        fdelclause(ex_route(NODE)), fail) .

del_ori :-
     ori_node(X), fdelclause(ori_node(X)), fail .
del_ori .

del_ex_r :-
     ex_route(X), fdelclause(ex_route(X)), fail .
del_ex_r .

del_ex_l :-
     ex_list(Y), fdelclause(ex_list(Y)), fail
```

```
del_ex_l .

del_ex :-
     del_ex_r, del_ex_l, del_ori . /

reverse_list(L,R) :-
     rev0(L,[],R) .

rev0([],ACCUM,ACCUM) .
rev0([HEAD|TAIL],ACCUM,RESULT) :-
     rev0(TAIL,[HEAD|ACCUM],RESULT) .

decide_ex_way(NODE,[]) :-
     fdelclause(ex_route(NODE)), write(")"), fail .
decide_ex_way(NODE,[EVIDENCE_H|EVIDENCE_T]) :-
     depth_ex_serch(NODE,EVIDENCE_H);
     decide_ex_way(NODE,EVIDENCE_T) .

depth_ex_serch(NODE,EVIDENCE_H) :-
     cut(depth_first_ex(EVIDENCE_H)) .

depth_first_ex(NODE) :-
     node_structure(NODE,logic(LOGIC),evidence(EVIDENCE),_),
       addclause(ex_route(NODE)), bagof(N,ex_route(N),EX_LIST),
       reverse_list(EX_LIST,R_LIST), get_second(R_LIST,S_NODE),
       node_structure(S_NODE,logic(S_LOGIC),_,_), ex_list(OLD),
       fdelclause(ex_list(OLD)), addclause(ex_list(EX_LIST)),
       length(OLD,LO), length(EX_LIST,LN), ori_node(ORI_NODE),
       (equal(LO,LN),/,/;less(LO,LN),(write(" ("),write(S_LOGIC)),
           /;/),
       (equal(explainable,explainable),(write(" "),write(NODE)),/;
           /), check_explainable(NODE),
       decide_ex_way(NODE,EVIDENCE) .

disp_exp(NODE) :-
     display_expression(NODE) .

get_second(R_LIST,S_NODE) :-
     length(R_LIST,1), ori_node(S_NODE) .
get_second([_,S_NODE|_],S_NODE) .

read_pro(NODE_NAME,TRUE_OR_FAIL) :-
     form_list(NODE_NME,EVI_LIST), read_ans(EVI_LIST,TRUE_OR_FAIL) .

form_list(NODE_NAME,[NODE_NAME]) .
```

```
$ ty comkb.log
/*                                                    *
/* Common Knowledge Base                              *
/*                                                    *
node_structure(spacecraft_lost,logic(and),evidence([
          electronics_innert,antenna_ineffective,telemetry_lost,
          spacecraft_mechanically_frozen]),"dead") .
node_structure(electronics_innert,logic(and),evidence([
          heaters_ineffective,electrical_shutdown]),"6.1)d.ii)2") .
node_structure(antenna_ineffective,logic(and),evidence([
          spacecraft_tumbles,electrical_shutdown]),"no antenna") .
node_structure(telemetry_lost,logic("terminal"),evidence([]),
          "6.1).i****") .
node_structure(spacecraft_mechanically_frozen,logic(or),evidence([
          heaters_ineffective]),"frozen") .
node_structure(heaters_ineffective,logic(or),evidence([
          electrical_shutdown]),"6.1)d.ii)1") .
node_structure(spacecraft_tumbles,logic(and),evidence([wheel_stops
          ,large_cone_develops,pitch_changes_greatly]),
          "6.1)d.ii)1)1") .
node_structure(wheel_stops,logic(or),evidence([electrical_shutdown
          ]),"6.1)d.ii)1") .
node_structure(electrical_shutdown,logic(or),evidence([uvs_trips])
          ,"6.1)d.ii") .
node_structure(uvs_trips,logic(and),evidence([batteries_exhausted,
          charging_limited]),"6.1)c.ii") .
node_structure(batteries_exhausted,logic(or),evidence([
          power_loss_1,power_loss_2]),"6.1.c.ia") .
node_structure(charging_limited,logic(or),evidence([
          tracking_partially_successful]),"6.1)c.i") .
node_structure(power_loss_1,logic(and),evidence([
          catalyst_bed_heater_on,heavy_tracking_power]),"6.1)c") .
node_structure(power_loss_2,logic(or),evidence([
          catalyst_bed_heater_on,heavy_tracking_power]),"6.1)c") .
node_structure(catalyst_bed_heater_on,logic(or),evidence([
          recovery_procedure_begins]),"heater") .
node_structure(heavy_tracking_power,logic(or),evidence([
          continuous_tracking]),"hard work") .
node_structure(tracking_partially_successful,logic(or),evidence([
          solar_array_off_angle,attitude_control_lost]),"_") .
node_structure(continuous_tracking,logic(and),evidence([
          solar_array_off_angle,attitude_control_lost]),"always") .
node_structure(attitude_control_lost,logic(and),evidence([
          command_not_receivable,pitch_changes_greatly,
          large_cone_develops]),"no control") .
node_structure(command_not_receivable,logic(or),evidence([
          receive_antenna_off_angle]),"no rcv") .
node_structure(o4_firing_stops,logic(and),evidence([
          main_tank_valve_closes,fuel_in_line_becomes_scarce,
          fuel_pressure_drops]),"firing stops") .
node_structure(solar_array_off_angle,logic(and),evidence([
          pitch_changes_greatly,large_cone_develops]),"off_angle") .
node_structure(receive_antenna_off_angle,logic(or),evidence([
          large_cone_develops,pitch_changes_greatly]),"no antenna"
          ) .
node_structure(shf_lost,logic(or),evidence([pitch_changes_greatly,
          large_cone_develops]),"no SHF") .
node_structure(fuel_in_line_becomes_scarce,logic(and),evidence([
          limited_fuel_in_fuel_line,o4_firing_continues]),
          "o4 cont") .
node_structure(pitch_changes_greatly,logic(or),evidence([
          o4_firing_continues]),"pitch") .
node_structure(large_cone_develops,logic(and),evidence([
          o4_firing_continues,negative_pitch_develops]),"cone") .
node_structure(o4_firing_continues,logic(and),evidence([
```

```
          high_rate_command_continues,
          pressure_in_fuel_line_maintains,o4_fires]),"big thrust") .
node_structure(high_rate_command_continues,logic(and),evidence([
          nesa_a_output_saturates,nesa_a_has_earth_presence,
          nesa_a_takes_over_roll_yaw_control]),"long cmd") .
node_structure(pressure_in_fuel_line_maintains,logic(or),evidence(
          [additional_fuel_vaporizes]),"pressure") .
node_structure(additional_fuel_vaporizes,logic(and),evidence([
          fuel_pressure_drops,multi_face_flow_in_fuel_line]),
          "more") .
node_structure(fuel_pressure_drops,logic(or),evidence([o4_fires]),
          "drop") .
node_structure(o4_fires,logic(and),evidence([
          roll_yaw_command_issued,limited_fuel_in_fuel_line]),
          "o4 fired") .
node_structure(roll_yaw_command_issued,logic(and),evidence([
          nesa_a_output_saturates,
          nesa_a_takes_over_roll_yaw_control]),"cmd") .
node_structure(nesa_a_takes_over_roll_yaw_control,logic(and),
          evidence([nesa_b_loses_earth_presence,
          nesa_a_has_earth_presence]),"takeover") .
node_structure(fuel_control_inaccurate,logic(or),evidence([
          multi_face_flow_in_fuel_line]),"inaccurate") .
node_structure(nesa_b_loses_earth_presence,logic(or),evidence([
          negative_pitch_develops]),"lose earth") .
node_structure(multi_face_flow_in_fuel_line,logic(and),evidence([
          multi_face_flow_potential_in_tank,
          limited_fuel_in_fuel_line]),"multiface") .
node_structure(negative_pitch_develops,logic(or),evidence([
          wheel_speed_drops]),"negpitch") .
node_structure(multi_face_flow_potential_in_tank,logic(or),evidence
          ([nitrogen_in_hydrazine,unresolved_nitrogen_in_tank,
          unspecified_gas_in_tank]),"potential") .
node_structure(limited_fuel_in_fuel_line,logic(and),evidence([
          main_tank_valve_closes,o4_previously_fired]),
          "limited fuel") .
node_structure(wheel_speed_drops,logic(or),evidence([cws_mode_on])
          ,"6.1)a.iii)1") .
node_structure(nitrogen_in_hydrazine,logic(or),evidence([
          nitrogen_thru_diaphragm,fuel_tank_temp_cycles]),
          "resolved") .
node_structure(unresolved_nitrogen_in_tank,logic(and),evidence([
          nitrogen_to_pressure,tank_pressure_low]),"unresolved") .
node_structure(unspecified_gas_in_tank,logic(and),evidence([
          fuel_tank_temp_cycles,impurities_in_tank]),"other gas") .
node_structure(main_tank_valve_closes,logic(or),evidence([
          afp_trips]),"6.1)a.i") .
node_structure(switch_to_redundant_ace_and_mwc,logic(or),evidence(
          [afp_trips]),"6.1)a.ii") .
node_structure(cws_mode_on,logic(or),evidence([afp_trips]),
          "6.1)a.iii") .
node_structure(o4_previously_fired,logic("terminal"),evidence([]),
          "****") .
node_structure(nitrogen_thru_diaphragm,logic(and),evidence([
          diaphragm_leaks,nitrogen_to_pressure]),"leak") .
node_structure(recovery_procedure_begins,logic(or),evidence([
          nesa_a_output_saturates]),"procedure!") .
node_structure(afp_trips,logic(or),evidence([
          nesa_a_output_saturates]),"6.1)a") .
node_structure(nesa_a_has_earth_presence,logic(or),evidence([
          nesa_a_output_saturates]),"earth") .
node_structure(diaphragm_leaks,logic("terminal"),evidence([]),
          "material") .
node_structure(nitrogen_to_pressure,logic("terminal"),evidence([])
          ,"nitrogen") .
node_structure(tank_pressure_low,logic(or),evidence(
```

