Final Report

Qualitative Mechanism Models

and the

Rationalization of Procedures

NASA Grant #NAG-~~323~~ 2-383

Submitted by:

Arthur M. Farley
Department of Computer and Information Science
University of Oregon
Eugene, OR 97403

Submitted on:

10 July 1989

Introduction

To conquer new frontiers, such as manned space travel, we continue to develop technologies of ever increasing sophistication. As a consequence, it is often the case that the people interacting with this technology, those actually operating or repairing complex electronic or mechanical devices, do not understand how the technology functions. They can not correctly rationalize steps of given operational and repair procedures: why manipulating or observing some aspect of a device satisfies a goal associated with a procedure or which component failures could possibly give rise to an observed abnormality and how the abnormality is caused by the failure. Unfortunately, limited understanding results in limited performance. Operators that must follow procedures by rote from checklists and make errors of omission or commission that would be obvious with an understanding of the reasons for actions. These operators often are not able to demonstrate effective flexibility in unexpected circumstances and can not troubleshoot a device when a component fails.

Qualitative mechanism models have been proposed as one form of basis for useful methods of reasoning about complex devices and rationalizing steps in operational and troubleshooting procedures. A *qualitative mechanism model* is an abstract, symbolic model of a device's structure and function. It approximates more precise mathematical models, simplifying their complex algebraic representations yet being sufficiently formal to allow straightforward reasoning about the effects of changes in device state. A complex device is modeled as a structure of interconnected components; these components may be grouped into substructures which interact in qualitatively different ways. Domains of the descriptive variables that represent aspects of a component's function are restricted to symbolic value sets of small size, determined by intervals of similar qualitative behavior and points at which the component's

behavior qualitatively changes. Constraints among variable values describe the behavior of a component for possible combinations of values for its descriptive variables (i.e., qualitiative states). A qualitative mechanism model is completed by a set of statements that describe the interconnection of device components. A change in the state of some component can be propagated to connected components when appropriate.

Changes in component state may arise from operator manipulation of device controls or from failure of particular elements of device components. Qualitative mechanism models can be used to explain why a given operational sequence will produce a desired outcome by describing the propagation of the effects of a component's state change through other connected components over time. Similarly, qualitative models of complex systems can serve as bases for predicting what abnormalities will be observed when a given failure occurs or, reasoning in the other direction, for suggesting possible failures that could account for observed abnormalities. For example, one may want to explain why stepping on the gas pedal results in the acceleration of a running car. A straightforward qualitative model involving representations of fuel line, carbuerator, drive shaft, differential and wheels as components could propogate increased fuel flow through greater rates of combustion and energy release to increased torque and rates of turning of the drive train, etc. If no acceleration is observed, the model could provide suggestions as to possible faults among those components, e.g., some blockage in the fuel line preventing the expected increase in fuel flow to the carbuerator.

When rationalizing elements of troubleshooting procedures, one desires to account for the processes whereby a possible fault could give rise to observed symptoms and to explain why replacement or adjustment of some component could be expected to alleviate a problem. Reasoning about operational plans that include interactions with complex systems requires not only representations of the goal

structure of the plans but representations of the mechanisms being manipulated. Qualitative models are approximate, indicating how changes are propagated within a complex device. Thus, they can serve to focus attention on relevant components of the device for more detailed, possibly quantitative, consideration. This focusing of activity seems to be a desirable feature for automated assistants to human operators.

NASA's manned space program affords many opportunities for exploring the applicability of computer-based, qualitative mechanism models to problems of complex system monitoring and control. Astronauts find themselves dealing with complex systems that they do not understand well; computers perform many important control functions. The potential is great for the application of qualitative models and associated inference tools as assistants to astronauts in reasoning about the function and operation of space shuttle or space station systems. Results of qualitative analyses could be presented to operators as summaries of system behaviors and explanations of procedural steps. Computer-generated rationalizations of considered actions or of possible faults could become important aspects of improved, cooperative, man-machine control systems.

As a consequence of the above discussion, this research project has focussed on the following issues:

[1] Developing an interactive environment for specifying and editing qualitative mechanism models and for simulating the behavior implied by models under various conditions.

[2] Defining a qualitative mechanism model for a particular system. We considered the ORS (Orbital Refueling System) to coordinate with ongoing research on this system being conducted at NASA Ames Research Center.

[3] Determine operational and troubleshooting procedures on the basis of the qualitative models and rationalize (explain) procedures in terms of elements of the models.

Qualitative Reasoning and Structural Abstraction

Methods for the formalization and formulation of qualitative mechanism models for electronic and mechanical systems have become topics of widespread interest within the artificial intelligence research community. This is evidenced by the recent publication of a significant number of articles discussing principles and initial applications of qualitative modeling, including a complete volume of the journal *Artificial Intelligence* [1]. Several articles in that volume constitute the primary intellectual roots from which our proposed research springs. The paper by Brown and Dekleer [2] discusses general criteria for the design and evaluation of qualitative models and presents elements for the representation of device function and structure. Confluences, i.e., simplified linear equations among derivatives of model variables, are proposed as bases for describing device function and for generating descriptions of the dynamic behavior of a system. Kuipers [10] defines a related approach to function representation, noting direct (increasing) and indirect (decreasing) relations between model variables and declaring necessary correspondences between landmark values of the variables. In both approaches one can characterize the behavior of a device as transitions among "states" defined in terms of model variables being at or moving between predefined landmark values. For example, in modeling the behavior of frozen water as heat is applied, the water temperature is initially below freezing and rises to the freezing point (a landmark, at which point behavioral properties of water change in the transition to liquid state), then the temperature rises to the boiling point (another landmark, with transition to the gaseous state) and continues beyond that. Thus, a straightforward qualitative model of this system would represent five states based upon a continuously increasing temperature passing through two landmark values.

When rationalizing elements of troubleshooting procedures, one desires to account for the processes whereby a possible fault could give rise to observed symptoms and to explain why a replacement or adjustment of some component could be expected to alleviate a problem. Davis [3] and Genesereth [6] describe related approaches to fault isolation based upon formal device design specifications. Davis demonstrates a means of diagnostic reasoning that employs hierarchic representations of device structure and function in the isolation of possible sources of incorrect behavior to more and more specific components. This approach propogates actual values at different levels of abstraction, looking for locations where incorrect values can be substituted to produce observed errors, then continuing the process within candidate substructures until primitive, repairable faults are determined. For example, in the VLSI domain a fault may be isolated to an adder structure by considering integer value propogations; the adder circuits can then be analyzed at the bit level and eventually a wire or pull-up at the voltage level to correct a faulty design or realization. Genesereth presents a propositional logic approach to the formalization of device design specifications and the defines techniques for the use of such specifications in the derivation of relevant tests during the process of diagnaostic fault isolation.

The research we have conducted coordinates well with other ongoing efforts supported by NASA. Reasoning about plans that include interactions with complex systems requires not only representations of the hierarchic goal structure of the procedure but also representations of the mechanisms being manipulated. Rationalizations as to why steps are taken (or considered) refer to tasks and their subgoals; reasons why these steps can be expected to produce the desired reults require inference that is based upon abstract mechanism models. Qualitative models are approximate, indicating how changes are propogated within a complex device. In work closely related to our effort, Scarl etal. at Mitre [15] have developed LES, a

frame-based expert system for representing structure and function of complex control systems, which is then used to perform diagnosis. Yoon and Hammer [17] propose research investigating the design of a cooperative diagnosis environment that would employ a qualitative model as knowledge base.

Another research perspective supported by NASA is that of human factors in man-machine interaction. One machine of interest is the airplane. Researchers under Rouse have been exploring the induction of pilot knowledge from behavior sequences. Their representation of knowledge is primarily rule-based, attempting to associate flying conditions with pilot actions in reactive situations involving air space intruders [11]. Much of their work has focused on the evaluation of proposed sets of rules, attempting to find small sets of rules which cover most of the observed behavior (i.e., account for observed variance in pilot behavior) [12].

Issues of man-machine cooperation range from the media used in presenting assistance to when and what is presented and who has the initiative [7]. Research indicates that people appear to work best when they are responsible for making decisions; thus, systems which can assist them by accurately focusing attention and summarizing relevant information are desirable [13]. Researchers that developed the STEAMER system [8] found that graphic displays representing system configuration and change were important in explanation and understanding.

A primary goal of our research has been to determine a means for simplifying the generation and explanation of plans for the operation and diagnosis of complex hydraulic systems. The traditional techniques of qualitative mechanism modeling described above have focussed on techniques for simplifying the values and constraints involved in reasoning about physical systems. We have concentrated on finding a method that would reduce the conceptual complexity of the structure of a hydraulic system by aggregating components into isolated subsystems where possible. The result

of our efforts has been a new structural abstaction, the cluster-based approach to modeling hydraulic system structure.

