

**MODELING AND REASONING ABOUT
PROBLEM-SOLVING SYSTEM BEHAVIOR**

Eva Hudlická and Victor Lesser

COINS Technical Report 86-04
February 1986

Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01003
CSnet address: eva.umass-cs@Csnet-relay

This research was sponsored, in part, by the National Science Foundation under Grants NSF# DCR-8500332 and NSF# DCR-8318776 and by the Defense Advanced Research Projects Agency (DOD) monitored by the Office of Naval Research under Contract N00014-79-C-0439, P00009.

ABSTRACT

The complexity of man-made systems is rapidly increasing to the point where it is becoming difficult for us to understand and maintain the systems we build. AI problem-solving systems are particularly susceptible to this information overload problem, due to their often ad hoc design, large knowledge-bases, and decentralized control mechanisms. This has recently resulted in a trend towards more autonomous systems; systems that can explain their behavior, aid the developers with debugging, and monitor and adapt their behavior, in order to function well in a changing environment. Central to all these functions is the ability of the problem-solving system to reason about its own behavior.

We describe a system component, the Diagnosis Module (DM), that enables a problem-solving system to reason about its own behavior. The problem-solving system being diagnosed is a distributed interpretation system, the Distributed Vehicle Monitoring Testbed (DVMT), which is based on a blackboard problem-solving architecture. The DM uses a causal model of the expected behavior of the DVMT to guide the diagnosis. The aim of the diagnosis is to identify inappropriate control parameters or faulty hardware components as the causes of some observed misbehavior. The DM has been implemented and successfully identifies faults in the DVMT.

Causal model based diagnosis is not new in AI. What is different about this work is the application of this technique to the diagnosis of problem-solving system behavior. Problem-solving systems are characterized by the availability of the intermediate problem-solving state, large amounts of data to process, and, in some cases, lack of absolute standards

for behavior. We have developed diagnostic techniques that exploit the availability of the intermediate problem-solving state and address the combinatorial problem arising from the large amounts of data to analyze. We have also developed a technique for dealing with cases where no absolute standard for correct behavior is available. In such cases the system selects its own "correct behavior criteria" from objects within the DVMT which did achieve some desired situation.

1. Introduction

The role of man-made systems is rapidly shifting, from that of mere tools, performing simple, tedious tasks, to that of partially autonomous assistants. As the system complexity increases, so does the amount of information that needs to be processed, both by the system developer and by its users. It would therefore be of great help if the system could take on greater responsibility for its own behavior, including more sophisticated control, some system maintenance tasks, and the ability to explain and justify its behavior. Central to all these tasks is the ability of a system to reason about its own behavior. In this paper we describe a component of a problem-solving system, the Diagnosis Module (DM), that reasons about a problem-solving system's behavior in order to diagnose the faults responsible for inappropriate system behavior. By faults we mean either hardware failures or inappropriate parameter settings which we call **problem-solving control errors**. The DM has been implemented and successfully diagnoses faults in a distributed problem-solving system: the Distributed Vehicle Monitoring Testbed [11]. The DM consists of about 5000 lines of code (including comments). It is written in a local dialect of lisp called CLISP and runs on VAX a 11/750 under the VMS operating system.

Why diagnose problem-solving system behavior? Diagnosis is a major activity at various stages of a system's life-cycle. During development, diagnosis is involved in **debugging**. Debugging is a time consuming, knowledge-intensive activity, requiring that the system designer have expectations about the system behavior as well as an understanding

of its structure.

During execution, diagnosis can be used by a problem-solving system as part of its control mechanism. In complex problem-solving system, a large portion of the system's resources (knowledge and processing) is spent on the control problem: deciding what to do next. The control problem is difficult and often a fixed control strategy is inappropriate. In order to allow a wider range of behaviors (more flexible control strategies), problem-solving systems are often parametrized. In the DVMT, for example, parameters control different aspects of control, such as load and knowledge distribution among the processing nodes, the search strategies within each node (top-down or bottom-up), and communication among the nodes. Deciding on the appropriate parameter values is a non-trivial task. AI still has no solutions for the problem of matching tasks to optimal (or even reasonable) control strategies. Usually, this is done by the system designers who empirically adjust the parameters, based on their own experience with the task and the system. Such "parameter twiddling" is tedious and time consuming. It requires an expert in both the task domain and the structure of the system itself. Not only is such manual parameter adjustment time-consuming but it may also be ineffective: there is no guarantee that the strategy selected will continue to be appropriate, because the characteristics of the system or the task may change. What is needed is a system that automatically adapts its own control strategies. Again, diagnostic reasoning is needed. During the fine-tuning of a system, diagnosis is involved in parameter adjustment, and helps to automatically determine which parameters caused some undesirable behavior and should therefore be

changed. Since these parameters determine the system control strategies, we can speak about meta-level control via parameter adjustment [7].

As AI systems take on expert tasks, they, like their human counterparts, are often called upon to explain and justify their conclusions. In order to do this, a system must have some understanding of its own structure and behavior and the ability to reason about its problem-solving in order to explain how and why it derived its results. Again then we see that diagnosis is involved in behavior explanation.

In conclusion, a system that can reason about its own behavior, has greatly enhanced capabilities which allow it to perform:

- debugging,
- meta-level control through parameter adjustment,
- explaining its own behavior.

What is Necessary to Diagnose Problem-Solving System Behavior? Diagnosis is not a new problem for AI. Many systems exist for diagnosing digital circuits [6,9,4], electrical devices [12], and large systems such as nuclear reactors [13]. There are of course numerous systems for medical diagnosis [14,16]. Is diagnosis of problem-solving behavior different from the techniques used in these domains? In our work we have found that while some diagnostic techniques are generally applicable across all domains¹ there are aspects of problem-solving system behavior that require new diagnostic techniques.

¹Given a symptom, go back through the events that led up to it until the cause is found.

We will focus on the following features of a system and will show how each requires different diagnostic techniques.

1. Is the internal structure of the system known?
2. Can the internal state of the system be determined directly or must it be deduced from observable inputs and outputs?
3. Does a fixed standard for behavior always exist?
4. Is the amount of data to analyze manageable?

Most AI diagnostic systems exist in medicine. Medical diagnostic systems must deal with uncertain knowledge since medical knowledge is often expressed in empirically observed associations rather than known internal system structure. In addition to this source of uncertainty, these systems cannot always determine exactly what the internal state of the system is, due to unreliable or unavailable tests. Medical AI systems therefore need to address the issue of uncertain knowledge. This is done in different ways: uncertainty factors in MYCIN, weights associated with causal links in CASNET [16], and complex methods for optimal test selection in ABEL [14]. The main difference between our work and diagnosis in medicine is that we do not have to deal with uncertain knowledge.

Many AI systems exist for diagnosing digital circuits [6,4,9]. Unlike the human body, the internal structure of a circuit is completely known. The internal states however, are not accessible and the focus of digital circuit diagnosis is therefore to devise discriminatory tests based on the observable inputs and outputs, which are then used to identify the faulty components. Our work differs from the work in digital circuit diagnosis, as exemplified by Genesereth and Davis, in several ways. While they use causal models which support

different types of reasoning (forward and backward chaining), their systems are much simpler. They stop with the fault identification whereas the Diagnosis Module described here simulates the effect of a fault on future system behavior. In short, the focus of the work in digital circuit diagnosis is on determining the internal system state. The focus of our work is handling the combinatorial problems and analyzing situations for which no a priori standards exist.

What characterizes problem-solving systems is:

- Complete knowledge of the internal system structure.
- Availability of the intermediate problem-solving state.
- In many cases, lack of absolute standards for correct behavior.
- Large amount of data to process during diagnosis.

Since the structure of the system is known we can use a causal model of the system and we do not need to address the issue of uncertain knowledge. Since the internal system state is available (by directly examining the system data structures), we do not need to address the problem of determining the internal state from the inputs and outputs.² The lack of absolute standards for behavior requires that the system automatically selects its own model for correct behavior. We have developed a technique which we call **Comparative Reasoning**, which addresses this problem. This is discussed in detail in Section 4. Finally,

²Many other diagnostic systems have dealt with the type of reasoning necessary given a black-box view of the system (Genesereth's DART [6] and Davis's digital circuit analyzer [4] systems deal mainly with the problems associated with this view), and our formalism certainly supports this type of reasoning. What this availability of the intermediate states has allowed us to do was to get beyond these reasoning mechanisms and explore other interesting problems associated with diagnosing the behavior of complex systems.

diagnosis of a problem-solving system requires manipulating vast quantities of data. We have developed techniques for handling the resulting combinatorial explosion. For example, the formalism for modeling the problem-solving system allows the representation of a class of objects. During diagnosis, the DM can then reason about an entire class of situations rather than the individual cases.

To summarize, problem-solving system diagnosis is different from other diagnostic tasks in the following ways.

- The system is complex but, unlike medicine, internal states are accessible, testing is cheap, and no uncertainty is involved.
- We cannot always know the correct standard for behavior a priori and must therefore construct the diagnostic system so that it automatically chooses its own standards for appropriate behavior
- Large number of objects to analyze creates combinatorial explosion. We therefore needed to develop techniques for making the representation and reasoning more efficient.

The next section describes the DVMT system and outlines by way of examples the types of diagnostic reasoning we would like to DM to perform. The structure and use of the DVMT model is described in Sections 3 and 4. A detailed description of the diagnosis is in Section 5. Section 6 discusses the methods we have developed for reducing the combinatorics in diagnosis. Section 7 summarizes the work and outlines some directions for future research.

2. Context and Motivating Examples

The problem-solving system we model and diagnose is the Distributed Vehicle Monitoring Testbed (DVMT). DVMT is distributed problem-solving system where a number of processors cooperate to interpret acoustic signals. The goal of the system is to construct a high-level map of vehicle movement in the sensed environment. The data is sensed at discrete time locations at the signal level. The final answer is a pattern track describing the path of a group of one or more vehicles, moving as a unit in some fixed pattern formation. In order to derive the final pattern track from the individual signal locations the data undergoes two types of transformation. The individual locations must be aggregated to form longer tracks, and both the tracks and the locations must be driven up several levels of abstraction, from the signal level, through the group and vehicle levels, up to the pattern level. Figure 1 illustrates the different levels of abstraction as well as all the possible pathways for deriving the final answer from the input data.