```
          fuel_in_tank_low]),"low_fuel") .
node_structure(fuel_tank_temp_cycles,logic(and),evidence([
          sun_position_always_changes,heat_dissipation_uneven]),
          "tank temp") .
node_structure(impurities_in_tank,logic("terminal"),evidence([]),
          "impure") .
node_structure(nesa_a_output_saturates,logic(or),evidence([
          nesa_a_saturation_1,nesa_a_saturation_2,
          excessive_nesa_a_power_cycling]),"6.1") .
node_structure(nesa_a_saturation_1,logic(or),evidence([
          charged_energy,mirror_stuck]),"6.1.4.b") .
node_structure(nesa_a_saturation_2,logic(or),evidence([
          and_electronics,or_electronics]),"6.1.4.b") .
node_structure(fuel_in_tank_low,logic("terminal"),evidence([]),
          "fuel low") .
node_structure(heat_dissipation_uneven,logic("terminal"),evidence(
          []),"uneven") .
node_structure(charged_energy,logic(or),evidence([sun_reflections,
          shf_radiation]),"6.1.4.3") .
node_structure(mirror_stuck,logic(or),evidence([
          scan_mechanism_fails,scan_motor_fails]),"6.1.3.a") .
node_structure(and_electronics,logic(and),evidence([
          control_electronics_fails,emi_to_electronics]),"6.1.4.1"
          ) .
node_structure(or_electronics,logic(or),evidence([
          control_electronics_fails,emi_to_electronics]),"6.1.4.2"
          ) .
node_structure(scan_mechanism_fails,logic(or),evidence([
          thermal_distortion,unstable_pivot,
          mechanism_contamination]),"6.1.3.a") .
node_structure(scan_motor_fails,logic(or),evidence([motor_fails,
          motor_overheats]),"6.1.3.b") .
node_structure(sun_reflections,logic("terminal"),evidence([]),
          "6.1.4.3.1") .
node_structure(shf_radiation,logic("terminal"),evidence([]),
          "6.1.4.3.2") .
node_structure(thermal_distortion,logic(and),evidence([
          sun_position_always_changes,anomalies_relate_to_sun_pos]
          ),"6.1.3.1") .
node_structure(unstable_pivot,logic("terminal"),evidence([]),
          "6.1.3.2") .
node_structure(mechanism_contamination,logic("terminal"),evidence(
          []),"6.1.3.3") .
node_structure(motor_fails,logic("terminal"),evidence([]),
          "6.1.3.4") .
node_structure(motor_overheats,logic("terminal"),evidence([]),
          "6.1.3.5") .
node_structure(control_electronics_fails,logic("terminal"),evidence
          ([]),"6.1.4.1.1") .
node_structure(emi_to_electronics,logic("terminal"),evidence([]),
          "6.1.4.1.2") .
node_structure(excessive_nesa_a_power_cycling,logic(and),evidence(
          [power_needs_to_be_cut_to_eliminate_output,
          nesa_a_output_must_be_cut_out]),"overcycled") .
node_structure(power_needs_to_be_cut_to_eliminate_output,logic(
          "terminal"),evidence([]),*) .
node_structure(sun_position_always_changes,logic("terminal"),
          evidence([]),"6.1.3.1.1") .
node_structure(anomalies_relate_to_sun_pos,logic("terminal"),
          evidence([]),"6.1.3.1.2") .
node_structure(power_cut_to_eliminate_output,logic("terminal"),
          evidence([]),"****") .
node_structure(nesa_a_output_must_be_cut_out,logic("terminal"),
          evidence([]),"****") .

node_message(msg1,["message1"]) .
```

```
node_message(msg2, ["message2"]) .
node_message(msg3, ["message3"]) .
node_message(msg4, ["message4"]) .
node_message(msg5, ["message5"]) .
node_message(msg6, ["message6"]) .
node_message(msg7, ["message7"]) .
node_message(msg8, ["message8"]) .
node_message(msg9, ["message9"]) .
node_message(msg10, ["message10"]) .
node_message(msg12, ["message12"]) .
node_message(msg11, ["message11"]) .
node_message(msg13, ["message13"]) .
node_message(msg14, ["message14"]) .
node_message(msg15, ["message15"]) .
node_message(msg16, ["message16"]) .
node_message(msg17, ["message17"]) .
node_message(msg18, ["message18"]) .
node_message(msg19, ["message19"]) .
node_message(msg20, ["message20"]) .
node_message(msg21, ["message21"]) .
node_message(msg22, ["message22"]) .
node_message(msg23, ["message23"]) .
node_message(msg24, ["message24"]) .
node_message(msg25, ["message25"]) .
node_message(msg26, ["message26"]) .
node_message(msg27, ["message27"]) .
node_message(msg28, ["message28"]) .
node_message(msg29, ["message29"]) .
node_message(msg30, ["message30"]) .
node_message(msg31, ["message31"]) .
node_message(msg32, ["message32"]) .
node_message(msg33, ["message33"]) .
node_message(msg34, ["message34"]) .
node_message(msg35, ["message35"]) .
node_message(msg36, ["message36"]) .
node_message(msg37, ["message37"]) .
node_message(msg38, ["message38"]) .
node_message(msg39, ["message39"]) .
node_message(msg40, ["message40"]) .
node_message(msg41, ["message41"]) .
node_message(msg42, ["message42"]) .
node_message(msg43, ["message43"]) .
node_message(msg44, ["message44"]) .
node_message(msg45, ["message45"]) .
node_message(msg46, ["message46"]) .
node_message(msg47, ["message47"]) .
node_message(msg48, ["message48"]) .
node_message(msg49, ["message49"]) .
node_message(msg50, ["message50"]) .
node_message(msg52, ["message52"]) .
node_message(msg51, ["message51"]) .
node_message(msg53, ["message53"]) .
node_message(msg54, ["message54"]) .
node_message(msg55, ["message55"]) .
node_message(msg56, ["message56"]) .
node_message(msg57, ["message57"]) .
node_message(msg58, ["message58"]) .
node_message(msg59, ["message59"]) .
node_message(msg60, ["message60"]) .
node_message(msg61, ["message61"]) .
node_message(msg62, ["message62"]) .
node_message(msg63, ["message63"]) .
node_message(msg64, ["message64"]) .
node_message(msg65, ["message65"]) .
node_message(msg66, ["message66"]) .
node_message(msg67, ["message67"]) .
```

```
node_message(msg68,["message68"]) .
node_message(msg69,["message69"]) .
node_message(msg70,["message70"]) .
node_message(msg71,["message71"]) .
node_message(msg72,["message72"]) .
node_message(msg73,["message73"]) .
node_message(msg74,["message74"]) .
node_message(msg75,["message75"]) .
node_message(msg76,["message76"]) .
node_message(msg77,["message77"]) .
node_message(msg78,["message78"]) .
node_message(msg79,["message79"]) .
node_message(msg80,["message80"]) .
node_message(msg81,["message81"]) .
node_message(msg82,["message82"]) .
node_message(msg83,["message83"]) .
node_message(msg84,["message84"]) .
node_message(msg85,["message85"]) .
node_message(msg86,["message86"]) .
node_message(msg87,["message87"]) .
node_message(msg88,["message88"]) .
node_message(msg89,["message89"]) .
node_message(msg90,["message90"]) .
node_message(uvs_trips_rec,["Disable UVS"]) .
node_message(uvs_trips_exp,["the UVS is known to malfunction."]) .