We consider the Orbital Refueling System, designed for use by space shuttle crews in the refueling of communication and other earth satellites, as a typical example of a hydraulic system that would be useful for space application. We begin our presentation with a description of the Orbital Refueling System, which will provide problem instances for later discussion. We then introduce our notions of cluster-based modeling, defining terms and noting relevant implications. Finally, we discuss the application of cluster-based modeling to the generation and explanation of standard operating and troubleshooting procedures.

The Orbital Refueling System

The Orbital Refueling System (ORS) is an experimental system designed for fuel management and transfer, needed to accomplish the replenishment of propellant and other liquid consumables on earth-orbiting satellites. The notion is that a space shuttle (or other space tanker) could rendezvous with an orbiting satellite; an astronaut could go out, connect the ORS to the satellite and transfer liquid fuel to it, allowing the satellite to remain in orbit and function much longer than at present.

The architectural design of the hydraulic subsystem of the ORS consists of two primary subsystems: pressurization and fuel transfer. Nitrogen gas, managed under high pressure within the pressurization subsystem, provides the propellant force for the fuel transfer subsystem. The two subsystems interact at two fuel tanks, each containing an internal bladder whereby pressure from the nitrogen gas in the presurization subsystem is transmitted to the hydrazine fuel found in the fuel transfer subsystem. The pressure provides the potential force to move the fuel between the tanks or from a tank to an external device, e.g., a satellite.

The pressurization subsystem, as shown in Figure 1, consists of a tank GTK holding nitrogen gas under high pressure, which is sensed by pressure gauge @P6. GTK is connected through a pipe and T-junction to two valves in parallel, V1 and V2 (where V2 is preceded by an orifice O1). These valves are then connected through parallel pipes and another T-junction to orifice O5 and pressure regulator REG. Following REG is another T-junction, one branch leading to a relief valve RV and the other branch to a check valve CV. The check valve branch then has a pressure gauge @P5 sensing pressure in the pipe that leads to another T-junction. This split results in two paths, each eventually leading to a fuel tank. One path has two valves in succession, V3 and V7, prior to fuel tank TK1, while the other has valves V11 and V10 prior to tank TK2. Before the branches leading to the fuel tanks was another T-junction, where a side path leads, through two valves V13 and V17, to sump0. Conceptually, sump0 is infinite size tank having pressure lower than any component of the pressurization system (i.e., atmospheric pressure).

The order in which components were given above reflects the order that components would be encountered by a gaseous nitrogen flow from GTK during pressurization of the fuel tanks. Flow is always from a high pressure source, in this case GTK, to a lower pressure sink, here the fuel tanks (TK1 or TK2). Each fuel tank is split by an internal bladder which transmits the pressure of the gaseous nitrogen to the liquid hydrazine fuel, allowing the fuel to be transferred between fuel tanks or to external satellites. We will focus on the pressurization subsystem of the ORS in our discussions that follow. The approach we consider for the representation of the pressurization subsystem can be applied to the fuel transfer subsystem as well.

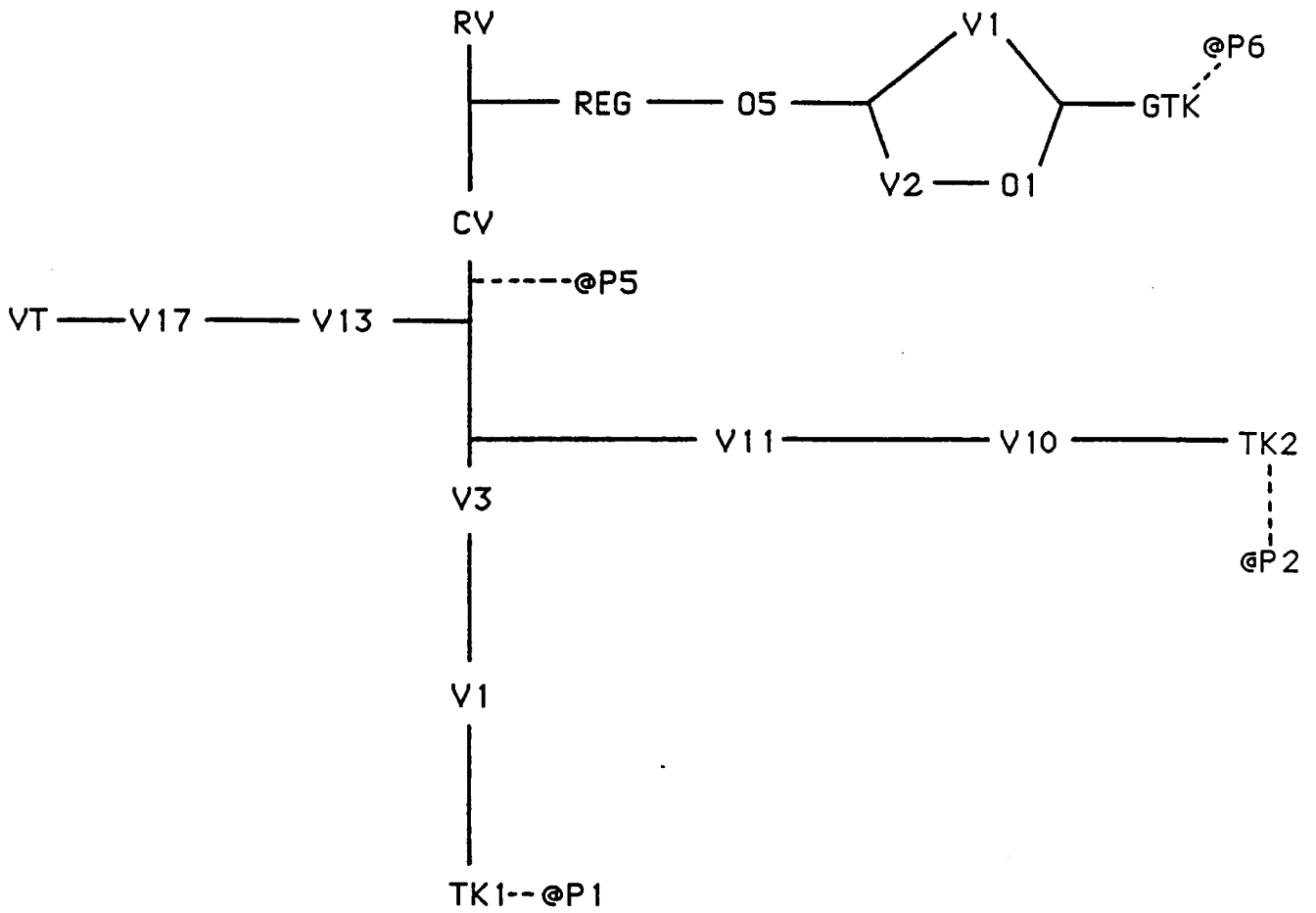


Figure 1. Component-Based Representation of the ORS

Cluster-Based Representation

In our description of the pressurization subsystem of the ORS above, we noted that a hydraulic system can be characterized as a set of interconnected components. Each component can be represented by a set of descriptive variables that capture relevant aspects of the component's internal state, a set of ports, each represented by a set of variables that capture the component's interface with other components, and a set of constraints among these variables that capture the behavior of the component. A set of constraints that identify port variables from various components represent the interconnections among components.

According to this perspective, a hydraulic system model can be described by a **component graph**, with vertices representing components (including pipes) and edges representing their port-to-port connections. The graph representing the design of the ORS which is presented in Figure 1 differs slightly from a component graph in that pipes are drawn as edges rather than vertices. System behavior can be generated from models consisting solely of representations of components and their port-to-port interconnections. STEAMER took a quantiative, interactive modeling approach to the representation of hydraulic systems based solely on component-level models [8]. However, this modeling approach may not necessarily capture the most appropriate level of abstraction for describing system behaviors of interest. Our concerns are with the clarification and explanation of system control and troubleshooting procedures. Since hydraulic systems can be architecturally complex, with many components and interconnections, we wish to find a level of structural abstraction above the component level, if possible.

The key element in our approach to the higher-level representation of hydraulic systems will be the notion of a cluster. A **cluster** consists of a maximal, connected

subset of system components that does not contain a closed valve. According to this definition, clusters are separated by closed valves. A closed valve effectively isolates the behaviors of its neighbors; as such, a set of closed valves partitions a hydraulic system into functionally independent subsystems. With each cluster we associate a set of **boundary elements**, being the closed valves that separate it from other (neighboring) clusters. We define a **cluster graph**, which can be thought of as an overlay to the component graph, wherein vertices represent clusters and edges represent boundary elements (i.e., closed valves). Two clusters are *adjacent* in the cluster graph if they have a common boundary element; there may be more than one boundary element shared by a pair of clusters. The cluster graph for the pressurization subsystem of the ORS, represented in its proposed launch configuration, is shown in Figure 2.