Each processor in the DVMT system is based on an extended Hearsay-II architecture where data-directed and goal-directed control are integrated [1]. The problem-solving cycle at each processor consists of creating a hypothesis that represents the position of a vehicle. Hypotheses generate goals that represent predictions about how the existing hypotheses can be extended by incorporating more of the sensed data. A hypothesis together with a goal triggers the scheduling of a knowledge source (knowledge source instantiation) whose execution will satisfy the goal by producing a more encompassing hypothesis (one which includes more information about the vehicle motion). This cycle begins with the input data

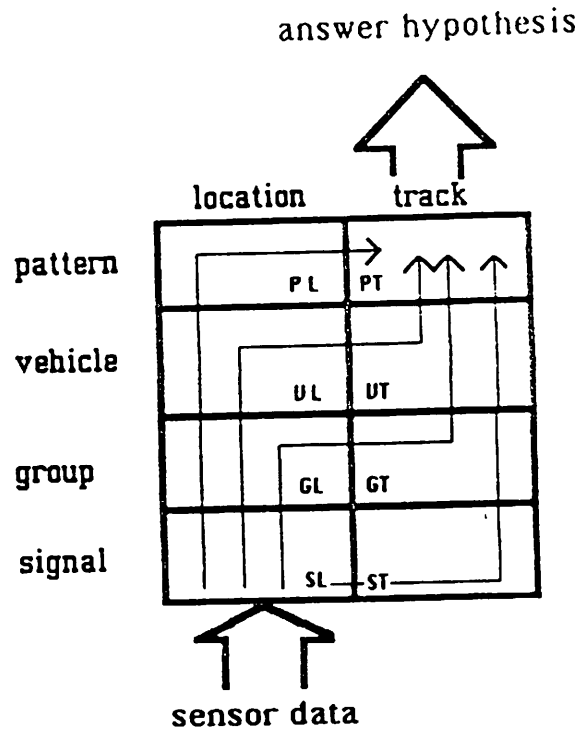


Figure 1: Levels of Abstraction in the DVMT Data Transformations
The data blackboard has eight levels of abstraction. The input data, acoustic signals representing vehicle positions at discrete time intervals, come in at the signal level (sl). The final answer, at the pattern track level, is an integrated picture of the “raw” sl data representing how the vehicles move through the environment.

and repeats until a complete map of the environment is generated. Figure 2 illustrates the processing structure at each node. This process begins with the input data and repeats until a complete map of the environment is generated or until there are no more knowledge source instantiations to invoke.

2.1 Examples Motivating the Diagnostic Techniques

In the introduction we mentioned that diagnosis of problem-solving systems requires techniques for dealing with the combinatorial explosion and for dealing with lack of fixed standards for behavior. Here we present two fault scenarios where the use of these techniques is illustrated. Both of these examples are discussed in more detail in Section 5.

Fault Scenario #1: Missing Knowledge Sources. This example illustrates the use of Underconstrained Objects to represent and reason about a class of objects. This capability is then applied to simulating the effects of an identified fault on system behavior. Suppose the DVMT system is configured such that there are four nodes, each receiving a portion of the environmental data. Since each node has a limited view of the environment, inter-node communication is necessary in order for the system to derive the overall map of the vehicular movement. Figure 3 shows such a system configuration.

The knowledge source (ks) allocation in the DVMT is controlled by parameters which are set at the beginning of each run. It is quite common for the system user to make a mistake, and omit an essential knowledge source (or include an undesirable one). In the above scenario, communication among the processing nodes is accomplished by communication

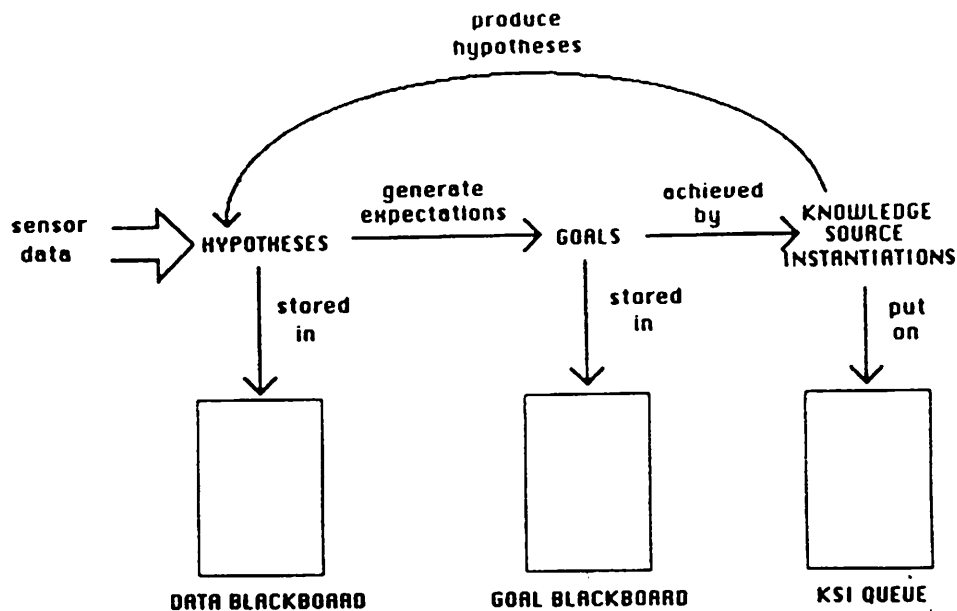
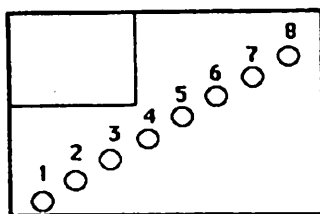
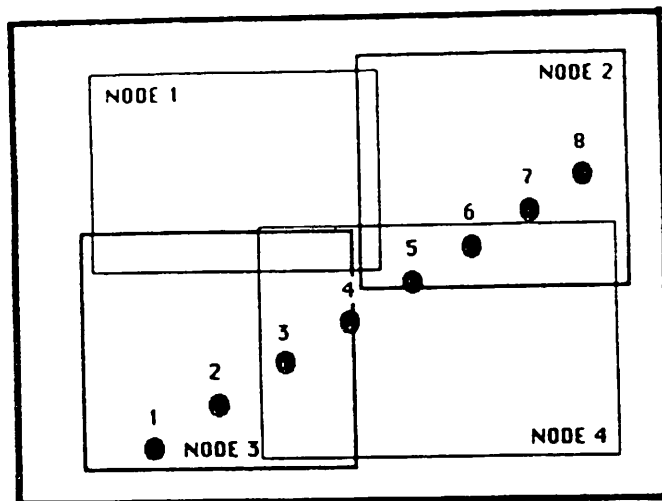
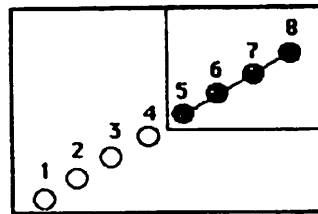


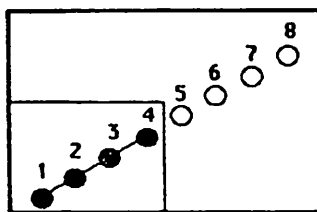
Figure 2: The Structure of Processing at Each Node in the DVMT System
The DVMT begins its interpretation task with the arrival of the sensed data. All data is represented by hypotheses and stored on the data blackboard. The arrival of a hypothesis stimulates the creation of a goal, which represents a prediction of how the hypothesis might be extended in the future. A hypothesis together with a goal stimulate the instantiation of a knowledge source (ksi). Each such instantiation is rated and, if this rating is high enough, the ksi is inserted onto the scheduling queue. At the beginning of each system cycle the highest rated ksi executes and produces additional hypotheses. This cycle repeats until the final answer is derived or until there is no more data to process.



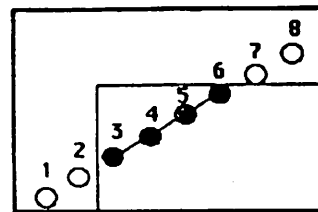
NODE 1



NODE 2



NODE 3



NODE 4

Figure 3: DVMT Configuration for Fault Scenario #1

The upper part of the figure shows the overall view of the four nodes and the data. The lower part shows the pattern track segments derived at each individual node.

knowledge sources (cks'). If these cks' are omitted from the nodes, each node will have only its local data available and the system will not derive the overall map of the environment. We would like for the DM to trace this lack of the overall map, represented by a specific hypothesis that is missing, to the fact that no communication occurred among the nodes, and, further, that this was caused by the lack of the cks' responsible for transmitting the messages (in this case hypotheses) among the nodes. Once the DM identifies fault, we would like it to predict that no communication of messages will occur as a result of the fault. In order to do this efficiently, the DM must reason about the entire class of objects (messages) affected by the missing cks.

Fault Scenario #2: Low Rated Hypothesis This example illustrates the use of Comparative Reasoning to track the low rating of a knowledge source instantiation (KSI) to a low rated hypothesis at an early point in the data transformation. The diagnosis begins with a missing high level hypothesis at the pattern track (pt) level. The Diagnosis Module reconstructs, based on a causal model of processing in the DVMT, the internal events and intermediate results that would be required to generate the desired data. As a result of this backward trace, it discovers that the desired hypothesis was not derived because lower level location hypotheses were never produced. Specifically, while hypotheses did exist at the group location (gl) level, they were never driven up to the next level, the vehicle location (vl). The Diagnosis Module further determines that this was due the fact that the KSI that would have created the desired vl hypotheses never executed, because its rating was too low. Recall that it is always the highest rated ksi on the scheduling

queue that executes. It is thus possible for a KSI with a low rating to remain on the queue for a long time.

The point of the diagnosis here is to determine what caused the ksi rating to be low. The DM has a model of how a ksi rating is derived from its components: a ks parameter called ks-goodness, the goal that is to be satisfied by the hypotheses produced by the KSI, and the data the KSI is to use. However, it is impossible to evaluate the individual ratings in terms of some standard measures of “goodness”, since the “lowness” or “highness” of a KSI rating is relative, depending on the ratings of the other KSIS’ on the queue; we do not have a fixed standards for normal ksi rating. What is needed here is to examine the queue, select a “successful” ksi, that is a high rated ksi which is about to run, and use this ksi as a model with which to compare the low rated one. This is exactly the aim of what Comparative Reasoning (CR).

In this case, CR examines the factors influencing the ksi rating for both the problem ksi and a model ksi selected from the queue. It notes that both the ks goodness parameters and the goal components are identical but that the data component rating is lower for the low rated ksi. This then is identified as the cause of the low rating. The next step is to trace back through the derivation of this low hypothesis rating and determine why is rated low. Again, the hypothesis from the model ksi are used as a standard with which to compare the low rated hypothesis. This continues through several levels of abstraction until either a primitive node in the model is reached which is responsible for the low rating (in this case, a low sensor weight or low rated data is identified), or, if the search is unsuccessful,

until the causal pathway can no longer be extended.

3. Structure of the System Behavior Model

This section describes the modeling formalism used to represent the possible behaviors of the DVMT system. By representing the internal structure of the DVMT and the causal relationships among events in the DVMT, the System Behavior Model (SBM) supports different types of reasoning. This makes it possible to use it not only for diagnosis but also for simulation of the system behavior. Unlike some other causal models, such as CASNET [16], which represent causal relationships among pathological states, the SBM represents the normal system behavior. The errors are represented as deviations from the expected situations that were not achieved by the system. The model can thus reason both about the causal sequences of expected events, and thereby simulate the correct system behavior, and about the sequences of abnormal events, and thereby diagnose faulty system behavior. Figure 4 contains the legend for the figures in this paper.













The SBM consists of three major components:

HIERARCHICAL STATE TRANSITION DIAGRAMS which represent the possible system behaviors at various levels of detail. The causal relationships among events in the system are represented as sequences of states in the model.

ABSTRACTED OBJECTS which represent individual objects (i.e., data structures) or classes of objects in the DVMT system.

CONSTRAINT EXPRESSIONS among the different attributes of the abstracted objects which represent the relationships among the objects.