node_explanation(spacecraft_lost,g_type(1),v_depth(2),_,_,_,
          description(["Spacecraft is lost"])) .
node_explanation(electronics_innert,g_type(1),v_depth(2),_,_,_,
          description(["Most on-board electronics are innert"])) .
node_explanation(antenna_ineffective,g_type(1),v_depth(2),_,_,_,
          description([
          "Command receive antenna is not functioning at all"])) .
node_explanation(telemetry_lost,g_type(1),v_depth(2),_,_,_,
          description(["Telemetry from the spacecraft is lost"])) .
node_explanation(spacecraft_mechanically_frozen,g_type(1),v_depth(
          2),_,_,_,description([
          "Specacraft is mechanically frozen"])) .
node_explanation(heaters_ineffective,g_type(1),v_depth(2),_,_,_,
          description([
          "On-board equipment heaters are not functioning anymore"
          ])) .
node_explanation(spacecraft_tumbles,g_type(1),v_depth(2),_,_,_,
          description(["Spacecraft is tumbling"])) .
node_explanation(wheel_stops,g_type(1),v_depth(2),_,_,_,description
          (["On-board momentum control wheel has stopped"])) .
node_explanation(electrical_shutdown,g_type(1),v_depth(2),_,_,_,
          description(["On-board electrical system is shut down"])
          ) .
node_explanation(uvs_trips,g_type(1),v_depth(2),_,_,_,description(
          ["Under voltage protection system is activated"])) .
node_explanation(batteries_exhausted,g_type(1),v_depth(2),_,_,_,
          description(["On-board batteries are exhausted"])) .
node_explanation(charging_limited,g_type(1),v_depth(2),_,_,_,
          description([
          "Solar array$'s ability to charge on-board batteries",
          " is now limited"])) .
node_explanation(power_loss_1,g_type(1),v_depth(2),_,_,_,
          description([""])) .
node_explanation(power_loss_2,g_type(1),v_depth(2),_,_,_,
          description([""])) .
node_explanation(catalyst_bed_heater_on,g_type(1),v_depth(2),_,_,_
          ,description([
          "Catalyst bed heater for thruster is turned on"])) .
node_explanation(heavy_tracking_power,g_type(1),v_depth(2),_,_,_,
          description(["There is a heavy power drain due to track$
```

```
          $ing of solar array"])) .
node_explanation(tracking_partially_successful,g_type(1),v_depth(2
          ),_,_,_,description([
          "Tracking of the sun by solar array",
          " is only partially successful"])) .
node_explanation(continuous_tracking,g_type(1),v_depth(2),_,_,_,
          description([
          "The solar array is now tracking the sun continuously"])
          ) .
node_explanation(attitude_control_lost,g_type(1),v_depth(2),_,_,_,
          description(["Attitude control is no longer effective"])
          ) .
node_explanation(command_not_receivable,g_type(1),v_depth(2),_,_,_
          ,description([
          "The spacecraft is not receiving command sequences"])) .
node_explanation(o4_firing_stops,g_type(1),v_depth(2),_,_,_,
          description(["Firing of the O4 thruster is stopped"])) .
node_explanation(solar_array_off_angle,g_type(1),v_depth(2),_,_,_,
          description(["The solar arrays are not facing the sun"])
          ) .
node_explanation(receive_antenna_off_angle,g_type(1),v_depth(2),_,
          _,_,description([
          "The command receive antenna is not properly aligned",
          " to the ground"])) .
node_explanation(shf_lost,g_type(1),v_depth(2),_,_,_,description([
          "The SHF communication channel is lost"])) .
node_explanation(fuel_in_line_becomes_scarce,g_type(1),v_depth(2),
          _,_,_,description([
          "Residual fuel in the fuel line becomes scarce"])) .
node_explanation(pitch_changes_greatly,g_type(1),v_depth(2),_,_,_,
          description(["The spacecraft$'s pitch changes greatly f$
          $rom nominal negative",
          " pitch to a large positive pitch"])) .
node_explanation(large_cone_develops,g_type(1),v_depth(2),_,_,_,
          description([
          "A large nutation cone develops around the pitch axis"])
          ) .
node_explanation(o4_firing_continues,g_type(1),v_depth(2),_,_,_,
          description(["The firing of the offset thruster ",
          "O4 is maintained"])) .
node_explanation(high_rate_command_continues,g_type(1),v_depth(2),
          _,_,_,description([
          "The thruster O4 fire command continues at a high rate"]
          )) .
node_explanation(pressure_in_fuel_line_maintains,g_type(1),v_depth
          (2),_,_,_,description([
          "The pressure in the fuel line is maintained"])) .
node_explanation(additional_fuel_vaporizes,g_type(1),v_depth(2),_,
          _,_,description([
          "An additional amount of fuel vaporizes"])) .
node_explanation(fuel_pressure_drops,g_type(1),v_depth(2),_,_,_,
          description([
          "The pressure of fuel in the fuel pipe drops"])) .
node_explanation(o4_fires,g_type(1),v_depth(2),_,_,_,description([
          "The negative pitch offset thruster O4 fires"])) .
node_explanation(roll_yaw_command_issued,g_type(1),v_depth(2),_,_,
          _,description(["The roll-yaw control command is issued"]
          )) .
node_explanation(nesa_a_takes_over_roll_yaw_control,g_type(1),
          v_depth(2),_,_,_,description([
          "NESA-A cross scan takes over the control of",
          " the roll/yaw axes"])) .
node_explanation(fuel_control_inaccurate,g_type(1),v_depth(2),_,_,
          _,description(["Fuel flow control is no longer accurate"
          ])) .
```

```
          "NESA-B primary scan loses the sight of the earth"])) .
node_explanation(multi_face_flow_in_fuel_line,g_type(1),v_depth(2)
          ,_,_,_,description([
          "Multi-face flow of fuel exists in the fuel line"])) .
node_explanation(negative_pitch_develops,g_type(1),v_depth(2),_,_,
          _,description(["A nominal negative rotation begins arou$
          $nd the pitch axix"])) .
node_explanation(multi_face_flow_potential_in_tank,g_type(1),
          v_depth(2),_,_,_,description([
          "There is a potential fuel multi-flow situation",
          " in the fuel tank"])) .
node_explanation(limited_fuel_in_fuel_line,g_type(1),v_depth(2),_,
          _,_,description(["There is a limited amount of fuel lef$
          $t in the fuel line"])) .
node_explanation(wheel_speed_drops,g_type(1),v_depth(2),_,_,_,
          description(["The speed of the reaction control wheel d$
          $rops nominal 15 rpm"])) .
node_explanation(nitrogen_in_hydrazine,g_type(1),v_depth(2),_,_,_,
          description([
          "Nitrogen gas is resolved in hydrazine fuel"])) .
node_explanation(unresolved_nitrogen_in_tank,g_type(1),v_depth(2),
          _,_,_,description([
          "Unresolved nitrogen gas permiates through diaphragm"])) .
node_explanation(unspecified_gas_in_tank,g_type(1),v_depth(2),_,_,
          _,description(["Unspecified gas exists in the fuel tank"
          ])) .
node_explanation(main_tank_valve_closes,g_type(1),v_depth(2),_,_,_
          ,description(["The main fuel tank valve closes"])) .
node_explanation(switch_to_redundant_ace_and_mwc,g_type(1),v_depth
          (2),_,_,_,description([
          "The ACE and MWC units are switched to redundant unit"])
          ) .
node_explanation(cws_mode_on,g_type(1),v_depth(2),_,_,_,description
          (["The Constant Wheel Speed mode is on"])) .
node_explanation(o4_previously_fired,g_type(1),v_depth(2),_,_,_,
          description(["The negative pitch offset thruster O4 has"
          ," previously been fired"])) .
node_explanation(nitrogen_thru_diaphragm,g_type(1),v_depth(2),_,_,
          _,description([
          "Nitrogen gas permiates through the diaphragm"])) .
node_explanation(recovery_procedure_begins,g_type(1),v_depth(2),_,
          _,_,description([
          "A predefined NESA-A saturation recovery procedure is",
          " put into effect"])) .
node_explanation(afp_trips,g_type(1),v_depth(2),_,_,_,description(
          ["The Automatic Failure Protection mode is enforced"])) .
node_explanation(nesa_a_has_earth_presence,g_type(1),v_depth(2),_,
          _,_,description(["NESA-A has the earth presence"])) .
node_explanation(diaphragm_leaks,g_type(1),v_depth(2),_,_,_,
          description(["The diaphragm material leaks nitrogen gas"
          ])) .
node_explanation(nitrogen_to_pressure,g_type(1),v_depth(2),_,_,_,
          description([
          "Nitrogen gas is used to pressure diaphragm"])) .
node_explanation(tank_pressure_low,g_type(1),v_depth(2),_,_,_,
          description(["Pressure in fuel tank is low"])) .
node_explanation(fuel_tank_temp_cycles,g_type(1),v_depth(2),_,_,_,
          description(["Temperature within fuel tank cycles"])) .
node_explanation(impurities_in_tank,g_type(1),v_depth(2),_,_,_,
          description([
          "There are impurities in fuel and/or tank materials"])) .
node_explanation(nesa_a_output_saturates,g_type(1),v_depth(2),_,_,
          _,description([
          "Both NESA-A prime and cross scan outputs saturate"])) .
node_explanation(fuel_in_tank_low,g_type(1),v_depth(2),_,_,_,
```