From a behavioral standpoint, a cluster can be in only one of two qualitative states: **stable**, where flow equals 0 (i.e., no flow occurs) and pressures are equal and constant throughout the cluster; and **unstable**, where flow is greater than 0, directed along flowpath(s) from a source of higher pressure to a sink of lower pressure within the cluster. In each unstable cluster there exists one or more flowpaths. A **flowpath** is a sequence of components and component ports, beginning with a tank (or pump) that is a source of high pressure and ending with a tank (or sump0) that is a sink of low pressure. Pressure values decrease from source to sink as resistances are encountered along the flowpath. The notion of a flowpath is fundamental to quantitiative analysis of hydraulic systems. Quantitative values for pressure drops and flow rates depend upon the pressures and resistances associated with components along a complete flowpath; for example, flow rate depends upon the pressure difference between source and sink and the sum of resistances along the flowpath.

Assuming finite sources and sinks, an unstable qualitative state tends toward stability (in the absence of external addition of material or thermal energy that would

LAUNCH CONFIGURATION

C1: GTK, 01, @P6

C2: 05, REG, CV, @P5
V3, V7, TK1 , @P1

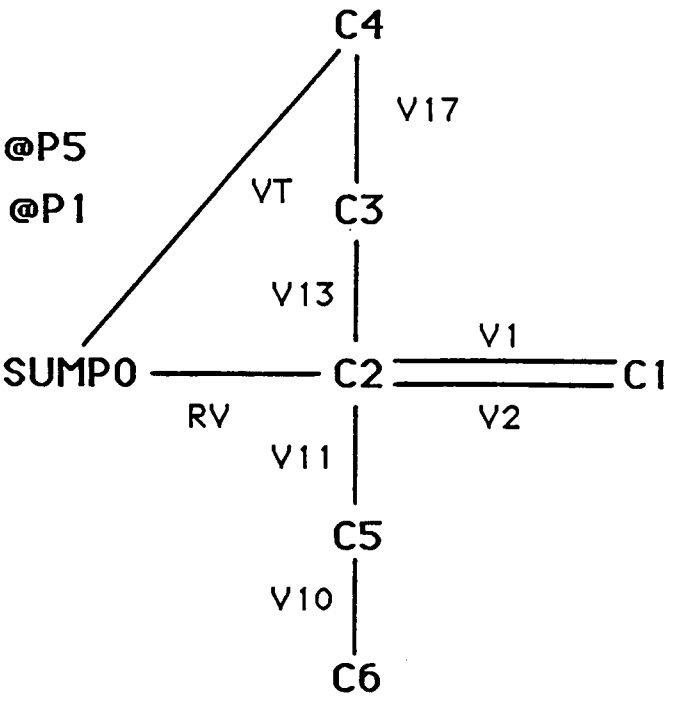
C3:

C4:

C5:

C6: TK2 , @P2

SUMP0:



Pipes and Junctions not included in cluster component listings.

Figure 2. Cluster-Based Representation of ORS Launch Configuration

raise pressures within the system); pressure values move toward equality and flow rates decrease. Components such as pressure regulators, check valves and relief valves automatically change the clustering when pressure values reach critical thresholds during the evolution of an unstable qualitative state. For example, a pressure regulator will not allow the pressure (relative to sump0) at its output side to rise above a specified pressure rating. As the output pressure reaches this limit, the regulator valve closes, resulting in two clusters that are stable at different pressures. If the cluster at the output side has its pressure subsequently lowered, the pressure regulator will open automatically, creating flow into the output side of the regulator to the reunited cluster (assuming the pressure on the input side has remained higher than the output pressure rating of the pressure regulator).

By opening and closing valves, one can change the cluster configuration of a system. By opening a valve, one most often creates an unstable cluster, resulting in the movement of gas or fluid within a hydraulic system. By closing a valve, one can eliminate an unstable cluster, cutting a flowpath and creating two stable clusters. It is the creation and elimination of unstable clusters that is the primary goal of most standard operating procedures for such systems. Through instability, work is accomplished (i.e., material is moved within the system); through stability, potential for work is maintained. In our research, we will investigate the use of cluster-based modeling of hydraulic systems as a basis for planning and explaining actions that are taken during standard operating and diagnostic procedures.

Cluster-based Reasoning: System Operation

We begin with some useful definitions. A hydraulic system is always in some **current configuration,** being a set of clusters and associated boundary elements, as determined by the valves that are currently closed. Each cluster is in a **current**

cluster state, being one of the two qualitative states described above. Detailed specifications for each of the qualitative states are assumed to be included, such as some, possibly qualitative, measure of the pressure relative to sump0 for a stable cluster and the flowpaths that exist in an unstable cluster. The **current system state** is simply the combination of current cluster states.

Given a current system state, we can reason about the future states of the system by a form of qualitative simulation according to the following two rules:

Rule 1: If all clusters are stable, then the current system state will continue until an external change in the current configuration is made.

Rule 2: If a cluster is unstable, it will become stable, either through culmination of change in the qualitative state or through a change made to the current configuration.

When all clusters are stable, no change can occur without external influence. All pressures are equal throughout each cluster, so no flow can occur. Operator-commanded changes to valve settings or faults occurring in system components may change the current configuration and disrupt the overall stability. During instability, the directions of change for pressure values and flow rates at various locations within a cluster tend toward reestablishment of stability, as discussed in the qualitative state descriptions given above. The time it takes to reach stability and the quantitative values of pressures relative to sump0 in the resultant stable state depend upon properties of components in the cluster and upon the initial pressures and capacities involved. If more than one cluster is unstable, it is generally unclear, when reasoning at a qualitative level, which cluster will reach stability first.

Operations planning is the process of determining a sequence of actions that will transform a given, current state into one satisfying a set of goal constraints. For standard operating procedures, the usual goal constraints are to raise or lower the pressure (ΔP^+ or ΔP^-) at some location and to create or alter a flow (ΔF^+ or ΔF^-)

past a given component on a flowpath in a certain direction or to stop a flow (ΔF^0) past a certain location. How can our cluster-based approach to hydraulic system representation assist us in generating standard operating procedures that satisfy these types of goals?

To raise (lower) the pressure at some location L, one must merge the cluster containing location L with another cluster having higher (lower) pressure; this results in a new cluster with pressure approaching or stabilizing at some value between the two initial values. By *merging* clusters, we mean opening a valve on the boundary between two clusters that are adjacent, thereby forming a composite, or merged, cluster. If no adjacent cluster has higher (lower) pressure, a subgoal of raising (lowering) the pressure of an adjacent cluster is generated and the search for a problem solution continues in a recursive fashion. This search is carried out in the cluster graph, which normally can be expected to be at least an order of magnitude lower in complexity than the underlying component graph. Search is focused immediately on the relevant components, the closed valves that lie on cluster boundaries. To achieve more precise, quantitative pressure changes requires reference to initial pressures, tank capacities and other component specifications (such as ratings for orifices or pressure regulators) that lie on the connecting flow paths or requires active monitoring of the pressure changes in real time during the subsequent unstable state.

To create a flow (ΔF^+ or ΔF^-) past a certain location in a certain direction (+ or -) requires that one put that location in an unstable cluster on a flowpath with a high pressure source and a low pressure sink on appropriate sides of the location in question. This requires reasoning at both levels of cluster and component topologies. The plan will again involve opening valves on the boundaries of appropriate adjacent clusters, thereby merging them into a single unstable cluster containing a flowpath of the desired form. To stop a flow (ΔF^0) past some location, one can cut off the source of

higher pressure by closing a valve on the flowpath between the location and the source, thereby isolating the high pressure source in a neighboring cluster, or one can disconnect the sink from the location, isolating the sink in a neighboring cluster.

With an understanding of the two basic qualitative states of clusters and the usual goals for standard operations as discussed above, we believe we will be able to generate plans for and explain most actions performed during standard operation. Computationally, the cluster-based representation lowers the complexity of the search for relevant flow paths. First, search proceeds in the cluster graph for a path between clusters having the desired pressure relationship. The overall flowpath is determined by joining subpaths, found by searching within clusters, that connect valves on boundaries or that connect a valve with a source or sink of pressure.

To demonstrate our approach, consider the operating goal of raising the pressure in fuel tank TK1 of the ORS, given the launch configuration shown in cluster-based form by Figure 2. At launch, the pressure in the pressurization tank GTK is greater than that in all other clusters (which are assumed to have approximately equal pressures). A trace of our cluster-based problem solving technique, reflecting its recursive, means-ends manner, would be as follows:

1. In order to raise the pressure at TK1, we must merge its cluster (C2) with a neighboring cluster having higher pressure.
2. Cluster C2 has three neighboring clusters (C1, C3, and C5), only one (C1) of which has pressure higher than C2.
3. We have the goal of merging C1 and C2, by opening a valve on their boundary.
4. We find there are two valves (V1 and V2) on the boundary between C1 and C2.
5. We open V1 to create an unstable cluster, raising the pressure at TK1.