LEGEND

	STATE (uninstantiated)		TRUE STATE
	UNDERCONSTRAINED STATE		PRIMITIVE STATE
	UNDERCONSTRAINED ABSTRACTED OBJECT		COMPARATIVE STATE
	ABSTRACTED OBJECT		MERGED STATE
	STATE TRANSITION ARC		FALSE STATE
	NON EXPANDABLE STATE		SYMPTOM STATE

RELATIONSHIP STATES




-  GREATER-THAN its parallel state
 -  LESS-THAN its parallel state
 -  EQUAL to its parallel state
- (in cases where there are multiple parallel states these signs may be associated with the arcs linking the parallel states)

Figure 4: Legend for Figures in the Paper

Figure 5 illustrates a high level view of the modeling formalism and its relationship to the DVMT system. We can view the model as two parallel networks. At the higher level is the state transition diagram, consisting of states and directed state transition arcs. At the lower level is the network consisting of the abstracted objects whose attributes are linked by the constraint expressions that capture the relationships among the object attributes. The constraint expressions relating the attributes of two abstracted objects are shown in Figure 6. The two networks are connected by state-object links.

The system behavior consists of a series of events. Each event results in the creation of an object (e.g., hypothesis, goal, or knowledge source instantiation) or the modification of the attributes of some existing object. **The states in the model represent the results of such events in the DVMT system.** Depending on what we want to model, a state may represent simply whether some event has occurred or it may represent some finer aspect of the event's outcome.³

States are linked to other states so as to represent the desired sequence of events in the system and to capture the causal relationships among these events. If an event is influenced independently by a number of preceding events, then the states representing these events will be ORed. That is, any one of the preceding events determines the outcome of the event in the same manner. If the outcome of an event is influenced by a number of preceding events acting together, then the states representing these events will be ANDed. The state

³There are two types of states in the model. **Predicate states**, which represent whether an event has occurred or not, and **relationship states**, which represent the relationship among two objects in the DVMT. The relationship states are used in Comparative Reasoning.

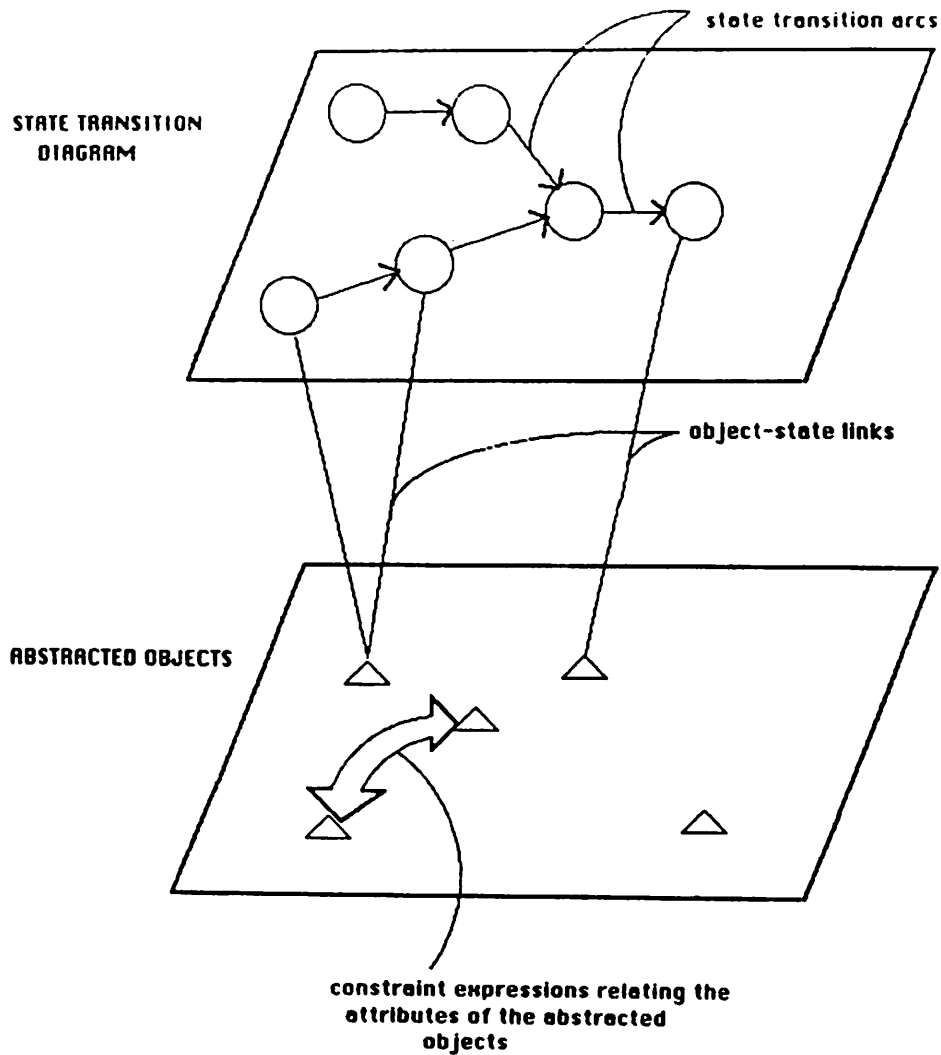


Figure 5: A High Level View of the Modeling Formalism

The SBM modeling formalism consists of three major components: the state transition diagram clusters, representing the expected sequences of events in the DVMT, the abstracted objects, representing the DVMT objects such as hypotheses or goals, and the constraint expressions, representing the relationships among the attributes of neighboring abstracted objects. The SBM can thus be viewed as two parallel networks: one containing the state transition diagrams, the other containing the abstracted objects and the constraint expressions.

This attribute points to objects that exist in the DVMT. If no such objects exist, it is false.

```
((vmt-ids (f phd:find-track-hyps
  (path self time-location-list)
  (path self event-class)
  (path self node)
  pt))
```

This attribute represents the path of the vehicle.

```
(time-location-list
  ((pt message-ob) (path x1 tll/trl))
  ((pt vt-hyp-ob) (path x1 time-location-list))
  ((pt pl-hyp-ob) (path x1 time-location-list))
```

This attribute represents a creation of shorter pt segments that could produce the desired segment. The function `create-track-segments` looks for existing shorter segments and then chooses the longest non-overlapping ones for instantiation.

```
((pt pt-hyp-ob) (f create-track-segments
  (path x1 time-location-list)
  (path x1 event-class)
  pt
  (path x1 node))))
```

This attribute represents the specific type of signal. The function `phd:higher-level-event-classes` determines the event classes for the pattern level from the vehicle level according to the system signal grammar.

```
(event-class (((pt message-ob) (path x1 event-classes))
  ((pt vt-hyp-ob)
  (f phd:higher-level-event-classes
    vt
    (path x1 event-class)))
  ((pt pt-hyp-ob) (path x1 event-class))
  ((pt pl-hyp-ob) (path x1 event-class))))
(level pt)
(node (path x1 node))..
```

Figure 6: Constraint Expressions Among Abstracted Objects

The constraint expressions linking the abstracted object attributes allow the DM to determine the attribute values of one object based on the attribute values of any of its neighboring objects. This figure shows the relationship among the attributes of the object representing a pattern track hypothesis and its neighboring objects: shorter pattern tracks (`pt-hyp-ob`), pattern location hypotheses (`pl-hyp-ob`), vehicle track hypotheses (`vt-hyp-ob`), and received pattern tracks (`message-ob`). The first part of each constraint expression specifies the context: the current state and the neighbor whose values are to be used. For example, `(pt message-ob)` means that the current state is `pt` and the object whose values are to be used is the neighboring message object. The second part of the expression specifies which of that object's attribute values should be used. For example, `(path x1 tll/trl)` specifies the `tll/trl` (`time-location-list/time-region-list`) attribute of the neighboring object (represented by the variable `x1` which always refers to the neighbor object that was instantiated most recently and whose values are to be used).

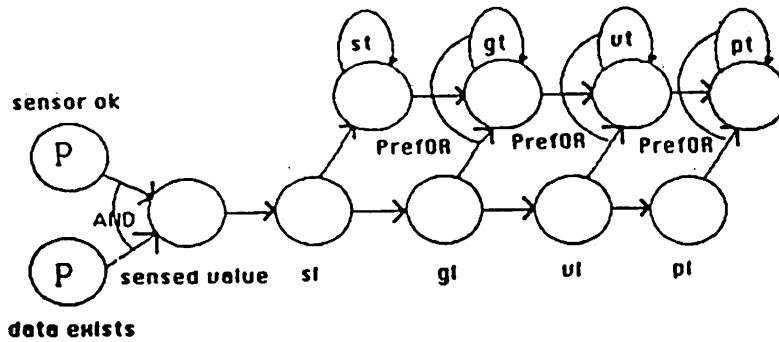


Figure 7: The Answer Derivation Model

This model cluster represents the data transformation the DVMT is expected to perform; that is, to produce pattern track (pt) hypotheses from the incoming sensor data at the signal level (sl). The arrival of the initial data depends on the sensor functioning (SENSOR-OK state) and the data's existence (DATA-EXISTS state). The SENSED-VALUE state represents a separate sensed signal for each sensor. The structure of the rest of the model parallels the data blackboard structure shown in Figure 1.

transition diagram is thus an AND/OR graph. Figure 7 shows a part of the system model.

The states are linked to the abstracted objects whose behavior they represent. The objects are represented as separate entities for efficiency reasons, to avoid the duplicate representation of similar sets of attributes. Since each state represents some aspect of the behavior of a DVMT object, that object must be represented somewhere in the system; either as part of the state or as a separate data structure. Since several states may need to refer to the same object, it is more convenient to represent that object as a separate entity rather than to repeat its representation in each state that refers to it. An abstracted object may represent an individual object or it may represent a whole class of objects. An abstracted object representing a class of objects is called an underconstrained object. For example, an underconstrained hypothesis object could have a list of levels in its level

attribute and thus represent the entire class of hypotheses at any of those blackboard levels. An abstracted object, unconstrained or not, represents an object which may or may not actually exist in the DVMT system. The states and the abstracted objects contain information necessary to instantiate the model for a specific situation in the DVMT system.

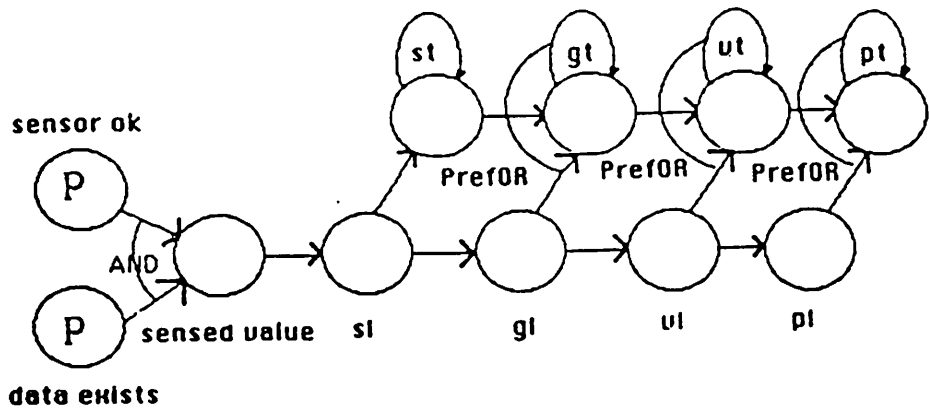
The state transition diagram representing the system behavior is organized into small clusters for manageability. These clusters are then organized into a hierarchy corresponding to increasingly detailed views of the system. Thus a high-level cluster represents selected events as contiguous states while a more detailed cluster represents other events which occur in between these states. Such hierarchical representation allows reasoning at different levels of abstraction. This is useful during diagnosis because it allows the system to focus quickly on the problem by postponing a more detailed analysis until it is necessary. For example, consider the case when the DM tries to determine why some hypothesis was not constructed. Rather than looking for the knowledge source instantiation that could have produced that kind of hypothesis, the DM first looks for the necessary supporting data and only if this data exists does it investigate the knowledge source instantiations to see whether they were scheduled and if so, why they did not run. This means that the diagnosis is first done using the Answer Derivation Cluster and only later using the Ksi Scheduling Cluster⁴ (see Figure 8). A subset of the states, designated as primitive, represents reportable faults during diagnosis.