```
          description(["remaining fuel in tank is low"])) .
node_explanation(heat_dissipation_uneven,g_type(1),v_depth(2),_,_,
          _,description([
          "Heat dissipation around fuel tank is uneven"])) .
node_explanation(nesa_a_saturation_1,g_type(1),v_depth(2),_,_,_,
          description([""])) .
node_explanation(nesa_a_saturation_2,g_type(1),v_depth(2),_,_,_,
          description([""])) .
node_explanation(charged_energy,g_type(1),v_depth(2),_,_,_,
          description(["The structure of the spacecraft is electr$
          $ically charged"])) .
node_explanation(mirror_stuck,g_type(1),v_depth(2),_,_,_,
          description(["The mirror scan mechanism is stuck"])) .
node_explanation(and_electronics,g_type(1),v_depth(2),_,_,_,
          description([""])) .
node_explanation(or_electronics,g_type(1),v_depth(2),_,_,_,
          description([""])) .
node_explanation(scan_mechanism_fails,g_type(1),v_depth(2),_,_,_,
          description(["The scanning mechanism fails"])) .
node_explanation(scan_motor_fails,g_type(1),v_depth(2),_,_,_,
          description(["scanning motor fails"])) .
node_explanation(sun_reflections,g_type(1),v_depth(2),_,_,_,
          description([
          "The sun reflections causes the spacecraft charged"])) .
node_explanation(shf_radiation,g_type(1),v_depth(2),_,_,_,
          description([
          "Radiation from the on-board SHF equipment causes",
          " the spacecraft to charge"])) .
node_explanation(thermal_distortion,g_type(1),v_depth(2),_,_,_,
          description([
          "The scanning mechanism is thermally distorted"])) .
node_explanation(unstable_pivot,g_type(1),v_depth(2),_,_,_,
          description([
          "The pivot of the scanning mechanism is unstable"])) .
node_explanation(mechanism_contamination,g_type(1),v_depth(2),_,_,
          _,description([
          "The scanning mechanism is contamminated by particles"])
          ) .
node_explanation(motor_fails,g_type(1),v_depth(2),_,_,_,description
          (["The motor of the scanning mechanism fails"])) .
node_explanation(motor_overheats,g_type(1),v_depth(2),_,_,_,
          description([
          "The motor of the scanning mechanism overheats"])) .
node_explanation(control_electronics_fails,g_type(1),v_depth(2),_,
          _,_,description(["The control electronics of the scanni$
          $ng mechanism fails"])) .
node_explanation(emi_to_electronics,g_type(1),v_depth(2),_,_,_,
          description([
          "The electro-magnetic interference(EMI) causes",
          " malfunction of the electronics"])) .
node_explanation(excessive_nesa_a_power_cycling,g_type(1),v_depth(
          2),_,_,_,description([
          "NESA-A has power-cycled excessively"])) .
node_explanation(sun_position_always_changes,g_type(1),v_depth(2),
          _,_,_,description([
          "The relative position of the sun to the spacecraft",
          " always changes"])) .
node_explanation(anomalies_relate_to_sun_pos,g_type(1),v_depth(2),
          _,_,_,description([
          "There is a correlation between the position of the sun"
          ," to the spacecraft and the occurances of on-board ano$
          $malies"])) .
node_explanation(power_cut_to_eliminate_output,g_type(1),v_depth(2
          ),_,_,_,description([
          "In order to eliminate the output from NESA,",
          " the power to the unit has to be cut"])) .
```

```
node_explanation(nesa_a_output_must_be_cut_out,g_type(1),v_depth(2
        ),_,_,_,description(["Ther is a situation in which the $
        $output of NESA-A must be",," eliminated"])) .

node_explanation(power_needs_to_be_cut_to_eliminate_output,g_type(
        1),v_depth(2),_,_,_,description([
        "This explanation does not defined yet"])) .

node_type(electronics_innert,fault,dependency(3),p1(1),p0(0),
        threshold(0),error_rate(12),_330,_331,_332) .
node_type(antenna_ineffective,fault,dependency(3),p1(1),p0(0),
        threshold(0),error_rate(12),_338,_339,_340) .
node_type(spacecraft_mechanically_frozen,fault,dependency(0),p1(1)
        ,p0(0),threshold(0),error_rate(12),_354,_355,_356) .
node_type(heaters_ineffective,fault,dependency(0),p1(1),p0(0),
        threshold(0),error_rate(12),_362,_363,_364) .
node_type(spacecraft_tumbles,fault,dependency(8),p1(1),p0(0),
        threshold(0),error_rate(12),_370,_371,_372) .
node_type(wheel_stops,fault,dependency(0),p1(1),p0(0),threshold(0)
        ,error_rate(12),_378,_379,_380) .
node_type(electrical_shutdown,fault,dependency(0),p1(1),p0(0),
        threshold(0),error_rate(12),_386,_387,_388) .
node_type(uvs_trips,fault,dependency(7),p1(1),p0(0),threshold(0),
        error_rate(12),_394,_395,_396) .
node_type(batteries_exhausted,fault,dependency(5),p1(1),p0(0),
        threshold(0),error_rate(12),_402,_403,_404) .
node_type(charging_limited,fault,dependency(0),p1(1),p0(0),
        threshold(0),error_rate(12),_410,_411,_412) .
node_type(power_loss_1,fault,dependency(5),p1(1),p0(0),threshold(0
        ),error_rate(12),_418,_419,_420) .
node_type(power_loss_2,fault,dependency(5),p1(1),p0(0),threshold(0
        ),error_rate(12),_426,_427,_428) .
node_type(catalyst_bed_heater_on,fault,dependency(0),p1(1),p0(0),
        threshold(0),error_rate(12),_434,_435,_436) .
node_type(heavy_tracking_power,fault,dependency(10),p1(1),p0(0),
        threshold(0),error_rate(12),_442,_443,_444) .
node_type(tracking_partially_successful,fault,dependency(8),p1(1),
        p0(0),threshold(0),error_rate(12),_450,_451,_452) .
node_type(continuous_tracking,fault,dependency(0),p1(1),p0(0),
        threshold(0),error_rate(12),_458,_459,_460) .
node_type(attitude_control_lost,fault,dependency(7),p1(1),p0(0),
        threshold(0),error_rate(12),_466,_467,_468) .
node_type(command_not_receivable,fault,dependency(8),p1(1),p0(0),
        threshold(0),error_rate(12),_474,_475,_476) .
node_type(o4_firing_stops,fault,dependency(10),p1(1),p0(0),
        threshold(0),error_rate(12),_482,_483,_484) .
node_type(solar_array_off_angle,fault,dependency(6),p1(1),p0(0),
        threshold(0),error_rate(12),_490,_491,_492) .
node_type(receive_antenna_off_angle,fault,dependency(0),p1(1),p0(0
        ),threshold(0),error_rate(12),_498,_499,_500) .
node_type(shf_lost,fault,dependency(4),p1(1),p0(0),threshold(0),
        error_rate(12),_506,_507,_508) .
node_type(fuel_in_line_becomes_scarce,fault,dependency(10),p1(1),p0
        (0),threshold(0),error_rate(12),_514,_515,_516) .
node_type(pitch_changes_greatly,fault,dependency(8),p1(1),p0(0),
        threshold(0),error_rate(12),_522,_523,_524) .
node_type(large_cone_develops,fault,dependency(10),p1(1),p0(0),
        threshold(0),error_rate(12),_530,_531,_532) .
node_type(o4_firing_continues,fault,dependency(5),p1(1),p0(0),
        threshold(0),error_rate(12),_538,_539,_540) .
node_type(high_rate_command_continues,fault,dependency(10),p1(1),p0
        (0),threshold(0),error_rate(12),_546,_547,_548) .
node_type(pressure_in_fuel_line_maintains,fault,dependency(9),p1(1
        ),p0(0),threshold(0),error_rate(12),_554,_555,_556) .
node_type(additional_fuel_vaporizes,fault,dependency(1),p1(1),p0(0
        ),threshold(0),error_rate(12),_562,_563,_564) .
node_type(fuel_pressure_drops,fault,dependency(0),p1(1),p0(0)
```