It is important to note that our search for an operating procedure proceeds through the cluster-graph representation of the system. In this case, we did not consider the other components in cluster C2 during our search. As noted before, the cluster graph is typically much simpler than the component graph of a hydraulic system. Any specification of the final quantitative pressure desired in TK1 can now be addressed by further refinement of the qualitatively determined solution. This could involve determining the length of time to leave the valve open by consideration of parameters characterizing system components on the flowpath or simply monitoring the actual pressure increase in TK1 until the goal is reached. When the specified pressure is reached, the goal of halting the flow into TK1 can be satisfied by closing one of the open valves (V1, V3, or V7) on the flowpath within the unstable cluster.

Cluster-Based Reasoning: System Troubleshooting

Troubleshooting is the problem solving process whereby aspects of a system's behavior that have become inconsistent with system specifications are again made to satisfy those specifications. System specifications can be seen as goal constraints on the behavior of the system, and thus troubleshooting is inherently a problem solving or planning process. We have previously discussed troubleshooting as a problem solving process involving three interrelated problem spaces: Observation Space, Diagnosis Space, and Repair Space [4]. Problem solving in Observation Space is concerned with the design of test procedures and the interpretation of their results in the form of fault-related symptoms. Problem solving in Diagnosis Space is concerned with reasoning about the relationships between observed symptoms and possible faults, managing the current set of possible faults, and proposing symptoms of interest for further observation. Problem solving in Repair Space is concerned with

generating plans for the replacement or adjustment of likely faulted components or with the changing of standard operating procedures to avoid ill effects of likely faults (i.e., generating work-arounds).

Faults refer to incorrect operating states of system components. An incorrect operating state results in component behaviors that differ from those specified in a component's definition. The outputs of these incorrect component behaviors are then propagated through other components of the system, eventually producing undesired changes to observable aspects of overall system behavior. These noticeable changes in system behavior are *symptoms* of the component fault. Diagnostic reasoning focuses on the problem of associating observed symptoms with possible incorrect operating states of particular system components or, in other words, determining those faulted components whose altered behaviors could have their effects propagated through the system to produce the observed symptoms.

Troubleshooting must deal with component-level models and these component-level models must include fault models, i.e., descriptions of how a component's behavior is altered when the component is faulted in various possible ways. Below we first describe component models for three basic hydraulic system components. We then discuss how the cluster-based approach to hydraulic system modeling will assist us in the generation of fault hypotheses in Diagnosis Space that account for observed symptoms. Finally, we note how a cluster-based approach to hydraulic system modeling can be useful in generating plans in Observation Space to determine the presence of relevant symptoms. Symptoms themselves will be defined relative to the qualitative state of the cluster in which they appear, the *affected cluster*.

The two simplest components in hydraulic systems are the *pipe* and the *orifice*. A pipe offers no resistance to material flow, and thus transmits flow and pressure from its input port to its output port without change. An orifice does offer resistance to flow, and thus reduces overall cluster flow and produces a drop in pressure P_{drop}

between its input and output ports according to the constraint P_{drop} = Flow * Resistance. A *valve* is an important component type in a hydraulic system. It is a commandable component, and thus is a primary focus of attention during the creation and execution of operational procedures. When open, a valve acts like a pipe, transmitting flow and pressure without resistance. When a valve is closed, it does not transmit pressure or flow; its two ports act as if they were part of two separate worlds. In such cases, the valve will be part of the boundary between neighboring clusters. Note that a closed valve's ports may be connected through other paths, to remain within a single cluster.

To complete a model sufficient for diagnostic reasoning, we must represent the possible faults for each component type. The qualitative values of interest at any time for a given location within a hydraulic system are flow and pressure and their derivatives with respect to time (i.e., whether they are increasing, decreasing or steady). When reasoning about a system for diagnostic purposes, we are mostly concerned with how observed values differ from expected, or normal, levels [4,5]. Thus, we will model component faults in terms of how they affect pressure and flow values with respect to normal levels during cluster instability and as well as how they can alter cluster topology.

A pipe may be faulted in one of several ways: leaking, clogged, or blocked. If a pipe is clogged, it acts as an orifice, introducing resistance and lowering expected flow along the flowpath(s) of which it is part within an unstable cluster. If it is blocked, it acts like a closed valve, permitting no flow or pressure transmission and possibly creating an unexpected cluster boundary within the system. A leaking pipe similarly changes the architecture of the system, creating an orifice connection to sump0. An orifice may fail in one of two ways: blocked or open. When an orifice is blocked, it acts as a closed valve. If an orifice has failed open, it acts as a pipe, transmitting flow and pressure without resistance and, therefore, at higher than expected levels. A valve can

fail in several ways: failed-open, failed-closed, clogged, or leaking. Valve faults are dependent upon the commanded state of the valve. If a valve is open, it can fail as failed-closed or clogged. In the first case, it acts like a closed valve, changing the commanded configuration; while in the second, it acts like an orifice, reducing expected flows. If a valve is closed, it can fail as failed-open or leaking, acting as a pipe or an orifice, respectively. In both cases, the fault alters the commanded configuration, most likely producing flows where none were expected.

How can these component models, together with a cluster-based model of a system, assist in hypothesizing faulted components based upon observed symptoms? For a cluster in a stable state, there should be no flow and equal pressures are expected throughout. Symptoms that could arise in this context are a pressure above or below the expected value or a pressure that is changing, thereby indicating flow where none is expected. In all of these cases, leaking or failed open faults for valves on cluster boundaries are possible faults; the commanded configuration differs from the current system configuration in these cases. If pressure is low and decreasing, the suspected valves would be on boundaries between the affected cluster and neighboring clusters of lower pressure. A leak in a pipe within the cluster, merging the cluster with sump0, is also a possibility. If, on the other hand, pressure is abnormally high and/or increasing, then valves on boundaries with neighboring clusters of higher pressure are suspect.

The cluster-based representation focusses our attention on valves at particular boundaries as the most likely faulted components. High or low pressure readings could actually be the result of a faulted pressure gauge that is reading consistently high or low. Externally generated heat, adding energy and thus increasing internal pressure throughout the system, may be considered as well (as is system cooling a possibile cause of low pressure symptoms).

If the symptom arises while the affected cluster is in the unstable qualitative state, components on flowpaths are candidates for being faulted. If flow or pressure is lower than expected at some location, consider either clogged valves, orifices or pipes along the flowpath in the unstable cluster. If pressures are too high, consider orifices (or pressure regulators) that have failed open (or are biased high). Again the cluster-based representation focusses our attention; here on particular components along a flowpath within the unstable cluster. The cluster-based representation also serves as a basis for explaining why the components are suspected as being faulted. The faulted components are playing important roles in the system's operation, either forming part of a stable cluster's boundary or lying on the path of an intended flow within an unstable cluster of the system.

The cluster-based representation can also assist us in generating plans in Observation Space to evaluate the presence of symptoms that could help distinguish between possible faults. For performing system monitoring and diagnosis, we assume that the hydraulic system is instrumented with sensors (e.g., pressure gauge) at particular points of interest that allow us to measure flow and pressure values. As an example of problem detection, an initial symptom of higher than expected pressure could be reported by a sensor. If a pressure sensor in a stable cluster is reading high, we can locate another pressure sensor in the same cluster (or open a valve to place another pressure sensor in the same cluster) and check if it reports the same high reading, thereby verifying that the initial sensor is reading correctly. If we suspect a certain valve on the boundary of what was to be a stable cluster is leaking and there is an open valve between the sensor reporting the unexpected, changing pressure and the possibly faulted valve, then we can close that other valve and check to see whether the pressure stabilizes. If it does, then the suspected valve is the faulted component; otherwise, it is not. These plans in Observation Space clearly depend upon a cluster-

based representation of the system and upon fault models of the various components for their generation and rationalization.

Troubleshooting and the ORS

When we proposed this research, one of our stated goals was to account for elements of diagnostic procedures specified in the documentation for the ORS system. The primary diagnostic procedure we have focussed on is presented as Figure 3, taken from ORS documentation. Consideration of that procedure and the generation of explanations for the actions prescribed there were instrumental in our specification of the cluster-based representation scheme for hydraulic systems. Here we turn the process around, and complete our efforts by specification of a rule-based system that reasons in terms of a cluster-based representation to diagnose valve and gauge faults in hydraulic systems. A trace of the rules that fire during an individual diagnostic session provide a form of teleological explanation of the actions taken, in terms of underlying goals of the system.