⁴The Ksi Scheduling Cluster represents the events that occur in between each pair of states in the Answer Derivation Cluster.

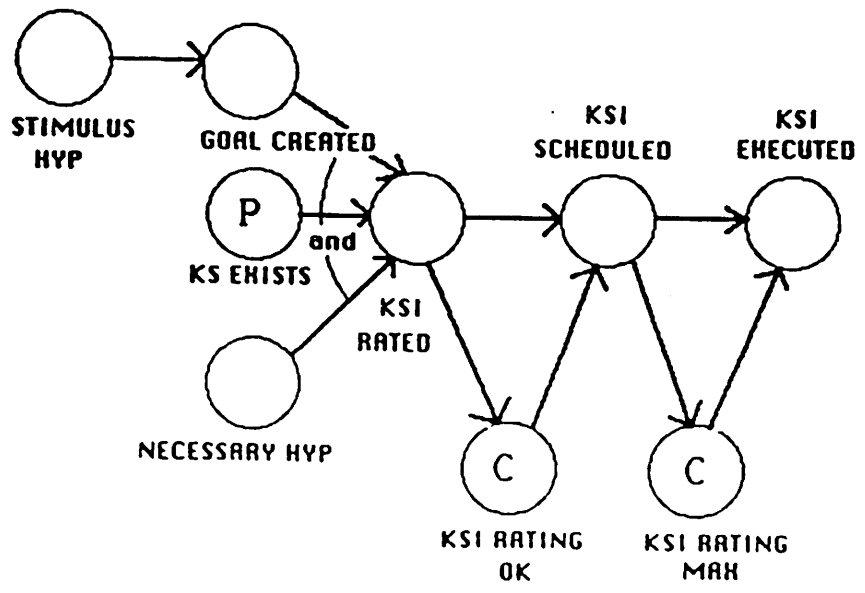
The SBM thus represents a generalized description of selected aspects of the DVMT system behavior. Figure 8 shows the set of model clusters we have constructed.

4. Use of SBM in Reasoning about DVMT Behavior

Reasoning about DVMT behavior consists of instantiating the part of the SBM that represents the system behavior relevant to the situation being analyzed. Reasoning in this context thus means the different strategies for model instantiation and the different uses of the information contained in the instantiated model. If we view the uninstantiated model as a representation of all the expected system behaviors, then instantiating a part of the model represents a search through this space. The instantiated model thus represents a small subset of the expected DVMT behaviors. Each reasoning type imposes a different search strategy on the model instantiation. The search stops when all possible pathways relevant to the situation being analyzed have been explored. For example, in order to determine why some hypothesis was not constructed, the DM must examine all possible ways in which it could have been generated; that is, via several pathways within a node, from locally available data, or from data received from other nodes. Within the instantiated model, the analysis is done exhaustively by a depth-first search. Exhaustive search is feasible here because the search space has already been reduced by the methods to be discussed in Section 6. For example, grouping together objects that behave similarly, using underconstrained objects to reason about a class of objects rather than the individual cases, and using existing data to constrain the search.



ANSWER DERIVATION CLUSTER



KSI SCHEDULING CLUSTER

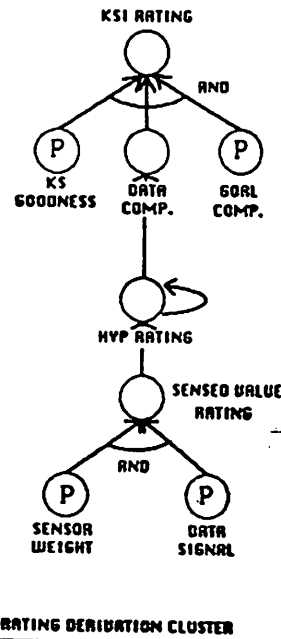
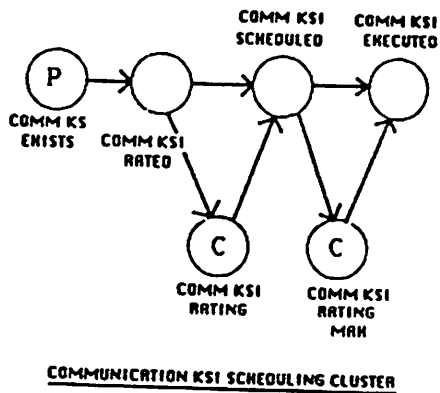
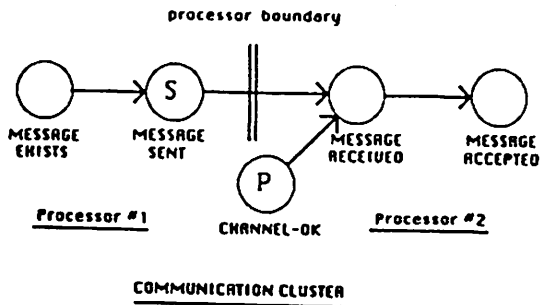


Figure 8: The Model Clusters Representing the DVMT Behavior

This figure shows the diagrams of all five model clusters used in diagnosis. At the highest level are the Answer Derivation Cluster and the Communication Cluster. The Ksi Scheduling and Comm Ksi Scheduling Clusters represent the events that take place in between each pair of states at the higher level models. The Ksi Rating Derivation Cluster represents the additional knowledge about value relationships among the rating components of the various objects.

The aim of all the different reasoning types is to explore the causes, or the effects, of an initial situation, provided as input to the DM.⁵ This situation is represented by an instantiated state and its abstracted object; that is, a state and object whose attributes have been evaluated (see Figure 9). The DM propagates these known values through the SBM, using the constraint expressions among the abstracted objects, and thereby instantiates a sequence of states causally related to the initial situation. We call such a sequence a causal pathway.

When the initial situation represents some desirable event in the DVMT that never occurred, we call it a symptom. A symptom represents an object that was never created by the DVMT, usually a hypothesis.⁶ Upon receiving a symptom, the DM traces back through the SBM in order to find out at which point the DVMT stopped working “correctly”. This is done by comparing the behavior necessary for the desired situation to occur, as represented by the instantiated model, with what actually did occur in the problem-solving system, as determined from the DVMT data structures. The aim is to construct a path from the symptom state to some false primitive states which caused it and thus explain, in terms of these primitive causes, why the DVMT system did not behave as expected. This type of diagnostic reasoning thus consists of backward chaining through the SBM. Since it constructs a causal pathway linking the initial symptom to the faults that caused it we call this type of reasoning **Backward Causal Tracing (BCT)**. Figure

⁵Currently this is done by hand. In a fully fault tolerant system the input would come from a detection component.

⁶A symptom could also represent a class of objects, such as all hypotheses at some blackboard level. This would be done using underconstrained objects.

State MESSAGE-SENT0111

f-n	(p:or (message-received0110))	; front neighbor
b-n	(p:or (message-exists0217))	; back neighbor
value	F	; state is false
object-ptrs	message-ob0071	; abstracted object

Abstracted Object MESSAGE-OB0071

vmt-ids	F
from-node	2
message-type	hyp-ob
tll/trl	((5 (14 10)) (6 (16 12)) (7 (18 14)) (8 (20 16)))
event-classes	1
level	vt
node	2
to-node	3

Figure 9: A Symptom Represented by an Instantiated State and an Abstracted Object

A situation in the DVMT is represented by an instantiated state and its associated abstracted object. This figure shows an instantiated MESSAGE-SENT state and its object MESSAGE-OBJECT. The object represents a hypothesis at the vehicle track (vt) level that was to be sent from Node #2 to Node #3. The fact that this hypothesis was never sent is represented by the value attribute of the state, which is f (false). Only a subset of the attributes is shown.

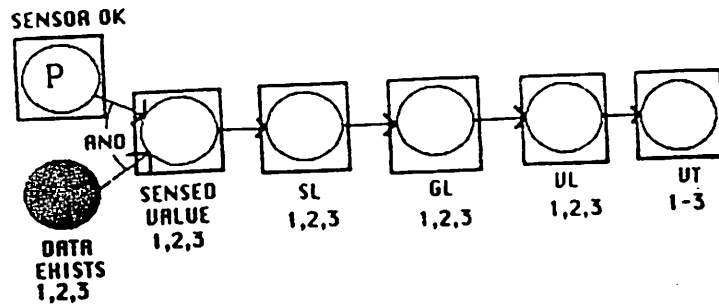


Figure 10: BCT-Constructed Causal Pathways

An instantiated Answer Derivation Cluster where BCT traced the lack of a vehicle track (vt) 1-3 hypothesis to a failed sensor, represented by the false SENSOR-OK state. The instantiated model shows the intermediate states that form the causal pathway from the false symptom state VT to the primitive false state SENSOR-OK. These intermediate states are: the VL state representing the three necessary locations and, similarly, the GL and SL states, and the SENSED-VALUE state, representing the signals sensed by the individual sensors (in this case there is only one sensor and therefore only one sensed value).

10 shows a BCT-constructed causal pathway.

In addition to a symptom, an initial situation may represent an arbitrary event in the DVMT whose effect on the DVMT behavior needs to be simulated. This is done, for example, when the DM needs to see what effects some identified fault, such as a faulty parameter setting or a failed hardware component, has on the DVMT system. This type of simulation thus consists of forward chaining through the SBM. Here the DM constructs a causal pathway which links the initial situation to all situations caused by it. We therefore call this type of reasoning **Forward Causal Tracing (FCT)**. FCT uses underconstrained objects to reason about the class of problems caused by the fault, rather than just the individual cases.

Both BCT and FCT are more complex than simple backward and forward chaining because the model is hierarchical and the interpreter must decide when to change the level of resolution; i.e., when to reason at different levels of abstraction. In addition to BCT and FCT, we have also found the need for a new type of diagnostic reasoning we call **Comparative Reasoning (CR)**. CR uses relationship states rather than predicate states.) CR compares two situations in the DVMT system and to explain why they are different. This is necessary due to the lack of absolute standards for behavior in a problem-solving system we cannot always understand the system behavior by looking at an isolated object in the system and comparing that object to some fixed standard. Instead we may have to compare the behavior of the faulty object to a similar object that seemed to behave properly. In understanding why these objects differ we often uncover a fault. CR brings up many interesting problems. The choice of a good object to use as a model for the object of interest is a nontrivial task, as is the matching up of the parallel states in the two instantiated models during this type of comparative diagnosis.

5. Examples

This section describes in more detail the diagnosis done by DM to analyze the two failure scenarios described in Section 2. We will be following the convention of representing both symptoms and faults (i.e., any undesirable situations) by false states, in the case of predicate states, and by lower-than or higher-than, in the case of relationship states. For example, a lack of some hypothesis at the vehicle track (vt) level will thus be represented

by a false state VT which will have an associated abstracted object with attribute values representing the vt hypothesis.