```
node_type(o4_fires, fault, dependency(0), p1(1), p0(0), threshold(0),
          error_rate(12), _578, _579, _580) .
node_type(roll_yaw_command_issued, fault, dependency(10), p1(1), p0(0)
          , threshold(0), error_rate(12), _586, _587, _588) .
node_type(nesa_a_takes_over_roll_yaw_control, fault, dependency(10),
          _594, _595, threshold(0), error_rate(12), _596, _597, _598) .
node_type(fuel_control_inaccurate, fault, dependency(0), p1(1), p0(0),
          threshold(0), error_rate(12), _604, _605, _606) .
node_type(nesa_b_loses_earth_presence, fault, dependency(0), p1(1), p0
          (0), threshold(0), error_rate(12), _612, _613, _614) .
node_type(multi_face_flow_in_fuel_line, fault, dependency(10), p1(1),
          p0(0), threshold(0), error_rate(12), _620, _621, _622) .
node_type(negative_pitch_develops, fault, dependency(0), p1(1), p0(0),
          threshold(0), error_rate(12), _628, _629, _630) .
node_type(multi_face_flow_potential_in_tank, fault, dependency(3),
          _636, _637, threshold(0), error_rate(12), _638, _639, _640) .
node_type(limited_fuel_in_fuel_line, fault, dependency(8), p1(1), p0(0
          ), threshold(0), error_rate(12), _646, _647, _648) .
node_type(wheel_speed_drops, fault, dependency(0), p1(1), p0(0),
          threshold(0), error_rate(12), _654, _655, _656) .
node_type(nitrogen_in_hydrazine, fault, dependency(-4), p1(1), p0(0),
          threshold(0), error_rate(12), _662, _663, _664) .
node_type(unresolved_nitrogen_in_tank, fault, dependency(6), p1(1), p0
          (0), threshold(0), error_rate(12), _670, _671, _672) .
node_type(unspecified_gas_in_tank, fault, dependency(10), p1(1), p0(0)
          , threshold(0), error_rate(12), _678, _679, _680) .
node_type(main_tank_valve_closes, fault, dependency(0), p1(1), p0(0),
          threshold(0), error_rate(12), _686, _687, _688) .
node_type(switch_to_redundant_ace_and_mwc, fault, dependency(0), p1(1
          ), p0(0), threshold(0), error_rate(12), _694, _695, _696) .
node_type(cws_mode_on, fault, dependency(0), p1(1), p0(0), threshold(0)
          , error_rate(12), _702, _703, _704) .
node_type(nitrogen_thru_diaphragm, fault, dependency(8), p1(1), p0(0),
          threshold(0), error_rate(12), _718, _719, _720) .
node_type(recovery_procedure_begins, fault, dependency(0), p1(1), p0(0
          ), threshold(0), error_rate(12), _726, _727, _728) .
node_type(afp_trips, fault, dependency(0), p1(1), p0(0), threshold(0),
          error_rate(12), _734, _735, _736) .
node_type(nesa_a_has_earth_presence, fault, dependency(0), p1(1), p0(0
          ), threshold(0), error_rate(12), _742, _743, _744) .
node_type(tank_pressure_low, fault, dependency(0), p1(1), p0(0),
          threshold(0), error_rate(12), _758, _759, _760) .
node_type(fuel_tank_temp_cycles, fault, dependency(10), p1(1), p0(0),
          threshold(0), error_rate(12), _774, _775, _776) .
node_type(nesa_a_output_saturates, fault, dependency(-5), p1(1), p0(0)
          , threshold(0), error_rate(12), _790, _791, _792) .
node_type(nesa_a_saturation_1, fault, dependency(-5), p1(1), p0(0),
          threshold(0), error_rate(12), _814, _815, _816) .
node_type(nesa_a_saturation_2, fault, dependency(0), p1(1), p0(0),
          threshold(0), error_rate(12), _822, _823, _824) .
node_type(charged_energy, fault, dependency(-8), p1(1), p0(0), threshold
          (0), error_rate(12), _830, _831, _832) .
node_type(mirror_stuck, fault, dependency(-5), p1(1), p0(0), threshold(
          0), error_rate(12), _838, _839, _840) .
node_type(and_electronics, fault, dependency(5), p1(1), p0(0), threshold
          (0), error_rate(12), _846, _847, _848) .
node_type(or_electronics, fault, dependency(5), p1(1), p0(0), threshold
          (0), error_rate(12), _854, _855, _856) .
node_type(scan_mechanism_fails, fault, dependency(7), p1(1), p0(0),
          threshold(0), error_rate(12), _862, _863, _864) .
node_type(scan_motor_fails, fault, dependency(8), p1(1), p0(0),
          threshold(0), error_rate(12), _870, _871, _872) .
node_type(thermal_distortion, fault, dependency(8), p1(1), p0(0),
          threshold(0), error_rate(12), _896, _897, _898) .
node_type(excessive_nesa_a_power_cycling, fault, dependency(10), p1(1
```

```
node_type(spacecraft_lost, fault, dependency(8), p1(1), p0(0), threshold
          (0), error_rate(50), _834, _835, _836).
node_type(nitrogen_to_pressure, fault, dependency(undefined), p1(1), p0
          (0), threshold(0), error_rate(999999), _866, _867, _868) .
node_type(impurities_in_tank, fault, dependency(9), p1(1), p0(0),
          threshold(0), error_rate(35000), _874, _875, _876) .
node_type(fuel_in_tank_low, fault, dependency(0), p1(1), p0(0),
          threshold(0), error_rate(300000), _882, _883, _884) .
node_type(heat_dissipation_uneven, fault, dependency(0), p1(1), p0(0),
          threshold(0), error_rate(745000), _890, _891, _892) .
node_type(sun_reflections, fault, dependency(_898), p1(1), p0(0),
          threshold(0), error_rate(29600), _899, _900, _901) .
node_type(shf_radiation, fault, dependency(_907), p1(1), p0(0),
          threshold(0), error_rate(425), _908, _909, _910) .
node_type(unstable_pivot, fault, dependency(_916), p1(1), p0(0),
          threshold(0), error_rate(25000), _917, _918, _919) .
node_type(mechanism_contamination, fault, dependency(_925), p1(1), p0(
          0), threshold(0), error_rate(2520), _926, _927, _928) .
node_type(motor_fails, fault, dependency(_934), p1(1), p0(0), threshold
          (0), error_rate(173000), _935, _936, _937) .
node_type(motor_overheats, fault, dependency(_943), p1(1), p0(0),
          threshold(0), error_rate(2950), _944, _945, _946) .
node_type(control_electronics_fails, fault, dependency(_952), p1(1), p0
          (0), threshold(0), error_rate(465000), _953, _954, _955) .
node_type(emi_to_electronics, fault, dependency(_961), p1(1), p0(0),
          threshold(0), error_rate(1500), _962, _963, _964) .
node_type(power_needs_to_be_cut_to_eliminate_output, fault,
          dependency(_970), p1(1), p0(0), threshold(0), error_rate(
          920000), _971, _972, _973) .
node_type(sun_position_always_changes, fault, dependency(_979), p1(1)
          , p0(0), threshold(0), error_rate(914000), _980, _981, _982) .
node_type(anomalies_relate_to_sun_pos, fault, dependency(_988), p1(1)
          , p0(0), threshold(0), error_rate(150000), _989, _990, _991) .
node_type(power_cut_to_eliminate_output, fault, dependency(_997), p1(
          1), p0(0), threshold(0), error_rate(12), _998, _999, _1000) .
node_type(nesa_a_output_must_be_cut_out, fault, dependency(_1006), p1
          (1), p0(0), threshold(0), error_rate(800000), _1007, _1008,
          _1009) .
node_type(telemetry_lost, fault, dependency(0), p1(1), p0(0), threshold
          (0), error_rate(500000), _991, _992, _993) .
node_type(o4_previously_fired, fault, dependency(undefined), p1(1), p0
          (0), threshold(0), error_rate(975000), _999, _1000, _1001) .
node_type(diaphragm_leaks, fault, dependency(undefined), p1(1), p0(0),
          threshold(0), error_rate(755000), _1007, _1008, _1009) .

node_action(spacecraft_lost, _, _, _, _, action([(a, advice, 8, _, _, msg1)]
          )) .
node_action(electronics_innert, _, _, _, _, action([(a, advice, 3, _, _,
          msg2)])) .
node_action(antenna_ineffective, _, _, _, _, action([(a, advice, 3, _, _,
          msg3)])) .
node_action(telemetry_lost, _, _, _, _, action([(a, advice, 0, _, _, msg4)])
          ) .
node_action(spacecraft_mechanically_frozen, _, _, _, _, action([(a,
          advice, 0, _, _, msg5)])) .
node_action(heaters_ineffective, _, _, _, _, action([(a, advice, 0, _, _,
          msg6)])) .
node_action(spacecraft_tumbles, _, _, _, _, action([(a, advice, 8, _, _,
          msg7)])) .
node_action(wheel_stops, _, _, _, _, action([(a, advice, 0, _, _, msg8)])) .
node_action(electrical_shutdown, _, _, _, _, action([(a, advice, 0, _, _,
          msg9)])) .
node_action(uvs_trips, _, _, _, _, action([(a, warning, 1, _, _, msg10), (r,
          recommendation, 9, _, _, uvs_trips_rec), (er, explanation, 9, _,
          _, uvs_trips_exp)])) .
```