The architecture of the rule-based sytem we propose consists of a *goal stack*, an *active memory,* and a *rule-set*. The goal stack is used to hold the current goal (at its top) and to maintain prior goals for reconsideration as necessary in a last-in-first-out order. A goal is a simple propositional statement; the goal type is represented as the predicate, with elements from the cluster and component representations as specific arguments. Active memory holds the current configuration information (i.e., components, connections, valve settings), together with system measurements acquired during diagnosis (e.g., gauge readings or comparisons). Finally, the rule-set represents the diagnostic procedure to be followed. Each rule of the set consists of a goal condition, to be matched by the top of the goal stack, followed by specific configuration and measurement conditions, to be matched by contents of active

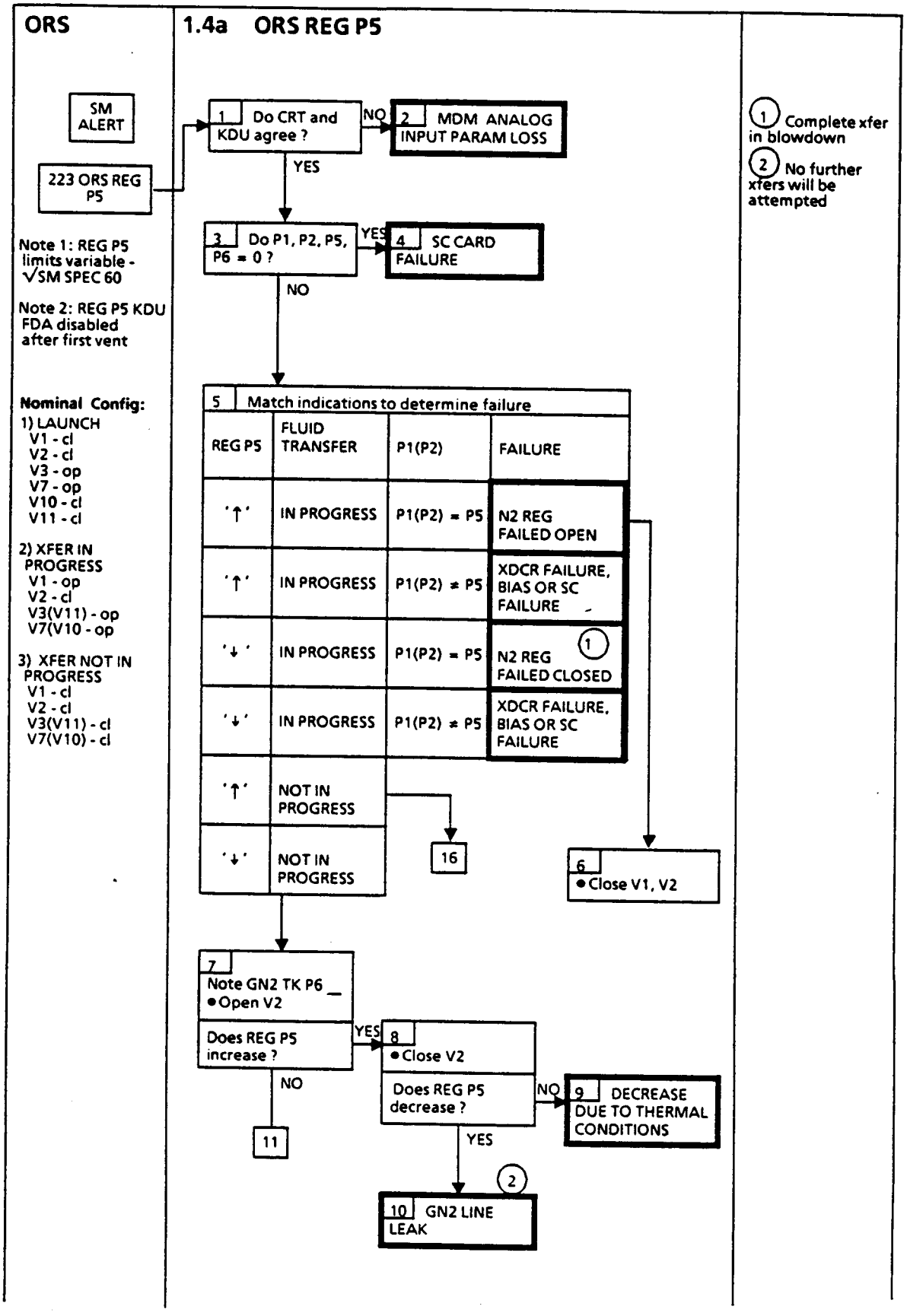


Figure 3. Part of ORS Diagnostic Procedure

memory; the rule is completed by an action sequence to be performed if the rule is fired.

The rule-set we have developed to diagnose a subset of valve and gauge faults is given in Appendix I. The rule-based system is cyclic in its action: first determining those rules that have their condition parts satisfied, selecting which of these rules to fire, and then performing the action part of the selected rule. Rules are considered in the order given in the appendix for possible selection and firing by the rule-based system. The use of a goal stack allows for the hierarchic, top-down development and execution of a diagnostic strategy. Further consideration of conditions will complete a specification of the diagnostic strategy, based on both component and cluster information, that we propose.

The diagnostic system begins operation with a current goal of diagnose, and an active memory with configuration information and an initial symptom (measurement) that has triggered diagnosis. When the goal is diagnose ("diagnose rules"), the system will either complete with success or failure, attempt to perform a test (if one is pending), select a test (if a suspect is known and no test has been selected), or select a suspect (if none is under consideration). Selection of a suspect fault ("get-suspect rules") depends upon symptoms thus far observed; no new measurements are acquired without having a suspect in mind. Once a suspected fault is selected, a test must be found ("get-test rules") to further support, confirm, or contradict existence of the suspected fault. If no test exists, the fault can not be further supported and remains possible; another fault is then selected.

If a test is found for a suspected fault, the system tries to perform actions ("do-test rules") that make up the test. It is here that the real power of the cluster-based representeation is realized. It is often the case in complex systems that various controls must be manipulated to establish an appropriate context for a desired measurement to be taken. These manipulations can not be canned, as they must reflect

the current state of the system. Very few diagnostic systems creared by research in artificial intelligence have the capability to produce plans for reconfiguring a system to establish conditions for test measurement performance. Our system is able to do this in a relatively straightforward manner, due to the cluster-based, structural abstraction.

As an example, consider when we suspect a stuck gauge fault and are attempting to perform the stuck gauge test ("stuck-gauge-test rules"). The goal of the test is to find another gauge in the same cluster as the suspect gauge. If there is none, then the attempt is made to merge the cluster with a neighbor containing a gauge ("merge-cluster and merge-clusters rules"), by opening a valve between the two clusters. If there is a gauge in the cluster with the suspect, then the cluster must be stable. If the cluster is not stable, then a valve must be closed that breaks the flow path, but leaves the two gauges of interest in the same cluster ("stabilize-cluster rules"). Finally, when both gauges are in the same cluster, the user is requested to compare their pressure readings. If they differ, then the suspect fault has been supported; here, we take it to be confirmed sufficiently to suggest it for repair by concluding it is the problem. If the readings are the same, then the gauge is concluded to be ok, and another suspect must be found by popping back to the diagnose goal.

Since the attempts at setting up the correct context for the gauge measurements may not be feasible, the rule set only plans to take the actions of opening and closing the various valves. The actions are saved as part of a plan list in active memory while the system updates its internal model of the system. The plan is only executed when it is found to be complete; this would amount to asking the user to perform the control and measurement actions on the hydraulic system (or by doing it automatically, if under computer control). If the attempt to perform the test fails, active memory is reset to the situation prior to attempting the test (RULE test-fail), which situation was checkpointed at the time the test was first considered (RULE do-test).