Details of Diagnosis for Scenario #1 We will assume that the DM has already identified the cause of a missing spanning hypothesis at Node #3 as the lack of a sent message from Node #2. Here we will show how the cause of this false MESSAGE-SENT state is traced the lack of a communication knowledge source (cks) and how this fault is then propagated forward, via an underconstrained object representing the class of messages at the vehicle track level, to account for any existing MESSAGE-SENT symptoms pending diagnosis. The instantiated model for this diagnosis is in Figure 11, part A.

Diagnosis begins with MESSAGE-SENT111, which represents the vt 5-8 hypothesis that was expected but never received by Node #3. Since this state is false, Backward Causal Tracing is invoked to instantiate the back neighbor of this state, this results in the creation of the state MESSAGE-EXISTS217. This state is true, since a vt 5-8 hypothesis does exist at Node #2. The problem must therefore occur in between these two states and the lower Communication Ksi Scheduling Cluster is therefore instantiated next. The front lower neighbor of the true state MESSAGE-EXISTS217 is instantiated, the state COMM-KSI-RATED221. The DM instantiates the back neighbor of the COMM-KSI-RATED221 state, the state COMM-KS-EXISTS220. This is a primitive state and represents the identified failure: the missing communication knowledge sources hyp-send:vt and hyp-reply:vt.

At this point the FCT is invoked to simulate the effects of the identified failures on

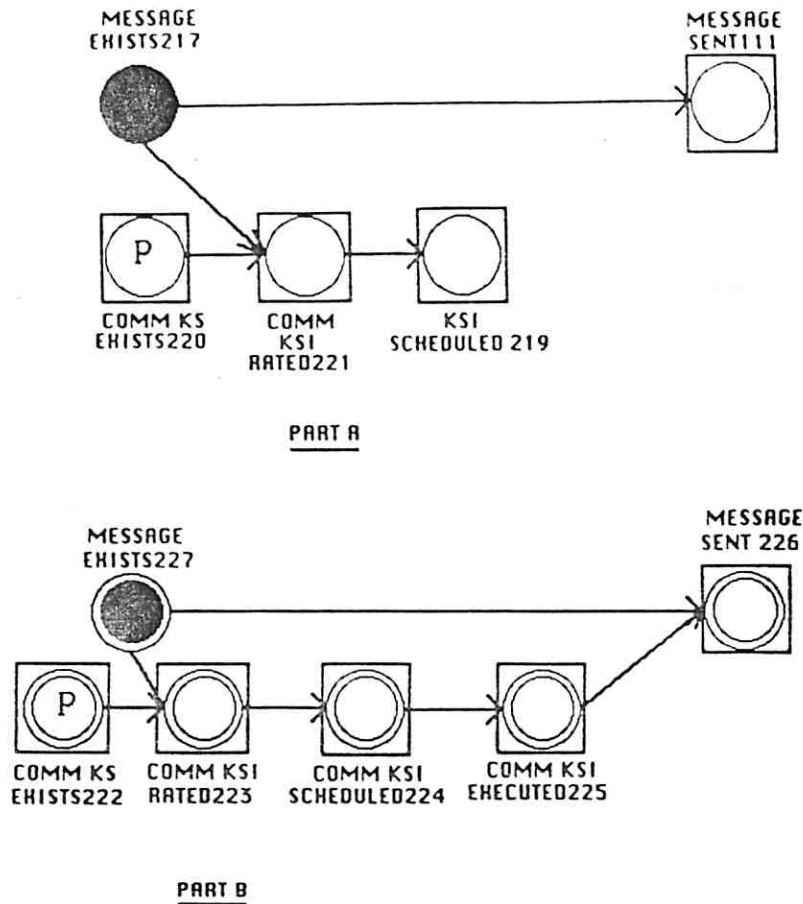


Figure 11: Instantiated SBM for Diagnosis of Pending Symptoms in Node #2

The upper part of the figure shows how BCT traced the cause of the false MESSAGE-SENT111 state to the lack of a communication ks (false primitive state COMM-KS-EXISTS220). The lower part of the figure shows how FCT propagated the effects of this fault forward, using underconstrained objects, and thereby accounted for all pending MESSAGE-SENT symptoms.

system behavior and thereby account for both pending and future symptoms caused by this failure (see Figure 11, part B). Such future symptoms are any states that are in some way effected by the false COMM-KS-EXISTS220 state. In terms of the system behavior, these are any events in some way dependent on the existence of these two cks'. In this case, these would be any events regarding the scheduling or execution of these cks's and any messages relying on these cks's for transmission. In terms of the SBM, the affected states are any states forward of the KS-EXISTS state, where forward means both at the same level of the model hierarchy and at any higher levels.

Recall that FCT works by propagating forward an underconstrained abstracted object which represents the whole class of objects affected by the fault. At each step in the diagnostic cycle, that is each time a new object/state is created, any abstracted objects associated with pending symptoms are checked to see whether they overlap with the newly created underconstrained object. If so, then their pending symptom has been successfully explained by the identified fault and need no longer be diagnosed.

Getting back to the example, FCT first creates underconstrained objects representing the missing communication knowledge sources. These are the objects COMM-KSI-OB129 and COMM-KSI-OB130, representing the hyp-send:vt and hyp-reply:vt cks's respectively. The corresponding state is then created, COMM-KS-EXISTS222. (In the diagrams, the states attached to underconstrained objects are represented by two concentric circles, instead of the usual single circle.) FCT continues simulating the fault's effect on the DVMT by expanding the front neighbors of this state. This results in the creation of the state

COMM-KSI-RATED223. This state is false and could not have been achieved via any other pathway. This means that any pending symptom states which overlap with this one, via their abstracted objects, can be explained by the simulated fault. In this case there aren't any overlapping COMM-KSI-RATED symptoms and FCT continues forward. The states COMM-KSI-SCHEDULED224 and COMM-KSI-EXECUTED225 are instantiated next. Both are false, since no cksi's exist, either on the scheduling queue or already executed, for the two missing cks's. Since the state COMM-KSI-EXECUTED225 has no forward neighbor at the same level, its forward upper neighbor is expanded,. This is the MESSAGE-SENT state in the Communication cluster above. Here FCT creates another underconstrained object, a MESSAGE-OBJECT representing any hypothesis at the vehicle track level. The object's level attribute is set to vt and all other attributes are left empty. The resulting underconstrained object represents the class of objects affected by the missing cks's, namely all message hypotheses at the vt level. This time the underconstrained MESSAGE-OBJECT does overlap with the MESSAGE-OBJECTS associated with the various pending MESSAGE-SENT symptoms. The FCT state MESSAGE-SENT226 is created and and BCT is used to ascertain that there is no other way to achieve this state. This does indeed turn out to be the case and as a result all MESSAGE-OBJECTS which overlap with the underconstrained MESSAGE-OB can be accounted for by the identified faults, the missing cks's. This means that their associated MESSAGE-SENT states that have been pending diagnosis at Node #2 are now explained by the failure. This explanation of the pending symptoms terminates diagnosis at Node #2.

Details of Diagnosis for Scenario #2 We will assume that the DM has identified the low rating of a knowledge source instantiation between the gl and vl levels as the reason for the lack of spanning hypothesis at the pt level. This situation is represented by the state KSI-RATING15. This is a relationship state and its value is low. The DM now switches to Comparative Reasoning (CR) and to a cluster representing the derivation of the ksi rating, the Ksi Rating Cluster, shown in Figure 12. At this point the DM invokes CR to determine the causes of this low rating. Let us call the low rated ksi the **problem ksi** and the maximally rated ksi on the queue the **model ksi**.

Before CR can continue, a model object must be found for the low rated ksi. Such an object must be of the same type of the problem object and it must of course be rated higher than the problem object. In this case we are looking for a ksi that takes a hypothesis at the gl level and produces a corresponding hypothesis at the vl level. The model ksi is represented by the state KSI-RATING19.

Notice that there are now two parallel instantiations of the Ksi Rating Cluster; one for the problem ksi and one for the model ksi. The two clusters will be instantiated one state at a time and compared, in an attempt to find an explanation for the low rating of the problem ksis. The search through the model (i.e., which neighbors will be expanded next) is now determined by the type of relationship found among the problem state and the model state (i.e., $<$, $+$, or $>$), rather than the state value (true or false), as was the case with predicate states. CR continues to expand back neighbors, as long as they can explain the current state's relative value with respect to the parallel state. A state's relative

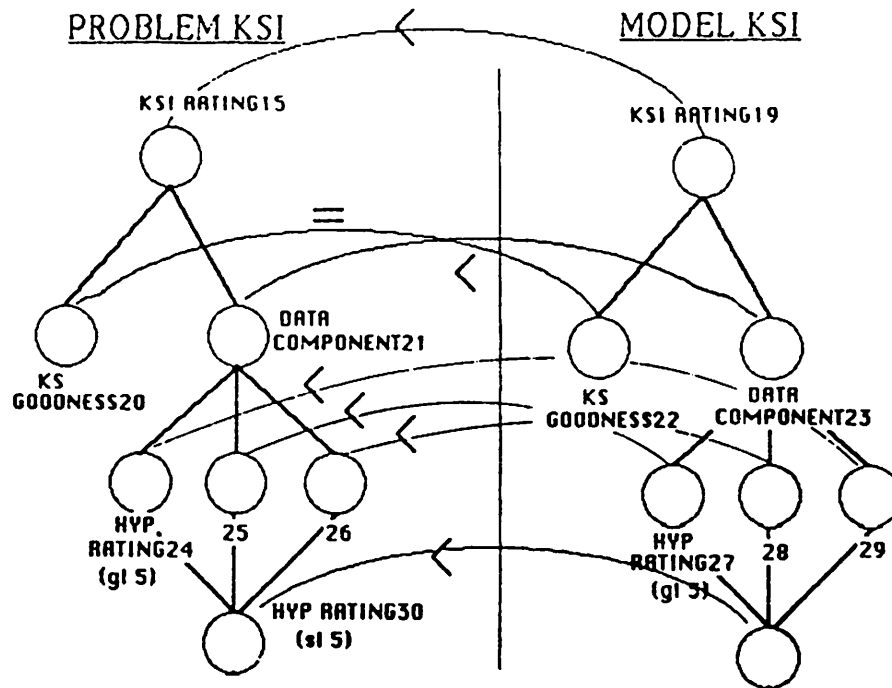


Figure 12: Instantiated SBM for Fault Scenario #2
Instantiated SBM for diagnosis of low ksi rating by Comparative Reasoning. Comparative Reasoning works by instantiating two copies of a model cluster, in this case the Ksi Rating Derivation Cluster, and comparing the "problem object rating" with the "model object rating". The "model object" is chosen by the Diagnosis Module based on selection criteria contained in the SBM. Here the low KSI-RATING15 is traced to the low HYP-RATING30 of an sl hypothesis.

value is explained by its predecessor states' values if they have the same relationship. For example, a < state is explained by its preceding < states, but not by > or by = states.