```
        msg11)])) .
node_action(charging_limited,_,_,_,_,action([(a,advice,0,_,_,msg12
        )])) .
node_action(power_loss_1,_,_,_,_,action([(a,advice,5,_,_,msg13)])) .
node_action(power_loss_2,_,_,_,_,action([(a,advice,5,_,_,msg14)])) .
node_action(catalyst_bed_heater_on,_,_,_,_,action([(a,advice,0,_,_
        ,msg15)])) .
node_action(heavy_tracking_power,_,_,_,_,action([(a,advice,10,_,_,
        msg16)])) .
node_action(tracking_partially_successful,_,_,_,_,action([(a,
        advice,8,_,_,msg17)])) .
node_action(continuous_tracking,_,_,_,_,action([(a,advice,0,_,_,
        msg18)])) .
node_action(attitude_control_lost,_,_,_,_,action([(a,advice,7,_,_,
        msg19)])) .
node_action(command_not_receivable,_,_,_,_,action([(a,advice,8,_,_
        ,msg20)])) .
node_action(o4_firing_stops,_,_,_,_,action([(a,advice,10,_,_,msg21
        )])) .
node_action(solar_array_off_angle,_,_,_,_,action([(a,advice,6,_,_,
        msg22)])) .
node_action(receive_antenna_off_angle,_,_,_,_,action([(a,advice,6,
        _,_,msg23)])) .
node_action(snf_lost,_,_,_,_,action([(a,advice,4,_,_,msg24)])) .
node_action(fuel_in_line_becomes_scarce,_,_,_,_,action([(a,advice,
        10,_,_,msg25)])) .
node_action(pitch_changes_greatly,_,_,_,_,action([(a,advice,8,_,_,
        msg26)])) .
node_action(large_cone_develops,_,_,_,_,action([(a,advice,10,_,_,
        msg27)])) .
node_action(o4_firing_continues,_,_,_,_,action([(a,advice,5,_,_,
        msg28)])) .
node_action(high_rate_command_continues,_,_,_,_,action([(a,advice,
        10,_,_,msg29)])) .
node_action(pressure_in_fuel_line_maintains,_,_,_,_,action([(a,
        advice,9,_,_,msg30)])) .
node_action(additional_fuel_vaporizes,_,_,_,_,action([(a,advice,1,
        _,_,msg31)])) .
node_action(fuel_pressure_drops,_,_,_,_,action([(a,advice,0,_,_,
        msg32)])) .
node_action(o4_fires,_,_,_,_,action([(a,advice,0,_,_,msg33)])) .
node_action(roll_yaw_command_issued,_,_,_,_,action([(a,advice,10,_
        ,_,msg34)])) .
node_action(nesa_a_takes_over_roll_yaw_control,_,_,_,_,action([(a,
        advice,10,_,_,msg35)])) .
node_action(fuel_control_inaccurate,_,_,_,_,action([(a,advice,120,
        _,_,msg36)])) .
node_action(nesa_b_loses_earth_presence,_,_,_,_,action([(a,advice,
        0,_,_,msg37)])) .
node_action(multi_face_flow_in_fuel_line,_,_,_,_,action([(a,advice
        ,10,_,_,msg38)])) .
node_action(negative_pitch_develops,_,_,_,_,action([(a,advice,0,_,
        _,msg39)])) .
node_action(multi_face_flow_potential_in_tank,_,_,_,_,action([(a,
        advice,3,_,_,msg40)])) .
node_action(limited_fuel_in_fuel_line,_,_,_,_,action([(a,advice,8,
        _,_,msg41)])) .
node_action(wheel_speed_drops,_,_,_,_,action([(a,advice,0,_,_,
        msg42)])) .
node_action(nitrogen_in_hydrazine,_,_,_,_,action([(a,advice,-4,_,_
        ,msg43)])) .
node_action(unresolved_nitrogen_in_tank,_,_,_,_,action([(a,advice,
        0,_,_,msg44)])) .
node_action(unspecified_gas_in_tank,_,_,_,_,action([(a,advice,0,_,
        _,msg45)])) .
```

```
          ,msg46)]) .
node_action(switch_to_redundant_ace_and_mwc,_,_,_,_,action([(a,
          advice,0,_,_,msg47)])) .
node_action(cws_mode_on,_,_,_,_,action([(a,advice,0,_,_,msg48)])) .
node_action(o4_previously_fired,_,_,_,_,action([(a,advice,
          undefined_,_,msg49)])) .
node_action(nitrogen_thru_diaphragm,_,_,_,_,action([(a,advice,8,_,
          _,msg50)])) .
node_action(recovery_procedure_begins,_,_,_,_,action([(a,advice,0,
          _,_,msg51)])) .
node_action(afd_trips,_,_,_,_,action([(a,advice,0,_,_,msg87)])) .
node_action(nesa_a_has_earth_presence,_,_,_,_,action([(a,advice,0,
          _,_,msg52)])) .
node_action(diaphragm_leaks,_,_,_,_,action([(a,advice,undefined_,_
          ,msg53)])) .
node_action(nitrogen_to_pressure,_,_,_,_,action([(a,advice,
          undefined_,_,msg54)])) .
node_action(fuel_tank_temp_cycles,_,_,_,_,action([(a,advice,0,_,_,
          msg55)])) .
node_action(impurities_in_tank,_,_,_,_,action([(a,advice,0,_,_,
          msg56)])) .
node_action(nesa_a_output_saturates,_,_,_,_,action([(a,advice,-5,_
          ,_,msg57)])) .
node_action(tank_pressure_low,_,_,_,_,action([(a,advice,11,_,_,
          msg58)])) .
node_action(fuel_in_tank_low,_,_,_,_,action([(a,advice,1,_,_,msg59
          )])) .
node_action(heat_dissipation_uneven,_,_,_,_,action([(a,advice,1,_,
          _,msg60)])) .
node_action(nesa_a_saturation_1,_,_,_,_,action([(a,advice,-5,_,_,
          msg61)])) .
node_action(nesa_a_saturation_2,_,_,_,_,action([(a,advice,0,_,_,
          msg62)])) .
node_action(charged_energy,_,_,_,_,action([(a,advice,-8,_,_,msg63)
          ])) .
node_action(mirror_stuck,_,_,_,_,action([(a,advice,-5,_,_,msg64)])
          ) .
node_action(and_electronics,_,_,_,_,action([(a,advice,5,_,_,msg65)
          ])) .
node_action(or_electronics,_,_,_,_,action([(a,advice,5,_,_,msg66)]
          )) .
node_action(scan_mechanism_fails,_,_,_,_,action([(a,advice,7,_,_,
          msg67)])) .
node_action(scan_motor_fails,_,_,_,_,action([(a,advice,8,_,_,msg68
          )])) .
node_action(sun_reflections,_,_,_,_,action([(a,advice,3,_,_,msg69)
          ])) .
node_action(shf_radiation,_,_,_,_,action([(a,advice,3,_,_,msg70)])
          ) .
node_action(thermal_distortion,_,_,_,_,action([(a,advice,8,_,_,
          msg71)])) .
node_action(unstable_pivot,_,_,_,_,action([(a,advice,3,_,_,msg72)]
          )) .
node_action(mechanism_contamination,_,_,_,_,action([(a,advice,3,_,
          _,msg73)])) .
node_action(motor_fails,_,_,_,_,action([(a,advice,3,_,_,msg78)])) .
node_action(motor_overheats,_,_,_,_,action([(a,advice,3,_,_,msg79)
          ])) .
node_action(control_electronics_fails,_,_,_,_,action([(a,advice,3,
          _,_,msg80)])) .
node_action(emi_to_electronics,_,_,_,_,action([(a,advice,3,_,_,
          msg81)])) .
node_action(excessive_nesa_a_power_cycling,_,_,_,_,action([(a,
          acvice,10,_,_,msg82)])) .
node_action(sun_position_always_changes,_,_,_,_,action([(a,acvice,
```

```
node_action(anomalies_relate_to_sun_pos,_,_,_,_,action([(a,advice,
        3,_,_,msg84)])) .
node_action(power_cut_to_eliminate_output,_,_,_,_,action([(a,
        advice,3,_,_,msg85)])) .
node_action(nesa_a_output_must_be_cut_out,_,_,_,_,action([(a,
        advice,3,_,_,msg86)])) .
node_action(power_needs_to_be_cut_to_eliminate_output,_,_,_,_,
        action([(a,advice,3,_,_,msg88)])) .
```

```
$ ty comdb.log
/*                  */
/* Common Data Base */
/*                  */
node_control(unresolved_nitrogen_in_tank,no_command,relieve,
        relieve,r,_326,_327) .
node_control(unspecified_gas_in_tank,no_command,relieve,relieve,r,
        _333,_334) .
node_control(antenna_ineffective,no_command,relieve,relieve,r,_340
        ,_341) .
node_control(electronics_innert,no_command,relieve,relieve,r,_347,
        _348) .
node_control(spacecraft_mechanically_frozen,no_command,relieve,
        relieve,r,_354,_355) .
node_control(charged_energy,no_command,relieve,relieve,r,_361,_362
        ) .
node_control(heaters_ineffective,no_command,relieve,relieve,r,_368
        ,_369) .
node_control(spacecraft_tumbles,no_command,relieve,relieve,r,_375,
        _376) .
node_control(wheel_stops,no_command,relieve,relieve,r,_382,_383) .
node_control(electrical_shutdown,no_command,relieve,relieve,r,_389
        ,_390) .
node_control(batteries_exhausted,no_command,relieve,relieve,r,_396
        ,_397) .
node_control(charging_limited,no_command,relieve,relieve,r,_403,
        _404) .
node_control(power_loss_1,no_command,relieve,relieve,r,_410,_411) .
node_control(power_loss_2,no_command,relieve,relieve,r,_417,_418) .
node_control(catalyst_bed_heater_on,no_command,relieve,relieve,r,
        _424,_425) .
node_control(heavy_tracking_power,no_command,relieve,relieve,r,
        _431,_432) .
node_control(tracking_partially_successful,no_command,relieve,
        relieve,r,_438,_439) .
node_control(continuous_tracking,no_command,relieve,relieve,r,_445
        ,_446) .
node_control(attitude_control_lost,no_command,relieve,relieve,r,
        _452,_453) .
node_control(command_not_receivable,no_command,relieve,relieve,r,
        _459,_460) .
node_control(o4_firing_stops,no_command,relieve,relieve,r,_466,
        _467) .
node_control(solar_array_off_angle,no_command,relieve,relieve,r,
        _473,_474) .
node_control(receive_antenna_off_angle,no_command,relieve,relieve,
        r,_480,_481) .
node_control(shf_lost,no_command,relieve,relieve,r,_487,_488) .
node_control(fuel_in_line_becomes_scarce,no_command,relieve,
        relieve,r,_494,_495) .
node_control(pitch_changes_greatly,no_command,relieve,relieve,r,
        _501,_502) .
node_control(large_cone_develops,no_command,relieve,relieve,r,_508
        ,_509) .
node_control(o4_firing_continues,no_command,relieve,relieve,r,_515
        ,_516) .
node_control(high_rate_command_continues,no_command,relieve,
        relieve,r,_522,_523) .
node_control(pressure_in_fuel_line_maintains,no_command,relieve,
        relieve,r,_529,_530) .
node_control(additional_fuel_vaporizes,no_command,relieve,relieve,
        r,_536,_537) .
node_control(fuel_pressure_droos,no_command,relieve,relieve,r,_543
        ,_544) .
node_control(o4_fires,no_command,relieve,relieve,r,_550,_551) .
```