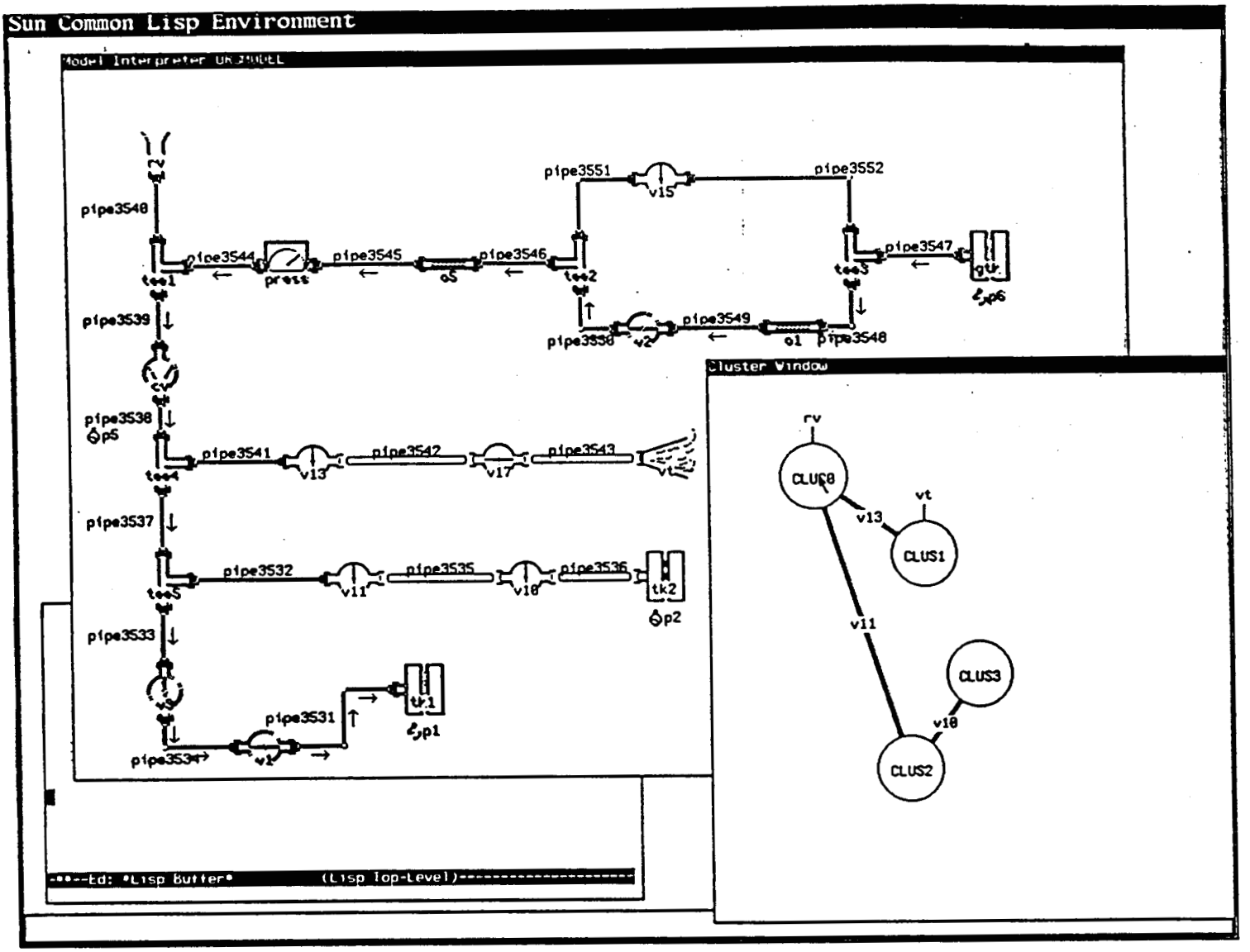


Figure 6. A Situation where Flow is in Progress

The rule set we present is not specific to the ORS model, but works for general hydraulic architectures made from the component types that we have discussed. It does not cover all possible faults in such systems, as outlined in a previous section, but demonstrates the means whereby such diagnostic procedures can be specified and the role played by cluster-level properties in those processes.

Conclusion

In this report, we have described a qualitative, cluster-based approach to the representation of hydraulic systems and demonstrated its potential for generating and explaining procedures. We have formalized many of our ideas and have implemented them as part of an interactive, computer-based system. The system allows for designing, displaying, and reasoning about hydraulic systems. Our interactive system has an interface consisting of three windows: a design/control window, a cluster window, and a diagnosis/plan window.

Figure 4 present examples of the design/control window for the ORS system. The design/control window allows a user to construct component-level models of hydraulic systems. The window operation is menu-based and mouse-driven; a user selects a component type from a menu and places a new instance of that type on the window, interconnecting various components by pipe segments as desired. In addition, valves can be opened or closed by selection with the mouse in the design window. Figure 5 presents the cluster window for the launch configuration with all valves closed Each time a valve is opened or closed the cluster graph in the cluster window is updated automatically. If a cluster is in an unstable qualitative state, the flowpath is marked by arrows on the component graph and the cluster is highlighted in the cluster graph. This is the case in Figure 6, where the ORS is in tank TK1 pressurization mode.