The value for state KSI-RATING15 is of course < (this is guaranteed by the process selecting the model object; if no appropriate model exists, no object will be instantiated, the problem state will not have a parallel state and diagnosis will stop.) In order to understand why KSI-RATING15 is low⁷ the DM expands its back neighbors to see if any of them are abnormally low. Since a ksi rating is a function of the ks goodness (a parameter which determines the quality of a knowledge source) and the data component (the ratings of the hypotheses the ksi is working with), the back neighbors of the state KSI-RATING are KS-GOODNESS and DATA-COMPONENT. These are instantiated and their values are determined from the values of their corresponding objects in the DVMT system. The resulting states are KS-GOODNESS20 and DATA-COMPONENT21.

Similarly, the back neighbors of the model ksi rating state, KSI-RATING19 must be expanded, so that the comparisons can continue. This expansion produces the states KS-GOODNESS22 and DATA-COMPONENT23. Before the values of these states can be determined, the parallel states must be matched. (Recall that the value of relationship states is determined by comparing the ratings of the problem and model objects.) This is done by evaluating the parallel-state attribute of each of the problem states. This attribute specifies the criteria for state matching in a declarative fashion so it can be changed easily. Currently the criteria are that the state must be of the same type, if the state represents a

⁷Low here really means "lower than the model object" since there is no such things as absolutely low or high.

hypothesis, the parallel hypotheses must be at the same level, and the value of the model object must be higher than the problem object. In the case of the ksi rating, the state matching is easy, since we have only one state of each type on both sides (the problem and the model) of the diagnosis. Evaluating the expression in the parallel-state attribute of the expanded states results in the matching up of the problem state KS-GOODNESS20 with the model state KS-GOODNESS22, and the problem state DATA-COMPONENT21 with the model state DATA-COMPONENT23. The relationship of = is found for the KS-GOODNESS states, because the ks goodness values are identical. The relationship of the DATA-COMPONENT states is <, because the value of the data component rating of the problem ksi is lower than the value of the data component rating of the model ksi.

The next step is to select a subset of the expanded back neighbors to expand further. As in BCT, we want to continue expanding only those states that explain the current problem. The current problem is a low ksi rating and we have determined that one of the components influencing this rating is normal (ks goodness) and one is below normal (data component); where "normal" means "same as the parallel state". Clearly the normal value could not have caused the ksi rating to be low. The KS-GOODNESS20 state is therefore a dead-end as far as the diagnosis is concerned, because this state is not causally related to the KSI-RATING15 state. The low DATA-COMPONENT21 state however is responsible for the low KSI-RATING15 and we therefore follow this state backward, by expanding its back neighbors.

The value of the data component of a ksi is a function of the data (i.e., the stimulus and

the necessary hypotheses). In this case, there were three gl hypotheses, one for each event class, whose rating determined the rating of the data component. (Knowledge sources that transform lower level hypotheses into higher level ones often combine several low level hypotheses of different event classes into one higher level one.) The back neighbor of DATA-COMPONENT, HYP-RATING, is therefore instantiated into three states, one for each of the three hypotheses of the problem ksi.⁸ These states are HYP-RATING24, HYP-RATING25, and HYP-RATING26. Their associated abstracted objects are objects representing group location hypotheses, GL-HYP-OBs.

The state matching is more difficult here than before because there are three parallel states to choose from on the model side; each problem HYP-RATING state has to select one of the model HYP-RATING states. In this case a heuristic is used to select the appropriate model state: the DM looks for a model state that minimizes the difference between the ratings of the two objects while maintaining the constraint that the model rating must be higher than the problem rating. (This is discussed in more detail in [8].) In the current example, this difference minimization results in the following state pairs: HYP-RATING24 and HYP-RATING29, HYP-RATING25 and HYP-RATING27, and HYP-RATING26 and HYP-RATING28. The values of these states are <, since all the hypotheses ratings are lower than the model hypotheses ratings. Diagnosis therefore continues with the expansion of the back neighbors of these states. We begin with the state HYP-RATING24 and expand its back neighbors. The back neighbor of this state

⁸No grouping of similarly behaving objects is done during CR. Therefore a state is created for each object, resulting in three HYP-RATING states.

is another HYP-RATING state, representing the rating of the hypothesis at the sl level from which the gl hypothesis referred to by the state HYP-RATING24 was derived. The resulting state is HYP-RATING30. We also expand the back neighbor of the parallel state which results in the instantiation of the state HYP-RATING31. The state matching is trivial here since there is only one model state to choose from. The two states are matched, the value of state HYP-RATING30 is determined to be lower than its parallel state HYP-RATING31, and diagnosis continues by expanding the back neighbors of this state. Thus the low rating of a ksi at the vl level has been traced to a low rated hypothesis at the sl level. We will end the diagnosis here although it could continue all the way to the SENSOR-WEIGHT and DATA-SIGNAL states, which represent the primitive causes that are ultimately responsible for the hypotheses ratings. See [8] where this example is discussed in more detail.

6. Reducing the Combinatorial Problems in Diagnosis

In a typical run, the DVMT creates hundreds of objects in each of its processing nodes. In order to diagnose a given situation, a subset of these objects has to be represented in the instantiated SBM, the model then has to be searched in an attempt to find the causes for the situation. This section describes the methods for dealing with this combinatorial explosion. These methods fall into two broad categories: those used in constructing the model, which we call **representation abstractions** and those used in instantiating and using the model, which we call **reasoning abstractions**. Below is a list of these methods:

1. Parametrizing a group of objects in order to represent the entire group by one parametrized abstracted object (occurs during model construction).
2. Parametrizing groups of states in order to represent them by one state in the model (occurs during model construction).
3. Allowing the existing data to constrain the search during diagnosis (occurs during model instantiation).
4. Selecting a representative from a group of related objects and reasoning about it (occurs during model instantiation).
5. Grouping similar object together and reasoning about them as a group in order to reduce the search (occurs during model instantiation).
6. Abstracting the common characteristics of a group of objects in order to represent and reason about the group by one abstracted object, usually an underconstrained object (occurs during model instantiation).

This rest of this section describes and motivates each of these methods for reducing the combinatorics. Figure 13 illustrates these broad categories and their relationship to the DVMT system, the SBM, and the instantiated SBM.

Representing Groups of Objects as one Abstracted Object. Much of the DVMT system behavior varies depending on the system parameters and the data. One problem we had to face in modeling the system was the construction of an concise abstraction of the wide range of possible system behaviors as the data and the system parameters change. In our formalism this translates to constructing a concise representation of a variable number

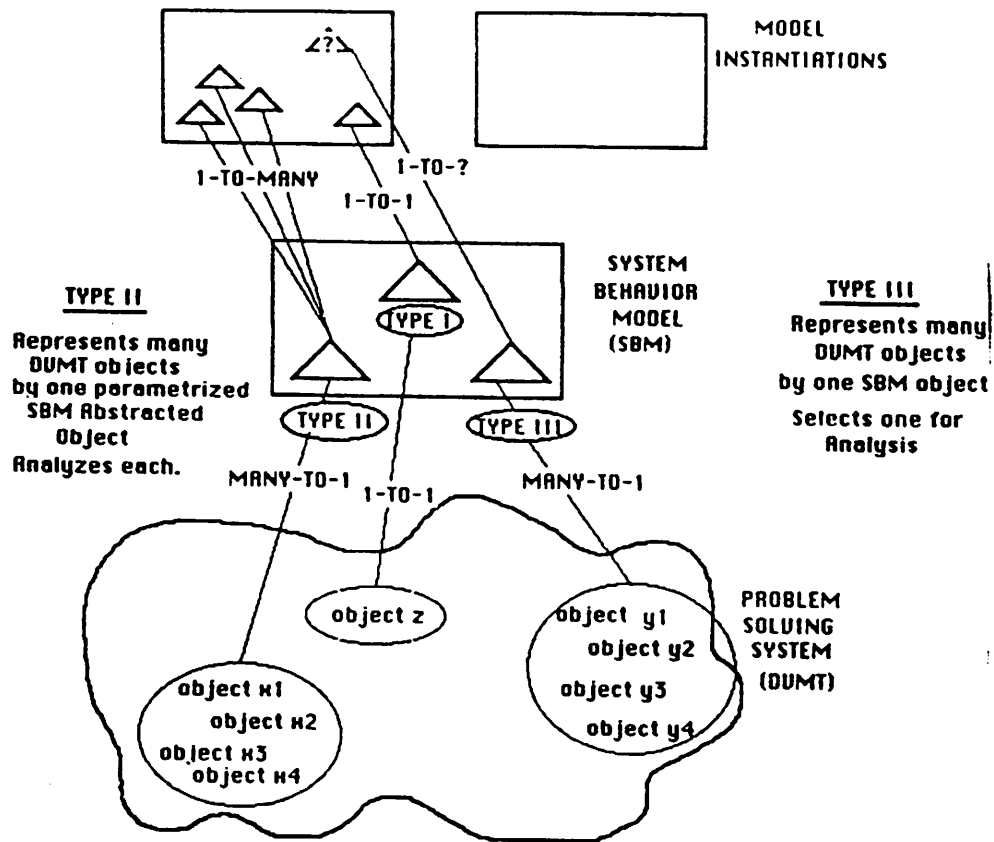


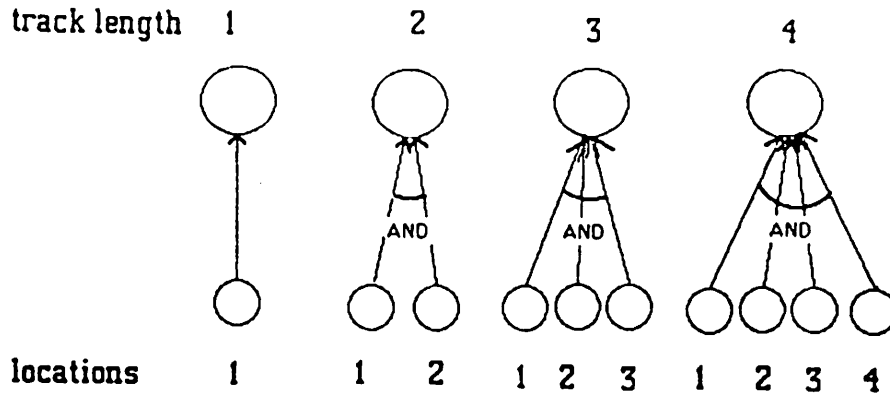
Figure 13: The Types of Abstractions Used in Model Construction and Reasoning

*This figure shows the types of abstractions necessary to represent a complex system. The bottom part of the figure represents the DVMT. The middle part represents the uninstan-
tiated SBM, and the upper part represents two instantiations of the SBM.*

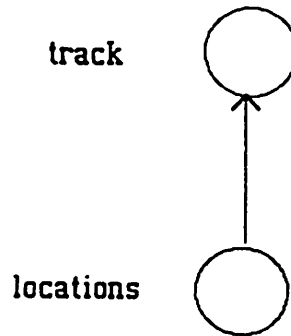
of objects in the DVMT system. We have solved this problem by grouping certain objects in the DVMT system into classes and representing the entire class as one abstracted object in the system model. This object is parametrized with respect to the data dependent attributes, whether they be the actual input data (the sensor signals) or whether they be system characteristics such as the signal grammar, the number of nodes in the system, or the number of knowledge sources in a node.