```
node_control(roll_yaw_command_issued, no_command, relieve, relieve, r,
        _557, _558) .
node_control(nesa_a_takes_over_roll_yaw_control, no_command, relieve
        , relieve, r, _564, _565) .
node_control(fuel_control_inaccurate, no_command, relieve, relieve, r,
        _571, _572) .
node_control(nesa_b_loses_earth_presence, no_command, relieve,
        relieve, r, _578, _579) .
node_control(multi_face_flow_in_fuel_line, no_command, relieve,
        relieve, r, _585, _586) .
node_control(negative_pitch_develops, no_command, relieve, relieve, r,
        _592, _593) .
node_control(multi_face_flow_potential_in_tank, no_command, relieve,
        relieve, r, _599, _600) .
node_control(limited_fuel_in_fuel_line, no_command, relieve, relieve,
        r, _606, _607) .
node_control(wheel_speed_drops, no_command, relieve, relieve, r, _613,
        _614) .
node_control(main_tank_valve_closes, no_command, relieve, relieve, r,
        _620, _621) .
node_control(switch_to_redundant_ace_and_mwc, no_command, relieve,
        relieve, r, _627, _628) .
node_control(cws_mode_on, no_command, relieve, relieve, r, _634, _635) .
node_control(nitrogen_thru_diaphragm, no_command, relieve, relieve, r,
        _641, _642) .
node_control(recovery_procedure_begins, no_command, relieve, relieve,
        r, _648, _649) .
node_control(afp_trips, no_command, relieve, relieve, r, _655, _656) .
node_control(nesa_a_has_earth_presence, no_command, relieve, relieve,
        r, _662, _663) .
node_control(fuel_tank_temp_cycles, no_command, relieve, relieve, r,
        _669, _670) .
node_control(nesa_a_output_saturates, no_command, relieve, relieve, r,
        _676, _677) .
node_control(nesa_a_saturation_1, no_command, relieve, relieve, r, _683
        , _684) .
node_control(nesa_a_saturation_2, no_command, relieve, relieve, r, _690
        , _691) .
node_control(mirror_stuck, no_command, relieve, relieve, r, _697, _698) .
node_control(and_electronics, no_command, relieve, relieve, r, _704,
        _705) .
node_control(or_electronics, no_command, relieve, relieve, r, _711, _712
        ) .
node_control(scan_mechanism_fails, no_command, relieve, relieve, r,
        _718, _719) .
node_control(scan_motor_fails, no_command, relieve, relieve, r, _725,
        _726) .
node_control(thermal_distortion, no_command, relieve, relieve, r, _732,
        _733) .
node_control(excessive_nesa_a_power_cycling, no_command, relieve,
        relieve, r, _739, _740) .
node_control(tank_pressure_low, no_command, relieve, relieve, r, _746,
        _747) .
node_control(nitrogen_in_hydrazine, no_command, relieve, relieve, r,
        _753, _754) .
node_control(spacecraft_lost, no_command, relieve, relieve, r, _760,
        _761) .
node_control(telemetry_lost, no_command, relieve, relieve, r, _767, _768
        ) .
node_control(o4_previously_fired, no_command, relieve, relieve, r, _774
        , _775) .
node_control(diaphragm_leaks, no_command, relieve, relieve, r, _781,
        _782) .
node_control(nitrogen_to_pressure, no_command, relieve, relieve, r,
        _788, _789) .
node_control(impurities_in_tank, no_command, relieve, relieve, r, _795,
```

```
        _796) .
node_control(fuel_in_tank_low, no_command, relieve, relieve, r, _802,
        _803) .
node_control(heat_dissipation_uneven, no_command, relieve, relieve, r,
        _809, _810) .
node_control(sun_reflections, no_command, relieve, relieve, r, _816,
        _817) .
node_control(shf_radiation, no_command, relieve, relieve, r, _823, _824) .
node_control(unstable_pivot, no_command, relieve, relieve, r, _830, _831
        ) .
node_control(mechanism_contamination, no_command, relieve, relieve, r,
        _837, _838) .
node_control(motor_fails, no_command, relieve, relieve, r, _844, _845) .
node_control(motor_overheats, no_command, relieve, relieve, r, _851,
        _852) .
node_control(control_electronics_fails, no_command, relieve, relieve,
        r, _858, _859) .
node_control(emi_to_electronics, no_command, relieve, relieve, r, _865,
        _866) .
node_control(power_needs_to_be_cut_to_eliminate_output, no_command,
        relieve, relieve, r, _872, _873) .
node_control(sun_position_always_changes, no_command, relieve,
        relieve, r, _879, _880) .
node_control(anomalies_relate_to_sun_pos, no_command, relieve,
        relieve, r, _886, _887) .
node_control(power_cut_to_eliminate_output, no_command, relieve,
        relieve, r, _893, _894) .
node_control(nesa_a_output_must_be_cut_out, no_command, relieve,
        relieve, r, _900, _901) .
node_control(uvs_trips, no_command, relieve, entrust, r, _307, _308) .

node_status(excessive_nesa_a_power_cycling, connected, active, _326,
        _327, _328, unchanged, breadth, relieved, w) .
node_status(thermal_distortion, connected, active, _334, _335, _336,
        unchanged, breadth, relieved, w) .
node_status(scan_motor_fails, connected, active, _342, _343, _344,
        unchanged, breadth, relieved, w) .
node_status(scan_mechanism_fails, connected, active, _350, _351, _352,
        unchanged, breadth, relieved, w) .
node_status(or_electronics, connected, active, _358, _359, _360,
        unchanged, breadth, relieved, w) .
node_status(and_electronics, connected, active, _366, _367, _368,
        unchanged, breadth, relieved, w) .
node_status(mirror_stuck, connected, active, _374, _375, _376, unchanged
        , breadth, relieved, w) .
node_status(charged_energy, connected, active, _382, _383, _384,
        unchanged, breadth, relieved, w) .
node_status(nesa_a_saturation_2, connected, active, _390, _391, _392,
        unchanged, breadth, relieved, w) .
node_status(nesa_a_saturation_1, connected, active, _398, _399, _400,
        unchanged, breadth, relieved, w) .
node_status(nesa_a_output_saturates, connected, active, _406, _407,
        _408, unchanged, breadth, relieved, w) .
node_status(fuel_tank_temp_cycles, connected, active, _414, _415, _416,
        unchanged, breadth, relieved, w) .
node_status(tank_pressure_low, connected, active, _422, _423, _424,
        unchanged, breadth, relieved, w) .
node_status(nesa_a_has_earth_presence, connected, active, _430, _431,
        _432, unchanged, breadth, relieved, w) .
node_status(afp_trips, connected, active, _438, _439, _440, unchanged,
        breadth, relieved, w) .
node_status(recovery_procedure_begins, connected, active, _446, _447,
        _448, unchanged, breadth, relieved, w) .
node_status(nitrogen_thru_diaphragm, connected, active, _454, _455,
        _456, unchanged, breadth, relieved, w) .
```