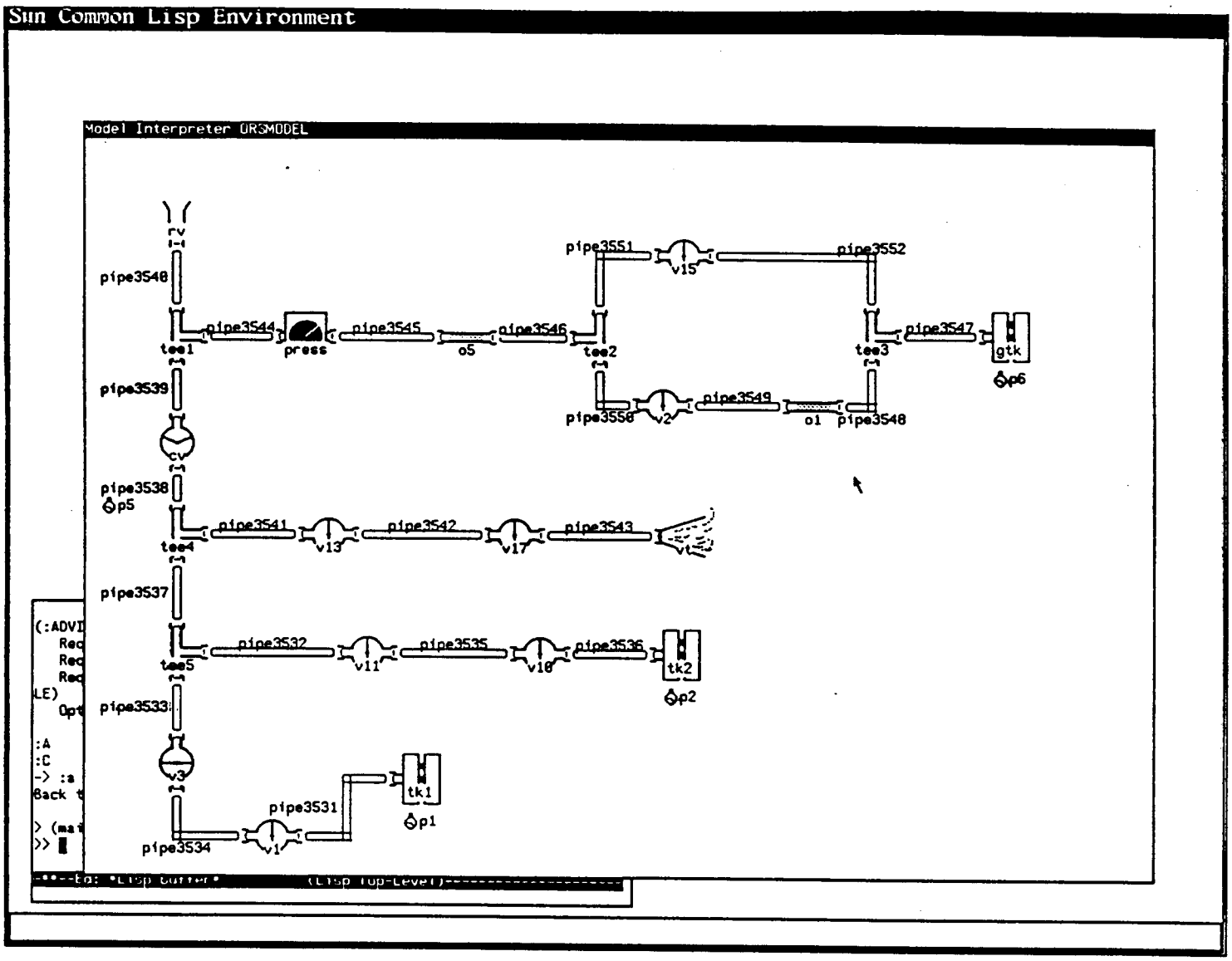


Figure 4. The Design Window of our System, with ORS Model shown

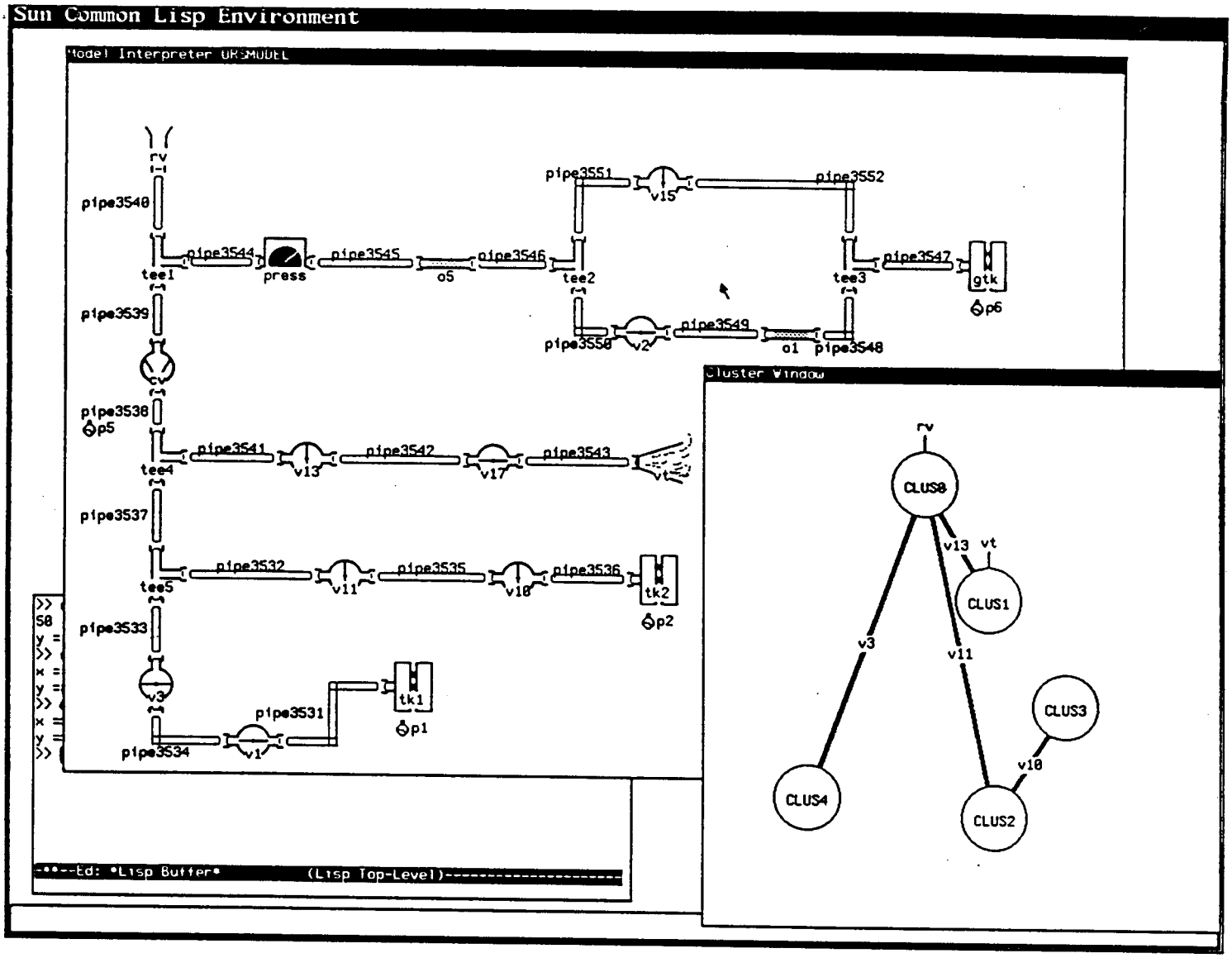


Figure 5. The Design Window
with Cluster Window

Figure 6 presents a typical screen when using our hydraulic modeling and diagnosis system.

The cluster window displays the current configuration for the hydraulic system in the design window as a graph of interconnected cluster nodes. The cluster graph bears topological resemblence to the component graph, as a cluster's node is positioned according to the center of mass of the cluster's components as they are displayed in the design window. An edge of the cluster graph is labelled by the valve(s) occurring on the boundary between the two clusters. The cluster graph display can be automatically updated as valves are opened or closed. A cluster node can be selected with the mouse and its components are highlighted in the design window.

We believe the simultaneous display of the component and cluster graphs of a hydraulic system during system control by human operators is potentially a very important application of our ideas. A simplified view of the system, as presented by a cluster graph, can assist the operator in generating actions by focussing attention on relevant components and in understanding the effects that actions have on system configurations. The cluster graph can act as a guide to the operator's scan of the component graph.

Mistakes in the interpretation of action effects are often primary factors in the occurrence of system failures involving human error. Many of these errors probably can be traced to misconceptions as to the current cluster configuration of a system. For instance, at Three Mile Island, cooling pumps were turned on but, due to closed valves, the pumps were not part of the cluster involving reactor cooling pipes; thus, the expected flow and cooling were not realized [14]. A cluster-based display could have made this error apparent to the operators.

The diagnosis/plan window allows a user to present standard diagnostic or operational goals to the system. If the goal is diagnosis, the system interacts with the user, commanding changes in system configuation and guage readings/comparisons to

be made. When a plan goal is received, an operating plan is returned that is explained in terms of a cluster-based representation of the current system. We have completed an initial set of diagnostic rules, described in the preceding section. Operation planning rules have not been written, but would follow notions discussed earlier.

References

[1] Bobrow, D. (ed), *Artificial Intelligence, Special Volume on Qualitative Modeling of Physical Mechanisms*, 24 (1-3), December, 1984; published as a book by MIT Press, September, 1985.

[2] Brown, J.S. and deKleer, J., " A qualitative physics based on confluences", in [1], p7-84.

[3] Davis, R., "Diagnostic reasoning based on structure and function", in [1], p347-410.

[4] Farley, A., "A general model of troubleshooting and its application in computer support", *Proceedings of the Fifth International Workshop on Expert Systems and Their Application*, May, 1985, Avignon. FR, p489-504.

[5] Farley, A., "Diagnostic mechanism modeling", to appear in *International Journal of Pattern Recognition and Artificial Intelligence*, 3(2), June, 1989.

[6] Genesereth, M.R., "The use of design descriptions in automated diagnosis", in [1], p411-436.

[7] Georgeff, M., "An expert system for representing procedural knowledge", SRI, AI Center Technical Report, 1985.

[8] Hollan, J., Hutchins, E.L. and Weitzman, L. "STEAMER: An interactive inspectable simulation-based training system", AI Magazine, p15-27, Summer, 1984.

[9] Hammer, J., "An intelligent flight management aid for procedural execution", to appear in *IEEE Transactions on System, Man, and Cybernetics.*

[10] Kuipers, B., "Commonsense reasoning about causality: deriving behavior from structure", in [1], p169-204.

[11] Lewis, C.M., "Rule-based analysis of pilot decisions", *Proceedings of the Human Factors Society, 29th Annual Meeting*, 1985.

[12] Lewis, C.M., "Issues in rule identification and logical induction", to appear in *Proceedings of the 1985 IEEE International Conference on Systems, Man, and Cybernetics*, November, 1985.

[13] Rouse, W.B. & Morris, N.M., "Understanding and avoiding potential problems in implementing automation", to appear in *Proceedings of the 1985 IEEE International Conference on Systems, Man, and Cybernetics*, November, 1985.

[14] Rubenstein, E. "The accident that shouldn't have happened", *IEEE Spectrum*, 16 (11), p33-42, November, 1979.

[15] Scarl, E.A., Jamieson, J.R., and Delaune, C.I., "Process monitoring and fault location at the Kennedy space center", unpublished research report, Mitre Corp.

[16] Vere, S.A., "Planning with time: windows and durations for activities and goals", Technical Report, NASA Jet Propulsion Laboratory, Pasadena, CA, November, 1981.

[17] Yoon, W.C. and Hammer, J.M., "Aiding the operator during novel fault diagnosis", to appear in *Proceedings of the 1985 IEEE International Conference on Systems, Man, and Cybernetics*, November, 1985.