For example, consider the aggregation of locations into tracks. The model needs to be able to represent tracks of arbitrary lengths. One way to do this is to represent tracks of all possible lengths as separate objects and then represent their relationship to the locations. Figure 14, part A illustrates how this might be done. This is clearly not a very efficient way to represent the simple fact that we need n locations to form a track of length n . We therefore chose an alternative representation, shown in Figure 14, part B. In order to represent the location to track transformation for a general case, we model the group of locations as one state/object pair and a track of arbitrary length as another state/object pair. We are thus collapsing tracks of arbitrary length into one state/object and the variable number of locations into another state/object. When the exact number of locations becomes known during model instantiation, we create as many objects as necessary in order to represent each of the locations. In this way we can represent relationships among variable number of objects. There are three classes of data-dependent objects which need to be parametrized:

1. those depending on some system parameters such as the signal grammar or the



PART A.



PART B.

Figure 14: Representing a Class of Objects in the Uninstantiated Model
 This figure shows how we deal with representing many related objects. For example, when representing the numerous locations necessary to derive a track, we represent all locations by one parametrized object in the uninstantiated SBM. At instantiation time, when the number of locations becomes known, the actual number of objects is created. The abstracted object attribute that determines the actual number of objects instantiated is called a *splitting attribute*. Part A of the figure shows a non-parametrized way of representing a variable number of objects. Part B shows the parametrized method.

- number of nodes in the system;
2. those depending on the relationship among the object structures and the desired data (track to location parametrization);
 3. those depending on the existing data (longer tracks to shorter track segments).

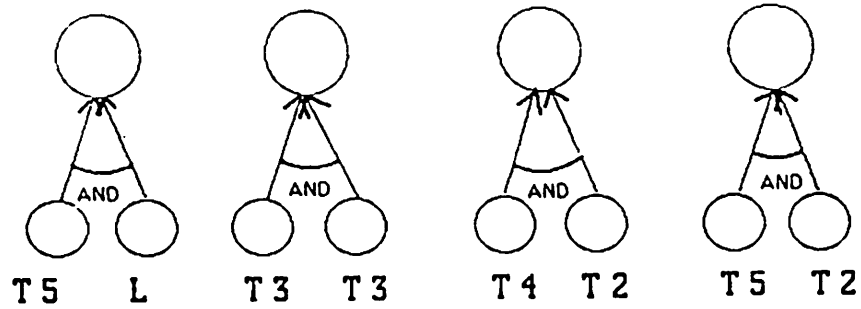
We have thus encoded a group of objects in the DVMT as one abstracted object in the SBM. Such parametrized objects require a special procedure when they are instantiated since one abstracted parametrized object may expand into a number of abstracted instantiated objects, corresponding to their actual number in the DVMT system. What we need here is to recognize when this should occur and know how many objects need to be created. In other words, we need to know the names of the object attributes which are data-dependent (and therefore parametrized) and whose values will determine the number of the actual instantiated objects. This information is contained in one of the control attributes in each abstracted object, called the **splitting-attributes**. This attribute contains the names of all the object's variable attributes that have been parametrized. During the object instantiation, when one of those attributes is evaluated, the number of values obtained determines the number of objects to instantiate. A separate object is then created and the parametrized attribute receives one of the values obtained initially. This causes a split in the branching of the graph, these attributes are called **splitting attributes**. Splitting attributes are thus attributes which have been parametrized for concise representation of objects in the system model. Examples of splitting attributes are time-location-list and event-classes of object hypothesis, goal-type of object goal, ks-type of object ksi, sensor-id

of object sensed-value, and nodes of object message. This type of abstraction which is many-to-one during the model construction and one-to-many during the model instantiation is called type I in Figure 13.

Representing Groups of States as One State. In the above example we grouped a number of objects into a class and represented that class by one abstracted object. In the following example we will illustrate a similar technique, but this time we will group together states and represent a number of possible series of events as one state in the model. This is necessary in cases where the number of the possible series of events is very large and we cannot represent all the possible series of steps of some process. A good candidate for this type of representation is the aggregation of shorter track segments and locations into longer tracks because the number of ways in which the shorter track segments and locations can be combined to form longer tracks is very large. The two choices here are shown in Figure 15. Notice that these states (the track states) point to themselves (the back neighbor of a track state is the state itself) and are therefore called **reflexive states**. During the model instantiation this state is expanded into as many states as necessary to represent how the longer track was derived from the shorter segments and individual locations. Reflexive states provide a mechanism for grouping a variable number of sequential states into one when representing the individual states explicitly would be too expensive.

Constraining the Search by Existing Data. In some cases the number of ways of deriving some object is too large unless we are constrained by the existing data. This is

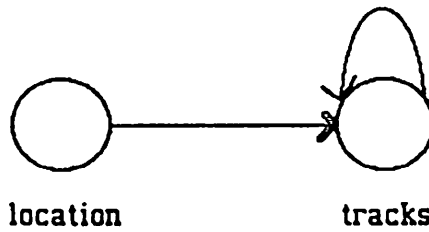
tracks of length 6



Legend

T n \longrightarrow track of length n
 L \longrightarrow location

PART A.



PART B.

Figure 15: Representing a Series of Steps in the Uninstantiated Model
 This figure shows how a series of related events in the DVMT is represented in the SBM. In this case, we need to represent the extension of a track into longer tracks. Rather than representing each possible track length in the uninstantiated model and then selecting the relevant ones at instantiation time (part A), we represent all tracks at one level by a parametrized track object (part B). At instantiation time as many of these objects are created as necessary, to represent the number of steps it actually took to create the current track. Thus one VT state in the uninstantiated SBM may be instantiated into several causally connected VT states, representing the progressive elongation or merging of some track pieces.

the case with track elongation. Suppose we are trying to analyze why a track of length 8 was not created. We could consider all the possible combinations of shorter track segments and locations and analyze why they were not created. In these cases not only would the combinatorics be prohibitive but it would not even be useful to explore all the possible derivation paths, since the system would not explore all of them either, being constrained by the data it has. We can reduce the number of pathways to explore by allowing the existing data to constrain the search during diagnosis. Rather than exploring all the possible ways of deriving some longer track we find the longest existing segment of the desired track and then analyze why it was not extended further.

Selecting a Representative Object from a Class of Objects. Another use of abstraction involves the selection of a representative object from a group of objects and analyzing its behavior rather than the behavior of each of the individual objects. In order for this to be effective, we must guarantee that the set of faults diagnosed when analyzing the representative object is the same as the set of faults diagnosed when each of the objects is analyzed separately. It turns out that track elongation satisfies this condition. In order to create a longer track, the DVMT system will aggregate shorter track segments and locations. Therefore at any time there will be a number of shorter track segments. In this case we choose the longest segment and analyze why it was not extended further. This does not reduce the number of faults we can identify because the undiagnosed shorter track segments fall into one of two categories. In one case the shorter, undiagnosed segments were later extended into a longer segment and there is no fault. In the other case the

reason they were not extended is the same as the reason the longest track of the group was not extended. This fault will be identified by analyzing why the longest track segment was not extended. This is abstraction of type III in Figure 13.

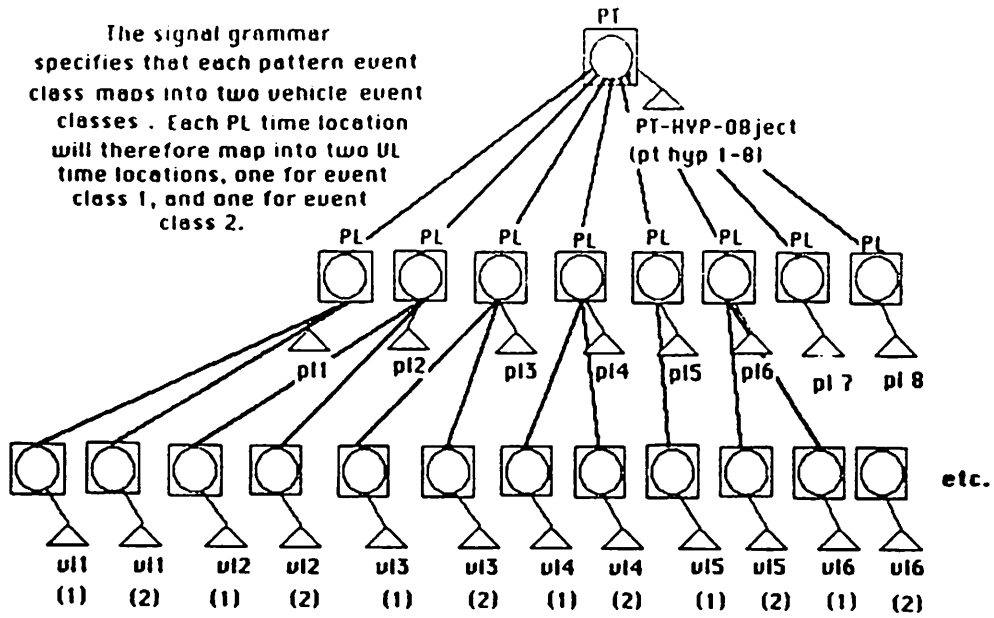
Grouping Together Similarly Behaving Objects. In cases where a single object in the model is expanded into a number of objects in the instantiated model it may be possible to group some of these objects together and diagnose each group as a unit. This is more efficient than creating a separate state for each of the objects and then repeating identical diagnostic paths with each of the states. In the initial stages of this project, every object created in the instantiated SBM had its corresponding state created and attached. During diagnosis then, each of these state-object pairs would be processed. This led to a combinatorics problem, even in relatively simple cases. Consider the diagnosis of the following situation. A PT hypothesis ranging from time 1 through time 8 is missing. In order to diagnose it, the system instantiates the SBM and traces the problem to the lack of the necessary data for this hypothesis. Because the hypothesis has 8 time locations, there are 8 locations missing. That means 8 objects and 8 states at every level, PL, VL, GL, SL, and SENSED-VALUE. Not only that, but because the system signal grammar tree has a branching factor of 2 at each level (this factor varies depending on the grammar used), we actually have 2 vl hypotheses for each pl hypothesis, 2 gl hypotheses for each vl hypothesis, and 2 sl hypotheses for each gl hypothesis. Thus for each pl hypothesis we end up with 8 sl hypotheses. Since there were 8 pl hypotheses this adds up to the grand total of 64 objects and 64 states at the sl level. Figure 16 illustrates the instantiated SBM

representing this situation.

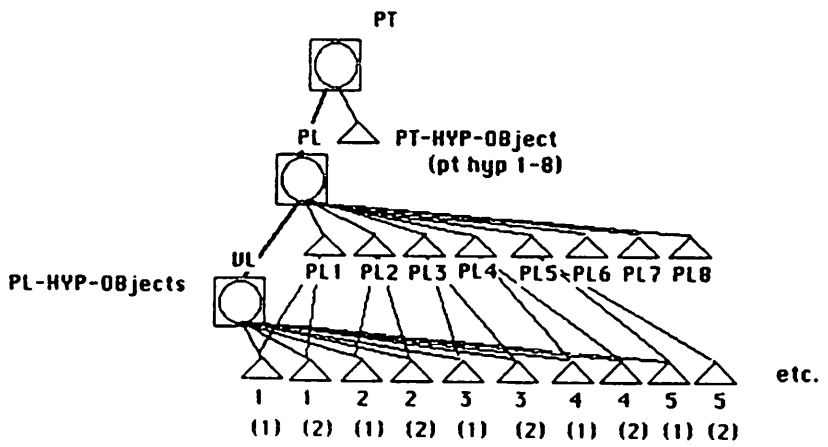
Note that there are eight pathways to follow during diagnosis. Notice also, that they all reduce the same problem: missing data. It would be much more efficient if we could recognize that all these objects behave similarly and therefore require the same diagnosis and can be grouped and diagnosed together. In other words, all the objects behaving similarly can be grouped under one state. This state is then the only one that needs to be followed during diagnosis because the predicate or value it represents is the same for all its associated objects. The resulting instantiated model is illustrated in part B of the figure. Note that here there is only one diagnostic path to follow through the model.