```
breadth, relieved, w) .
node_status(switch_to_redundant_ace_and_mwc, connected, active, _470,
        _471, _472, unchanged, breadth, relieved, w) .
node_status(main_tank_valve_closes, connected, active, _478, _479, _480
        , unchanged, breadth, relieved, w) .
node_status(unspecified_gas_in_tank, connected, active, _486, _487,
        _488, unchanged, breadth, relieved, w) .
node_status(unresolved_nitrogen_in_tank, connected, active, _494, _495
        , _496, unchanged, breadth, relieved, w) .
node_status(nitrogen_in_hydrazine, connected, active, _502, _503, _504,
        unchanged, breadth, relieved, w) .
node_status(wheel_speed_drops, connected, active, _510, _511, _512,
        unchanged, breadth, relieved, w) .
node_status(limited_fuel_in_fuel_line, connected, active, _518, _519,
        _520, unchanged, breadth, relieved, w) .
node_status(multi_face_flow_potential_in_tank, connected, active,
        _526, _527, _528, unchanged, breadth, relieved, w) .
node_status(negative_pitch_develops, connected, active, _534, _535,
        _536, unchanged, breadth, relieved, w) .
node_status(multi_face_flow_in_fuel_line, connected, active, _542,
        _543, _544, unchanged, breadth, relieved, w) .
node_status(nesa_b_loses_earth_presence, connected, active, _550, _551
        , _552, unchanged, breadth, relieved, w) .
node_status(fuel_control_inaccurate, connected, active, _558, _559,
        _560, unchanged, breadth, relieved, w) .
node_status(nesa_a_takes_over_roll_yaw_control, connected, active,
        _566, _567, _568, unchanged, breadth, relieved, w) .
node_status(roll_yaw_command_issued, connected, active, _574, _575,
        _576, unchanged, breadth, relieved, w) .
node_status(o4_fires, connected, active, _582, _583, _584, unchanged,
        breadth, relieved, w) .
node_status(fuel_pressure_drops, connected, active, _590, _591, _592,
        unchanged, breadth, relieved, w) .
node_status(additional_fuel_vaporizes, connected, active, _598, _599,
        _600, unchanged, breadth, relieved, w) .
node_status(pressure_in_fuel_line_maintains, connected, active, _606,
        _607, _608, unchanged, breadth, relieved, w) .
node_status(high_rate_command_continues, connected, active, _614, _615
        , _616, unchanged, breadth, relieved, w) .
node_status(o4_firing_continues, connected, active, _622, _623, _624,
        unchanged, breadth, relieved, w) .
node_status(large_cone_develops, connected, active, _630, _631, _632,
        unchanged, breadth, relieved, w) .
node_status(pitch_changes_greatly, connected, active, _638, _639, _640,
        unchanged, breadth, relieved, w) .
node_status(fuel_in_line_becomes_scarce, connected, active, _646, _647
        , _648, unchanged, breadth, relieved, w) .
node_status(shf_lost, connected, active, _654, _655, _656, unchanged,
        breadth, relieved, w) .
node_status(receive_antenna_off_angle, connected, active, _662, _663,
        _664, unchanged, breadth, relieved, w) .
node_status(solar_array_off_angle, connected, active, _670, _671, _672,
        unchanged, breadth, relieved, w) .
node_status(o4_firing_stops, connected, active, _678, _679, _680,
        unchanged, breadth, relieved, w) .
node_status(command_not_receivable, connected, active, _686, _687, _688
        , unchanged, breadth, relieved, w) .
node_status(attitude_control_lost, connected, active, _694, _695, _696,
        unchanged, breadth, relieved, w) .
node_status(continuous_tracking, connected, active, _702, _703, _704,
        unchanged, breadth, relieved, w) .
node_status(tracking_partially_successful, connected, active, _710,
        _711, _712, unchanged, breadth, relieved, w) .
node_status(heavy_tracking_power, connected, active, _718, _719, _720,
        unchanged, breadth, relieved, w) .
```

```
          , unchanged, breadth, relieved, w) .
node_status(power_loss_2, connected, active, _734, _735, _736, unchanged
          , breadth, relieved, w) .
node_status(power_loss_1, connected, active, _742, _743, _744, unchanged
          , breadth, relieved, w) .
node_status(charging_limited, connected, active, _750, _751, _752,
          unchanged, breadth, relieved, w) .
node_status(batteries_exhausted, connected, active, _758, _759, _760,
          unchanged, breadth, relieved, w) .
node_status(uvs_trips, connected, active, _766, _767, _768, unchanged,
          breadth, relieved, w) .
node_status(electrical_shutdown, connected, active, _774, _775, _776,
          unchanged, breadth, relieved, w) .
node_status(wheel_stops, connected, active, _782, _783, _784, unchanged,
          breadth, relieved, w) .
node_status(spacecraft_tumbles, connected, active, _790, _791, _792,
          unchanged, breadth, relieved, w) .
node_status(heaters_ineffective, connected, active, _798, _799, _800,
          unchanged, breadth, relieved, w) .
node_status(spacecraft_mechanically_frozen, connected, active, _806,
          _807, _808, unchanged, breadth, relieved, w) .
node_status(antenna_ineffective, connected, active, _814, _815, _816,
          unchanged, breadth, relieved, w) .
node_status(electronics_inert, connected, active, _822, _823, _824,
          unchanged, breadth, relieved, w) .
node_status(power_cut_to_eliminate_output, connected, active, _830,
          _831, _832, unchanged, breadth, relieved, w) .
node_status(telemetry_lost, connected, active, _838, _839, _840,
          unchanged, breadth, relieved, w) .
node_status(spacecraft_lost, connected, active, _846, _847, _848,
          unchanged, breadth, relieved, w) .
node_status(multi_face_flow_in_line, connected, active, _854, _855,
          _856, unchanged, breadth, relieved, w) .
node_status(impurities_in_tank, connected, active, _862, _863, _864,
          unchanged, breadth, relieved, w) .
node_status(o4_previously_fired, connected, active, _870, _871, _872,
          unchanged, breadth, relieved, w) .
node_status(diaphragm_leaks, connected, active, _878, _879, _880,
          unchanged, breadth, relieved, w) .
node_status(fuel_in_tank_low, connected, active, _886, _887, _888,
          unchanged, breadth, relieved, w) .
node_status(heat_dissipation_uneven, connected, active, _894, _895,
          _896, unchanged, breadth, relieved, w) .
node_status(nitrogen_to_pressure, connected, active, _902, _903, _904,
          unchanged, breadth, relieved, w) .
node_status(nesa_a_output_must_be_cut_out, connected, active, _910,
          _911, _912, unchanged, breadth, relieved, w) .
node_status(power_needs_to_be_cut_to_eliminate_output, connected,
          active, _918, _919, _920, unchanged, breadth, relieved, w) .
node_status(control_electronics_fails, connected, active, _926, _927,
          _928, unchanged, breadth, relieved, w) .
node_status(emi_to_electronics, connected, active, _934, _935, _936,
          unchanged, breadth, relieved, w) .
node_status(shf_radiation, connected, active, _942, _943, _944,
          unchanged, breadth, relieved, w) .
node_status(sun_reflections, connected, active, _950, _951, _952,
          unchanged, breadth, relieved, w) .
node_status(mechanism_contamination, connected, active, _958, _959,
          _960, unchanged, breadth, relieved, w) .
node_status(motor_fails, connected, active, _966, _967, _968, unchanged,
          breadth, relieved, w) .
node_status(motor_overheats, connected, active, _974, _975, _976,
          unchanged, breadth, relieved, w) .
node_status(unstable_pivot, connected, active, _982, _983, _984,
          unchanged, breadth, relieved, w) .
```

```
          ,_997,unchanged,breadth,relieved,W) .
node_status(sun_position_always_changes,connected,active,_998,_999
          ,_1000,unchanged,breadth,relieved,W) .

endmod /* unnamed_module */  .
```

Appendix A.4 HLKB Listings

```
$ ty hlkb.log
/*                                  */
/* High Level Knowledge Base        */
/*                                  */
hlks_action(warning,NODE) :-
     node_status(NODE,_,_,t,P,_,_,_,_,w), number(P),
       decide_true(NODE,_,P), get_action_list(NODE,ACTION_LIST),
       check_warning(ACTION_LIST) .
hlks_action(no_action,NODE) :-
     node_status(NODE,_,_,f,_,_,_,_,_,_) .
hlks_action(suspend,NODE) :-
     node_control(NODE,C,_,ENTRUST,_,_,_), ne(C,suspend),
       string(ENTRUST), equal(ENTRUST,entrust),
       node_status(NODE,_,_,t,P,_,_,_,_,_), number(P),
       decide_true(NODE,_,P), get_action_list(NODE,ACTION_LIST),
       check_warning(ACTION_LIST) .
hlks_action(beam,NODE) :-
     node_status(NODE,_,_,_,_,_,_,_,beam,_,_) .
hlks_action(breadth,NODE) :-
     node_status(NODE,_,_,_,_,_,_,breadth,_,_) .

take_hlks_action(NODE,warning) :-
     write("WARNING:  ** ("), write(NODE), write(") **"), nl, nl,
       write("HLKS Autonomy Control:  probe("), write(NODE),
       write(")."), nl, probe(NODE),
       change_node_status_for(NODE,_,_,_,_,_,_,_,_,done) .
take_hlks_action(NODE,no_action) .
take_hlks_action(NODE,suspend) :-
     suspend(NODE), write("HLKS Autonomy Control:  suspend("),
       write(NODE), write(")."), nl, write("("), write(NODE),
       write(")"), write(" is autonomously suspended by HLKS."),
       nl .
take_hlks_action(NODE,breadth) .
take_hlks_action(NODE,beam) :-
     (ask_conti_beam(NODE,R),/,equal(R,y),node_structure(NODE,_,
        evidence(E_LIST),_)), flexible_breadth_first(E_LIST), /;
     / .
```

Appendix A.5 LLKB Listings

```
$ ty llkb.log
/*                                  *
/* Low Level Knowledge Base         *
/*                                  *
llks_command(suspend) :-
     node_status(X,connected,active,_,_,_,_,_,_,_),
       node_control(X,suspend,_,_,_,_,_) .
llks_command(activate) :-
     node_control(X,resume,_,_,_,_,_) .

llks_action(suspend) :-
     suspend_llks, initialize_control(_,connect,_,_,_,_,_) .
llks_action(resume) :-
     resume_llks, initialize_control(_,resume,_,_,_,_,_) .
```

GOMI, T.
--A proof-of-concept experiment system for the spacecraft...