APPENDIX I

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; RULES for DIAGNOSIS of GUAGE and VALVE FAULTS
;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;; diagnose rules ;;;;;;;;;;;;;;;;;;;;

(define-rule  done-fail
   (goal (diagnose))
   (suspect no-more)
 ==>
   (say "no fault found")
   (halt))

(define-rule  done-faulted
   (goal (diagnose))
   (faulted ?c ?f)
 ==>
   (say "fault found:" ?c ?f)
   (halt))

(define-rule  do-test
   (goal (diagnose))
   (test ?t)
   (not (tried-test ?t))
 ==>
   (add-fact (tried-test ?t))
   (remember-state)
   (push-goal (do-test ?t)))

(define-rule  find-tests
   (goal (diagnose))
   (suspect ?c ?f)
   (not (ok ?c ?f))
   (not (tried-all-tests ?c ?f))
 ==>
   (push-goal (get-test ?c ?f)))

(define-rule  find-suspect
   (goal (diagnose))
 ==>
   (push-goal (get-suspect)))
```

```
;;;;;;;;;;;;;;;;;; get-suspect rules ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-rule  gauge-stuck-high
   (goal (get-suspect))
   (reading-high ?g)
   (not (ok ?g stuck-high))
 ==>
   (add-fact (suspect ?g stuck-high))
   (pop-goal))

(define-rule  gauge-stuck-low
   (goal (get-suspect))
   (reading-low ?g)
   (not (ok ?g stuck-low))
 ==>
   (add-fact (suspect ?g stuck-low))
   (pop-goal))

(define-rule  valve-leaking
   (goal (get-suspect))
   (reading-high ?g)
   (rising ?g)
   (in-cluster ?g ?c)
   (separates ?v ?c ?d)
   (is-stable ?c)
   (! (higher-pressure ?c ?d))
   (not (suspect ?v ?g leaking))
 ==>
   (add-fact (suspect ?v ?g leaking))
   (pop-goal))

(define-rule  no-more-suspects
   (goal (get-suspect))
 ==>
   (add-fact (suspect no-more))
   (pop-goal))

;;;;;;;;;;;;;;;;;;;;;; get-test rules ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-rule  leaky-valve-test-1
   (goal (get-test ?v ?g leaking))
   (not (tried (leaking-valve-test-1 ?v ?g)))
 ==>
   (add-fact (test (leaking-valve-test-1 ?v ?g)))
   (pop-goal))

(define-rule  stuck-high-gauge-test
   (goal (get-test ?g stuck-high))
   (not (tried (stuck-gauge-test ?g)))
 ==>
   (add-fact (test (stuck-gauge-test ?g)))
   (pop-goal))
```

```
(define-rule  stuck-low-gauge-test
   (goal (get-test ?g stuck-low))
   (not (tried (stuck-gauge-test ?g)))
 ⇒
   (add-fact (test (stuck-gauge-test ?g)))
   (pop-goal))

(define-rule  no-more-tests
  (goal (get-test ?c ?f))
 ⇒
   (add-fact (tried-all-tests ?c ?f))
   (pop-goal))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; test-fail rule ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-rule  test-fail
  (goal (do-test ?t))
  (test-fail ?t)
 ⇒
   (recall-state)
   (pop-goal))

;;;;;;;;;;;;;;;;;;;;;;;;;; do-test rules ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;stuck-gauge-test rules;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-rule  gauge-high-ok
   (goal (do-test (stuck-gauge-test ?g)))
  (in-cluster ?g ?c)
  (gauge ?h)
  (in-cluster ?h ?c)
  (is-stable ?c)
   (comparative-gauge-reading ?g ?h same)
  (suspect ?g stuck-high)
 ⇒
  (add-fact (ok ?g stuck-high))
  (pop-goal))

(define-rule  gauge-high-faulted
   (goal (do-test (stuck-gauge-test ?g)))
  (in-cluster ?g ?c)
  (gauge ?h)
  (in-cluster ?h ?c)
  (is-stable ?c)
   (comparative-gauge-reading ?g ?h different)
  (suspect ?g stuck-high)
 ⇒
  (add-fact (faulted ?g stuck-high))
  (pop-goal))
```

```
(define-rule  gauge-low-ok
   (goal (do-test (stuck-gauge-test ?g)))
   (in-cluster ?g ?c)
   (gauge ?h)
   (in-cluster ?h ?c)
   (is-stable ?c)
    (comparative-gauge-reading ?g ?h same)
   (suspect ?g stuck-low)
 ⇒
   (add-fact (ok ?g stuck-low))
   (pop-goal))

(define-rule  gauge-low-faulted
   (goal (do-test (stuck-gauge-test ?g)))
   (in-cluster ?g ?c)
   (gauge ?h)
   (in-cluster ?h ?c)
   (is-stable ?c)
    (comparative-gauge-reading ?g ?h different)
   (suspect ?g stuck-low)
 ⇒
   (add-fact (faulted ?g stuck-low))
   (pop-goal))

(define-rule  compare-gauges
   (goal (do-test (stuck-gauge-test ?g)))
   (in-cluster ?g ?c)
   (gauge ?h)
   (in-cluster ?h ?c)
   (is-stable ?c)
 ⇒
   (execute-plan)
  (ask "Does gauge ~A have the same pressure as gauge ~A?" (?g ?h)
       (comparative-gauge-reading ?g ?h same)
       (comparative-gauge-reading ?g ?h different)))

(define  stabilize-cluster-fail
   (goal (do-test (stuck-guage-test ?g)))
   (fail (stabilize-component-cluster ?g ?h))
 ⇒
   (add-fact (test-fail (stuck-gauge-test ?g))))

(define-rule  stabilize-cluster
   (goal (do-test (stuck-gauge-test ?g)))
   (in-cluster ?g ?c)
   (gauge ?h)
   (in-cluster ?h ?c)
   (not (is-stable ?c))
 ⇒
   (push-goal (stabilize-component-cluster ?g ?h)))
```

```
(define rule merge-cluster-fail
   (goal (do-test (stuck-gauge-test ?g)))
   (fail (merge-component-clusters ?g ?h))
 ⇒
   (add-fact (test-fail (stuck-guage-test ?g))))

(define-rule merge-cluster
   (goal (do-test (stuck-gauge-test ?g)))
  (in-cluster ?g ?c)
  (gauge ?h)
  (in-cluster ?h ?d)
  (separates ?v ?c ?d)
 ⇒
   (push-goal (merge-component-clusters ?g ?h)))

;;;;;;;;;;;;;;;;;;;;;; leaking-valve-test rules ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-rule valve-faulty
   (goal (do-test (leaking-valve-test-1 ?v ?g)))
   (no-longer-rising ?g)
 ⇒
  (add-fact (faulted ?v leaky))
  (pop-goal))

(define-rule valve-ok
   (goal (do-test (leaking-valve-test-1 ?v ?g)))
  (still-rising ?g)
 ⇒
  (add-fact (ok ?v leaky))
  (pop-goal))

(define-rule test-valve
   (goal (do-test (leaking-valve-test-1 ?v ?g)))
   (can-ask (leaking-valve-test-1 ?v ?g))
 ⇒
   (execute-plan)
  (ask "Is ~A still rising~%" (?g)
       (still-rising ?g)
         (no-longer-rising ?g)))

(define-rule test-valve
   (goal (do-test (leaking-valve-test-1 ?v ?g)))
  (fail (isolate-valve ?v ?g))
 ⇒
   (add-fact (test-fail (leaking-valve-test-1 ?v ?g)))
  (pop-goal))

(define-rule try-isolate-valve
   (goal (do-test (leaking-valve-test-1 ?v ?g)))
 ⇒
  (push-goal (isolate-valve ?v ?g)))
```

```
;;;;;;;;;;;;;;;;;;;;;;;; stabilize-cluster rules ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-rule  close-valve
   (goal  (stabilize-component-cluster  ?g  ?h))
  (in-cluster ?g  ?c)
  (tank  ?t)
  (in-cluster  ?t  ?c)
  (valve  ?v)
  (in-cluster  ?v  ?c)
  (!  (on-flow-path  ?v  ?c))
  (!  (isolates  ?v  ?t  ?g  ?h))
 ⟹
   (plan-to  (close-valve  ?v))
   (pop-goal))

(define-rule  stabilize-fail
   (goal  (stabilize-component-cluster  ?g  ?h))
 ⟹
   (add-fact  (fail  (stabilize-component-cluster  ?g  ?h)))
   (pop-goal))

;;;;;;;;;;;;;;;;;;;;;;;; merge-clusters rules ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-rule  open-valve
   (goal  (merge-component-clusters  ?g  ?h))
  (in-cluster  ?g  ?c)
  (in-cluster  ?h  ?d)
  (closed-valve  ?v)
  (separates  ?v  ?c  ?d)
 ⟹
   (plan-to  (open-valve  ?v))
   (pop-goal))

(define-rule  merge-fail
   (goal  (merge-component-clusters  ?g  ?h))
 ⟹
   (add-fact  (fail  (merge-component-clusters  ?g  ?h)))
   (pop-goal))

;;;;;;;;;;;;;;;;;;;;;;;; isolate-valve rules ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-rule  isolate-valve
   (goal  (isolate-valve  ?v  ?g))
  (separates  ?v  ?c  ?d)
   (in-cluster  ?g  ?c)
   (!  (higher-pressure  ?d  ?c))
  (tank  ?t)
   (in-cluster  ?t  ?d)
   (open-valve  ?v2)
  (!  (lies-between  ?v2  ?t  ?v))
 ⟹
   (plan-to  (close-valve  ?v2))
   (add-fact  (can-ask  (leaking-valve-test-1  ?v  ?g)))
   (pop-goal))
```

```
(define-rule   isolate-valve-fail
   (goal  (isolate-valve  ?v  ?g))
  ⟹
     (add-fact  (fail  (isolate-valve  ?v  ?g)))
     (pop-goal))
```