In order to achieve this more efficient diagnosis we need to add the following information to the model: we need to add a control attribute to each state specifying the criteria for grouping the objects associated with the state, and we need to add object-to-object links, since now we may not be able to determine which objects are directly related just by looking at their associated states. The attribute that contains the object-grouping criteria is called **object-grouping-specs**. The information is specified declaratively so it can be changed easily as the object grouping criteria change.

Currently, the grouping is performed by applying a function to some combination of the state or object attributes. The result of this function determines the number of equivalence classes for the objects. A separate state is then created for each of those equivalence classes and all the objects in that class are attached to that state. Figure 17 illustrates this iterative grouping process.



PART A



PART B

Figure 16: Reduction of the Number of States by Object Grouping
 This figure illustrates how the number of instantiated states can be reduced by grouping similarly behaving objects are under one state. Without such grouping the diagnosis of even a simple situation in the DVMT would be too costly to be useful. Part A shows the number of states and objects that would be created without grouping. Part B shows the instantiated model for the same diagnosis using object grouping.

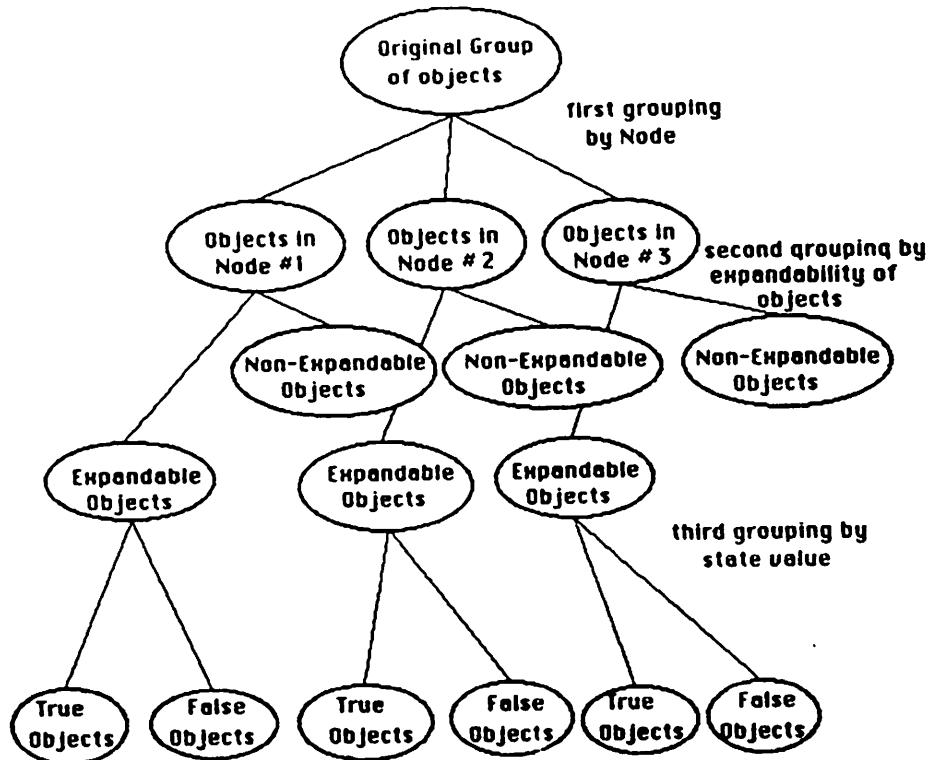


Figure 17: The Iterative Object Grouping Process

An initial group of instantiated objects is divided into classes according to grouping specifications in the SBM. A separate state is instantiated not for each instantiated object, but for each group of similar objects, as represented by the leaves of the tree shown in the figure.

This object grouping greatly reduces the number of paths that need to be examined during diagnosis. There is no loss of resolution however, because as soon as the object behavior changes and requires a different diagnostic pathway, the system creates a separate state for it, as specified in the grouping criteria. An example of a criterion for grouping objects is the existence of a corresponding object in the DVMT. All the abstracted objects which do not have a corresponding object in the DVMT are grouped together under one state (which is false) and all those that do are grouped under another (which is true). This type of object grouping can only occur when dealing with parametrized objects; i.e., objects containing splitting attributes using type I abstraction.

Underconstrained Objects. Another way of using abstraction when reasoning about the system is to group together a number of objects and represent the whole class as one abstracted object in the instantiated model. This is done by underconstraining some of the attribute values of the abstracted object. Underconstrained objects are useful when it is known that a group of objects will behave identically and we can therefore save time by reasoning about the group as a whole. A situation where this is useful is the simulation of the effects of an identified fault. Suppose the system has identified a bad sensor in the DVMT system. It would be useful to propagate the effects of this sensor forward through the model and thereby not only explain any pending symptoms, which were caused by the same fault, but also account for future symptoms. In this case the simulation involves reasoning about the class of hypotheses generated from data in the area covered by the failed sensor. It is clearly more efficient to represent this whole class as one object rather

than reasoning about all the possible hypotheses. Examples where this occurs is during the forward simulation of an identified fault.

This is different from grouping together objects and representing them as a class in the uninstantiated model, as described above because these are not for representation purposes, to be expanded later during model instantiation, but for reasoning about a smaller number of objects by abstracting away the unessential characteristics. It is thus more closely related to the above described object grouping.

7. Summary and Future Research

This paper discussed our work in the area of **problem-solving system diagnosis**. It pulls together a number of known techniques (diagnosis, simulation, qualitative reasoning, and constraint networks) and describes a few new ones (Comparative Reasoning and the use of Underconstrained Objects) in an attempt to solve the problems encountered in representing and reasoning about problem-solving system behavior. We have implemented a component of a problem-solving system, the Diagnosis Module, that diagnoses faults in the problem-solving system behavior by using a causal model of the system. The faults can be either hardware failures and or inappropriate control parameter settings, which we call **problem-solving control errors**.

Our approach to diagnosis has been determined by the following characteristics of the DVMT problem-solving system.

- The system maintains a number of data structures (blackboards, queues) that contain

its recent history. We exploit this availability of the system's intermediate states in constructing an instantiation of the system model to represent how a particular situation was reached.

- For many events in the DVMT system there are no absolute criteria for behavior. This means that in many cases we cannot determine whether an event is appropriate simply by comparing it to some fixed, ideal event. We must instead take a more global view and examine the event's relationship with other events in the system. The lack of absolute standards for system behavior necessitates a new type of diagnostic reasoning we call comparative reasoning.
- Because of the complexity of the DVMT system we could not represent every aspect of the system in the model. The problem of devising a concise representation of a large set of possible system behaviors led to various types of abstractions, both in the model construction and in the use of the instantiated model. These abstractions allow us to represent and reason about classes of objects rather than individual cases.

The Diagnosis Module is currently being used to help explain the DVMT system behavior and we are planning to integrate it into the DVMT to provide more sophisticated control.

We see many possibilities for further research in this area based on our initial experience.

1. Extending the abstracted objects to represent object components and composite objects and devising reasoning techniques for these new object types. Such an object hierarchy could be used to represent both low level domain and system knowledge and high level expectations about the system behavior. We have already begun work in this area.
2. Extending the model to represent not only what the system should do but the assumptions underlying the reasons for doing it. This would permit a much deeper

analysis of the system behavior.

3. Formalizing the issues in comparative reasoning. Extending comparative reasoning to be able to compare several system runs with different parameter settings and understand why they differ.

The work described here is a first pass at this large problem. Diagnosis is an all-pervasive activity in problem-solving system development and use; it plays a role in debugging, meta-level control, and in explaining system behavior. We hope to have demonstrated that causal model based diagnosis of problem-solving system behavior is feasible. We believe that while our work dealt with a specific problem-solving system, the techniques described here are applicable to other systems.

REFERENCES

- [1] Daniel D. Corkill, Victor R. Lesser, and Eva Hudlicka. *Unifying data-directed and goal-directed control: An example and experiments*. Proceedings of the Second National Conference on Artificial Intelligence, August 1982, pp. 143-147
- [2] Daniel D. Corkill. *A framework for organizational self-design in distributed problem solving networks*. Ph.D. Dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, February 1983.
- [3] Stephen E. Cross. *An Approach to Plan Justification Using Sensitivity Analysis*. Sigart, No. 93, July 1985, pp. 48-55,
- [4] R. Davis, H. Shrobe, W. Hanscher, K. Wieckert, M. Shirley, and S. Polit. *Diagnosis based on descriptions of structure and function*. Proceedings of the Second National Conference on Artificial Intelligence, August 1982, pp. 137-142.

- [5] Randall Davis. *Diagnostic Reasoning Based on Structure and Behavior*. *Artificial Intelligence*, Vol. 24, 1985, pp. 347–410.
- [6] M. Genesereth. *Diagnosis using hierarchical design models*. *Proceedings of the National Conference on Artificial Intelligence*, August 1982, pp. 278–283.
- [7] Eva Hudlická and Victor Lesser. *Meta-level control through fault detection and diagnosis*. *Proceedings of the National Conference on Artificial Intelligence*, August 1984, pp. 153–161.
- [8] Eva Hudlická. *Diagnosing Problem-Solving System Behavior*. Ph.D. Dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, February 1986.
- [9] Van E. Kelly and Louis I. Steinberg. *The CRITTER system: Analyzing Digital Circuits by Propagating Behaviors and Specifications*. *Proceedings of the National Conference on Artificial Intelligence*, 1982, pp. 284–289.
- [10] Victor R. Lesser and Lee D. Erman. *Distributed Interpretation: A Model and an Experiment*. *IEEE Transactions on Computers*, Special Issue on Distributed Processing Systems, Vol.C-29, No. 12, December 1980, pp. 1144–1162.
- [11] Victor Lesser and Daniel D. Corkill. *The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks*. *AI Magazine* 4(3):15–33, Fall 1983.
- [12] Drew McDermott and Ruven Brooks. *ARBY: Diagnosis with Shallow Causal Models*. *Proceedings of the National Conference on Artificial Intelligence*, 1982, pp. 370–372.
- [13] William R. Nelson. *REACTOR: An Expert System for Diagnosis and Treatment of Nuclear Reactor Accidents*. *Proceedings of the National Conference on Artificial Intelligence*, August 1982, pp. 296–301.
- [14] Ramesh S. Patil, Peter Szolovits, and William B Schwartz. *Causal Understanding of Patient Illness in Medical Diagnosis*. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vol. 2, 1981, pp. 893–899.

- [15] Chuck Reiger and Milt Grinberg. *The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms*. **Proceedings of the Fifth Joint Conference on Artificial Intelligence**, Vol. 1, August 1977.
- [16] Sholom M. Weiss, Casimir A. Kulikowski, Saul Amarel, and Aran Safir. *A Model-Based Method for Computer-Aided Medical Decision Making*. **Artificial Intelligence**, 11:145–172, 1978.